Machine Intelligence Research
www.mi-research.net

21(6), December 2024, 1192-1200
DOI: 10.1007/s11633-023-1434-8

Branch Convolution Quantization for Object Detection

Miao Li Feng Zhang Cuiting Zhang

The National Engineering & Technology Research Center for Application Specific Integrated Circuit Design (NECFAD), Institute of Automation, Chinese Academy of Sciences, Beijing 100080, China

Abstract: Quantization is one of the research topics on lightweight and edge-deployed convolutional neural networks (CNNs). Usually, the activation and weight bit-widths between layers are inconsistent to ensure good performance of CNN, meaning that dedicated hardware has to be designed for specific layers. In this work, we explore a unified quantization method with extremely low-bit quantized weights for all layers. We use thermometer coding to convert the 8-bit RGB input images to the same bit-width as that of the activations of middle layers. For the quantization of the results of the last layer, we propose a branch convolution quantization (BCQ) method. Together with the extremely low-bit quantization of the weights, the deployment of the network on circuits will be simpler than that of other works and consistent throughout all the layers including the first layer and the last layer. Taking tiny_yolo_v3 and yolo_v3 on VOC and COCO datasets as examples, the feasibility of thermometer coding on input images and branch convolution quantization on output results is verified. Finally, tiny_yolo_v3 is deployed on FPGA, which further demonstrates the high performance of the proposed algorithm on hardware.

Keywords: Branch convolution quantization, thermometer coding, extremely low-bit quantization, hardware deployment, object detection.

Citation: M. Li, F. Zhang, C. Zhang. Branch convolution quantization for object detection. *Machine Intelligence Research*, vol.21, no.6, pp.1192–1200, 2024. http://doi.org/10.1007/s11633-023-1434-8

1 Introduction

Convolutional neural networks (CNNs) perform well in a variety of tasks, extending their applications to more complex situations. However, the improvement of network performance is often accompanied by the explosive growth of parameters and computing resources. For edge device deployment, networks are usually compressed, 8-bit quantization of weights and activations for example. However, for ultra-low energy applications, the required computing resources, storage resources and power consumption still cannot meet the demands, and thus researches on extremely low-bit quantization of networks have received lots of attention. In the earliest extremely low-bit quantization algorithms, 32-bit float weights are replaced directly by their signs, such as BinaryConnect[1], Bitwise Neural Networks[2] and Binarized Neural Network[3]. Afterwards, algorithms such as BWN[4], XNOR-Net[4] and DoReFa-Net[5] utilize scaling factors to reduce the accuracy loss in weight quantization.

For network accuracy, there are two consensuses on extremely low-bit quantization on object detection networks. One is that the weights of the middle layers can be quantized to as extremely low as 1 bit, but the quantization of the weights cannot be applied to the last layer as it says "It is a common sense that the first layer and the last layer should be kept in higher precision, which means that these layers play a more important role in the prediction of neural networks." in [6]; the other is to quantize the activations of the middle layers to 4 bits, but quantization of the input images and the output results is forbidden. This is due to unreasonable results resulting from unpredictable noises introduced by the quantization of input images, as well as deteriorated accuracy of the network on account of the quantization which strictly restricts the range of output results.

The problem brought about by the two consensuses is that the first layer and the last layer must be deployed on circuits separately from the middle layers and different computing and storage structures are required when the network is compressed with extremely low bits. Thus, the size of the circuits and the complexity of data scheduling will increase dramatically.

To solve the above problem, we propose a branch convolution quantization (BCQ) method aiming at object detection. This method can unify the whole network with extremely low-bit quantization, and all layers can be deployed on one hardware module, avoiding extra circuits for the first layer and the last layer. Our contributions include:

1) The weights are binarized and the activations are quantized to 4-bit in every layer of the object detection network. The hardware module on which every layer is

Research Article
Manuscript received on December 23, 2022; accepted on March 2, 2023
Recommended by Associate Editor Deng-Ping Fan
Colored figures are available in the online version at https://link.springer.com/journal/11633

implemented is designed based on these bit-widths.

2) Thermometer coding is utilized to convert 3-channel 8-bit RGB images to 51-channel 4-bit inputs to meet the bit-width of the above designed module. The convolution kernels in the first layer are also copied and converted to 51 channels as needed.

3) BCQ is utilized to quantize the output results of the last layer to satisfy the unified bit-width of the whole network.

2 Related works

2.1 Extremely low-bit quantized weights of all layers

Yang et al.[7] propose a differentiable method to quantize weights and activations, but the input and output layers are not quantized in both classification and object detection tasks. The weights and activations are quantized to 4-bit in [8]. The experiments in [9] show that the bit-widths of weights and activations cannot be extremely compressed to maintain the accuracy of the object detection network. The feature extracting part in [10] has mixed precision, where the bit-width of the weights and activations are 8-bit, but the batch normalization parameters are still in 32-bit. Guo et al.[11] quantize the network with mixed fixed points and binaries, reducing the bandwidth and computational complexity. Nguyen et al.[12] use the binary method, but the last layer is still quantized to 8-bit weights and 16-bit activations inevitably. There are some other quantization methods, such as [13], that are based on incremental quantization or divide weights into several blocks represented by one number each. For example, Sakuma et al.[14] approximate all the weights to the power of 2 so that the multiplication can be substituted by shifting in hardware. Cardinaux et al.[15] use LUT quantization to achieve a weight dictionary with k-means, and the 32-bit weights correspond to a certain value in the dictionary. Fang et al.[16] show that the weights have bell-shaped distribution with a long tail by statistical analysis, and the range of quantization is divided into two non-overlapping halves with the same quantization levels, minimizing quantization error. Park et al.[17] propose a quantization method based on weighted entropy, which is network-dependent and not universally appliable.

