applied sciences

MDPI

Article

VLSI-Friendly Filtering Algorithms for Deep Neural Networks

Aleksandr Cariow, Janusz P. Papliński * and Marta Makowska

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology in Szczecin, Żołnierska 49, 71-210 Szczecin, Poland; acariow@wi.zut.edu.pl (A.C.)
* Correspondence: janusz.paplinski@zut.edu.pl

Abstract: The paper introduces a range of efficient algorithmic solutions for implementing the fundamental filtering operation in convolutional layers of convolutional neural networks on fully parallel hardware. Specifically, these operations involve computing M inner products between neighbouring vectors generated by a sliding time window from the input data stream and an M-tap finite impulse response filter. By leveraging the factorisation of the Hankel matrix, we have successfully reduced the multiplicative complexity of the matrix-vector product calculation. This approach has been applied to develop fully parallel and resource-efficient algorithms for M values of 3, 5, 7, and 9. The fully parallel hardware implementation of our proposed algorithms achieves approximately a 30% reduction in embedded multipliers compared to the naive calculation methods.

Keywords: filtering algorithms; deep neural networks; very large-scale integration; multiplicative complexity

check for updates

Citation: Cariow, A.; Papliński, J.P.; Makowska, M. VLSI-Friendly Filtering Algorithms for Deep Neural Networks. *Appl. Sci.* **2023**, *13*, 9004. https://doi.org/10.3390/app13159004

Academic Editors: Panagiotis G. Asteris and Alessandro Lo Schiavo

Received: 5 July 2023
Revised: 31 July 2023
Accepted: 4 August 2023
Published: 6 August 2023

1. Introduction

The need for high-speed processing of large amounts of information stimulates the development and use of highly effective data processing systems. In such systems, the primary requirement for implementing computing methods is to minimise the time of data processing, ensuring the ability to fulfil the planned task within the allocated time for this application. This requirement is especially relevant in the implementation of algorithms for processing digital information in deep neural networks (DNNs) [1–5]. As is known, in deep neural networks, the primary and time-consuming operation is digital convolution. The need to quickly calculate digital convolution arises in both convolutional and capsule neural networks. Digital convolution calculations can be accelerated by algorithmic and hardware methods. In general, algorithmic methods primarily focus on minimising the number of arithmetic operations involved. One widely employed strategy for reducing the computational complexity of the digital convolution operation is utilising the Fast Fourier Transform (FFT) algorithm. This approach has found application in some deep neural networks [6–11]. However, modern convolutional and capsule neural networks use small filters more often than the traditionally used large filters computed using the FFT approach. The Winograd's minimal filtering algorithm [1,12–15], which has recently gained significant popularity, is widely regarded as well-suited for such scenarios. This approach exhibits enhanced efficiency, specifically when employing small filters and tile sizes. In such cases, it performs linear convolution with minimal computational complexity. Indeed, this method calculates the dot products of adjacent vectors obtained from a sliding time window in the current data stream. It employs a third-order finite impulse response filter (FIR) for this purpose.

2. State of the Art

Since we are talking mainly about convolutional and capsular neural networks, it is clear that linear convolution is the main operation in their implementation. In general, convolutional layers tend to be the most time-intensive component, often accounting for over half of the total computation time in a typical implementation [16,17]. The convolution itself is also a time-consuming operation. For this reason, deep neural network builders are looking for and creating efficient ways to minimise convolution computation complexity [11].

Another opportunity to speed up calculations in deep neural networks is by utilising high-performance field-programmable gate arrays (FPGAs) [18–33], graphics processing units (GPUs) [33–36], and specialised application-specific integrated circuits (ASICs) [33,37–39].

When modern stationary data processing systems possess ample computing power, the design of mobile on-board systems powered by batteries encounters various conflicting factors that hinder peak performance. The conventional approach of parallelising computations to enhance data processing speed increases the data processing unit's size, weight, and power consumption. Consequently, there is a necessity for solutions that effectively leverage computational parallelisation while concurrently optimising hardware costs.

Extensive research is being conducted on algorithms and structures for high-performance computing devices intended to process digital signals and images with practical applications in embedded systems. Developing micro-miniature processing units tailored for image processing and recognition in on-board mobile neural networks, known as Tiny ML or Edge AI, is of particular interest. These cutting-edge applications are at the forefront of modern technology.

For these reasons, the Tiny ML Summit has been held since 2019, bringing together experts from major companies and universities. The primary focus of this conference is to discuss the potential of transitioning machine learning from high-performance mainframes to small battery-powered signal microprocessors. The Tiny ML concept is continually evolving, driven by the development of dedicated chips designed for these applications. Notably, digital signal processing algorithms are pivotal in the systems under discussion. Over time, numerous algorithms and processor structures have been developed to address the challenges posed by these systems. With a focus on flexibility and versatility, the prevailing approach involves using universal signal microprocessors and FPGAs.

However, the high flexibility of the developed processors contradicts a highly efficient implementation. For example, a programmable signal processing unit is flexible, scalable and upgradable but highly inefficient in terms of performance, die area, weight, and power consumption.

Therefore, developing ASIC-centric solutions is best suited for portable applications as minimising power consumption, weight, and size of the processing unit in battery-powered systems has become an essential aspect of on-board processing.

At the algorithmic level, methods for reducing the above parameters usually focus on minimising the number of arithmetic operations, especially multiplications. In this regard, developing algorithms for performing the main filtering operations, characterised by minimal multiplicative complexity, is an urgent task. So, we again emphasise that convolution calculation is an essential mathematical macro operation in DNNs. And usually (though not always) it is computed using Winograd's minimum filtering algorithm [1,12,14,18,20–22,40–42]. However, since, as already noted [43], this algorithm can only calculate two adjacent dot products, it is not suitable for all possible situations that can arise in neural networks. For example, Winograd's minimum filtering algorithm is redundant for $M = 3$ and tile size (5×5) or for $M = 5$ and tile size (9×9). Many other examples could be given. In this article, we present algorithmic solutions for FIR filters with short-length impulse responses, which can be more efficient in some cases than Winograd's minimal filtering algorithm.

3. Preliminary Remarks

The primary step in computing a 2D convolution involves taking the dot product between the vectors created by the sliding time window from the present data stream and the impulse response of an M-order finite impulse response (FIR) filter.

The procedure for computing convolution elements can be represented as follows in the most general case:

$$y_j = \sum_{i=0}^{M-1} x_{i+j} w_i \tag{1}$$

$$j = 0, 1, \ldots, N - M + 1,$$

where N represents the length of the current data stream, with $\{x_{i+j}\}$ denoting the elements of the data stream, and $\{w_i\}$ represents the constant coefficients of the FIR filter's impulse response.

In a more detailed form, expression (1) can be represented as follows:

$$\begin{gathered} y_0 = w_0 x_0 + w_1 x_1 + \cdots + w_{M-1} x_{M-1} \\ y_1 = w_0 x_1 + w_1 x_2 + \cdots + w_{M-1} x_M \\ \vdots \\ y_{N-M} = w_0 x_{N-M} + w_1 x_{N-M+1} + \cdots + w_{M-1} x_{N-1} \end{gathered} \tag{2}$$

Figure 1 illustrates the sequence of steps in calculating the moving dot product.

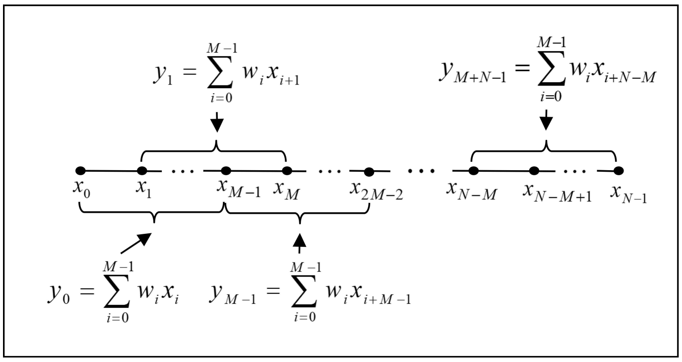

Figure 1. The illustration of the step sequences during calculating the moving dot product.

