A Deep Learning Inference Accelerator Based on Model Compression on FPGA

Lu Jing
AI & HPC, Inspur Corporation
Beijing, China
jinglu@inspur.com

Jun Liu
AI & HPC, Inspur Corporation
Beijing, China
liu.jun@inspur.com

Fuhai Yu
AI & HPC, Inspur Corporation
Beijing, China
antony1987@126.com

ABSTRACT

Convolutional neural networks (CNN) have demonstrated state-of-the-art accuracy in image classification and object detection owing to the increase in data and computation capacity of hardware. However, this state-of-the-art achievement depends heavily on the DSP floating-point computing capability of the device, which increases the power dissipation and cost of the device. In order to solve the problem, we made the first attempt to implement a CNN computing accelerator based on shift operation on FPGA. In this accelerator, an efficient Incremental Network Quantization (INQ) method was applied to compress the CNN model from full precision to 4-bit integer, which represents values of either zero or power of two. Then the multiply and accumulate (MAC) operations for convolution layer and fully-connected layer was converted to shift and accumulation (SAC) operations, and SAC could be easily implemented by the logic elements of FPGA. Consequently, parallelism of CNN inference process can be further expanded. For the SqueezeNet model, single image processing latency was 0.673ms on Intel Arria 10 FPGA (Inspur F10A board) showing a slightly better result than on NVIDIA Tesla P4, and the compute capacity of FPGA increased by 1.77 times at least.

CCS CONCEPTS

• **Hardware** → Reconfigurable logic and FPGAs; • **Computing methodologies** → Neural networks;

KEYWORDS

CNN, FPGA, Shift and Accumulation, Model Compression, Quantization, Energy Efficiency

ACM Reference Format:
Lu Jing, Jun Liu, and Fuhai Yu. 2019. A Deep Learning Inference Accelerator Based on Model Compression on FPGA . In *Proceedings of ACM Woodstock conference (FPGA'19),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/00.0000/1234567890

1 INTRODUCTION

Convolutional neural networks (CNN) have demonstrated state-of-the-art results on a variety of computer vision tasks [21] ranging from image classification, semantic segmentation to object detection. With robust increase of labelled data and computation capacity of hardware, CNN can be trained to deliver incredible accuracy in object classification and other applications through back-propagations [4]. Among them, the ResNet-152 [15] has shown superior performance compared to human beings. However, what can't be ignored is that the record-breaking results are achieved by dramatic increase in network depths, model sizes and computation costs.

To alleviate the computation burden, different hardware, such as GPU, FPGA and ASIC are most widely adopted [25]. GPU can accelerate the calculation very well, but it has higher power consumption. ASIC is specifically designed for certain neural networks, which means it is not transferrable to applications of other network structures. But the energy efficiency is beyond that of GPU. Therefore, certain compromise exists between the feasibility and energy efficiency of hardware. FPGA [28] becomes the potential programmable logic alternatives because of its low power consumption and flexibility. As is known, FPGA devices can be programmed by hardware description languages, such as VHDL and Verilog [24] which is quite time-consuming. Intel FPGA SDK for OpenCL [2] improves the development efficiency of FPGA programs and facilitate the development of large-scale programs on FPGA. With the assistance of compiler, programmers can focus on the design of algorithms leaving all tedious hardware designs and timing-convergence problem to the SDK.

In addition to the development of hardware, prune and compress of networks [12] also play their roles in decreasing the CNN model size and alleviating the computation burden. Transferring the data type from FP32 to FP16 can decrease the model size by 50% with ignorable decrease in accuracy. However, the FP16 algorithm is not intrinsically supported by most hardware and in the meanwhile, the compress ratio is far from satisfactory for real applications. The traditional fixed point method [23], which is feasible for different networks, directly transfers images and weights into integers. The network weight format can be transformed to 8 and even lower bit integer values to reduce the model size. The direct transform of network weight format accelerates the feed-forward process, but leads to decrease of model accuracy. Some aggressive compress methods (expectation backpropagation (EBP)) [6] even constrain the network weights to +1 and -1 during feed-forward tests in a probabilistic way. The binary network [7] decreases the model size greatly, achieving state-of-the-art accuracy for shallow CNN on small datasets (MNIST and CIFAR-10). However, when applied for large datasets and deep CNN, the accuracy of binary networks can hardly get recovered through re-training. Compared with the binary networks, ternary networks [3, 22] show increased capability

FPGA'19, February 24-26, 2019, Seaside, California, USA

ACM ISBN 978-1-4503-0000-0/18/06.
https://doi.org/00.0000/1234567890

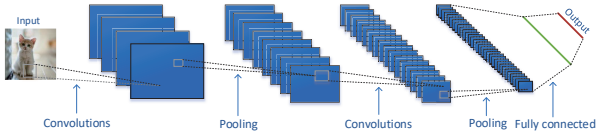

Figure 1: CNN architecture

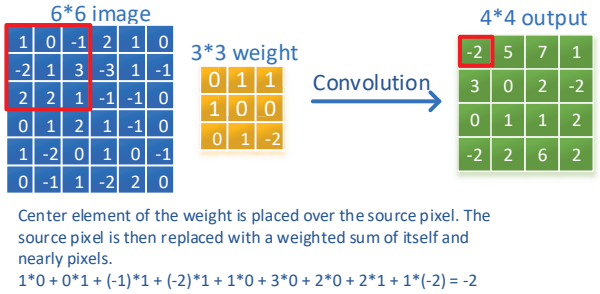

Figure 2: Convolution layer computation

in weights expressions. However, decrease in accuracy of binary networks is not negligible when applied to classification of IMAGENET datasets by Resnet-18 [22]. An efficient inference engine [11] can be applied to compress deep neural networks. Through the combination of pruning redundant connections, sharing weights of multiple connections and Huffman coding, the compress ratio of model can be pushed to 35-49x which is mainly ascribed to the redundant connections in the fully-connected layers. Moreover, the complexity of networks and over-fitting problems can be solved to some extent. However, extra space is in need for the storage of connection index. In the meanwhile, the sequential access to data stored in memory will break down owing to the prune of networks which is not favored for the performance.

