Efficient Architecture for Convolution and Softmax Function in Deep Learning Accelerator

Zhenyu Jiang[1(✉)], Zhifeng Zhang[1], Haoqi Ren[1], and Jun Wu[2]

[1] College of Electronic and Information Engineering, Tongji University, Shanghai, China
{1832925, zhangzf, renhaoqi}@tongji.edu.cn
[2] School of Computer Science, Fudan University, Shanghai, China
wujun@fudan.edu.cn

Abstract. Convolutional neural network (CNN) has been widely used in deep learning. However, the **hardware** consumption **of the convolutional neural network** is very large. Traditional Central Processing Units (CPUs) and Graphic Processing Units (GPUs) are inefficient and expensive for neural network, so an efficient hardware design is required. The proposed design based on Digital Signal Processor (DSP) has rapid operating speed and strong computation ability for training and inference of CNN. In this paper, the hardware architecture of convolution and softmax function is specially optimized. Winograd algorithm can reduce multiplications of convolution, thus decreases hardware complexity, since multiplication is much more complex in hardware implementation than addition. The softmax function is also simplified by replacing divider by subtractor and logarithmic function which cost fewer resources. The proposed hardware architecture dramatically decreases the complexity and hardware resources.

Keywords: Convolutional neural network · Hardware architecture · Convolution · Winograd algorithm · Softmax function

1 Introduction

With the great success achieved by deep learning techniques in many areas, the convolutional neural network (CNN) is widely used in many fields [1]. CNN is one of the most popular algorithms for deep learning and generally consists of an input layer, multiple convolutional layers, multiple pooling layers, a fully connected layer, and an output layer [2]. The convolutional layer is used to extract the feature information from the input image [3], it mainly consists of convolutional kernels, and mapping the input map to the output feature map [4], which is the most computationally intensive module of the convolutional neural network.

Common convolutional neural network models are very large in terms of computation and parameters, so both training and inference usually require high-performance hardware to provide support for compute and storage requirements [5]. Traditional CPUs and GPUs can be used to execute convolutional neural network

H. Gao et al. (Eds.): ChinaCom 2020, LNICST 352, pp. 631–643, 2021.
https://doi.org/10.1007/978-3-030-67720-6_43

algorithms, but they are slow and have low performance. FPGA, because of the flexibility of programming and the parallelism of computing, is often more favored, but the speed and energy consumption ratio are worse compared to ASICs [6]. To reduce the training and inference time of neural networks, the hardware needs to be specifically optimized.

Google proposed a tensor processing unit (TPU) that can run multiple machine learning algorithms [7]. Performance is 15 to 30 times better than the general-purpose CPUs and GPUs, and 30 to 80 times less power consumption. Compared with GPUs, TPUs use low-precision (8-bit) computing, which significantly reduces power consumption and speeds up operation while maintaining accuracy. Also, the TPU uses pulsed arrays, which are optimized to reduce I/O operations.

Digital Signal Processor (DSP) has strong parallel computing capabilities with SIMD (single instruction multiple data) and VLIW (very long instruction word) functions [8], so it is suitable for massively parallel computation of deep learning neural networks [9]. This paper proposes a deep learning accelerator (DLA) based on DSP, which acts as a DSP co-processor and specializes in neural network operations. The Winograd algorithm is used to accelerate the convolution operations and reduces the MAC resources required for the convolution computation [10]. Totally 256 MAC cells, each contains four 16-bit multipliers. The proposed accelerator supports the direct convolution of 2×2 convolution kernels as well as the 3×3 Winograd convolution. The softmax function is also specially optimized to ensure accuracy while reducing hardware resources.

2 The Overview of the Accelerator

2.1 Overall Architecture

The accelerator can be divided into the following modules based on the function. Configuration registers are for storing the weight of kernels, size of the feature map, channels, and other main parameters. The control unit controls the behavior of the accelerator according to the configuration parameters, mainly including operation requests, data read requests, and write responses. The input interface is for receiving the input feature map and parameters. The pre-processing module is for Winograd matrix pre-processing, and the MAC array performs matrix multiplication operations. The POOL and RELU perform pooling and activation respectively. The FC performs the fully connected function to get the final results.

