algorithms

MDPI

Article

Convolution Accelerator Designs Using Fast Algorithms

Yulin Zhao [1,2,3,*], Donghui Wang [1,2] and Leiou Wang [1,2]

[1] Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; wangdh@mail.ioa.ac.cn (D.W.); wangleiou@mail.ioa.ac.cn (L.W.)
[2] Key Laboratory of Information Technology for Autonomous Underwater Vehicles, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[3] University of Chinese Academy of Sciences, Beijing 100049, China
* Correspondence: zhaoyulin14@mails.ucas.ac.cn

Received: 11 March 2019; Accepted: 21 May 2019; Published: 27 May 2019

Abstract: Convolutional neural networks (CNNs) have achieved great success in image processing. However, the heavy computational burden it imposes makes it difficult for use in embedded applications that have limited power consumption and performance. Although there are many fast convolution algorithms that can reduce the computational complexity, they increase the difficulty of practical implementation. To overcome these difficulties, this paper proposes several convolution accelerator designs using fast algorithms. The designs are based on the field programmable gate array (FPGA) and display a better balance between the digital signal processor (DSP) and the logic resource, while also requiring lower power consumption. The implementation results show that the power consumption of the accelerator design based on the Strassen–Winograd algorithm is 21.3% less than that of conventional accelerators.

Keywords: convolutional neural network; fast convolution; FPGA; Strassen; Winograd

1. Introduction

In recent years, convolutional neural networks (CNNs) have been widely used in computer analysis of visual imagery, automatic driving and other fields because of their high accuracy in image processing [1–4]. Many embedded applications have limited performance and the power consumption [5,6] and CNNs are computationally intensive and consume large amounts of processing time, making it difficult to apply them to these types of applications.

At present, there are two solutions to accelerate the use of CNNs. One is to reduce the computational complexity of the neural network; many such methods have been proposed that maintain the accuracy, including quantification, tailoring, sparsity and fast convolution [7–13]. The other solution is to use a high-performance, low-power hardware accelerator. A graphics processing unit (GPU) is an outstanding processor, but its power consumption is very large [14]. Instead, a field-programmable gate array (FPGA) and the application-specific integrated circuit (ASIC) are energy-efficient integrated circuits. The FPGA has been extensively explored because of its flexibility [15–19].

In general, the convolution operation occupies most of the computation time of the CNN. There are several fast algorithms that can reduce the computational complexity of convolutions without losing accuracy [19,20]. Reference [21] evaluated the Winograd algorithm on FPGAs and proved its efficiency. We propose a faster algorithm based on the Strassen and Winograd algorithms reported in our previously published paper [22]. We propose several convolution accelerator designs using fast algorithms. The implementation results using an FPGA show that the design based on the Strassen–Winograd algorithm can reduce the power consumption by 21.3% compared to the traditional design.

The rest of this paper is organized as follows: Section 2 provides a detailed description of the CNN and fast algorithms, including the Strassen, Winograd and Strassen–Winograd algorithms; Section 3 describes the architecture of the accelerator designs based on fast algorithms; Section 4 describes the implementation results; and Section 5 presents the conclusions of our study.

2. Related Works

2.1. Convolutional Neural Network (CNN)

A CNN generally consists of several layers, each composed of input feature maps, kernels and output feature maps. The convolutional layers carry most of the computation requirements in the network. Figure 1 shows the main process of the convolution where the convolutional layer extracts different features from the input feature maps via different kernels.

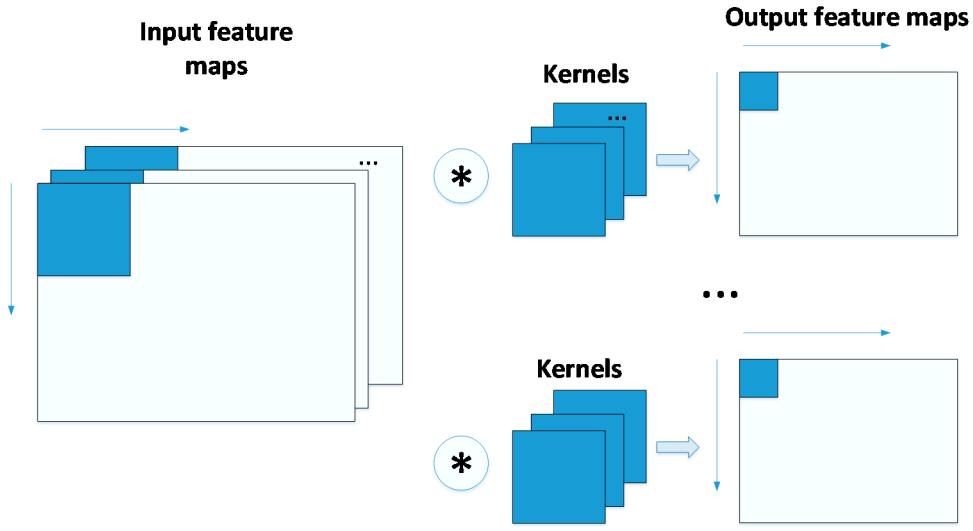

Figure 1. Description of a convolution in CNNs.

