Efficient field-programmable gate array-based reconfigurable accelerator for deep convolution neural network

Xiaohong Hu,1,2 Taosheng Chen,1 Hongmin Huang,1 Zihao Liu,1 Xueming Li,1 and Xiaoming Xiong1,2,✉
1School of Automation, Guangdong University of Technology, Guangzhou, 510006, China
2Company of Chipeye Microelectronics Foshan Ltd., Foshan, 528225, China
Correspondence
X. Xiong, School of Automation, Guangdong University of Technology, Guangzhou, 510006, China.
Email: xmxiong@gdut.edu.cn

Deep convolutional neural networks (DCNNs) have been widely applied in various modern artificial intelligence (AI) applications. DCNN's inference is a process with high calculation costs, which usually requires billions of multiply-accumulate operations. On mobile platforms such as embedded systems or robotics, an efficient implementation of DCNNs is significant. However, most previous field-programmable gate array-based works on accelerators for DCNNs just support one DCNN or just support convolution layers. In order to address this limitation, this work proposes a reconfigurable accelerator. The accelerator is flexible and can support multiple DCNNs and different layer types, such as convolution, pooling, activation function, and full connection layers. It is equipped with a five-level pipeline convolution engine whose main component is two processing element arrays. Furthermore, a design space exploration method is proposed to make full advantage of the proposed accelerator. This accelerator is implemented with the ZYNQ-7 ZC706 evaluation board and achieves a high performance of 53.29 Giga operations per second (GOPS) on AlexNet and 45.09 GOPS on YOLOv2-tiny at 100 MHz. Further performance of the accelerator is compared with the previous works, and it achieves multiple advantages: High performance, high configurability, and efficient resource utilisation.

Introduction: Deep convolutional neural networks (DCNNs) have become one of the most popular approaches to many robotics’ visual processing tasks, such as object detection, image classification, and scene understanding. Recently, DCNNs are usually implemented on platforms including graphical processing unit (GPU), field-programmable gate array (FPGA) [1], and application-specific integrated circuit (ASIC) [2]. GPU has high power consumption, and ASIC requires significant fabrication cost. FPGA has advantages of high configurability, low power, and reasonable price, which make FPGAs attractive for DCNN implementation on robotic. Besides DCNNs, the neural networks using a non-iterative training mechanism [3, 4] can be accelerated by FPGA.

Numerous FPGA-based works are proposed for accelerating DCNNs. However, most works just support convolution layers and do not support pooling, activation function, and full connection layers, such as the designs in [1, 5–7]. Furthermore, among these works, the designs in [1, 5, 6] only support one DCNN. The design in [5] supports AlexNet and achieves a performance of 38.4 GOPS. The design in [6] supports YOLOv2-tiny and achieves a low performance of 21.6 GOPS. NullHop in [8] can achieve different layers’ operations. However, it supports limited kernel sizes (1 × 1, 3 × 3, 5 × 5, 7 × 7) and one kernel stride (S = 1) and has a low performance of 17.196 GOPS on FPGA platform. There are some exceptions, such as the designs in [9, 10] supporting multiple whole DCNNs.

With the ongoing advancements in visual processing on robotics, a wide variety of DCNNs appear. Therefore, designing a flexible accelerator supporting different DCNNs is all that matters. This work highlights the applicability and configurability of the accelerator for DCNNs, as well as performance. A five-level pipeline convolution engine (ConvEngine) and configurable processing element (PE) are designed for supporting different DCNNs and layer types. For the sake of better performance, a design space exploration method is proposed to search for the optimal design corner. In order to show the effectiveness of the proposed accelerator, it is compared with the emerging FPGA-based accelerators [5–8]. From the experiment results, there is no doubt that our accelerator is better than other state-of-the-art works in the metric of configurability and performance.

Fundamental theory: Figure 1 shows the YOLOv2-tiny, which consists of convolution, pooling, activation function, batch normalisation layers.

1. Convolution layer and the full connection layer

Convolution is used to extract different features in the image. It involves billions of multiplications and additions between the filters and input feature maps. The operations can be described below:

\[ Y = \sum_{i=1}^{n} (X^i \times W^i) + b \]  

It is notable that the full connection layer can also be illustrated by formula (1).

2. Batch normalisation layer

Batch normalisation is often followed by the convolution layer to provide any layer with inputs that are zero mean or unit variance. It is illustrated as follows:

\[ Y = \frac{X - \mu}{\sigma + \xi} \times \sqrt{\sigma^2 + \xi} + \frac{-\mu}{\sqrt{\sigma^2 + \xi}} = aX + b \]  

where \( \sigma^2 \) stands for variance, \( \xi \) is constant. It can be implemented with multiplication and addition.