Figure 1 shows the architecture of DLA. First, the configuration registers need to be configured. The input and output interfaces communicate with the processor via the AXI bus by DMA. At the pre-processing layer, Winograd pre-processing of the matrix is performed, as well as the floating-point to fixed-point conversion to reduce the network data storage and data transmission bandwidth. After padding and rearranging, data will be sent to the MAC array to complete the convolution operation. After pooling and convolution, data will be sent to the full connection layer for feature extraction. The final results will be transferred through the output interface to SRAM.

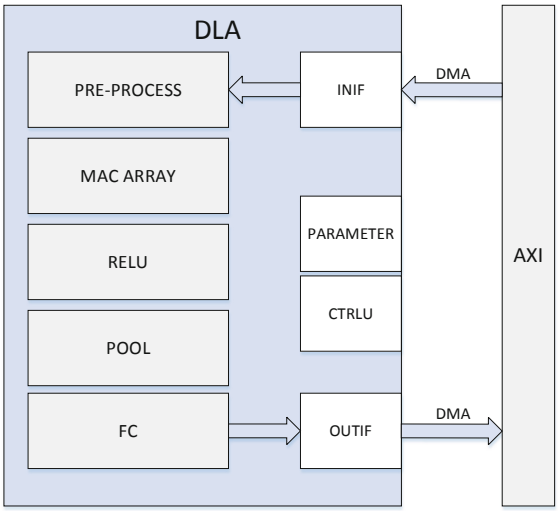

Fig. 1. The architecture of the DLA

2.2 Pre-processing Layer

In the pre-processing layer, the floating-point data will be transferred to fixed-point format first, which supports up to 16-bit fixed-point operation. Then the padding of the image will be done according to the length and width in the configuration register. The image data will then be pre-processed for the Winograd algorithm.

The convolution operation of an image can be represented by Fig. 2 as an example of a 3×3 convolution, the kernel is smoothed over the padding image with a step of strip. Multiply the elements of the image by ones of the convolution kernel at the corresponding positions and then add them together to get the result at this position. Convolution results can be got by iterating over the image following the above steps.

The process of convolution can be written in the form of matrix multiplication [11].

$$r_1 = K \cdot G \tag{1}$$

Where $K = [k_{11}, k_{12}, k_{13}, k_{21}, k_{22}, k_{23}, k_{31}, k_{32}, k_{33}]$,
$G = [g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9]^T$.

As we can see, each convolution requires at least 9 multiplications and 8 additions. Multiplication is much more complex to implement in hardware than addition. Therefore, to reduce the number of multiplications, the Winograd algorithm is deliberately introduced.

First, take the 1D convolution as an example. Input $K = [k_0, k_1, k_2, k_3]$ and the convolution kernel is $G = [g_1, g_2, g_3]^T$. Then the convolution can be written as the following matrix Multiplication form:

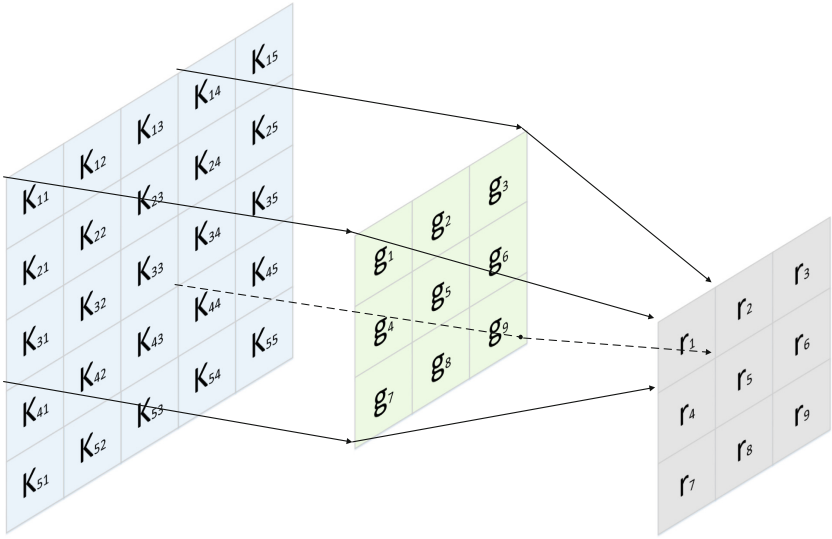

Fig. 2. Matrix convolution diagram

$$F(2,3) = \begin{bmatrix} k_0 & k_1 & k_2 \\ k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} \tag{2}$$

Since there are many repeating elements regularly distributed in the matrix, the result can be written in the following form:

$$F(2,3) = \begin{bmatrix} k_0 & k_1 & k_2 \\ k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} a_1 + a_2 + a_3 \\ a_2 - a_3 - a_4 \end{bmatrix} \tag{3}$$

Where $a_1 = (k_0 - k_2)g_0$, $a_2 = (k_1 + k_2)\frac{g_0 + g_1 + g_2}{2}$, $a_4 = (k_1 - k_3)g_2$, $a_3 = (k_2 - k_1)\frac{g_0 - g_1 + g_2}{2}$. A total of 4 multiplications and 12 additions are required. Since the elements of the convolution kernel are fixed in advance, these additions only need to be calculated once. The hardware implementation of multiplication is more complex than addition. The proposed convolution calculation is more resource-efficient than the direct convolution calculation.

Two-dimensional convolution can be written in the following form. Take for example a 4 × 4 image and a 3 × 3 convolution kernel. The result of their convolution

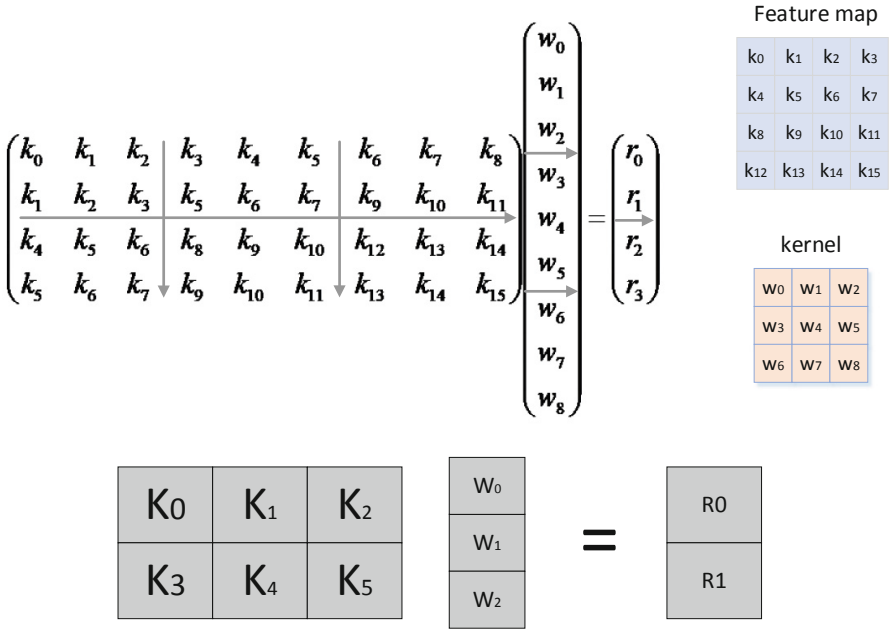

Fig. 3. Matrix convolution partition

can be expressed as $F = KW$. The convolution matrix can be represented by dividing K and W into the following submatrices.

