A Multi-mode Convolution Coprocessor Based on RISC-V Instruction Set Architecture

Wenqiang Gong
College of Electronic and Information Engineering
Nanjing University of Aeronautics and Astronautics
Nanjing 211106, China
eswen@nuaa.edu.cn

Fang Zhou
College of Electronic and Information Engineering
Nanjing University of Aeronautics and Astronautics
Nanjing 211106, China
zfnuaa@nuaa.edu.cn

Fen Ge*
College of Electronic and Information Engineering
Nanjing University of Aeronautics and Astronautics
Nanjing 211106, China
gefen@nuaa.edu.cn

2023 IEEE 15th International Conference on ASIC (ASICON) | 979-8-3503-1298-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASICON58565.2023.10396531

***Abstract*—Different modes of convolution, as the main operations of convolutional neural networks, determine the efficiency of the entire network. This paper proposes a multi-mode convolution coprocessor based on the RISC-V instruction set in order to accelerate convolution operation. In the coprocessor, a round-based convolution unit and dataflows for different convolution modes are designed to reduce memory access frequency. A custom instruction subset based on RISC-V ISA is designed, and the common convolution algorithm is implemented on the coprocessor. Finally, performance analysis and resource consumption evaluation of the coprocessor are completed on a Xilinx FPGA. Implementing different modes of convolution based on the designed instruction subset takes fewer cycles than using the standard instruction set. And the acceleration ratio of pointwise convolution is 8.74 times that of the standard instruction set.**

Keywords—convolutional neural network(CNN), coprocessor, multi-mode convolution, RISC-V

I. INTRODUCTION

Convolutional neural network (CNN) algorithm is often applied to face recognition, object detection and other scenes requiring high real-time and low power consumption on embedded platforms. However, such platforms usually cannot be equipped with general-purpose programmable accelerators such as GPUs due to the limited resources. Therefore, a lightweight network is more suitable for them[1]. Some lightweight networks, such as MobileNet, in order to deepen the network depth without increasing the number of network parameters and the amount of operations, will use depthwise separable convolution instead of standard convolution. So far, there have been many accelerators designed for these networks[2-4], and balancing performance and resource consumption brings new challenges to the design of CNN accelerators.

Among the studies on CNN accelerators applied to edge device, each one focuses on different aspects. Some focus on a specific network architecture, e.g. a hardware acceleration architecture based on GoogLeNet is proposed in [5], which can achieve high throughput by completely implementing the network. In [6] , an architecture is designed to accelerate depthwise separable convolutions, and optimized according to the characteristics of the network. These architectures have a high acceleration ratio, but are applicable to fewer network structures. Others pursue adapting to different networks, providing a general solution through parameterized and configurable designs. In [7-8], two reconfigurable convolution accelerator are proposed, both supporting online configuration. The characteristic of these accelerators is that it can accelerate various networks, but the control of the hardware operation architecture is so complicated that it will consume a lot of resources.

In order to solve the control problems of accelerators, an architecture with a RISC-V controller called PicoRV32 is designed in [9]. The PicoRV32 core completes its dataflow control, reducing the resource consumption. Reference [10] proposed a coprocessor based on the RISC-V to accelerate CNN algorithms, and supports reconfiguring parameters for convolution. But due to the fixed dataflow of its processing elements, it cannot support depthwise separable convolution. Moreover, the configuration instructions are too fussy to implement the application on edge device.

To address the issues raised in [10], this paper proposes a convolution coprocessor for both standard convolution and depthwise separable convolution by configurable parameters. Firstly, the convolution unit based on round operations is proposed, and two dataflows for different convolution modes is designed. Then, a concise control instruction subset is designed for the accelerator by using the RISC-V extended instruction space. Finally, performance analysis and resource consumption evaluation of the designed coprocessor are completed based on a Xilinx FPGA.

The rest of this paper is organized as follows: Section II describes the background of the coprocessor. Section III introduces the architecture and dataflow design of the coprocessor, and elaborates on the instruction subset design of it. Section IV discusses the experiment and resource analysis. The conclusion was drawn in Section 5.

