FPGA-Based Design for Accelerating 3D Convolutional Neural Networks

Yuesong Yang[1,2], Wenjing He[1,*], Jian Hu[1], Sai Cheng[1], Wanqiu Xu[1]

[1]Key Laboratory of Quantitative Remote Sensing Information Technology, Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing, 100094, China
[2]School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, 100049, China
**Corresponding author: hewj@aircas.ac.cn*

Abstract: *Three-dimensional convolutional neural networks (3D CNNs) have been shown to be excellent for classification in human action recognition, hyperspectral image classification and many other aspects. However, 3D CNNs often contain convolution kernels of different sizes and a large number of operations, and the numerous operations make 3D CNNs consume a lot of time when executing inference on resource-constrained terminals. Aiming at these two points, this paper proposes a 3D CNN accelerator design based on loop optimization and weight reuse. The design adopts loop tiling, loop unrolling and loop fusion to optimize the direct convolution form of the 3D convolution operation; and guided by the optimized form of the 3D convolution operation, a module that can accelerate different 3D convolution operations is designed; and a memory access method based on weight reuse is also designed to reduce the memory access time of features and weights. Experimental results show that the proposed accelerator not only supports convolution kernels of different sizes, but also has a performance density of 1.59, which outperforms most existing accelerators.*

Keywords: *3D CNN, Loop Optimization, Parallel Computing, FPGA*

1. Introduction

By sliding the convolution kernel in three dimensions, three-dimensional convolutional neural networks (3D CNNs) have excellent 3D feature extraction capability, and have been brilliant in many aspects such as human action recognition [1] and hyperspectral image classification [2]. However, due to the high computational complexity of 3D CNNs, it takes a lot of time to execute inference on resource-constrained terminals, which greatly limits its application in practice. Therefore, more and more studies begin to pay attention to the acceleration of the inference phase of 3D CNNs. Common CNN acceleration platforms include GPUs, FPGAs and ASICs. GPUs generally contain thousands of stream processors, which can perform operations in parallel, but their power consumption is too large. ASICs can be customized for a specific layer, but their development cycle is too long and the cost is too high. FPGAs are programmable logic devices, and have the characteristics of low power consumption, reconfigurability and high performance, making them ideal for accelerating 3D CNNs.

Currently, several FPGA-based inference accelerators for 3D CNNs have been proposed. Fan et al. [3] propose a 3D residual module that reduces the number of parameters in the layers at the algorithmic level and a hardware architecture that accelerates different types of convolutional layers at the hardware level, but the performance density (i.e. the performance that a single DSP can provide per clock period) of this architecture is low. Shen et al. [4] propose a uniform acceleration architecture for 2D and 3D CNNs based on the Winograd algorithm. The Winograd algorithm can significantly reduce the number of multiplications in the operation, but it is only applicable to the case where the stride of the convolution kernel is 1, and the transform matrix of the Winograd algorithm is very sensitive to the size of the convolution kernel. Liu et al. [5] also proposed a uniform architecture design for accelerating 2D and 3D CNNs, which includes a module that maps convolution operations to matrix multiplication and uses a 2D multiply-and-accumulate array to improve the performance of matrix multiplication. The acceleration performance of this architecture is good, but due to the high-level synthesis design, the hardware cannot be precisely controlled, and there is still room for further improvement in terms of acceleration performance.

In this paper, we analyze common 3D CNNs, and propose an FPGA-based design for accelerating common 3D CNNs. According to the characteristics of 3D convolution operation, we adopt loop optimization design such as multi-dimensional loop tiling, loop unrolling, and adopt pipeline design to further improve the efficiency of parallel computing. Correspondingly, we also designed a memory access method based on reusing weight, which greatly reduces the pressure of memory access. Experimental results show that the accelerator proposed in this paper has excellent acceleration performance while supporting convolution kernels of different sizes.

2. CNN basics and characteristics

2.1 Basic principles of CNNs

CNNs are developed from neural networks. The basic unit of a neural network is the neuron, which can perform linear binary classification of the input data, and multiple layers of neurons combined together make up a neural network [6]. In some neural networks, neurons in each layer of the neural network are connected to all of the neurons in the next layer. As the number of layers increases, the network may contain a massive amount of weight parameters. In CNNs, taking the convolutional layer as an example, one neuron is only connected to some neurons in the next layer and the weight parameters of the same convolution kernel are constant when convolving with the input feature. The features of local connection and weight sharing significantly reduce the weight parameters in the network and simplify the complexity of the network model. Figure 1 demonstrates the 3D convolution operation. For ease of presentation, this paper uses the same data dimensional representation as TensorFlow, i.e. N for output channel, C for input channel, D for depth, H for height and W for width.

