Analysis of Optimum 3-Dimensional Array and Fast Data Movement for Efficient Memory Computation in Convolutional Neural Network Models

Deepika Selvaraj, Arunachalam Venkatesan, David Novo

Analysis of Optimum 3-Dimensional Array and Fast Data Movement for Efficient Memory Computation in Convolutional Neural Network Models

Deepika Selvaraj[1], Arunachalam Venkatesan[1(✉)], and David Novo[2]

[1] Department of Micro and Nanoelectronics, Vellore Institute of Technology, Vellore, India
deepika.2018a@vitstudent.ac.in, varunachalam@vit.ac.in
[2] French National Centre for Scientific Research (CNRS), University of Montpellier, LIRMM, Montpellier, France
david.novo@lirmm.fr

Abstract. CNN-based inference engine's performance and efficiency always depend on the computational and dataflow-control complexity. Instead of considering a 2-dimensional (2D) feature array for processing, a 3D array of features/weights would improve the dataflow movement & memory computation. The optimum $8 \times 8 \times 32$ 3D-feature array size was chosen based on the factor of on-chip memory requirement, data reuse, and PE utilization. Using the optimum $8 \times 8 \times 32$ feature array, seven different combinations of data-flow scheduling strategies were analyzed by varying row, column, and depth-wise parameters on the workload model using a MATLAB environment. From the analysis, strategy-V (depth-wise parallel & row/column-wise sequence) is found to be the best with a 4×8 processor array. Compared to the state-of-the-art processor strategy, strategy-V achieves the data transfer rate (off-chip to on-chip) and on-chip memory requirement of 3.3 times (higher) and 16 times (lesser) with a small overhead of processor cost.

Keywords: Convolutional Neural Network · Data flow Movement · Efficient Memory · Feature Array · Processing Element · Scheduling

1 Introduction

Automation in agricultural, automobile, medical and industrial fields universally use sensing devices such as sensors and digital cameras. These devices bring a sequence of data (image/video data) into the inference engines. Therefore, inference engines have to adopt a convolutional neural network (CNN) algorithm to automate and process the huge data in a limited time [1, 2]. The computer vision adopted standard deep learning-CNN models such as AlexNet, VGG Net, ResNet, and GoogleNet for image classification and detection [3]. Since 2010, IE-based hardware is using prominently for the CNN models. ASIC-based CNN inference engines are challenging to implement for a specific

Published by Springer Nature Switzerland AG 2023
E. Mercier-Laurent et al. (Eds.): ICCCSP 2023, IFIP AICT 670, pp. 94–108, 2023.
https://doi.org/10.1007/978-3-031-39811-7_8

image application. This implementation has concentrated on major factors to improve the CNN accelerator efficiently such as reducing arithmetic precision, energy efficiency, silicon area, and performance [4]. A generic CNN structure consists of a stack of convolutional layers with activation, pooling layers, and fully connected layers. Input feature maps (IFMs)/Output feature maps (OFMs) from the previous layer and weights/filters (FL) are input to the convolution layer and fully connected layer. It is computed using Multiply- Accumulate unit (MAC) followed by Rectified Linear Unit (RELU) activation function present in the convolutional and fully connected layer. Weights/filters (FL) are extracted from the pre-trained CNN models. Convolutional and fully connected layers are repetitive and used n-times based on the CNN models. Finally, the vector-based fully connected layer classifies the image based on the probability of output value. The general structure for the CNN convolution layer is shown in Fig. 1.

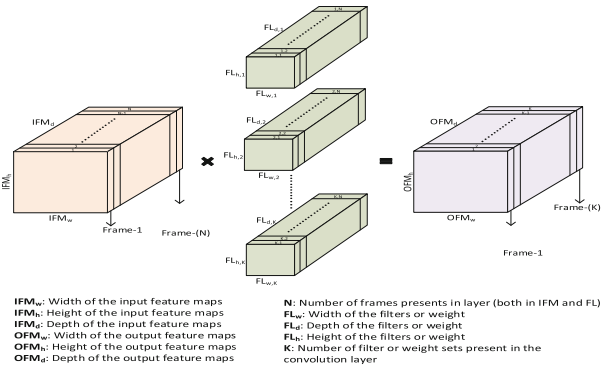


Fig. 1. Generic structure of the convolutional layer in the CNN model.

In this Fig. 1, the Input feature map ($IFM_h \times IFM_w \times IFM_d$) are convoluted with each set of weights/filters ($FL = FL_h \times FL_w \times FL_d \times K$) to produce the output feature map ($OFM = OFM_h \times OFM_w \times OFM_d$). Each depth of the filter has equal to the IFM depth ($IFM_d = FL_d = N$) and the number of filters has equal to the output feature map depth ($K = OFM_d$). For example, $IFM = 55 \times 55 \times 64$ convoluted $FL = 3 \times 3 \times 64 \times 512$ with one stride (S) and gives $OFM = 53 \times 53 \times 512$. Therefore, the convolution layer needs many MAC computations to process IFMs and weights. It is challenging to organize the data flow to the processing elements (MAC units) array processor during the inference stage.