3. Activation function layer

Activation function is used to transform the input before the pooling layer. On hardware, the sigmoid activation function is often implemented by a piecewise linear function, which can be realised with multiplication and addition. Rectified linear unit (ReLU) and LeakyReLU activation functions are also be realised with multiplication and addition. They are illustrated as follows:

\[ Y = AX + B \]  

4. Pooling layer

Pooling is a form of a dimensional reduction in DCNN by the way of throwing away redundant information so that the critical information can be preserved. Max and mean poolings are given as follows:

\[ \text{max pooling}: Y = \max (X^1, X^2, X^3, X^4) \]  

\[ \text{mean pooling}: Y = (X^1 + X^2 + X^3 + X^4) / 4 \]

where the max pooling can be completed by subtraction and comparison with the sign bit and mean pooling by addition and shifting.

Reconfigurable accelerator: As described above, all layers can be implemented by the combination of multiplications and additions. We can design a configurable PE to realise these computing functions. In this work, we propose a reconfigurable accelerator based on customised PEs for DCNNs as shown in Figure 2. The whole system consists of acorn RISC machine (ARM), instruction memory (Mem), controller, ConvEngine, and on-chip memory. On-chip memory contains input, partial, output, and weight buffers. ConvEngine contains two 14 × 14 PE arrays. The size of PE arrays is modified to 14 × 14 because we think that 14 × 14 is more suitable for most modern DCNN networks, like AlexNet, VGG, and ResNet. The high and weight of feature maps in modern DCNN are...
mostly a multiple of 14, and $14 \times 14$ tiling feature maps will fit for the $14 \times 14$ PE array, leaving few PEs idle. The two CEs handle convolution operations in the same input channels but different output channels, using different weights. For the purpose of efficient data transmission between off-chip dynamic random access memory (DRAM) and on-chip buffer, memory interface generator is chosen, offering a bandwidth of 5.4 GB/s at 100 MHz in our testing experiment.

The ARM transfers instructions or configurations to instruction Mem module, and then controller module reads instructions from instruction Mem and starts transferring data from DRAM and executing convolution.

ConvEngine: ConvEngine is a five-level structure as shown in Figure 3. The first level is two input register arrays (IREG), exchange pixels in ping-pong mode. The second level is the input share register array (ISREG), which broadcasts pixels to improve PE utilisation for different kernel strides. The third level contains a weight register array (WREG) and a partial sum register array (PREG). ISREG and WREG are used to improve PE utilisation and to reduce critical path delay. The fourth level is PE array, which efficiently supports different computing of various layer types. The fifth level is the output register array REG (OREG). ConvEngine contains two CEs and each CE has individual WREG, PREG, PE array, and OREG. Two CEs share the same IREG and ISREG. Two CEs use the same input, along with different weights and partial sums (psum) of different channels, after which they generate outputs of different channels. In this way, the hardware resources for IREG and ISREG will be reduced.

Processing element: As described earlier, convolution, full connection, batch normalisation, activation function, and pooling can be realised with multiplications and/or additions. Therefore, a flexible PE is proposed in this work and is shown in Figures 4–6. It contains a multiplier, an adder, a register, and several multiplexers. The multiplier and adder are reused in almost every operation, except for pooling and ReLU, so the utilisation of PE in different operations can be improved as much as possible. Figures 4–6 show the configured data paths of major operations, where red and pink lines are data paths. Since the convolution layer adopts a tiling strategy, convolution operations have two kinds of data paths, as the psum may be stored in psum buffer between different tiling convolution (Figure 4(a)), or the psum may be stored in PE’s register during the same tiling convolution (Figure 4(b)). Since full connection layers do not use a tiling strategy, their operations have only one data path (Figure 4(b)).

The data paths of mean and max poolings are shown in Figures 5(a) and (b). Figure 6(a) depicts the data path of sigmoid function, and Figure 6(b) describes the data path of batch normalisation or LeakyReLU.