2.2 Quantization of input images and output results

For the quantization of input images, every 8-bit RGB pixel in [18] is decomposed into 1-bit, such that there is only bit operation in all network layers and all convolutions are deployed within one computing logic without any dedicated logics. Guo et al.[19] propose a fully binarized neural network accelerator. The quantization consistency of the input and middle layers is achieved by binary quantization and pruning, and odd-even padding is designed to solve the padding problem of the input feature maps. So far, the effectiveness of these quantization methods on input images has only been verified on classification, and further examination is required on object detection.

For the quantization of the output results of the network, Zhu et al.[20] use the Boost to take the BNN with binarized weights and activations as a weak classifier, and multiple BNNs are trained via AdaBoost to get the final classification results. This algorithm is insensitive to the quantization of the input layer and output layer, however, Boost is only suitable for classification tasks, instead of regression tasks as in object detection. Therefore, they only talk about the classification accuracy and do not mention the application in object detection. The quantization of weights and activations of middle layers in [12] is 1 bit and 3–6 bit respectively, but the input images are still in 8-bit RGB. The weights of the output layers are 8-bit and the outputs are 16-bit.

3 Algorithm background

Network quantization includes the quantization of weights and activations. As depicted in Fig. 1, typically in object detection, quantization involves only the weights and activations in the backbone and head, excluding the input images and output results. The proposed algorithm of this paper unifies the quantization bit-width of the whole network, in order to deploy the first layer, the last layer and the middle layers on the same hardware module. There are two reasons why we want to deploy the whole network on the same hardware module. First, if the first layer and the last layer are deployed separated from the middle layers, the data scheduling will be more complex among different hardware modules. In contrast, the data scheduling will be simpler within one hardware module. Second, without a specialized hardware module for the first layer and the last layer, most resources on FPGA can be utilized to deploy more unified hardware modules that can be used throughout the whole network. These unified hardware modules can work concurrently and improve the throughput of the system which we will discuss in Section 7.

The quantization of weights in the backbone and head is binarized as in [4], and only the signs of the weights are

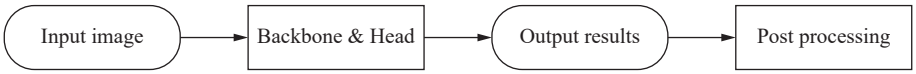


Fig. 1 A simplified neural network structure

retained in this paper. The advantage of binarization is omitting multiplications in convolution which greatly saves hardware resources. Meanwhile, a scaling factor α is shared through each layer to compensate for the loss of accuracy due to the binarization of the weights. The scaling factor α is the average of absolute weight values by solving an optimization problem, as stated in [4]. The binarized weights are then described as

$$W \approx \alpha B \tag{1}$$

where W and B are weights before and after binarization respectively. The scaling factor α is combined with the scaling factor in batch normalization and the combined factor is quantized to the power of 2, which can be realized in the hardware logic by shifters instead of multipliers. As a result, there is no multiplication in our hardware design.

The quantization of activations in the backbone and head as in [5] is

$$r_o = \frac{1}{2^j - 1} round((2^j - 1) r_i) \tag{2}$$

where j is the quantization bit-width; r_i and r_o are activations before and after quantization respectively, and they are both truncated to the range of $[0, 1]$. The activation bit-width we use is 4-bit.

4 Thermometer coding on input images

To be consistent with the bit-width of the activations in the middle layers, the 8-bit input images are converted to n-bit with thermometer coding. The input image is formatted in 3-channel RGB data, which are 8-bit in the range of $[0, 255]$. The maximum value that an n-bit number can represent is $2^n - 1$. A random number X in the range of $[0, 255]$ can be converted to several n-bit numbers as follows. The required number C of n-bit numbers is

$$C = ceil(255/(2^n - 1)). \tag{3}$$

In the C numbers, there are C_1 numbers that all of the n-bit are 1s. C_1 is expressed as

$$C_1 = floor(X/(2^n - 1)). \tag{4}$$

The C_1+1 number is

$$C_2 = X - C_1(2^n - 1). \tag{5}$$

The remaining $C - C_1 - 1$ numbers are all zeros, as shown in Fig. 2. In Fig. 2, n is 4 for example. In thermometer coding, the lower part is the first to be filled with 1s. If 4 bits are filled with 1s, it will be 15 then the lower part of the 4-bit thermometer coding will naturally be 15.