To solve the above problems, in this paper, we made the following contributions: **first attempt to implemented CNN computing accelerator based on shift operation on FPGA**. The INQ method [29] is applied to compress pre-trained full-precision CNN weights to 4-bit integer weights gradually through weight partition, group-wise quantization and re-train. The 4-bit integer weights is either zero or power of two. The compressed weights transferred the MAC operations to SAC operations, which removed the dependence of computing on DSP units. Our program is designed for batchsize = 1 which is commonly used for real-time applications of CNN in autonomous driving [8, 19] and other cases where real-time processing is of great significance. **For SqueezeNet, the image processing latency is 0. 674ms based on Intel's Arria 10 FPGA embedded in F10A FPGA board produced by Inspur corporations, which is slightly better than that of NVIDIA P4. In the meanwhile, the compute capacity of FPGA increased by 1.77 times at least.**

2 BACKGROUND

2.1 CNN

Deep neural network (DNN) is a type of machine learning algorithm inspired by the human brain. It is widely used in the field of artificial intelligence. Convolutional neural network (CNN) is a very important part of the DNN algorithm applied in the field of computer vision, and it has achieved incredible performance on image classification and target detection [5].

The CNN consists of chains of multiple layers. The output of the each layer feed into the next layer for feature abstraction. The main operations of CNN layer contains: convolution layer, activation layer, pooling layer, fully-connected layer, etc., as is shown in Figure 1.

Convolutional layers: Convolutional layers are the core units of CNN. It sums up the locally associated data in the receptive field. As indicated by Figure 2, convolutional layers carry out the features extraction from local receptive fields across different channels. Compared with the fully-connected layer, the sharing of weights for convolutional layers decrease the model size greatly.

Activation layers: A convolutional layer can be viewed as the linear conversion of feature maps, which hardly match nonlinear situations. In order to enrich the expression capacity of network, activation functions such as Rectified Linear Unit (ReLU), Tanh and Sigmoid functions are applied to increase the nonlinearity representation capacity of the CNN.

Pooling layers: A pooling layer is often inserted after convolutional layer in CNN to reduce the dimension of feature maps. A pooling layer also improves the translational invariance of the model. As a consequence, the model will be less sensitive to the small translational changes of the input images.

Fully-connected layers: Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is terminated by a fully-connected layer. Neurons in a fully-connected layer have connections to all activations in the previous layer, as is seen in regular neural networks. Their activations are hence computed with a matrix multiplication followed by a bias offset.

2.2 Model Compression

Neural networks are computationally and storage intensive with high requirements for hardware resources, which increases the consumption of computation energy and the difficulty to be deployed on embedded systems with limited resources [12]. To alleviate these limitations, a variety of model compression methods are proposed, aiming at reducing the size of neural network.

Learning both weights and connections [14], the primary target of model compression is to learn important connections and prune unimportant connections. It includes three stages: train the network, prune unimportant connections and retrain the network. For example, it can reduce the number of parameters for AlexNet and VGG by a factor of 9X-13X without accuracy loss [14].

Based on the previous method, a deep compression [13] is proposed. In deep compression, synchronous trainings of weight and connection are introduced. First, the network gets pruned leaving only significant connections. Then weights sharing are carried out for the remaining connections. Finally, Huffman coding is applied

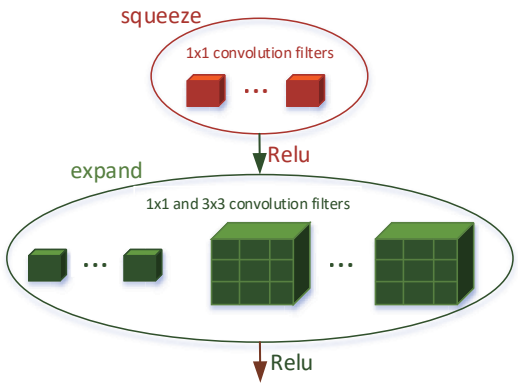

Figure 3: Architecture of Fire Module

to compress the model further. The deep compression method can reduce the model size of AlexNet and VGG by 35X and 49X.

To avoid the removal of significant connections, Dynamic Network Surgery (DNS) [10] is incorporated into the whole process. For DNS, the removed connections can get recovered during the retraining process if the pruned connections are found to be important. Compared to previous compress models, a 108X and 17.7X compress of LeNet-5 and AlexNet can be achieved.

Incremental Network Quantization (INQ) [29] can be applied to quantize the full-precision neural networks to low bit values of either zero or power of two. Compared with other quantization methods, the accuracy of INQ model can get recovered through 5-8 epochs of retrain and the accuracy of the quantized model can get improved to some extent [5]. In this study, the INQ method is adopted for compressing models and alleviating computation burdens.

2.3 SqueezeNet

Although the accuracy of deep convolutional neural networks have achieved state-of-the-art accuracy for computer vision, the depth of neural networks and corresponding model sizes increase dramatically. A network named SqueezeNet shows AlexNet level accuracy with only 1/50 of the model size [17]. SqueezeNet is a typical CNN, with convolutional layers, ReLU activation layers, max pooling layers and the last average pooling layers. The removal of fully-connected layers is the major reason accounting for the decrease of model size. The SqueezeNet is composed of squeeze modules made up of squeeze layers and following expand layers. The introduction of squeeze layers reduces the amount of parameters and channels for expand layers, and further decreases the model size.