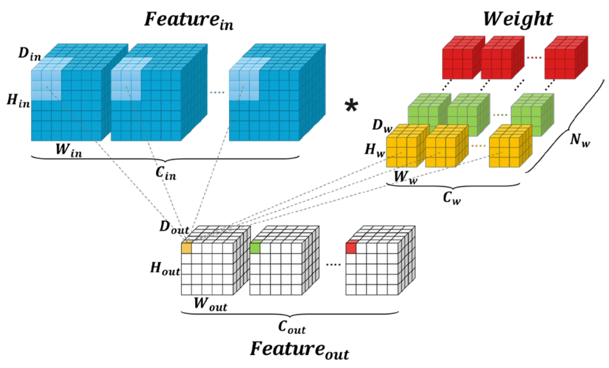


Figure 1: Schematic diagram of 3D CNN

CNNs generally contain convolutional layers, pooling layers and fully connected layers. Among them, convolutional layers play the role of feature extraction, pooling layers are to reduce the dimension of the features, and fully connected layers are to synthesize features of the previous layer. When various layers are combined into a CNN, the weight parameters contain enough information to characterize the input feature after sufficient training. In practical applications, only the inference phase of the CNN is used, so this paper will only focus on the acceleration of the inference phase.

2.2 Characteristics of common 3D CNNs

Common 3D CNNs include C3D [1], HybridSN [2] and other neural networks. When the input of a neural network is data such as hyperspectral images or videos, 3D CNNs can perform better 3D feature extraction. This is because compared to 2D convolution kernels, 3D convolution kernels slide in the dimension of the channel while sliding in the spatial dimension, and the network retains information about the channel after several convolutional layers, thus allowing more 3D features related to space and channel to be extracted [7].

By summarizing the characteristics of common 3D CNNs, the following rules can be obtained:

(1) The number of input channels of the first layer is the number of channels of the input feature, and the number of input channels of subsequent layers is mostly a multiple of 8.

(2) The convolution kernel sizes of different 3D CNNs are generally different, and the convolution kernel sizes of different layers of the same CNN are generally different. In general, the depth of the convolution kernel is 3, 5, 7, and the width and height are 3; in special cases, the width and height of the convolution kernel may also be 1.

(3) The number of parameters in the convolutional layers is generally smaller than that in the fully connected layers.

(4) The number of operations in the convolutional layers is generally larger than that in the fully connected layers.