II. BACKGROUD

A. Different Convolution Modes

The coprocessor proposed in this paper supports two of the most widely used convolutional modes, namely standard convolution and depthwise separable convolution.

Standard convolution (SC) is the most commonly used convolution mode among convolution layers of various CNN algorithms. The convolution kernel convolves with each input channel and accumulates the results of each channel to obtain the output features.

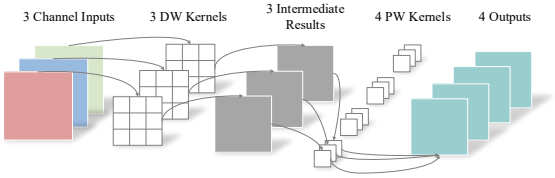

Fig. 1. Depthwise seperable convolution

This work was supported by the National Key R&D Program of China (No. 2022YFD2000700).

Depthwise separable convolution (DSC) is commonly used in lightweight networks to reduce single layer parameters and improve computational efficiency. As shown in Fig. 1, its key idea is to break up a standard convolution process into depthwise convolution (DW) and pointwise convolution (PW). In a SC, a single kernel convolves and accumulates with each channel. While in the case of DSC, DW is performed by convolving each channel of the input feature map with a single kernel, and then arranges the results of each channel independently to obtain the intermediate feature map. That is to say, DW does not change the number of channels of the input feature map. PW is actually a standard convolution with a kernel size of 1×1, which plays two roles in DSC: one is to freely change the number of output channels, and the other is to fuse the feature belonging to each output channel of DW.

B. *RISC-V Instruction Set*

RISC-V is an open-source instruction set that summarizes the advantages of traditional instruction sets, ensuring its simplicity and efficiency. RISC-V consists of three basic instruction sets and six extended instruction sets. In practical applications, users can freely tailor according to the needs of the scene[11]. RISC-V architecture reserves instruction coding space for special domain architectures in the processor design. The custom instruction spaces reserved in the RISC-V architecture are shown in Table I.

TABLE I. RISC-V INSTRUCTION RESERVED CODING SPACE

<table>
<tr><th rowspan="2">opcode [6:5]</th><th colspan="5">opcode[4:2]</th></tr>
<tr><th>000~001</th><th>010</th><th>011~101</th><th>110</th><th>111</th></tr>
<tr><td>00</td><td rowspan="4">Others</td><td>custom-0</td><td rowspan="4">Others</td><td rowspan="2">Others</td><td rowspan="4">Others</td></tr>
<tr><td>01</td><td>custom-1</td></tr>
<tr><td>10</td><td rowspan="2">Others</td><td rowspan="2">custom-2</td></tr>
<tr><td>11</td><td>custom-3</td></tr>
</table>

The instruction space with the opcode[6:2] 00010 and 01010, are reserved for custom extended instructions and will not be used for extension of standard instructions. The instruction space with opcode[6:2] 10110 and 11110 are reserved for the rv128 in the future and will not be occupied as well. Therefore, the four instruction spaces mentioned above can be used to extend custom instructions.

The E203 core is a 32-bit RISC-V-based processor core with a two-stage variable-length pipeline, and its performance is close to that of ARM Cortex-M0+. E203 has designed NICE coprocessor interface for user-friendly extension. The coprocessor extension instructions follow the custom extension instruction standard of RISC-V. In this paper, the custom-0 space is used to extend custom instructions through NICE interface.

III. HARDWARE DESIGN

A. *Architecture Overview*

The coprocessor is located at the instruction execution stage of the main processor through the NICE coprocessor interface of the E203. Fig. 2 shows the overall architecture of the designed coprocessor, which consists of Decoder, Data Interactor, Multi-mode Controller, Feature Rearranger, Weight Loader, Conv Unit, Post-processing Module and RAM Bank.

When encountering a custom instruction, the coprocessor first decodes the instructions dispatched by the main processor, and then distributes them to Data Interactor or Multi-Mode Controller by category. After receiving the read and store instruction, Data Interactor requests the main processor to occupy the bus, and starts to transfer the feature map and kernel map to the coprocessor. After Multi-mode Controller receives the configuration instruction, it completes the setting of the relevant parameter registers within one cycle.