SqueezeNet adopted in this paper is consisted of a standalone convolution layer (conv1), followed by 8 Fire Modules (fire2-9), ending with a final conv layer (conv10). Besides, there are 3 max pooling layers with stride equaling to 2, and an average pooling layer. The final Softmax layer outputs a 1000-element vector representing for 1000 possible classes of input images. The architecture of Fire module shows in Figure 3. The architecture of SqueezeNet shows in Table 1.

Table 1: Architecture of SqueezeNet

Layer	Width	Height	Channel
Data	227	227	3
Conv1	113	113	64
Maxpool	56	56	64
Fire2	56	56	128
Fire3	56	56	128
Fire4	28	28	256
Fire5	28	28	256
Maxpool	14	14	256
Fire6	14	14	384
Fire7	14	14	384
Fire8	14	14	512
Fire9	14	14	512
Conv10	14	14	1000
Avgpool	1	1	1000

2.4 Intel FPGA SDK For OpenCL

Traditionally, FPGA devices are programmed by hardware description languages, such as VHDL and Verilog [24], which is quite time-consuming. Intel FPGA SDK for OpenCL [2] improves the development efficiency of FPGA and facilitates the development of large-scale programs on FPGAs. With the assistance of compiler, all tedious processes ranging from establishing and controlling data paths, timing convergence problems to the connecting to the underlying IP cores using system-level tools can be solved by the SDK automatically. Consequently, programmers can focus on the algorithms. In the meanwhile, the OpenCL based programs can be easily updated and migrated to different FPGA devices, leaving all hardware related details to the SDK.

OpenCL uses a master-slave model, where a master host controls execution of kernels and memory transfers from host to devices. The device side calculates and returns calculation results to the host side. The main flowchart of developing heterogeneous programs using the Intel FPGA SDK shown in Figure 4 are classified as:

(1) Design the host program and allocate of memory for host and device;
(2) Design the device kernels for calculations;
(3) Compile the device program and load the programs to the FPGA devices;
(4) Call host segment program to run the entire application.

3 IMPLEMENTATION

3.1 Design Goals

For this inference accelerator, our design goal is to improve the performance, energy efficiency of FPGA and to expand the applications of FPGA to CNN. For current implementations of deep convolution neural networks, what hinders the applications of CNN are:

1. Increased network depths and model size

For most CNN, model parameters and intermediate feature map data produced during the inference processes occupy hundreds of megabytes memory. However, because of the 6MB on-chip M20K

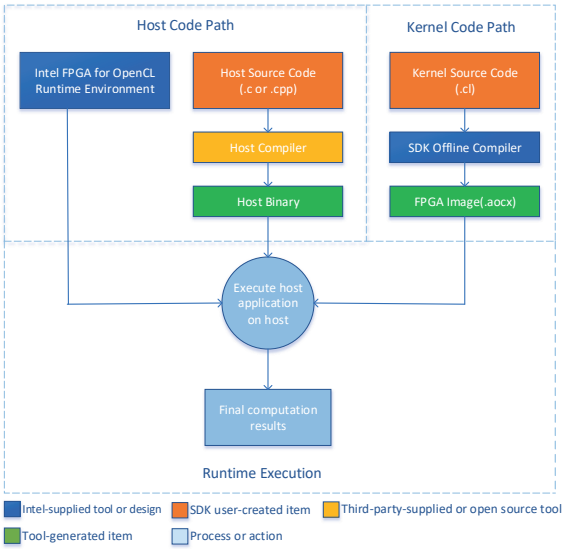

Figure 4: The main flowchart of developing heterogeneous programs using the Intel FPGA SDK [18]

memory of FPGA [1], model parameters or feature map data must store in external DRAM memory, frequent access to external DRAM memory leads to higher energy consumption (Table 2). In order to avoid the influence on computing performance of FPGA, a high DRAM transmission bandwidth is in need. For neural network of single precision floating point computation, if 125 data per clock cycle is required to transmission, and the running clock frequency of the FPGA is 250 MHz, then the required DRAM transmission bandwidth is $125 * 4 * 250 = 125GB/s$. This is almost impossible for all types of DRAM.

Table 2: Energy table for 45nm CMOS process [16]

Operation	Energy[pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM memory	**640**	**6400**

2. Intensive MAC

In most CNN, there could be hundreds to even up to billions of multiply and accumulate (MAC) operations for the inference of a single image, ascribed to intensive convolution calculation and fully-connected calculation. For SqueezeNet, the amount of floating-point multiplication operations is 387.75M. Intensive MAC operations impose pressure on the high performance of DSP units, which affects the parallelism of FPGA computing. In the meanwhile, the introduction of DSP units also improves the cost and energy consumption of FPGA devices [9].

Accordingly, we improved the computational performance and energy efficiency of FPGA by reducing the amount of data transferred between external DRAM memory and FPGA and eliminating the computational dependence of DSP.

3.2 Design Scheme

In order to improve the performance and energy efficiency of inference programs on FPGA, the solutions of model compression and quantization calculation are put forward:

(1) The INQ method are applied to compress the 32-bit floating-point model to 4-bit integer, which represents the values of either zero or power of two, so the model data size can be compressed to 1/8 compared to the original one;
(2) Feature map data are quantified to 8-bit integer with a compress ratio of 4;
(3) Convert the MAC operations of features and compressed weights to SAC operations, and the latter one is more efficient for hardware.

Compress process mentioned above will alleviate the requirement for model storage space, and the bandwidth for model parameter transmission from external DRAM memory to FPGA. The feature map data quantized to 8-bit can be stored in FPGA M20K memory, further reducing the transformation and computation cost. Moreover, a preferable performance and higher energy efficiency are achieved with quantized feature map data, compressed model data, and the removal of MAC operations highly dependent on DSP units.