3. Accelerator design

3.1 Overall accelerator architecture design

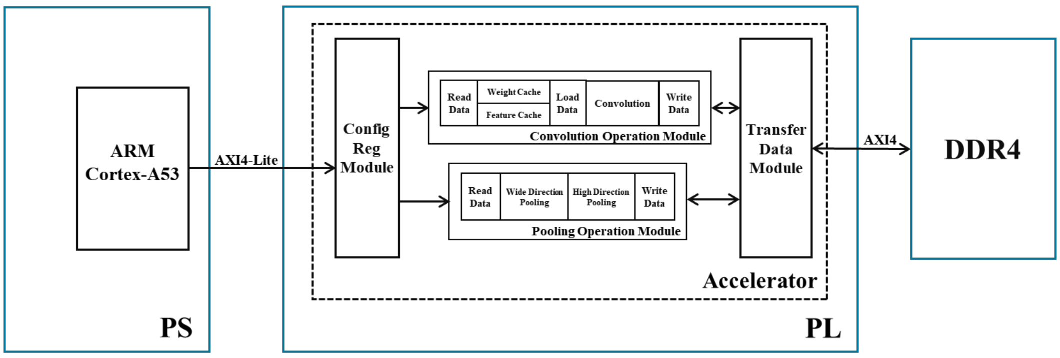


Figure 2: Overall architecture of the accelerator

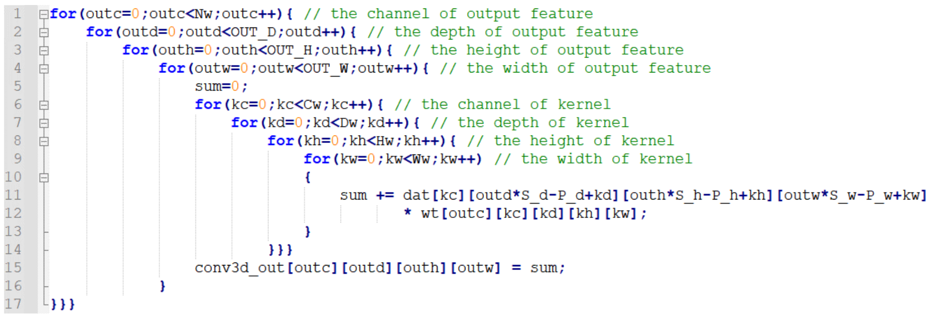
In order to design an accelerator that supports common 3D CNNs, this paper proposes an accelerator design with flexible parallelism and high efficiency. The accelerator proposed in this paper consists of three parts: control, computing and storage. Figure 2 shows the overall architecture of the accelerator designed on ZYNQ platform and the accelerator is all implemented on the programmable logic side, that is, FPGA. It can be seen from Figure 2 that the accelerator mainly includes a configuration register module, a convolution operation module, a pooling operation module, and a transfer data module, all of which are implemented on the programmable logic side. The configuration register module is responsible for receiving the information sent from CPU on the processing system side, and configuring the corresponding registers of the operation module. The convolution operation module can perform parallel computing in dimensions such as the channel of the output feature, the channel of the convolution kernel, and the depth of the convolution kernel to accelerate the 3D convolution operation. The pooling operation module performs parallel computing in the dimension of the input channel to accelerate the 2D pooling operation. In order to complete data transfer between the operation modules and the external memory while avoiding the introduction of complex control logic, a transfer data module is designed between the operation modules and the external memory, which can convert multiple sets of custom data buses to AXI4 bus.

The accelerator in this paper is a dataflow processing architecture, i.e. the accelerator already contains internal circuit structures that implement the configuration and computation of a particular layer, and the corresponding output is available after the input data has flowed through the circuit [8]. From the dataflow perspective, CPU on the processing system side sends configuration information to the accelerator, such as the kernel sizes, the base address of the read/write data, whether to start convolution, etc. The configuration register module of the accelerator receives the information and configures the information to the corresponding registers in operation modules. After configuring the registers and receiving the message to start the operation, the operation module sends a read data request to the external memory via the transfer data module and stores the returned data in the on-chip cache before starting the corresponding operation. After completing the operation, the operation module sends a write request to the external memory via the transfer data module to write the result of the operation to the specified address of the external memory.

The key to the acceleration of the accelerator is the design of the operation module. The convolution operation module achieves acceleration of convolution operation by optimizing access memory by adding caches and reusing weights, performing parallel computing in multiple dimensions such as the channel of the output feature, the channel of the convolution kernel and the depth of the convolution kernel, and performing loading data, convolution and writing data in a pipelined fashion. In addition, the key for the accelerator to support setting convolution kernels of different sizes is that the loading data submodule contains two loops of the height of the convolution kernel and the width of the convolution kernel. When facing some CNNs with different kernel sizes, it can be achieved by configuring the upper limit of these two loops in the accelerator. The pooling operation module performs parallel computing in the input channel dimension, and uses pipeline design to read data, wide-direction pool, high-direction pool and write data, both of which also achieve good acceleration of pooling operation.

3.2 Loop optimization design

Convolution operations are in fact the operations of a large amount of repeated data over multiple loops, so the focus of this paper is the optimized design of multiple loops. In an ideal convolution operation, all loops can be unrolled and computed at the same time, but it is often difficult to unroll all loops due to the limitation of hardware computing resources. Therefore, in order to reduce the consumption of hardware resources, it is necessary to use loop tiling and loop unrolling in the actual design process. Loop tiling means that a large loop can be split into smaller loops to reduce the consumption of hardware resources, while loop unrolling means that the contents of a loop can be unrolled multiple times to increase the chance of parallel computing.