Then R_0 and R_1 can be express by:

$$R_0 = (A_0 + A_1 + A_2),\ R_1 = (A_1 - A_2 - A_3) \tag{4}$$

Where,

$$A_0 = (K_0 - K_2) \cdot W_0,$$

$$A_1 = (K_1 + K_2) \cdot \frac{W_0 + W_1 + W_2}{2},$$

$$A_2 = (K_2 - K_1) \cdot \frac{W_0 - W_1 + W_2}{2},$$

$$A_0 = (K_1 - K_3) \cdot W_2 \tag{5}$$

Since the pattern of the data repetition distribution of each sub-matrix is the same, the number of multiplication can also be reduced by the Winograd algorithm. Each sub-matrix multiplication requires four multiplications, so the total multiplication times are 16, which greatly reduces the multiplication complexity compared with the direct convolution requiring 36 multiplications.

The convolutional layer of the convolutional neural network performs a 3D convolution, which is equivalent to summing the 2d convolutional results on each channel

[12]. For feature maps of different sizes, they can be divided into overlapping tiles of equal-sized (2 × 3), and perform Winograd convolution on each tile.

The proposed DLA supports both 3 × 3 and 2 × 2 convolution kernels. 3 × 3 convolution will be pre-processed by the Winograd algorithm. Each sub-matrix requires four multiplications to get the result. The 2 × 2 direct convolution also requires four multiplications for each sub-matrix. In order to maintain the hardware regularity of the MAC array and reduce the complexity of the algorithm, 2 × 2 convolution adopts the direct convolution method while 3 × 3 convolution is pre-processed by Winograd algorithm and is rearranged as input into the MAC array.

2.3 MAC Array

MAC array is used to perform convolution operations. Each MAC cell in the array has the same size and function. The rearranged data is transmitted and calculated in adjacent processing units (PE). Convolution computation contains a large number of highly regular and parallelizable repetitive computations that require huge computing power. The MAC array takes full advantage of the regularity and parallelism, data are passed in parallel to the PE array by using a modular and pipeline design, so that the architecture is more regular and the backend design is easier to route, thus greatly increasing the frequency.

Taking 2 × 2 convolution operation as an example, each MAC cell contains 4 PEs which can perform a 16-bit fixed-point multiplication operation. The convolution kernel parameters are pre-stored in the configuration register and only need to be read once at the beginning. The 2 × 2 convolution takes the direct convolution method, the results of

