A Low-Latency Hardware Accelerator for YOLO Object Detection Algorithms

Aibin Wang[1,2], Youshi Ye[2], Yu Peng[2], Dezheng Zhang[1], Zhihong Yan[1], and Dong Wang[1](✉)

[1] Institute of Information Science, Beijing Jiaotong University, Beijing 100044, China
wangdong@bjtu.edu.cn

[2] Beijing Institute of Control Engineering, Beijing 100190, China

Abstract. Object detection is an important computer vision task with a wide range of applications, including autonomous driving, smart security, and other domains. However, the high computational requirements poses challenges on deploying object detection on resource-limited edge devices. Thus dedicated hardware accelerators are desired to delever improved performances on detection speed and latency. Post-processing is a key step in object detection. It involves intensive computation on the CPU or GPU. The non-maximum suppression (NMS) algorithm is the core of post-processing, which can eliminate redundant boxes belonging to the same object. However, NMS becomes a bottleneck for hardware acceleration due to its characteristics of multiple iterations and waiting for all predicted boxes to be generated.

In this paper, we propose a novel hardware-friendly NMS algorithm for FPGA accelerator design. Our proposed algorithm alleviates the performance bottleneck of NMS by implementing the iterative algorithm into an efficient pipelined hardware circuit. We validate our algorithm on the VOC2007 dataset and show that it only brings 0.27% difference compared to the baseline NMS. Additional, the exponential function and sigmoid function are also extremely hardware-costly. To address this issue, we propose an approximate exponential function circuit to calculate the two functions with minimum logic cost and zero DSP cost.

We deploy our post-processing accelerator on Xilinx's Alveo U50 FPGA board. The final design achieves a end-to-end detection latency of 283us for YOLOv2 model, According to the user guide provided by Xilinx and Intel, we converted the logic resources of different implementations on the FPGA into LUT resources. After that, we compared the resource utilization of acceleration module in the current state-of-the-art object detection system deployed on Intel with ours. Compared with it, we consumed 13.5× lower LUT resources and used much fewer DSP resources.

Keywords: Object detection · Neural network accelerator · Hardware-friendly NMS algorithm

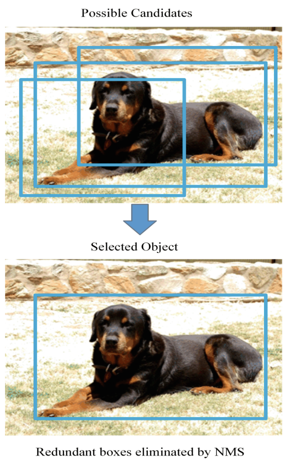
This work was partially supported by Open Fund (NO. OBCandETL-2022-06) of Space Advanced Computing and Electronic Information Laboratory of BICE.

C. Li et al. (Eds.): APPT 2023, LNCS 14103, pp. 265–278, 2024.
https://doi.org/10.1007/978-981-99-7872-4_15

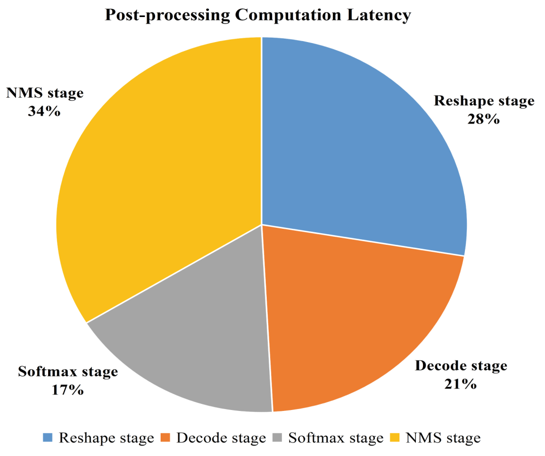
1 Introduction

In the field of computer vision, object detection algorithms based on deep learning have received wide attention and are widely applied [16]. According to the different detection methods, object detection algorithms can be divided into single-stage and two-stage detection algorithms. Generally, single-stage object detection algorithms have faster detection speed than two-stage algorithms, but there is a slight loss in accuracy. Single-stage object detection algorithms are often more preferred for embedded devices such as autonomous driving applications, as they simplify the object detection process and are more suitable for mobile embedded devices. Traditional single-stage object detection network frameworks include SSD [8] and YOLO series [11–13]. The YOLO network unified the feature extraction, object classification, and object bounding box regression into a complete convolutional neural network, simplifying the object detection process and making it more suitable for mobile devices.

