Hardware Implementation and Quantization of Tiny-Yolo-v2 using OpenCL

Yap June Wai, Zulkalnain bin Mohd Yussof, Sani Irwan bin Md Salim

Abstract*: The trend of increasingly model size in Deep Neural Network (DNN) algorithms boost the performance of visual recognition tasks. These gains in performance have come at a cost of increase in computational complexity and memory bandwidth. Recent studies have explored the fixed-point implementation of DNN algorithms such as AlexNet and VGG on Field Programmable Gate Array (FPGA) to facilitate the potential of deployment on embedded system. However, there are still lacking research on DNN object detection algorithms on FPGA. Consequently, we propose the implementation of Tiny-Yolo-v2 on Cyclone V PCIe FPGA board using the High-Level Synthesis Tool: Intel FPGA Software Development Kit (SDK) for OpenCL. In this work, a systematic approach is proposed to convert the floating point Tiny-Yolo-v2 algorithms into 8-bit fixed-point. Our experiments show that the 8-bit fixed-point Tiny-Yolo-v2 have significantly reduce the hardware consumption with only 0.3% loss in accuracy. Finally, our implementation achieves peak performance of 31.34 Giga Operation per Second (GOPS) and comparable performance density of 0.28GOPs/DSP to prior works under 120MHz working frequency.

Keywords: DNN, FPGA, Tiny-Yolo-v2, Quantization.

I. INTRODUCTION

Recent advances in deep neural networks have substantially led to significant progress in many machine learning problems involving object classification [1] [2] [3], speech recognition [4] [5] [6], and object detection [7] [8] [9]. The advance of deep neural networks led to impressively high accuracy which outperform conventional machine learning algorithms. However, all these advances come with the cost of increase in computational complexity and memory footprint. The state-of-the art of modern DNN algorithms consist over millions of weights and over billion operations to compute an input. Consequently, it is often a challenge to implement DNN using conventional general processing unit (Von Neumann architecture). Today, most of the training of DNN model are accelerated using the Graphic Processing Unit (GPU), which is extremely power hungry, and it is often a challenge to practice DNN for real life applications targeted on low-power embedded system.

Revised Manuscript Received on July 22, 2019.

Yap June Wai, Center for Telecommunication Research and Innovation, Faculty of Electronic and Computer Engineering, Universiti Teknikal Malaysia Melaka, Melaka Malaysia. Email: junewai1993@gmail.com

Zulkalnain bin Mohd Yussof, Center for Telecommunication Research and Innovation, Faculty of Electronic and Computer Engineering, Universiti Teknikal Malaysia Melaka, Melaka Malaysia. Email: zulkalnain@utem.edu.my

Sani Irwan bin Md Salim, Center for Telecommunication Research and Innovation, Faculty of Electronic and Computer Engineering, Universiti Teknikal Malaysia Melaka, Melaka Malaysia. Email: sani@utem.edu.my

To alleviate this problem, FPGAs have been explored recently as an alternative solution to implement the DNN algorithms due to their high performance, power efficiency and reconfigurability. Prior works have shown that it is more efficient to implement DNN algorithms in a fixed point, which offers the advantages of reducing in power consumption, computations load and memory bandwidth. Prior work [10] proposed that a method to train DNN with low precision weights and achieved a comparable accuracy as the 32-bit model using only 4-bit. However, in this work, an alternative approach to convert the DNN without training is focused as it is more efficient to convert well-trained DNN object detection model in real life deployment.