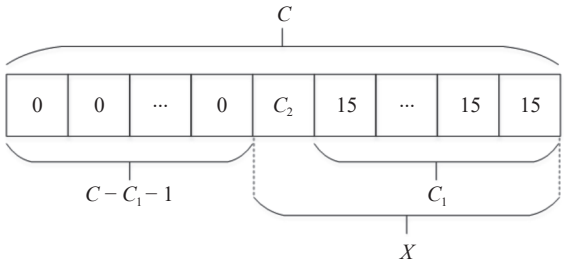


Fig. 2 A random X is depicted as C numbers with n-bit.

The conventional input layer in CNN is shown in Fig. 3 where the input images are 3-channel RGB data. We implement the 4-bit thermometer coding on input image. According to (3) every color channel of RGB is converted to 17 4-bit numbers. Therefore, the total number of input channels is 51. Because the input image is a 51-channel image and the rule for convolution is that the kernels have the same channel number as the activations, the corresponding convolutional kernels must also be expanded to 51 channels in the first layer. This is also keeping with how the middle layers are implemented. To be more specific, the kernel channel corresponding to a certain channel of RGB needs to be copied and expanded to 17 channels still in the order of RGB and each 17-channel is a group. The input layer after thermometer coding is shown in Fig. 4. Fig. 5 depicts one of the RGB channels after thermometer coding, and Fig. 6 is the channel-expanded convolutional kernel.

From the above analysis, although one channel is split into 17 channels, the weights in all these 17 channels are the same as one channel. According to (3)–(5), the random input X can be expressed as

$$C = C_1(2^n - 1) + C_2. \tag{6}$$

Then, the product of input and weight in one channel can be expressed as

$$\begin{aligned} X \times W = (C_1 \times (2^n - 1) + C_2) \times W = \\ (2^n - 1) \times W \times C_1 + C_2 \times W. \end{aligned} \tag{7}$$

The above expression further applies to the accumulation in 3-channel RGB data. As a result, the thermometer coding on the input image is an equivalent transformation and has no effect on the accuracy of the network.

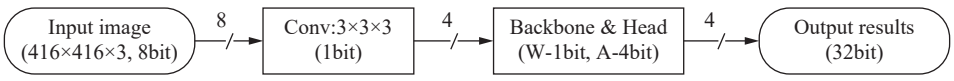


Fig. 3 The original network structure and bit-width

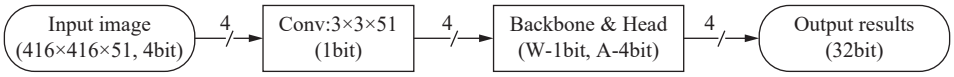


Fig. 4 The network structure and bit-width after thermometer coding

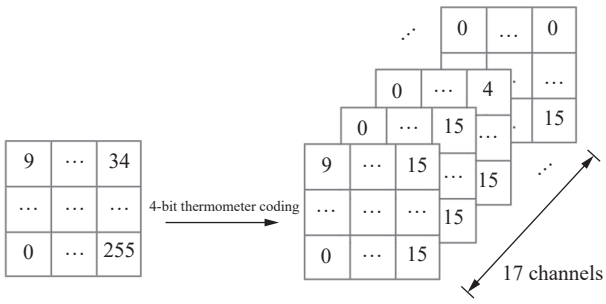


Fig. 5 The 4-bit thermometer coding on one input image channel

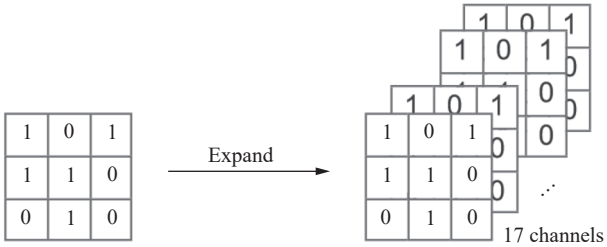


Fig. 6 Expand the kernel channel according to the corresponding input channel

5 Branch convolution quantization

Unlike the activations in middle layers, the output results of the last layer have implications that are to be modified on the object detection anchors. Fig. 7 shows the distribution of activations in a middle layer, and Fig. 8 shows the output results of the last layer in a full-precision network. From Figs.7 and 8, we can see that the activations in the middle layers are all positives after 4-bit quantization, which is acceptable since the missed negatives will be restored afterwards in the next layer with negative weights. However, the positives and negatives are nearly evenly distributed in the output results of the last layer. In addition, the output results have implications which will be used in the postprocessing module, and thus, the results have to be accurate. Now that the 4-bit activation cannot cover all of these values exactly, simply using the same quantization method described in (2) as the middle layers are not suitable for the output results. On the other hand, we want all the layers can be quantized with (2) such that all layers can be deployed on the same circuits. As a result, the only thing we can do is changing the structure of the last layer to meet our demands. Therefore, there is barely any algorithm where the output results of the network are quantized.

As described above, to maintain the consistency of quantization of every layer, the output results must be

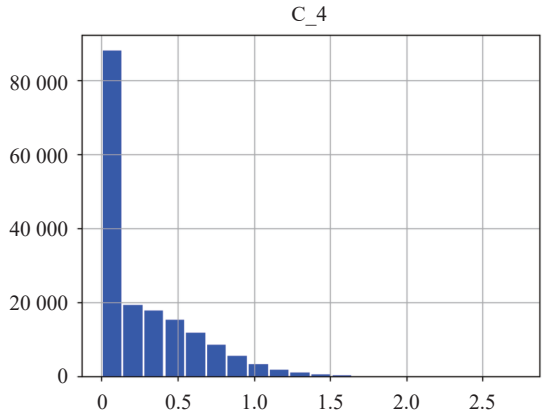


Fig. 7 The distribution of activations in the middle layers. The horizontal axis represents the values of the activations in a certain step and the vertical axis is the numbers of activations in the corresponding step range. The title "C_4" is a middle layer in the network and it can represent all the other middle layers in the network.