3.3 Model Compression and Quantization

3.3.1 Model Compression and Improvement. INQ method applied to quantize different CNN includes three processes, namely weight partition, group-wise quantization and re-training [29]. Weights of convolutional layer are divided into two groups according to their size in the weight partition process. For the l^{th} layer of neural network, the weights grouping is defined as following:

$$A_l^{(1)} \cup A_l^{(2)} = \{W_l(i,j)\}, \; A_l^{(1)} \cap A_l^{(2)} = \Phi \tag{1}$$

Among them, $A_l^{(1)}$ represents the weights group to be quantified, $A_l^{(2)}$ indicates the weights group that needs to be retrained to compensate for the accuracy loss.

Then weights of one group $A_l^{(1)}$ can be transferred to either powers of two or zero from the following set:

$$P_l = \left\{\pm 2^{n1}, \ldots \pm 2^{n2}, 0\right\} \tag{2}$$

For the INQ algorithm, the bit number can be set in advance (e.g.,3-bit,4-bit,5-bit) which is calculated as:

$$n_1 = floor\left(\log_2 \frac{4 \times max(abs(W_l))}{3}\right), \tag{3}$$

where $floor(\cdot)$ represents the down rounding number, function $max(\cdot)$ calculates the maximum value of all weights. The value of n_2 depends on the value of n_1 expressed as $n_2 = n_1 + 2 - 2^{(b-1)}$. In this way, the weights can be quantized to either $[-2^{n_1}, -2^{n_2}]$ or $[2^{n_2}, 2^{n_1}]$ $(n_2 \leq n_1)$.

Table 3: A accuracy of SqueezeNet before and after compression

Accuracy	Original	Compressed	Increase in top-1/top-5 accuracy
Top1	57.50%	59.00%	1.50%
Top5	80.30%	80.40%	0.10%

Table 4: The relationship between quantized parameters and 4-bit code

4-bit code	0000	0001	0010	0011
parameter	$-2^{\exp}$	$-2^{\exp+1}$	$-2^{\exp+2}$	$-2^{\exp+3}$
4-bit code	0100	0101	0110	0111
parameter	$-2^{\exp+4}$	$-2^{\exp+5}$	$-2^{\exp+6}$	0.0
4-bit code	1000	1001	1010	1011
parameter	$2^{-\exp}$	$2^{-(\exp+1)}$	$2^{-(\exp+2)}$	$2^{-(\exp+3)}$
4-bit code	1100	1101	1110	1111
parameter	$2^{-(\exp+4)}$	$2^{-(\exp+5)}$	$-2^{-(\exp+6)}$	N/A

The weights conversion are based on the following rules:

$$\widehat{W}_l(i,j) = \begin{cases} \beta \mathrm{sgn}(W_l(i,j)) & if\ (\alpha+\beta)/2 \le abs(W_l(i,j)) \le 3\beta/2 \\ 0 & otherwise \end{cases} \tag{4}$$

where α and β are adjacent elements in set P_l.

Finally, the whole neural network is retrained to compensate for the accuracy loss caused by weight quantization. During retraining, the quantized weights remain unchanged with other weights updated. The process is repeated until all weights are quantized. For SqueezeNet, the accuracy before and after compression is shown in Table 3. As we can see from the table, the Top1 or Top5 accuracy improved respectively when the model is compressed.

Compared with the original INQ compression method, we improved the representation of the compression model. Taking 5-bit quantization as an example, 1 bit is used to represent zero value, and the remaining 4 bits are to represent at most 16 different values for the powers of two. That is, the number of candidate quantum value is at most $2^{b-1}+1$, and the number of $2^b - 2^{b-1} - 1$ value is not used. In our improved compression method, we use 4 bits to represent 4-bit compressed data. Compared with previous quantization methods, we use almost all of the value to represent compressed model, and what's more, multiple 4-bit data can be easily combined into char/short/int and other types of data used in OpenCL, which is convenient for data transmission and FPGA processing. For each layer parameters of compressed model, parameter *exp* is used to represents the minimum exponential. The relationship between actual compressed parameters and 4-bit code is shown in Table 4. For example, if exp = −7 and the 4-bit code is *0001*, the actual compressed parameter will be $-2^{-7+1} = -0.015625$.

3.3.2 Quantization. The compressed weights are expressed as:

$$Qweight = (-1)^s * 2^m, \tag{5}$$

where *s* is the sign of weight, *m* represents to *(exp + i)* and *-(exp + i)* in Table 4 and is less than 0 in most cases.

In the meanwhile, the 8-bit quantization [20, 26] of feature map data are adopted to make a compromise between representation accuracy and data range. Suppose the input feature map data is *feature*, the input weight data is *weight*, the channel of input feature map is *N*, the size of weight is *k*k*, and output of convolution is *result*. Convolution calculation is expressed as:

$$result = \left(\sum_{i=1}^{N} \sum_{j=1}^{k*k} feature[i][j] * weight[i][j] \right) + bias \tag{6}$$

Suppose the 8-bit quantified feature map data is *Qfeature*, which can be expressed as:

$$Qfeature = feature * 2^{-Q}, \tag{7}$$

where Q is quantization coefficient, its calculation equation is:

$$Q = log_2^{\max\left(fabs\left(feature(i)\right)\right) * ratio}, \tag{8}$$

where $max(fabs(feature(i)))$ represents the maximum absolute value of all input data, *feature(i)* represents the value of single data. *ratio* can be adjusted between 0.8~1.0.