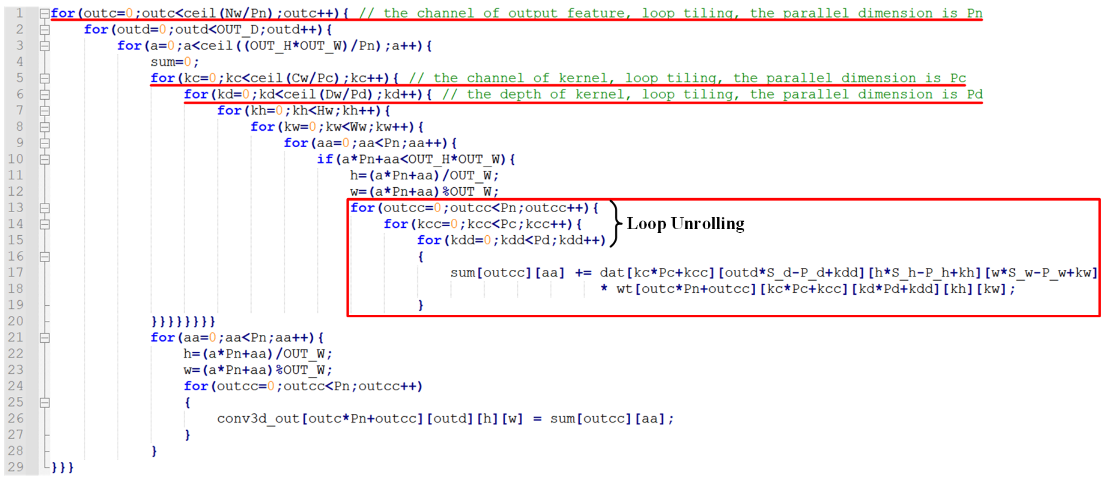


```
for(outc=0;outc<Nw;outc++){ // the channel of output feature
    for(outd=0;outd<OUT_D;outd++){ // the depth of output feature
        for(outh=0;outh<OUT_H;outh++){ // the height of output feature
            for(outw=0;outw<OUT_W;outw++){ // the width of output feature
                sum=0;
                for(kc=0;kc<Cw;kc++){ // the channel of kernel
                    for(kd=0;kd<Dw;kd++){ // the depth of kernel
                        for(kh=0;kh<Hw;kh++){ // the height of kernel
                            for(kw=0;kw<Ww;kw++) // the width of kernel
                            {
                                sum += dat[kc][outd*S_d-P_d+kd][outh*S_h-P_h+kh][outw*S_w-P_w+kw]
                                        * wt[outc][kc][kd][kh][kw];
                            }
                        }}}
                conv3d_out[outc][outd][outh][outw] = sum;
            }
}}}
```

Figure 3: Pseudocode for the direct convolution form of the 3D convolution operation

Figure 3 shows the pseudocode for the direct convolution form of the 3D convolution operation. From Figure 3, it can be seen that the direct convolution form of 3D convolution operation contains eight loops, namely the channel of the output feature, the depth of the output feature, the height of the output feature, the width of the output feature, the channel of the convolution kernel, the depth of the convolution kernel, the height of the convolution kernel and the width of the convolution kernel.

Combined with the analysis of common 3D CNNs, the width and height of the convolution kernel are generally between 1×1 and 9×9, and if loop tiling is performed in these two dimensions, it is easy to cause the problem that the utilization rate of the multiply-and-accumulate array is not high. The depth of the convolution kernel generally does not exceed 9 and is not 1, which is suitable for loop tiling. In the dimension of the channel of the convolution kernel, generally except for the first convolutional layer, the number of channels is a multiple of 8 or 16, so this dimension is also suitable for loop tiling. The channel of the output feature is generally a multiple of 8 or 16, which is suitable for loop tiling. The height, width and depth of the output feature of different CNNs varies greatly, which need to be analyzed in combination with specific network structures.



```
for(outc=0;outc<ceil(Nw/Pn);outc++){ // the channel of output feature, loop tiling, the parallel dimension is Pn
    for(outd=0;outd<OUT_D;outd++){
        for(a=0;a<ceil((OUT_H*OUT_W)/Pn);a++){
            sum=0;
            for(kc=0;kc<ceil(Cw/Pc);kc++){ // the channel of kernel, loop tiling, the parallel dimension is Pc
                for(kd=0;kd<ceil(Dw/Pd);kd++){ // the depth of kernel, loop tiling, the parallel dimension is Pd
                    for(kh=0;kh<Hw;kh++){
                        for(kw=0;kw<Ww;kw++){
                            for(aa=0;aa<Pn;aa++){
                                if(a*Pn+aa<OUT_H*OUT_W){
                                    h=(a*Pn+aa)/OUT_W;
                                    w=(a*Pn+aa)%OUT_W;
                                    for(outcc=0;outcc<Pn;outcc++){    } Loop Unrolling
                                        for(kcc=0;kcc<Pc;kcc++){
                                            for(kdd=0;kdd<Pd;kdd++)
                                            {
                                                sum[outcc][aa] += dat[kc*Pc+kcc][outd*S_d-P_d+kdd][h*S_h-P_h+kh][w*S_w-P_w+kw]
                                                        * wt[outc*Pn+outcc][kc*Pc+kcc][kd*Pd+kdd][kh][kw];
                                            }
            }}}}}}}}
            for(aa=0;aa<Pn;aa++){
                h=(a*Pn+aa)/OUT_W;
                w=(a*Pn+aa)%OUT_W;
                for(outcc=0;outcc<Pn;outcc++)
                {
                    conv3d_out[outc*Pn+outcc][outd][h][w] = sum[outcc][aa];
                }
            }
}}}
```