However, due to the high computational complexity and excessive parameter count, deployment of convolutional neural networks (CNNs) on embedded devices still faces challenges. Recently, various hardware accelerators have been proposed, such as UNPU [5], Eyeriss [3], and TPU [4], which are designed for general neural network operations (i.e., convolutions). Some other researchers make efforts to the deployment of neural networks on FPGAs with low-bitwidth or sparse representations to reduce the excessive parameter count [10,14]. However, the post-processing stage of object detection algorithms, which involves removing redundant boxes and generating bounding boxes, has not been previously designed with hardware acceleration in mind.



(a)



(b)

Fig. 1. (a) The process of non maximum suppression; (b) The computation latency of NMS in the post-processing stage

Most CNN hardware accelerators implement the post-processing stage on the CPU, where processing speed is slow, making it a bottleneck in achieving fast object detection. Non-maximum suppression (NMS) is a common algorithm for the post-processing stage of object detection, aimed at removing redundant boxes belonging to the same object, as shown in Fig. 1(a). The standard NMS algorithm is a greedy algorithm that requires all bounding boxes to be sorted. This property creates a strict sequential dependency between the prediction head and NMS algorithm, which is not friendly to hardware acceleration. NMS must wait for all bounding boxes to be generated before processing them, resulting in significant delay overhead throughout the entire system. The computation latency of NMS in the post-processing stage is shown in Fig. 1(b).

To address the aforementioned challenges in achieving high-speed object detection on edge devices, we propose a novel NMS algorithm to eliminate the strict sequential dependency between the NMS algorithm and the prediction head, while achieving minimal functionality degradation. Additionally, We exploit the property that the output of the convolution layer of a neural network is a fixed-point number with a fixed range after quantization. This allows us to simplify the computational complexity of complex functions in the post-processing process. We propose a configurable post-processing hardware acceleration structure that can be deployed on datasets with varying numbers of classes in a pipelined manner. In summary, our contributions are as follows:

- We proposed a novel NMS algorithm that is suitable for hardware pipeline implementation eliminating the strict sequence dependency. The hardware-friendly NMS algorithm exhibited a negligible performance decrease of 0.27% on the VOC2007 dataset and 0.03% on SAR ship images. This suggests that the algorithm is well-suited for hardware acceleration, while maintaining high accuracy.
- Accordingly, we developed a low-latency hardware accelerator design to implement the improved NMS algorithm on FPGA device with the flexible capability of multi-class and single-class object detection.
- Finally, we use a fixed-point implementation of the sigmoid function to simplify the computational complexity in the post-processing process. This is a more accurate approximation of the sigmoid function than the Taylor series expansion implementation, and it can be implemented efficiently in hardware.

2 Background and Related Work

2.1 The Network Architecture of Yolov2

Object detectors mainly include one-stage and two-stage detectors. One-stage detectors directly perform regression and classification on the input image, outputting the position, size, and category of the target. In contrast, two-stage detectors first generate candidate boxes and then perform regression and classification on each candidate box, outputting the final target. Common one-stage object detectors include SSD [8] and YOLOv2 [12]. In this work, we prefer one-stage object detectors, especially YOLOv2, because of its better real-time performance. The overall architecture of YOLOv2 is shown in Fig. 2.

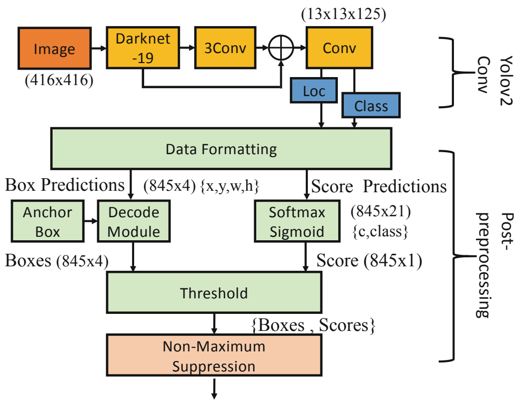


Fig. 2. YOLOv2 Structure Overview

As shown in Fig. 2, after the input image has been extracted features by Darknet-19, the prediction head outputs bounding box location information relative to anchor boxes. Anchor boxes are idealized boundaries that are predefined in different sizes and aspect ratios during model training for bounding objects in the image. The purpose of using anchor boxes is to improve the efficiency and accuracy of object detection. When the input image resolution is 416×416, after 32-times downsampling, YOLOv2 will divides the input image into 13×13 grids, and each grid has 5 anchor boxes. Therefore, a total of 845 candidate box information is generated.