The entire convolution can be expressed as Equation (1), where Y denotes the output feature maps, W denotes the kernels of size Mw × Nw and X denotes the input feature maps of size Mx × Nx. The number of input feature maps is Q, and the number of output feature maps is R. The subscripts x and y indicate the position of the pixel in the feature map, while the subscripts u and v indicate the position of the parameter in the kernel.

$$Y_{r,p} = \sum_{q=1}^{Q} \sum_{u=1}^{Mw} \sum_{v=1}^{Nw} W_{r,q,u,v} X_{q,x+u-1,y+v-1}. \quad (1)$$

Equation (1) can be rewritten simply as Equation (2):

$$Y_{r,p} = \sum_{q=1}^{Q} W_{r,q} * X_{q,p}. \quad (2)$$

We can use the special matrices Y, W and X to indicate all of the output feature maps, kernels and input feature maps, respectively. As Equation (4) shows, each element in the matrix denotes a feature map or a kernel. The computation of Y can be expressed as the special matrix convolution in Equation (3). In this case, we can use the fast algorithms to accelerate this special matrix convolution.

$$Y = W * X, \quad (3)$$

$$Y = \begin{bmatrix} Y_{1,1} & \cdots & Y_{1,P} \\ \vdots & \ddots & \vdots \\ Y_{R,1} & \cdots & Y_{R,P} \end{bmatrix} W = \begin{bmatrix} W_{1,1} & \cdots & W_{1,Q} \\ \vdots & \ddots & \vdots \\ W_{R,1} & \cdots & W_{R,Q} \end{bmatrix} X = \begin{bmatrix} X_{1,1} & \cdots & X_{1,P} \\ \vdots & \ddots & \vdots \\ X_{Q,1} & \cdots & X_{Q,P} \end{bmatrix}. \quad (4)$$

2.2. Strassen Algorithm

The Strassen algorithm is a fast way to perform matrix multiplication (a detailed description of the Strassen algorithm is given in Appendix A). For the product of 2×2 matrices, the Strassen algorithm requires 7 multiplications and 18 additions, in contrast to the conventional algorithm, which requires 8 multiplications and 4 additions. Winograd's variant of the Strassen algorithm only needs 15 additions [23], and it achieves relatively good results on GPUs [24].

Moreover, the Strassen algorithm is applicable for the special matrix convolution in Equation (3) [19] and is able to reduce the number of convolutions from eight to seven. The implementation of a 2×2 matrix is shown in Algorithm 1. The function Conv() completes the convolution between one feature map and a kernel.

Algorithm 1. Implementation of the Strassen Algorithm

1 $M1 = Conv(W_{1,1} + W_{2,2}, X_{1,1} + X_{2,2})$
2 $M2 = Conv(W_{2,1} + W_{2,2}, X_{1,1})$
3 $M3 = Conv(W_{1,1}, X_{1,2} - X_{2,2})$
4 $M4 = Conv(W_{2,2}, X_{2,1} - X_{1,1})$
5 $M5 = Conv(W_{1,1} + W_{2,2}, X_{2,2})$
6 $M6 = Conv(W_{2,1} - W_{1,1}, X_{1,1} + X_{1,2})$
7 $M7 = Conv(W_{1,2} - W_{2,2}, X_{2,1} + X_{2,2})$
8 $Y_{1,1} = M1 + M4 - M5 + M7$
9 $Y_{1,2} = M3 + M5$
10 $Y_{2,1} = M2 + M4$
11 $Y_{2,2} = M1 - M2 + M3 + M6$

2.3. Winograd Algorithm

The Winograd minimal filtering algorithm is a fast approach for convolutions. For a convolution between a 4×4 map and a 3×3 kernel, this algorithm reduces the number of multiplications from 36 to 16 (a detailed description of the Strassen algorithm is given in Appendix B) To apply this algorithm to the convolution in Equation (3), we need to divide each input feature map into smaller 4×4 feature maps.

2.4. Strassen–Winograd Algorithm

Our previous work [22] showed that the Strassen and Winograd algorithms can be used together in matrix convolutions. Algorithm 1 provides the implementation of the Strassen algorithm where the function Conv() is the main computation in the entire process. We can apply the Winograd algorithm to the function Conv() and denote this as the Strassen–Winograd algorithm. A brief description is given below in Algorithm 2. The function Winograd() is the convolution using the Winograd algorithm.

Algorithm 2. Implementation of the Strassen–Winograd Algorithm

1 M1 = Winograd($W_{1,1} + W_{2,2}$, $X_{1,1} + X_{2,2}$)
2 M2 = Winograd($W_{2,1} + W_{2,2}$, $X_{1,1}$)
3 M3 = Winograd($W_{1,1}$, $X_{1,2} - X_{2,2}$)
4 M4 = Winograd($W_{2,2}$, $X_{2,1} - X_{1,1}$)
5 M5 = Winograd($W_{1,1} + W_{2,2}$, $X_{2,2}$)
6 M6 = Winograd($W_{2,1} - W_{1,1}$, $X_{1,1} + X_{1,2}$)
7 M7 = Winograd($W_{1,2} - W_{2,2}$, $X_{2,1} + X_{2,2}$)
8 $Y_{1,1}$ = M1 + M4 − M5 + M7
9 $Y_{1,2}$ = M3 + M5
10 $Y_{2,1}$ = M2 + M4
11 $Y_{2,2}$ = M1 − M2 + M3 + M6

3. Architecture Design

In this paper, we propose four convolution accelerator designs based on the conventional, Strassen, Winograd and Strassen–Winograd algorithms. All our designs are for the same matrix convolution with a 3 × 3 kernel, a 224 × 224 input feature map and a 2 × 2 matrix. A detailed description of the architecture is given below.

3.1. Conventional Design

There are eight convolutions between the input feature maps and the kernels for the convolution of a 2 × 2 matrix. Figure 2 shows the architecture of our design with eight convolution modules (ConvNxN) instantiated in parallel. We divide the conventional algorithm into two stages. In Stage 1, input feature maps and kernels are sent to the module ConvNxN, which is designed for the convolution between the feature map and the kernel. In Stage 2, the output results from the convolutions are summarized to obtain the output feature maps.

