



# USEFUSE: Uniform stride for enhanced performance in fused layer architecture of deep neural networks<sup>☆</sup>

Muhammad Sohail Ibrahim<sup>a,1</sup>, Muhammad Usman<sup>b,\*,1</sup>, Jeong-A Lee<sup>c,\*</sup>

<sup>a</sup> Department of Mechanical Systems Engineering, Kumoh National Institute of Technology, Gumi-si, Republic of Korea

<sup>b</sup> Faculty of Informatics and Data Science, University of Regensburg, Regensburg, 93053, Germany

<sup>c</sup> Department of Computer Engineering, Chosun University, Gwangju, Republic of Korea

## ARTICLE INFO

### Keywords:

Convolution neural network  
Online arithmetic  
Most-significant-digit-first arithmetic  
CNN acceleration  
Layer fusion

## ABSTRACT

Convolutional Neural Networks (CNNs) are crucial in various applications, but deploying them on resource-constrained edge devices poses challenges. This study presents the Sum-of-Products (SOP) units for convolution, which utilize low-latency left-to-right bit-serial arithmetic to minimize response time and enhance overall performance. The study proposes a methodology for fusing multiple convolution layers to reduce off-chip memory communication and increase the overall performance. An effective mechanism detects and skips inefficient convolutions after ReLU layers, minimizing power consumption without compromising accuracy. Additionally, efficient tile movement guarantees uniform access to the fusion pyramid. An analysis demonstrates the uniform stride strategy improves operational intensity. Two designs cater to varied demands: one focuses on minimal response time for mission-critical applications, and another focuses on resource-constrained devices with comparable latency. This approach notably reduced redundant computations, improving the efficiency of CNN deployment on edge devices.

## 1. Introduction

Deep neural network (DNN) is an artificial neural network comprised of several layers between input and output layers. They have been widely used in image recognition [1], semantic segmentation [2], medical imaging [3], bioinformatics [4], and signal processing [5] etc. A class of DNN is the convolution neural networks (CNNs) which play a pivotal role in many applications such as computer vision, recognition, object detection, etc. This has been made possible due to the advancements in high performance computing technologies and the availability of cutting-edge compute resources. The use of CNNs with many layers has enabled the swift progress in a number of diverse application domains. CNN designs, inspired by the behavior of optic nerves in human brain, perform data processing in multiple layers of neurons to achieve human brain-like performance in image recognition.

There is a pressing need to execute complex neural networks in mission-critical applications with low latency demands. However, due to limited compute and storage resources, the implementation of neural networks on edge devices is limited [6]. Various efforts have been made to reduce the complexity of neural networks at algorithm level,

including compression by pruning, quantization, approximation, zero skipping, etc., at the expense of accuracy [7–9]. Furthermore, several spatial architectures exploiting the effective data such as weight stationary [10], output stationary [11] and row stationary [12], have been proposed to accelerate the computation of neural networks.

In the context of network compression for resource-constrained hardware, serial processing is usually favored in DNN implementation where the models can have layer-specific input precision for either activation or weight (bit-serial with one operand in parallel) [13, 14] or both (bit-serial with both operands in serial) as in [15]. Parallel processing accounts for larger area, whereas serial approaches have longer computation time. Bit-serial designs however, require simpler circuitry and the adjustable precision makes them favorable for domain-specific hardware accelerators. Bit-serial designs also suffer from high latency and low throughput issues which are usually addressed by employing multiple instances of small serial circuits to deliver higher throughput [13].

In most modern CNNs, convolution accounts for more than 90% of operations. Although the computation of the convolution operations is

<sup>☆</sup> This research was supported by Basic Science Research Program funded by the Ministry of Education through the National Research Foundation of Korea (NRF-2020R1I1A3063857). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

\* Corresponding authors.

E-mail addresses: [msohail@kumoh.ac.kr](mailto:msohail@kumoh.ac.kr), [msohail@chosun.ac.kr](mailto:msohail@chosun.ac.kr) (M.S. Ibrahim), [muhammad.usman@ur.de](mailto:muhammad.usman@ur.de) (M. Usman), [jalee@chosun.ac.kr](mailto:jalee@chosun.ac.kr) (J.-A. Lee).

<sup>1</sup> Muhammad Sohail Ibrahim and Muhammad Usman contributed equally to this work.

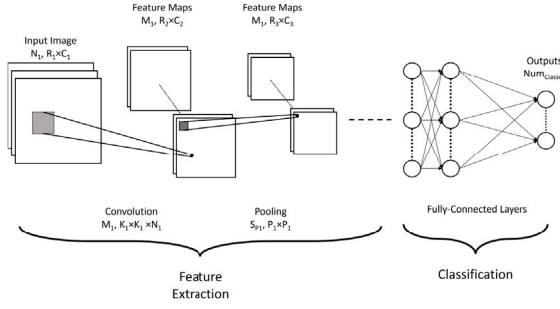


Fig. 1. A general CNN architecture.

very simple, involving multiplication and addition, but due to the depth of such networks the computational complexity increases, subsequently raising the number of operations. It is found that nearly 85% of the overall time in a CNN-based classification model is consumed by these multiplication and addition operations to perform convolution in the DaDianNao accelerator [13].

A generic CNN architecture is illustrated in Fig. 1, in which the feature extraction module includes stacks of convolution, activation, and pooling layers while the classification module contains stacks of fully connected layers. Activation functions to add non-linearity are placed either after convolution layer or pooling layer. One of the simplest and most commonly used activation function in CNNs is the rectified linear unit (ReLU), which returns same value if the input is positive and zero if the input is negative. It is reported that around 60% of the convolutions produce negative output and therefore, their results are not used due to ReLU in the subsequent operations [16]. Such convolutions have no effect on the output and are termed as ineffectual convolutions. In order to mitigate the computation of ineffectual convolutions, researchers have proposed methods to detect such negative activation and terminate them at an early stage, consequently resulting in energy and power savings [17]. Several methods have been proposed to detect these negative results at an early stage and terminate them to reduce energy consumption [16,18,19]. However, it adds an overhead to the model. This overhead can be alleviated by utilizing an unconventional arithmetic known as *online arithmetic* [20], where all the computations are performed from left-to-right in most significant digit first (MSDF) manner.

Moreover, the information in the CNNs flows sequentially, i.e., the results from one layer are utilized by subsequent layers. This inter-layer connectivity forms the backbone of CNNs, allowing them to progressively uncover intricate features as the data travels deeper into the network. Conventional CNN accelerator implementations perform the network computation in a layer-by-layer manner where the data is repeatedly transferred between the processing units and memory. Such communication burden increases rapidly with the increasing depth of the network.

In this avenue, exploring the dataflow across the CNN layers can help in the reduction of memory traffic. Instead of layer-by-layer computation in a CNN, the CNN layers can be *fused* together to reduce the communication between memory and the compute engines [21]. In such architectures, it is well-defined in the literature that each pixel or receptive field in the activation of a particular hidden layer is dependent on a region in the input activation of the initial layer of the network. Using a layer fusion strategy can help in merging the operations of various subsequent layers, in-turn reducing the off-chip memory access substantially.

In this work, we develop a CNN evaluation scheme by fusing the convolution layers of the network to reduce the off-chip memory traffic due to the intermediate data. The proposed approach reduces the number of duplicate computations by an efficient uniform fusion

pyramid movement scheme aided by a uniform stride for each level of the fusion design. The goal is to select the smallest possible tile sizes for each layer in the fusion design, maintaining a uniform tile movement while ensuring minimum overlap between the adjoining tiles. The contributions of this study can be summarized as follows:

- Design of SoP units for convolution using low-latency left-to-right bit-serial arithmetic based computation units to minimize the response time.
- A methodology to fuse several layers of convolution neural network using the proposed SoP computation units to decrease the off-chip memory communication.
- Mechanism to early detect and skip the computation of convolutions which are ineffectual after ReLU layers without any loss in accuracy during inference to minimize the power consumption.
- A methodology for tile movement to ensure efficient data access and uniform movement of fusion pyramid.
- An analysis depicting that the proposed uniform stride strategy improves the operational intensity irrespective of the dataflow of the underlying computation units.
- Additionally, we propose two alternate designs possessing the aforementioned properties; (1) a design aimed at minimal response time for mission critical applications, (2) a design suitable for resource constrained devices with comparable latency as the contemporary approaches.

The rest of the paper is organized as: a comprehensive review of relevant literature is discussed in Section 2, the proposed CNN evaluation scheme is presented in the subsequent Section 3. The experimental results and relevant discussion is presented in Section 4 followed by the conclusion of the study in Section 6.

## 2. Related work

This section will review existing bit-serial architectures for CNN computation, followed by a discussion on fused-layer architectures. Finally, methods for early detection and termination of negative computations to enhance energy efficiency will be explored.

### 2.1. Bit-serial accelerators

Over the past decade, researchers have addressed various challenges in CNN acceleration, such as unnecessary computations and the need for variable precision across CNN layers [22,23]. These challenges can lead to increased energy and resource demands in accelerator designs. Stripes [13], a leading CNN acceleration design, employs bit-serial arithmetic compute units to exploit variable precision, thereby speeding up CNN inference.

The primary goal of bit-serial arithmetic designs is to reduce unnecessary computations and create energy-efficient accelerators. In this direction, Bitlet [24] proposed a bit-interleaving architecture that leverages bit-level sparsity and variable precision to accelerate DNN inference. A mixed-precision CNN accelerator is presented in [25], achieving high throughput with minimal accuracy loss by quantizing inputs and weights. Similarly, T-DLA [26] uses 2-bit quantized weights for performance improvements. TALIPOT [27] enhances energy efficiency using hybrid number representations in most significant bit first (MSBF) arithmetic units, allowing early operations in subsequent layers without waiting for complete computation results. Other variable precision, bit-serial computation-based designs include implementations such as [14,28,29].

Bit-serial designs provide advantages such as reduced memory bandwidth requirements and the ability to leverage variable precision across different DNN layers. However, these designs face drawbacks, including higher latency, lower throughput, and reduced performance compared to conventional bit-parallel architectures. Additionally, in bit-serial designs, the accumulation operation is hindered by carry propagation, which significantly increases cycle time and lowers the operating frequency of the processing units [30].

## 2.2. Fused-layer accelerators

Conventional CNN accelerator designs focus on iterative layer computations, generating large amounts of intermediate data. Depending on the design and tile size, this data can be intra-layer or inter-layer, requiring off-chip memory storage and retrieval for subsequent operations. As CNN models grow deeper, memory traffic increases. To address this, a novel accelerator design that \*fuses\* multiple CNN layers was introduced in [21], reducing intermediate memory traffic by directly feeding data between adjacent compute units, minimizing off-chip memory use by up to 95% for models like VGGNet-E.

Fused-layer architectures can also take advantage of variable precision requirements in different layers, improving efficiency. Bit-Fusion [15] introduced a flexible architecture where bit-level processing elements dynamically fuse to match precision needs, increasing speed and energy efficiency without accuracy loss. Efficient computation scheduling is critical for fused-layer designs, as highlighted by ConvFusion [31], which proposed a cost model for scheduling computation and memory communication, optimizing tiling, loop reordering, data reuse, layer fusion, and convolution execution schemes. Other layer-fusion-based designs include DeepThings [32], TGPA [33], and further approaches explored in [34–36].

The data flow between computation units and external memory presents significant design challenges and increased energy consumption due to the large volume of data generated during CNN convolution operations [21]. Fused-layer architectures attempt to mitigate this by reusing intermediate data, but exploring the design space for data scheduling, loop tiling, and loop reordering remains challenging.