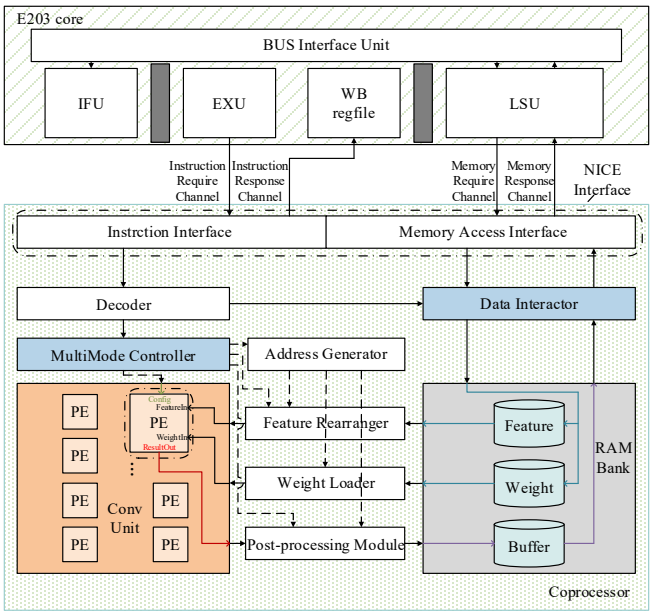

Fig. 2. Hardware structure of coprocessor

When a convolution operation begins, Address Generator sequentially transmits the access address to Feature Rearranger and Weight Loader according to the configured parameters. The feature map will be rearranged and stored in FIFO under the control of Multi-mode Controller, and the weight data will be directly transmitted to the processing element (PE) array after being arranged in sequence. When the weight data in the PEs is in place, Feature Rearranger sequentially pops data to the PEs. PE completes the corresponding accumulation according to the parameters setting by Multi-mode Controller, and sends the results to Post-processing Module in the order of PE labels. After post-processing (including relu, quantization, et al.), the results are stored in the output buffer at the address given by Address Generator. After the work is completed, Data Interactor sends the output feature map back to the main processor memory under instructions.

B. *A Round-based Convolution Unit*

A major characteristic of convolution is that the parameters involved in the operation is few, but the amount of calculation is large. Optimizing convolution operations can improve resource utilization and performance of the coprocessor. A convolution operation of a two-dimensional $A \times A$ matrix X, and a $k \times k$-sized convolution kernel W with a sliding window stride of s can be expressed as:

$$Y(p,q) = \sum_{i=-(k-s)/2}^{(k-s)/2} \sum_{j=-(k-s)/2}^{(k-s)/2} X(p+i, q+j) \cdot W(i,j) \qquad (1)$$

It can be seen from (1) that the calculation amount of a convolution operation is $k^2((A-k)/s+1)^2$, and the corresponding amount of data to be loaded is $2k^2((A-k)/s+1)^2$. Both standard convolution and depthwise separable convolution essentially use two-dimensional convolution as the smallest operation unit. The convolution kernel is reused

multiple times. The repeated loading of data brings additional resource consumption to the coprocessor，thus affecting the performance of the entire system. To improve the utilization of data retrieved once, a round-based convolution unit is designed from the perspective of data reuse.

Taking the convolution with a 3×3 convolution kernel as an example, the feature map of each layer is divided into 16 parts of size $i \times i$ according to the sliding window stride. Each part is calculated with the kernel in exactly the same way. As shown in Fig. 3, each PE calculates a set of data from each part within 16 cycles, which is called a round of operation. In one round of operation, the position of the feature map data relative to the convolution kernel is the same. The convolution kernel data does not need to be updated within 16 cycles, which reduces the repeated loading of data.