Each convolutional layer requires hundreds of MAC computations in gigabytes. It could impact the performance, and memory storage of the CNN accelerator when the data are not stored in an orderly for the further process [5]. Such that impacts the final accuracy in the CNN model which deploys the resource-limited device and IoT-based CNN models. From the above consideration, improvement can be accomplished by fine-tuning the following factors: 1. Optimum data format to represent the IFMs and weights for maintaining the accuracy of the CNN models, 2. Data-flow movement between memory to processing elements (PEs) for improving the data reuse capability, 3. Pipelining and parallel scheduling for near 100% utilization of PEs, 4. Designing

the sparse accelerator by skipping the trivial MAC computation. To achieve the above-mentioned concern, several hardware accelerators are proposed in the previous works as follows

The representation of short floating-point (4-bit exponent and 4-bit mantissa) was proposed in [11] which reduces the computation complexity of the processor and memory storage. But the short floating-point [11] achieves the accuracy of the AlexNet model by 79% and the VGG-16 model by 88%. A 16-bit Fix/Float [12] represents the IFMs as 16-bit fixed points and weights as half-precision floating points. The representation achieves an accuracy near 97%. The hardware performance and energy efficiency have improved by 750MOP/s and 24TOPS/W at 250MHz.

Eyeriss processor [6] uses the efficient dataflow called row-stationary which involves the data reuse concept and the high parallelism level. Eyeriss accelerator accepts 168 PEs which adopts the flexibility of the kernel size from 3×3 to 11×11. Eyeriss achieves 0.0029 DRAM access/MAC access and 35 frames/s for the convolutional layer in AlexNet at 250MHz with a power consumption of 278mW. KOP3 processor [13] uses the n-tile parallel structure to improve the speed for the convolutional and fully connected layers. Also, it adopts the kernel size of 3×3 and a circular buffer strategy to reduce the power consumption and memory access in the CNN accelerator. The average speedup of the hardware implementation achieves by 3.77TOPS/W. However, the above methods have been discussed to reduce the off-chip memory access or on-chip memory access, data flow, and data reuse approach [7–10]. The contributions are framed based on the challenges of reducing the data flow computation complexity between the on-chip memory and PEs, and achieving maximum data reuse in IFMs and FL

The key contribution of this article is as follows

- Analyze and choose the optimum 3-Dimensional (3D) feature array (F_a) of $8 \times 8 \times 32$ with consideration of the four key factors including data reuse, the number of F_a required per CNN layer, hardware utilization, and memory requirements. 3D feature array has to improve the efficiency of the computation between on-chip memory to a processor for CNN inference.
- The combinations of data-flow scheduling strategies (I to VII) have been examined using the optimum 3D feature array ($8 \times 8 \times 32$) with a (4×8) PE processor. Strategy-V (depth-wise parallelly and row/column-wise sequence) has been adopted to maximize the data reuse in both IFMs and FLs.
- Also, scheduling strategy-V has been analyzed for the worst-case convolutional layer present in the standard CNN model (Input workload- $IFM : 55 \times 55 \times 512$ and $FL : 3 \times 3 \times 512 \times 512$). It balanced both data transfer rate and processor cost.
- The proposed strategy-V has been validated with standard and existing data-flow model [6, 14, 15] which is adopted from the previous implementation. All the analyses were done using the MATLAB-based design modeling algorithm.

Section 2 discussed the analysis of the optimum 3-Dimensional feature array (F_a) of size $8 \times 8 \times 32$ and different combination data-flow movement strategies I to VII are examined in a MATLAB-based environment and describes the mathematical model. MATLAB-based simulation results are illustrated and discussed for the CNN-based inference engine in terms of processing element, data reuse, and memory requirement in Sect. 3. Finally, Sect. 4 concludes the findings

2 Analysis of 3-dimensional Feature Array Size and Data-Flow Movement Scheduling Strategy

To improve the efficiency of the processor array and memory requirement, the optimum 3D feature array for computation is chosen from the various combination of feature array sizes. From the optimum 3D feature array size, different data flow movement, and scheduling strategies I to strategies VII have been modeled (parallel data flow/sequence data flow/ combination of parallel and sequential data flow). The appropriate 3D feature array (F_a) and suitable scheduling strategy are chosen based on the prime factors as follows

(1) Maximize the data reuse capability of both IFMs and weights,
(2) Maximize the processing element utilization,
(3) Reduce the intermediate memory storage, and
(4) Minimize the number of data transaction

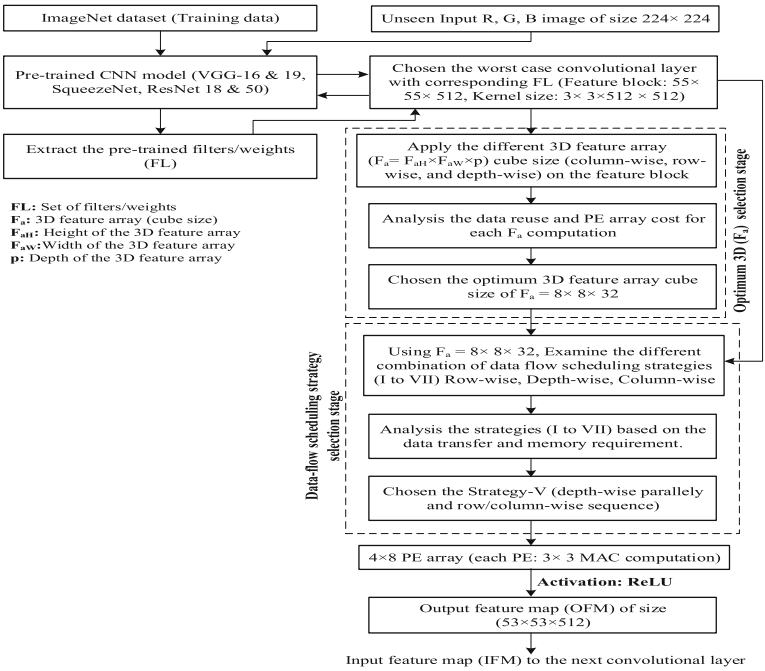


Fig. 2. MATLAB analysis for selection of 3- dimensional feature array (F_a) and data-flow movement strategies for the convolutional layer