The equations above comprehensively describe all the mathematical operations required for the calculations. However, strictly speaking, they do not constitute an algorithm since they do not reveal the specific sequence of calculations. In some instances, expressing the sliding dot product operation as a matrix-vector product is more convenient:

$$\mathbf{Y}_{(N-M+1)\times 1} = \mathbf{W}_{(N-M+1)\times N} \mathbf{X}_{N\times 1} \tag{3}$$

where:

$$\mathbf{Y}_{(N-M+1)\times 1} = [y_0, y_1, \ldots, y_{N-M}]^{\mathrm{T}},$$

$$\mathbf{X}_{N\times 1} = [x_0, x_1, \ldots, x_{N-1}]^{\mathrm{T}},$$

$$\mathbf{W}_{(N-M+1)\times N} = \begin{bmatrix} w_0 & w_1 & \cdots & w_{M-1} & & & & \\ & w_0 & w_1 & \cdots & w_{M-1} & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & w_0 & w_1 & \cdots & w_{M-1} \end{bmatrix}.$$

However, regarding the research task at hand, such a representation is unhelpful as it does not facilitate identifying opportunities for reducing the computational complexity of the procedure for determining the sliding inner product when the sequence is comprised of input signal samples.

Let us rewrite expressions (2) in the following form:

$$\mathbf{Y}_{(N-M+1)\times 1} = \mathbf{X}_{(N-M+1)\times M}\mathbf{W}_{M\times 1} \tag{4}$$

where:

$$\mathbf{W}_{M\times 1} = [w_0, w_1, \dots, w_{M-1}]^{\mathrm{T}},$$

$$\mathbf{X}_{(N-M+1)\times M} = \begin{bmatrix} x_0 & x_1 & \cdots & x_{M-1} \\ x_1 & x_2 & \cdots & x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-M} & x_{N-M+1} & \cdots & x_{N-1} \end{bmatrix}.$$

This form of writing is much more useful and legible, which will be visible in the next steps. It turns out that considering the structural properties of the matrix $\mathbf{X}_{(N-M+1)\times M}$ in the expression (4) allows for a fairly significant reduction in the number of arithmetic operations. Let us consider this problem in more detail. Let us impose certain conditions on the sizes of the input sequences. Suppose $N = M(K+1) - 1$, where $K = 1, 2, 3, \dots$ is a positive integer number. It is obvious that if the supposed requirement for N is not satisfied, sequences $\{x_n\}, n = 0, 1, \dots, N-1$ can be padded with zeros without losing computation precision.

Then the expression (4) takes the following form:

$$\mathbf{Y}_{KM\times 1} = \mathbf{X}_{KM\times M}\mathbf{W}_{M\times 1} \tag{5}$$

where:

$$\mathbf{Y}_{KM\times 1} = [y_0, y_1, \dots, y_{KM-1}]^{\mathrm{T}},$$

$$\mathbf{X}_{KM\times M} = \left[\mathbf{X}^{(0)}_{1\times M}, \mathbf{X}^{(1)}_{1\times M}, \dots, \mathbf{X}^{(KM-1)}_{1\times M}\right]^{\mathrm{T}},$$

and

$$\mathbf{X}^{(i)}_{1\times M} = [x_i, x_{i+1}, \dots, x_{M-1+i}], \quad i = 0, 1, \dots, KM-1.$$

To see the structure of the matrix $\mathbf{X}_{KM\times M}$, we present expression (5) in a more detailed form:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{KM-1} \end{bmatrix} = \left[\begin{array}{cccc} x_0 & x_1 & \cdots & x_{M-1} \\ x_1 & x_2 & \cdots & x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_{M-1} & x_M & \cdots & x_{2M-2} \\ \hline x_M & x_{M+1} & \cdots & x_{2M-1} \\ x_{M+1} & x_{M+2} & \cdots & x_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{2M-1} & x_{2M} & \cdots & x_{3M-2} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline x_{(K-1)M} & x_{(K-1)M+1} & \cdots & x_{KM-1} \\ x_{(K-1)M+1} & x_{(K-1)M+2} & \cdots & x_{KM} \\ \vdots & \vdots & \ddots & \vdots \\ x_{KM-1} & x_{KM} & \cdots & x_{(K-1)M-2} \end{array}\right] \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{bmatrix} \tag{6}$$

By examining this description, it becomes evident that the matrix $\mathbf{X}_{KM\times M}$ possesses a block structure and comprises K submatrices of Hankel type. Thus, the calculation of the sliding dot product, taking into account the imposed conditions, comes down to multiplying the sequence of K sub-matrices (i.e. the Hankel matrices) by the vector $\mathbf{W}_{M\times 1}$ and then combining the individual calculation results.

Hence, we establish the fundamental filtering operation in DNNs as multiplying the Hankel sub-matrix (formed from the current input data sequence using a sliding window of size M) by a vector whose elements are the impulse response coefficients of an M-order FIR filter. There are efficient algorithms for multiplying Hankel matrices by a vector. However, they are mainly focused on large matrices, the order of which is a power of two [44,45]. But in most cases of image processing in neural networks, the impulse responses of FIR filters are short and contain an odd number of coefficients. Under these conditions, we are dealing with Hankel matrices of small orders. When multiplying small-size Hankel matrices by small-length vectors, known algorithms are inefficient or even counterproductive. Therefore, we have developed our own algorithms explicitly focused on calculating matrix-vector products with Hankel matrices of small orders.

So, we define the basic filtering macrooperation as:

$$\mathbf{Y}_{M\times 1} = \mathbf{X}_M \mathbf{W}_{M\times 1} \tag{7}$$

where:

$$\mathbf{X}_M = \begin{bmatrix} x_0 & x_1 & \cdots & x_{M-1} \\ x_1 & x_2 & \cdots & x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_{M-1} & x_M & \cdots & x_{2M-2} \end{bmatrix},$$

$$\mathbf{Y}_{M\times 1} = \left[y_0^{(M)}, y_1^{(M)}, \ldots, y_{M-1}^{(M)}\right]^{\mathrm{T}},$$

$$\mathbf{W}_{M\times 1} = \left[w_0^{(M)}, w_1^{(M)}, \ldots, w_{M-1}^{(M)}\right]^{\mathrm{T}}.$$

(Kindly take note that from this point forward, the superscript M will signify quantities associated with the basic filtering macrooperation employing an M-order filter).

We emphasise once again that, as a rule, small-order filters are used in deep neural networks when the impulse response vectors contain a small number of elements. And almost always, we are dealing with an odd number of records. Based on the information provided above, this article aims to create and describe resource-efficient filtering algorithms for FIR filters with widely used orders: M = 3, 5, 7, and 9.

4. Minimal Filtering Algorithms

Let us show, based on specific examples, how it works.

4.1. Algorithm 1, M = 3

Let $\mathbf{X}_{5\times 1} = [x_0, x_1, x_2, x_3, x_4]^{\mathrm{T}}$ be a vector that represents the input data set, $\mathbf{W}_{3\times 1} = \left[w_0^{(3)},\ w_1^{(3)}, w_2^{(3)}\right]^{\mathrm{T}}$ be a vector that contains the coefficients of the impulse response of a 3-tap FIR filter, and $\mathbf{Y}_{3\times 1} = \left[y_0^{(3)}, y_1^{(3)}, y_2^{(3)}\right]^{\mathrm{T}}$ be a vector describing the results of using a 3-tap FIR filter:

$$\mathbf{Y}_{3\times 1} = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} w_0^{(3)} \\ w_1^{(3)} \\ w_2^{(3)} \end{bmatrix} \tag{8}$$

As can be seen, calculating the product (8) requires 9 multiplications and 6 additions.