Figure 4: Pseudocode for 3D convolution operation based loop optimization

As shown in Figure 4, this paper performs loop optimization on the basis of the direct convolution form of the 3D convolution operation, and selects the channel of the output feature, the channel of the convolution kernel and the depth of the convolution kernel for loop tiling. The tiling sizes of the three dimensions are P_N, P_C and P_D in sequence, which can be flexibly adjusted according to the specific network structure and hardware resources. Afterwards, loop unrolling is performed on the subloop of the three dimensions after loop tiling, and these subloops are mapped into a multiply-and-accumulate array of the accelerator to complete the parallel computing on the P_D depth on the P_C input channels on the P_N output channels. In addition, since the height and width of the output feature are equivalent, this paper also fuses the height and width dimensions of the output feature, and performs loop tiling on the new fused loop.

Compared with the unoptimized pseudocode, the loop-optimized pseudocode can perform parallel computing in three dimensions, including the channel of the output feature, the channel of the convolution kernel, and the depth of the convolution kernel, and perform serial loops in other dimensions, which provides great convenience for the subsequent design of the convolution operation module.

3.3 Pipeline design

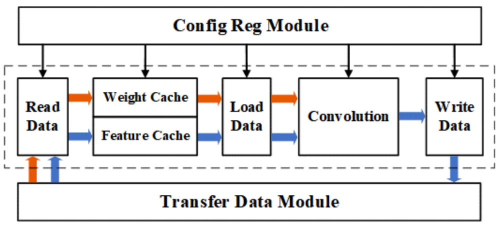


Figure 5: Schematic diagram of the internal structure of the convolution operation module

The core part of the accelerator is the convolution operation module. The internal structure of the convolution operation module is shown in the dashed box in Figure 5, which includes five submodules, namely reading data, weight cache and feature cache, loading data, convolution, and writing data.

In order to further improve the performance, pipeline design is introduced into the convolution operation module, where each of the five submodules respectively corresponds to five-stage pipeline, and the end signal of each submodule corresponds to the start signal of the previous one.

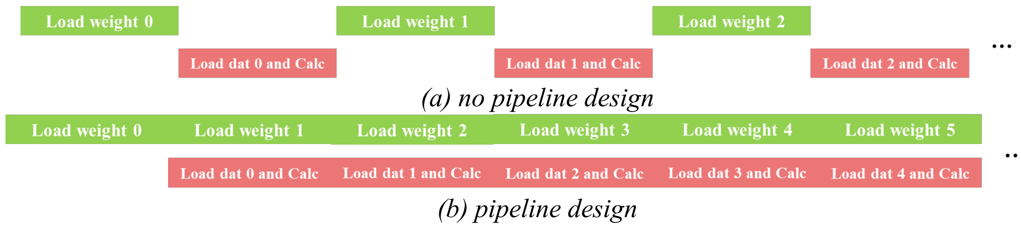


Figure 6: Comparison of no pipeline design and pipeline design in convolution operation module

Among the five submodules of the convolution operation module, the convolution submodule takes the longest time and has the most potential for acceleration. As shown in Figure 6(a), to complete a convolution operation without pipeline design, the convolution operation module needs to load the weight data first, then load the feature data and complete the operation; as shown in Figure 6(b), the pipeline-optimized convolution operation module loads the feature data and completes the operation while also loading the weight data, which enables the submodules to run at full capacity without idle states, thus further improving the acceleration efficiency of the convolution operation module.

3.4 Data reuse and memory access design

The design of the neural network accelerator should consider not only the computing capacity of operation modules, but also the data transfer between operation modules and the memory, i.e. to reduce the transfer of duplicate data and to set the cache reasonably to optimize the memory access. During the convolution operation, a large amount of data participates in the operation on multiple loops, and some data is reused. Therefore, it is of great help to improve the performance of the accelerator to reasonably design the data reuse mode by using characteristics of convolution operation.

