Songklanakarin J. Sci. Technol.
43 (6), 1831-1839, Nov. - Dec. 2021

http://www.sjst.psu.ac.th

Original Article

Motorcycle detection based on deep learning implemented on FPGA♣

Feng Peng[1], Kittikhun Thongpull[1], Masami Ikura[2], and Nattha Jindapetch[1*]

[1] *Department of Electrical Engineering, Faculty of Engineering, Prince of Songkla University, Hat Yai, Songkhla, 90112 Thailand*

[2] *Toyota Tsusho Nexty Electronics (Thailand) Co., Ltd, Pathum Wan, Bangkok, 10330 Thailand*

Received: 25 December 2020; Revised: 12 December 2021; Accepted: 15 December 2021

Abstract

This paper proposes a hardware accelerator design for motorcycle detection based on deep learning. We designed the training parameters by K-means algorithm and created the motorcycle dataset from Thailand's urban scene. Due to the rapid evolution of deep learning and the need for high-performance, low-power, and scalable models for application platforms, we designed the YOLOv2 accelerator architecture on the PYNQ platforms by using five optimization methods, including loop unrolling/pipeline, loop tiling, data quantization, memory ping-pong, and multi-channel data transmission. The proposed training parameters can increase the accuracy from the original 76.8% to 89.45%. The hardware experimental results obtained 14.10 GOP/s (100MHz) and 25.98 GOP/s (150MHz) on the PYNQ (ZYNQ 7020). The performance of the acceleration platform that we designed is 6.32 times faster than that of the CPU (i7), and the energy consumption is 1/26 of the CPU. In addition, the hardware accelerated deep learning applications have in recent years improved a lot in accuracy and calculation speed.

Keywords: motorcycle detection, deep learning, YOLOv2, FPGA, high-level synthesis

1. Introduction

In developing countries, motorcycles have become the primary means of transportation. Because of their ability to reach a high speed and few protective measures, they have always been the vehicle type with the highest mortality rate in traffic accidents. According to the traffic environment characteristics, in different urban areas the drivers of cars will pay extra attention to various objects. In the complex traffic environment of developing countries, autonomous driving should pay more attention to high-speed motorcycles. These reasons raise the requirements to demonstrate high-efficiency detection of motorcycles on a low-power-consumption platform. It is urgent to develop accurate and fast detection platforms to achieve efficient detection results for motorcycles (Lee, Pino, Lin, & Peters, 2013; Siebert & Lin, 2020).

There are many detection algorithms based on deep neural networks (Lecun, Bengio, & Hinton, 2015). While two-stage approaches such as fast RCNN (Girshick, 2015), achieve higher accuracy than single-stage approaches, they are very time-consuming. In contrast, single-stage approaches simultaneously conduct object location and identification. The single-stage approaches like YOLO (Redmon, Divvala, Girshick, & Farhadi, 2016) and RetinaNet (Lin, Goyal, Girshick, He, & Dollar, 2020) are much faster than the two-stage approaches. YOLOv2 was introduced to detect objects more quickly and accurately (Redmon & Farhadi, 2017). Based on the characteristics of the application and the complexity of CNN, it is difficult for the CPU to provide sufficient computing power. The relatively powerful GPU has a larger power consumption and is not suitable for removable devices (Gschwend, 2016). To improve computing performance and save energy consumption, FPGA-based acceleration has become one of the most attractive alternatives (Guo, Zeng, Yu, Wang, & Yang, 2017). FPGA is the next possible solution to surpass GPU in speed and energy efficiency (Blaiech, Khalifa, Valderrama, Fernandes, & Bedoui, 2019). It is a programmable device that can be

♣Peer-reviewed paper selected from The 9th International Conference on Engineering and Technology (ICET-2021)
*Corresponding author
Email address: nattha.s@psu.ac.th

configured as a custom circuit to perform a specific task. In (Zhao *et al.*, 2019), the characteristics of low power consumption, strong computing capability, and high flexibility, were used to design an underwater object detection platform, because when designed for normal circumstances, the high-power platforms cannot support underwater operations. The paper (Ye, Hong, Chen, Hsiao, & Fu, 2020) proposed a two-stage YOLOv2-based network to tackle distorted road marking detection as well as to balance precision and recall. But the model worked on GTX1070, making it challenging to deploy in Autonomous Driving Systems (ADS). Although the efficiency of reconfigurable logic platforms is lower than of custom circuits (ASICs) (Fang, Mulder, Hidders, Lee, & Hofstee, 2020), their design cycle is shorter than of ASICs.

Compared to deploying deep neural networks on CPUs and GPUs, the FPGA deployment methods focuses on the allocation and invocation of underlying hardware resources. Therefore, it is much more challenging to implement a neural network on FPGA than on CPU or GPU (Shawahna, Sait, & El-Maleh, 2019). The paper (Hao *et al*., 2019) proposed the method of hardware and software co-design, which helps analyze the development of designs and explore the design space. In the paper (Zhang *et al*., 2015), an SIMD convolutional neural network accelerator architecture expands the two-dimensional input and output features to achieve high performance. The actual transmission delay is not considered during the modeling process, resulting in a performance difference of almost 47% to the actual measured result. In (Ding *et al*., 2019), an efficient acceleration architecture placed depthwise separable convolution on FPGA to realize a customized acceleration architecture. Due to the small size of depthwise separable networks and the low flexibility of hardware architecture, the same approach cannot be applied to large networks. As shown in (Ma, Cao, Vrudhula, & Seo, 2017), a full expansion operation was proposed during the optimization process, and all feature maps and weights were cached in on-chip memory, which makes it challenging to achieve the performance on a low-cost FPGA. The paper (Li *et al*., 2016) improves the bandwidth utilization rate by increasing the burst transmission length. It implements parameter reuse and multi-batch processing, which improve the computing unit's utilization rate with fully connected layer. (Shan, Zhang, Deng, & Gong, 2016) proposed a dynamic multi-precision fixed-point data quantization strategy for CNN. The floating-point data on CNN after training was quantized into a fixed-point type. Compared with the previous static quantization strategy, dynamic quantization will reduce the loss in the process.