We can formulate a streamlined algorithm for computing $\mathbf{Y}_{3\times 1}$ by utilising the following matrix-vector calculation procedure:

$$\mathbf{Y}_{3\times 1} = \mathbf{T}_{3\times 6}^{(3)} \mathbf{D}_6^{(3)} \mathbf{T}_{6\times 5}^{(3)} \mathbf{X}_{5\times 1} \tag{9}$$

where

$$\mathbf{T}_{3\times6}^{(3)} = \left[\begin{array}{ccc:ccc} 1 & 1 & & 1 & & \\ & 1 & 1 & & 1 & \\ & & & 1 & 1 & 1 \end{array}\right],$$

$$\mathbf{T}_{6\times5}^{(3)} = \left[\begin{array}{ccccc} 1 & -1 & -1 & & \\ & 1 & & & \\ & -1 & 1 & -1 & \\ \hdashline & & 1 & & \\ & & & 1 & \\ & & -1 & -1 & 1 \end{array}\right],$$

and

$$\mathbf{D}_6^{(3)} = diag\left(s_0^{(3)}, s_1^{(3)}, \ldots, s_5^{(3)}\right),$$

$$s_0^{(3)}, = w_0^{(3)}, \quad s_1^{(3)}, = w_0^{(3)} + w_1^{(3)}, \quad s_2^{(3)}, = w_1^{(3)}$$

$$s_3^{(3)} = w_0^{(3)} + w_2^{(3)}, \quad s_4^{(3)} = w_1^{(3)} + w_2^{(3)}, \quad s_5^{(3)} = w_2^{(3)}.$$

Figure 2 depicts a data flow graph of the proposed algorithm for implementing the basic filtering operation for a 3-tap FIR filter. In this paper, the data flow diagrams are arranged from left to right, and straight lines in the figures represent data transfer operations. The circles in these figures indicate multiplication operations, with the corresponding numbers written inside the circles. The convergence points of the lines indicate summation, while dashed lines represent data transfer operations with a simultaneous change of sign. To maintain clarity, the figures utilise simple lines without arrows. Furthermore, to simplify the presentation, the superscripts of variables have been omitted in all figures, as the vector sizes involved in each case can be inferred from the figures themselves.

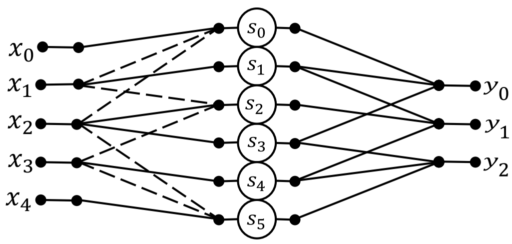

Figure 2. Data flow graph of the algorithm for implementing the basic filtering operation for the case $M = 3$.

4.2. Algorithm 2, M = 5

Let $\mathbf{X}_{9\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]^{\mathrm{T}}$ be a vector that represents the input data set, $\mathbf{W}_{5\times1} = \left[w_0^{(5)}, w_1^{(5)}, w_2^{(5)}, w_3^{(5)}, w_4^{(5)}\right]^{\mathrm{T}}$ be a vector that contains the coefficients of the impulse response of a 5-tap FIR filter, and $\mathbf{Y}_{5\times1} = \left[y_0^{(5)}, y_1^{(5)}, y_2^{(5)}, y_3^{(5)}, y_4^{(5)}\right]^{\mathrm{T}}$ be a vector describing the results of using a 5-tap FIR filter:

$$\mathbf{Y}_{5\times1} = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ x_2 & x_3 & x_4 & x_5 & x_6 \\ x_3 & x_4 & x_5 & x_6 & x_7 \\ x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix} \begin{bmatrix} w_0^{(5)} \\ w_1^{(5)} \\ w_2^{(5)} \\ w_3^{(5)} \\ w_4^{(5)} \end{bmatrix} \tag{10}$$

As can be seen, calculating the product (10) requires 25 multiplications and 20 additions.

We can devise a streamlined algorithm to compute $\mathbf{Y}_{5\times1}$ by employing the following matrix-vector calculation procedure:

$$\mathbf{Y}_{5\times1} = \mathbf{T}^{(5)}_{5\times14}\mathbf{D}^{(5)}_{14}\mathbf{T}^{(5)}_{14\times9}\mathbf{X}_{9\times1} \tag{11}$$

where

$$\mathbf{T}^{(5)}_{5\times14} = \begin{bmatrix} 1 & 1 & 1 & 1 & & 1 & & & & & & & & \\ & 1 & & 1 & 1 & & 1 & & & 1 & & & & \\ & & 1 & 1 & & & & 1 & 1 & & & 1 & & \\ & & & 1 & & & 1 & 1 & & & 1 & & 1 & \\ & & & & & 1 & & & & 1 & & 1 & 1 & 1 \end{bmatrix},$$

$$\mathbf{T}^{(5)}_{14\times9} = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & & & & \\ & 1 & & -1 & & & & & \\ & & 1 & -1 & & & & & \\ & & & 1 & & & & & \\ & -1 & 1 & 1 & -1 & -1 & & & \\ & & & & 1 & & & & \\ & & & -1 & 1 & & & & \\ & & & -1 & & 1 & & & \\ & & -1 & 1 & 1 & -1 & -1 & & \\ & & & & & 1 & & & \\ & & & 1 & -1 & -1 & 1 & -1 & \\ & & & & & & 1 & & \\ & & & & & & & 1 & \\ & & & & -1 & -1 & -1 & -1 & 1 \end{bmatrix},$$

and

$$\mathbf{D}^{(5)}_{14} = diag\left(s^{(5)}_0, s^{(5)}_1, \ldots, s^{(5)}_{13}\right),$$

$$s^{(5)}_0 = w^{(5)}_0,\quad s^{(5)}_1 = w^{(5)}_0 + w^{(5)}_1,\quad s^{(5)}_2 = w^{(5)}_0 + w^{(5)}_2,\quad s^{(5)}_3 = w^{(5)}_0 + w^{(5)}_1 + w^{(5)}_2 + w^{(5)}_3,$$

$$s^{(5)}_4 = w^{(5)}_1,\quad s^{(5)}_5 = w^{(5)}_0 + w^{(5)}_4,\quad s^{(5)}_6 = w^{(5)}_1 + w^{(5)}_3,\quad s^{(5)}_7 = w^{(5)}_2 + w^{(5)}_3,$$

$$s^{(5)}_8 = w^{(5)}_2,\quad s^{(5)}_9 = w^{(5)}_1 + w^{(5)}_4,\quad s^{(5)}_{10} = w^{(5)}_3,\quad s^{(5)}_{11} = w^{(5)}_2 + w^{(5)}_4,$$

$$s^{(5)}_{12} = w^{(5)}_3 + w^{(5)}_4,\quad s^{(5)}_{13} = w^{(5)}_4.$$

Figure 3 illustrates a data flow graph of the proposed algorithm for implementing the basic filtering operation for a 5-tap FIR filter.

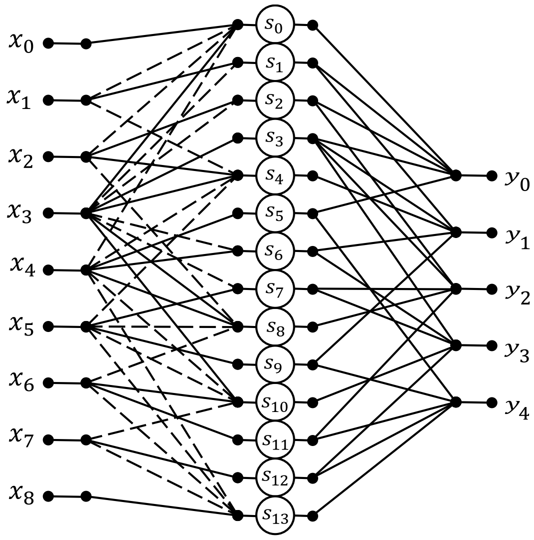

Figure 3. Data flow graph of the algorithm for implementing the basic filtering operation for the case $M = 5$.

4.3. Algorithm 3, M = 7

Let $\mathbf{X}_{13\times 1} = [x_0, x_1, \ldots, x_{12}]^{\mathrm{T}}$ be a vector that represents the input data set, $\mathbf{W}_{7\times 1} = \left[w_0^{(7)}, w_1^{(7)}, \ \ldots, w_6^{(7)}\right]^{\mathrm{T}}$ be a vector that contains the coefficients of the impulse response of a 7-tap FIR filter, and $\mathbf{Y}_{7\times 1} = \left[y_0^{(7)}, y_1^{(7)}, \ldots, y_6^{(7)}\right]^{\mathrm{T}}$ be a vector describing the results of using a 7-tap FIR filter:

$$\mathbf{Y}_{7\times 1} = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\ x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} \\ x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} \end{bmatrix} \begin{bmatrix} w_0^{(7)} \\ w_1^{(7)} \\ w_2^{(7)} \\ w_3^{(7)} \\ w_4^{(7)} \\ w_5^{(7)} \\ w_6^{(7)} \end{bmatrix} \tag{12}$$

As can be seen, calculating the product (12) requires 49 multiplications and 42 additions.