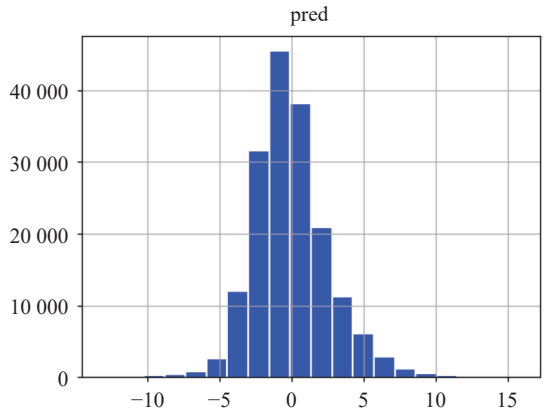


Fig. 8 The distribution of activations in the last layers. The title "pred" represents the last predict layer of the network.

quantized in the same way as the activations in the middle layers. Without sacrificing the accuracy, we consider adding extra branches along with the quantization of output results. The outputs of the extra branches are summed up with unbalanced weights. We name this the Branch Convolution Quantization (BCQ) as shown in Fig. 9. The initial thoughts of these unbalanced weights come from Fig. 8. In Fig. 8, we can see that there are both positives and negatives ranging from approximately −10 to +10. In addition, there must be decimals in these numbers which are used in the post-processing part afterwards. It is necessary that the summed up numbers in BCQ are as close to their true values as possible. Therefore, the four branches in BCQ represent positive integers, positive decimals, negative integers, and negative

decimals respectively. Because (2) is utilized to quantize the outputs of the four branches and the results of (2) range from 0 to 1, we need to multiply the convolutional results by 10 to the integer branches. However, multiplication of 10 is more expensive in the hardware implementation than 16, as 16 is 2^4 and can be realized by shifting 4 bits left with the shifters while multiplication of 10 should be realized by DSPs in FPGA. As a result, quantized results described as "Cur_Output Results" in Fig. 9 are multiplied by weighted numbers such as +16, +1, −16, −1 and then summed up to mimic the original 32-bit floating point results. The integer branches multiply 16 shifting to the higher 4 bits and the lower 4 bits on the right can be filled with the branches multiplied by 1. This is exactly a concatenation of the ×16 branch and the ×1 branch, which is the reason why there are two "Concat"s in Fig. 9. The concatenation is a large hardware resource reduction compared with the implementation of multiplication by 10. Because there are positive branches and negative branches in BCQ and the convolution results are positive when applying (2), subtraction naturally plays the role of negative branches in Fig. 9. To be clear and strict, the part to the right of the dotted line in Fig. 9 is complished in the post-processing procedure, as shown in Fig. 1.

In tiny_yolo_v3 as depicted in Figs. 10 and 9, the original network includes the backbone and head. The backbone extracts the features of the images, and the head returns the results of classification and regression of the features. Almost no existing algorithms quantize the output results as shown in Fig. 10. In this paper, we use the BCQ to replace the last layer with four convolution branches, the weights and outputs of which are quantized to 1-bit and 4-bit respectively. The BCQ layer needs to be retrained after all the weights and activations are quantized in the backbone and head. Afterwards, as shown in Fig. 9, the two concatenated branches will be 8-bit. If we directly quantize the 32-bit to 8-bit, we will lose too much information, as there are only 2^8 numbers in the convolutional results of the last layer no matter how many parameters there are in the weights of the last layer. 2^8 is far from the amount that the post-processing needs. In BCQ, there are two 8-bit numbers. In addition, there is an extra sign bit represented by positive branches and negative branches. All of these consist of the BCQ results which is a 10-bit value space. Without BCQ, there is only one branch and the coordinates of each bounding box can only be fine-tuned by the weights of one branch. However, with BCQ, the coordinates can be fine-tuned with 4 branches, while the 4 branches are independent of each other and have a wider value space as stated before. Therefore, the BCQ results are much higher than the quantization version but they are still worse than the full precision version in Tables 1–3 in Section 6. Although there are three more branches in BCQ, the data size is still small compared to the parameters of the entire network. The most important thing is that the output results are 4-bit, which means that all the branches can be deployed on the hardware where the middle layers are implemented on. There is a clear development of the bitwidth of the whole network in Figs. 3, 4, 9 and 10. All of the weights are 1-bit and all the activations are 4-bit till now. As a result, every layer of the network can be implemented on the same hardware logic as we expect.

6 Experimental results and analysis

After extremely low-bit quantization, thermometer coding and BCQ, the network structure, bitwidth of input and output, and the size of weights in every layer are listed in Table 4.

There is no need to list the experimental results of thermometer coding as it is an identity transform and has no effect on the accuracy of the network.