Olympus [37] addresses memory access traffic by optimizing both intra-layer and inter-layer data reuse. It employs a memory-oriented network scheduling technique to reduce memory traffic and enhance energy efficiency in DNN processors. Other strategies for minimizing memory access and exploring accelerator design space include [38–41].

Despite the benefits of fused-layer dataflow, certain limitations remain. Many fused-layer designs overlook the stride of the fusion tile, which determines how the tile moves after computation. Incorrect stride determination can lead to excessive duplicate data being reused or recomputed, requiring large buffers or on-chip storage for intermediate data and causing significant under-utilization of compute resources. Storing intermediate data has been shown to be more energy-efficient than recomputation [21]. The need for large data buffers arises due to two factors: (1) ineffective computation of tile stride and (2) the use of conventional arithmetic units that fail to process the generated data immediately.

## 2.3. Early negative detection techniques

Rectified Linear Activation (ReLU) is a popular activation function in neural networks, which sets negative values to zero while keeping positive values unchanged. With the advancement in deep learning architectures, various derivatives of the ReLU activation functions have been proposed, such as PReLU [42], LeakyReLU [43], etc., while many recent architectures still rely on the ReLU activation [44–47]. Additionally, recent research suggests that ReLU can serve as a viable alternative to softmax, offering advantages in computation efficiency and parallelization. For instance, [48] proposed a ReLU based self-attention and feed-forward network to replace softmax in transformer models, showing that ReLU improves scalability by efficiently handling a large number of memory slots. Similarly, [49] replaced softmax with ReLU in vision transformers and demonstrated that the ReLU-based attention achieves comparable performance to softmax-based attention in terms of scaling behavior while enabling better parallelization over the sequence length dimension, reducing the need for gather operations. These findings indicate that ReLU is not only relevant but also increasingly explored as a substitute for more complex functions

in deep learning architectures. The introduction of ReLU in the network architecture facilitates faster convergence and helps address the vanishing gradient problem. However, ReLU introduces the issue of ineffectual convolutions, where a significant portion of a convolution layer's output consists of zero activations after applying ReLU. These zeros are propagated through the network without contributing to the final output, leading to wasted computational resources. This inefficiency consumes memory bandwidth, energy, and processing cycles, ultimately slowing down inference and increasing energy consumption.

As mentioned earlier, while many DNN acceleration techniques focus on designing fast and energy-efficient computation units, fewer approaches address the early termination of convolution operations due to ReLU activations. SnaPEA [17] introduced an early negative prediction scheme with two modes to address this: (1) Exact Mode: A single-bit sign check is performed iteratively on the sum of partial products, and computation stops as soon as the sum falls below zero, and (2) Predictive Mode: The partial sum is compared to a threshold, and computation is terminated if it drops below this threshold. This mode is faster but slightly reduces accuracy. Other methods aimed at early termination of convolution operations include ComPREND [16], TerminiNETor [50], CompRRAE [19], CompEND [18], BitSET [51], and [52].

Left-to-right or MSDF arithmetic operations can significantly enhance the early detection of negative activations. Shuvo et al. [53] proposed a novel circuit implementation for convolution that allows for early detection of negative results, enabling the subsequent termination of related operations. However, existing methods for early detection of negative activations often rely on digit encoding schemes, threshold-based predictions, or complex circuitry, which can result in erroneous decisions or increased overhead.

## 3. Materials and methods

To address the limitations of the existing works, we propose to utilize digit serial left-to-right arithmetic-based computation units, terminating the computation of ineffective convolutions at an early stage, and minimize the communication between memory and compute units by fusing several successive convolution layers. The details of which have been explained in the ensuing subsections.

### 3.1. Online arithmetic

In online arithmetic, computations proceed digit-by-digit, from the most to the least significant position, for both inputs and outputs. Algorithms require  $(j + \delta)$  input digits to compute the  $j$ th digit of the result, where  $\delta$  is the online delay, typically a small integer (1–4) depending on the operation. This method employs a redundant number system to generate the most significant digits first, making the cycle time independent of the working precision. Online algorithms involve recurrence relations where residuals are iteratively fed back into computations. The residual part from intermediate calculations contributes to generating subsequent output digits efficiently.

Online arithmetic enables the overlap of dependent operations, as the subsequent unit can begin computation once the most significant digit (MSD) of the preceding unit is available. In contrast, conventional digit-serial arithmetic requires all digits before starting. Although overlapping is possible in conventional systems if all operations use either MSDF or least significant digit-first (LSDF) modes, issues arise when combining MSDF (e.g., division) with LSDF (e.g., multiplication). Since online arithmetic consistently uses MSDF, it supports seamless overlapping of dependent operations. In conventional arithmetic, the subsequent unit can only begin computation if the output of the preceding unit is generated bit-by-bit and the subsequent unit also accepts input bit-by-bit. Otherwise, if it requires a parallel input, it must wait until the entire output is available. Online arithmetic, however, takes input serially and produces results serially, enabling a technique called

computation while communication, where processing and data transfer occur simultaneously, reducing latency and improving efficiency.

In parallel or pipelined systems where full-precision communication between modules is not feasible, online arithmetic excels due to its reduced bandwidth needs. This is particularly advantageous in signal processing applications where full-precision output is unnecessary. For instance, in multiplying two  $N$ -bit operands to generate a  $2N$ -bit result, often only the most significant half is required, as in many DSP applications. Conventional multipliers produce output starting from the least significant bits, discarding the lower half and wasting resources. In contrast, online arithmetic generates output digit-by-digit from the most significant side, allowing computation to stop once the desired precision is reached.

The computation from the most significant digit (MSD) to the least significant digit (LSD) relies on generating output based on partial information about the input operands. This flexibility is achieved by introducing redundancy in the input and output operands, which is why a redundant number representation system is used in online arithmetic. Typically, a signed digit (SD) redundant number system is employed, where numbers are represented in radix  $r$  form, and each signed digit belongs to the set  $\{-a, \dots, -1, 0, 1, \dots, a\}$  with the condition  $\frac{r}{2} \leq a < r$ . In this work, we utilize a symmetric radix-2 digit set with  $\{-1, 0, 1\}$ .

### 3.1.1. Online multiplier and adder overview

---

#### Algorithm 1 Serial-Parallel Online Multiplication

---

```

1: Initialize:
    $x[-2] = w[-2] = 0$ 
2: for  $j = -2, -1$  do
3:    $v[j] = 2w[j] + (x_{j+2} \cdot Y) 2^{-2}$ 
4:    $w[j+1] \leftarrow v[j]$ 
5: end for

6: Recurrence:
7: for  $j = 0 \dots n + \delta$  do
8:    $v[j] = 2w[j] + (x_{j+2} \cdot Y) 2^{-2}$ 
9:    $z_{j+1} = SELM(v[j])$ 
10:   $w[j+1] \leftarrow v[j] - z_{j+1}$ 
11:   $Z_{out} \leftarrow z_{j+1}$ 
12: end for

```

---

The fundamental component of the accelerator is the window processing unit (WPU), which serves as the core for computing convolutions. The WPU is composed of online multipliers and reduction trees based on online adders. In the online serial-parallel multiplier, one operand is fed in serially in a MSDF manner, while the other operand is a constant available in parallel at implementation time. A radix-2 serial-parallel online multiplier has an online delay of 2, and its selection function requires 2 fractional bits and 1 integer bit for output digit selection. Methods for developing online algorithms and derivations are discussed in [20]. The online multiplication algorithm generally consists of two steps: (1) Initialization, during which  $\delta$  input digits (in serial) are collected without generating any output, resulting in an execution length equal to the online delay ( $\delta$ ); and (2) Recurrence, which runs for  $n$  iterations, where  $n$  is the input precision, producing one output digit in each iteration. A pseudo-code for the online serial-parallel multiplication algorithm is presented in Algorithm 1.

Here,  $x$  and  $Y$  are the bit-serial and parallel inputs, respectively, and  $z$  is the serial MSDF output. The residual registers to store the temporary results are denoted by  $w$  and  $v$ . At any  $j$ th iteration, the serial output digit (input digit) is represented by  $z_j$  ( $x_j$ ), where  $z_j = SUB(z^+, z^-)$ , such that the subtraction of the two bits represents the value of the digit.  $SELM(\cdot)$  is the output selection module/function

that selects an output from a look-up table on the basis of a few most significant ( $r$ ) bits of the residual.

Serial online addition involves full adders and registers to add two redundant numbers in a MSDF manner. A detailed description of the online adder and its relevant derivations can be found in Ercegovic and Lang [20]. Additionally, this reference provides the design and methodology for the online serial-serial multiplier, where both input operands are supplied as serial inputs. In this work, we utilize the online serial-parallel multiplier proposed in our previous research [54]. This online serial-parallel multiplier is employed to design the processing units in the proposed USEFUSE accelerator, with further details and derivations available in [54].

### 3.2. Early termination of negative computations

Most CNN accelerator designs concentrate on efficiently generating the SOP for the activation layer (ReLU). However, few studies have investigated the early detection of negative values in the SOP, which presents a significant challenge in accelerators based on conventional arithmetic. For instance, conventional bit-serial multipliers take the multiplicand in parallel while processing the multiplier serially. In each iteration, a partial product is generated and stored in a register, then shifted into the appropriate position before being added to other partial products to compute the final result. This process typically involves a series of adders for reduction. A second level of reduction is necessary to add  $k \times k$  products to yield the output pixel, along with an additional level of reduction for summing multiple input channels. With conventional bit-serial multipliers, the most significant bit and the polarity of the result cannot be determined until all partial products have been generated and added to the previous partial sums.

The challenge of early detection and termination of negative activations can be addressed by the intrinsic ability of online arithmetic to generate output digits in an MSDF manner. The proposed design supports the termination of negative activation computation in  $p < \mathcal{N}$  cycles, where  $\mathcal{N}$  is the number of cycles to compute complete result. This is done by observing the output digits. The process of detecting the negative activations and subsequently terminating the relevant computation is summarized in Algorithm 2.

---

#### Algorithm 2 Early detection and termination of negative activations

---

```

 $z_j^+, z_j^-$  bits
for  $j = 1$  to  $\mathcal{N}$  do
   $z^+[j] \leftarrow z^+[j] \cup z_j^+$ 
   $z^-[j] \leftarrow z^-[j] \cup z_j^-$ 
  if  $z^+[j] < z^-[j]$  then
    Terminate
  else
    Continue
  end if
end for

```

---

The proposed early negative detection unit (END-U) is equipped with registers to store  $z_j^+$  and  $z_j^-$  bits, which represent the positive and negative output bits of the SOP in redundant form. During each iteration, new bits are appended to their respective registers. As soon as the value of  $z^+[j]$  falls below the value of  $z^-[j]$ , a termination signal is generated, resulting in the cessation of the SOP computation. The END-U is integrated into each processing unit, as described in Section 3.4.

### 3.3. Proposed layer fusion method

This section outlines the proposed layer fusion method and its components, including the calculation and selection of tile sizes and the calculation of the uniform stride for tiles. A comprehensive description



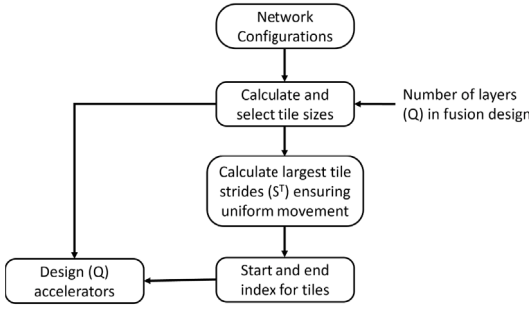


Fig. 2. Proposed layer fusion accelerator design pipeline.