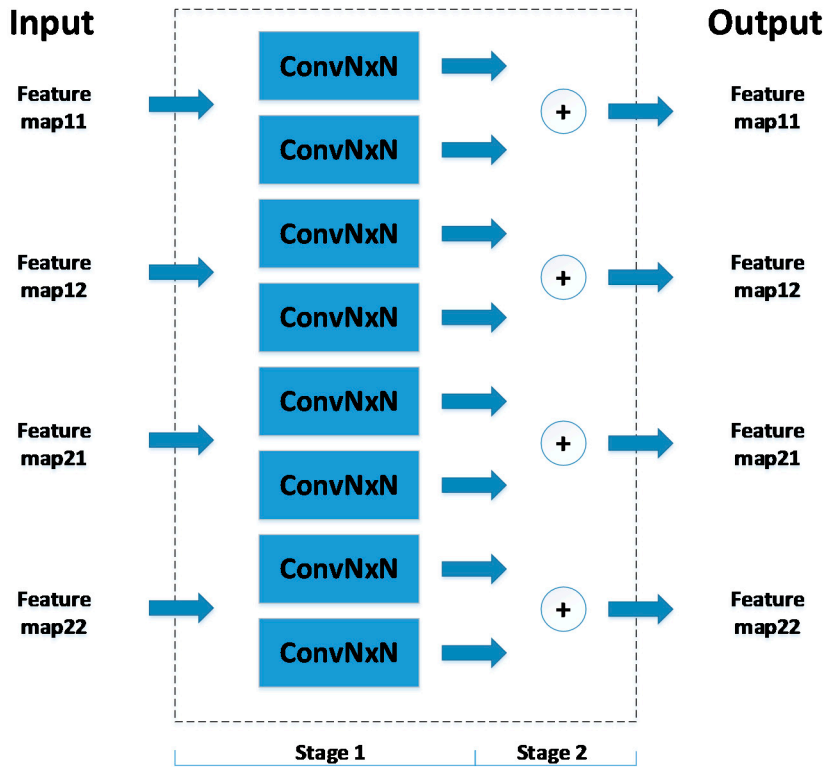

Figure 2. Schematic representation of the architecture for the conventional design.

Figure 3 shows the architecture of the module ConvNxN. Usually, a convolution requires high memory bandwidth. To reduce this bandwidth demand, we designed a data buffer to maximize data reuse and stored all kernels in memory on chip. Four pixels of the feature map are sent into the buffer module, which is an FIFO consisting of a shift registers group, every clock cycle. The width of the FIFO is 4 pixels, and the depth of the FIFO is 112 + 2 pixels. The data buffering process can be divided into three stages as follows:

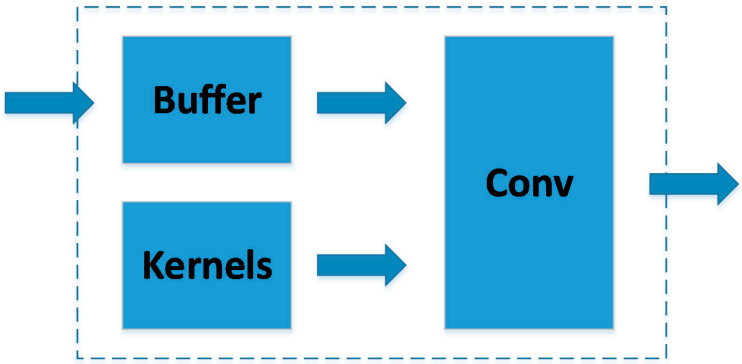

Figure 3. Schematic representation of the ConvNxN architecture.

Stage a: as Figure 4a shows, four pixels are written into the FIFO each clock cycle (the FIFO is not full during this stage). No valid pixels are sent to the Conv module during this clock cycle.

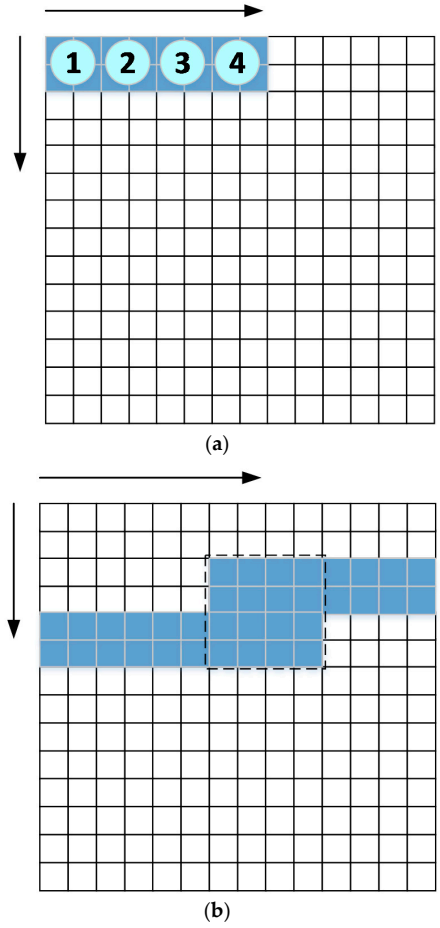

Figure 4. *Cont.*

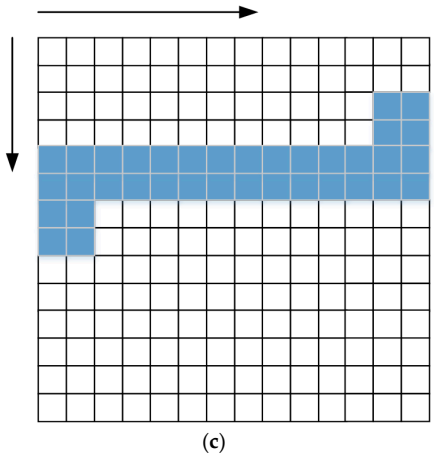

Figure 4. Schematic representation of the input sequence for the ConvNxN module.