We can formulate a streamlined algorithm for computing $\mathbf{Y}_{7\times 1}$ by utilising the following matrix-vector calculation procedure:

$$\mathbf{Y}_{7\times 1} = \mathbf{T}_{7\times 15}^{(7)} \mathbf{T}_{15\times 25}^{(7)} \mathbf{D}_{25}^{(7)} \mathbf{T}_{25\times 18}^{(7)} \mathbf{T}_{18\times 13}^{(7)} \mathbf{X}_{13\times 1} \tag{13}$$

where

$$\mathbf{T}_{7\times 15}^{(7)} = \left[\begin{array}{cccccc:cccc:ccccc} 1 & & & & & & 1 & & & & & & & & \\ & 1 & & & & & & 1 & & & 1 & & & & \\ & & 1 & & & & & & 1 & & & 1 & & & \\ \hdashline & & & 1 & & & -1 & & & & & & 1 & & \\ & & & & 1 & & & -1 & & & & & & 1 & \\ & & & & & 1 & & & -1 & & & & & & 1 \\ & & & & & & & & & 1 & & & & & \end{array}\right],$$

$$\mathbf{T}_{9\times18}^{(7)}=\left[\begin{array}{c:c:c}\mathbf{T}_{3\times6}^{(3)} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6}\\ \hdashline \mathbf{0}_{3\times6} & \mathbf{T}_{3\times6}^{(3)} & \mathbf{0}_{3\times6}\\ \hdashline \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6} & \mathbf{T}_{3\times6}^{(3)}\end{array}\right],\quad \mathbf{T}_{9\times7}^{(7)}=\left[\begin{array}{c:c}1 & \mathbf{0}_{1\times6}\\ \hdashline \mathbf{0}_{8\times1} & \mathbf{0}_{8\times6}\end{array}\right],$$

$$\mathbf{T}_{6\times7}^{(7)}=\begin{bmatrix}1&1&1&1&1&1&1\\ &1&&&&&\\ &&1&&&&\\ &&&1&&&\\ &&&&1&&\\ &&&&&1&\end{bmatrix},\quad \mathbf{T}_{15\times25}^{(7)}=\left[\begin{array}{c:c}\mathbf{T}_{9\times18}^{(7)} & \mathbf{T}_{9\times7}^{(7)}\\ \hdashline \mathbf{0}_{6\times18} & \mathbf{T}_{6\times7}^{(7)}\end{array}\right],$$

$$\mathbf{T}_{6\times8}^{(7)}=\left[\begin{array}{c:c}\mathbf{T}_{6\times5}^{(3)} & \mathbf{0}_{6\times3}\end{array}\right],\quad \tilde{\mathbf{T}}_{6\times8}^{(7)}=\left[\begin{array}{c:c}\mathbf{0}_{6\times3} & \mathbf{T}_{6\times5}^{(3)}\end{array}\right],\quad \mathbf{T}_{12\times8}^{(7)}=\left[\begin{array}{c}\mathbf{T}_{6\times8}^{(7)}\\ \hdashline \tilde{\mathbf{T}}_{6\times8}^{(7)}\end{array}\right],$$

$$\mathbf{T}_{10\times5}^{(7)}=\left[\begin{array}{c:c:c:c:c}-1&&&&\\ \hdashline &&&&\\ \hdashline &&&&\\ \hdashline &-1&&&\\ \hdashline &&-1&&\\ \hdashline &&&&\\ \hdashline &&&-1&\\ \hdashline &&&&\\ \hdashline &&&&\\ \hdashline &&&&-1\end{array}\right],\quad \mathbf{T}_{10}^{(7)}=\left[\begin{array}{c:c:c}\mathbf{0}_{10\times4} & \mathbf{T}_{10\times5}^{(7)} & \mathbf{0}_{10\times1}\end{array}\right],$$

$$\mathbf{T}_{12\times10}^{(7)}=\left[\begin{array}{c}\mathbf{0}_{2\times10}\\ \hdashline \mathbf{T}_{10}^{(7)}\end{array}\right],\quad \mathbf{T}_{6\times10}^{(7)}=\left[\begin{array}{c:c}\mathbf{T}_{6\times5}^{(3)} & \mathbf{0}_{6\times5}\end{array}\right],\quad \mathbf{T}_{6\times7}^{(7)}=\left[\begin{array}{c:c}\mathbf{I}_{6} & \mathbf{0}_{6\times1}\end{array}\right],$$

$$\mathbf{T}_{1\times7}^{(7)}=\begin{bmatrix}-1&-1&-1&-1&-1&-1&1\end{bmatrix},$$

$$\mathbf{T}_{7}^{(7)}=\left[\begin{array}{c}\mathbf{T}_{6\times7}^{(7)}\\ \hdashline \mathbf{T}_{1\times7}^{(7)}\end{array}\right],\quad \mathbf{T}_{7\times10}^{(7)}=\left[\begin{array}{c:c}\mathbf{0}_{7\times3} & \mathbf{T}_{7}^{(7)}\end{array}\right],\quad \mathbf{T}_{13\times10}^{(7)}=\left[\begin{array}{c}\mathbf{T}_{6\times10}^{(7)}\\ \hdashline \mathbf{T}_{7\times10}^{(7)}\end{array}\right],$$

$$\mathbf{T}_{25\times18}=\left[\begin{array}{c:c}\mathbf{T}_{12\times8}^{(7)} & \mathbf{T}_{12\times10}\\ \hdashline \mathbf{0}_{13\times8} & \mathbf{T}_{13\times10}\end{array}\right],$$

$$\mathbf{T}_{8\times11}^{(7)}=\begin{bmatrix}1&&&1&&&-1&&&&\\ &1&&&1&&&&&&\\ &&1&&&1&&&&&\\ &&&1&&&1&&&&\\ &&&&1&&&1&&&\\ &&&&&1&&&1&&\\ &&&&&&1&&&1&\\ &&&&&&&1&&&1\end{bmatrix},$$

$$\mathbf{T}_{8\times13}^{(7)}=\left[\begin{array}{c:c}\mathbf{T}_{8\times11}^{(7)} & \mathbf{0}_{8\times2}\end{array}\right],\quad \mathbf{T}_{10\times13}^{(7)}=\left[\begin{array}{c:c}\mathbf{0}_{10\times3} & \mathbf{I}_{10}\end{array}\right],\quad \mathbf{T}_{18\times13}=\left[\begin{array}{c}\mathbf{T}_{8\times13}^{(7)}\\ \hdashline \mathbf{T}_{10\times13}^{(7)}\end{array}\right],$$

and

$$\mathbf{D}_{25}^{(7)}=diag\left(s_0^{(7)},s_1^{(7)},\ldots,s_{24}^{(7)}\right),$$

$$s_0^{(7)}=w_0^{(7)},\quad s_1^{(7)}=w_0^{(7)}+w_1^{(7)},\quad s_2^{(7)}=w_1^{(7)},\quad s_3^{(7)}=w_0^{(7)}+w_2^{(7)},\quad s_4^{(7)}=w_1^{(7)}+w_2^{(7)},$$

$$s_5^{(7)} = w_2^{(7)}, \quad s_6^{(7)} = w_3^{(7)}, \quad s_7^{(7)} = w_3^{(7)} + w_4^{(7)}, \quad s_8^{(7)} = w_4^{(7)}, \quad s_9^{(7)} = w_3^{(7)} + w_5^{(7)},$$

$$s_{10}^{(7)} = w_4^{(7)} + w_5^{(7)}, \quad s_{11}^{(7)} = w_5^{(7)}, \quad s_{12}^{(7)} = w_3^{(7)} - w_0^{(7)}, \quad s_{13}^{(7)} = w_3^{(7)} + w_4^{(7)} - w_0^{(7)} - w_1^{(7)},$$

$$s_{14}^{(7)} = w_4^{(7)} - w_1^{(7)}, \quad s_{15}^{(7)} = w_3^{(7)} + w_5^{(7)} - w_0^{(7)} - w_2^{(7)}, \quad s_{16}^{(7)} = w_4^{(7)} + w_5^{(7)} - w_1^{(7)} - w_2^{(7)},$$

$$s_{17}^{(7)} = w_5^{(7)} - w_2^{(7)}, \quad s_{18}^{(7)} = w_0^{(7)} + w_6^{(7)}, \quad s_{19}^{(7)} = w_1^{(7)} + w_6^{(7)}, \quad s_{20}^{(7)} = w_2^{(7)} + w_6^{(7)},$$

$$s_{21}^{(7)} = w_3^{(7)} + w_6^{(7)}, \quad s_{22}^{(7)} = w_4^{(7)} + w_6^{(7)}, \quad s_{23}^{(7)} = w_5^{(7)} + w_6^{(7)}, \quad s_{24}^{(7)} = w_6^{(7)}.$$

Figure 4 illustrates a data flow graph of the proposed algorithm for implementing the basic filtering operation for a 7-tap FIR filter.