For F_a and scheduling strategy analysis, the standard CNN models such as VGG-16 &19, SqueezeNet, and ResNet- 18 &50 are adopted. We propose a MATLAB-based model for choosing the suitable 3D feature array (F_a) and data flow scheduling strategy between on-chip memory to PE processor array. 3-channel images of 226 × 226 pixels

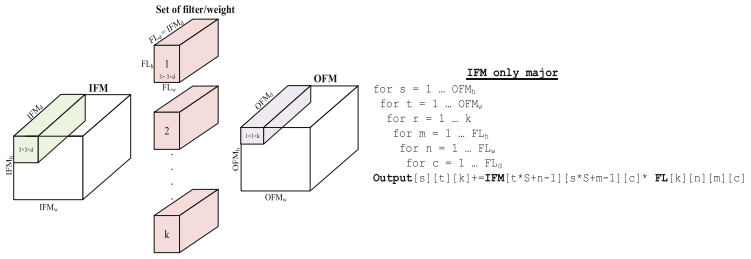
for VGG-16 &19, SqueezeNet, and ResNet- 18 &50 is chosen as inputs. For the analysis, the worst-case convolutional layer concerning a maximum number of sets of filter and filter depth, and IFM depth is taken from the above-mentioned standard CNN models. The corresponding set of filters/weights is extracted from the pre-trained CNN models. The worst-case input load of size $55 \times 55 \times 512$ for IFM and $3 \times 3 \times 512 \times 512$ for FL is considered for the selection of the 3D feature array. The optimum 3D feature array of size F_a:$8 \times 8 \times 32$ has been considered based on the above four major factors. The major four factors have mainly related to the on-chip memory inside the processor. The optimum feature array size is required to transfer the group of IFM and weights from off-chip to on-chip memory/on-chip to PE array. Considering the 3D F_a:$8 \times 8 \times 32$, the combination of different data scheduling strategies I to VII are analyzed on the worst-case input load. From the strategies analysis, strategy-V (depth-wise = 3 times parallelly and row/column-wise = 4 times sequence) gives a promising outcome compared to the other strategy combination (I to IV, and VI to VII). The proposed MATLAB-based analysis for the selection of a 3D feature array and data-flow scheduling strategy for an efficient CNN-based inference engine is depicted in Fig. 2. The optimum 3D feature array (F_a) selection and data flow scheduling strategy of V are explained in the subsection.

2.1 Choosing the Optimum 3D Feature Array (Fa)

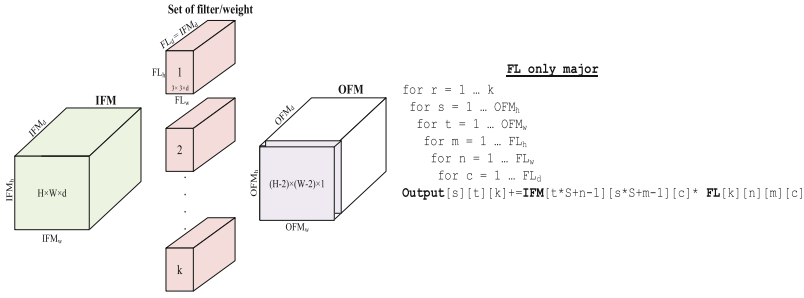
Based on the factors of an efficient CNN accelerator, we present the selection of feature arrays for the on-chip memory techniques that optimize the off-chip memory bandwidth as a result of the computations. There are two standard potential feature array computation strategies (1) IFM major feature array computation and (2) weight/filter major feature array computation that utilizes data reuse of either the weights or the IFMs for the convolutional layers are provided in this section

In IFM major computation approach [16], each set of weights $FL_w \times FL_h \times FL_d$ performs the element-wise multiplication with each of the K weights/filters, producing the OFM (1×1) of length K. This computation operation is replicated for each OFM, where the IFM frames are shifted by a stride ($S = 1$). In the standard approach, the weight values are reused for every iteration of the output (OFM_h, OFM_w) as shown in Fig. 3(a). In the filter-major computation approach in [16], one set of filter/weight is convolved across the entire IFM frames ($IFM_w \times IFM_h \times IFM_d$), producing a single OFM frame of size $OFM_w \times OFM_h$. This computation operation is repeated for each of the k sets of filters. In this approach, the IFM values are reused multiple times for each set of filter/weights as shown in Fig. 3(b). In Fig. 3(b), IFMs and weights are processed through single frame-wise computation. So, it requires larger memory issues to store the IFMs and weights as well as an intermediate buffer to store the partial OFMs. Considering these approaches, we proposed the optimum 3D structure computation gives a more advantageous data reusing approach on both IFM and weights as shown in Fig. 3(c). The 3D structure depends on the depth *'p'* of the IFMs and weights. The filters/weights are sized in this method so that the working set of filters ($FL_a = FL_h \times FL_w \times p$) fits on the on-chip memory (288 KB). In the same way, we need to size the IFM to be able to hold only the 3D feature array (F_a) size of IFM, namely $F_a = F_{ah} \times F_{aw} \times p$. For example, $F_{ah} \times F_{aw} \times p = 8 \times 8 \times 32$ with a 16-bit fixed-point required 4 KB. This ensures that each input feature map is only read from off-chip once. We can process all