The output feature map ($13 \times 13 \times 125$) generated after passing through darknet-19 for feature extraction undergoes data formatting operations (e.g. reshapes, divisions, convert fixpoint to float) to produce a list of 845 boxes (each described by $\{x, y, w, h\}$) and score information (each described by $\{c, cls\}$) for downstream processing. These generated box information are then passed through a decode module and matched with predefined anchor boxes to produce a series of predicted boxes, as shown in Eq. 1. Moreover, sigmoid and softmax functions are applied to the score list to generate prediction scores for 20 classes, as shown in Eq. 2. The resulting lists (containing boxes and scores) are passed through a threshold function, such that only boxes with scores higher than a predefined threshold are processed by NMS.

$$\begin{aligned} b_x &= \delta(t_x) + c_x \\ b_y &= \delta(t_y) + c_y \\ b_w &= P_w e^{t_w} \\ b_h &= P_h e^{t_h} \end{aligned} \tag{1}$$

where b_x, b_y, b_w, b_h are the center and width and height of the predicted bounding box. c_x, c_y are the distances from the top left corner of the current grid to the top left corner of the image. P_w, P_h are the width and height of the anchor. δ is the sigmoid function. tx, ty, tw, th are the parameters to be learned, which are used to predict the center and width and height of the bounding box respectively. Here they refer to the outputs of the convolutional layer.

$$Sigmoid = \frac{1}{1+e^{-x}}, \quad Softmax = \frac{exp(S_i)}{\sum_{i=1}^{N} exp(S_i)} \tag{2}$$

2.2 Non-maximum Suppression

The baseline NMS algorithm implemented in the object detection system based on YOLOv2 is shown in Algorithm 1. The NMS algorithm is used to filter out redundant and overlapping boxes and select the ones with the highest confidence scores. The algorithm takes as input a list of boxes and scores and outputs a list of final detections. The algorithm works as follows: (1) For each of the 21 classes, sorting the boxes by their scores of this class in descending order by using the argSort function. (2) Initialize an empty list for selected boxes. (3) For each class, while the list of candidate boxes is not empty, pop the box with the highest score from the candidate list and append it to the selected boxes. For each remaining box in the candidate boxes, calculate the intersection over union (IoU) between the popped box and the current box. If IoU is greater than a predefined threshold, remove the current box from the list. (4) Repeat step 3 until there are no more boxes left.

Algorithm 1. Baseline NMS algorithm

```
1: procedure NON-MAXIMUM SUPPRESSION
2:     $D \leftarrow$ sorted list of bounding boxes and their scores
3:     $S \leftarrow$ empty set of selected bounding boxes
4:     while $D \neq \varnothing$ do
5:         $B \leftarrow$ bounding box with highest score in $D$
6:         add $B$ to $S$
7:         for $B' \in D$ do
8:             if $IoU(B, B') > threshold$ then
9:                 remove $B'$ from $D$
10:            end if
11:        end for
12:    end while
13:    return $S$
14: end procedure
```

2.3 Bottleneck Analysis

Implement of Complex Function. Complex functions such as exponential functions consume a large amount of DSP resources when implemented on hardware circuits [6]. If deployed on edge embedded devices, a more resource-optimized deployment method needs to be considered. In the post-processing stage of calculating coordinates and scores, there will be a large number of calculations involving complex functions composed of exponential functions (such as softmax function, sigmoid function). We use the characteristics of quantized networks to design simplified calculation methods for corresponding functions according to their mapping relationships.

Limitations of NMS Implement. Firstly, the sorting operation at the beginning of the baseline NMS algorithm poses a challenge for hardware deployment. The sorting operation brings extra time and space overhead to the hardware. The sorting operation requires waiting for all the boxes to be generated, which causes a considerable time delay. Moreover, the process of removing redundant boxes by NMS has a strict sequential dependency, and the hardware cannot execute it in a pipelined manner, which reduces the throughput of the entire processing process. In addition, waiting for all the boxes to be generated requires more intermediate storage space.