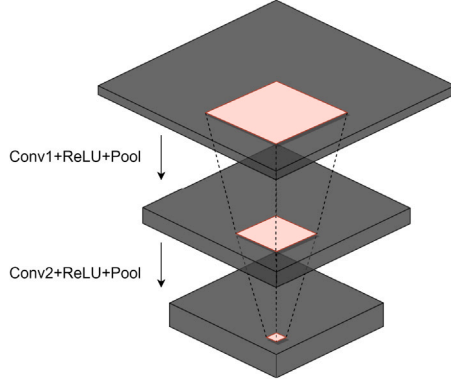


Fig. 3. General layer fusion scheme.

of the proposed design flow is presented in Fig. 2. The design flow begins by taking the CNN network configurations and the number of  $Q$  convolution layers intended for the fusion design, followed by the calculation of tile sizes for each layer. This is followed by calculating the uniform stride to ensure uniform tile movement across the respective layers in the fusion design. Next, the start and end indices of the feature maps intended for each layer are determined. The information accumulated throughout this process is then utilized to design accelerators for each layer in the fusion design, with detailed descriptions of these processes provided later in this section.

### 3.3.1. Overview

The proposed design follows a layer fusion scheme as depicted in Fig. 3, where a particular region, referred as *Tile*, is selected by tracking the output activation (or a region) of the final layer of the fusion pyramid to the first layer. The dimensions of the tile depends on the CNN architecture as well as the dimensions of the intended region of the output feature map.

The pyramid dimensions are calculated by selecting a suitable region of the output feature map and the tile dimension of its preceding layer according to relation (1), presented in [21].

$$D_l = (D_o - 1) \times S_l + K_l \quad (1)$$

where  $D_l$  is the dimension of the layer preceding the output layer of the fusion pyramid,  $D_o$  is the dimension of the selected region of the output feature map,  $S_l$  and  $K_l$  are the stride and kernel size of the layer preceding to the output layer, respectively. This procedure is done from the final layer until the first layer of the fusion architecture to obtain the tile sizes of the respective layers in the fusion pyramid.

Consider an example of a simple CNN such as LeNet-5 whose first two convolution layers are to be fused. Each convolution layer is followed by a sub-sampling layer, like Maxpooling. In a fusion of two convolution layers,  $R = C = 1$  output pixels from the second sub-sampling layer serve as input to the third layer. To determine tile

dimensions in the fusion pyramid, Eq. (1) applies to both convolution and sub-sampling layers. For instance, the input to the third layer follows a Maxpooling operation ( $MPL2$ ) on a  $2 \times 2$  output from the second convolution layer ( $CL2$ ), which operates on a  $6 \times 6$  input. Tracing back,  $MPL1$  requires a  $12 \times 12$  input to produce this, derived from a  $16 \times 16$  input to  $CL1$ . The generation of neighboring pixels at the same level requires a separate, overlapping pyramid computation. The starting index for each layer in this process, known as the *tile stride*, differs from the convolution stride. Determining this *tile stride* is crucial for two reasons: (1) It ensures the fusion pyramid covers the entire input feature map without skipping pixels, generating all necessary output activations. (2) It guarantees consistent execution rounds at every pyramid level, removing the need for synchronization after each round [33].

Furthermore, in most CNN models, the feature map dimensions are downsampled along the depth of the network while the number of filters increase. The proposed scheme ensures reduction in memory traffic in earlier as well as later convolution layers. This is due to the reason that the proposed design incorporates input and output channel tiling [55]. This means that the filters are loaded into the kernel buffers only once, while the input feature map sections are loaded into the input buffers as the fusion tile moves across the input feature map.

To this end, we propose an algorithm in Section 3.3.2 for the calculation of the *tile stride* to ensure a uniform movement of the fusion pyramid for various output region configurations including the tile dimensions for each layer in the fusion pyramid. It is also worth noting that this work focuses on the assumption that the tile at each pyramid level is square-shaped, which is most commonly used.

### 3.3.2. Algorithm

The pseudocode presented in algorithm 3 depicts a simple framework for calculating the fusion tile sizes of any network using Eq. (1). It takes the name of the network and the number of layers ( $Q$ ) intended for fusion as its input and returns the fused-layer tile sizes for all possible squared output dimensions in the output feature map of the final layer in the fusion pyramid. It ensures that the tile size  $H$  for each layer in the fusion design is bounded by the size of the input feature map ( $IFM$ ) of the respective layer. The *For* loop iterates over the various squared dimensions of the output feature map ( $R_Q$ ) of the fused-layer design and results in an  $(R_Q \times Q)$  matrix consisting of tile sizes  $(H_Q, H_{Q-1}, \dots, H_1)$  for each layer in the fusion pyramid. This results in all possible fused-layer tile configurations considering that the tile sizes and respective outputs and inputs of each layer are square.

#### Algorithm 3 Calculation for the Fusion Pyramid Tile Sizes

**Require:** Network, Number of Layers  $Q$

**Ensure:**  $H \leq IFM$

```

1: for (i in  $R_Q$ ) do
2:   for ( $j = Q, j \geq 1, j--$ ) do
3:      $H_{(i,j)} = (i - 1) \times S_j + K_j$ 
4:   end for
5: end for
6: Return  $\mathbf{H} \in \mathbb{R}^{R_Q \times Q}$ 
  
```

Algorithm 3 results in a relatively large design-space which can be narrowed down further by determining the appropriate stride for each tile in the fusion pyramid. The algorithm determines the number of movements  $\alpha$  that a particular tile should take under various tile stride  $S^T$  values. The  $S^T$  values are calculated using the condition that  $\alpha$  can only be an integer. Each value of  $S^T$  dictates the amount of overlap between the adjoining tiles in a layer in the fusion pyramid. In order to ensure the least amount of overlap, an  $S^T$  value of  $H - K + S$  can be selected. Although this selection ensures the least amount of overlap as well as the least number of  $\alpha$ , but it can result in a different number of movements at different levels of the pyramid. For instance,

in the previous example of LeNet-5, the tile size for  $CL1$  and  $CL2$  were selected to be  $16 \times 16$  and  $6 \times 6$  respectively. The tile stride for  $CL1$  and  $CL2$  will result in  $S_1^T = 16 - 5 + 1 = 12$  and  $S_2^T = 6 - 5 + 1 = 2$ , respectively. The value of  $S_2^T$  shows that the tile representing  $CL2$  results in  $\alpha_2 = 5$ , while the value of  $S_1^T$  results in a non-integer value of  $\alpha_1 = 7/3$  which has to be ruled-out. Also, the movement parameters for  $CL1$  and  $CL2$  do not agree, resulting in an asymmetric movement of different tiles in the fusion pyramid. This can lead to a number of issues; (1) requiring some synchronization delay between the execution of tiles caused by the stall cycles inserted between the execution of adjoining tiles, (2) increased latency due to one tile being executed several times more due to repeated computations compared to others in-turn decreasing the overall operating frequency of the design, and (3) the mismatch in synchronization may require for some intermediate data to be shuttled back to the memory in case of limited buffer space.

#### Algorithm 4 Calculation for the Tile Stride

**Require:**  $\mathbf{H} \in \mathbb{R}^{R_Q \times Q}$

```

1: for  $i = 1, i \leq R_Q, i++$  do
2:   for  $j = 1, j \leq Q, j++$  do
3:     for  $p = 1, p \leq H_j, p++$  do
4:        $\alpha_{(i,j,p)} = \frac{IFM_j - H_j}{p} + 1$ 
5:       if  $\alpha_{(i,j,p)} \in \mathbb{Z}$  then
6:          $\alpha_{ij} \leftarrow \alpha_{(i,j,p)}$ 
7:          $S_{ij}^T \leftarrow p$ 
8:       end if
9:     end for
10:   end for
11: end for
12: Return  $\alpha, \mathbf{S}^T \in \mathbb{R}^{R_Q \times Q}$ 

```

After calculating the  $S^T$  and  $\alpha$  parameters for the fusion tile size  $\mathbb{H} \in \mathbf{H}$  of choice, the values of  $S^T$  resulting in the same  $\alpha$  parameter values for each layer in the fusion pyramid can be evaluated and the corresponding  $S^T$  values for each layer can be obtained. The appropriate  $S^T$  values for each layer resulting in a synchronized fusion pyramid movement can simply be obtained by analyzing that the candidates for  $S^T$  do not result in skipping the computation of some regions in any layer. Among these  $S^T$  candidates, the maximum values for  $S^T$  for each layer is carefully selected after satisfying the condition stated earlier. Such  $S^T$  values ensure a uniform movement of each tile in the fusion pyramid, thereby addressing the three problems stated earlier.

#### 3.4. Accelerator designs

In order to show the efficacy of the proposed technique, we present two distinct approaches to the accelerator design. In one of the configurations, we aim to minimize the latency of the computations by exploiting the spatial parallelism in convolution operation at the cost of area. However, we show that conventional arithmetic-based design with the same configuration does not match the latency and the performance provided by the use of online arithmetic-based components. Additionally, an alternative, more pragmatic design is introduced, which performs convolution in a temporal manner and efficiently utilizes limited computational resources.

Both the aforementioned designs have similar general overall accelerator architecture. The overall architecture of the proposed accelerator is presented in Fig. 4. Depending on the number of convolution layers in a CNN model, there can be many pyramid levels in the proposed fused-layer design. Each pyramid level represents a tile in a particular convolution layer of the CNN model. The depth  $Q$  of the fusion determines the number of levels in the pyramid. The selection of the depth  $Q$  of the fusion pyramid can also help in optimizing the performance of the layer fusion acceleration designs. However, this work focuses

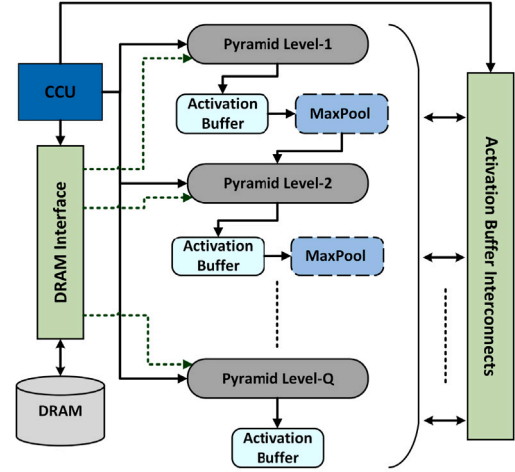


Fig. 4. Overall architecture. The solid black arrows represent the input, output, and control connections, while the dotted green arrows represent the filter/weight data. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

primarily on the calculation and selection of tile sizes and uniform stride, the use of online arithmetic-based compute units, early negative detection, etc. The optimization related to the selection of the number of layers ( $Q$ ) in the fusion pyramid is considered as future work. In the current work, the parameter  $Q$  is selected as referenced in literature [21,33]. Each pyramid level is followed by an activation buffer and an optional pooling block. The activation buffer block offers an on-chip buffer storage for the output features of the previous pyramid level. It is also noteworthy that for implementation on FPGAs, a large value of  $Q$  cannot be feasible due to the limited resources. However, we performed an experiment with 4 convolution layers of VGG-16 CNN and the experimental results show that with enough hardware resources, the proposed technique can be utilized for larger  $Q$  values. The proposed technique for the tile stride selection ensures uniform tile movement across the different pyramid levels. However, it leads to a slightly larger area of overlap regions within the fusion pyramid feature maps compared to the  $(H - K + S)$  region, which ensures minimum overlap. However, it is noteworthy that the proposed tile stride calculation technique not only ensures uniform movement across the pyramid levels but also keeps the number of pyramid movement  $\alpha$  to a minimum. This ensures that the overlap region does not increase drastically (ensured by the larger tile stride values). Furthermore, the overlapped output pixels of a pyramid level are stored in the output buffers to be reused by the subsequent level in the fusion pyramid as the fusion tile moves across the input feature map for its computation. This means that the proposed USEFUSE design performs output pixel reuse instead of recompute as suggested in [21].