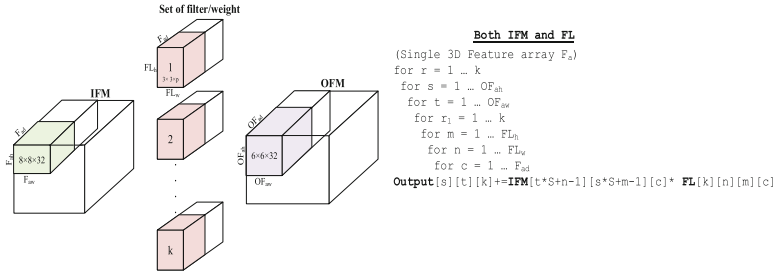
frames in the F_a with all on-chip resident filters while loading the next set of F_a and its corresponding filters/weights in parallel, resulting in the next set of IFM and filters based on F_a size on the on-chip by the end. This method has improved the reuse of both the IFM values k times and weight values OF_{aw} times to reduce off-chip bandwidth usage.



(a) Input feature map- reuse-based computations



(b) Filter/weight- reuse based computations



(c) Improved both Input feature map and filter/weight reuse-based computations

Fig. 3. Different combinations of IFM & FL reuse computation and its computation order.

For both IFM and weight reuse approach, the different size of the 3D feature array (F_a) is formed based on the convolutional layer input load. The different size of $F_a = F_{ah} \times F_{aw} \times p$ varies from $(6 \times 6 \times 8)$ to $(20 \times 20 \times 32)$. F_{ah} and F_{aw} are varying by the addition of 2. Likewise, p is varying by multiplies of 2. p denotes the depth of the 3D feature (F_a) array. F_{ah} and F_{aw} denote the height and weight of the 3D feature array (F_a). Worst-case input load ($IFM : 55 \times 55 \times 512$ and $FL : 3 \times 3 \times 512 \times 512$) has been processed with each combination of F_a. Each 3D F_a combination has undergone the

trade-off analysis in terms of data reuse (both IFMs and weights), memory requirement, and the number of times F_a requirement per input load using a MATLAB-based analysis as plotted in Fig. 4. MATLAB-based analysis helps to understand the impacts on each 3D feature array (F_a). From the understanding, $8 \times 8 \times 32$ has balanced the required key factors optimally. $10 \times 10 \times 8$ has next close to a better feature array. But data reuse and the number of 3D-F_a are high compared to chosen feature array $8 \times 8 \times 32$. Equation (1) to (3) defines the number of times F_a processes depth-wise (x), row-wise (y), and column-wise (z) on the chosen convolutional layer in the CNN model. N refers to the depth of the IFM (IFM_d), p represents the depth of the 3D feature array, and P_d represents padding (0 or 1). For example, Input load IFM $= 16 \times 16 \times 64$ & FL $= 3 \times 3 \times 9 \times 64$, then F_a requires 2 times for depth wise (x) computation, 3 times for row-wise (y) and column-wise (z) computation. The F_a required $3 \times 3 \times 2$ times for the $16 \times 16 \times 64$ and each F_a output size ($OF_a = 6 \times 6 \times 9$). In this approach, an optimum 3D feature array has improved the data reuse on both IFM and weights as well as minimizes the off-chip bandwidth access and maximizes the hardware utilization of the PE array

$$x = \lceil \frac{N}{p} \rceil \; where, p = 32 \tag{1}$$

$$y = \lceil \frac{IFM_h + P_d}{F_{aH} - 2} \rceil, row - wise P_d = [0, 0] \tag{2}$$

$$z = \lceil \frac{IFM_w + P_d}{F_{aW} - 2} \rceil, column - wise P_d = [0, 0] \tag{3}$$

Furthermore, $8 \times 8 \times 32$ feature array size is used for the different combinations of data-flow scheduling strategies to reduce the intermediate memory utilization and efficient PE computation as explained in the next section.

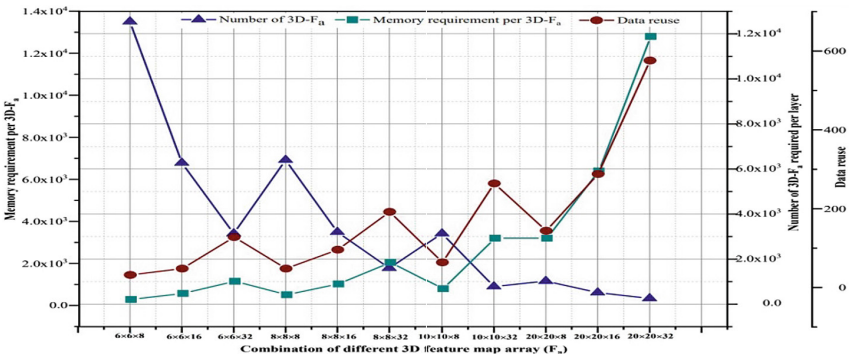


Fig. 4. Optimum feature array, F_a plot for the worst-case input load from the CNN model.