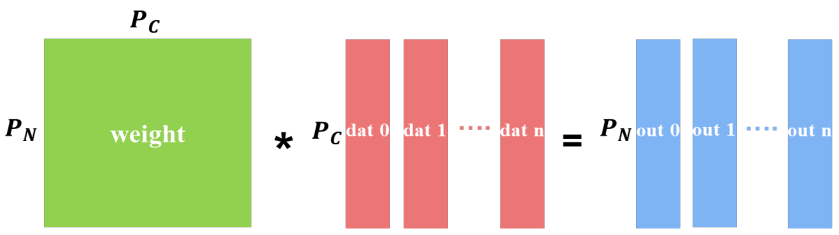


Figure 7: Schematic diagram of convolution operation with weight reuse

Through the analysis of the pseudocode after loop optimization, it is found that the upper-level loop outside the red box is related to the height and width of output feature after loop fusion and loop tiling. The feature data on this loop changes, while the weight data is fixed. As shown in Figure 7, when this loop is executed, it is only necessary to fix the weight data and update the feature data to complete the operation on this loop; after the weight data completes all operations, then to load new weight data. Therefore, the accelerator in this paper adopts the method of reusing weight data to reduce the pressure of data transfer, and sets the weight cache and feature cache separately as a data cache bridge between the convolution operation module and the external memory.

The ways of weight reuse and parallel computing affect the data storage method. On the one hand, the feature and weight data should be stored in the external memory in the order of participating in parallel computing; on the other hand, the data storage method cannot violate the requirements of weight reuse. From the above two points, this paper designs a suitable data storage method for the 3D convolution operation accelerator.

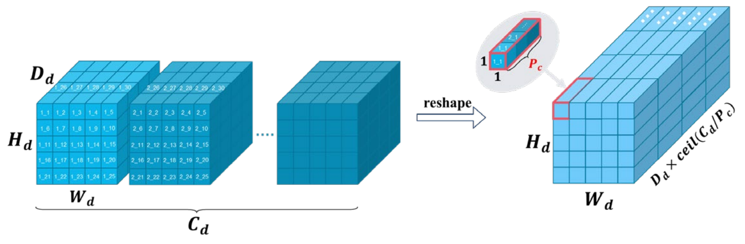


Figure 8: Schematic diagram of features rearrangement

The first is the feature storage method. Taking Figure 8 as an example, the features in the 3D convolution operation are four-dimensional, expressed as (C_d, D_d, H_d, W_d). In order to facilitate the expression, the original four-dimensional features are first rearranged into the new three-dimensional features. The rearrangement method is to arrange the first depth data of all input channels in the original features in sequence, then arrange the second depth data in sequence, and so on until the last depth data are arranged, and finally to concatenate these data together along the depth direction. Considering every P_C data with a width of 1 and a height of 1 as a subblock, the new three-dimensional features formed by these subblocks can be represented as (D_d×ceil(C_d/P_C), H_d, W_d). The feature storage method is to store the data of the first subblock in the first row and first column of the new three-dimensional features in sequence, and then store the data of the first subblock in the first row and second column in sequence, and so on until all the data of the first subblock in the first depth are stored following the order of width and height. Then repeat the above operations on the depth dimension.