Applying BCQ on tiny_yolo_v3 and yolo_v3, the results on VOC and COCO datasets are listed in Tables 1–3 respectively. As we can see, compared with normal quantization, the accuracy of this work has been greatly improved. There is almost the same level of reduction in accuracy compared with the full precision model of each category in this work. The accuracy loss trends of normal quantization and this work are almost the same. Although the accuracy of BCQ applied on tiny_yolo_v3 drops on both datasets compared to the full precision network, the advantages of BCQ are apparent on yolo_v3. Since the structure of tiny_yolo_v3 is very simple and has

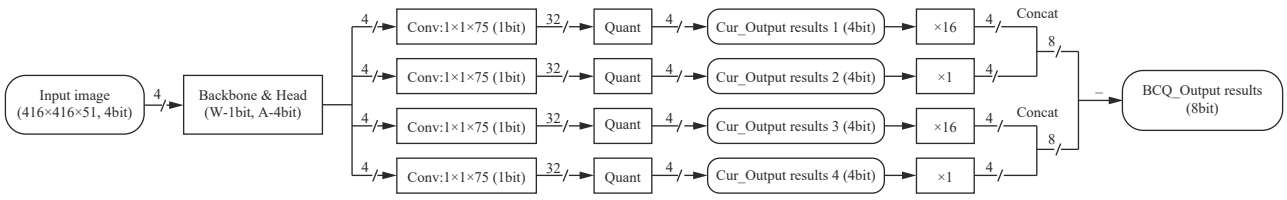


Fig. 9 The network structure with BCQ

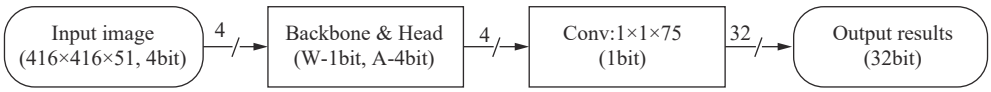


Fig. 10 The network structure without BCQ

Table 1 Accuracy on VOC

Model type	Full precision*	Quantization**	This work***
tiny_yolo_v3	65.2	41.12	**50.11**
yolo_v3	81.4	67.46	**73.32**

*Full precision means that there is no quantization in the entire network.
**Quantization means that the weights are binarized and the activations are 4-bit in backbone and head, but the output results are 32-bit floating point.
***This work means implementing BCQ on the output results of quantization.

a limited number of parameters as shown in Table 5, the loss of accuracy is significant after violent compression. While yolo_v3 has a more robust backbone with more layers, the accuracy of yolo_v3 with BCQ is very impressive. It is expected to reach a better result on deeper network quantization[12].

7 Hardware implementation

To verify the advantages of implementing low-bit quantization network on hardware, we deploy tiny_yolo_v3 on the Virtex UltraScale+ series FPGA VU9P from Xilinx, and the designed block diagram is shown in Fig. 11. One neural computing array (NCA) consists of 128 neural computing units (NCU), which work in parallel to improve the throughput and speed of computing. Every NCU works in the same mode and is responsible for one convolution kernel. The input feature maps are read from the on-chip memory and the weights and batch normalization parameters are also handed out to the 128 NCUs from the on-chip memory. The weights are reused for the entire input feature maps, and the results of every NCU, which will be the input feature maps of the next layer, are written back to the on-chip memory with the control of the read-write state machine. In summary, all data in the network are accessed on-chip in our hardware design, which greatly reduces the cost of data transferring.

The FPGA resource consumption of NCA is shown in Fig. 12. The LUTs and FFs are mainly used for intermediate values and weights cache in convolution computing. The BRAMs are applied for the on-chip storage of weights, batch normalization parameters, input and output feature maps. It is worth noting that there is not any DSP in our implementation. This is because the weights in convolution are binarized where the multiplications are replaced by additions. In addition, we also simplify the batch normalization operations and retrain the network. To implement BN on hardware logic, the biases are rounded to integers and the scale factors approximate the nearest power of two which can be easily realized in hardware logic with shifters.

Compared to the deployment of Tiny YOLO-v2 and Sim-YOLO-v2 on hardware in [12], the resource consumption, throughput, efficiency, power efficiency and so on

Table 2 Accuracy of each category on VOC in different tiny_yolo_v3 network

Category	Full precision*	Quantization**	This work***
Aeroplane	74.65	55.76	**62.45**
Bicycle	75.58	47.89	**66.13**
Bird	59.92	23.52	**38.51**
Boat	52.48	24.69	**38.00**
Bottle	38.36	12.85	**20.98**
Bus	73.07	56.81	**60.34**
Car	78.97	58.88	**68.88**
Cat	73.20	55.82	**58.26**
Chair	43.37	20.47	**27.46**
Cow	69.24	38.10	**53.83**
Diningtable	60.96	47.44	**46.96**
Dog	68.38	51.97	**54.87**
Horse	77.48	63.09	**70.57**
Motorbike	74.75	57.86	**66.77**
Person	72.58	52.48	**62.08**
Pottedplant	37.92	16.50	**21.12**
Sheep	66.65	28.00	**49.90**
Sofa	64.03	48.95	**41.24**
Train	78.93	50.74	**67.95**
Tvmonitor	64.70	35.97	**49.80**

*Full precision means that there is no quantization in the entire network.
**Quantization means that the weights are binarized and the activations are 4-bit in backbone and head, but the output results are 32-bit floating point.
***This work means implementing BCQ on the output results of quantization.