2.2 Data-Flow Scheduling Using 8 × 8 × 32 3D Feature Array

Once chosen the optimum 3D feature array (F_a), the hardware CNN-based inference engine can also be improved by data movement and scheduling strategies between on-chip memory to processor element array/on-chip to off-chip memory efficiently. Therefore, the different combinations of data movement and scheduling have been examined based on the parallel/sequence order to process the PEs/MACs computation for the chosen convolutional layer. Strategies (I to VII) strategies have been validated using the x, y, and z values from the optimum 3D feature array formation. Each strategy outcome has been understood by the improvements of the major key factors as follows: (1) processor hardware cost, (2) data transfer rate, and (3) Number of times data transactions between on-chip memory to the processor array. MATLAB-based mathematical analysis for processor array cost and data transfer rate were done based on the above-mentioned factors as described in Eqs. (4) are related to the on-chip storage, and processor array cost in terms of unit cost using the 3D feature array. According to the 3D- F_a approach, the estimation of the data transfer (number of times) for the IFM and filter is based on $(x.y.z)$ and x. $M_{on-chiptoPE}$ refers to the size of the memory access required in the on-chip. $D_{tperlayer}$ refers to the number of times data is transferred from off-chip to on-chip memory/buffer. $OF_a = (F_a - 2) \times (F_a - 2)$ refers to the output feature map and intermediate registers. PE_{area} refers to the number of PEs present in the processor, $FL_a = 3 \times 3 \times p$ refers to the size of the weight/kernel required for processing the F_a

$$M_{on-chiptoPE} = \left((F_a + OF_a + FL_a)_{memory\ access} + PE_{area}\right) \tag{4}$$

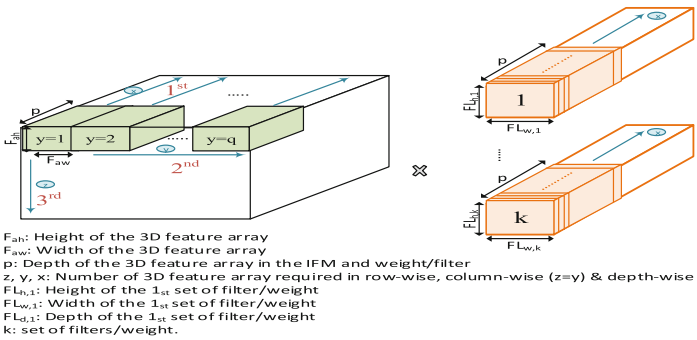


Fig. 5. Strategy-V (depth-wise 'x' parallel and row/column-wise 'q' sequence): Data-flow movement and scheduling method with k set of filters using F_a.

Considering the different strategies, strategy V archives a better data transfer rate and on-chip memory compared with strategies (I to IV & VI to VII) as shown in Table 1. The analysis of different combinations has been done in a MATLAB environment with consideration of different CNN parameters. The $M_{on-chiptoPE}$ and $D_{tperlayer}$ are calculated in terms of unit cost and unit data transfer rate in unit times. So, the evaluation is common for all the data formats and word lengths. Strategies are adopted to different

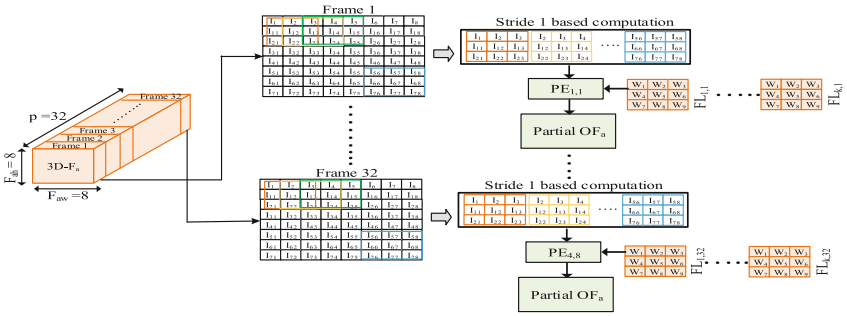


Fig. 6. Visual depiction of the frame-wise computation based on the 3D- F_a with strategy V approach.

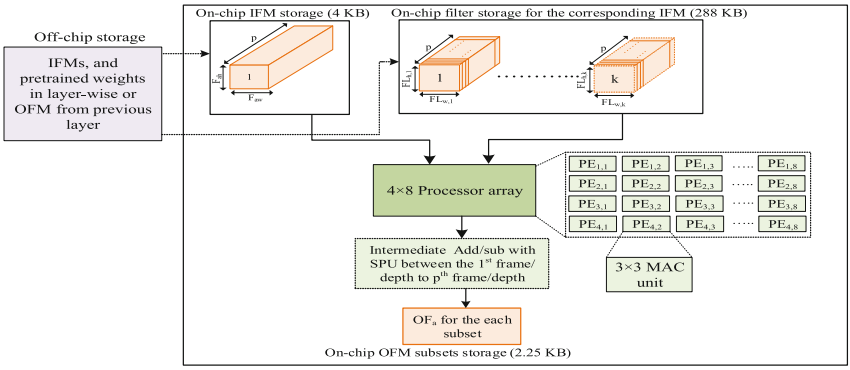


Fig. 7. Overall CNN inference engine for strategy V with 4 × 8 processor array.

feature array sizes in the standard CNN model such as AlexNet, VGG-16 & 19, ResNet, SqueezeNet, and GoogleNet

The scheduling strategy of V is illustrated in Fig. 5. In Fig. 5, the 3D- F_a is convoluted depth-wise 'x' with a k set of filters based on the *x*, and then F_a follows the row-wise and column-wise based on the y and z values. For example, input load ($IFM : 55 \times 55 \times 512$ and $FL : 3 \times 3 \times 512$) has been processed the depth-wise-parallel at x = 16 and row-wise/column-wise sequence at y = z = 10. It means that the data transfer rate for IFM and weight is 1600 times and 16 times. These 3D feature arrays have been processed with 4×8 processing elements. The processor contains 32 PEs. Each PEs is consisting of a 3×3 MAC unit and adopts a 16-bit fixed point. Under this circumstance, an optimum $8 \times 8 \times 32$- 3D feature array with the set of filters calculates the OFM using the PE array. It performs the MAC computation with consideration of the depth (p) of F_a, as listed in Algorithm I