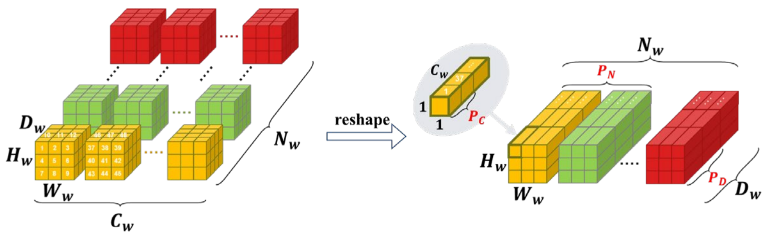


Figure 9: Schematic diagram of weights rearrangement

Next is the weight storage method. Taking Figure 9 as an example, the weights in the 3D convolution operation is five-dimensional, expressed as (N_w, C_w, D_w, H_w, W_w). For ease of introduction, it is also necessary to rearrange the original five-dimensional weights into the new four-dimensional weights. The rearrangement method is to arrange the first depth data of all input channels on each output channel in the original weights in sequence, then arrange the second depth data in sequence, and so on until the last depth data, and finally to concatenate these data together along the depth direction on each output channel. Considering each C_w data with a width of 1 and a height of 1 on each output channel as a subblock, then the new four-dimensional weights formed by these subblocks can be expressed as (N_w, D_w, H_w, W_w). The dimensions of parallel computing related to weights are P_N, P_C and P_D. The weight storage method is to store the P_C data of each subblock in the P_D subblock in the first row and first column of each output channel of the first P_N output channel of the new four-dimensional weights in sequence, and then store the P_C data of each subblock in the first row and second column of each output channel in sequence, and so on until the first P_C data of each P_D subblock on each output channel are stored in the order of width and height; then store the second P_C data in each subblock of P_D subblock in the first row and first column of each output channel of the first P_N output channel, and so on until all the weights on the first P_N output channels are stored. Finally, repeat the above operations in the order of depth and output channel.

4. Experimental results

4.1 Experimental platform

The CNN inference accelerator designed in this paper is based on an FPGA platform. This paper uses ZYNQ UltraScale+ development board from Alinx as the development platform. The core board of this development board uses XILINX Zynq UltraScale+ EG chip, which has a capacity of 4GB high-speed DDR4 SDRAM chip on the processing system side. The simulation environment of this paper is in Vivado 2020.2 and the hardware description language is Verilog.

4.2 Results analysis

In this paper, the acceleration effect of CNNs will be illustrated from the aspects of acceleration time, performance, hardware resource consumption and comparison with other accelerators based on FPGA, so as to verify the characteristics of the proposed accelerator.

4.2.1 Results analysis on HybridSN

The CNN selected in this paper is HybridSN. In addition to including 3D convolutional layers and a 2D convolutional layer, the kernel sizes for the three 3D convolutional layers of HybridSN are different, which are 7×3×3, 5×3×3 and 3×3×3 respectively. The above two aspects can fully verify the versatility of the accelerator in the context of different kernel sizes. The accelerator uses a 100MHz synchronous clock. The input features and weights are int8 type data that have been quantized, and the output features are also int8 type data. And the performance is the ratio of the number of operations to the acceleration time.

Table 1: Acceleration effect of each layer of HybridSN on FPGA

Layer	Shape of the input feature map	Kernel's number/size/stride	Number of operations /MFLOPs	Acceleration time /ns	Performance /GOPS
Conv3d_1	(1, 30, 25, 25)	8/ 7×3×3/ 1	12.8	8156240	1.57
Conv3d_2	(8, 24, 23, 23)	16/ 5×3×3/ 1	101.6	4067760	24.98
Conv3d_3	(16, 20, 21, 21)	32/ 3×3×3/ 1	179.7	3600080	49.92
Conv2d_1	(576, 19, 19)	64/ 3×3/ 1	191.8	4292450	44.68
FC_1	(18496)	256	9.5	5959660	1.59
FC_2	(256)	128	0.07	41440	1.58
Classification	(128)	16	0.004	2660	1.54

Considering the conditions of each layer and hardware resources, P_N is set to 16, P_C is set to 16 and P_D is set to 8. The acceleration effects of the accelerator on each layer of HybridSN are summarized in Table.1. It can be seen from the table that the acceleration effect of the 3D convolutional layers increases with the increase of the input channel and output channel of the input feature map, and the performance of the third 3D convolutional layer reaches the maximum, which is 49.92 GOPS. This is because the input channel of the first 3D convolutional layer is 1 and the output channel is 8, while the input channel of the second 3D convolutional layer is 8 and the output channel is 16. When P_N is 16 and P_C is 16, these two convolutional layers do not exert the parallel computing capability of the accelerator at all, so the acceleration time is longer and the performance is lower.

Because the 2D convolution operation can be regarded as the 3D convolution operation with the depth of 1, the accelerator in this paper is also compatible with the 2D convolution operation. Although the input channel and output channel of the 2D convolutional layer in HybridSN can give full play to the parallel computing capability of the accelerator, the acceleration performance of the 2D convolutional layer is slightly lower than that of the third 3D convolutional layer because the accelerator needs to spend more time loading data between the cache and the convolution submodule during the whole acceleration process. In addition, the accelerator is also compatible with the acceleration of the fully connected layers.