Table 3 Accuracy on COCO

	Full precision*		Quantization**		This work***	
Model type	Ap50-95	Ap50	Ap50-95	Ap50	Ap50-95	Ap50
Tiny_yolo_v3	16.0	33.8	2.89	9.46	**6.66**	**17.48**
Yolo_v3	36.0	57.6	8.56	27.87	**25.49**	**47.92**

*Full precision means that there is no quantization in the entire network.
**Quantization means that the weights are binarized and the activations are 4-bit in backbone and head, but the output results are 32-bit floating point.
***This work means implementing BCQ on the output results of quantization.

Table 4 Extremely low-bit network structure of tiny_yolo_v3

Layer	Data width of input	Data width of output	Data width of weights	Size of kernel
Backbone.layer1 (First layer)	4	4	1	3, 3, 51, 16
Backbone.layer2	4	4	1	3, 3, 16, 32
Backbone.layer3	4	4	1	3, 3, 32, 64
Backbone.layer4	4	4	1	3, 3, 64, 128
Backbone.layer5	4	4	1	3, 3, 128, 256
Backbone.layer6	4	4	1	3, 3, 256, 512
Backbone.layer7	4	4	1	3, 3, 512, 1 024
Head.layer1	4	4	1	3, 3, 1 024, 256
Head.layer2	4	4	1	1, 1, 256, 128
Head.layer3	4	4	1	3, 3, 256, 512
Head.pred2 (Last layer)	4	4	1	(1, 1, 512, 75) × 4
Head.pred4	4	4	1	3, 3, 384, 256
Head.pred1 (Last layer)	4	4	1	(1, 1, 256, 75) × 4

Table 5 Model size

Model type	Data width of input	Data width of output	Data width of weights	Backbone.layer1 (First layer)
Model size	44 MB	235 MB	1.4 MB	7.7 MB

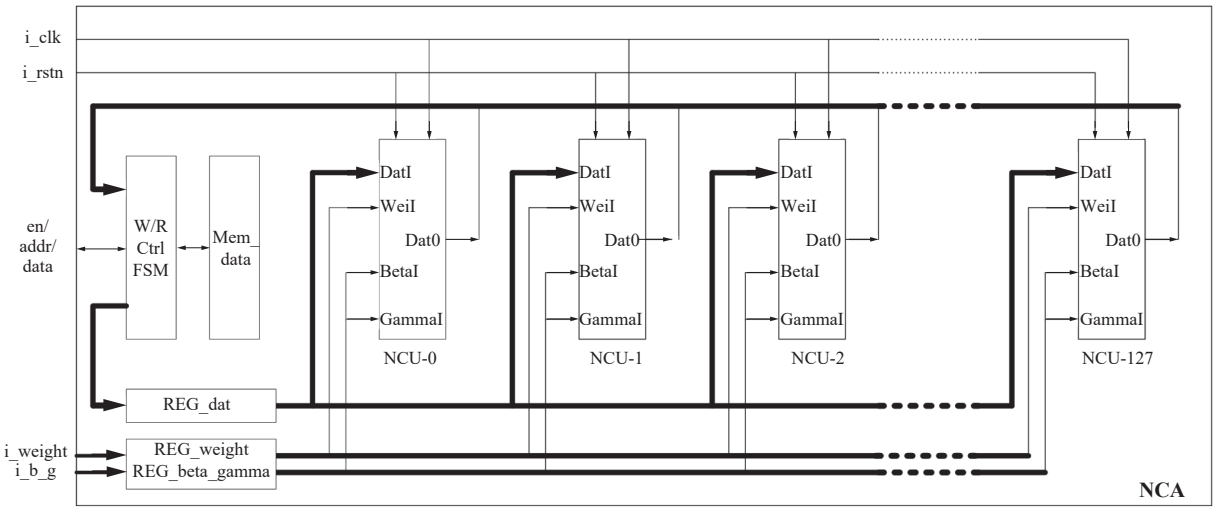


Fig. 11 The block diagram of NCA

are listed in Table 6. The results in Table 6 are not composed of the post-processing procedure, so as to the Tiny YOLO-v2 and Sim-YOLO-v2 in [12], so we put the results together to make a comparison. The DSPs in [12] are used for batch normalization and the last layer. In our work, there is not any DSP since the last layer with BCQ can share the same hardware logic with middle layers. In addition, with BCQ, there is no need to design a separate module to implement the first and last layers and more unified parallel computing modules designed for all layers can be implemented on FPGA. This is why the LUTs and FFs consumption which is the most commonly used resource in hardware implementation in our work is much more than that in [12] according to Table 6 and Fig. 12. The evidence that there are more parallel computing modules in our work than that in [13] is the throughput in Table 6. There is not any multiplication in our design, and only additions and shifters are used. Moreover, there is no high bit-width in the last layer. All of these factors lead to the higher efficiency (GOPS/kLUT) of our work than that of [12].