Assuming that the quantified input feature map data is *Qfeature*, the quantization coefficient is *Q1*, the result of convolution is *Qresult*, and the quantization coefficient of result is *Q2*, then the equation (6) is converted to:

$$Qresult * 2^{Q2} = \left(\sum_{i=1}^{N} \sum_{j=1}^{k*k} Qfeature[i,j] * 2^{Q1} * weight[i,j] \right) + bias \tag{9}$$

The equation (9) can be further converted to:

$$Qresult = \left(\sum_{i=1}^{N} \sum_{j=1}^{k*k} Qfeature[i][j] * 2^{Q1-Q2} * weight[i,j] \right) + bias * 2^{-Q2} \tag{10}$$

What can be obtained from equation (10) and equation (5) is:

$$Qresult\_part = \sum_{i=1}^{N} \sum_{j=1}^{k*k} Qfeature[i][j] * 2^{Q1-Q2} * (-1)^s * 2^m \tag{11}$$

The MAC operations of weights and feature maps heavily relied on the DSP resources can be expressed as:

$$Qresult\_part = \sum_{i=1}^{N} \sum_{j=1}^{k*k} \Big(Qfeature[i,j] \ll (Q1 - Q2 + m) \Big) * (-1)^s \tag{12}$$

Because m is negative in most cases, the gap between *Q1* and *Q2* is small, so $(Q1 - Q2 + m)$ is less than 0 in most cases. If $(Q1 - Q2 + m) < 0$, the equation (12) is converted to:

$$Qresult\_part = \sum_{i=1}^{N} \sum_{j=1}^{k*k} \left(\Big(Qfeature[i,j] \gg \big(-(Q1 - Q2 + m)\big) \Big) * (-1)^s \right) \tag{13}$$

The direct transformation of MAC to right-shift and accumulate operations lead to unneglectable errors especially when the number of accumulate operation is large. For example, with the increase of

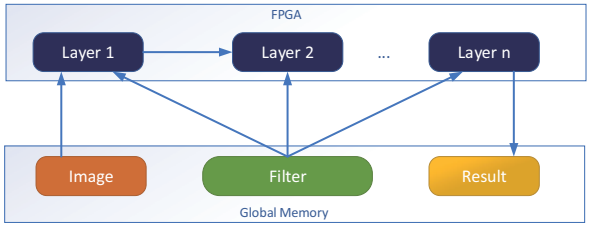

Figure 5: The overall architecture

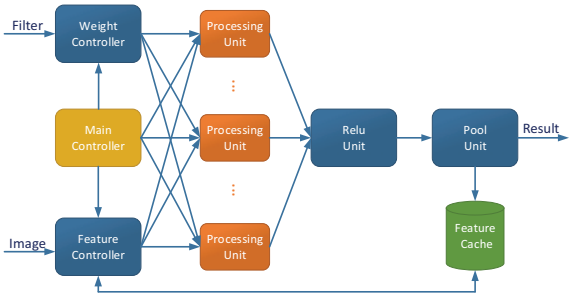

Figure 6: The single layer computing architecture

input channel number and weight size, the accumulate operation will increase. Therefore, an inflation factor $Inflat$ is introduced to transfer the right-shift operations to left-shift operations and after the accumulation operation the corresponding results are right-shifted to get the value back, as shown in equation (14) and equation (15). After the convolution operations, corresponding rounding operations are adopted to compensate for the limited data representation capacity of 8-bit integer and improve the calculation accuracy to some extent.

$$Qresult = \sum_{i=1}^{N}\sum_{j=1}^{k*k}\Big(\Big(Qfeature1\,[i,j] \ll (Inflat + Q1 - Q2 + m)\Big)* (-1)^{s}\Big) + bias*2^{inflat-Q2} \tag{14}$$

$$rounding\_result = (Qresults \gg (Inflat - 1) + 1) \gg 1 \tag{15}$$

3.4 FPGA Implementation

3.4.1 Overall Architecture. The deep convolution neural network inference process is implemented on FPGA. In sequential processing of multi-layer neural networks, convolution, activation and pooling layers are processed in parallel with data transferred between kernels through channels. Owing to the restriction of on-chip memory, the weight data are stored in external memory and transferred to the kernels during inference computation process. In our design, the data transmission process is concealed by the parallel execution of kernels on FPGA, which avoids the influence on the computing performance of FPGA. The overall architecture for our design is shown in Figure 5.

Single layer computing architecture is shown in Figure 6. The main functional module of the programs can be classified as Weight Controller, Feature Controller, Main Controller and Processing Unit. The Weight Controller is responsible for loading weight data from external DRAM memory. The Feature Controller is used to store and load feature map data from on-chip M20K RAM, and then send them to Processing Unit. It has a double buffer, which can be read and written at the same time in each clock cycle, so the storage of pool output feature map data and the reading of convolution calculation feature map data can be carried out simultaneously. Main Controller generates signals of flow control of the whole programs, access address for weight cache in the Processing unit and feature cache in the Feature Cache.

The Processing Unit is the core computing unit for intensive convolution computation. It has a double buffer, which can be read and written at the same time in each clock cycle, so the storage of the next group of weight data and the reading of the current group weight data can be carried out simultaneously. Therefore, in every cycle of FPGA running, it can be calculated effectively. To achieve a higher throughput, Processing Units are duplicated and parallelized. The Outputs of Processing Unit are sent to the following ReLU Unit and Pooling Unit. Pooling outputs are sent to feature Controller for successive inferences. As a consequence, the whole inference process can be achieved.

3.4.2 Convolution Implementation. To take full advantage of FPGA device, input channel, column of input feature map, output channel, column of output feature map and column of the convolution kernel are vectorized [2].

Vectorization values for input channel, column of input feature, output channel, column of output feature map, weight column are defined as *IN_C_VEC, IN_W_VEC, OUT_C_VEC* and *OUT_W_VEC*, respectively. In the meanwhile, the convolution kernels are vectorized by a factor of *F_W_VEC*. It should be noted that when the convolution kernel size is greater than 1, *F_W_VEC* is set to 3, and when the convolution kernel size is 1, *F_W_VEC* is set to 1. The stride of convolution is fixed to 1. Therefore, $OUT\_W\_VEC = IN\_W\_VEC - F\_W\_VEC + 1$.