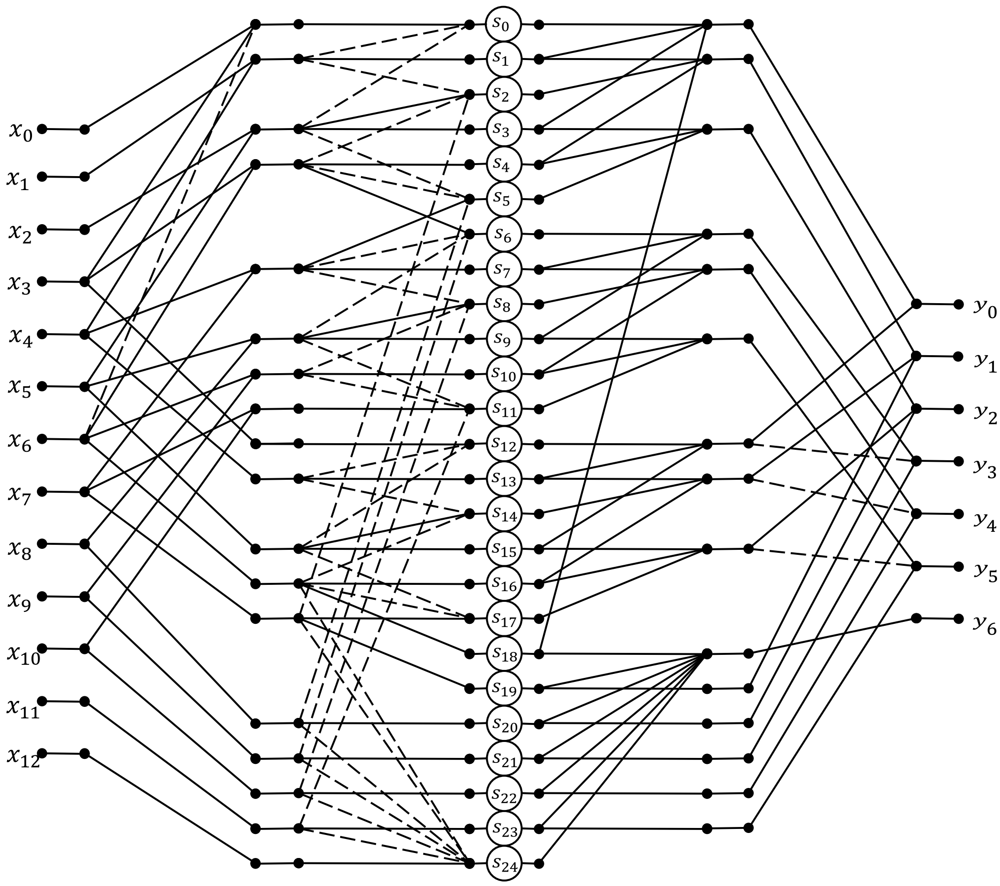

Figure 4. Data flow graph of the algorithm for implementing the basic filtering operation for the case $M = 7$.

4.4. Algorithm 4, M = 9

Let $\mathbf{X}_{17\times1} = [x_0, x_1, \ldots, x_{16}]^{\mathrm{T}}$ be a vector that represents the input data set, $\mathbf{W}_{9\times1} = \left[w_0^{(9)}, w_1^{(9)}, \ \ldots, w_8^{(9)}\right]^{\mathrm{T}}$ be a vector that contains the coefficients of the impulse response of a 9-tap FIR filter, and $\mathbf{Y}_{9\times1} = \left[y_0^{(9)}, y_1^{(9)}, \ldots, y_8^{(9)}\right]^{\mathrm{T}}$ be a vector describing the results of using a 9-tap FIR filter:

$$\mathbf{Y}_{9\times1} = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\ x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} \\ x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} \\ x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} \\ x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} \\ x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \begin{bmatrix} w_0^{(9)} \\ w_1^{(9)} \\ w_2^{(9)} \\ w_3^{(9)} \\ w_4^{(9)} \\ w_5^{(9)} \\ w_6^{(9)} \\ w_7^{(9)} \\ w_8^{(9)} \end{bmatrix} \tag{14}$$

As can be seen, calculating the product (14) requires 81 multiplications and 72 additions.

We can formulate a streamlined algorithm for computing $\mathbf{Y}_{9\times1}$ by utilising the following matrix-vector calculation procedure:

$$\mathbf{Y}_{9\times1} = \mathbf{T}_{9\times18}^{(9)} \mathbf{T}_{18\times36}^{(9)} \mathbf{D}_{36}^{(9)} \mathbf{T}_{36\times30}^{(9)} \mathbf{T}_{30\times17}^{(9)} \mathbf{X}_{17\times1} \tag{15}$$

where

$$\mathbf{T}_{9\times18}^{(9)} = \left[\begin{array}{cccccc:ccc:ccccccccc} 1 & & & 1 & & & & & & 1 & & & & & & & & \\ & 1 & & & 1 & & & & & & 1 & & & & & & & \\ & & 1 & & & 1 & & & & & & 1 & & & & & & \\ & & & 1 & & & 1 & & & & & & 1 & & & & & \\ & & & & 1 & & & 1 & & & & & & 1 & & & & \\ & & & & & 1 & & & 1 & & & & & & 1 & & & \\ \hdashline & & & & & & & & & 1 & & & 1 & & & 1 & & \\ & & & & & & & & & & 1 & & & 1 & & & 1 & \\ & & & & & & & & & & & 1 & & & 1 & & & 1 \end{array}\right],$$

$$\tilde{\mathbf{T}}_{9\times18}^{(9)} = \left[\begin{array}{c:c:c} \mathbf{T}_{3\times6}^{(3)} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6} \\ \hdashline \mathbf{0}_{3\times6} & \mathbf{T}_{3\times6}^{(3)} & \mathbf{0}_{3\times6} \\ \hdashline \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6} & \mathbf{T}_{3\times6}^{(3)} \end{array}\right], \quad \mathbf{T}_{18\times36}^{(9)} = \left[\begin{array}{c:c} \tilde{\mathbf{T}}_{9\times18}^{(9)} & \mathbf{0}_{9\times18} \\ \hdashline \mathbf{0}_{9\times18} & \tilde{\mathbf{T}}_{9\times18}^{(9)} \end{array}\right],$$

$$\mathbf{T}_{6\times10}^{(9)} = \left[\begin{array}{c:c} \mathbf{T}_{6\times5}^{(3)} & \mathbf{0}_{6\times5} \end{array}\right], \quad \tilde{\mathbf{T}}_{6\times10}^{(9)} = \left[\begin{array}{c:c} \mathbf{0}_{6\times5} & \mathbf{T}_{6\times5}^{(3)} \end{array}\right], \quad \mathbf{T}_{12\times10}^{(9)} = \left[\begin{array}{c} \mathbf{T}_{6\times10}^{(9)} \\ \hdashline \tilde{\mathbf{T}}_{6\times10}^{(9)} \end{array}\right],$$