##### 3.4.1. Design strategy-1 (DS-1) - spatial design

Each pyramid level in Fig. 4 represents an accelerator of the respective convolution layer in the fusion pyramid. A general architecture of the accelerator is presented in Fig. 5. It is composed of  $P = R \times C$  rows and  $M$  columns, where  $R \times C$  is the dimension of the output of a tile  $\mathbb{H}$ , and  $M$  is the number of output feature maps of the respective convolution layer. Each *input buffer* broadcasts the input data of a unique convolution window to its corresponding row of pixel processing units (PPUs). The *kernel buffers* broadcast the convolution filter to each PPU in a column. The *output buffers* collect the pixels from each PPU.

Each row of the array computes a unique  $K \times K \times N$  window of the input feature map of the corresponding layer within the fusion pyramid. The  $K \times K \times N$  input to each row, represented by the *input*

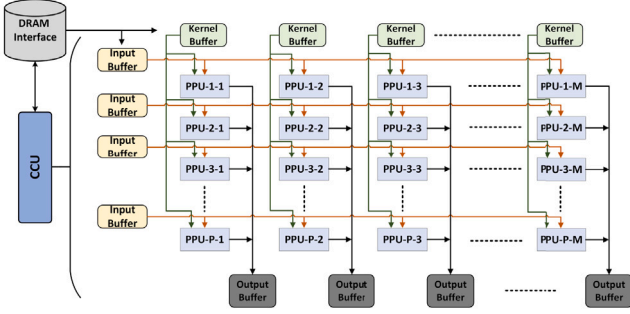


Fig. 5. Tile/Pyramid level design.

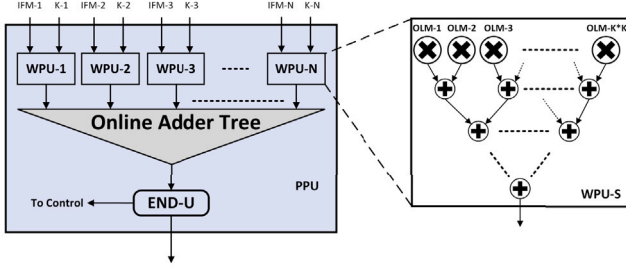


Fig. 6. Internal architecture of the proposed pixel processing unit with the window processing unit (WPU-S) that performs convolution in a spatial manner.

buffer, is broadcasted to each PPU in the corresponding row. The array architecture presented in Fig. 5 also shows that the accelerator array supports output tiling [55] ( $t_m = M$ ) as the number of columns in the array represent the number of output feature maps. It consists of an array of pixel processing units (PPU), where each column of the PPU computes a distinct output feature map (OFM). The filter corresponding to each OFM is broadcasted to every PPU in the respective column.

Each PPU supports input tiling [55] by the provision of ( $t_n = N$ ) window processing units (WPU-S), where each WPU-S is responsible to generate the inner product of  $K \times K$  pixels from one of the  $N$  input features maps. The output of each WPU-S is forwarded to an adder tree which results in one output pixel in one of the output feature maps. Each PPU also contains an early negative detection unit (END-U) responsible for the detection and generating control signals if the output of a PPU is going to result in a negative value. The architecture of the PPU is presented in Fig. 6.

### 3.4.2. Design strategy-2 (DS-2) - temporal design

An alternate design is presented that aims to perform convolutions in a temporal fashion. Consequently the amount of basic computation units required to compute a  $K \times K$  convolution window are reduced. In contrast to the WPU-S in the PPU design presented in Fig. 6, the window processing unit (WPU-T) in the present design allocates only one online arithmetic-based multiplier for the computation of a convolution window. This computation is carried out such that the online multiplier (OLM) is followed by an activation register that collects and stacks these output digits until all the output digits pertaining to one multiplication have been collected in the activation register. The contents of the activation register are then forwarded to an accumulation buffer until the results of the  $K \times K$  multiplications have been accumulated. The contents of this accumulation register are then forwarded, in an MSDF manner, to an online arithmetic-based adder tree responsible to generate the sum across the  $N$  input channels, ultimately resulting in the final output to be forwarded to the next operation in the CNN. It is also worth noting that the WPU-T pertaining to the temporal design can be replaced with WPU-S used in the PPU design presented in Fig. 6. The architecture of the WPU-T that leverages the temporal computation pattern is presented in Fig. 7.

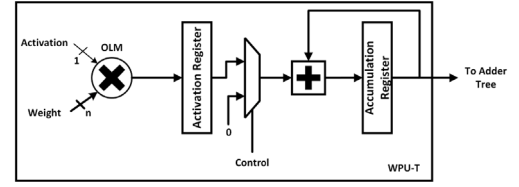
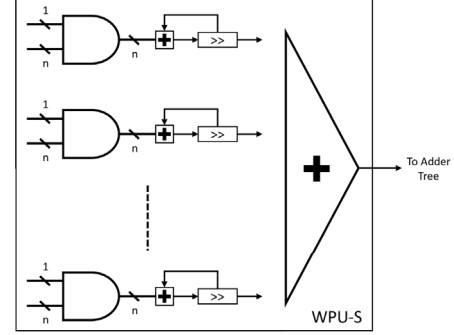


Fig. 7. Architecture of the proposed window processing unit (WPU-T) that leverages the temporal computation pattern in convolution.

Fig. 8. Architecture of the window processing unit (WPU-S), for conventional bit-serial design, that performs convolution of a  $K \times K$  convolution window spatially.

## 4. Experimental results and discussion

This section presents the experimental setup, performance evaluation parameters, results, comparisons, and discussion on the results obtained after the evaluation of the proposed designs.

### 4.1. Experimental setup

In order to evaluate and compare the performance of the proposed designs with conventional bit-serial architectures, three baseline designs are used; (1) Baseline-1: conventional bit-serial design based on the processing element from UNPU [14] with the *tile stride* matching the convolution stride, (2) Baseline-2: online arithmetic-based design, also using the *tile stride* as the convolution stride, and (3) Baseline-3: conventional bit-serial design where the *tile stride* matches the proposed designs. All baselines utilize the same accelerator architecture and array layout as the proposed designs. The architecture for both baseline conventional bit-serial designs follows a similar structure to the proposed design. However, in conventional bit-serial designs, the window processing units (WPUs) use AND gate arrays for partial product generation, followed by an accumulator to sum the partial products. The WPU-S design for spatial design (DS-1) is shown in Fig. 8.

Each of the baseline designs use the same accelerator architecture and array layout as the proposed designs. The architecture of the conventional bit-serial arithmetic-based baseline designs for both the design strategies also follow a similar accelerator architecture as the proposed design. However, the design of the window processing units (WPUs) for both the design strategies of conventional bit-serial designs contain AND gate arrays for partial product generation, followed by an accumulator to obtain the sum of the partial products. The WPU-S design, for spatial design (DS-1), is presented in Fig. 8. The accumulation process in the figure handles the summing of partial products, while the subsequent adder tree computes the sum of  $K \times K$  products. The resulting SoP from this adder tree is then passed to another adder tree, shown in the PPU in Fig. 6, which performs the final summation over  $N$  input channels.

In contrast to the spatial conventional bit-serial design presented in Fig. 8, a temporal design similar to that presented in Section 3.4.2 is

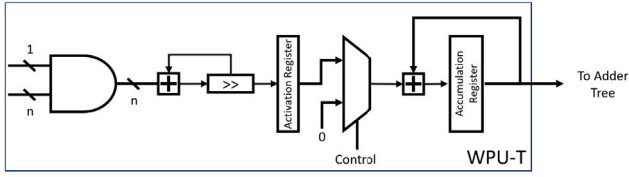


Fig. 9. Architecture of the window processing unit (WPU-T), for conventional bit-serial design, that performs convolution of a  $K \times K$  convolution window in a temporal fashion.

also devised. The WPU-T architecture for conventional bit-serial design follows a similar strategy as presented in Fig. 7, where the product of each of the  $K \times K$  multiplications is carried out using a single multiplier. The architecture of the conventional bit-serial WPU-T that leverages the temporal computation pattern in convolution is shown in Fig. 9.

In our experiments, we utilized three popular CNNs: LeNet-5 [56], AlexNet [57], and VGG-16 [58]. For LeNet-5 and AlexNet, the first two convolution layers, along with their corresponding non-linear activation and pooling layers, were selected for fusion. In VGG-16, the first two convolution blocks, comprising four convolution layers, including their respective activation and pooling layers, were used for the fused-layer experiments.

The RTL for the proposed and baseline accelerators was designed in Verilog and functionally verified using Xilinx Vivado 2023.2. We implemented the proposed designs on the Xilinx Virtex-7 VU19P FPGA. This FPGA platform was selected based on the availability of logic resources, as both the proposed and conventional bit-serial designs do not utilize built-in DSP resources for multiplication; instead, these resources are reserved for implementing the control units of the accelerators. This is due to the fundamental architectural differences between conventional DSP operations and MSDF arithmetic. MSDF multipliers rely on a residual recurrence method rather than traditional partial product reduction. This approach requires cycle-to-cycle state tracking, and bidirectional digit propagation which are not supported by current DSP architectures. Consequently, we have developed the MSDF-based arithmetic operators implemented using FPGA fabric resources.

#### 4.2. Performance evaluation parameters

The performance of the proposed method can be evaluated using various parameters such as performance, number of cycles, area, latency per image, inference speed-up, power efficiency, etc. The performance can be calculated using the following relation.

$$Performance = \frac{Num_{operations}}{number\ of\ execution\ cycles} \quad (2)$$

Where, the number of operations ( $Num_{operations}$ ) for a given convolution layer can be calculated as  $2 \times M \times N \times R \times C \times K \times K$ . Where  $M$  and  $N$  represent the number of output and input feature maps respectively,  $R$  and  $C$  represent the height and width of the output feature map, and  $K \times K$  is the dimension of the convolution kernel. Furthermore, the number of execution cycles (referred as  $Cycles$  from here-on) in Eq. (2) for the proposed online arithmetic-based design DS-1 can be calculated as;

$$\begin{aligned} Cycles = & \alpha^2 \times (\delta_{OLM} + \delta_{OLA} \times \lceil \log(K_1 \times K_1) \rceil \\ & + \delta_{OLA} \times \lceil \log N_1 \rceil + \lceil \log(K_1 \times K_1) \rceil + \lceil \log N_1 \rceil \\ & + MP_1 + \dots + \delta_{OLM} + \delta_{OLA} \times \lceil \log(K_Q \times K_Q) \rceil \\ & + \delta_{OLA} \times \lceil \log N_Q \rceil + \lceil \log(K_Q \times K_Q) \rceil \\ & + \lceil \log N_Q \rceil + MP_Q + n) \end{aligned} \quad (3)$$

where  $\delta_{OLM}$  and  $\delta_{OLA}$  represents the online delay for the multiplier and the adder respectively. These delays designate the number of cycles, usually up to 4, that an online arithmetic-based component takes prior to generating the first digit (MSD) as its output. The expression

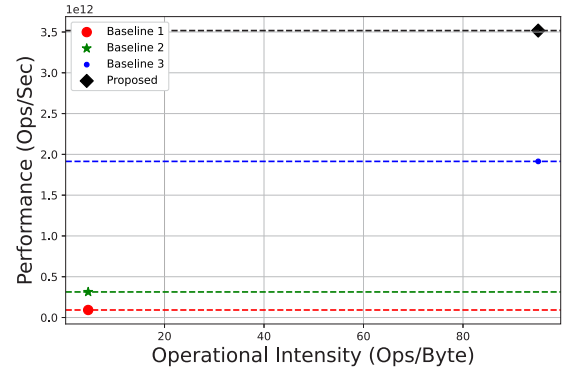


Fig. 10. Performance vs. operational intensity comparison of the proposed spatial design (DS-1) with the baseline designs for the first convolution layer of AlexNet.