Assuming that the channel number of the input feature map is *IN_C*, the height and width of the convolution kernel are *F_H* and *F_W*. In a clock cycle, for a single output channel, the *IN_C_VEC * F_W_VEC* input feature map data can be computed, so the number of clock cycles *N*, which requires to calculate *OUT_W_VEC* output feature map data for a single channel is calculated as:

$$N = ceil\left(\frac{IN\_C}{IN\_C\_VEC}\right) * ceil\left(\frac{F\_W}{F\_W\_VEC}\right) * F\_H \tag{16}$$

The convolution calculation process of a single output channel shows in the Figure 7. The graph assumes *IN_W_VEC = 3, F_H = F_W = 3, F_W_VEC = 3, IN_C = 4, IN_C_VEC = 2, OUT_C_VEC = 4*, therefore, *OUT_W_VEC = 3 − 3 + 1 = 1*. The clock cycles N of a single output channel is$(4/2) * (3/3) * 3 = 6$. The "<<+" sign in the figure indicates that the shift-addition computation of feature data and weight data.

Assuming that the width, height and channel number of output feature map are *OUT_W, OUT_H, OUT_C*, respectively. The number of clock cycles needed to complete all output feature maps of the

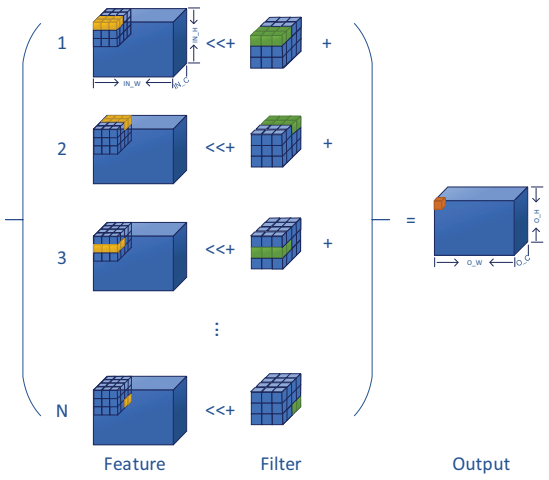

Figure 7: The convolution calculation process of a single output channel: the graph assumes $IN\_W\_VEC = 3$, $F\_H = F\_W = 3$, $F\_W\_VEC = 3$, $IN\_C = 4$, $IN\_C\_VEC = 2$, $OUT\_C\_VEC = 4$, **therefore,** $OUT\_W\_VEC = 3 - 3 + 1 = 1$**. The clock cycles N of a single output channel is** $(4/2)*(3/3)*3 = 6$**.**

current layers is calculated as:

$$M = ceil\left(\frac{OUT\_W}{OUT\_W\_VEC}\right) * OUT\_H * ceil\left(\frac{OUT\_C}{OUT\_C\_VEC}\right) \quad (17)$$

3.4.3 Processing Unit. Processing Unit performs the shift-addition calculation is shown in Figure 7. After N clock cycles, it outputs the convolution result of OUT_W_VEC. OUT_C_VEC Processing Unit parallelism calculates the convolution of OUT_C_VEC output channels.

The code for the convolution calculation of Processing Unit is shown below. For simplicity, the sign of feature and weights are neglected in the code. Weigh in the below code represents the value of $(Inflat + Q1 - Q2 + m)$ in equation (14).

```
void compute(feature, weight)
{
 int Result[OUT_C_VEC] = { 0 };
 for (int i = 0; i < N; i++)
 {
   #pragma unroll
   for(int j = 0; j < OUT_W_VEC; j++)
   {
     #pragma unroll
     for(int k = 0; k < F_W_VEC; k++)
     {
       #pragma unroll
       for(int m = 0; m < IN_C_VEC; m++)
       {
       Result[j] += feature[j+k][m] << \
       weight[k][m];
       }
     }
   }
 }
}
```

After the calculation shown above, the convolution result is in 32-bit integer type, and we convert it back to 8-bit integer type. According to equation (15), the code is as follows. In the code, we implement ReLU calculation.

```
void conversion()
{
  char output[OUT_C_VEC] = { 0 };
  #pragma unroll
  for(int i = 0; i < OUT_W_VEC; i++)
  {
    char temp = 0;
    if(result[i] > 0)
    {
      short int stemp = ((result[i] >> \
      (Inflat - 1)) + 1) >> 1;
      temp = stemp > 128 ? 128 : stemp & 0x7f;
    }
      output[i] = temp;
  }
}
```

4 RESULTS

4.1 Validation

The FPGA chip selected for this experiments is the Intel Arria 10 GX1150 chip embedded in the F10A FPGA board produced by Inspur corporations. The software environment is the Intel SDK for OpenCL 16.1 with the BSP provided by the Inspur FPGA develop team. The peak computing power of a single chip is 1.5 TFlops, and the power consumption is about 45 W, with a characteristic of 34 GFlops per watt. BSP (Board Support Package) is a layer between the motherboard hardware and the driver program in the operating system. It provides a function package for the upper driver to access the hardware device registers and make it run better on the hardware motherboard. Currently, SqueezeNet is implemented based on the accelerator.

To compare accuracy and throughput of the scheme based on SAC operations, the MAC operations of 8-bit and 6-bit quantization of weights and features are also realized. In order to improve the performance of MAC, 1D Winograd Transformation are adopted to reduce the MAC operations needs [27]. However, it can hardly take any advantage to the quantized weights of either zero or power of two because it will transform them back to floating-point data, so the Winograd Transformation is removed when convert the MAC operations to SAC operations.

It is worth noting that, in our design, we also implemented floating-point data type computation. However, with $IN\_C\_VEC = 4$, $IN\_W\_VEC = 4$ and $OUT\_C\_VEC = 4$, the compilation of FPGA code fails because the M20K memory need is far beyond the M20K resources. Therefore, the performance of the floating-point version is not listed. Currently, all accuracy is tested based on the IMAGENET dataset.