Table 2: FPGA resource utilization

Resource	LUT	FF	BRAM	DSP
Utilization	94804	59947	456	1221
Available	341280	682560	744	3528
Utilization (%)	27.78	8.78	61.29	34.61

The total on-chip power of the accelerator is 4.359 W. Table.2 shows the resource utilization of FPGA with P_N of 16, P_C of 16 and P_D of 8. The size of the weight cache and feature cache is 1MB each. Among them, DSPs are mainly used to realize the operation logic of the multiplication part in the convolution operation module, and DSP utilization rate is 34.61%; BRAMs are mainly used to realize on-chip cache, and BRAM utilization rate is 61.29%. The utilization rates of FFs and LUTs are 8.78% and 27.78% respectively.

4.2.2 Comparison with other accelerators on C3D

Table 3: Comparisons with other accelerators based on FPGA

	[3]	[5]	[4]	Ours
CNN	C3D	C3D	C3D	C3D
FPGA	Xilinx ZC706	Xilinx XC7VX690T	Xilinx XC7VX690T	Xilinx XCZU15EG
Clock (MHz)	172	120	150	100
DSP Utilization	810	3595	1536	1317
Performance (GOPS)	142	667.7	431	209.4
Performance Density (GOPS/DSPs/MHz)	1.02	1.55	1.87	1.59

Table.3 compares the acceleration effect of the accelerator in this paper with that of accelerators from other papers. Since the kernel sizes of 3D convolutional layers in C3D are all 3×3×3, [4] uses the 3D Winograd algorithm to reduce the computational complexity of the network, that is, to reduce multiplication and increase addition. In this way, the impact on resources is to reduce the use of DSPs and increase the use of LUTs, which is also the reason why [4] has the best performance density. However, in terms of flexibility, the accelerator of [4] still has some limitations. Winograd algorithm is only applicable when the stride of the convolution kernel is 1, and its transformation matrix changes with the convolution kernel size, thus affecting the performance density. However, through the design of loading data submodule, the accelerator in this paper can be applied to CNNs with different convolution kernel sizes and can be applied with good generality. Moreover, through loop optimization and pipeline design, the performance density is superior to that of [3, 5].

5. Conclusions

This paper analyzes the characteristics of 3D convolution operations, summarizes the rules of common 3D CNNs, and proposes a 3D CNN inference accelerator based on FPGA. The accelerator adopts methods such as loop tiling, loop unrolling and loop fusion to obtain an optimized form of 3D convolution operation, and determines the parallel computation in three dimensions. And the accelerator also determines a memory access method based on weight reuse and the corresponding feature and weight storage method. The accelerator can not only accelerate 3D convolutional layers with different convolution kernels, but also accelerate 2D convolutional layers with different convolution kernels and fully connected layers. Experimental results show that compared with the accelerator designed by the 3D Winograd algorithm, the accelerator proposed in this paper is more versatile; and compared with other existing acceleration methods, the performance density of the accelerator is better. At present, the accelerator only considers parallel computing from three dimensions. In the future, it will explore the feasibility of parallel computing in terms of the height, width, and depth of output feature in combination with specific CNNs.

Acknowledgements

The National Key Research and Development Program of China, Grant No.2022YFB3902304.

The Pilot Fund of Aerospace Information Research Institute, Grant No. E2Z206010F.

References

[1] Ji S, Xu W, Yang M, et al. 3D convolutional neural networks for human action recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2012,35(1):221-231.
[2] Roy S K, Krishna G, Dubey S R, et al. HybridSN: Exploring 3-D–2-D CNN feature hierarchy for hyperspectral image classification [J]. IEEE Geoscience and Remote Sensing Letters, 2019,17(2):277-281.
[3] Fan H, Niu X, Liu Q, et al. F-C3D: FPGA-based 3-dimensional convolutional neural network[C]. 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017. IEEE.
[4] Shen J, Huang Y, Wang Z, et al. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA[C]. Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2018.
[5] Liu Z, Chow P, Xu J, et al. A uniform architecture design for accelerating 2D and 3D CNNs on FPGAs [J]. Electronics, 2019, 8(1):65.
[6] Bishop C M. Neural networks and their applications[J]. Review of scientific instruments, 1994, 65(6):1803-1832.
[7] Tran D, Bourdev L, Fergus R, et al. Learning spatiotemporal features with 3d convolutional networks[C]. Proceedings of the IEEE international conference on computer vision, 2015.
[8] Hou R, Zhang L, Huang M C, et al. Efficient data streaming with on-chip accelerators: Opportunities and challenges[C]. 2011 IEEE 17th International Symposium on High Performance Computer Architecture, 2011. IEEE.