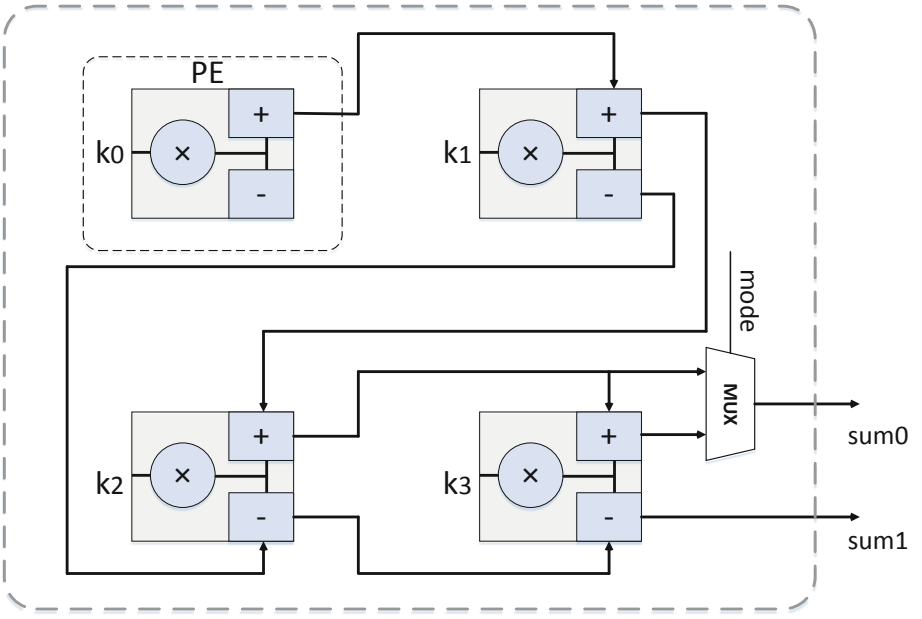

Fig. 4. MAC cell architecture

multiplying the input and weight in each PE can be added directly to get the final result. The 3 × 3 Winograd convolution also requires 4 multiplication operations and multiplexes the same MAC cell with 2 × 2 convolution. However, the 3 × 3 Winograd convolution operation has two results that cannot be added directly while 2 × 2 convolution has only one result. The basic architecture of the MAC cell which combines the 3 × 3 convolution with the 2 × 2 convolution operation is shown in Fig. 4.

The mode bit in the configuration register indicates whether a 3 × 3 or a 2 × 2 convolution operation is used. If mode indicates a 2 × 2 direct convolution, sum0 is output as the only result. If 3 × 3 mode is indicated, the results are sum0 and sum1.

2.4 Softmax Function Implementation

After convolution, pooling and activation, the final layer of the deep neural network is usually a fully connected layer and softmax function (classification network). Softmax function maps the output of the fully connected layer to probabilities, ensuring that the sum of the output is 1 [13].

The softmax function can be expressed by:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, (i = 1, 2, \ldots, N) \tag{6}$$

where $x_1, x_2, \ldots, x_N$ are the output vectors of the fully connected layer. The hardware

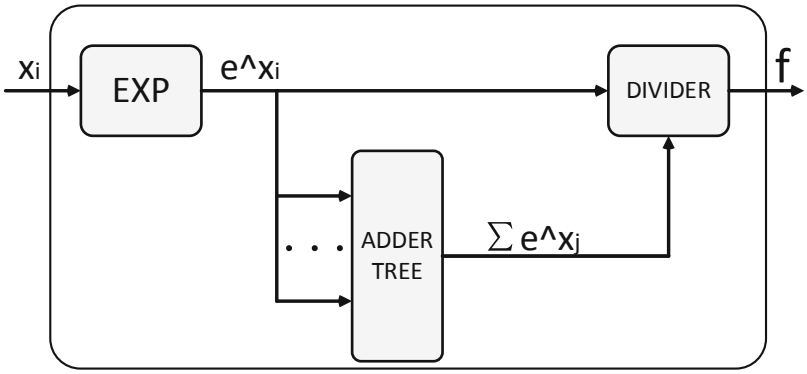

Fig. 5. The architecture of softmax function direct implementation

architecture that directly implements the softmax function can be expressed in the following form.

The division operation is very complex in hardware implementation. In order to simplify the implementation of the softmax function, the above formula can be expressed in the following form [14].

$$f(x_i) = \frac{e^{x_i - x_{\max}}}{\sum_{j=1}^{N} \exp(x_j - x_{\max})} = \exp\left(x_i - x_{\max} - \ln\left(\sum_{j=1}^{N} e^{x_j - x_{\max}}\right)\right) \tag{7}$$

The above equation converts division to addition (subtraction) as well as logarithmic operation and prevents overflow from the calculation $(x_i - x_{\max} \leq 0)$. The sorting block aims to find out the maximum value of x_i. An example of 4-input sorting block consisting of multilevel compare and select units is shown in Fig. 6. Compared with the original direct computation architecture, divisors are replaced by subtractors and logarithmic units which have much lower hardware complexity. The hardware architecture of softmax function is shown in Fig. 7. Since the path length of logarithmic units and subtractors are shorter than the data paths of divisors, the critical path length of the proposed architecture can be significantly shortened.