Table 5: The accuracy of different implementations of SqueezeNet

Accuracy	Original	Compressed	8-bit	6-bit	Shift-Accumulate
Top1	57.5%	59.00%	57.52%	57.68%	57.62%
Top5	80.3%	80.40%	80.10%	79.99%	79.98%

The accuracy of the original model, compressed model, and the different implementations are shown in Table 5. Judging from the Table 5, the compressing process improves the accuracy of the SqueezeNet by nearly 1.5% for top 1 accuracy, 0.1% for top 5 accuracy, but the quantization computing reduced the accuracy by about 1.4% and 0.3%.

For dynamic 8-bit and 6-bit quantization methods, these quantization are based on locally sharing-exponent method [2]. Integer quantization of weights and features will decrease the accuracy ascribed to the limited representation capacity of lower bit integers. The number of feature map data or weight data that share the same exponent is fixed to 4, instead of *IN_C_VEC*, because the increase of *IN_C_VEC* will lead to the decrease of calculation accuracy. What amazing and exciting is that the accuracy of 8-bit and 6-bit quantization method is almost the same, which means the data features represented by 6-bit are almost the same as those represented by 8-bit. Therefore, we can calculate the deep neural network with fewer computing bits, and it implies less resources and power consumption.

For the shift-accumulate algorithm, there is no accuracy loss for the representation of the weights while the representation of features is the main reason that accounts for the accuracy loss. According to our analysis, only the quantization of feature map data leads to about 1.4% accuracy loss is because the quantization is based on all the input feature map data of the current layer. But for 8-bit or 6-bit quantization, it is only based on 4 feature map data. Therefore, reducing the quantized data density can increase the computation accuracy to a certain extent, for example, we can use each input channel data as a unit to quantify. Although there are different extents of accuracy loss in the top 1 and top 5 accuracy, the accuracy for the inference process is basically the same as the original one.

4.2 Resource

For FPGA programs, the throughput are mainly restricted by the capacity of Logic Elements, RAM Blocks, DSP and frequency of the programs. The values of *IN_C_VEC*, *IN_W_VEC* and *OUT_C_VEC* determine the number of the SAC and MAC operations done in a single cycle, so they influence the throughput directly. However, increasing these values also lead to dramatic increase in the hardware resources and decrease of frequency. Currently, the optimum performance can be realized when the configuration of *IN_C_VEC = 16, IN_W_VEC = 9* and *OUT_C_VEC = 16* for SAC operations, and the corresponding value is 8, 6 and 16 for 8-bit or 6-bit MAC operations. The used resources and frequency of different implementations listed in Table 6. It should be noted that the 6-bit quantization method are the same to that of 8-bit method, the only difference lies in the value of local sharing exponent, so the used resources is same.

Table 6: The used resources and frequency of different implementations

Implementations	Logic	Memory	DSP	Freq (MHz)
Shift-Accumulate	85%	84%	4%	230
8-bit	65%	72%	80%	213
6-bit	65%	72%	80%	213

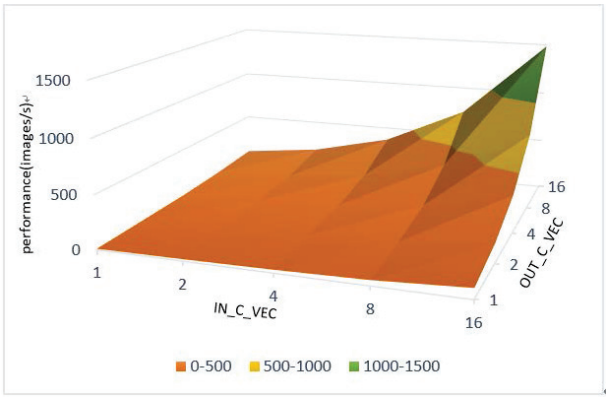

Figure 8: Performance of various *IN_C_VEC* and *OUT_C_VEC* with *IN_W_VEC* = 9

From the results shown in Table 6, the 8-bit and 6-bit quantization method are mainly restricted by the DSP units on the FPGA. Further increasing the value of *IN_C_VEC* or *OUT_C_VEC* will run out of all DSP units on FPGAs. The SAC method is mainly restricted by Logic Elements and Memory Blocks.

It is worth noting that, for 8-bit or 6-bit MAC implementation, DSP is used by quantization of feature map data and weight data, Winograd Transformation, MAC computation, and computation of the access address of weight cache and feature cache. However, for SAC implementation, DSP is only used by computation of the access address of weight cache and feature cache.

For SAC implementation, with *IN_C_VEC = 16, OUT_W_VEC = 7, F_W_VEC = 3, OUT_C_VEC = 16*, in a clock cycle, the number of SAC computed parallelly is *16 * 7 * 3 * 16 = 5376*. But the number of 18 x 19 multipliers of Arria GX1150 is 3036 [9], **so we increased the computing capacity of FPGA by 1.77 times at least.**

4.3 Performance

Performance of various *IN_C_VEC* and *OUT_C_VEC* with *IN_W_VEC = 9* show as Figure 8. We can see from Figure 8 is that computing performance increasing proportionally with the increase of *IN_C_VEC* and *OUT_C_VEC*.