Stage b: as Figure 4b shows, four pixels are written into the FIFO, and four pixels are read out simultaneously (the FIFO is full in this stage). The eight earliest written pixels and the eight latest written pixels constitute a group of 4×4 pixels, shown as the dotted box in Figure 4b. This group of pixels is sent to the Conv module for the convolution operation.

Stage c: as Figure 4c shows, the eight earliest written pixels and the eight latest written pixels cannot constitute a group of 4×4 pixels. No valid pixels are sent to the Conv module during this clock cycle.

The Conv module is designed to compute the convolution with Figure 5 showing its data flow, which completes the multiplications and additions in the convolution. The Conv module is designed in the pipeline mode. Four convolutions between the 4×4 input feature map and the 3×3 kernel are executed in parallel in the Conv module. Thus, the process requires $4 \times 3 \times 3 = 36$ multipliers. There are eight ConvNxN modules in Figure 2, which need a total of $8 \times 36 = 288$ multipliers.

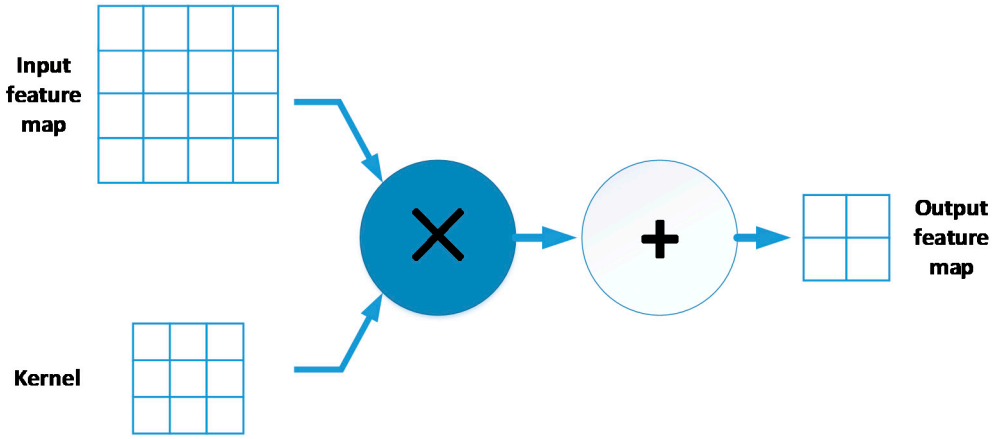

Figure 5. Data flow of the conventional convolution algorithm.

3.2. Strassen Design

The Strassen algorithm can reduce the number of convolutions from eight to seven. The architecture of the matrix convolution based on the Strassen algorithm is shown schematically in Figure 6, which shows the Strassen algorithm can be divided into three stages. In Stage 1, the input data are transformed using the parameter matrix in Equation (A16) before being sent to the ConvNxN module. In Stage 2, only seven ConvNxN modules are instantiated in parallel (the ConvNxN module is the same as in

Figure 2). In Stage 3, to obtain the output feature maps, the output results from the ConvNxN module should be transformed again as Equation (A16) in Appendix A.

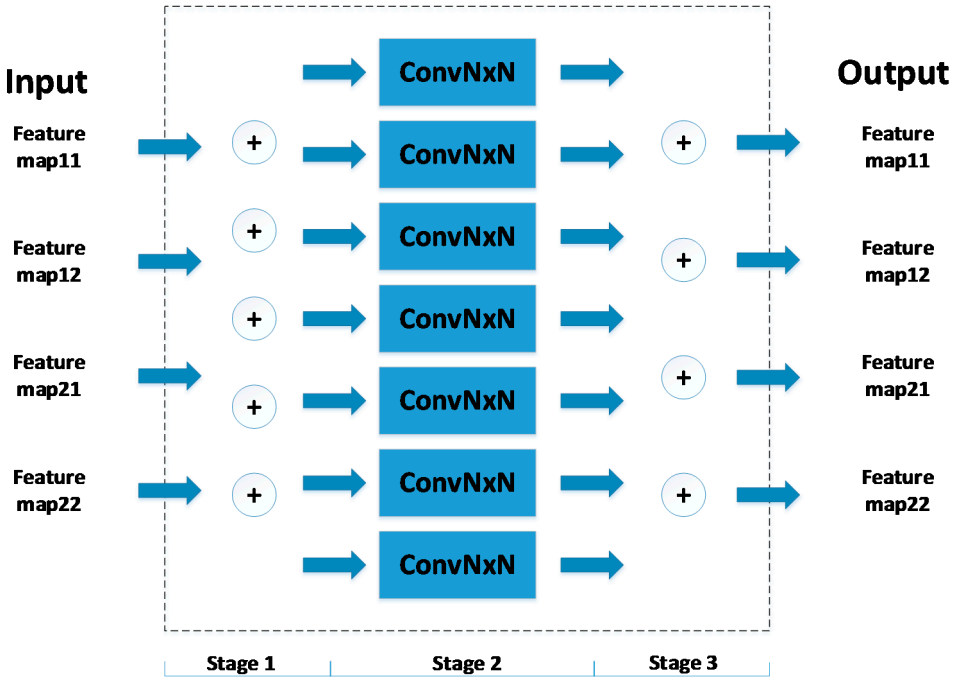

Figure 6. Architecture of the matrix convolution based on the Strassen algorithm.

3.3. Winograd Design

The architecture of the matrix convolution based on the Winograd algorithm is shown schematically in Figure 7. In the figure, the design based on the Winograd algorithm is divided into two stages. In Stage 1, eight convolution modules are needed to complete the convolution between the input feature maps and the kernels. In Stage 2, the output results from the convolutions are summarized to obtain the output feature maps.