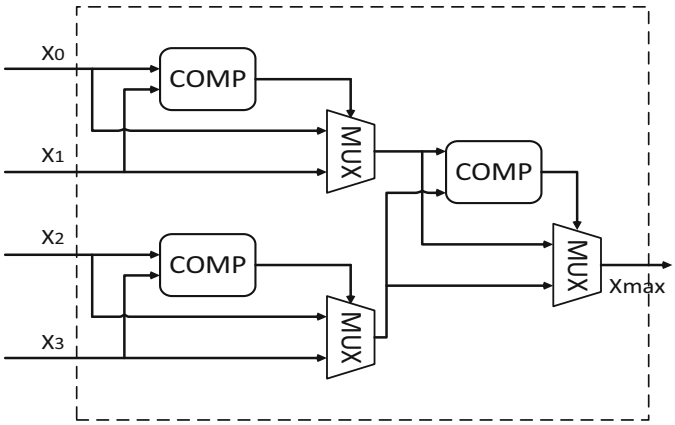

Fig. 6. The architecture of sorting block

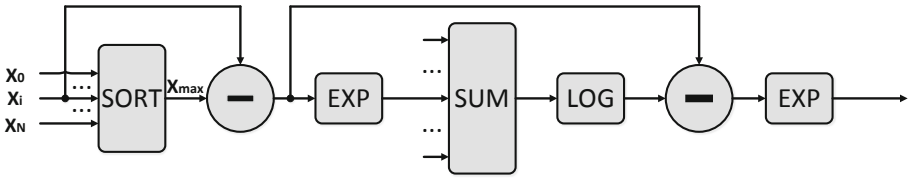

Fig. 7. The architecture of softmax function

Exponential calculation of e can be usually implemented by using the lookup table. This method is the fastest with the least error, but the largest in area and resources. Especially when multiple exponential units are computed in parallel, hardware resource consumption is extremely significant. In order to reduce the hardware resources of the

exponential function, the following method is used to simplify the exponential computation.

Set $F = \exp(x_i - x_{\max})$, since $x_i - x_{\max} \leq 0$, $F \in (0, 1]$. Then

$$F = 2^{(x_i - x_{\max}) \log_2 e} = 2^{-(x_{\max} - x_i) \log_2 e} \tag{8}$$

Since $\log_2 e$ is a constant, it is not necessary to use conventional multipliers to calculate $(x_{\max} - x_i) \log_2 e$, but instead a modified constant multiplier can be used. The approximate value of $\log_2 e$ in binary can be expressed as 1.0111. If a certain bit of the multiplier is 1, simply shift the multiplicand to the left by corresponding bits, and then add up all the shifting results.

Set the integer and decimal parts of $(x_{\max} - x_i) \log_2 e$ to be u and v respectively.

$$F = 2^{-(u+v)} = 2^{-u} \times 2^{-v} \tag{9}$$

The above equation converts the exponential operation of e to an exponential operation of 2 and a shift operation.

Since $v \in (0, 1]$, it is possible to fit the function $f(v) = 2^{-v}$ with a linear function. We use the function $f(v) = kv + b$ to approximate the function [15], where k and b are different when v is in different ranges. By varying the segmentation range, a design with adjustable precision can be achieved. With this fitting method, the implementation of the softmax function can be highly efficient and low-complexity. In the proposed design, two decimal bits of v are chosen as the selection bits, and there are $2^2 = 4$ kinds of corresponding fitting functions, the number of both parameters k and b are also 4. Through this method, the exponential implementation function can be greatly simplified. The basic architecture of the EXP implementation unit is shown in Fig. 8.

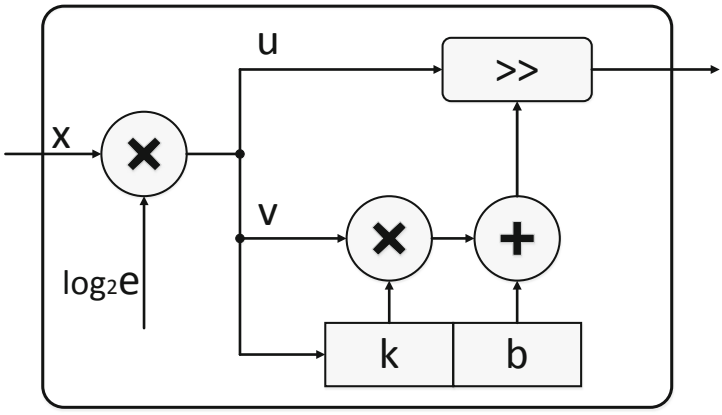

Fig. 8. Exponential unit diagram

Set $S = \sum_{j=1}^{N} e^{x_i - x_{\max}}$. Obviously $\boldsymbol{S} > 1$. Set $m \in (1, 2]$, n is a non-negative integer. Then S can be expressed by the following equation:

$$S = m \cdot 2^n \tag{10}$$

So,

$$\ln S = \ln 2^n + \ln m = \ln 2 \cdot (n + \log_2 m) \tag{11}$$

Powers of 2 can be calculated simply by shifting. The values of m and n then need to be determined. According to the characteristic of binary numbers, it is only necessary to know the first non-zero bit of the binary number $\boldsymbol{S}$, y, and the position of the decimal point, z, then $n = y - z$. Shift $\boldsymbol{S}$ to the right by n bits to get the value of m.

Also using the approximation method to calculate $\log_2 m$, since $m \in (1, 2]$, we can take the approximate function as $f(m) = m-1$. Then

$$\ln S = \ln 2 \cdot (n + m - 1) \tag{12}$$

Again since ln2 is a constant, a modified constant multiplier can be used. In combination with the above modules, the final result of ln$S$ can be calculated. Logarithm calculation module is shown in Fig. 9.

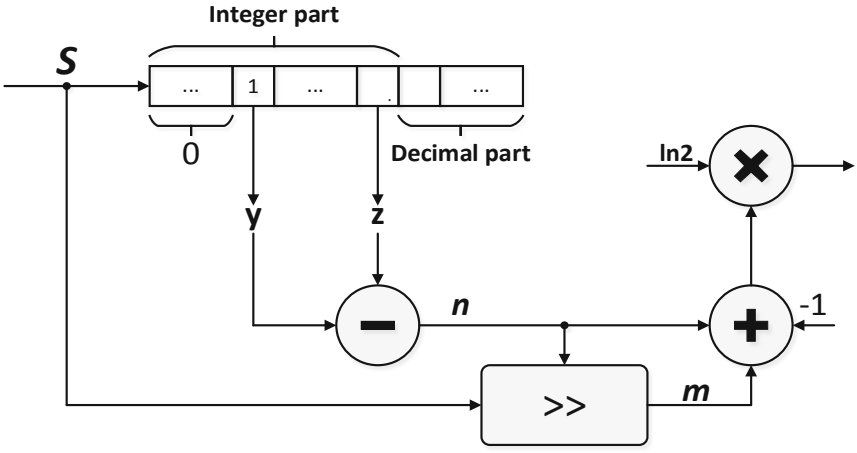

Fig. 9. Logarithm unit diagram

3 Results

Table 1 shows the hardware resource consumption of the direct convolution and Winograd convolution respectively. Both are synthesized in TSMC 28 nm technology by the tool Design Compiler of synopsys and each performs 256 3 × 3 convolutions. As can be seen, the Winograd algorithm leads to significant reduction in gate cell count and area.

Table 1. Hardware resource of convolution

Algorithm	Area (um2)	Cell count
Direct convolution	181639	200152
Winograd convolution	112546	144474

Table 2 compares the resource consumption and performance of the direct calculation architecture with the improved architecture of softmax function, also synthesized in TSMC 28 nm technology by Design Compiler. And Fig. 10 shows the precision comparison of lookup table and the proposed design for exponential function. The proposed design reduces the hardware complexity while maintaining accuracy.