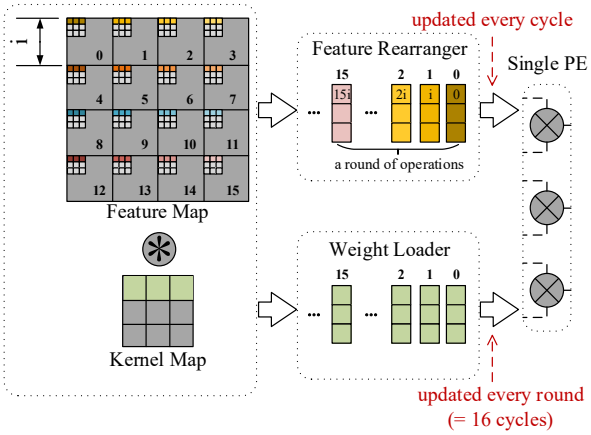

Fig. 3. Round-based operation

According to the previous analysis, the amount of data to be loaded is $2k^2((A-k)/s+1)^2$，while the optimized amount of data to be loaded is $k^2((A-k)/s+1)^2$-$k^2((A-k)/s+1)^2/16$. The amount of data loaded for a round of two-dimensional convolution operation is reduced by 46.88%.

C. Dataflows of Two Convolution Modes

Two processes of data loading for SC, DW and PW of DSC are shown in Fig. 4. For DW, as shown in Fig. 4(a), the calculations between channels are independent of each other, and it is suitable to use the row-first data loading method. Limited by the continuity of data storage, in each clock cycle, we take out k points of data from the feature map RAM to participate in the operation. These data are from a single channel and belong to a same convolution window. After the accumulation calculation by the adder tree, the intermediate result will be temporarily stored in the configurable accumulator of corresponding part. After multiple rounds of operations, the intermediate result belonging to a single convolution window has been accumulated, that is, the first output feature values of each part is obtained. In DW mode, the utilization rate of PE is:

$$r_{DW} = 1 - \frac{k \cdot |k-4|}{4k \cdot \lceil k/4 \rceil} \tag{2}$$

For SC and PW, the intermediate results of operations between channels need to be accumulate, which is different from DW. To improve PE utilization, it is appropriate to adopt a channel-first data loading method. As shown in Fig. 4(b), in each clock cycle, 4 points of data are preferentially taken out according to the channel dimension to participate in the operation. After each round of operation, as shown in clk_{16}, the next 4 points of data are sequentially taken out to participate in the operation. Until the data in a convolution window is fully loaded, the configurable accumulators of each part also obtain a set of output feature values.

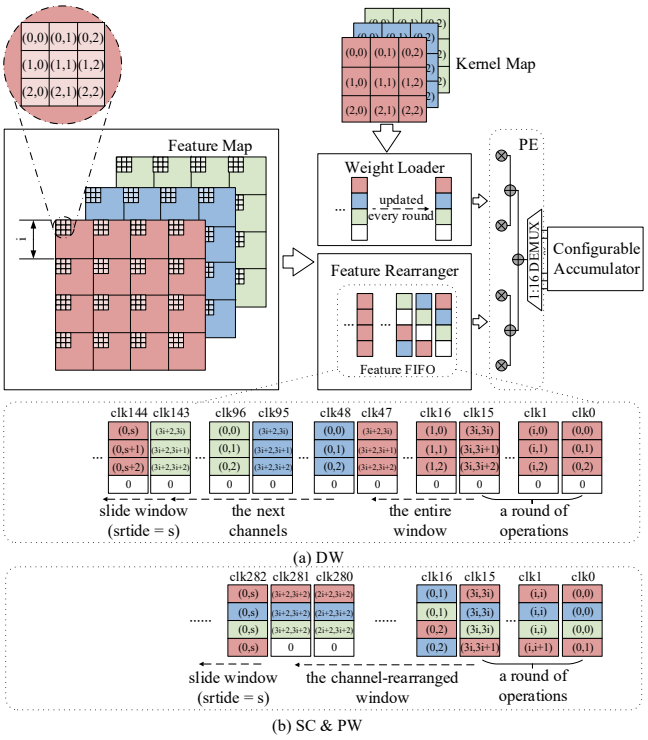

Fig. 4. Dataflows of twoconvolution modes

For SC or PW with C channels, the PE utilization is:

$$r_{SC,PW} = \frac{k^2 C}{4 \cdot \lceil k^2 C / 4 \rceil} \tag{3}$$

Compared with the row-first data loading method, the utilization rate of PE is increased by:

$$\eta = 1 - \frac{r_{SC,PW}}{r_{DW}} \tag{4}$$

Taking the 3×3×3 convolution kernel as an example, this loading method makes the PE utilization rate reach 96.4%, which reduces the idle situation of PE.