$$\mathbf{T}_{36\times30}^{(9)} = \left[\begin{array}{c:c:c} \mathbf{T}_{12\times10}^{(9)} & \mathbf{0}_{12\times10} & \mathbf{0}_{12\times10} \\ \hdashline \mathbf{0}_{12\times10} & \mathbf{T}_{12\times10}^{(9)} & \mathbf{0}_{12\times10} \\ \hdashline \mathbf{0}_{12\times10} & \mathbf{0}_{12\times10} & \mathbf{T}_{12\times10}^{(9)} \end{array}\right],$$

$$\mathbf{T}_{5\times11}^{(9a)} = \begin{bmatrix} 1 & & & -1 & & & -1 & & & & \\ & 1 & & & -1 & & & -1 & & & \\ & & 1 & & & -1 & & & -1 & & \\ & & & 1 & & & -1 & & & -1 & \\ & & & & 1 & & & -1 & & & -1 \end{bmatrix},$$

$$\mathbf{T}_{5\times 11}^{(9b)} = \begin{bmatrix} -1 & & & 1 & & & -1 & & & & \\ & -1 & & & 1 & & & -1 & & & \\ & & -1 & & & 1 & & & -1 & & \\ & & & -1 & & & 1 & & & -1 & \\ & & & & -1 & & & 1 & & & -1 \end{bmatrix},$$

$$\mathbf{T}_{5\times 11}^{(9c)} = \begin{bmatrix} -1 & & & -1 & & & 1 & & & & \\ & -1 & & & -1 & & & 1 & & & \\ & & -1 & & & -1 & & & 1 & & \\ & & & -1 & & & -1 & & & 1 & \\ & & & & -1 & & & -1 & & & 1 \end{bmatrix},$$

$$\mathbf{T}_{5\times 17}^{(9a)} = \left[\ \mathbf{T}_{5\times 11}^{(9a)} \ \vdots\ \mathbf{0}_{5\times 6}\ \right], \quad \mathbf{T}_{5\times 17}^{(9b)} = \left[\ \mathbf{0}_{5\times 3} \ \vdots\ \mathbf{I}_5 \ \vdots\ \mathbf{0}_{5\times 9}\ \right],$$

$$\mathbf{T}_{5\times 17}^{(9c)} = \left[\ \mathbf{0}_{5\times 3} \ \vdots\ \mathbf{T}_{5\times 11}^{(9b)} \ \vdots\ \mathbf{0}_{5\times 3}\ \right], \quad \mathbf{T}_{5\times 17}^{(9d)} = \left[\ \mathbf{0}_{5\times 6} \ \vdots\ \mathbf{I}_5 \ \vdots\ \mathbf{0}_{5\times 6}\ \right],$$

$$\mathbf{T}_{5\times 17}^{(9e)} = \left[\ \mathbf{0}_{5\times 9} \ \vdots\ \mathbf{I}_5 \ \vdots\ \mathbf{0}_{5\times 3}\ \right], \quad \mathbf{T}_{5\times 17}^{(9f)} = \left[\ \mathbf{0}_{5\times 6} \ \vdots\ \mathbf{T}_{5\times 11}^{(9c)}\ \right],$$

$$\mathbf{T}_{30\times 17}^{(9)} = \begin{bmatrix} \mathbf{T}_{5\times 17}^{(9a)} \\ \hdashline \mathbf{T}_{5\times 17}^{(9b)} \\ \hdashline \mathbf{T}_{5\times 17}^{(9c)} \\ \hdashline \mathbf{T}_{5\times 17}^{(9d)} \\ \hdashline \mathbf{T}_{5\times 17}^{(9e)} \\ \hdashline \mathbf{T}_{5\times 17}^{(9f)} \end{bmatrix},$$

$$\mathbf{D}_{36}^{(9)} = diag\left(s_0^{(9)}, s_1^{(9)}, \ldots, s_{35}^{(9)}\right),$$

$$s_0^{(9)} = w_0^{(9)}, \quad s_1^{(9)} = w_0^{(9)} + w_1^{(9)}, \quad s_2^{(9)} = w_1^{(9)}, \quad s_3^{(9)} = w_0^{(9)} + w_2^{(9)}, \quad s_4^{(9)} = w_1^{(9)} + w_2^{(9)},$$

$$s_5^{(9)} = w_2^{(9)}, \quad s_6^{(9)} = w_0^{(9)} + w_3^{(9)}, \quad s_7^{(9)} = w_0^{(9)} + w_1^{(9)} + w_3^{(9)} + w_4^{(9)}, \quad s_8^{(9)} = w_1^{(9)} + w_4^{(9)},$$

$$s_9^{(9)} = w_0^{(9)} + w_2^{(9)} + w_3^{(9)} + w_5^{(9)}, \quad s_{10}^{(9)} = w_1^{(9)} + w_2^{(9)} + w_4^{(9)} + w_5^{(9)}, \quad s_{11}^{(9)} = w_2^{(9)} + w_5^{(9)},$$

$$s_{12}^{(9)} = w_3^{(9)}, \quad s_{13}^{(9)} = w_3^{(9)} + w_4^{(9)}, \quad s_{14}^{(9)} = w_4^{(9)}, \quad s_{15}^{(9)} = w_3^{(9)} + w_5^{(9)}, \quad s_{16}^{(9)} = w_4^{(9)} + w_5^{(9)},$$

$$s_{17}^{(9)} = w_5^{(9)}, \quad s_{18}^{(9)} = w_0^{(9)} + w_6^{(9)}, \quad s_{19}^{(9)} = w_0^{(9)} + w_1^{(9)} + w_6^{(9)} + w_7^{(9)}, \quad s_{20}^{(9)} = w_1^{(9)} + w_7^{(9)},$$

$$s_{21}^{(9)} = w_0^{(9)} + w_2^{(9)} + w_6^{(9)} + w_8^{(9)}, \quad s_{22}^{(9)} = w_1^{(9)} + w_2^{(9)} + w_7^{(9)} + w_8^{(9)}, \quad s_{23}^{(9)} = w_2^{(9)} + w_8^{(9)},$$

$$s_{24}^{(9)} = w_3^{(9)} + w_6^{(9)}, \quad s_{25}^{(9)} = w_3^{(9)} + w_4^{(9)} + w_6^{(9)} + w_7^{(9)}, \quad s_{26}^{(9)} = w_4^{(9)} + w_7^{(9)},$$

$$s_{27}^{(9)} = w_3^{(9)} + w_5^{(9)} + w_6^{(9)} + w_8^{(9)}, \quad s_{28}^{(9)} = w_4^{(9)} + w_5^{(9)} + w_7^{(9)} + w_8^{(9)}, \quad s_{29}^{(9)} = w_5^{(9)} + w_8^{(9)},$$

$$s_{30}^{(9)} = w_6^{(9)}, \quad s_{31}^{(9)} = w_6^{(9)} + w_7^{(9)}, \quad s_{32}^{(9)} = w_7^{(9)}, \quad s_{33}^{(9)} = w_6^{(9)} + w_8^{(9)}, \quad s_{34}^{(9)} = w_7^{(9)} + w_8^{(9)},$$

$$s_{35}^{(9)} = w_8^{(9)}.$$

Figure 5 illustrates a data flow graph of the proposed algorithm for implementing the basic filtering operation for a 9-tap FIR filter.