Prior works [10] [11] [12] [13] mainly focus on the implementation of 8-bit fixed point in classification algorithms: AlexNet and VGG which are running on ImageNet datasets. The fixed-point representations designed for AlexNet and VGG are not applicable in others DNN model such as Tiny-Yolo-v2. This is due to the fixed-point representation is dependent to the dynamic range of weights and activations of a DNN model. In this work, the focus will be on the implementation of fixed-point representation designed for DNN object detection algorithms: Tiny-Yolo-v2 running on Pascal VOC 2007 [14] and COCO [15]. Prior works [16] [17] have shown the effort on the 16-bit fixed-point implementation of in Tiny-Yolo algorithm and achieved less than 1% loss in accuracy. However, in this work, the 8-bit fixed point implementation of DNN algorithms: Tiny-Yolo-v2 is explored. The key contributions are summarized as follows:

- A scalable FPGA accelerator to run Tiny-Yolo-v2 on object detection datasets: Pascal VOC 2007 and COCO.
- A systematic approach to convert the floating-point Tiny Yolo-v2 to 8-bit fixed point model.
- An analysis on the impact of the 8-bit fixed-point implementation of Tiny Yolo-v2 in term of hardware resource consumption, accuracy and computation throughput.

The rest of the paper is arranged as follows. Section II has background on the current works on the implementation of CNNs targeting on FPGA. Section III presents a brief description on the proposed accelerator design using General Matrix-Matrix Multiplication (GeMM) approach and the proposed quantizer to convert floating-point of Tiny-Yolo-v2 to fixed-point. In section IV,

the experimental results discussion will be carried out to present the utilization of hardware resource on accelerator designed in this work. Section 4 concludes the paper.

II. BACKGROUND

A. Intel FPGA SDK for OpenCL

In fact, Intel FPGA SDK for OpenCL [18][19] is developed to provide a high-level abstraction for FPGA design development. It allows users to perform hardware synthesis from high-level programming language to Register-Transfer-Level (RTL). In the OpenCL-based FPGA accelerator model, the Central Processing Unit (CPU) serves as the host and it is connected to the OpenCL device, FPGA via Peripheral Component Interconnect Express (PCIE) interface. Eventually, this model forms a heterogeneous computing system. The kernel written in OpenCL is compiled into hardware image running on the FPGA. In other hand, on the host side, a program written in C/C++ is running to communicate with the implemented OpenCL kernel.

B. Tiny-Yolo-v2 Object Detection Algorithm

In this section, the overview of object detection model, Tiny-Yolo-v2 is briefly discussed. Figure 1 indicates how object detection task in Tiny-Yolo-v2 is performed as a single regression problem straight from image pixels to bounding box coordinates and class probability. The input image is divided into S × S grid. Each grid cell predicts B bounding boxes, confidence for those boxes and C class probabilities. These predictions are encoded as an S × S × (B * 5 + C). Unlike prior object detection algorithms which apply classifiers at multiple location, Tiny-Yolo-v2 implements a single convolutional network to whole input image. This makes Tiny-Yolo-v2 is much faster than prior object detection algorithms.

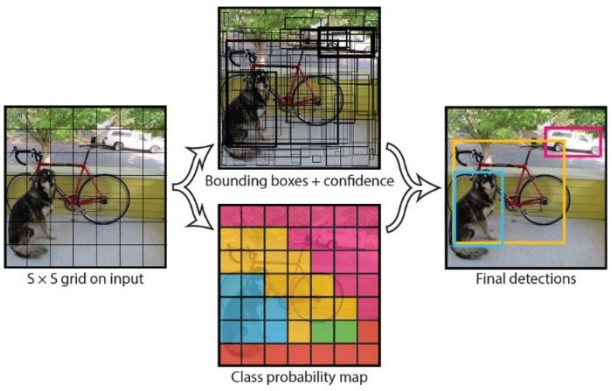

Fig. 1. Yolo Object Detection Algorithm.(adopted from [8])

C. Performance Evaluation Metrics

This section discussed about the evaluation metrices that are used to evaluate the performance of the accelerator designed to run Tiny-Yolo-v2 on Cyclone V PCIe board. In this research, the computation is performed in a lower precision (8-bit) compared to the original Tiny-Yolo-v2which is computed in floating-point (32-bit). It is expected that the overall accuracy performance might be affected. The accuracy of the accelerator is evaluated in mean average precision (mAP). In addition, to make a comprehensive comparison on the computation throughput of proposed accelerator running on different hardware, performance density is used to measure the performance of proposed design. The performance density is formulated as in (1).