This paper combines the above mentioned practical acceleration methods for deep learning by analyzing the target platform and proposing the accelerated architecture. The target hardware architectures (Li *et al.*, 2016; Shan *et al.*, 2016) are briefly described and implemented through a high-level synthesis tool. At the same time, considering the low power consumption of the removable platform, we propose to implement the neural network on the PYNQ and use the high-level programming language (Python) to control the work of the platform. The YOLOv2 model achieved 25.98 GOP/s performance on the PYNQ platform and 89.45% detection accuracy.

2. Related Work and Background Theories

State-of-the-art CNNs for large visual recognition tasks usually contain billions of neurons and their trend is to grow deeper and larger. These networks contain millions of neurons and hundreds of millions of parameters, including weights and intermediate feature maps. We need to store these parameters in DRAM and implement large-scale data interaction. The regression model is used to adjust the boundary. The various models data are not shared, so the computational complexity of R-CNN is enormous. YOLO regards the recognition problem as a regression problem. It takes the entire picture as input and can better recognize the background. It has the following three characteristics:

1. Fast speed: YOLO solves object detection as a regression problem and uses a single network to complete the whole detection process.

2. Low recall rate: Low recall rate is reflected in the low background error detection rate.

3. Strong generalization ability: For other kinds of things, the recognition effect after training is also excellent (Ioffe & Szegedy, 2015).

3. Training YOLOv2 Using Motorcycle Dataset

This section describes how to perform training of YOLOv2 using a motorcycle dataset. The motorcycle dataset has been prepared in VOC style.

VOC data are often used for object detection. Since 2005, it has held a competition every year. In the beginning, there were only four categories. By 2007, this had expanded to 20 categories. There are two commonly used versions: 2007 and 2012. In the experiment, the required categories are extracted from the VOC dataset and other datasets, combined with the pictures taken in the application scene and made into a VOC format dataset. This dataset contains 16,000 training pictures and 3,200 validation pictures. During the labelling process, the wheels, engines, fuel tanks, and handlebars of motorcycles are specially marked. Then, K-means clustering algorithm has been applied to optimize the accuracy of training. Finally, this experiment can reach the mAP of 89.45%. More details are given as follows.

Through the analysis of the literature (Redmon & Farhadi, 2017) and the source code of YOLOv2, the target detection steps are as follows:

(1) Image pre-processing: input RGB images of any resolution, convert to [0,1] interval, adjust to 416×416 according to the original image aspect ratio, and fill in 0's if necessary for target size.

(2) Network detection: input the 416×416×3 image array obtained by pre-processing, and after the YOLOv2 network detection, output a 13×13×30 size array containing the detection target information.

(3) Image post-processing: process the output 13×13×30 array to obtain the detection frame center, length and width, frame reliability, and object prediction probability based on the image aspect ratio to recover the scale of the original image.

In the anchor-based object detection algorithm, the anchor is generally designed manually. In SSD and Faster-RCNN (Girshick, 2015), nine anchors with different sizes and

aspect ratios are designed. However, the artificially designed anchors have a drawback. They cannot be guaranteed to match the dataset well. If the anchor's size is different from the object size, it will affect the accuracy of the model. K-means clustering has been used (Zhong, Wang, Peng, & Zhang, 2020) instead of manual design, by clustering the bounding boxes in the dataset to generate a set of anchors that are more suitable. This experiment generated better anchor values based on the K-mean algorithm. During the training phase, YOLOv2 generates the bounding box more accurately.

The software experiment part, as shown in Table 1, performs quantitative analysis on the number of data sets (training and validation), learning rate, training times (Max-batches), training methods (Random), and anchors. Figure 1 shows the final loss and IOU (Intersection over union) in training of iteration 7.

Table 1. Training based on motorcycle dataset

Iteration	Train+val	Learning-rate	Max-batches	Random	Anchors
1	800+160	0.0005	20000	1	(1.3221, 1.73145) (3.19275, 4.00944) (5.05587, 8.09892) (9.47112, 4.84053) (11.2364, 10.0071)
2	1600+320	0.0005	10000	1	(1.3221, 1.73145) (3.19275, 4.00944) (5.05587, 8.09892) (9.47112, 4.84053) (11.2364, 10.0071)
3	800+160	0.0005	10000	1	(1.565, 2.3456) (2.4562, 4.4568) (6.6525, 8.8898) (11.6848, 6.7475) (13.5656, 11.2523)
4	800+160	0.0001	15000	1	(1.565, 2.3456) (2.4562, 4.4568) (6.6525, 8.8898) (11.6848, 6.7475) (13.5656, 11.2523)
5	1600+320	0.0001	15000	0	(1.565, 2.3456) (2.4562, 4.4568) (6.6525, 8.8898) (11.6848, 6.7475) (13.5656, 11.2523)
6	800+160	0.0001	15000	0	(1.08,1.19) (3.42,4.41) (6.63,11.38) (9.42,5.11) (16.62,10.52)
7	1600+320	0.0001	15000	1	(1.08,1.19) (3.42,4.41) (6.63,11.38) (9.42,5.11) (16.62,10.52)
8	800+160	0.0001	10000	0	(1.08,1.19) (3.42,4.41) (6.63,11.38) (9.42,5.11) (16.62,10.52)
9	800+160	0.0001	10000	1	(1.08,1.19) (3.42,4.41) (6.63,11.38) (9.42,5.11) (16.62,10.52)