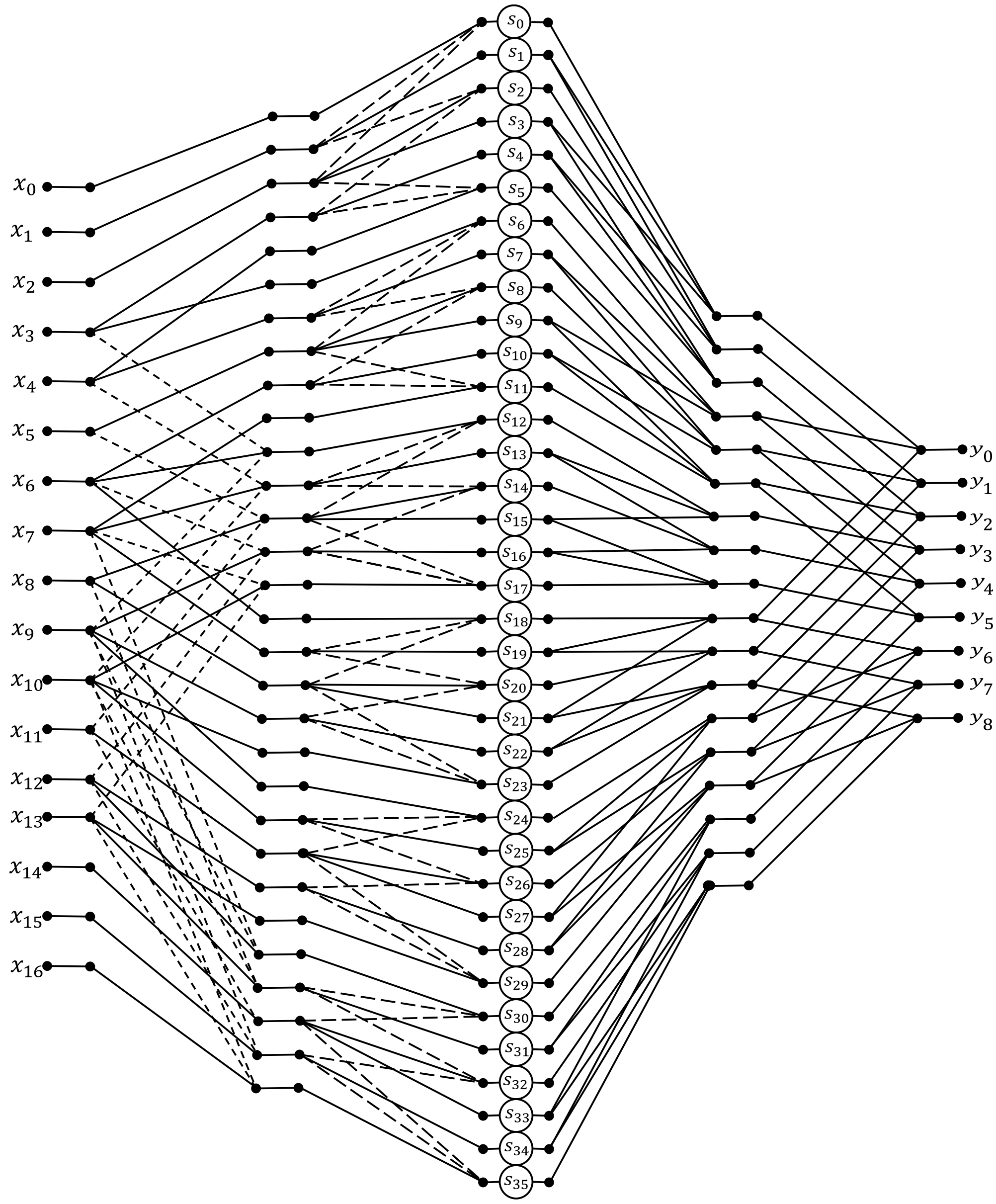

Figure 5. Data flow graph of the algorithm for implementing the basic filtering operation for the case $M = 9$.

5. Implementation Complexity

Due to the relatively small lengths of the input sequences and the straightforward nature of the data flow diagrams depicting the computation process, it is easy to assess the implementation complexity of the proposed solutions. Table 1 estimates the number of arithmetic blocks required for the fully parallel implementation of the filtering algorithms designed for short lengths. The values presented in the table can be regarded as an approximate measure of the implementation cost on an ASIC.

Table 1. The complexities of implementing of the naive and proposed solutions.

Size M	Numbers of Arithmetic Blocks Naive Method		Proposed Algorithm				
	Multipliers	M-Input Adders	Multipliers	2-Input Adders	3-Input Adders	4-Input Adders	M-Input Adders
3	9	3	6	-	6	-	-
5	25	5	14	4	-	-	10
7	49	7	25	7	24	1	1
9	81	9	36	-	60	-	-

As we can see, using the proposed algorithmic solutions to construct digital filtering cores results in fewer multipliers being needed than using naive approaches to their design. In the context of designing specialised fully parallel VLSI processors, minimising the number of multipliers is of paramount importance. This approach significantly reduces the cost of implementing the entire system and mitigates power dissipation. This is due to the hardware multiplier's higher level of complexity and larger chip area occupation than the adder. It has been demonstrated that the hardware cost of a binary adder rises linearly with the operand size. In contrast, the implementation cost of a hardwired multiplier escalates quadratically with the operand size [46]. Hence, reducing the number of multipliers, even if it results in a slight increase in the number of adders, significantly influences the hardware implementation of digital filtering cores.

The proposed algorithms have been exemplarily implemented in FPGAs on the simplest possible devices of Xilinx's Spartan 3 series. The criterion for selecting a model from the Spartan 3 family was to provide a sufficient number of inputs and outputs. The 8-bit inputs $\mathbf{X}_{(2M-1)\times 1}$, 16-bit outputs $\mathbf{Y}_{9\times 1}$, and fixed 8-bit coefficients of the impulse response of the FIR filter $\mathbf{W}_{M\times 1}$, were assumed. Table 2 shows the number of slices used in the Spartan 3 FPGA implementation. The number of uses multipliers MULT 18×18 is also shown in this table, but both algorithms mostly used all hardware-accessible multipliers. Only for size $M = 5$ was the number of available multipliers is for the proposed algorithm greater than required, and only in this case did the algorithm not use all of them. Table 3 shows the number of four input LUTs used in the Spartan 3 FPGA implementation. For each size M, the proposed algorithms achieved a reduction of the logic blocks used. The smallest was for the size $M = 3$, where the reduction was only about 1% for the slices and 2,4% for four inputs LUTs. The biggest was for size $M = 5$, which achieves nearly a 40% decrease in logical blocks.

Table 2. The number of the multipliers MULT 18 × 18, and the slices used in the Spartan 3 FPGA implementations.

Size M	Devices	MULT 18 × 18 School	Proposed	School	Slices Proposed	Reduction
3	xc3s50-4pq208	4	4	111	110	0.9%
5	xc3s400-4fg456	16	14	563	342	39.3%
7	xc3s400-4fg456	16	16	776	684	11.9%
9	xc3s1000-4fg676	24	24	1303	1066	18.2%

Table 3. The number of the 4 input LUTs used in the Spartan 3 FPGA implementations.

Size M	Devices	School	4 Input LUTs Proposed	Reduction
3	xc3s50-4pq208	207	202	2.4%
5	xc3s400-4fg456	1040	636	38.8%
7	xc3s400-4fg456	1441	1284	10.9%
9	xc3s1000-4fg676	2456	1994	18.8%

6. Conclusions

This study explores methods to reduce the multiplicative complexity of conducting basic filtering operations for M-tap FIR filters with short impulse responses, commonly used in convolutional neural networks. Additionally, new algorithms for resource-efficient implementations of these algorithms have been devised for M values of 3, 5, 7, and 9. By utilising these algorithms, basic filtering operations computational complexity is reduced, which also lessens the difficulty of their hardware implementation. Reducing the number of multiplications in the algorithms comes at the expense of some increase in the number of additions. However, this is not significant due to the much higher implementation cost of the hardware multiplier relative to the adder. Some limitation of the proposed algorithms is the increased complexity of data manipulation. For this reason, it seems particularly advantageous to implement the proposed solutions in ASICs. The distinctive feature of all the proposed algorithms is their evident parallel and modular structures. The modularity allows unifying the implementation of the algorithms in FPGAs and makes it easier to map them into ASIC structures. Consequently, the parallelisation of computing processes enables accelerated computations during the execution of these algorithms. The implementation of the proposed algorithms in DNNs will be a target for further research.