$\lceil \log(K_Q \times K_Q) \rceil$  and  $\lceil \log N_Q \rceil$  define the number of stages of the adder trees dedicated for computing the SoP for convolution window and input channels respectively for a convolution layer.  $Q$  denotes the number of layers in the fusion design,  $MP$  denotes the number of cycles required to perform the maxpooling operation, and  $n$  denotes the precision of the input. Similarly, for the design DS-2, the number of cycles can be calculated as follows.

$$\begin{aligned} Cycles = & \alpha^2 \times ((\delta_{OLM} + (n - 1) + Acc) \times K \times K \\ & + \delta_{OLA} \times \lceil \log N_1 \rceil + \lceil \log N_1 \rceil + MP_1 \\ & + \dots + (\delta_{OLM} + (n - 1) + Acc) \times K \times K \\ & + \delta_{OLA} \times \lceil \log N_Q \rceil + \lceil \log N_Q \rceil + MP_Q + n) \end{aligned} \quad (4)$$

Here  $Acc$  denotes the number of cycles that the accumulator takes to perform the sum of 2 operands. Both the relations also include the number of cycles elapsed due to the growth in the output precision due to the adder trees, and it is denoted by  $\lceil \log(K \times K) \rceil$  and  $\lceil \log N \rceil$  in the equations.

Other performance evaluation parameters are platform-specific such as, logic utilization, memory utilization, throughput, inference time per image, etc. The selection of implementation platform relies on the capacity of the hardware resources in coordination with the resource requirements of the accelerator design.

#### 4.3. Experimental results

The proposed tile stride strategy coupled with the online arithmetic-based accelerator design not only improves the performance but can also improve the memory communication categorized by the operational intensity metric [59]. An analysis depicting the efficacy of the proposed technique is presented in Fig. 10. The figure shows that the proposed design and Baseline-3 design, using the proposed tile stride technique, have the same operational intensity as the other baseline designs. However, it is noteworthy that the performance of the proposed design surpasses that of Baseline-3 design. This demonstrates that the proposed tiling strategy, in combination with the superior capabilities of the online arithmetic paradigm, can outperform the conventional bit-serial design in terms of performance.

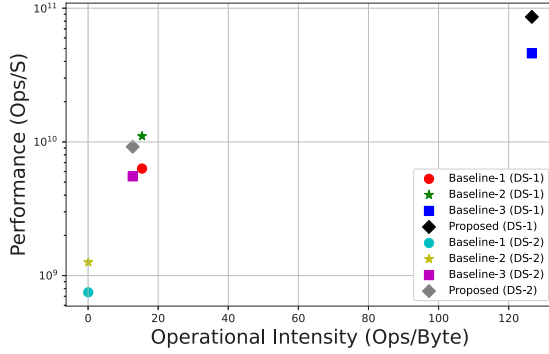
Similarly, a comparison of performance vs. operational intensity of the fused-layer designs for LeNet-5, AlexNet, and VGG CNN models has been presented in Fig. 11. The performance vs. operational intensity plots also confirm the findings presented in Fig. 10, that the proposed tile stride evaluation technique improves the operational intensity. For instance, the proposed spatial design (DS-1) improves the operational intensity for the LeNet-5, AlexNet, and VGG models by 8.20×, 17.80×, and 279.40×, respectively. Similarly, utilizing online modules for arithmetic-based computations can result in significant performance enhancements, as demonstrated in Figs. 11(a), 11(b), and 11(c).



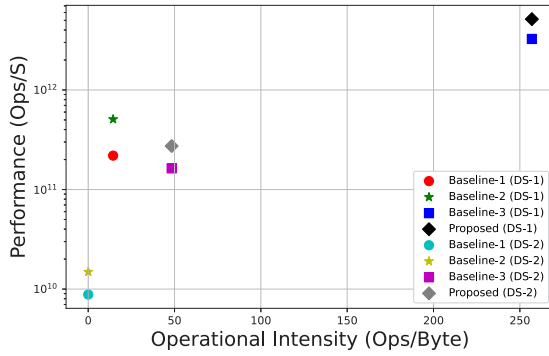
**Table 1**

Performance comparison of the proposed Spatial design (DS-1) with the baseline designs.

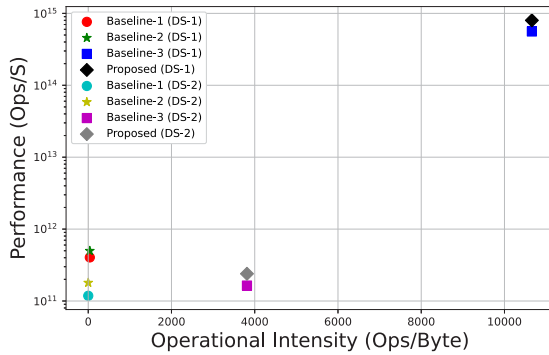
Network	Layer	Number of operations	Baseline-1		Baseline-2		Baseline-3		Proposed	
			Duration	Performance	Duration	Performance	Duration	Performance	Duration	Performance
LeNet	CONV1	235 200	138.72 $\mu$ S	1.69 GOPS	57.80 $\mu$ S	4.07 GOPS	12 $\mu$ S	19.60 GOPS	5 $\mu$ S	47.04 GOPS
	CONV2	940 800	41.31 $\mu$ S	22.77 GOPS	21.06 $\mu$ S	44.67 GOPS	12.75 $\mu$ S	73.8 GOPS	6.50 $\mu$ S	144.74 GOPS
	Fused	1 183 880	187.43 $\mu$ S	6.32 GOPS	107.19 $\mu$ S	11.04 GOPS	25.75 $\mu$ S	45.97 GOPS	13.75 $\mu$ S	86.10 GOPS
AlexNet	CONV1	105 415 200	1109 $\mu$ S	0.095 TOPS	623 $\mu$ S	0.169 TOPS	53.46 $\mu$ S	1.97 TOPS	29.97 $\mu$ S	3.517 TOPS
	CONV2	223 948 800	337.50 $\mu$ S	0.664 TOPS	268.75 $\mu$ S	0.833 TOPS	43.74 $\mu$ S	5.12 TOPS	34.83 $\mu$ S	6.43 TOPS
	Fused	329 659 136	1499.30 $\mu$ S	0.219 TOPS	648.7 $\mu$ S	0.508 TOPS	101.25 $\mu$ S	3.26 TOPS	63.99 $\mu$ S	5.15 TOPS
VGG	CONV1	173 408 256	8.11 ms	21.30 GOPS	5.41 ms	32.10 GOPS	3.78 $\mu$ S	45.87 TOPS	2.52 $\mu$ S	68.80 TOPS
	CONV2	3 699 376 128	9.14 ms	404.50 GOPS	7.95 ms	465.30 GOPS	4.14 $\mu$ S	893.6 TOPS	3.60 $\mu$ S	1027.60 TOPS
	CONV3	1 849 688 064	2.45 ms	754.56 GOPS	2.13 ms	867.75 GOPS	4.14 $\mu$ S	446.8 TOPS	3.60 $\mu$ S	513.80 TOPS
	CONV4	3 699 376 128	2.64 ms	1399.3 GOPS	2.42 ms	1529.50 GOPS	4.23 $\mu$ S	874.56 TOPS	3.87 $\mu$ S	955.90 TOPS
	Fused	9 429 625 856	23.36 ms	403.66 GOPS	18.92 ms	498.40 GOPS	16.83 $\mu$ S	560.30 TOPS	11.79 $\mu$ S	799.80 TOPS



(a) LeNet-5



(b) AlexNet



(c) VGG

**Fig. 11.** Performance vs. operational intensity comparison of the proposed spatial (DS-1) and temporal (DS-2) designs with the baseline designs for LeNet-5, AlexNet, and VGG models.

As outlined in the experimental setup, we evaluate the proposed designs on LeNet-5, AlexNet, and VGG-16 networks. The performance and evaluation duration using the proposed design (DS-1) compared to the baseline designs are presented in Table 1. All designs are evaluated at a frequency of 100 MHz, with inference time (referred to as duration) and performance listed in the table. Notably, online arithmetic-based designs consistently outperform conventional bit-serial designs, regardless of the tile stride strategy. Specifically, the fused layer design based on online arithmetic achieves performance improvements of 1.75 $\times$ , 2.32 $\times$ , and 1.23 $\times$  for LeNet-5, AlexNet, and VGG, respectively, without the proposed tile stride strategy. When using the proposed tile stride technique, the online arithmetic design outperforms Baseline-3 by achieving 1.87 $\times$ , 1.58 $\times$ , and 1.43 $\times$  superior performance for LeNet-5, AlexNet, and VGG, respectively.

For the temporal design DS-2, we present the comparative results of inference time and performance in-terms of GOPS for the conventional bit-serial design (Baseline-3) and the proposed design that use the proposed tile stride technique. Table 2 clearly shows that the proposed online arithmetic-based temporal design achieves 1.66 $\times$ , 1.68 $\times$ , and 1.46 $\times$  superior performance, in-terms of operations per second, for the fused layer designs of LeNet-5, AlexNet, and VGG respectively. The results presented in Tables 1 and 2 not only showcases the ability of online arithmetic-based designs over the conventional bit-serial arithmetic-based designs, but also confirm the utility of the proposed layer-fusion technique.

A comparison of the FPGA implementations of the proposed designs with the conventional bit-serial design (Baseline-3) is presented in Table 3 for the LeNet-5, AlexNet, and VGG models, all evaluated at a frequency of 100 MHz. The results indicate that the proposed method utilizes more logic resources and BRAM compared to the baseline designs. However, for larger networks like VGG, the BRAM requirement for the proposed design is significantly lower than that of the baseline design. This reduction is attributed to the arithmetic nature of the proposed design, where output digits in MSDF format can be directly forwarded to the next processing units, minimizing the need for large intermediate buffers. Additionally, the proposed design achieves speedups of 1.87 $\times$ , 1.58 $\times$ , and 1.43 $\times$  for the implementations of LeNet-5, AlexNet, and VGG, respectively. For instance, for the LeNet-5 design, the proposed fusion tile size and tile stride calculation resulted in a tile size of  $(16 \times 16)$  and  $(6 \times 6)$  for the first and second convolution layers, respectively. Particularly, the proposed and the baseline designs process  $(16 \times 16)$  and  $(6 \times 6)$  MACs in parallel for the first and second convolution layers respectively. The tile size and tile stride configuration resulted in the uniform movement parameter ( $\alpha = 5$ ). The obtained tile sizes and uniform tile strides resulted in the execution of one image in 1375 cycles, with 1.18M operations for the fused convolution layers, resulting in a throughput of 86.1 TOPS. Similarly, the 9429.6M operations for the first 4 convolution layers of the VGG-16 model, with the uniform tile movement parameter ( $\alpha = 3$ ), were executed in 84818 cycles, eventually resulting in a throughput of 799.8 TOPS. It is also worth noting that the results presented in Table 3 correspond to the

**Table 2**

Performance comparison of the proposed Temporal design (DS-2) with the conventional bit-serial design (Baseline-3) using the proposed tile stride technique.