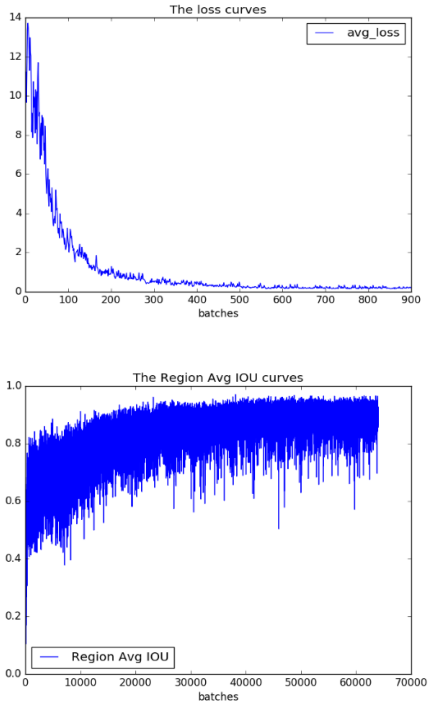

Figure 1. The loss and IOU of final training (Loss is taken every 100 cycles.)

After training, this paper uses the mean average precision (mAP) (1) and recall (2) as the main evaluation criteria.

$$mAP = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Training result as Figure 2, this experiment adjusted the YOLOv2 network parameters for the specificity of single object detection. The mAP reached 89.45%.

4. Hardware Accelerator Design

After the software training, this paper implements YOLOv2 on the hardware platform, which can achieve an efficient performance. Combined with the application requirements, a comprehensive analysis of the hardware and software collaborative design theory (Hao *et al.*, 2019), this study proposed the hardware platform architecture shown in Figure 3. The system's core is to generate IP cores on the HLS tool based on the software model. The CNN accelerator IP design on FPGA consists of four main parts: processing elements (PEs), on-chip buffers, external memory (DDR3), and on-chip storages (built from Block RAMs, LUT, and FIFO). Due to the limitation of on-chip resources, all data will

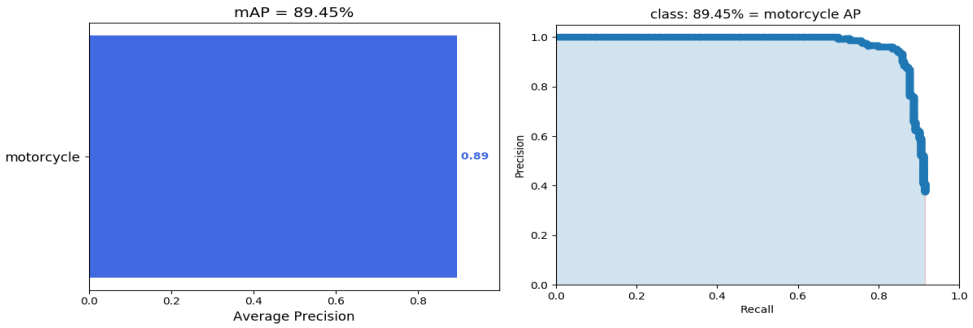

Figure 2. mAP with different training parameters

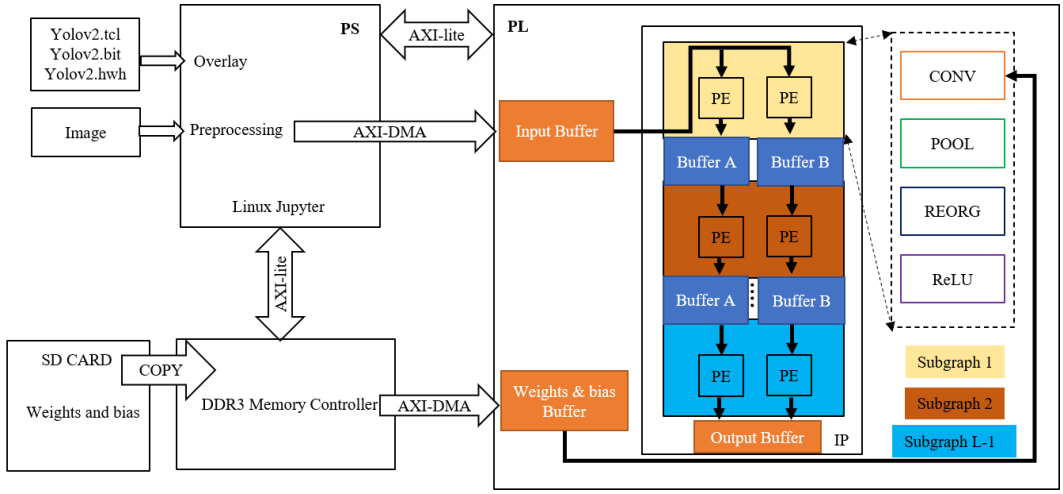

Figure 3. Overview of the YOLOv2 accelerator architecture based on the PYNQ

be stored in the external memory. Before sent to the PEs, the data will be cached in the on-chip buffers (input buffers and weights & bias buffers). Double buffers are used to cover data calculation time and transfer time. After analyzing the central part of the neural network calculation, the convolutional operations will occupy more than 90% calculation amount (Qiu *et al.*, 2016), and the fully connected layer occupies more than 90% of the parameter amount (Ding *et al.*, 2019). Since the fully connected layer can be regarded as a particular convolutional layer, the optimization of calculation is mainly for convolution operations.