$$\text{Perf.Density} = \frac{\text{Throughput}}{\text{DSP\_Consumed}} \quad (1)$$

D. Related Works

Recent studies show the trend of using FPGA as an alternative solution in hardware acceleration, where FPGA is used to accelerate part of the computation in compute-intensive algorithms such as DNN-based algorithms. Zhang et al. (2015) introduced the implementation of DNNs algorithm on FPGA using the High-Level Synthesis (HLS) tool and it proved that the HLS implementation can achieve comparable performance with the traditional Register-Transfer-Level (RTL) approach. Later implementation by Suda et al. (2016) showed a fixed-point implementation of AlexNet and VGG on FPGA using Intel FPGA SDK for OpenCL platform. The implementation 8-16 fixed point design finally achieved 72.4 GOPS and 117.8 GOPs on AlexNet and VGG running on an Intel Stratix-V GSD8 FPGA chip.

Soon, in 2017, Wang et al. (2017) implemented FPGA accelerator, namely PipeCNN with a deeply pipelined OpenCL kernels to fully utilize the capability of pipelining kernel functions to minimize the memory bandwidth requirement on two representative large-scale image classification model, AlexNet and VGG on Stratix V A7 FPGA. It was reported that the shortest classification time achieved to be 43 ms for AlexNet and 718ms for VGG-16. The concept of "Performance Density" was introduced by Wang et al. to make a better and fair comparison in the computational throughput of the design with previous works (Zhang et al., 2015; Suda et al., 2016).

In the other hand, Ma et al. (2017) first explored the hardware acceleration using FPGA on DNN object detection algorithm: Tiny Yolo. In this work, a singular value decomposition (SVD) was proposed to reduce the number of weights in the fully connected layers. The proposed SVD method is proved to be an effective approach to reduce the number of parameters in fully connected at the scale of 5.15x. However, this technique was not applicable to the Tiny Yolo-v2, due to the depreciated fully connected layers in the new version of YOLO algorithm.

III. METHODOLOGY

In this section, the overview of the implementation of GeMM based convolution will be presented. In addition, the steps of converting the floating point Tiny-Yolo-v2 to 8-bit fixed-point precision is also discussed in detailed.

A. Implementation of General Matrix-Matrix Multiplication Based Convolution

As a classical DNN algorithms, Tiny-Yolo-v2 forwards the trained model for object detection, which more than 90% of operations are involved in convolutional layers. Noted that the convolution operations are actually performing addition and multiplication operations between the kernel filters and local region of inputs. The insights of 3 dimensional convolutions is mapped into General Matrix-Matrix Multiplication based convolution is borrowed from previous work to accelerate Tiny-Yolo-v2 algorithm running on FPGA. To perform GeMM based convolution, the convolutional layers have to be flattened and rearranged into a 2-dimensional matrix. Figure 2 shows the data rearrangement process to flatten the first convolutional layer in Tiny-Yolo-v2. The dimension of the input of first layer in Tiny-Yolo-v2 is 416 × 416 × 3 (Hin × Win × Nin) and the size of kernel is 3 × 3 (K × K). The input image is flattened and rearranged vertically to the dimension of 416 × 416 × 3 × 3× 3.

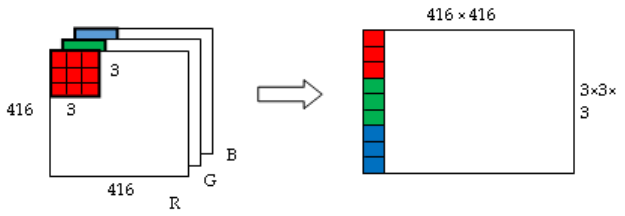

Fig. 2. Data Rearrangement Process.