Design space exploration: The design space exploration is used for finding the optimal tiling parameter combination \{Tr, Tc, Tm, Tn\}. For a reconfigurable architecture, different tiling parameter combinations will result in various performances, and these performances can vary considerably. Thus, finding the optimal tiling parameter is significantly important. As demonstrated, the tiling strategy adopted in this work is shown in Figure 7.
We modify the roofline model to fit our accelerator, illustrated as follows:

\[
\text{Attainable Perf.} = \min \left\{ \frac{\text{Peak Perf.}}{\text{OptInt} \times BW} \right\}
\]

\[
\text{Peak Perf.} = \frac{2 \times R \times C \times M \times N \times K \times K}{N_{\text{max}} \times T_{\text{m}} \times T_{\text{r}} \times T_{\text{n}} \times T_{\text{p}}}
\]

\[
\text{OptInt} = \frac{2 \times R \times C \times M \times N \times K \times K}{N_{\text{max}} \times T_{\text{m}} \times T_{\text{r}} \times T_{\text{n}} \times T_{\text{p}}}
\]

\[
T_{\text{p}} = \left( T_{\text{i}} + K^2 \times T_{\text{n}} \right) \times T_{\text{m}}
\]

where Peak Perf. stands for the highest performance supported by PEs, and OptInt \( \times BW \) represents the highest performance supported by off-chip memory bandwidth; OptInt is operational intensity. The maximum attainable performance depends on the smaller of the above two variables.

### Table 1. Implementation results of our design

| Design benchmark | Platform      | Freq (MHz) | Perf (GOPS) | DSP | Power (W) | Perf/DSP | Power/DSP |
|------------------|---------------|------------|-------------|-----|-----------|----------|-----------|
| YoloV2-tiny      | Zynq 7045    | 100        | 45.09       | 392 | 6.92      | 0.115    | 0.136     |
| AlexNet          | Zynq 7100    | 122        | 17.20       | 128 | 8.18      | 0.136    | 0.145     |
| VGG16            | Cyclone V    | 21.6       | 12.60       | 175 | 8.70      | 0.145    | 0.145     |
| ResNet-34        | Zynq 7045    | 38.4       | 38.4        | 391 | 5.11      | 0.115    | 0.136     |
| Ours             | Zynq 7045    | 38.4       | 57.02       | 392 | 6.92      | 0.115    | 0.136     |

### Table 2. Comparison with other designs

| Design          | Platform      | Perf (GOPS) | DSP | Perf/DSP | Config |
|-----------------|---------------|-------------|-----|----------|--------|
| [5]             | Zynq 7045    | 38.4        | 391 | 0.098    | Low    |
| [8]             | Zynq 7100    | 17.20       | 128 | 0.134    | Low    |
| [6]             | Cyclone V    | 21.6        | 122 | 0.177    | Low    |
| [7]             | Zynq 7045    | 77.51       | 392 | 0.198    | Low    |
| Ours            | Zynq 7045    | 57.02       | 392 | 0.145    | High   |

We conclude that our accelerator is highly configurable and performs well in comparison with other designs. The low bit-width operation or shift operation can greatly reduce the complexity of multiplication; thus, these designs have higher performance than ours.

### References

1. Guo, J., et al.: Bit-width adaptive accelerator design for convolution neural network. In: IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, pp. 1–5 (2018)
2. Yin, S., et al.: An energy-efficient reconfigurable processor for binary- and ternary-weight neural networks with flexible data bit width. IEEE J. Solid-State Circuits 54(4), 1120–1136 (2019)
3. Cao, W., et al.: A review on neural networks with random weights. Neurocomputing 275, 278–287 (2018)
4. Wang, X., Cao, W.: Non-iterative approaches in training feed-forward neural networks and their applications. Soft Comput. 22, 2473–2486 (2018)
5. Moini, S., et al.: A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications. IEEE Trans. Circuits Syst. II Express Briefs, 64(10), 1217–1221 (2017)
6. Wai, Y.J., et al.: Fixed point implementation of Tiny-Yolo-v2 using OpenCL on FPGA. Int. J. Adv. Comput. Sci. Appl. 9(10), (2018)
7. Hu, X., et al.: A Resources-efficient configurable accelerator for deep convolutional neural networks. IEEE Access 7, 72113–72124 (2019)
8. Aimar, A., et al.: NullHop: a flexible convolutional neural network accelerator based on sparse representations of feature maps. IEEE Trans. Neural Networks Learn. Syst. 30(3), 1–13 (2018)
9. Luo, C., et al.: Towards efficient deep neural network training by FPGA-based batch-level parallelism. In: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, pp. 45–52 (2019)
10. Kaarmann, S.P., et al.: FPGA based deep learning models for object detection and recognition comparison of object detection comparison of object detection models using FPGA. In: 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, pp. 471–474 (2020)
11. Tu, F., et al.: Reconfigurable architecture for neural approximation in multimedia computing. IEEE Trans. Circuits Syst. Video Technol. 29(3), 902–906 (2019)
12. Fong, C.F.B., et al.: A cost-effective CNN accelerator design with configurable PU on FPGA. In: 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, pp. 31–36 (2019)