Algorithm I for subset-1 using optimum 8×8×32 3D-F_a structure

(a) Calculate 4×8 processor

for X in 1 to x do
for (L, K) in (0,0) to (F_{aw} -2, F_{aH}-2) do
p=32, FL_w= FL_h= 3
PE_{p-31}=$Frame_1$(L_X+1:FL_h+L_X, K_X+1:FL_w+K_X)*FL_X
PE_{p-30}=$Frame_2$(L_X+1:FL_h+L_X, K_X+1:FL_w+K_X)*FL_X
PE_{p-29}=$Frame_3$(L_X+1:FL_h+L_X, K_X+1:FL_w+K_X)*FL_X
⋮
PE_p=$Frame_{32}$(L_X +1:FL_h+L_X, K_X+1:FL_w+K_X)*FL_X
I_n=add/Sub (PE_{p-31}, PE_{p-30}, PE_{p-29},…, PE_p)
end
end
return I_n

(b) Calculate OFM

for (Y, Z) in (1, 1) to (y, z) do
for X in 1 to x do
calculate 4×8 processor (I_n)
OFM+=I_n
end
end
return OFM

With each PE processing a 2-D frame, multiple PEs can be aggregated to complete the 3D input with corresponding shown in Fig. 6. The 3D feature array of size 8 × 8 × 32 has *p* frames. Each frame is processed with the corresponding set of weights using the stride '1' and produces the partial output feature maps. Each partial sum from each is further accumulated across the PEs vertically. The partial OF_a of all the frames is added together to extract the OFM. Likewise, the next set of 3D- F_a is processing the same. For example, $PE_{1,1}$: Frame 1 from the 3D- F_a of size 8 × 8 is convoluted with the weight of size 3 × 3 ($FL_{1,1}$ to $FL_{k,1}$), $PE_{2,1}$: Frame 1 from the 3D- F_a of size 8 × 8 is convoluted with the weight of size 3 × 3 ($FL_{1,1}$ to $FL_{k,1}$) up to frame 32. All the frames in the 3D-F_a are processed using the 4 × 8 processing element parallelly and attain the reuse on both weights and IFMs. Initially, pre-trained IFMs and weights are stored in the off-chip memory (DRAM). Based on the 4 × 8 PEs size, on-chip memory (SRAM/ buffer size) for IFMs and weights are chosen as shown in Fig. 7. Finally, OFM has to be stored in the off-chip which is IFM to the next CNN layer

3 Simulation Results and Discussion

In this section, the optimum 3D feature array of size 8 × 8 × 32 and the approach of data-flow scheduling in depth-wise (*x*) parallel and row/column-wise (*y*) sequence under the strategy-V has been evaluated for the efficient CNN hardware implementation.

The proposed 3D- F_a and data-flow scheduling strategy-V has been adopted and tested with the critical layer (IFM : $55 \times 55 \times 512$, $Filter/weight$: $3 \times 3 \times 512$) present in the state-of-the-art CNN models such as VGG-16 &VGG-19, ResNet-18 &50, and SqueezeNet. The critical layer results are evidence that scheduling strategy-V is suitable for the other layers present in the CNN models. Also, the results demonstrated that our data-flow scheduling strategy-V with optimum 3D feature array can efficiently reduce the off-chip to on-chip data movement and processor cost with higher utilization of the PEs

3.1 Evaluation of 3-D Feature Array (F_a) for the CNN Model

Array ($8 \times 8 \times 32$), the major key factors are followed as explained in Sect. 2. The software-based model for different pre-trained CNN models has been implemented using the MATLAB environment as depicted in Fig. 2. The software environment has the flexibility to adapt the different data formats and word lengths to maintain the accuracy of the model before implementation in the hardware module. In this subsection, the optimum feature array size of $8 \times 8 \times 32$ is selected based on the memory requirement per F_a and the number of times F_a is required per layer and data reuse. By varying the 11 different combinations of 3-dimensional feature arrays, the optimum size has been chosen from the trade-off plot as shown in Fig. 4. Considering all the factors, $8 \times 8 \times 32$ reuses the k set of filters/weights by $k \times 6$ times and reuses the IFM k times per F_a, and storage required for IFMs and weights of size 0.25KB and 18KB, respectively. Likewise, $6 \times 6 \times 32$ reuses the k set of filters/weights by $k \times 4$ times with a small area overhead. From the plot Fig. 5., $8 \times 8 \times 32$ achieves the better trade-off of the abovementioned three factors compared to the other 3D feature array. In the existing method [6, 14, 16], IFM major and filter/weight major reuse computations are followed in the CNN models as shown in Fig. 2. IFMs major computation requires a larger memory size of $3 \times 3 \times k \times N$ for the set of the filters but our method requires the memory size of $3 \times 3 \times p \times k$. compared to the [16], on-chip memory requirement has 16 times lesser and 2 times higher data transfer rate in our optimum method. Similarly, filter major computation has adopted all the IFM values in the on-chip. It is an expensive approach compared to our optimum 3D feature array