In this design, the HLS tool can be used to implement the YOLOv2 network by C++ and automatically generate VHDL code. This conversion method will simplify the difficulty of developing on FPGA and maintain the accuracy of the original neural network. As shown in Figure 3, the PS part uses the advantages of the OpenCV library to pre-process the original image, obtaining a fixed-size grayscale image. Then, through the DMA data transfer protocol, the input feature map and weight parameters are transferred to the PL part in the form of blocks (combined with the size of the hardware resources, the block parameters can be dynamically modified). Figure 4 shows each optimization section.

4.1 Convolutional computation engine design

This section explains the design of a convolutional computation engine. Sections 4.1.1 and 4.1.2 mainly describe the convolutional optimization methods, and sections 4.1.3 and 4.1.4 will describe the dynamic fixed-point quantization and memory optimization.

4.1.1 Loop unrolling and loop reconstruction

As shown in Figure 5, the inputs of the convolutional layer are *C-Hin* input feature maps, and the outputs are *C-Hout* output feature maps. Each "input-output" feature map has a specific convolution kernel for convolution calculation. There are *CHout*×*CHin* convolution kernels, and the size of each convolution kernel is *K*×*K*. In the process of the convolution calculation, each convolution kernel slides through an input feature map. The result of the convolution will be accumulated to the corresponding output feature. Therefore, the algorithm flow of convolution calculation is:

$$\text{Out}\ [\text{cho}][R][C] = \sum_{chi=0}^{CH_{in}-1} \sum_{kr=0}^{K-1} \sum_{kc=0}^{K-1} \text{I n}[\text{chi}][R+kr][C+kc] \times W\ [\text{cho}][\text{chi}][kr][kc] \quad (3)$$

According to the architecture schematic, the pseudo-code is written as in Figure 5. *CHin* multipliers are used to process convolution operations in parallel, which will generate a single PE (processing element). The architecture is composed of multiple PEs, forming the structure in Figure 5.

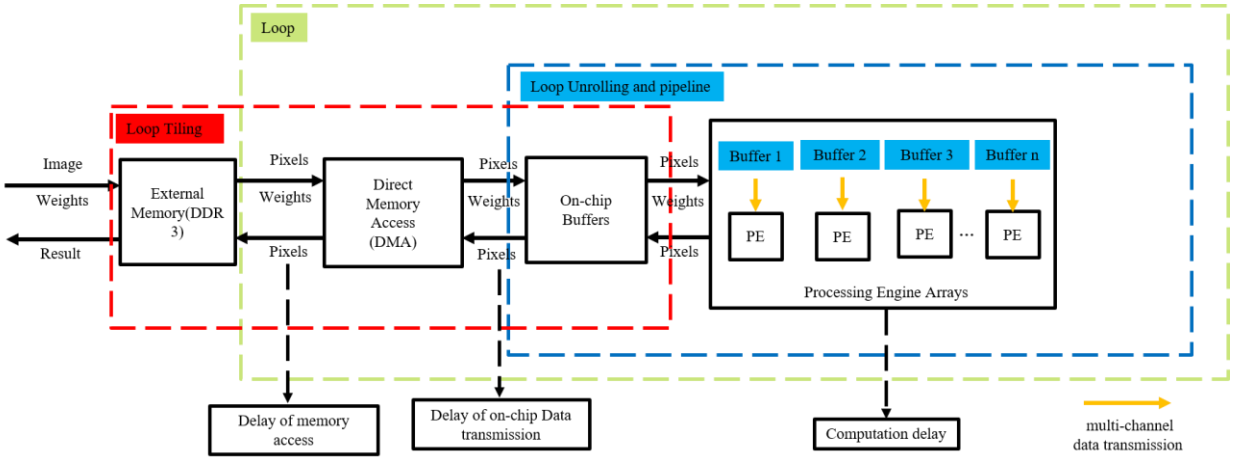

Figure 4. The optimization methods on the PL part

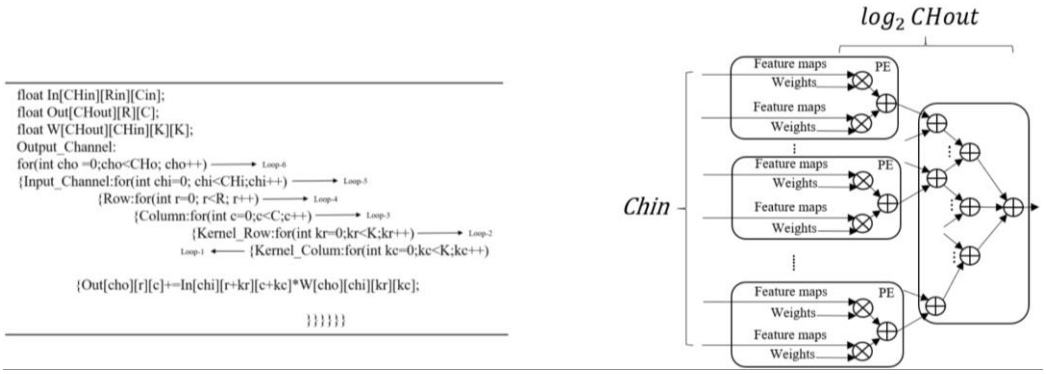

Figure 5. Pseudo code for convolution operations on FPGA

It is necessary to call this acceleration module multiple times to complete the six nested loop calculations of convolution. By partially expanding the number of output feature maps and input feature maps, a *CHin*×*CHout* parallel multiplication calculation unit and *Chin* [log_2*CHout*] depth addition trees are formed as shown in Figure 5.