2.4 FPGA-Based Object Detection System.

Although most previous works have deployed object detection systems on FPGA, most of them did not implement the post-processing stage on FPGA, but rather executed them on CPU [2,9]. A few works proposed hardware acceleration for various modules in the post-processing stage (including the NMS module), but these works did not have corresponding quantization schemes for the post-processing stage after low-bit quantization. Most convolutional outputs had to be dequantized and calculated in floating-point format in function [1,7,15].

The authors of [13] built an FPGA-based object detection system on YOLO. They deployed an NMS module with bubble sorting on the FPGA, which leveraged the parallelization and reuse of IOU computation units to enhance the computational efficiency of the NMS algorithm. The final latency of the two modules working together on the Xilinx Viertex-7 FPGA was 680us. The authors of [15] implemented an end-to-end object detection system on the FPGA (including the post-processing stage) with extremely low latency and high throughput. The NMS module only introduced a negligible delay (only 0.13us) on the Xlinx Stratix 10 GX2800 FPGA. Compared to previous work, they were the first eliminated the sorting constraint in the NMS stage, and validated their approach on the COCO dataset with a mean average precision (mAP) of 22.5%. However, their NMS module consumed a large amount of resources (the post-processing module consumed 695 DSP resources), which posed a great challenge for deployment on resource-constrained embedded edge devices.

3 Proposed Design

3.1 Hardware-Friendly NMS Algorithm

The flow of the proposed hardware-friendly NMS algorithm is depicted in Algorithm 2.

Sorting are often essential in the Baseline NMS algorithm, but due to their high iteration and the limit of waiting for all candidate boxes to be generated, deploying them on hardware can cause significant delays. Unlike Baseline NMS,

Algorithm 2. Novel hardware-friendly NMS algorithm

```
Input: $scores$,$boxes$,$IOU_{thr}$
Output: $detected\_objects$
 1: Instantiate $selected\_boxes$
 2: for each $box$ in $boxes$ do
 3:     $box\_inserted = Ture$;
 4:     $box\_deleted = False$;
 5:     $box\_replaced = False$;
 6:     $box\_suppressed = False$;
 7:     for each $box$ in $sboxes$ do
 8:         $IOU$ = Calculate_IOU($box$,$sbox$)
 9:         if same_class($box$,$sbox$) & $IOU > IOU_{thr}$ then
10:             $box\_inserted = False$;
11:             if $box.score > sbox.score$ then
12:                 $box\_replaced = Ture$;
13:                 Remember the sbox index
14:                 if !$box\_replaced$ then
15:                     $box\_deleted = Ture$;
16:                     Remember the sbox index
17:                 end if
18:             else
19:                 $box\_suppressed = Ture$;
20:             end if
21:         end if
22:     end for
23:     replaceIf($box\_suppressed$,$box\_replaced$,$boxes$,$sbox$);
24:     deleteIf($box\_suppressed$,$box\_deleted$,$boxes$,$sbox$);
25: end for
26: $detected\_objects = selected\_boxes$;

27: replaceIf($flag$,$boxes$,$sbox$)is
28: if !$box\_suppressed$ & $box\_replaced$ then
29:     $sbox[r\_index]$.replaceWith($box$)
30: end if

31: deleteIf($flag$,$boxes$,$sbox$)is
32: if !$box\_suppressed$ & $box\_deleted$ then
33:     delete $sbox[d\_index]$
34: end if
```

which sorts all candidate boxes at the beginning and requires boxes to be input in descending order of score, our algorithm does not require candidate boxes to be sorted by score. Our algorithm initializes a selected box list as empty at the beginning. For each input candidate box, it is first declared as the candidate object to be inserted (line 3), and then the IOU calculation operation and score comparison are performed between the input candidate box and all elements in the selected box list one by one.