To conclude, by compressing the network weights to the limit, and proposing the quantization method of the input and output layers, the network quantization consistency is maintained, and the network is more efficient, simplified and universal in hardware deployment.

Name	CLB LUTs (1182240)	CLB Registers (2364480)	CARRY8 (147780)	F7 Muxes (591120)	F8 Muxes (295560)	Block RAM Tile (2160)	DSPs (6840)
u_nca0 (nca__xdcDup__1)	468690	282714	3882	728	56	512	0
ncgx8[0].u_ncg (ncg_136)	101109	36371	480	125	0	0	0
ncgx8[1].u_ncg (ncg_137)	113971	36080	480	48	0	0	0
ncgx8[2].u_ncg (ncg_138)	38215	34044	480	31	0	0	0
ncgx8[3].u_ncg (ncg_139)	47361	34480	480	23	0	0	0
ncgx8[4].u_ncg (ncg_140)	38209	34043	480	23	0	0	0
ncgx8[5].u_ncg (ncg_141)	38156	34032	480	32	0	0	0
ncgx8[6].u_ncg (ncg_142)	38130	34032	480	16	0	0	0
ncgx8[7].u_ncg (ncg_143)	38154	34032	480	25	0	0	0
u_arb (arb_nca_144)	4595	62	0	98	24	0	0
u_mema (mem_nca__xdcDup__1)	1331	32	0	63	0	256	0
u_memb (mem_nca__xdcDup__2)	2600	32	0	128	32	256	0
u_rctl (rctl_145)	879	4893	8	1	0	0	0
u_wctl (wctl_146)	5980	581	34	115	0	0	0

Fig. 12 The resource consumption of NCA. The left side of the list is the modules in NCA. In the implementation of NCA, there are 8 NCGs and every NCG consists of 16 NCUs. Arb is the arbiter. Mema and memb are the ping-pong memory. Rctl and wctl are the read and write controller respectively. The top of the list is the logical resources consumed by each module.

Table 6 Overall comparison of our work to others

	Tiny YOLO-v2[13]	Sim-YOLO-v2[13]	Data width of input
Platform	Virtex-7 VC707	Virtex-7 VC707	XCVU9P
Frequency	300 MHz	200 MHz	80 MHz
BRAMs	1 026	1 144	512
DSPs	168	272	**0**
LUTs-FFs	86 K–60 K	155 K–115 K	468 K–282 K
CNN size (GOP)	6.97	17.17	8.69
Precision (W,A)	(1, 6)	(1, 6)	**(1, 4)**
First layer (W,A)	(1, 8)	(1, 8)	**(1, 4)**
Last layer (W,A)	(8, 16)	(8, 16)	**(1, 4)**
Image size	416×416	416×416	416×416
Frame rate	66.56	109.3	60
Accuracy (mAP)	51.38	64.16	50.11
Throughput (GOPS)	464.7	1877	**5 898***
Data width of output	5.40	12.11	**12.68**
Power (W)	8.7	18.29	**10.8**
Data width of weights	53.29	102.62	**546.11**

*5 898GOPS: Our unified computing module requires 4 clock beats from input to output. In 80MHz clock, the frequency of output results is 20MHz. In NCU, one output needs 3×3×128 additions and multiplications. As a result, the throughput of one NCU in 20MHz will be 3×3×128×2×20MHz=46.08 GOPS, and the throughput of 128 NCUs (1 NCA) will be 46.08×128=5 898.24 GOPS.

8 Conclusion and future work

In this paper, we perform extremely low-bit quantization to the weights of all layers, thermometer coding to input images, and BCQ to output results and accomplish the consistent quantization of all layers of the object detection network. When deployed on specific hardware, all layers in the network can share the same logic, which simplifies the hardware design and improves the energy efficiency as more computing modules can be implemented. Moreover, the BCQ improves the accuracy of quantization of object detection networks. Future work can focus on the flexible selection of the number and weights of branches in BCQ, as well as the structure of hardware design. This work provides a direction for the research of the fully quantized network algorithm and powerful support for the edge-deployment of network.

Declarations of conflict of interest

The authors declared that they have no conflicts of interest to this work.

References

[1] M. Courbariaux, Y. Bengio, J. P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, Montreal, Canada, pp. 3123–3131, 2015. DOI: 10.5555/2969442.2969588.

[2] M. Kim, P. Smaragdis. Bitwise neural networks, [Online], Available: https://arxiv.org/abs/1601.06071, 2016.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Barcelona, Spain, pp. 4114–4122, 2016. DOI: 10.5555/3157382.3157557.

[4] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the 14th European Conference on Computer Vision*, Springer, Amsterdam, The Netherlands, pp. 525–542, 2016. DOI: 10.1007/978-3-319-46493-0_32.

[5] S. C. Zhou, Y. X. Wu, Z. K. Ni, X. Y. Zhou, H. Wen, Y. H. Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients, [Online], Available: https://arxiv.org/abs/1606.06160, 2016.

[6] H. T. Qin, R. H. Gong, X. L. Liu, X. Bai, J. K. Song, N. Sebe. Binary neural networks: A survey. *Pattern Recognition*, vol. 105, Article number 107281, 2020. DOI: 10.1016/j.patcog.2020.107281.