Loop reconstruction: Since the accelerator is parallelized in the input channel and output channel, the input channel loop and the output channel loop need to be placed in the innermost loop. The polyhedron-based optimization framework (Ma *et al.*, 2017) can be used to perform automatic loop conversion, which can avoid loop dependence.

4.1.2 Loop tiling

PYNQ's on-chip storage resources (generally less than 10 MB) are extremely insufficient compared to the storage space required for CNN calculation. Especially on low-level development boards, most of the research mainly uses convolution calculations' data locality characteristics. Corresponding to HLS implementation, it needs to be applied to loop tiling. Some data only needs to be read or written a few times from off-chip storage. It dramatically reduces the number of memory accesses and the amount of data accessed.

It can allow an acceleration unit that reads *CHin* input feature maps with ((R -1) ×S +K) × ((C -1) ×S +K) pixels from the off-chip DRAM each time and corresponding weight parameters with the size of *CHin* ×*CHout* ×*K*^2.

Multiplexing is done on-chip input feature map pixel blocks and weight parameters to keep the intermediate results in the on-chip buffer. After the final output pixel block is obtained, *CHout* pieces of *R*×*C* pixel blocks are written to an extra chip. As mentioned before, the on-chip storage resources can only accommodate one input data block (In array), one output data block (Out array) and one weight data block (W array). In HLS, the data port of DRAM is expressed in the pointers, which need to interact with the corresponding location of the off-chip DRAM when switching a data block.

4.1.3 Timing and memory system design

Each layer of YOLOv2 has different convolution sizes. When different convolution layers are mapped to the same architecture, the computing unit is often not fully utilized, resulting in low dynamic utilization. This paper proposes to set up a dynamic reusable architecture, combined with pipeline operation, to maximize FPGA resource utilization and minimize operating latency. To reduce the number of interactions between the accelerator and off-chip memory, we design the architecture shown in Figure 6. The accelerator and off-chip memory only interact twice, in the proposed multi-layer cascaded accelerator.

This study performs pipeline and data flow operations on the entire architecture based on multi-layer

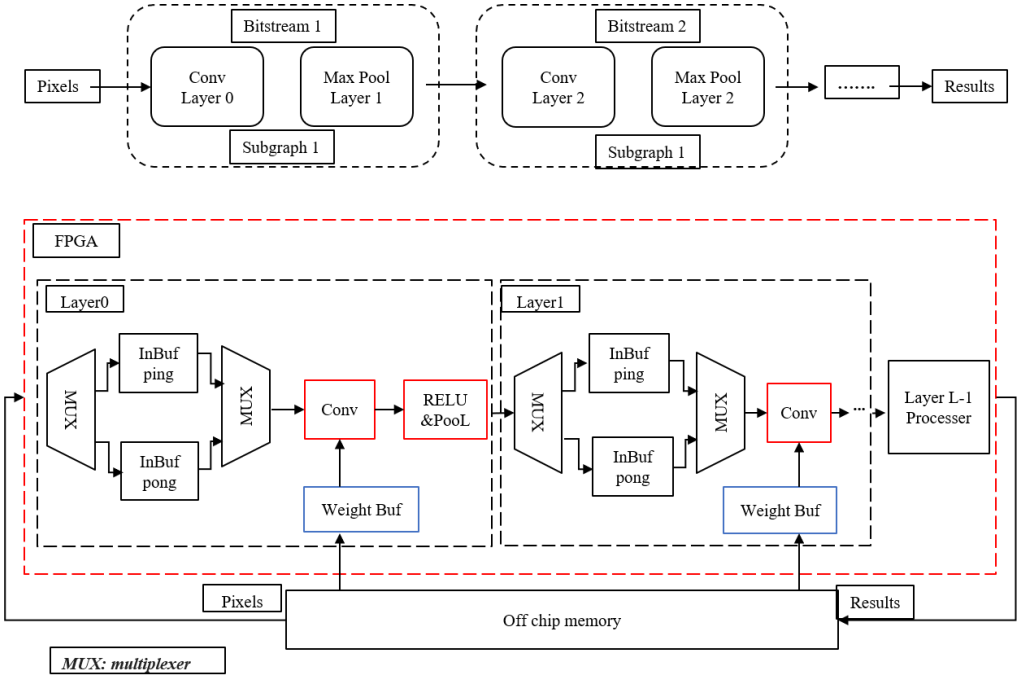

Figure 6. Build subgraphs of different sizes to create a multi-layer cascaded accelerator

cascaded architecture and combining the advantages of FPGA computing parallelism. The pipeline design realizes the concurrency of computing and data interaction at each layer. When Layer 0 reads the input feature map from off-chip memory to the input buffer (InBuf pong), the convolution module of Layer 0 also reads data from the input buffer (InBuf ping) for calculation at the same time. When Layer 0 writes the result of the convolution calculation to the output buffer (OutBuf pong), the latency of the entire system can be judged according to the degree of parallelism. Figure 7 shows an example of the pipeline and dataflow, three-level (Bytyn, Ahlsdorf, Leupers, & Ascheid, 2020).