Taking advantages of the data reuse and data parallelism in the nature of GeMM based convolution, the accelerator is optimized by using the block tiling technique. Figure 3 shows that instead of performing data fetching one by one, the data is transferred to the local memory in block to reduce the latency. After that, the computation is carried out in Single Instruction and Multiple Data (SIMD) manner to improve the computation throughput. Two parameters: BLOCK_SIZE and SIMD factor are introduced in the accelerator configuration. While the former parameter represents the size of block of fetched data to local memory and the second parameter decides the extent of parallelism in data level of accelerator. By adjusting the parameters of BLOCK_SIZE and SIMD, a scalable accelerator for Tiny-Yolo-v2 algorithms can be achieved. In this work, the best configuration of the parameters: BLOCK_SIZE and SIMD are 32 and 4 respectively due to the limited hardware resources of Cyclone V PCIE Development Kit.

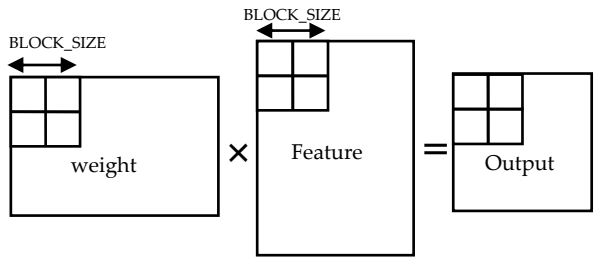

Fig. 3. Implementation of Block Tiling Technique.

B. Floating Point Tiny-Yolo-v2 to Fixed-Point Conversion

The proposed steps to convert the floating-point Tiny-Yolo-v2 to fixed-point (8-bit) are illustrated in diagram 4. Firstly, the floating-point Tiny-Yolo-v2 is forwarded using a large set of input images. In this work, the validation image set from Pascal VOC 2007 is used as the input to collect the statistic of weights, biases and activations for each layer in Tiny-Yolo-v2. The data distribution of the collected weights, biases and activations for each layer in Tiny-Yolo-v2 are analyzed accordingly to determine the range of data. Based on the distribution and the data range, the fixed-point format of weights, biases and activations are determined for each layer.

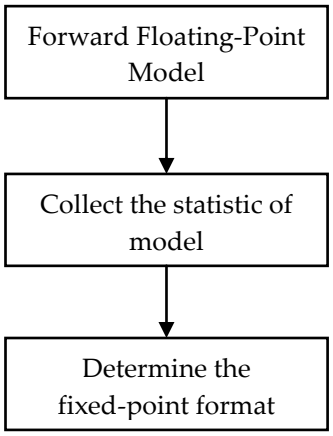

Fig. 4. Steps of converting the floating point Tiny-Yolo-v2 to fixed-point.

The analysis shows that the data distribution and data range vary from image to image and layer to layer, hence, in this work, a dynamic range fixed-point quantization technique is proposed to perform quantization on the weights, biases and activations instead of the uniform quantizer proposed in previous work. The formula of the proposed dynamic quantizer is shown in equation below:

$$Quantized\_Input \approx \frac{1}{S}\left(input_f \times 2^{bw-1}\right) \quad (2)$$

Where

Inputf = weights, biases and activations
S = Scale
bw = bit width

The proposed equation is relatively simple by introduce a variable S where S represents a scale ratio which relatively big to scale down the range of distribution of weights and activations in each layer of Tiny-Yolo-v2 algorithms. The value S is obtained based on the data collected from the feedforward of floating point Tiny-Yolo-v2 running on Pascal VOC 2007 validation datasets.

In this work, the value of S of each layer is the mean of the largest value of all input images in each layer of Tiny-Yolo-v2 in the validation set of Pascal VOC 2007. The equation to obtain the value of S is illustrated as shown:

$$Sj = \frac{1}{n}\sum_{i=1}^{n} max(x_j{}^i) \quad (3)$$

Where
n= number of input images
j = number of layers in Tiny-Yolo-v2
x = inputs in floating point

Noted that the offset of collected data of each layer are eliminated using the Interquartile Range (IQR) technique. Lastly, to avoid overflow of data, saturation will be performed to data which are out of the range that can be represented in proposed 8-bit fixed-point format. To further optimize the hardware resources in FPGA, the value S is round up to nearest power of 2, which the multiplication and division process can be carried using the bit shifting

IV. RESULT AND ANALYSIS

In this section, the hardware specification of the Cyclone V PCIe FPGA is firstly presented in Table I. The board is selected in this work due to its' relatively great amount of Logic Elements, Embedded Memory Block and the newest version of supported OpenCL SDK.