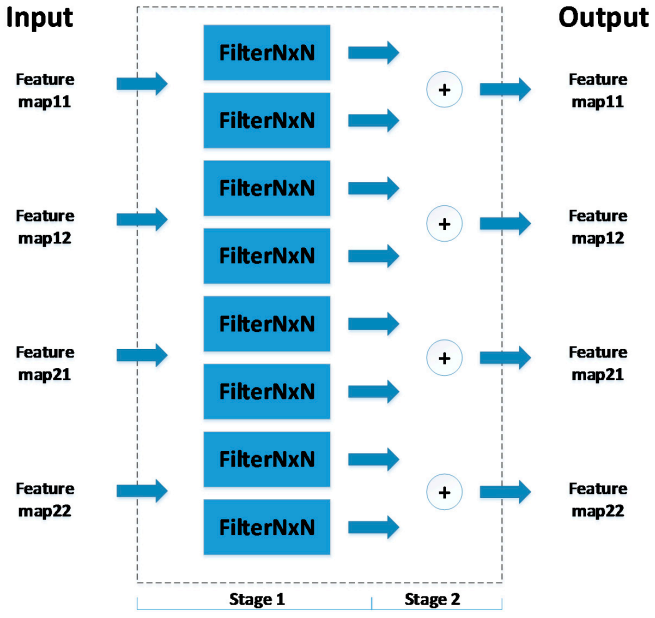

Figure 7. Architecture of the matrix convolution based on the Winograd algorithm.

The FilterNxN module utilizes the Winograd algorithm to complete the convolution. Figure 8 shows the architecture of the FilterNxN module, which adopts the same buffering strategy and module as the ConvNxN module. All the kernels are stored in registers.

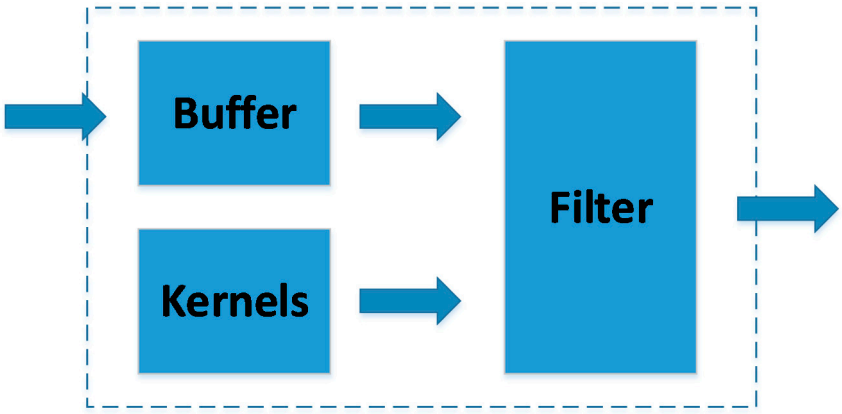

Figure 8. Schematic representation for the architecture of the FilterNxN module.

Figure 9 shows the dataflow of the FilterNxN module. We can see from the figure that the 4×4 data and 3×3 kernel are sent each clock cycle, which are first transformed according to Equation (A26). Element-wise multiplication is performed immediately following the transformation. Finally, the product is transformed using the matrix K and the matrix K^T. We can see from Equation (A26) that 16 multipliers are needed in this module. That is, the design in Figure 7 requires a total of $8 \times 16 = 128$ multipliers.

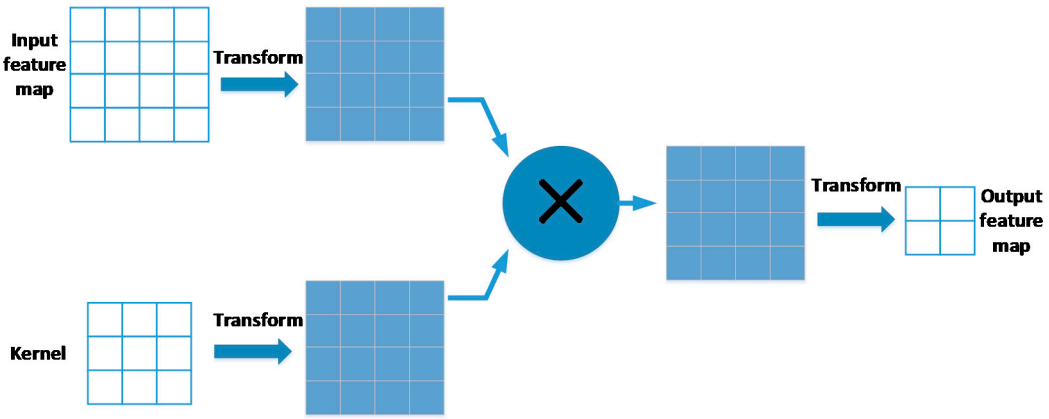

Figure 9. Dataflow of the convolution based on the Winograd algorithm.

3.4. Strassen–Winograd Design

As shown in Section 2.4, the Strassen and Winograd algorithms can be used together in the matrix convolution. The architecture of the convolution based on the Strassen–Winograd algorithm is shown in Figure 10.