This helps calculate and compare the latency (Venieris & Bouganis, 2017): The T_1-T_3 is the latency of Task1,2,3.

$$Sum_1 = T_1 + T_2 + T_3 \tag{4}$$

$$Sum_2 = T_1 \cup T_2 \cup T_3 - T_1 \cap T_2 \cap T_3 \tag{5}$$

$$Sum_{save} = Sum_1 - Sum_2 \tag{6}$$

4.1.4 Dynamic fixed-point quantization

YOLOv2 has high computing performance, and its computing power reaches 29.47 GOP. The network needs to be compressed and quantized (Ding *et al*., 2019). Most of the previous work adopted the 16-bit quantization strategy. (Chen *et al*., 2014) showed that using 16-bit numbers instead of 32-bit ones only caused 0.26% increase in error rate on the MNIST dataset. Using short fixed-point numbers instead of long floating-point numbers is efficient for implementations on the FPGA and can significantly reduce memory footprint and bandwidth requirements. As shown in Table 2, this study performed multiplication and addition tests at different precisions. The resources consumed by the multiplication and addition operations of fixed-16 in FPGA are almost half of the float-32.

Table 2. Resource consumption for different types of data

	Type	DSP	LUT
Adder	Float-32	2	214
Mul	Float-32	3	135
Adder	Fixed-16	-	47
Mul	Fixed-16	1	101

Based on analyzing the dynamic fixed-point quantization method (Qiu *et al*., 2016; Shan *et al*., 2016), this study used quantized weights, bias, input feature and output feature. According to the data size range of each layer, various decimal point positions are formulated.

Fixed point number can be expressed as:

$$x_{\text{fixed}} = \sum_{i=0}^{bw-1} B_i \times 2^{-exp} \times 2^i \tag{7}$$

The *bw* represents the bit width of the fixed-point number, and the *exp* represents the exponent, while $B_i \in \{0,1\}$. The conversion between floating point and fixed point is as in (8) and (9):

$$x_{\text{fixed}} = (\text{int})(x_{\text{float}} \times 2^{bw}) \tag{8}$$

$$x_{\text{float}} = (\text{float}) x_{\text{fixed}} \times 2^{-bw} \tag{9}$$

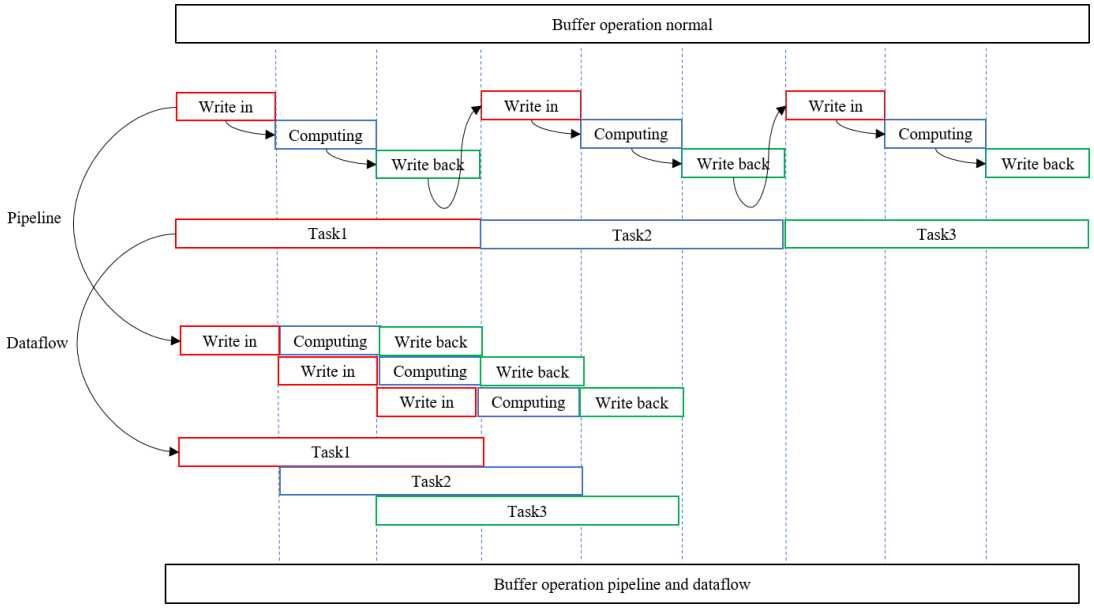

Figure 7. Pipeline for the loop operation and dataflow for the task-level function

5. Experiment Using Hardware

The hardware architecture is built in Vivado 2020.1. The acceleration platform is developed in a Xilinx PYNQ-z2 board. The main chip is Zynq XC7Z020-1CLG400C, which contains a 630 KB block RAM, 220 DSP slices,140 BRAMs, an ARM dual-core Cortex-A9 processor, and an external 512 MB DDR3. On the PS side, use python to call the overlay and edit the python instructions to control the PL side. The hardware part is not friendly to image pre-processing, so the pre-processing part is placed on the PS side. The PL part designs a new IP based on the multi-layer cascade acceleration architecture. The process consists of copying weights and bias files from SD card to DDR3 memory, transferring data through AXI-DMA, and performing convolution acceleration calculations in PL.

5.1 Results and discussion

This study mainly analyzes hardware resource utilization and computing ability per second (GOP/s). Table 3 is our resource usage on PYNQ-z2.