The goal of NMS is to search for local maximum values and suppress non-maximum values. However, algorithms with sequential structures are not easy to search for local maximum values. The same data input in different orders may result in different outcomes. To solve this problem and make the results of Novel NMS more accurate, we have imposed some constraints on the insertion and replacement of new candidate boxes. When the result of IOU calculation between candidate box and select box is greater than the threshold, we compare their scores (line 7–11). If the candidate box is better, we mark it (it may replace the selected box later, line 12). We still need to compare the candidate boxes with the remaining part of the selected box list to remove the boxes that were previously selected from the same category with lower prediction scores and IOUs higher than the threshold. We mark it when the predicted score of the input candidate box is lower than that of the selected box and IOU is higher than the threshold (not allowing it to replace any candidate box, line 19). We perform replacement and deletion operations only after comparing all elements in the selected box list (line 23–24, 27–34).

Although our algorithm does not involve sorting scores, it achieves consistent functionality with baseline algorithms and ensures consistent results even under conditions of different order inputs. Sorting is implicitly performed because we replace, delete, insert and suppress selected boxes based on their scores and calculated IOUs.

3.2 Post-processing Implementation Overview

The proposed architecture of the Post-Processing Accelerator is presented in Fig. 3. The convolution layers of Yolov2 predicts 845 box coordinates as described in Sect. 2, each with score predictions for the 20 classes and 1 Confidence. These outputs are processed in dataflow by the Decode module and Softmax module to consolidate the boxes for the final detection.

3.3 Post-processing Implementation Details

In order to effectively complete NMS calculations, we have proposed some circuit improvements in the following sections.

Approximated Method of Exponential Arithmetic. Exponential and sigmoid functions are very challenging for hardware implementation, but they are essential for Softmax computation. Since our quantization algorithm can quantize the post-processing input to -128 to 127, we can design an approximate

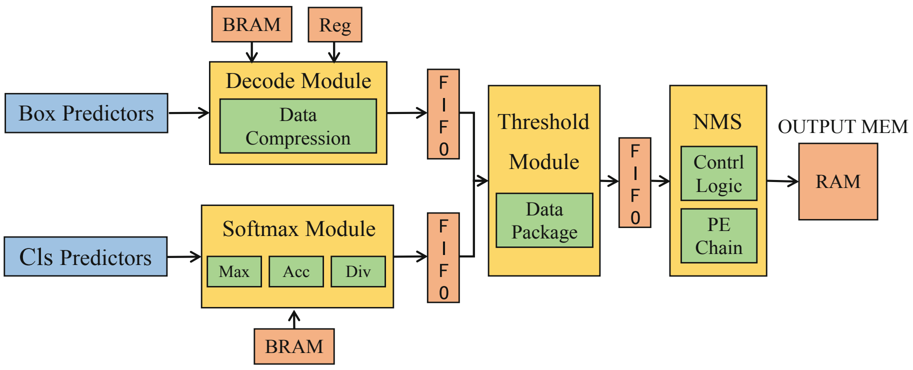


Fig. 3. Architecture of Post-Processing Accelerator

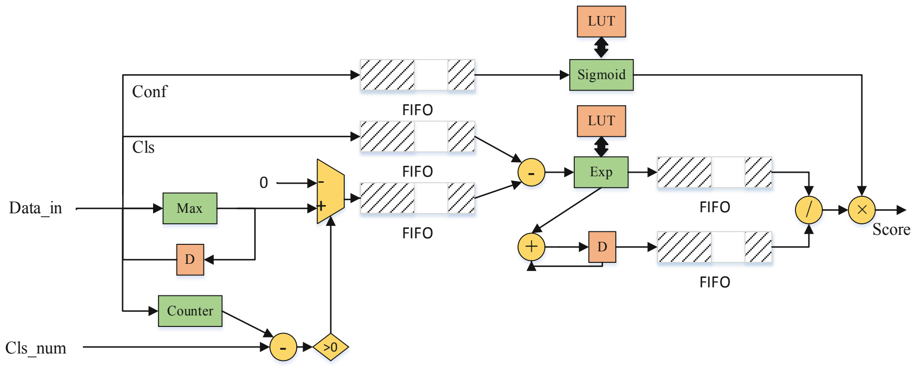


Fig. 4. The Process of Softmax Module

exponential function circuit easily. As shown in Fig. 4, we use a lookup table to store the dequantized function values in Block RAM (BRAM). In the post-processing stage, we need to compute exponential and sigmoid functions for Score calculation. The dequantized values of these functions are stored in different BRAMs for parallel access.