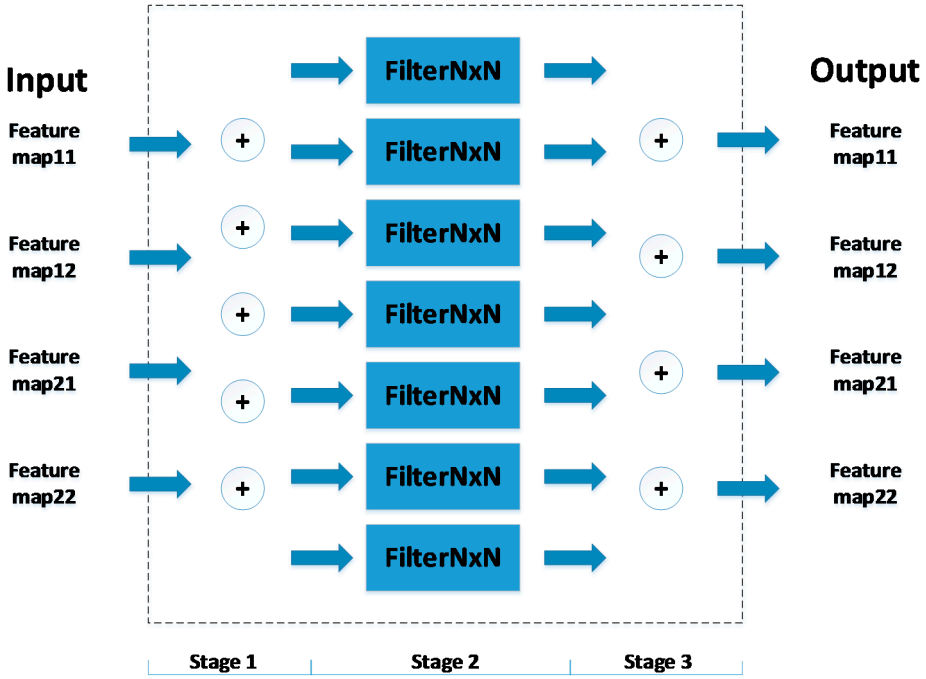

Figure 10. Architecture of the matrix convolution based on the Strassen–Winograd algorithm.

As shown in Figure 10, the design based on the Strassen–Winograd algorithm is divided into three states. In Stage 1, the input data are transformed before being sent to the FilterNxN module. In Stage 2, seven convolution FilterNxN modules are instantiated in parallel. The FilterNxN module is the same as those in Figure 7. The output data should be then transformed in Stage 3 before being sent out. The FilterNxN module needs 16 multipliers; thus the design based on the Strassen–Winograd algorithm requires a total of $7 \times 16 = 112$ multipliers.

4. Implementation Results

All the designs were evaluated using the Xilinx kintex-7 325t FPGA. The designs were simulated and implemented in Vivado 2014.3 using Verilog, all the data used 16-bit fixed-point precision, and the FPGA implementation was performed at 100 Mhz.

We simulated our designs using the Vivado Simulator with the same input feature maps and kernels and recorded the time consumptions to compare their processing performances, as shown in Table 1. The time consumption counts from the first to the last output pixels. We can see from Table 1 that all the designs require almost the same time to complete the same matrix convolution. That is, these designs have the same processing performance.

Table 1. Time consumption records for the different designs.

	Time Consumption
Conventional	125.48 µs
Strassen	125.49 µs
Winograd	125.49 µs
Strassen–Winograd	125.50 µs

Table 2 provides a detailed description of the resource utilizations. The table shows that the design based on the Strassen algorithm uses fewer resources than the conventional design. The design based on the Winograd algorithm requires less than half of the Digital Signal Processors (DSPs) required by the conventional design, but more than the other resources, like registers and look up tables(LUTs). Based on the utilization rate of the resources, the Winograd algorithm is observed to improve the

overall resource utilization. Though more DSPs have been added in recent FPGAs, the DSP is still a limiting resource in most cases, except for the Winograd algorithm. Compared to the Winograd algorithm, the design based on the Strassen–Winograd algorithm requires even fewer resources.

Table 2. Device utilization summary.

	Slice Registers	Slice LUTs	Slices	DSP48E1s
Available	407,600	203,800	50,950	840
Conventional	5780	4635	2248	288
Strassen	5543	4642	2041	252
Winograd	9544	7752	3049	128
Strassen–Winograd	8751	7396	2713	112

Xilinx provides the Vivado Power Analysis for power estimations. It provides accurate estimations because it can read the exact logic and routing resources from the implemented design and presents the power report from different views. The power consumption of the FPGA consists of the static and dynamic power, the latter accounting for most of the total power consumption. Dynamic power consists of the clock power, signal power, logic power and the DSP power. The detailed power consumption is recorded in Table 3 and Figure 11.

Table 3. Power consumption for different designs.

Power (W)	Total	Clocks	Signals	Logic	DSP
Conventional	0.479	0.038	0.098	0.073	0.270
Strassen	0.460	0.035	0.110	0.078	0.237
Winograd	0.381	0.055	0.106	0.099	0.120
Strassen–Winograd	0.377	0.038	0.121	0.113	0.105

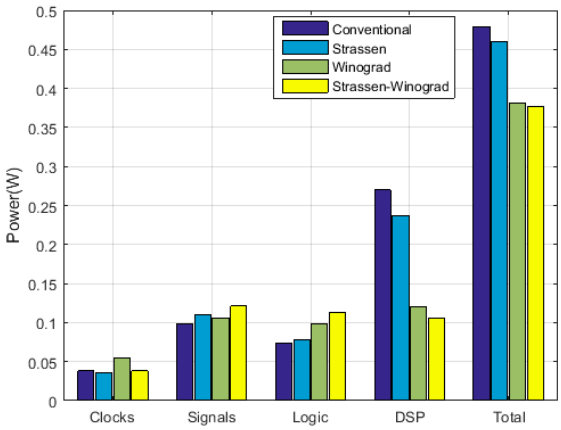

Figure 11. Power consumption of designs based on different algorithms.

Table 3 and Figure 11 give detailed descriptions of the power distribution. We can see from the table that the DSP power occupies a large portion of the dynamic power consumption of the FPGA. The total dynamic power of the design based on the Strassen algorithm is 4% less than that of the conventional design. However, the signal and logic powers of the design based on the Strassen algorithm are higher than those of the conventional design. This is because the Strassen algorithm increases the signal rate. The total dynamic power of the design based on the Winograd algorithm is 20.5% less than that of the conventional design. The total dynamic power of the design based on the Strassen–Winograd algorithm is 21.3% less than that of the conventional design. Similarly, since the Strassen algorithm increases the signal rate, the signal and logic powers of the design based on the Strassen–Winograd algorithm are higher than those of the Winograd algorithm-based design.