Table 2. Comparison of different softmax function architectures

	Path length	Area (um2)	Cell count
Propose design	0.59	15926	10467
Direct design	1.26	34102	20374

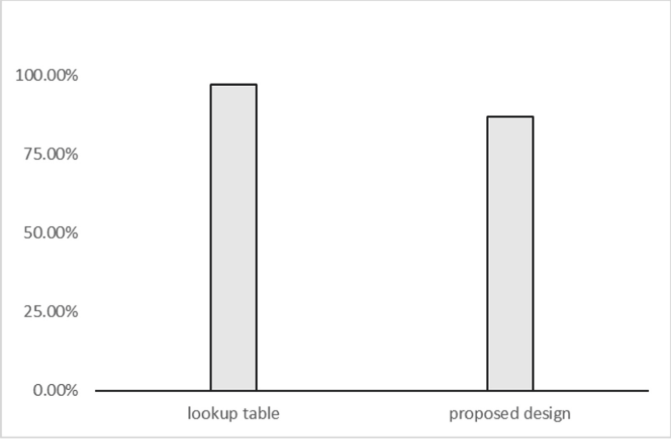

Fig. 10. Accuracy of lookup table and the proposed exponential unit

4 Conclusion

This paper proposed a deep learning accelerator based on DSP. The overall architecture of the accelerator is introduced. Data is pre-processed for the Winograd algorithm in the proposed design, also the MAC array is specially optimized. Softmax function is simplified to reduce the hardware complexity. The design's performance is evaluated. The results show a significant resources reduction and energy efficiency improvement of the proposed design.

Acknowledgement. The authors would like to thank the editors and the reviewers for providing comments and suggestions for this paper. This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61831018, 61901199, and 61631017, and Guangdong Province Key Research and Development Program Major Science and Technology Projects under Grant 2018B010115002.

References

1. Graves, A., Jaitly, N.: Towards end-to-end speech recognition with recurrent neural networks. In: International Conference on Machine Learning, pp. 1764–1772 (2014)
2. Jogin, M., et al.: Feature extraction using convolution neural networks (CNN) and deep learning. In: 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT). IEEE (2020)
3. Wang, Z., Chen, J., Wang, X.: Convolutional neural network for image feature extraction based on concurrent nested inception modules. In: 2019 15th International Conference on Computational Intelligence and Security (CIS). IEEE (2020)
4. Li, R., et al.: A convolutional neural network with mapping layers for hyperspectral image classification. IEEE Trans. Geoence Remote Sens. **58**(5), 3136–3147 (2020)
5. Long, P.M., Sedghi, H.: Size-free generalization bounds for convolutional neural networks (2019)
6. Shao, R., Zhong, S., Yan, L.: ASIC-based architecture for the real-time computation of 2D convolution with large kernel size. In: International Symposium on Multispectral Image Processing and Pattern Recognition International Society for Optics and Photonics (2015)
7. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1–12. IEEE (2017)
8. Ren, H., Zhang, Z., Wu, J.: SWIFT: a computationally-intensive DSP architecture for communication applications. Mobile Networks Appl. **21**(6), 974 (2016)
9. Spagnolo, F., et al.: Designing fast convolutional engines for deep learning applications. In: 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE (2019)
10. Meng, L., Brothers, J.: Efficient Winograd Convolution via Integer Arithmetic (2019)
11. Asgari, B., Hadidi, R., Kim, H.: Proposing a fast and scalable systolic array for matrix multiplication. In: 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE (2020)
12. Wang, Z., Lan, Q., He, H., Zhang, C.: Winograd algorithm for 3D convolution neural networks. In: Lintas, A., Rovetta, S., Verschure, P.F.M.J., Villa, A.E.P. (eds.) ICANN 2017. LNCS, vol. 10614, pp. 609–616. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68612-7_69

13. Yuan, B.: Efficient hardware architecture of softmax layer in deep neural network. In: System-on-chip Conference. IEEE (2017)
14. Zhu, D., et al.: Efficient Precision-adjustable architecture for softmax function in deep learning. IEEE Trans. Circuits Syst. II Express Briefs, **99**, 1 (2020)
15. Farabet, C., et al.: Hardware accelerated convolutional neural networks for synthetic vision systems. In: IEEE International Symposium on Circuits and Systems. IEEE (2010)