Table- I: Hardware Specification of Cyclone V PCIe FPGA Board

	C5P Development Kit
Logic Element	301 K
Embedded Memory	13,917 Kbits
Type	Server Based
FPGA	Cyclone V 5CGXFC9D6F27C7 N
OpenCL SDK version	Ver 17.1 Intel SDK OPENCL

The hardware resources utilization of proposed architecture on Cyclone V PCIe FPGA are summarized in Table II. Figure 5 clearly shows that by performing 8-bit quantization, the proposed accelerator design can achieve 115% improvement in ALUTs consumption, 34% improvement in RAM consumption and 24% improvement in DSP block consumption compared to the 32-bit precision design (floating point).

Table- II: Hardware Specification of Cyclone V PCIe FPGA Board

	ALUTs	RAM	DSP
32-bit	161%	70%	59%
16-bit	64%	40%	41%
8-bit	46%	36%	35%

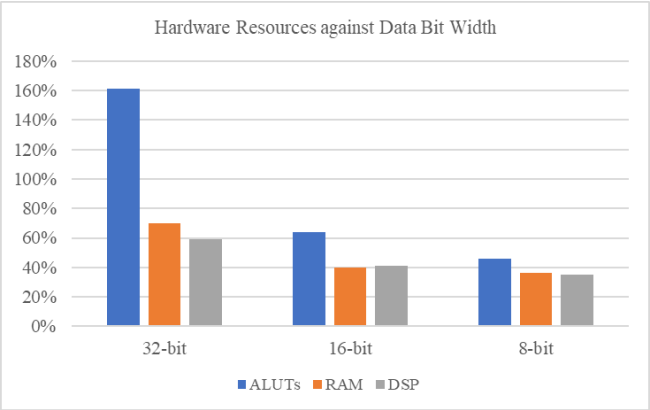

Fig. 5. Hardware Resources Utilization of Proposed 8-bit Fixed-Point Tiny-Yolo-v2.

The throughput of the accelerator with different parameter configuration are also analyzed to make sure the accelerator is scalable, so that it can be implemented in the FPGA board with more hardware resources. The throughput of accelerator with different BLOCK_SIZE configurations: 4, 8, 16 and 32 are shown in figure 6. The figure shows that the throughput increases almost x2 when the parameter increases in a scale of 2. In this work, the best configuration of BLOCK_SIZE and SIMD are 32 and 4 respectively due to the constraints in hardware resources in Cyclone PCIE FPGA Development Kit.

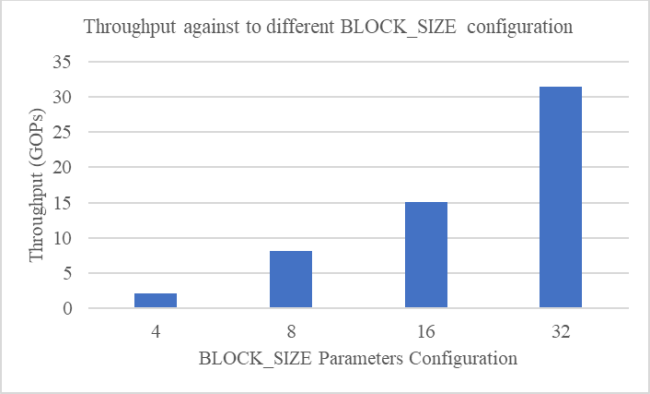

Fig. 6. Performance of Proposed 8-bit Fixed-Point Tiny-Yolo-v2 in different BLOCK_SIZE Configuration.