Likewise, [14] has adopted the depth-wise data-flow strategy. Therefore, this method has implemented that the IFM size is the same as the filter/weight size, so the weight reuse method has not been adopted. But the depth-wise strategy optimizes the memory with maximum utilization of PEs. [6, 15] uses the row-stationary (RS) and Weight stationery (WS) data-flow strategy to reduce the expensive data movement by reusing the data maximally and this strategy are suitable for efficient DRAM access. Considering all the methods, our optimum 3D feature array is suitable for the trade-off of the above-mentioned prime factors which are the major parameters in the efficient CNN hardware models

3.2 Analysis and Evaluation of Strategies for the CNN Layer

To improve the data-flow scheduling between the memories, the different combinations of strategies are framed based on the data transfer rate (off-chip to on-chip memory)

and processor cost (on-chip memory and processor area). From the strategy (I to VII) analysis, strategy-V is chosen by considering the trade-off of the key factors as discussed in Sect. 2.2. The specification for each supported data-flow scheduling strategy is listed in Table 1

Generally, the data-flow scheduling depends on the stage-wise row/column and depth in a parallel/sequence manner from the off-chip memory to the on-chip memory for the CNN models. The IFM and weights of each CNN layer bring with deluge amount of data. Therefore, the different combinations of stage-wise data scheduling are arranged with parallel or sequence mode using a 3D feature array (F_a) as illustrated in Table 1. Processor cost ($M_{on-chiptoPE}$) and data tranfer rate ($D_{tperlayer}$) are evaluated from each scheduling combination. Strategy V gives a better trade-off between two major parameters as shown in Fig. 8. The analysis is done in software-based implementation. Hence, the unit for the $M_{on-chiptoPE}$ and $D_{tperlayer}$ are in terms of unit cost and unit times

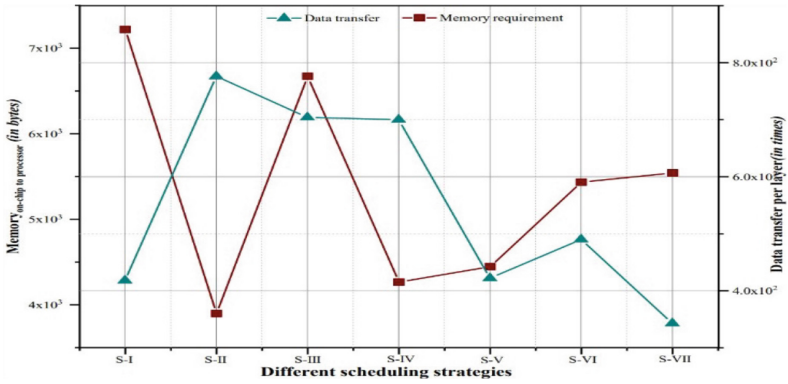


Fig. 8. Trade-off plot for cost and data transfer rate in the different scheduling strategies (I to VII) using the (8 × 8 × 32) 3D- F_a.

Generally, the data-flow scheduling depends on the stage-wise row/column and depth in a parallel/sequence manner from the off-chip memory to the on-chip memory for the CNN models. The IFM and weights of each CNN layer bring with deluge amount of data. Therefore, the different combinations of stage-wise data scheduling are arranged with parallel or sequence mode using a 3D feature array (F_a) as illustrated in Table 1. Processor cost ($M_{on-chiptoPE}$) and data tranfer rate ($D_{tperlayer}$) are evaluated from each scheduling combination. Strategy V gives a better trade-off between two major parameters as shown in Fig. 8. The analysis is done in software-based implementation. Hence, the unit for the $M_{on-chiptoPE}$ and $D_{tperlayer}$ are in terms of unit cost and unit times

In Table 1, Strategy-I [14] has adopted first- row-wise, depth-wise, and then column-wise in a sequence manner. This strategy I and II give better processor memory but the data transfer rate is higher (3.3 times) due to the off-chip to on-chip data-flow scheduling than strategy-V. Similarly, strategy III is accepted both parallelly and sequence transfer and it achieves a data transfer rate the same as strategy V. But, moderately higher in the processor cost. Strategy-VI [6, 15] uses the weight stationary (kernel reuse) and row stationery (input reuse) which utilizes more processor cost (1.05 times) than strategy V.

But the data transfer rate (1.44 times higher) has the same moderate as in the strategy V. Compared to all the combination of the data-flow scheduling, strategy-V with optimum 3D feature array achieves the better trade-off between the data transfer and processor cost

Table 1. Different combinations of scheduling strategies with a 3D feature array of 8 × 8 × 32.

Strategy	Different scheduling methods using 3D- Feature array of size 8 × 8 × 32			$M_{on-chiptoPE}$ in terms of unit cost	$D_{tperlayer}$ in terms of unit times
I [14]	1st: Row-wise, *y* (sequence)	2nd: Depth-wise, *x* (sequence)	3st: Column-wise, z (sequence)	Low	High
II	1st: Column-wise, z (sequence)	2nd: Depth-wise, *x* (sequence)	3st: Row-wise, *y* (sequence)	Low	High
III	1st: Depth-wise, *x* (sequence)	2st: Column-wise, *z* (parallel)	3st: Row-wise, *y* (sequence)	Moderate	Moderate
IV	1st Row-wise, *y* (parallel)	2nd: Column-wise, *z* (parallel)	3nd: Depth-wise, *x* (sequence)	High	Low
V	1st: Depth-wise, ***x*** (parallel)	2st: Row-wise, ***y*** (sequence)	3st: Column-wise, ***z*** (sequence)	Low	Moderate
VI [6, 15]	1st: Row & column-wise, *y* × *x* (sequence)	2nd: Depth-wise, *x* (parallel)	-	High	Moderate
VII	1st: Row-wise, *y* (parallel)	2nd: Depth-wise, *x* (parallel)	3st: Column-wise, *z* (parallel)	High	Low

Note:
1. A feature block of size 55 × 55 × 512 with a kernel size of 3 × 3 × 512 is considered for analysis.
2. Everything on-chip produces 100% high memory with a 0% low transfer rate from off-chip to on-chip.
3. Considering this as a baseline, the memory and data transfer rate for the strategies (I to VII) are categorized into: High is 70 to 100; Moderate is 30 to 70, and Low is 0 to 30.