Author Contributions: Conceptualization, A.C.; methodology, A.C., J.P.P. and M.M.; formal analysis, A.C., J.P.P. and M.M.; writing—original draft preparation, A.C.; writing—review and editing, A.C. and J.P.P.; visualization, A.C., M.M. and J.P.P.; supervision, A.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

VLSI	very large-scale integration
DNN	deep neural networks
FFT	fast Fourier transform
FIR	finite impuls responce
FPGA	field programmable gate array
GPU	graphics processing unit
ASIC	application-specific integrated circuit
Tiny ML	Tiny machine learning
Edge AI	Edge artificial intelligence
MULT	Multiplication
LUT	look up table

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
3. Adhikari, S.P.; Kim, H.; Yang, C.; Chua, L.O. Building cellular neural network templates with a hardware friendly learning algorithm. *Neurocomputing* **2018**, *312*, 276–284. [CrossRef]
4. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 1–74.
5. Habib, G.; Qureshi, S. Optimization and acceleration of convolutional neural networks: A survey. *J. King Saud-Univ.-Comput. Inf. Sci.* **2022**, *34*, 4244–4268. [CrossRef]
6. Lin, S.; Liu, N.; Nazemi, M.; Li, H.; Ding, C.; Wang, Y.; Pedram, M. FFT-based deep learning deployment in embedded systems. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1045–1050. [CrossRef]
7. Mathieu, M.; Henaff, M.; LeCun, Y. Fast Training of Convolutional Networks through FFTs. *arXiv* **2014**, arXiv:1312.5851.
8. Abtahi, T.; Kulkarni, A.; Mohsenin, T. Accelerating convolutional neural network with FFT on tiny cores. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4. [CrossRef]
9. Abtahi, T.; Shea, C.; Kulkarni, A.; Mohsenin, T. Accelerating Convolutional Neural Network with FFT on Embedded Hardware. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **2018**, *26*, 1737–1749. [CrossRef]
10. Lin, J.; Yao, Y. A Fast Algorithm for Convolutional Neural Networks Using Tile-based Fast Fourier Transforms. *Neural Process. Lett.* **2019**, *50*, 1951–1967. [CrossRef]
11. Wu, Y. Review on FPGA-Based Accelerators in Deep learning. In Proceedings of the 2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chongqing, China, 23–26 February 2023; Volume 6, pp. 452–456. [CrossRef]
12. Lavin, A.; Gray, S. Fast Algorithms for Convolutional Neural Networks. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 4013–4021. [CrossRef]
13. Zhao, Y.; Wang, D.; Wang, L. Convolution accelerator designs using fast algorithms. *Algorithms* **2019**, *12*, 112. [CrossRef]
14. Yang, D.S.; Xu, C.H.; Ruan, S.J.; Huang, C.M. Unified energy-efficient reconfigurable MAC for dynamic Convolutional Neural Network based on Winograd algorithm. *Microprocess. Microsyst.* **2022**, *93*, 104624. [CrossRef]
15. Dolz, M.F.; Barrachina, S.; Martínez, H.; Castelló, A.; Maciá, A.; Fabregat, G.; Tomás, A.E. Performance–energy trade-offs of deep learning convolution algorithms on ARM processors. *J. Supercomput.* **2023**, *79*, 1–18. [CrossRef]
16. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
17. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 6848–6856.
18. Wang, X.; Wang, C.; Zhou, X. Work-in-Progress: WinoNN: Optimising FPGA-based Neural Network Accelerators using Fast Winograd Algorithm. In Proceedings of the 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Turin, Italy, 30 September–3 October 2018; pp. 1–2. [CrossRef]
19. Farabet, C.; Poulet, C.; Han, J.Y.; LeCun, Y. CNP: An FPGA-based processor for convolutional networks. In Proceedings of the FPL 2009, IEEE, Prague, Czech Republic, 31 August–2 September 2009; pp. 32–37.
20. Lu, L.; Liang, Y. SpWA: An Efficient Sparse Winograd Convolutional Neural Networks Accelerator on FPGAs. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018; pp. 1–6. [CrossRef]
21. Yu, J.; Hu, Y.; Ning, X.; Qiu, J.; Guo, K.; Wang, Y.; Yang, H. Instruction driven cross-layer CNN accelerator with winograd transformation on FPGA. In Proceedings of the 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VI, Australia, 11–13 December 2017; pp. 227–230. [CrossRef]
22. Liang, Y.; Lu, L.; Xiao, Q.; Yan, S. Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 857–870. [CrossRef]
23. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access* **2019**, *7*, 7823–7859.
24. Guo, K.; Zeng, S.; Yu, J.; Wang, Y.; Yang, H. A Survey of FPGA-Based Neural Network Accelerator. *arXiv* **2018**, arXiv:1712.08934.
25. Hoffmann, J.; Navarro, O.; Kästner, F.; Janßen, B.; Hübner, M. A Survey on CNN and RNN Implementations. In Proceedings of the PESARO 2017: The Seventh International Conference on Performance, Safety and Robustness in Complex Systems and Applications, Pesaro, Italy, 23–27 April 2017; pp. 33–39.
26. Liu, Z.; Chow, P.; Xu, J.; Jiang, J.; Dou, Y.; Zhou, J. A Uniform Architecture Design for Accelerating 2D and 3D CNNs on FPGAs. *Electronics* **2019**, *8*, 65. [CrossRef]

27. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 15–24. [CrossRef]
28. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; pp. 161–170. [CrossRef]
29. Li, Y.; Liu, Z.; Xu, K.; Yu, H.; Ren, F. A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. *arXiv* **2017**, arXiv:1702.06392.
30. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. Going deeper with embedded fpga platform for convolutional neural network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2016; pp. 26–35.
31. Li, H.; Fan, X.; Jiao, L.; Cao, W.; Zhou, X.; Wang, L. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), IEEE, Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9.
32. Hardieck, M.; Kumm, M.; Möller, K.; Zipf, P. Reconfigurable Convolutional Kernels for Neural Networks on FPGAs. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 22–24 February 2019; pp. 43–52. [CrossRef]
33. Ghimire, D.; Kil, D.; Kim, S.H. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics* **2022**, *11*, 945. [CrossRef]
34. Strigl, D.; Kofler, K.; Podlipnig, S. Performance and Scalability of GPU-Based Convolutional Neural Networks. In Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Pisa, Italy, 17–19 February 2010.
35. Li, X.; Zhang, G.; Huang, H.H.; Wang, Z.; Zheng, W. Performance Analysis of GPU-Based Convolutional Neural Networks. In Proceedings of the 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 16–19 August 2016; pp. 67–76. [CrossRef]
36. Cengil, E.; Cinar, A.; Guler, Z. A GPU-based convolutional neural network approach for image classification. In Proceedings of the 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, Turkey, 16–17 September 2017; pp. 1–6. [CrossRef]
37. Chen, Y.H.; Krishna, T.; Emer, J.; Sze, V. 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 31 January 2016; pp. 262–263. [CrossRef]
38. Ovtcharov, K.; Ruwase, O.; Kim, J.Y.; Fowers, J.; Strauss, K.; Chung, E.S. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Res.* **2015**, *2*, 1–14.
39. Tu, F.; Yin, S.; Ouyang, P.; Tang, S.; Liu, L.; Wei, S. Deep convolutional neural network architecture with reconfigurable computation patterns. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **2017**, *25*, 2220–2233. [CrossRef]
40. 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]
41. Kala, S.; Jose, B.R.; Mathew, J.; Nalesh, S. High-performance CNN accelerator on FPGA using unified winograd-GEMM architecture. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **2019**, *27*, 2816–2828. [CrossRef]
42. An, Y.; Li, B.; Bu, J.; Gao, Y. Optimizing Winograd convolution on GPUs via multithreaded communication. In Proceedings of the Second International Conference on Algorithms, Microchips, and Network Applications (AMNA 2023), SPIE, Zhengzhou, China, 13–15 January 2023; Volume 12635, pp. 204–212.
43. Cariow, A.; Cariowa, G. Minimal filtering algorithms for convolutional neural networks. In *Reliability Engineering and Computational Intelligence*; Springer: Cham, Switzerland 2021; pp. 73–88.
44. Cariow, A.; Gliszczyński, M. Fast algorithms to compute matrix-vector products for Toeplitz and Hankel matrices. *Electr. Rev.* **2012**, *88*, 166–171.
45. Beliakov, G. On fast matrix-vector multiplication with a Hankel matrix in multiprecision arithmetics. *arXiv* **2014**, arXiv:1402.5287.
46. Oudjida, A.K.; Chaillet, N.; Berrandjia, M.L.; Liacha, A. A New High Radix-2^r ($r \geq 8$) Multibit Recoding Algorithm for Large Operand Size ($N \geq 32$) Multipliers. *J. Low Power Electron.* **2013**, *9*, 50–62. [CrossRef]