Network	Layer	Number of Operations	Baseline-3		Proposed	
			Duration	Performance	Duration	Performance
LeNet	CONV1	235 200	0.11 mS	2.21 GOPS	62.50 $\mu$ S	3.80 GOPS
	CONV2	940 800	0.11 mS	8.80 GOPS	64 $\mu$ S	14.70 GOPS
	Fused	1 183 880	0.21 mS	5.53 GOPS	128.25 $\mu$ S	9.20 GOPS
AlexNet	CONV1	105 415 200	1.67 mS	63.2 GOPS	0.983 mS	107.20 GOPS
	CONV2	223 948 800	0.35 mS	641.50 GOPS	0.22 mS	1039.40 GOPS
	Fused	329 659 136	2.02 mS	163.20 GOPS	1.21 mS	273.50 GOPS
VGG	CONV1	173 408 256	13.95 $\mu$ S	1243.10 GOPS	8.64 $\mu$ S	2007.04 GOPS
	CONV2	3 699 376 128	14.31 $\mu$ S	258.50 TOPS	9.72 $\mu$ S	380.60 TOPS
	CONV3	1 849 688 064	14.31 $\mu$ S	128.30 TOPS	9.72 mS	190.30 TOPS
	CONV4	3 699 376 128	15.03 $\mu$ S	246.10 TOPS	9.90 mS	370.30 TOPS
	Fused	9 429 625 856	57.50 $\mu$ S	163.90 TOPS	39.40 $\mu$ S	239.20 TOPS

**Table 3**

Comparison of FPGA implementation of proposed spatial design (DS-1) with the conventional bit-serial design with the proposed tiling scheme (Baseline-3). The FPGA device used for this experiment is Xilinx Ultrascale+ Vertex-7 VU19P.

Design	Baseline-3	Proposed	Baseline-3	Proposed	Baseline-3	Proposed
CNN Model	LeNet-5		AlexNet		VGG	
Logic Utilization	18.40K (0.21%)	28.80K (0.322%)	5619.30K (63%)	8645K (96.70%)	7091K (79.30%)	7555.50K (94.5%)
BRAM Utilization	2 (0.05%)	3 (0.06%)	62 (2.90%)	113 (5.20%)	740 (34.30%)	211 (9.80%)
Throughput (TOPS)	45.97 GOPS	86.10 GOPS	3.26	5.15	560.30	799.80
Latency/Image ( $\mu$ S)	25.75	13.75	101.25	63.99	16.83	11.79
Speedup	1	1.87 $\times$	1	1.58 $\times$	1	1.43 $\times$

**Table 4**

Comparison of FPGA implementation of proposed temporal design (DS-2) with the conventional bit-serial design with the proposed tiling scheme (Baseline-3). The FPGA device used for this experiment is Xilinx Ultrascale+ Vertex-7 VU19P.

Design	Baseline-3	Proposed	Baseline-3	Proposed	Baseline-3	Proposed
CNN Model	LeNet-5		AlexNet		VGG	
Logic Utilization	4.50K (0.05%)	14.20K (0.16%)	277K (3.10%)	874.20K (9.80%)	1270K (14.20%)	4012.20K (44.90%)
BRAM Utilization	2 (0.05%)	2 (0.05%)	44 (2.04%)	75 (3.5%)	701 (32.5%)	134 (6.21%)
Throughput (GOPS)	5.53	9.20	163.20	273.50	164 TOPS	239 TOPS
Latency/Image ( $\mu$ S)	214.25	128.25	2020.14	1205.30	57.51	39.42
Speedup	1	1.67 $\times$	1	1.68 $\times$	1	1.46 $\times$

exploitation of the maximum potential of the proposed uniform tiling method and the online arithmetic-based computation units.

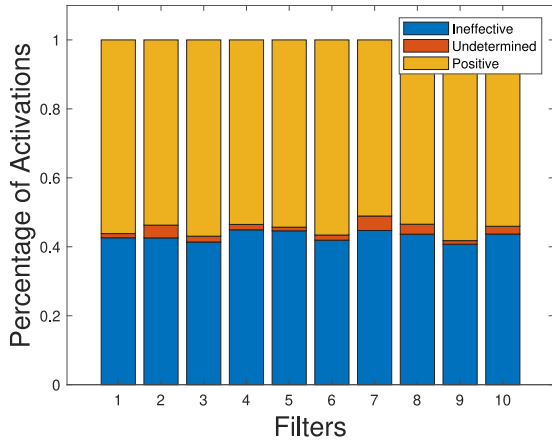
Similarly, the comparison of the proposed temporal design (DS-2) with the conventional bit-serial baseline design (Baseline-3) is presented in Table 4. A similar trend in the BRAM utilization can be observed where for the VGG model fusion design, the proposed method requires nearly 5.2 $\times$  less BRAMs compared to the baseline design. This is due to the inherent property of the proposed online arithmetic-based design where the intermediate output digits can be used directly for the computation of the subsequent layer or operations. The results also show that the proposed temporal design achieves speedup of 1.67 $\times$ , 1.68 $\times$ , and 1.46 $\times$  for the implementation of LeNet-5, AlexNet, and VGG respectively, compared to the conventional bit-serial baseline design.

We also present the effect of early negative activation detection caused by ReLU activation function. For this experiment, we present the results of the proposed early negative detection technique on 10 randomly selected filters for the first convolution layers of AlexNet and VGG models in Figs. 12(a) and 12(b) respectively. The analysis of the early negative detection technique show that an average of 43.1% and 41.08% activations per convolution filter were effectively determined as negative activations for the first convolution layers of AlexNet and VGG respectively. Nearly 2.36% and 2.11% activations were undetermined as either negative or positive. Upon examining the intermediate feature maps, it is determined that most of these undetermined activations were zero and hence did not cause any accuracy loss in the model classification performance.

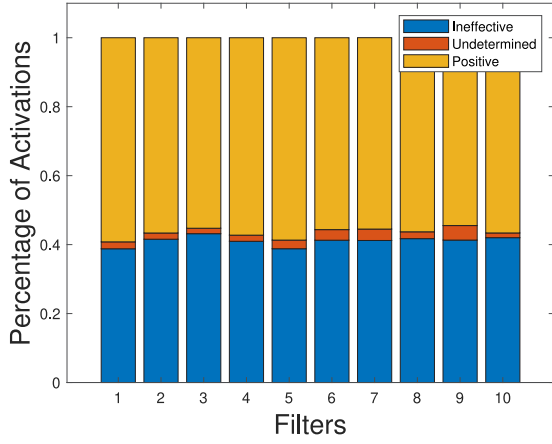
Substantial energy savings can be achieved by detecting ineffective activations. In this context, results of the energy savings for the three

networks used in this study are presented in Fig. 13. The figure illustrates the energy consumption corresponding to 10 randomly selected output feature maps of the first convolution layers. We performed our experiments with the proposed early negative detection (END) technique as well as without the proposed END technique using 10 000 images for all three networks. The proposed END technique resulted in substantial energy savings of 46.80%, 48.50%, and 42.60% for LeNet-5, AlexNet, and VGG networks respectively.

Another experiment was conducted to demonstrate the effectiveness of the proposed END technique in reducing computation cycles within a fusion pyramid, using the ResNet-18 network. For this experiment, we fused two consecutive convolution layers, excluding the first convolution layer to ensure that each convolution block contains two fusion pyramids. We tested this setup on 100 images and report the average number of effective computation cycles with and without the proposed END scheme, for both the online arithmetic-based design and the conventional bit-serial (Baseline-3) design. The impact of the END technique on effective computation cycles is illustrated in Fig. 14. It can be observed from the figure that the proposed END technique saves up to 50.1% cycles for the end-to-end execution of ResNet-18 workload using the proposed online arithmetic-based design. The comparison also shows the effectiveness of the online arithmetic-based computation where the online arithmetic designs with and without the END technique achieve 59.12% and 18.4% lower number of computation cycles compared to the conventional bit-serial design that uses the same accelerator architecture and the proposed tile stride technique.



(a) AlexNet



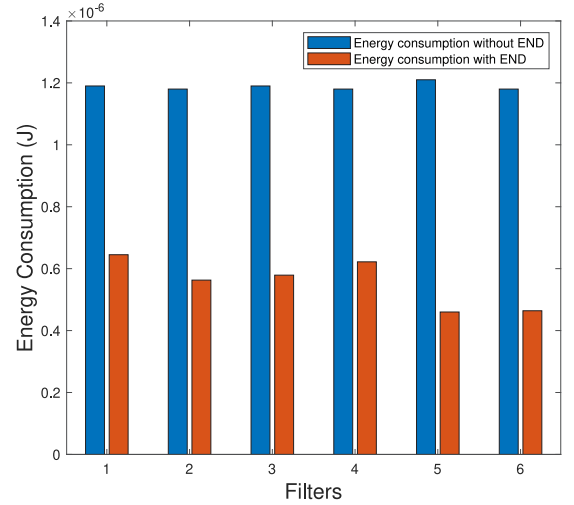
(b) VGG

Fig. 12. Percentage of detected negative/ineffective activations for 10 randomly selected filters in AlexNet and VGG models. The mean number of negative activations per output feature map is 43.10% and 41.08% for the first convolution layers of AlexNet and VGG respectively.

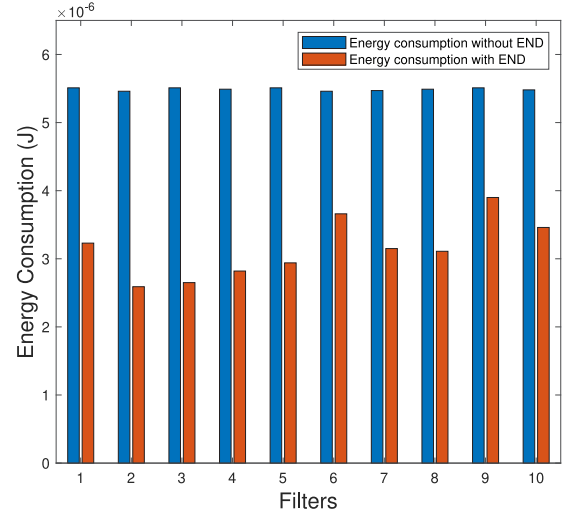
#### 4.4. Comparison with previous works

For the comparison with existing accelerators, we aim to accelerate the convolution layers in VGG-16 and ResNet-18 workloads in an end-to-end fashion. We conduct the performance comparison on several performance metrics such as latency per image, throughput in terms of GOPS, etc. For this experiment, we developed a fusion strategy by fusing 2 convolution layers in the fusion pyramid. The tiling parameters such as tile size, tile stride, etc., are calculated using Algorithms 3 and 4. The ImageNet weights from the pre-trained VGG-16 and ResNet-18 models from TorchVision [60] library were loaded and used in the computation of the convolution layers. Furthermore, the accuracy is computed offline using the feature maps of the convolution layer before the classifier layers of the model.