For the Shift-Accumulate implementation, when *IN_C_VEC = 16, IN_W_VEC = 9* and *OUT_C_VEC = 16*, the total cycles to processing a image is 150732 computed by equation (17). Because FPGA running frequency is 230MHz, theoretically, the time to process a image is $150732 * (1 / 230 * 10^6) = 0.655ms$. Compared with the measured

Table 7: Computational Efficiency of F10A

Device	Theoretical	Measured	Efficiency
F10A	0.655ms	0.673ms	97.3%

Table 8: Accuracy of F10A and P4 of different implementations

Device	Implementations	Top1 accuracy	Top5 accuracy
F10A	Shift-Accumulate	57.62%	79.98%
F10A	8-bit	57.52%	80.10%
P4	Floating-point	58.14%	80.79%
P4	8-bit	56.79%	79.76%

Table 9: Performance of F10A and P4 of different implementations

Device	Implementations	Peak Power (Watts)	Latency (ms)
F10A	Shift-Accumulate	45	0.673
F10A	8-bit	45	2.481
P4	Floating-point	75	0.756
P4	8-bit	75	0.687

performance, the computational efficiency of FPGA is *0.655/0.673 = 97.3%*, shown in Table 7. Even though the computational efficiency is relatively high, we still try to find out the reasons that affect the computational efficiency. Using OpenCL, we can see the running status of the FPGA through the profile function, for example, external memory bandwidth, channel access status, M20K RAM access status, etc.. But when adding profile function to compilation, the compilation failed because the resources need increase and beyond the resources limit of FPGA. Since the balance of the computing speed of each kernel are considered in the design, the reason for this phenomenon may be that there is no good synchronization between the kernels.

The accuracy and best throughput for 8-bit and Shift-Accumulate implementation are shown in Table 8 and Table 9. In the meanwhile, the floating-point and 8-bit quantized SqueezeNet are also tested on NVIDIA P4 platform with TensorRT 4. For F10A, **the lowest latency is 0.673ms, which is a little better than P4 of 0.687ms**. With the best throughput, **the Top1 and Top5 accuracy of FPGA is about 0.5% higher than that of P4**. This is the best performance we have ever seen.

5 CONCLUSION

In this paper, we introduce an acceleration engine based on INQ model compression method and shift-accumulate calculation. This engine removes the strong dependence of DSP processing ability of neural network computation and improves the processing ability of FPGA. After test the SqueezeNet with IMAGENET dataset. The accuracy and latency are slightly better than that of GPU P4.

However, the scheme also has some drawbacks, without fully-connected layer or batchnorm layer calculation. In a fully-connected layer, the calculation is similar to convolution layer, but with more model parameters. Because this scheme compresses the model parameters, and the transformation of them parallel with computation, the reading of the model parameters will not have a great impact on the computational performance. In a batchnorm layer, it is only the basic multiplication and addition operation, so it does not occupy too much logic and DSP resources. In short, after preliminary analysis, the subsequent addition of fully-connected layer, batchnorm layer and other calculations will not have a great effect on the overall performance.

From the Table 6, we can see that the frequency of two implementations are 213MHz and 230MHz. As far as we know, normal working frequency of FPGA can reach to 500MHz, so improve the frequency of program can increase the throughput significantly. We tried many methods, such as finding the critical path to optimize the code, adding random seed to the compilation options and so on, without increasing the clock frequency. Therefore, how to improve the running frequency of program with OpenCL is still an important aspect in our future study.

Future work also includes mapping other CNN such as ResNet, GoogLeNet and AlexNet to our architecture, combining network pruning with compression algorithm, and exploring how run-time configurability may affect performance of our architecture.

ACKNOWLEDGMENTS

We would like to thanks the contributions of Shaohua Wu, Qikai Xie, Jiwei Zhang, Shu Liu and Tong Yu.

REFERENCES

[1] Intel Arria. 2017. Device Overview.
[2] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C Ling, and Gordon R Chiu. 2017. An OpenCL™ deep learning accelerator on arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 55–64.
[3] Yoonho Boo and Wonyong Sung. 2017. Structured sparse ternary weight coding of deep neural networks for efficient hardware implementations. In *Signal Processing Systems (SiPS), 2017 IEEE International Workshop on*. IEEE, 1–6.
[4] L-W Chan and Frank Fallside. 1987. An adaptive training algorithm for back propagation networks. *Computer speech & language* 2, 3-4 (1987), 205–218.
[5] Convolutional neural network 2018. Convolutional neural network. https://en.wikipedia.org/wiki/Convolutional_neural_network.
[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.
[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).
[8] Mike Foedisch and Aya Takeuchi. 2004. Adaptive real-time road detection using neural networks. In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*. IEEE, 167–172.
[9] Sameh Galal and Mark Horowitz. 2011. Energy-efficient floating-point unit design. *IEEE Transactions on computers* 60, 7 (2011), 913–922.
[10] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*. 1379–1387.
[11] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 243–254.
[12] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[13] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.

arXiv preprint arXiv:1510.00149 (2015).

[14] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision*. 630–645.

[16] Mark Horowitz. [n. d.]. Energy table for 45nm process.

[17] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[18] FPGA Intel. 2017. SDK for OpenCL Programming Guide. *UG-OCL002* 8 (2017).

[19] Cheng-Bin Jin, Shengzhe Li, Trung Dung Do, and Hakil Kim. 2015. Real-time human action recognition using CNN over temporal images for static video surveillance cameras. In *Pacific Rim Conference on Multimedia*. Springer, 330–339.

[20] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. 2015. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236* (2015).

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[22] Fengfu Li and Bin Liu. 2016. Ternary Weight Networks. *CoRR* abs/1605.04711 (2016). arXiv:1605.04711 http://arxiv.org/abs/1605.04711

[23] P. Štukjunger M. Bečvář. 2005. Fixed-point arithmetic in FPGA. *Acta Polytechnica* 45, 2 (2005), 389–393.

[24] Armaan Hasan Nagpurwala, C Sundaresan, and CVS Chaitanya. 2013. Implementation of HDLC controller design using Verilog HDL. In *Electrical, Electronics and System Engineering (ICEESE), 2013 International Conference on*. IEEE, 7–10.

[25] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. 2016. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 1–4.

[26] G Alonzo Vera, Marios Pattichis, and James Lyke. 2011. A dynamic dual fixed-point arithmetic architecture for FPGAs. *International Journal of Reconfigurable Computing* 2011 (2011).

[27] Shmuel Winograd. 1980. *Arithmetic complexity of computations*. Vol. 33. Siam.

[28] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 161–170.

[29] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).