Moreover, we exploit the property of Softmax computation to compress some of the dequantized outputs. Because Softmax involves normalization, the normalized output ranges from negative infinity to zero, and the corresponding exponential output ranges from zero to one. This can be easily quantized with 8-bit fixed-point numbers. By doing so, we further reduce the storage space consumption.

Overall, we only need 3 M32K BRAMs to store all the dequantized results. This method introduces very small errors compared with function approximation methods such as Taylor series expansion, and most importantly, it does not consume any DSP resources. As shown in Table 1, we compare the resource utilization with different method of sigmoid function implement.

Table 1. Comparation of implementations of sigmoid function approximations

	Logic	Register	Memory	DSP	M32K BRAMs
[15]	67	134	2304	2	–
Ours	38	23	–	0	1.5

Data and Space Compression. The object detection algorithm produces a large number of candidate boxes, which poses a great challenge for hardware storage. To address this issue, we apply data compression and spatial compression techniques to reduce the storage space demand.

For data compression, the convolution layer outputs are converted back to floating-point numbers after dequantization in the Decode and Softmax modules. Floating-point numbers offer higher accuracy for computation, but they also require more storage space. A single-precision floating-point number needs 4 bytes of storage space. To quantize the output coordinate data(i.e., $\{x_{min}, y_{min}, x_{max}, y_{max}\}$, which are fed into the NMS module), we scale them to between 0 and 1. We experiment with different quantization bit-widths and their effects on accuracy. We finally choose 8-bit fractional quantization.

For spatial compression, the NMS module computes IOU for each category of boxes and obtains the detection results for each category. Normally, data from different categories are stored separately, which avoids interference but also leads to redundancy. We store data from different categories in a contiguous memory space to save storage consumption. We use RAM to store data contiguously and use category tags to mark them in memory. We also store some control logic for NMS in the same memory space and pack all information related to a box together. Figure 7 illustrates how data are stored when there are two categories (Fig. 5).

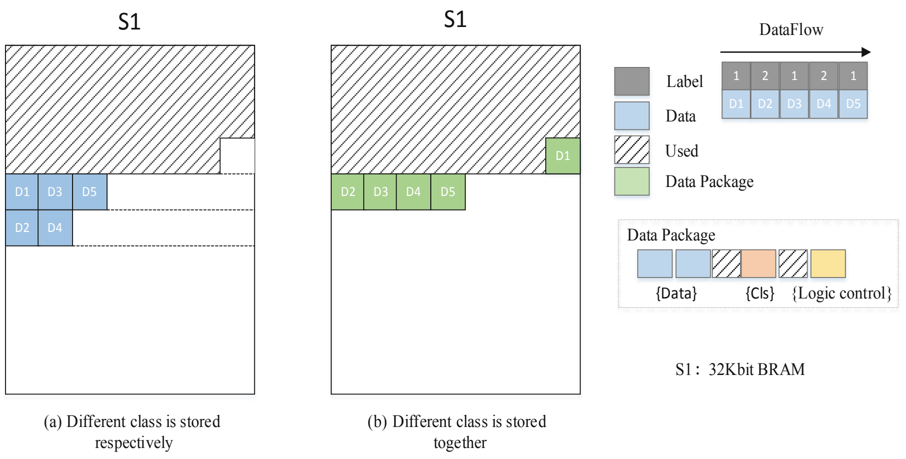


Fig. 5. The Process of Data Package

NMS Hardware Implementation. As shown in Fig. 6, the post-processing module is composed of a processing element (PE) and a control logic block. The PE contains a buffer that stores a packed data of a single bounding box (including coordinates, score, class and control logic) and an IOU computation unit. The PE traverses the select boxes in the buffer and performs a series of comparisons and IOU calculations with the new input box. The control logic block determines whether to replace, ignore or delete the local or incoming bounding box based on the results. Each bounding box in the PE has a suppression flag to prevent erroneous deletion in the sequential structure, which is explained in detail previously. The NMS module does not wait for all bounding boxes to be generated by the convolutional layer and Decode/Softmax module of Yolov2. Instead, it uses a first-in first-out (FIFO) queue to buffer the input.