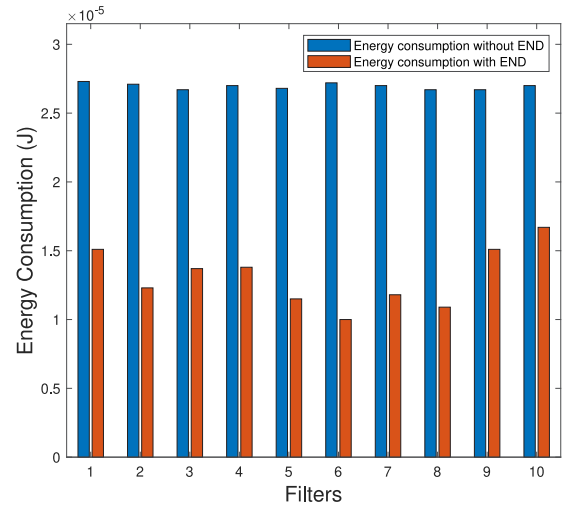
The comparison of the proposed USEFUSE design with previous designs is presented in Table 5. The results presented in the table indicate that the proposed design uses comparatively large number of logic resources than its contemporary counterparts. However, the proposed USEFUSE design achieves 64.8%, 50.9%, and 38.9% less BRAMs compared to TPGA [33,61], and ShortcutFusion [62] for VGG-16 workloads, respectively. For ResNet-18 workloads, USEFUSE uses 34.5% less BRAM resources compared to the design presented in [25]. The proposed design achieves significant throughput improvements of 3.7 $\times$ , 3.48 $\times$ , 9.2 $\times$ , and 1.9 $\times$  for VGG-16 workloads compared to



(a) LeNet-5



(b) AlexNet



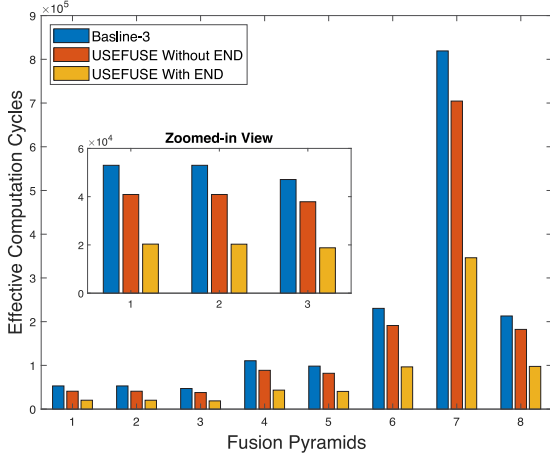
(c) VGG

Fig. 13. Energy savings with the proposed early negative detection (END) technique for the first convolution layers of LeNet-5, AlexNet, and VGG models. On average, 46.80%, 48.50%, and 42.60% reduction in energy consumption is observed for LeNet-5, AlexNet, and VGG respectively.

**Table 5**

Comparison with existing CNN accelerators. The baseline top-1 accuracy for VGG-16 and ResNet-18 are reported as 71.6% and 69.76% on their respective Pytorch [60] websites.

Model Design	VGG-16					ResNet-18				
	TGPA [33]	[61]	Shortcut-Fusion [62]	[63]	Proposed	[25]	T-DLA [26]	[64]	RLDA [65]	Proposed
FPGA	VU9P	Stratix 10 GX2800	KCU1500	Alveo U50	VU5P	Stratix V	Zynq-7000	Arria10 SX660	Ultrascale+ XCZU7EV	VU5P
Frequency (MHz)	210	300	200	200	100	124	125	170	150	100
Input/Filter Precision	16	16/8	16	8	8/8	8/8	8/2	8	8	8/8
Accuracy (%)	—	—	—	72.32	71.21	69.75	65.6	—	65.5	69.13
Logic Utilization	493K (42%)	469K (50%)	215.3K (33%)	601.7K (69%)	538.1K (89.5%)	380.35K	37.92K (71.28%)	102.6K (41%)	230.4K (88.2%)	542.6K (90.2%)
BRAM Utilization	3380	2421	1945 (45%)	1084 (81%)	1188 (58%)	1644	68.93%	—	307 (98.4%)	1076 (52.54%)
Throughput (GOPS)	1510	1604.57	607.5	2895.5	5594.7	926.84	400	89.286	620	1130.7
Latency per Image (ms)	22.35	19.29	39.27	13.90	9.18	—	—	—	—	14.44



**Fig. 14.** The average effective computation cycles for each fusion pyramid were compared between the Baseline-3 design and the proposed design, with and without the END technique. The results showed that the END technique achieved an average savings of 50.1% in computation cycles for the end-to-end flow.

TGPA [33,61], ShortcutFusion [62], and [63] respectively. Similarly, USEFUSE achieved throughput improvements of 1.2×, 2.82×, 12.6×, and 1.82× compared to the designs presented in [25], T-DLA [26,64], and RLDA [65] respectively, for ResNet-18 workloads. Furthermore, the proposed design achieved 2.43×, 2.1×, 4.27×, and 1.5× improvement in latency per image, compared to TGPA [33,61], ShortcutFusion [62], and [63] for VGG-16 workloads, respectively.

The experimental results indicate that the use of online arithmetic-based compute units in the processing element can not only perform efficient computation of the convolution SOP, but also support the fusion of convolution layers in a CNN. Moreover, the MSDF nature of online arithmetic also aids in the early detection and subsequent termination of the ineffective computations that result in negative outputs. The proposed method of tile size and uniform stride calculation, coupled with online arithmetic-based compute units showcase superior performance compared to the state-of-the-art accelerator designs on VGG-16 and ResNet-18 workloads.

## 5. Limitations and future work

While the proposed method offers significant advantages in terms of computational efficiency, it has certain limitations that we aim to address in future research. Firstly, the proposed early negative detection technique limits the applicability to models relying on ReLU. While ReLU is fundamental and widely adopted, modern architectures also employ complex activation functions such as GELU, Sigmoid, Softmax, etc. Additionally, the proposed uniform stride method is specifically tested on ResNet-18, where skip connections are limited within individual residual blocks and do not span across multiple convolution blocks. This restriction allows for a simpler implementation of layer fusion, as

the input from the skip connection can be integrated directly into the pipeline without requiring extensive reconfiguration. However, skip-connections spanning several convolution blocks may pose a challenge in determining the effective stride and tile size calculation which will be addressed in future research.

To overcome these limitations, our future work will focus on extending the method to support complex activation functions by developing efficient hardware implementations based on online arithmetic operators. This includes operations such as division, exponentiation, and power functions, which are commonly used in activation functions like Sigmoid and Softmax. This development will address both the challenges of accelerating modern architectures with complex activation functions and the need for efficient implementation of these same activation functions in the final output layers of neural network architectures to distinguish target applications. Therefore, the development of these activation functions will enhance the practicality of the proposed method by solving both issues simultaneously.

Additionally, for architectures with longer skip connections, we propose integrating an adder within the pipeline to sum convolution outputs with skip connection inputs, requiring minimal structural changes and maintaining performance. Furthermore, a dynamic data flow control mechanism using multiplexers will be explored, allowing seamless switching between outputs from activation registers, skip connection registers, or zero values. These enhancements will allow the proposed accelerator to efficiently support a wider range of neural network models.

**Extension to modern architectures.** For transformer-based models, the computational dataflow significantly differs from CNNs, as multiple tokens are processed in parallel, and several attention heads operate simultaneously. While our current approach is primarily optimized for CNNs, its underlying principles can be extended to optimize transformer workloads. Specifically, the attention mechanism, which involves a sequence of dependent operations, could benefit from our MSDF mode of operation by enabling efficient pipelining. By restructuring the computation flow to exploit temporal parallelism, our approach could contribute to the acceleration of self-attention mechanisms. As part of our future research, we aim to explore tailored acceleration strategies for both depthwise convolutions in MobileNet and self-attention mechanisms in transformers, thereby extending the applicability of our method beyond CNNs.

**Hardware optimizations for low-resource deployment.** To enhance the feasibility of our design for deployment on low-resource edge devices, several hardware optimizations can be explored. Our proposed temporal design (DS-2) reduces logic utilization by reusing computational resources over multiple cycles, and additional efficiency gains can be achieved through composite MSDF arithmetic operators. By designing a single SOP unit, we can minimize both logic area and latency, effectively decreasing the online delay while maintaining performance.

Furthermore, incorporating quantization and sparsity-aware optimizations can significantly reduce on-chip memory requirements. In our design, storage is already structured in multiples of 8-bit precision,



making it inherently compatible with quantization techniques. Reducing precision lowers both memory footprint and compute latency without substantial accuracy degradation. Additionally, sparsity-aware optimizations can further decrease BRAM utilization by eliminating redundant computations and avoiding unnecessary storage of zero-valued parameters. Adaptive tiling strategies can be employed to maximize data reuse, thereby minimizing on-chip memory overhead for edge-device deployments. Moreover, resource-sharing mechanisms can optimize memory access patterns, ensuring efficient use of available storage. These optimizations, combined with our proposed design principles, pave the way for high-performance yet resource-efficient deep learning accelerators, particularly suited for edge computing applications.

By addressing these challenges, we aim to enhance the practicality and versatility of the proposed uniform stride and tiling method, enabling the accelerator to cater to a wide range of applications such as classification, detection, and segmentation.

## 6. Conclusion

This study introduces the use of low-latency left-to-right bit-serial arithmetic-based SOP units for convolution in fused CNN accelerators. Two designs cater to varied demands, emphasizing minimal response time (DS-1) for mission-critical applications and resource-constrained devices (DS-2). DS-1, a spatial computation pattern-based design, enhances operational intensity by 8.20 $\times$ , 17.80 $\times$ , and 279.40 $\times$  for LeNet-5, AlexNet, and VGG networks, respectively. The temporal computation pattern-based design achieves speedups of 1.67 $\times$ , 1.68 $\times$ , and 1.46 $\times$  for LeNet-5, AlexNet, and VGG networks respectively, surpassing conventional bit-serial baselines. An effective mechanism skips inefficient convolutions after ReLU layers, reducing power consumption without accuracy loss which demonstrates substantial energy savings of 46.80%, 48.50%, and 42.60% for LeNet-5, AlexNet, and VGG networks, respectively. Furthermore, the proposed USEFUSE has also exhibited superior performance compared to the existing CNN accelerator designs. These results underscore the efficacy of the proposed Uniform Stride strategy for an improved operational intensity and optimizing energy consumption and computational speed in neural network implementations.