Since the performance is directly related to the development board's operating frequency, this paper conducted independent tests on the performance of 100MHz and 150MHz cases. Table 4 shows the acceleration of deep learning neural networks on various platforms in recent years.

Table 4 compares the frameworks and methods for accelerating object detection on various FPGA levels in recent years. The main comparison metrics are DSP usage, data precision, energy consumption, performance, and accuracy. (Zhao *et al*., 2017) uses fixed-32 to reduce data accuracy loss, but the amount of calculation is increased. Performance is not excellent when using 800 DSPs. (Nakahara, Fujii, Yonekawa, & Sato, 2018) quantifies the data to achieve a very high calculation speed, but the loss of accuracy is excellent on the software level, using technologies such as network compression and pruning reduce the network's parameters and

Table 3. Resource usage on PYNQ-z2

Resource	Utilization	Available	Utilization%
LUT	37230	53200	69.98
LUTRAM	6082	17400	34.95
FF	34012	106400	31.97
BRAM	87.50	140	62.50
DSP (100Mhz)	151	220	68.64
DSP (150Mhz)	153	220	69.54
BUFG	3	32	9.38
MMCM	1	4	25.00

calculations. In the deployment of YOLOv2-Tiny, the performance is very high, but due to changes in the original network architecture during the pruning process, its accuracy is reduced a lot. The performance of the acceleration platform we designed is 6.32 times faster than in the CPU (i7), and the energy consumption is 1/26 of the CPU. About the FPGA platform, under the premise of improving effective accuracy, we optimize the architecture of the hardware platform to exceed the test performance of ZC106 and Cyclone V.

5.2 Conclusions and Future Work

In building a motorcycle detection platform, we propose a motorcycle detection model and acceleration on the FPGA. This paper considers the accuracy and timeliness involved in the actual application. By adopting fixed-16 precision, combined with multiple FPGA parallel optimization methods, this experiment has achieved a performance of 25.98GOP/s on a low-level FPGA platform. At the same time, this paper used the K-means algorithm to reorganize the training parameters and create a YOLOv2-Motorcycle version, which increased the accuracy to 89.45%. Furthermore, the energy consumption was reduced to 2.32W, which solves the problem of insufficient energy for a

Table 4. Comparison of recent hardware optimization studies

	-	Ref (Zhao *et al.*, 2017)	Ref (Wai *et al.*, 2018)	Ref (Nakahara *et al.*, 2018)	Ref (Nguyen *et al.*, 2019)	This	
Platform	CPU(i7)	ZC706	Cyclone V	ZCU102	VC707	PYNQ	
Model	YOLOv2	YOLOv1	YOLOv2-Tiny	YOLOv2	YOLOv2-Tiny	YOLOv2	
Precision	Float-32	Fixed-32	Fixed-16	Fixed-2	Fixed-16	Fixed-16	Fixed-16
Frequency	2.1G	200	117	300	200	100	150
DSP (used/total)	-	800/900	122/224	377/2520	272/2804	151/220	153/220
Power (W)	85	1.17	-	-	11.11	2.32	2.32
Operation (GOP)	29.46	14	5.41	14.18	-	29.46	29.46
Performance (GOP/s)	4.11	18.82	21.60	347.44	464.7	14.10	25.98
Accuracy (%)	-	83.6	-	69.1	51.38	89.45	89.45

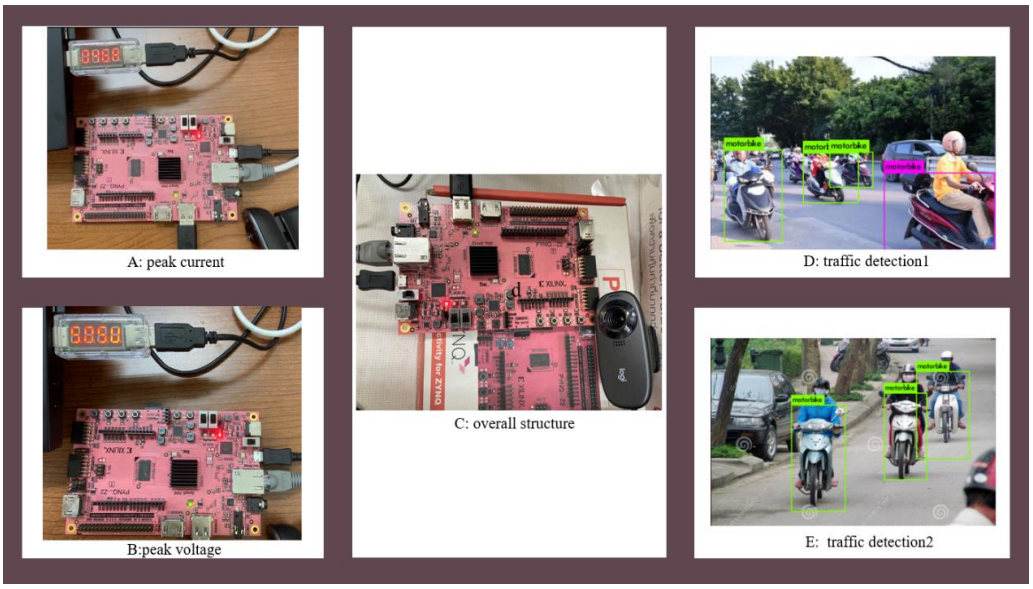

Figure 8. Experimental results

removable platform. In the future, we will be committed to porting the network to a higher-level FPGA, which will support more optimization methods.

Acknowledgements

This work is supported by the electrical engineering of Prince of Songkla University, 6110120028. We also appreciate the careful works and the constructive suggestions of the anonymous reviewers.