The effect of improving PE utilization is positively correlated with the number of channels C, especially when the convolution kernel size is 1×1. The PW of DSC is essentially a SC with a convolution kernel size of 1×1. The utilization of PEs can be significantly optimized by using the above data loading method. Taking the PW layer with an input size of 56×56×144 in MobileNetV1 as an example, the channel-first data loading method can make the PE utilization rate reach 100%. Compared with the row-first loading method, the utilization rate is increased by 3 times.

D. Custom Instruction Subset Design

The instructions designed for the multi-mode convolution coprocessor are shown in Table II.

There are 7 coprocessor instructions extended by using the custom-1 instruction space of opcode=0001011, which

perform two functions: configuration and operation. Configuration instructions are used to configure each module of the coprocessor, such as data loading mode, PE accumulation mode, et al. Operation instructions read and write to the coprocessor RAM, change or reset the coprocessor state.

TABLE II. CUSTOM INSTRUCTION SUBSET

instruction	fc7	rd	rs1	rs2	xd/xs1/xs2
conv.op.rst	0	-	-	-	000
conv.op.ram	1	Length	RamMode |RamAddr	BusAddr	011
conv.op.ctl	2	State	-	-	100
conv.cfg.adr	4	Part	Addr1	Addr2	011
conv.cfg.std	8	-	XStride |YStride	CStride |PStride	011
conv.cfg.mod	16	-	ConvSize |PoolSize	Mode |Dropout	011
conv.cfg.cnt	32	-	CCount	SlideCnt	011

The assembly format of coprocessor instructions is shown in Fig. 5.

```
conv.op.rst
conv.op.ram     rd rs1 rs2
conv.op.ctl     rd
conv.cfg.adr    rd rs1 rs2
conv.cfg.std       rs1 rs2
conv.cfg.mod       rs1 rs2
conv.cfg.cnt       rs1 rs2
```

Fig. 5. Assembly format of coprocessor instructions

During software programming, the use of the designed coprocessor instructions generally follows the 4 steps:

1. Reset the coprocessor state and clear the corresponding registers. Use the conv.op.rst instruction to reset it.
2. Load the input feature map and weight coefficient. Use the conv.op.ram instruction in batches to move the data from the memory to the coprocessor RAM at the corresponding address, in order to complete the data preparation before the operation.
3. Configure coprocessor parameters:
   (1) The base address of the 16 parts of the feature map need to be configured. In order to save instruction space and simplify the logic of the decoder, we use one instruction conv.cfg.adr to realize it. The data at the *rd* bits represents the label of each part, ranging from 0 to 7. The rs1 and rs2 each hold the base address of a part.
   (2) During the convolution operation and post-processing, the sliding of the window and sequential reading in a single window mean that the read address has to change accordingly. XStride and YStride represent the address offset that needs to be increased for each sliding window in the row or column dimension. CStride represents the address offset that needs to be increased when fetching data across a row within a single window. PStirde represents the stride of the pooling operation in post-processing. These stride parameters are configured by the conv.cfg.std instruction.
   (3) The working mode of the coprocessor needs to be informed, including the convolution operation mode, the size of the convolution and pooling window, and the dropout method of the feature map. These parameters are configured by the conv.cfg.mod instruction.
   (4) In a single convolution operation, the number of convolution sliding windows contained in each part is fixed. Additionally, in the SC and PW modes, the number of consecutively available data batches in the channel dimension needs to be configured. They are collectively referred to as the counter configuration, done by the conv.cfg.cnt instruction.
4. Use the conv.op.ctl instruction to change the coprocessor state. If the coprocessor is in the idle state, make it enter the work state, and start the calculation according to the configured parameters. At the end of an operation, write back the current state to the *rd*, and call the conv.op.ram instruction to retrieve the corresponding results.