[7] J. W. Yang, X. Shen, J. Xing, X. M. Tian, H. Q. Li, B. Deng, J. Q. Huang, X. S. Hua. Quantization networks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Long Beach, USA, pp. 7300–7308, 2019. DOI: 10.1109/CVPR.2019.00748.

[8] R. D. Li, Y. Wang, F. Liang, H. W. Qin, J. J. Yan, R. Fan. Fully quantized network for object detection. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Long Beach, USA, pp. 2805–2814, 2019. DOI: 10.1109/CVPR.2019.00292.

[9] S. Kim, H. Kim. Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors. *IEEE Access*, vol. 9, pp. 20828–20839, 2021. DOI: 10.1109/ACCESS.2021.3054879.

[10] H. X. Fan, S. L. Liu, M. Ferianc, H. C. Ng, Z. Q. Que, S. Liu, X. Y. Niu, W. Luk. A real-time object detection accelerator with compressed SSDLite on FPGA. In *Proceedings of International Conference on Field-Programmable Technology*, IEEE, Naha, Japan, pp. 14–21, 2018. DOI: 10.1109/FPT.2018.00014.

[11] J. I. Guo, C. C. Tsai, J. L. Zeng, S. W. Peng, E. C. Chang. Hybrid fixed-point/binary deep neural network design methodology for low-power object detection. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 388–400, 2020. DOI: 10.1109/JETCAS.2020.3015753.

[12] D. T. Nguyen, T. N. Nguyen, H. Kim, H. J. Lee. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019. DOI: 10.1109/TVLSI.2019.2905242.

[13] A. J. Zhou, A. B. Yao, Y. W. Guo, L. Xu, Y. R. Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, 2017.

[14] Y. Sakuma, H. Sumihiro, J. Nishikawa, T. Nakamura, R. Ikegaya. n-hot: Efficient bit-level sparsity for powers-of-two neural network quantization, [Online], Available: https://arxiv.org/abs/2103.11704, 2021.

[15] F. Cardinaux, S. Uhlich, K. Yoshiyama, J. A. García, L. Mauch, S. Tiedemann, T. Kemp, A. Nakamura. Iteratively training look-up tables for network quantization. *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 860–870, 2020. DOI: 10.1109/JSTSP.2020.3005030.

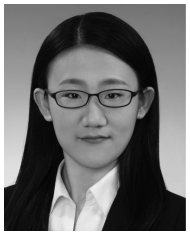
[16] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, J. Hassoun. Post-training piecewise linear quantization for deep neural networks, [Online], Available: https://arxiv.org/abs/2002.00104, 2020.

[17] E. Park, J. Ahn, S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, Honolulu, USA, pp. 7197–7205, 2017. DOI: 10.1109/CVPR.2017.761.

[18] M. Shimoda, S. Sato, H. Nakahara. All binarized convolutional neural network and its implementation on an FPGA. In *Proceedings of International Conference on Field Programmable Technology*, IEEE, Melbourne, Australia, pp. 291–294, 2017. DOI: 10.1109/FPT.2017.8280163.

[19] P. Guo, H. Ma, R. Z. Chen, P. Li, S. L. Xie, D. L. Wang. FBNA: A fully binarized neural network accelerator. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications*, IEEE, Dublin, Ireland, pp. 51–513, 2018. DOI: 10.1109/FPL.2018.00016.

[20] S. L. Zhu, X. Dong, H. Su. Binary ensemble neural network: More bits per network or more networks per bit? In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Long Beach, USA, pp. 4918–4927, 2019. DOI: 10.1109/CVPR.2019.00506.



Miao Li received the B.Eng. and M.Eng. degrees in mechanical and electrical engineering from Beijing Institute of Technology, China in 2014 and 2017, respectively. She is an engineer in the National Engineering & Technology Research Center for Application Specific Integrated Circuit Design (NECFAD), Institute of Automation, Chinese Academy of Sciences, China. She has published two patents in AI.

Her research interests include computer vision, IC design and optimization.

E-mail: miao.li@ia.ac.cn

ORCID iD: 0009-0001-4741-3043

Feng Zhang received the B.Eng. and M.Eng. degrees in mechanical and electrical engineering from Beijing Institute of Technology, China in 1998 and 2003, respectively, and the Ph.D. degree in electronic and information from University of Science and Technology of China, China in 2020. Currently, he is a professor in the National Engineering & Technology Research Center for Application Specific Integrated Circuit Design (NECFAD), Institute of Automation, Chinese Academy of Sciences, China. He has published more than ten papers and tens of patents in these areas. As a project leader, he has completed a number of scientific research projects.

His research interests include micro-electronics, computer vision, IC design and optimization.

E-mail: zhangfeng@ia.ac.cn (Corresponding author)

ORCID iD: 0000-0003-2586-5505

Cuiting Zhang received the B.Eng. and M.Eng. degrees in engineering management from Hebei University of Technology, and computer technology from Capital Normal University in Beijing, China in 2017 and 2020, respectively. She is an engineer in the National Engineering & Technology Research Center for Application Specific Integrated Circuit Design (NECFAD), Institute of Automation, Chinese Academy of Sciences, China.

Her research interests include computer vision, IC design and optimazation.

E-mail: cuiting.zhang@ia.ac.cn