## CRedit authorship contribution statement

**Muhammad Sohail Ibrahim:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Conceptualization. **Muhammad Usman:** Writing – review & editing, Software, Methodology, Conceptualization. **Jeong-A Lee:** Supervision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## References

- [1] Y. Sun, B. Xue, M. Zhang, G.G. Yen, J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification, *IEEE Trans. Cybern.* 50 (9) (2020) 3840–3854.
- [2] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.
- [3] Y.H. Yoon, S. Khan, J. Huh, J.C. Ye, Efficient B-mode ultrasound image reconstruction from sub-sampled RF data using deep learning, *IEEE Trans. Med. Imaging* 38 (2) (2018) 325–336.
- [4] M. Usman, S. Khan, S. Park, A. Wahab, AFP-SRC: identification of antifreeze proteins using sparse representation classifier, *Neural Comput. Appl.* (2022) 1–11.
- [5] Y.-W. Chen, K.-H. Hung, Y.-J. Li, A.C.-F. Kang, Y.-H. Lai, K.-C. Liu, S.-W. Fu, S.-S. Wang, Y. Tsao, CITISEN: A deep learning-based speech signal-processing mobile application, *IEEE Access* 10 (2022) 46082–46099.
- [6] X. Chen, M. Li, H. Zhong, Y. Ma, C.-H. Hsu, DNNOff: offloading DNN-based intelligent IoT applications in mobile edge computing, *IEEE Trans. Ind. Inform.* 18 (4) (2021) 2820–2829.
- [7] C.-C. Lin, C.-Y. Liu, C.-H. Yen, T.-W. Kuo, P.-C. Hsiu, Intermittent-aware neural network pruning, in: *2023 60th ACM/IEEE Design Automation Conference, DAC, IEEE*, 2023, pp. 1–6.
- [8] S. Oh, H. Sim, J. Kim, J. Lee, Non-uniform step size quantization for accurate post-training quantization, in: *European Conference on Computer Vision, Springer*, 2022, pp. 658–673.
- [9] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, J. Henkel, Adapt: Fast emulation of approximate dnn accelerators in pytorch, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* (2022).
- [10] H.-J. Yoo, S. Park, K. Bong, D. Shin, J. Lee, S. Choi, A 1.93 tops/w scalable deep learning/inference processor with tetra-parallel mind architecture for big data applications, in: *IEEE International Solid-State Circuits Conference, IEEE*, 2015, pp. 80–81.
- [11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, O. Temam, ShiDianNao: Shifting vision processing closer to the sensor, in: *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.
- [12] Y.-H. Chen, T. Krishna, J.S. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, *IEEE J. Solid-State Circuits* 52 (1) (2016) 127–138.
- [13] P. Judd, J. Albericio, T. Hetherington, T.M. Aamodt, A. Moshovos, Stripes: Bit-serial deep neural network computing, in: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE*, 2016, pp. 1–12.
- [14] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, H.-J. Yoo, UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision, *IEEE J. Solid-State Circuits* 54 (1) (2018) 173–185.
- [15] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, H. Esmaeilzadeh, Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network, in: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE*, 2018, pp. 764–775.
- [16] N. Kim, H. Park, D. Lee, S. Kang, J. Lee, K. Choi, ComPreEND: Computation pruning through predictive early negative detection for ReLU in a deep neural network accelerator, *IEEE Trans. Comput.* (2021).
- [17] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R.K. Gupta, H. Esmaeilzadeh, Snapea: Predictive early activation for reducing computation in deep convolutional neural networks, in: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE*, 2018, pp. 662–673.
- [18] D. Lee, S. Kang, K. Choi, ComPEND: Computation pruning through early negative detection for ReLU in a deep neural network accelerator, in: *Proceedings of the 2018 International Conference on Supercomputing*, 2018, pp. 139–148.
- [19] X. Chen, J. Zhu, J. Jiang, C.-Y. Tsui, CompRRAE: RRAM-based convolutional neural network accelerator with reduced computations through an runtime activation estimation, in: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 133–139.
- [20] M.D. Ercegovac, T. Lang, *Digital Arithmetic*, Elsevier, 2004.
- [21] M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE*, 2016, pp. 1–12.
- [22] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N.E. Jerger, R. Urtasun, A. Moshovos, Proteus: Exploiting precision variability in deep neural networks, *Parallel Comput.* 73 (2018) 40–51.
- [23] S. Shin, Y. Boo, W. Sung, Fixed-point optimization of deep neural networks with adaptive step size retraining, in: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE*, 2017, pp. 1203–1207.
- [24] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, M. Zhang, Distilling bit-level sparsity parallelism for general purpose deep learning acceleration, in: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 963–976.
- [25] C. Latotzke, T. Ciesielski, T. Gemmeke, Design of high-throughput mixed-precision CNN accelerators on FPGA, in: *2022 32nd International Conference on Field-Programmable Logic and Applications, FPL, IEEE*, 2022, pp. 358–365.
- [26] Y. Chen, K. Zhang, C. Gong, C. Hao, X. Zhang, T. Li, D. Chen, T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA, in: *2019 IEEE Computer Society Annual Symposium on VLSI, ISVLSI, IEEE*, 2019, pp. 13–18.
- [27] M.B. Karadeniz, M. Altun, TALIPOT: Energy-efficient DNN booster employing hybrid bit parallel-serial processing in MSB-first fashion, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 41 (8) (2021) 2714–2727.
- [28] W. Liu, J. Lin, Z. Wang, A precision-scalable energy-efficient convolutional neural network accelerator, *IEEE Trans. Circuits Syst. I. Regul. Pap.* 67 (10) (2020) 3484–3497.

- [29] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, A. Moshovos, Bit-pragmatic deep neural network computing, in: 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2017, pp. 382–394.
- [30] K. Al-Hawaj, O. Afuye, S. Agwa, A. Apse, C. Batten, Towards a reconfigurable bit-serial/bit-parallel vector accelerator using in-situ processing-in-sram, in: 2020 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2020, pp. 1–5.
- [31] L. Waeijen, S. Sioutas, M. Peemen, M. Lindwer, H. Corporaal, ConvFusion: A model for layer fusion in convolutional neural networks, IEEE Access 9 (2021) 168245–168267.
- [32] Z. Zhao, K.M. Barijough, A. Gerstlauer, Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (11) (2018) 2348–2359.
- [33] X. Wei, Y. Liang, X. Li, C.H. Yu, P. Zhang, J. Cong, TGPA: Tile-grained pipeline architecture for low latency CNN inference, in: 2018 IEEE/ACM International Conference on Computer-Aided Design, ICCAD, ACM, 2018, pp. 1–8.
- [34] Q. Xiao, Y. Liang, L. Lu, S. Yan, Y.-W. Tai, Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs, in: Proceedings of the 54th Annual Design Automation Conference 2017, 2017, pp. 1–6.
- [35] M. Li, N. Wang, H. Zhou, Y. Duan, J. Wu, Fused-layer-based DNN model parallelism and partial computation offloading, in: GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022, pp. 5195–5200.
- [36] H. Zhou, M. Li, N. Wang, G. Min, J. Wu, Accelerating deep learning inference via model parallelism and partial computation offloading, IEEE Trans. Parallel Distrib. Syst. 34 (2) (2022) 475–488.
- [37] X. Cai, Y. Wang, K. Tu, C. Gao, L. Zhang, Olympus: Reaching memory-optimality on DNN processors, IEEE Trans. Comput. 71 (8) (2021) 1939–1951.
- [38] S. Tewari, A. Kumar, K. Paul, Minimizing off-chip memory access for CNN accelerators, IEEE Consum. Electron. Mag. 11 (3) (2021) 95–104.
- [39] H. Ahmad, T. Arif, M.A. Hanif, R. Hafiz, M. Shafique, SuperSlash: A unified design space exploration and model compression methodology for design of deep learning accelerators with reduced off-chip memory access volume, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 4191–4204.
- [40] S. Tewari, A. Kumar, K. Paul, Bus width aware off-chip memory access minimization for CNN accelerators, in: 2020 IEEE Computer Society Annual Symposium on VLSI, ISVLSI, IEEE, 2020, pp. 240–245.
- [41] D. Kang, D. Kang, S. Ha, Multi-bank on-chip memory management techniques for cnn accelerators, IEEE Trans. Comput. 71 (5) (2021) 1181–1193.
- [42] K. Zhang, Y. Li, J. Liang, J. Cao, Y. Zhang, H. Tang, D.-P. Fan, R. Timofte, L.V. Gool, Practical blind image denoising via swin-conv-unet and data synthesis, Mach. Intell. Res. 20 (6) (2023) 822–836.
- [43] M. Li, W. Liu, W. Chen, An image denoising method based on swin transformer V2 and U-net architecture, in: 2024 IEEE 16th International Conference on Advanced Infocomm Technology, ICAIT, IEEE, 2024, pp. 204–209.
- [44] K. Zhang, Y. Li, W. Zuo, L. Zhang, L. Van Gool, R. Timofte, Plug-and-play image restoration with deep denoiser prior, IEEE Trans. Pattern Anal. Mach. Intell. 44 (10) (2021) 6360–6376.
- [45] D.-Y. Chen, H. Tennent, C.-W. Hsu, ArtAdapter: Text-to-image style transfer using multi-level style encoder and explicit adaptation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 8619–8628.
- [46] A. Bhattad, J. Soole, D. Forsyth, StyLitGAN: Image-based relighting via latent control, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 4231–4240.
- [47] S. Tafasca, A. Gupta, J.-M. Odobez, Shangan: A transformer architecture for multi-person gaze following, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 2008–2017.
- [48] K. Shen, J. Guo, X. Tan, S. Tang, R. Wang, J. Bian, A study on relu and softmax in transformer, 2023, arXiv preprint arXiv:2302.06461.
- [49] M. Wortsman, J. Lee, J. Gilmer, S. Kornblith, Replacing softmax with relu in vision transformers, 2023, arXiv preprint arXiv:2309.08586.
- [50] U. Mallappa, P. Gangwar, B. Khaleghi, H. Yang, T. Rosing, TerminiNETor: Early convolution termination for efficient deep neural networks, in: 2022 IEEE 40th International Conference on Computer Design, ICCD, IEEE, 2022, pp. 635–643.
- [51] Y. Pan, J. Yu, A. Lukefahr, R. Das, S. Mahlke, BitSET: Bit-serial early termination for computation reduction in convolutional neural networks, ACM Trans. Embed. Comput. Syst. 22 (5s) (2023) 1–24.
- [52] M. Asadikouhanjani, S.-B. Ko, A novel architecture for early detection of negative output features in deep neural network accelerators, IEEE Trans. Circuits Syst. II: Express Briefs 67 (12) (2020) 3332–3336.
- [53] M.K. Shuvo, D.E. Thompson, H. Wang, MSB-first distributed arithmetic circuit for convolution neural network computation, in: 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems, MWCAS, IEEE, 2020, pp. 399–402.
- [54] M. Usman, M. D. Ercegovac, J.-A. Lee, Low-latency online multiplier with reduced activities and minimized interconnect for inner product arrays, J. Signal Process. Syst. (2023) 1–20.
- [55] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, in: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015, pp. 161–170.
- [56] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
- [57] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012).
- [58] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
- [59] G. Ofenbeck, R. Steinmann, V. Caparros, D.G. Spampinato, M. Püschel, Applying the roofline model, in: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE, 2014, pp. 76–85.
- [60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Adv. Neural Inf. Process. Syst. 32 (2019).
- [61] Y. Ma, Y. Cao, S. Vruthula, J.-s. Seo, Automatic compilation of diverse CNNs onto high-performance FPGA accelerators, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (2) (2018) 424–437.
- [62] D.T. Nguyen, H. Je, T.N. Nguyen, S. Ryu, K. Lee, H.-J. Lee, ShortcutFusion: From tensorflow to FPGA-based accelerator with a reuse-aware memory allocation for shortcut data, IEEE Trans. Circuits Syst. I: Regul. Pap. 69 (6) (2022) 2477–2489.
- [63] S. Hong, Y.F. Arthanto, J.-Y. Kim, et al., Accelerating deep convolutional neural networks using number theoretic transform, IEEE Trans. Circuits Syst. I: Regul. Pap. 70 (1) (2022) 315–326.
- [64] X. Xie, J. Lin, Z. Wang, J. Wei, An efficient and flexible accelerator design for sparse convolutional neural networks, IEEE Trans. Circuits Syst. I: Regul. Pap. 68 (7) (2021) 2936–2949.
- [65] H. Fuketa, T. Katashita, Y. Hori, M. Hioki, Multiplication-free lookup-based CNN accelerator using residual vector quantization and its FPGA implementation, IEEE Access (2024).