Since the number of instructions is 7, the funct7 of the instruction code adopts one-hot code, which could simplify the combination logic of the decoder and optimize the decoding process.

When the main processor fetches an instruction, the mini-decoder of the IFU first judges whether the instruction is a custom extended one according to the opcode. For the custom extended instruction, the IFU determines whether to read the source operand according to the xs1 and xs2 bits in the instruction by the full-decoder. When the main processor judges that there is no data correlation between the instructions, it will fetch the operands from the rs1 and rs2, and send them with the instruction itself to the coprocessor from the instruction request channel of the NICE interface. The instructions will be further decoded in the coprocessor and dispatched to different modules according to the functions of instructions.

IV. EXPERIMENTAL RESULTS

This section completes the performance analysis and resource analysis of the designed multi-mode convolution coprocessor based on Xilinx FPGA. The FPGA model is Xilinx xc7z045ffg900-2, and the synthesis is Vivado 2021.2. The test of the coprocessor is completed in cooperation with the E203 MCU, and the performance analysis of the coprocessor is completed by RTL Simulator. To evaluate the performance of the coprocessor, SC, DW and PW of DSC are implemented in two compilations methods. One is implemented by the RISC-V standard instruction set, and the other is implemented using the instruction set of the extended coprocessor instruction. Store the instruction machine code generated by compilation into ITCM of E203 as an input stimulus, and count the operation cycle of each type of convolution in two implementations. Compared with [10], while supporting SC, it also supports the calculation of DSC. The experimental results are shown in Table III.

TABLE IV. NUMBER OF EXECUTION CYCLES OF EACH ALGORITHM

Algorithm		RV321M	[10]	Coprocessor Instruction
Standard Convolution (Feature size:4×4×4 Kernel size: 3×3×4×16)		830848	132480	123813
DSC	Depthwise Convolution (Feature size: 4×4×4 Kernel size: 3×3×4)	54661	-	10866
	Pointwise Convolution (Feature size:2×2×4 Kernel size: 1×1×4×16)	90874	-	10397

As shown in Table III, it can accelerate two convolution operations by using the designed coprocessor. And the acceleration ratio of each operation can be seen in Fig. 5.

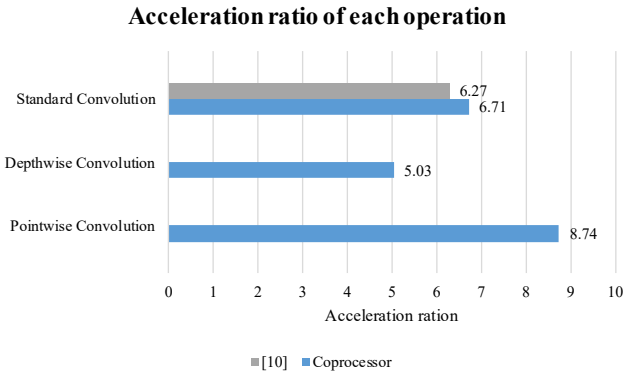

Fig. 6. Acceleration ratio of each operation

The coprocessor can obvious accelerate both convolution operations, and the performance for SC and PW is particularly obvious. On the one hand, the coprocessor implements convolution calculation through a dedicated hardware unit, while the RISC-V-based main processor implements it through a general-purpose ALU. On the other hand, the memory access frequency is reduced by using the coprocessor. In the SC and PW, the data loading method has been optimized to allow as many PEs as possible to participate in the calculation, further speeding up the processing speed of the operation. The acceleration of PW by coprocessor extended instructions is 8.74 times that of the standard instruction set.

The hardware resource consumption of each component unit of the system can be seen in Table IV. The E203 core and coprocessors account for most of the resource consumption. The coprocessor accounts for 44.1% of the total LUT resource consumption, of which 2.6% is used as DRAM for handling calculation data. In addition, the E203 core accounts for 46.4% of BRAM resource consumption for instruction and data storage, while the coprocessor accounts for 53.6% for storing feature maps, weight data and calculation results.