We can see from Table 2 that the DSP is still a limiting resource for most designs. The designs in the paper are used for Matrix convolution with a 2×2 matrix. The designs require multiple parallel instances to achieve a high performance. We can calculate the maximum number of instantiations on this FPGA from Table 2, which is 2 for the conventional design, 3 for the Strassen design, 6 for the Winograd design and 7 for the Strassen-Winograd design. Thus, performance of the best Strassen-Winograd design on this FPGA is 3.5 times that of the conventional design, as each one achieved nearly the same processing performance.

5. Conclusions

CNNs have achieved a high accuracy in many image-processing fields, but one major problem is the heavy computational burden, especially for matrix convolutions. Several fast algorithms can reduce the computational complexity, but they incur difficulties in hardware implementation. Therefore, several matrix convolution accelerator designs based on the Strassen, Winograd and Strassen–Winograd algorithms are proposed. The implementation results confirm that the proposed designs improve overall resource utilization and reduce power consumption. Moreover, they increase performance of the best designs that fit on the same FPGA.

Author Contributions: Conceptualization, Y.Z.; Validation, Y.Z.; Writing–review & editing, D.W. and L.W.

Funding: This work was supported in part by the National Natural Science Foundation of China under Granted 61801469, and in part by the Young Talent Program of Institute of Acoustics, Chinese Academy of Science, under Granted QNYC201622.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

This section describes the mathematical theory behind the Strassen algorithm.

Suppose there are two 2×2 matrices A and B, and matrix C is the product of A and B. The computation of matrix C is given by Equation (A2).

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}; B = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}; C = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}, \tag{A1}$$

$$C = A \times B = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix}. \tag{A2}$$

As shown in Equation (A2), the entire process needs eight multiplications and four additions. When using the Strassen algorithm, the number of multiplications can be reduced from eight to seven [25]. The process is shown as follows:

$$m_1 = (a_{1,1} + a_{2,2}) \times (b_{1,1} + b_{2,2}), \tag{A3}$$

$$m_2 = (a_{2,1} + a_{2,2}) \times b_{1,1}, \tag{A4}$$

$$m_3 = a_{1,1} \times (b_{1,2} - b_{2,2}), \tag{A5}$$

$$m_4 = a_{2,2} \times (b_{2,1} - b_{1,1}), \tag{A6}$$

$$m_5 = (a_{1,1} + a_{2,2}) \times b_{2,2}, \tag{A7}$$

$$m_6 = (a_{2,1} - a_{1,1}) \times (b_{1,1} + b_{1,2}), \tag{A8}$$

$$m_7 = (a_{1,2} - a_{2,2}) \times (b_{2,1} + b_{2,2}), \tag{A9}$$

$$c_{1,1} = m_1 + m_4 - m_5 + m_7, \tag{A10}$$

$$c_{1,2} = m_3 + m_5, \tag{A11}$$

$$c_{2,1} = m_2 + m_4, \tag{A12}$$

$$c_{2,2} = m_1 - m_2 + m_3 + m_6, \tag{A13}$$

where m_1, m_2, m_3, m_4, m_5, m_6, and m_7 are the seven temporary variables. The algorithm is effective as long as there are an even number of rows and columns in the matrix [19].

If we expand matrices A, B, and C into vector form, the entire process of this algorithm can be expressed as Equation (A14).

$$\begin{bmatrix} c_{1,1} \\ c_{1,2} \\ c_{2,1} \\ c_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \left(\left(\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} a_{1,1} \\ a_{1,2} \\ a_{2,1} \\ a_{2,2} \end{bmatrix} \right) \bullet \left(\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{2,1} \\ b_{2,2} \end{bmatrix} \right) \right) \tag{A14}$$

We denote the first, second and third parameter matrices in Equation (A14) as matrices E, G, and D, respectively. Equation (A14) can be rewritten as Equation (A16), where the • indicates element-wise multiplication. We can regard matrices G, D, and E as transform parameter matrices. The Strassen algorithm can be expressed as follows. First, matrices G and D are used to transform vectors A and B. Then, element-wise multiplication is performed. Finally, matrix E is used to transform the product.

$$E = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}, G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, D = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{A15}$$

$$\begin{bmatrix} c_{1,1} \\ c_{1,2} \\ c_{2,1} \\ c_{2,2} \end{bmatrix} = E \left(\left(G \begin{bmatrix} a_{1,1} \\ a_{1,2} \\ a_{2,1} \\ a_{2,2} \end{bmatrix} \right) \bullet \left(D \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{2,1} \\ b_{2,2} \end{bmatrix} \right) \right) \tag{A16}$$

Appendix B

This section explains the mathematical theory behind Winograd's minimal filtering algorithm.

First, we introduce the Winograd algorithm with a one-dimensional vector convolution. We denote a three-tap Finite Impulse Response (FIR) filter with two outputs as F(2,3). The input data are x1, x2, x3, and x4 and the parameters of the filter are w1, w2, w3. The conventional algorithm for F(2,3) is given by Equation (A17).

$$F(2,3) = \begin{bmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \\ x_2 w_1 + x_3 w_2 + x_4 w_3 \end{bmatrix}. \tag{A17}$$

The process of the minimal filtering algorithm is given by Equations (A18)–(A22) [26]:

$$m_1 = (x_1 - x_3) w_1, \tag{A18}$$

$$m_2 = (x_2 + x_3) \frac{1}{2} (w_1 + w_2 + w_3), \tag{A19}$$

$$m_3 = (x_3 - x_2) \frac{1}{2} (w_1 - w_2 + w_3), \tag{A20}$$

$$m_4 = (x_2 - x_4) w_3, \tag{A21}$$

$$F(2,3) = \begin{bmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \\ x_2 w_1 + x_3 w_2 + x_4 w_3 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}. \tag{A22}$$