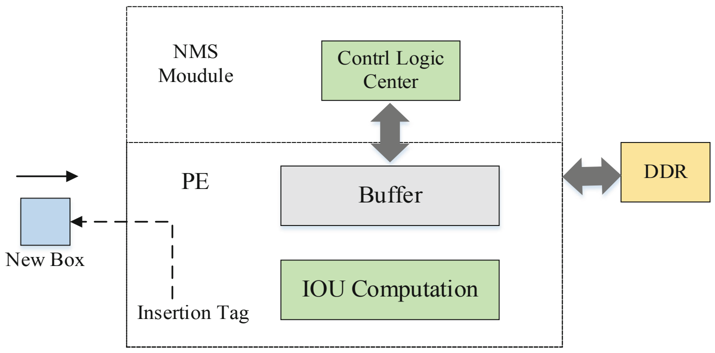


Fig. 6. NMS Hardware Implementation.

4 Experiment Result

We implemented and deployed neural network accelerators and post-processing accelerators on Xilinx Alveo U50 FPGA attached as a PCIe accelerator card to an AMD RyZenq 5950X server with 16 dual-threaded cores, a working frequency of 3.4 GHz and 64 GB RAM. In this setup, the host CPU sends input images to the FPGA accelerator card and receives the returned output prediction via the PCIe link. Similarly, the post-processing acceleration module is part of the entire system and outputs the corresponding results to the host CPU via the PCIe link after processing the convolution layer output results. The actual execution time of the post-processing module is measured using Xilinx Runtime Library (XRT) analysis API. The resource consumption of the post-processing module on the entire development board was also calculated using Vivado2022 tools.

We tested our post-processing accelerator on multi-class datasets and single-class datasets separately. We validated the system's functionality on 4952 validation images from VOC2007 dataset and 232 validation images from remote

Table 2. Comparison with general-purpose hardware

	Frequency (MHz)	Accuracy	Latency (us)	Dataset
AMD Ryzenq 5950X	3400	75.09%	555	VOC
		90.41%	141	SAR
Our Work	300	74.82%	**283**	VOC
		90.38%	**78**	SAR

Table 3. Comparison of our work to prior FPGA-based postprocessing accelerator

	LUTRAM	DSPs	BRAMs	FF	Latency	FPGA Device	Frequency (Mhz)	Boxes number	Best boxes	Multi Class
F. Liang et al. [7]	2890	36	203	11842	680 us	Viertex-7 485T	100	1960	3	No
H. Zhang et al. [15]	714	22	32	11139	32 us	Zynq-7 VC 706	100	3000	5	No
A. Anupreetham et al. [1]	–	695	425	86,704	0.13 us	Stratix 10 GX2800	350	1917	–	Yes
This work	1364	77	24.5	13929	283 us	Alveo U50	300	845	128	Yes
	1181	77	11	16071	78 us	Alveo U50	300	845	5	No

sensing ship dataset. This proves that our architecture is versatile and suitable for not only single-class object detection tasks.

Table 2 present a performance comparison of our post-processing accelerator with general hardware on the VOC2007 dataset and SAR image ship dataset. The post-processing accelerator uses a hardware-friendly NMS algorithm to significantly reduce latency while only incurring an acceptable loss of accuracy. Specifically, the mAP decreases by 0.17% on the VOC2007 dataset and only 0.03% on the SAR image ship dataset. Our results demonstrate that the post-processing accelerator is a promising approach for FPGA neural network acceleration. The accelerator achieves significant latency reductions while maintaining high accuracy, making it a valuable tool for real-time applications.

Table 3 compares our work with other object detectors that deploy post-processing on FPGAs. Some of the works in the Table 3 are implemented on different feature extraction networks and object detectors, but this does not affect the comparison and evaluation of their post-processing modules. The most relevant comparison is against the work of [1], which is implemented on the same generation of Intel FPGA and is the only object detection accelerator that verifies processing acceleration modules on multi-class datasets. Compared with its high throughput and low latency implementation, our accelerator has lower resource utilization. According to the user guide provided by Xilinx and Intel, we converted the logic resources of different implementations on the FPGA into LUT resources. The work of [1] used 86704 Adaptive Logic Module (ALM) resources in the post-processing module, each containing 2 LUT resources and 1 register resource. We converted this and compared it to the 13929 register

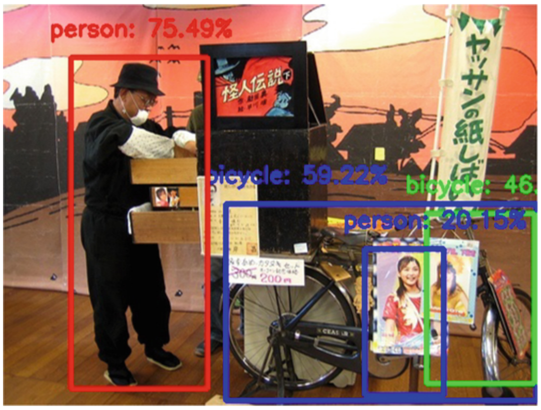


Fig. 7. The detection examples of Yolov2

resources and 12837 LUT resources we used in our work, we consumed 13.5× lower LUT resources and 6× lower register resources.