TABLE V. SYSTEM RESOURCE CONSUMPTION

Module	Slice LUTs	Slice Registers	BRAM
E203 core	4409	2198	32
Coprocessor	9111	8132	37
CLINT	33	132	0
PLIC	179	173	0
DEBUG	217	580	0
Periphrials	5597	6942	0
Others	1093	1305	0
Total	20639	19462	69

V. CONCLUSION

This paper proposes a general-purpose convolution coprocessor based on the RISC-V instruction set, which supports two convolution modes and allows online reconfiguration of convolution parameters. Two types of dataflows are designed to improve the utilization ratio of PEs and utilize hardware resources efficiently. The convolution unit with round operation is designed to reduce the frequency of memory access through data reuse. Moreover, concise custom instructions are extended for the coprocessor based on the NICE interface. Finally, the performance evaluation of the entire coprocessor is completed based on Xilinx FPGA. The acceleration performance of the coprocessor is evaluated by comparing the number of operation cycles of the two convolution operations implementing by the RISC-V standard instruction set and the custom instruction set. The results show that the acceleration ratios of the coprocessor instruction set for SC, DW and PW are 6.71, 5.03, and 8.74, respectively.

REFERENCES

[1] X. Qin, X. Liu and J. Han, "A CNN Hardware Accelerator Designed for YOLO Algorithm Based on RISC-V SoC," 2021 IEEE 14th International Conference on ASIC (ASICON), Kunming, China, 2021, pp. 1-4.

[2] E. Elfatimi, R. Eryigit and L. Elfatimi, "Beans Leaf Diseases Classification Using MobileNet Models," in IEEE Access, vol. 10, pp. 9471-9482, 2022.

[3] Y. Lu, K. Li and R. Ni, "Research on Lip Recognition Algorithm Based on Optimized MobileNet," 2022 4th International Conference on Advances in Computer Technology, Information Science and Communications (CTISC), Suzhou, China, 2022, pp. 1-6.

[4] T. Addabbo, A. Fort, M. Mugnaini, V. Vignoli, M. Intravaia, M. Tani, M. Bianchini, F. Scarselli and B. Corradini, "Smart Gravimetric System for Enhanced Security of Accesses to Public Places Embedding a MobileNet Neural Network Classifier," in IEEE Transactions on Instrumentation and Measurement, vol. 71, pp. 1-10, 2022.

[5] M. N. Islam, R. Shrestha and S. R. Chowdhury, "A New Hardware-Efficient VLSI-Architecture of GoogLeNet CNN-Model Based Hardware Accelerator for Edge Computing Applications," 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Nicosia, Cyprus, 2022, pp. 414-417.

[6] H. Srivastava and K. Sarawadekar, "A Depthwise Separable Convolution Architecture for CNN Accelerator," 2020 IEEE Applied Signal Processing Conference (ASPCON), Kolkata, India, 2020, pp. 1-5.

[7] Y. -G. Chen, H. -Y. Chiang, C. -W. Hsu, T. -H. Hsieh and J. -Y. Jou, "A Reconfigurable Accelerator Design for Quantized Depthwise Separable Convolutions," 2021 18th International SoC Design Conference (ISOCC), Jeju Island, Korea, Republic of, 2021, pp. 290-291.

[8] G. Boonyuu and S. Wisayataksin, "Configurable Hardware Architecture of Multidimensional Convolution Coprocessor," 2021 Second International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), Bangkok, Thailand, 2021, pp. 1-4.

[9] L. Zhang, X. Zhou and C. Guo, "A CNN Accelerator with Embedded Risc-V Controllers," 2021 China Semiconductor Technology International Conference (CSTIC), Shanghai, China, 2021, pp. 1-3

[10] N. Wu, T. Jiang, L. Zhang, F. Zhou and F. Ge, "A Reconfigurable Convolutional Neural Network-Accelerated Coprocessor Based on RISC-V Instruction Set," Electronics (Basel), vol. 9, (6), pp. 1005, 2020.

[11] Patterson D., Waterman A., The RISC-V Reader: An Open Architecture Atlas; Strawberry Canyon: Berkeley ,CA, USA, 2017.