We see from the process that only four multiplications are needed. The entire process can be written in matrix form as Equation (A23).

$$F(2,3) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \left(\left(\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \right) \bullet \left(\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) \right) \tag{A23}$$

We denote the first, second and third parameter matrices in Equation (A23) as K, L, and O, respectively. Equation (A23) can be rewritten as Equation (A24).

$$F(2,3) = K \left(\left(L \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \right) \bullet \left(O \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) \right), \tag{A24}$$

$$K = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}, L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}, O = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \tag{A25}$$

The convolution of the two-dimensional image can be generalized with a filter as Equation (A26) [20].

$$F(2 \times 2, 3 \times 3) = K \left(\left(L \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} L^T \right) \bullet \left(O \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} O^T \right) \right) K^T \tag{A26}$$

where the superscript T indicates the transpose operator. We see from Equation (A26) that calculating the convolution between data sized at 4×4 and a 3×3 kernel requires using matrix L and matrix L^T to transform the kernel and the use of matrix O and matrix O^T to transform the data before element-wise multiplication. Finally, we use matrix K and matrix K^T to transform the product. Compared with the conventional algorithm, this algorithm can reduce the number of multiplications from 36 to 16.

References

1. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**. Available online: https://arxiv.org/pdf/1409.1556.pdf (accessed on 27 May 2019).
2. Szegedy, C.; Liu, W.; Jia, Y. Going deeper with convolutions. *arXiv* **2014**. Available online: https://arxiv.org/pdf/1409.4842.pdf (accessed on 27 May 2019).
3. Liu, N.; Wan, L.; Zhang, Y.; Zhou, T.; Huo, H.; Fang, T. Exploiting Convolutional Neural Networks With Deeply Local Description For Remote Sensing Image Classification. *IEEE Access* **2018**, *6*, 11215–11228. [CrossRef]
4. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; Volume 60, pp. 1097–1105.
5. Le, N.M.; Granger, E.; Kiran, M. A comparison of CNN-based face and head detectors for real-time video surveillance applications. In Proceedings of the Seventh International Conference on Image Processing Theory, Tools and Applications, Montreal, QC, Canada, 28 November–1 December 2018.
6. Ren, S.; He, K.; Girshick, R. Faster R-CNN: Towards real-time object detection with region proposal networks. In Proceedings of the International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 91–99.

7. Denil, M.; Shakibi, B.; Dinh, L. Predicting Parameters in Deep Learning. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 2148–2156.
8. Han, S.; Pool, J.; Tran, J. Learning Both Weights and Connections for Efficient Neural Networks. In Proceedings of the International Conference on Neural Information Processing Systems, Istanbul, Turkey, 9–12 November 2015; pp. 1135–1143.
9. Guo, Y.; Yao, A.; Chen, Y. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2016; pp. 1379–1387.
10. Colangelo, P.; Nasiri, N.; Nurvitadhi, E.; Mishra, A.; Margala, M.; Nealis, K. Exploration of Low Numeric Precision Deep Learning Inference Using Intel® FPGAs. In Proceedings of the IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines, Boulder, CO, USA, 29 April–1 May 2018.
11. Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; Narayanan, P. Deep Learning with Limited Numerical Precision. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015.
12. Rastegari, M.; Ordonez, V.; Redmon, J. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016; pp. 525–542.
13. Zhu, C.; Han, S.; Mao, H. Trained Ternary Quantization. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
14. Vasilache, N.; Johnson, J.; Mathieu, M. Fast convolutional nets with fbfft: A GPU performance evaluation. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
15. Qiu, J.; Wang, J.; Yao, S. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays ACM, Monterey, CA, USA, 21–23 February 2016; pp. 26–35.
16. Courbariaux, M.; Bengio, Y.; David, J. Training deep neural networks with low precision multiplications. *arXiv* **2015**, arXiv:1412.7024.
17. Hajduk, Z. High accuracy FPGA activation function implementation for neural networks. *Neurocomputing* **2017**, *247*, 59–61. [CrossRef]
18. Bai, L.; Zhao, Y.; Huang, X. A CNN Accelerator on FPGA Using Depthwise Separable Convolution. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 1415–1419. [CrossRef]
19. Cong, J.; Xiao, B. Minimizing Computation in Convolutional Neural Networks. In Proceedings of the Artificial Neural Networks and Machine Learning—ICANN 2014, Hamburg, Germany, 15–19 September 2014.
20. Lavin, A.; Gray, S. Fast Algorithms for Convolutional Neural Networks. In Proceedings of the Computer Vision and Pattern Recognition, Caesars Palace, NV, USA, 26 June–1 July 2016; pp. 4013–4021.
21. Lu, L.; Liang, Y.; Xiao, Q.; Yan, S. Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs. In Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines, Napa, CA, USA, 30 April–2 June 2017; pp. 101–108.
22. Zhao, Y.; Wang, D.; Wang, L.; Liu, P. A Faster Algorithm for Reducing the Computational Complexity of Convolutional Neural Networks. *Algorithms* **2018**, *11*, 159. [CrossRef]
23. Coppersmith, D.; Winograd, S. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* **1990**, *9*, 251–280. [CrossRef]
24. Lai, P.; Arafat, H.; Elango, V.; Sadayappan, P. Accelerating Strassen-Winograd's matrix multiplication algorithm on GPUs. In Proceedings of the International Conference on High Performance Computing, data, and analytics, Karnataka, India, 18–21 December 2013.
25. Strassen, V. Gaussian elimination is not optimal. *Numer. Math.* **1969**, *13*, 354–356. [CrossRef]
26. Winograd, S. *Arithmetic Complexity of Computations*; SIAM: Philadelphia, PA, USA, 1980.