We also deployed a single-class object detection task on FPGA based on the post-processing architecture proposed in this paper using the VITIS HLS toolchain. In order to ensure its versatility, we still retain the calculation process that can be omitted, because this will affect its normal operation under multi-class object detection tasks. The post-processing accelerator implemented by [15] has lower resource utilization on DSP and LUTRAM. Nevertheless, our solution ensures that the post-processing accelerator architecture we proposed is feasible for different object detection tasks, and our solution still achieves lower resource utilization and delay at higher detection accuracy.

5 Conclusion

In this paper, we propose a post-processing accelerator for FPGA-based YOLOv2 object detection. We propose a novel hardware-friendly NMS algorithm for FPGA accelerator design, which alleviates the performance bottleneck of NMS and deploys the corresponding hardware architecture. We validate our algorithm on the VOC2007 dataset. In addition, we propose an approximate exponential function circuit to simplify the computational complexity in the post-processing stage. Our work achieves considerable detection accuracy while using much fewer resources. This makes it possible to deploy object detection on edge devices with more limited resources.

References

1. Anupreetham, A., et al.: End-to-end FPGA-based object detection using pipelined CNN and non-maximum suppression. In: 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), pp. 76–82 (2021). https://doi.org/10.1109/FPL53798.2021.00021

2. Cai, L., Dong, F., Chen, K., Yu, K., Qu, W., Jiang, J.: An FPGA based heterogeneous accelerator for single shot multibox detector (SSD). In: 2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT), pp. 1–3. IEEE (2020)
3. Chen, Y.H., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J. Solid-State Circuits **52**(1), 127–138 (2016)
4. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 1–12 (2017)
5. Lee, J., Kim, C., Kang, S., Shin, D., Kim, S., Yoo, H.J.: UNPU: a 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In: 2018 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 218–220. IEEE (2018)
6. Li, Z., Zhang, Y., Sui, B., Xing, Z., Wang, Q.: FPGA implementation for the sigmoid with piecewise linear fitting method based on curvature analysis. Electronics **11**(9), 1365 (2022)
7. Liang, F., Yang, S., Mai, T., Yang, Y.: The design of objects bounding boxes non-maximum suppression and visualization module based on FPGA. In: 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), pp. 1–5 (2018). https://doi.org/10.1109/ICDSP.2018.8631668
8. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
9. Ma, Y., Zheng, T., Cao, Y., Vrudhula, S., Seo, J.: Algorithm-hardware co-design of single shot detector for fast object detection on FPGAs. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8. IEEE (2018)
10. Mo, R., Xu, K., Liu, L., Liu, L., Wang, D.: Adaptive linear unit for accurate binary neural networks. In: 2022 16th IEEE International Conference on Signal Processing (ICSP), vol. 1, pp. 223–228 (2022). https://doi.org/10.1109/ICSP56322.2022.9965306
11. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788 (2016)
12. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7263–7271 (2017)
13. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
14. Wang, Z., Xu, K., Wu, S., Liu, L., Liu, L., Wang, D.: Sparse-YOLO: hardware/software co-design of an FPGA accelerator for YOLOv2. IEEE Access **8**, 116569–116585 (2020). https://doi.org/10.1109/ACCESS.2020.3004198
15. Zhang, H., Wu, W., Ma, Y., Wang, Z.: Efficient hardware post processing of anchor-based object detection on FPGA. In: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 580–585 (2020). https://doi.org/10.1109/ISVLSI49217.2020.00089
16. Zou, Z., Chen, K., Shi, Z., Guo, Y., Ye, J.: Object detection in 20 years: a survey. Proc. IEEE **111**(3), 257–276 (2023). https://doi.org/10.1109/JPROC.2023.3238524