References

Bytyn, A., Ahlsdorf, R., Leupers, R., & Ascheid, G. (2020). Dataflow aware mapping of convolutional neural networks onto many-core platforms with network-on-chip interconnect, 1–11. Retreived from http://arxiv.org/abs/2006.12274

Blaiech, A. G., Khalifa, K. B., Valderrama, C., Fernandes, M. A. C., & Bedoui, M. H. (2019). A survey and taxonomy of FPGA-based deep learning accelerators. *Journal of Systems Architecture, 98*, 331–345. doi:10.1016/j.sysarc.2019.01.007

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2014). DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pp. 269–283. doi:10.1145/2541940.2541967

Ding, W., Huang, Z., Huang, Z., Tian, L., Wang, H., & Feng, S. (2019). Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *Journal of Systems Architecture, 97*, 278–286. doi:10.1016/j.sysarc.2018.12.008

Fang, J., Mulder, Y. T. B., Hidders, J., Lee, J., & Hofstee, H. P. (2020). In-memory database acceleration on FPGAs: a survey. *The VLDB Journal, 29(1)*, 33–59. doi:10.1007/s00778-019-00581-w

Gschwend, D. (2016). ZynqNet: An FPGA-accelerated embedded convolutional neural network. August 2016, 1–102. Retreived from https://github.com/dgschwend/zynqnet

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 1440–1448. doi:10.1109/ICCV.2015.169

Guo, K., Zeng, S., Yu, J., Wang, Y., & Yang, H. (2017). A survey of FPGA-based neural network accelerator, 9(4), 1–26. Retreived from http://arxiv.org/abs/1712.08934

Hao, C., Zhang, X., Li, Y., Huang, S., Xiong, J., Rupnow, K., Hwu, W., & Chen, D. (2019). FPGA/DNN Co-Design: An efficient design methodology for IoT intelligence on the edge. Retreived from https://arxiv.org/abs/1904.04421

Lee, C., Pino, J., Lin, P., & Peters, E. (2013). Motorcycle type matters: Use of helmet, speeding, and drinking in motorcycle crashes. *Proceeding of the 2013 TRB 92nd Annual Meeting*.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521(7553)*, 436–444. doi:10.1038/nature14539

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2020). Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2)*, 318–327. doi:10.1109/TPAMI.2018.2858826

Li, H., Fan, X., Jiao, L., Cao, W., Zhou, X., & Wang, L. (2016). A high performance FPGA-based accelerator for large-scale convolutional neural networks. *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, pp.1–9. doi:10.1109/FPL.2016.7577308

Ma, Y., Cao, Y., Vrudhula, S., & Seo, J. S. (2017). Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 45–54. doi:10.1145/3020078.3021736

Nakahara, H., Fujii, T., Yonekawa, H., & Sato, S. (2018). A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA. *FPGA 2018 - Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 31–40. doi:10.1145/3174243.3174266

Nguyen, D. T., Nguyen, T. N., Kim, H., & Lee, H. J. (2019). A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(8)*, 1861–1873. doi:10.1109/TVLSI.2019.2905242

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., . . . Yang, H. (2016). Going deeper with embedded FPGA platform for convolutional neural network. *FPGA '16: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 26-35. doi:10.1145/2847263.2847265

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016) You only look once: unified, real-time object detection. *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788. doi:10.1109/CVPR.2016.91

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517–6525. doi:10.1109/CVPR.2017.690

Siebert, F. W., & Lin, H. (2020). Detecting motorcycle helmet use with deep learning. *Accident Analysis and Prevention, 134 (January 2020)*, 105319. doi:10.1016/j.aap.2019.105319

Shawahna, A., Sait, S. M., & El-Maleh, A. (2019). FPGA-Based accelerators of deep learning networks for learning and classification: A review. *IEEE Access, 7*, 7823–7859. doi:10.1109/ACCESS.2018.2890150

Shan, L., Zhang, M., Deng, L., & Gong, G. (2016). A dynamic multi-precision fixed-point data quantization strategy for convolutional neural network. *Communications in Computer and Information Science, 666 CCIS(20124307110016)*, 102–111. doi:10.1007/978-981-10-3159-5_10

Venieris, S. I., & Bouganis, C. S. (2017). Latency-driven design for FPGA-based convolutional neural networks. *Proceeding of the 27th International Conference on Field Programmable Logic and Applications (FPL) 2017*, 1-8. doi:10.23919/FPL.2017.8056828

Wai, Y. J., Yussof, Z. bin M., bin Salim, S. I., & Chuan, L. K. (2018). Fixed point implementation of Tiny-YOLO-v2 using OpenCL on FPGA. *International Journal of Advanced Computer Science and Applications, 9(10)*, 506–512. doi:10.14569/IJACSA.2018.091062

Ye, X. Y., Hong, D. S., Chen, H. H., Hsiao, P. Y., & Fu, L. C. (2020). A two-stage real-time YOLOv2-based road marking detector with lightweight spatial transformation-invariant classification. *Image and Vision Computing, 102*, 103978. doi:10.1016/j.imavis.2020.103978

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 448-456.

Zhao, M., Hu, C., Wei, F., Wang, K., Wang, C., & Jiang, Y. (2019). Real-time underwater image recognition with FPGA embedded system for convolutional neural network. *Sensors (Switzerland), 19(2)*, 350. doi:10.3390/s19020350

Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. *FPGA 2015 - 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 161–170. doi:10.1145/2684746.2689060