4 Conclusion

In this paper, optimization of the data-flow approaches for the CNN models has been discussed. Data-flow complexity depends on the size of on-chip memory and its controller in the processing elements (PEs). Data reuse, and data-flow strategies can influence the size of on-chip memory. First, to improve the data reuse on both IFMs and weights a 3D feature array (F_a) of size $8 \times 8 \times 32$ is proposed. Secondly, the different (I to VII) data-flow strategies using the proposed 3D feature array are analyzed in the understanding of the requirements of processor cost and data transfer rate (off-chip to on-chip). From the analysis of data scheduling strategies, strategy V uses the optimum number of PEs and on-chip memory size. It works on the input features and weights of the CNN workload layer in a depth-wise parallelly and row/column-wise sequence manner. The proposed strategy-V utilizes the 3-dimensional feature array size with considering the examination of prime factors including the number of F_a required per CNN workload layer, data reuse, and PEs utilization. Also, the strategy-V data-flow model of $8 \times 8 \times 32$ size provides a better trade-off between the processor cost in terms of unit cost and data transfer rate in terms of unit times. The MATLAB- based software analysis shows that the strategy-V can achieve the data transfer rate by 3.3 times (faster) compared to the Eyeriss processor data-flow strategy with a small area overhead and the requirement of on-chip memory has 16 times (lesser) than the Mem-Opt processor. In this paper, we present heuristic design principles that aim to optimize for particular dataflow scenarios. So, the scope of the future is to implement the optimum 3D feature array (F_a) and data scheduling based on strategy V in hardware inference for pre-trained CNN models

Acknowledgment. The work was supported in part by the Council of Scientific and Industrial Research (CSIR), New Delhi, under CSIR-SRF Grant 09/844(0104)/2020- EMR.I.

References

1. Hatcher, W.G., Yu, W.: A survey of deep learning: platforms, applications and emerging research trends. IEEE Access **6**, 24411–24432 (2018)
2. Moolchandani, D., Kumar, A., Sarangi, S.R.: Accelerating CNN Inference on ASICs: a survey. J. Syst. Architect., preprint, Sept. (2020)
3. Chen, Y., Xie, Y., Song, L., Chen, F., Tang, T.: A survey of accelerator architectures for deep neural networks. Engineering **6**(3), 264–274 (2020)
4. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017)
5. Peemen, M., Setio, A.A., Mesman, B., Corporaal, H.: Memory-centric accelerator design for convolutional neural networks. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 13–19. IEEE (2013)
6. Chen, Y.H., et al.: Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J. Solid-State Circuits **52**, 127–138 (2017)
7. Moons, B., Uytterhoeven, R., Dehaene, W., Verhelst, M.: 14.5 envision: a 0.26-to-10tops/w subword-parallel dynamic-voltageaccuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI. In: 2017 IEEE International Solid-State Circuits Conference (ISSCC), pp. 246–247. IEEE (2017)

8. Han, S., et al.: EIE: efficient inference engine on compressed deep neural network. In: Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA), Seoul, South Korea, Jun, pp. 243–254 (2016)
9. Parashar, A., et al.: SCNN: an accelerator for compressed-sparse convolutional neural networks. In: Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Jun. 2017, pp. 27–40
10. Yuan, Z., et al.: STICKER: an energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm CMOS. IEEE J. Solid-State Circuits **55**(2), 465–477 (2020)
11. Kang, H.J.: Short floating-point representation for convolutional neural network inference. IEICE Electronics Express, pp. 15-20180909 (2018)
12. Deepika, S., Arunachalam, V.: Analysis & design of convolution operator for high speed and high accuracy convolutional neural network-based inference engines. IEEE Trans. Comput. **71**(2), 390–396 (2021)
13. Yue, J., et al.: A 3.77 TOPS/W convolutional neural network processor with priority-driven kernel optimization. IEEE Trans. Circuits Syst. II, Exp. Briefs **66**, 277–281 (2018)
14. Dinelli, G., Meoni, G., Rapuano, E., Pacini, T., Fanucci, L.: MEM-OPT: a scheduling and data re-use system to optimize on-chip memory usage for CNNs On-Board FPGAs. IEEE J. Emerg. Selected Top. Circ. Syst. **10**(3), 335–347 (2020)
15. Chong, Y.S., Goh, W.L., Ong, Y.S., Nambiar, V.P., Do, A.T.: An energy-efficient convolution unit for depthwise separable convolutional neural networks. IEEE Int. Symp. Circ. Syst. (ISCAS) **2021**, 1–5 (2021)
16. Siu, K., Stuart, D.M., Mahmoud, M., Moshovos, A.: Memory requirements for convolutional neural network hardware accelerators. In: 2018 IEEE International Symposium on Workload Characterization (IISWC), pp. 111–121. IEEE (2018)