In this section, the benchmarking of the hardware implementation for Tiny-Yolo-v2 on FPGA compared to the software implementation is presented. The designed hardware accelerator with 8-bit fixed-point arithmetic is built on the Cyclone V PCIE Development Kit. On the other hand, the software implementation of Tiny-Yolo-v2 is implemented on the Central Processing Unit (CPU): Intel ® Core TM i7-7700. The performance is evaluated based on the *Execution Time* of the algorithm, which referring the total time taken to complete the detection on one single input image.

The execution time of between the proposed accelerator and the work on CPU is shown in Table-III. It shows that the proposed hardware accelerator on FPGA able to accelerate the computation time at the scale of 6.17 times faster than the software implementation on CPU.

Table- III: Hardware and Software Implementation

Layer	CPU (Intel R Core TM i7-7700)	FPGA (This Work)
1	0.083	0.027
2	0.123	0.018
3	0.112	0.016
4	0.094	0.017
5	0.096	0.015
6	0.098	0.016
7	0.372	0.034
8	0.381	0.068
9	0.019	0.012
Total Execution Time (second)	1.378	0.223

In Table-IV, the work is compared to other prior works designs. In this work, the FPGA board used: Cyclone V PCIe which is different from the board used in previous study. Hence, the performance of accelerator is measured using the performance density metrices as discussed in previous section. It is clearly shown that 8-bit implementation of Tiny-Yolo-v2 achieved comparable performance to previous work.

Table- IV: Comparison to Previous Work

	[11]	[17]	This work
Device	Stratix-VGXA7	Cyclone V PCIe	Cyclone V PCIe
FPGA Capacity	622k LUTs	113K LUTs	113K LUTs
Model	AlexNet, VGG	Tiny-Yolo-v2	Tiny-Yolo-v2
Frequency	120MHz	117MHz	120MHz
Precision	Fixed (8-bit)	Fixed (16-bit)	Fixed (8-bit)
Throughput	117.8 GOPs	21.6GOPs	31.43GOPs
DSP Consumed	246	122	110
Performance Density	0.29GOPs/DSP	0.18GOPs/DSP	0.28GOPs/DSP

Lastly, the performance of the 8-bit implementation of Tiny-Yolo-v2 running on Cyclone V FPGA board is also measured in mean Average Precision (mAP). Figure 7 shows that pictures tested by the accelerator. Despite the computation is carried out at the lower precision (8-bit), the accelerator still able to detect all person and motorbike correctly. The accelerator has achieved only 0.3% loss in mAP compared to original Tiny-Yolo-v2 while substantially reduce a great amount of hardware resources.

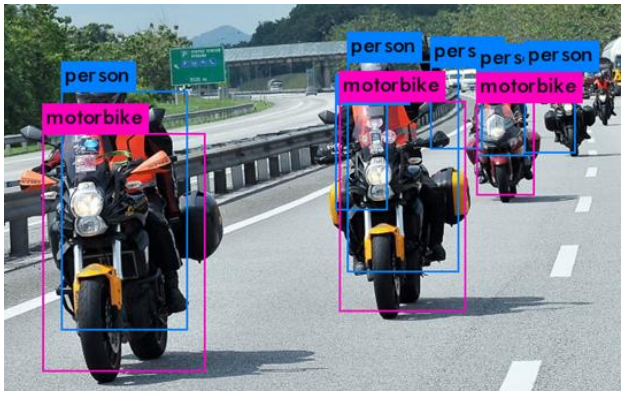

Fig. 7. Tested Image.

V. CONCLUSION

In conclusion, a scalable accelerator for DNN object detection algorithm: Tiny-Yolo-v2 is implemented on Cyclone V PCIe FPGA board. By implementing the proposed approach to quantize the parameters in Tiny-Yolo-v2, despite the computations are performed in lower precision (8-bit), the accelerator could achieve a peak throughput of 31GOPs with only 0.3% loss in mAP under 120MHz. The performance density of 0.28GOPs/DSP of the accelerator is comparable to previous work.

ACKNOWLEDGMENT

Authors would like to thank the to the support from Center for Telecommunication Research and Innovation (CeTRI) and the support of UTeM Zamalah Scheme, Faculty of Electronic and Computer (FKEKK), Universiti Teknikal Malaysia Melaka.

REFERENCES

1. A. Krizhevsky, I. Sutskever and GE. Hinton, “Imagenet classification with deep convolutional neural networks”, Advances In Neural Information Processing Systems, 2012, pp. 1097-1105.
2. Zeiler, Matthew D., and Rob Fergus, "Visualizing and understanding convolutional networks" In European conference on computer vision, Springer, Cham, 2014, pp. 818-833.
3. K. Simonyan, and A.Zisserman, “Very deep convolutional networks for large-scale image recognition”, arXiv preprint arXiv:1409.1556, 2014.
4. G. Hinton, D. Li, Y. Dong, George E. Dahl, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal processing magazine 29, no. 6, 2012, pp. 82-97.
5. D. Li, Geoffrey Hinton, and Brian Kingsbury. "New types of deep neural network learning for speech recognition and related applications: An overview." In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference, IEEE, 2013, pp. 8599-8603.
6. A. Graves, M. Abdel-rahman, and H. Geoffrey. "Speech recognition with deep recurrent neural networks." In Acoustics, speech and signal processing (icassp), 2013 ieee international conference, IEEE, 2013, pp. 6645-6649.
7. S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, Advances In Neural Information Processing systems, 2015, pp. 91-99.
8. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, Proceedings of IEEE Conference on Computer Vision and Pattern recognition, 2016, pp. 779-788.

9. J. Redmon, and Farhadi, “A. YOLO9000: better, faster, stronger”, arXiv, 2017.
10. Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized neural networks: Training neural networks with low precision weights and activations." The Journal of Machine Learning Research 18, no. 1 (2017): 6869-6898.
11. N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.S. Seo, and Y. Cao, “Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks,” in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016, pp. 16-25.
12. J. Zhang, and J. Li, “Improving the performance of OpenCL-based fpga accelerator for convolutional neural network”, Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 25-34.
13. Lin, Darryl, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks." In International Conference on Machine Learning, pp. 2849-2858. 2016.
14. M. Everingham, L. Van Gool, C.K. Williams, J. Winn and A. Zisserman, “The pascal visual object classes (voc) challenge”, International journal of computer vision, 2010, 88(2), pp.303-338.
15. T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick, “Microsoft coco: Common objects in context”, European conference on computer vision, Springer, 2017, pp. 740-755.
16. J. Ma, L. Chen, Gao, “Hardware Implementation and Optimization of Tiny-YOLO Network”, International Forum on Digital TV and Wireless Multimedia Communications, Springer, Singapore, 2017, pp. 224-234.
17. J.W. Yap, Z.M. Yussof, S.I. Salim, K.C. Lim, “Fixed Point Implementation of Tiny-Yolo-v2 using OpenCL on FPGA”, International Journal of Advanced Computer Science and Applications, 9(10), 2018, pp. 506-512.
18. FPGA SDK for OpenCL Programming Guide., Intel, 2017, pp. 70-80.
19. FPGA SDK for OpenCL Best Practice Guide, Intel, 2017, pp. 17-20.

AUTHORS PROFILE

Yap June Wai received his B.Eng degree in year 2017 in Electronic & Computer Engineering from University Teknikal Malaysia Melaka. He is currently working towards the M.S degree in science at University Teknikal Malaysia Melaka, Malaysia. His major interests include artificial intelligent, machine learning, and embedded system design.

Zulkalnain bin Mohd Yussof received his Ph.D in Electrical Engineering from Washington State University, USA. He is working as a Professor, Center for Telecommunication Research and Innovation, Universiti Teknikal Malaysia Melaka. His major interests include image/video Compression, digital system design, high-speed digital signal processing, baseband signal processing for wireless communication system, RF systems, and embedded system.

Sani Irwan bin Md Salim received his M.s in science computer & Communication from Queensland University of Technology, Australia. He is working as a Senior Lecturer, Center for Telecommunication Research and Innovation, Universiti Teknikal Malaysia Melaka...