An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs

An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs

FAREED QARARYAH, MUHAMMAD WAQAR AZHAR, and PEDRO TRANCOSO, Chalmers University of Technology, Sweden

Resource-efficient Convolutional Neural Networks (CNNs) are gaining more attention. These CNNs have relatively low computational and memory requirements. A common denominator among such CNNs is having more heterogeneity than traditional CNNs. This heterogeneity is present at two levels: intra-layer type and inter-layer type. Generic accelerators do not capture these levels of heterogeneity, which harms their efficiency. Consequently, researchers have proposed model-specific accelerators with dedicated engines. When designing an accelerator with dedicated engines, one option is to dedicate one engine per CNN layer. We refer to accelerators designed with this approach as single-engine single-layer (SESL). This approach enables optimizing each engine for its specific layer. However, such accelerators are resource-demanding and unscalable. Another option is to design a minimal number of dedicated engines such that each engine handles all layers of one type. We refer to these accelerators as single-engine multiple-layer (SEML). SEML accelerators capture the inter-layer-type but not the intra-layer-type heterogeneity.

We propose the Fixed Budget Hybrid CNN Accelerator (FiBHA), a hybrid accelerator composed of an SESL part and an SEML part, each processing a subset of CNN layers. FiBHA captures more heterogeneity than SEML while being more resource-aware and scalable than SESL. Moreover, we propose a novel module, Fused Inverted Residual Bottleneck (FIRB), a fine-grained and memory-light SESL architecture building block. The proposed architecture is implemented and evaluated using high-level synthesis (HLS) on different Field Programmable Gate Arrays representing various resource budgets. Our evaluation shows that FiBHA improves the throughput by up to $4x$ and $2.5x$ compared to state-of-the-art SESL and SEML accelerators, respectively. Moreover, FiBHA reduces memory and energy consumption compared to an SEML accelerator. The evaluation also shows that FIRB reduces the required memory by up to 54%, and energy requirements by up to 35% compared to traditional pipelining.

CCS Concepts: • **Hardware → Emerging architectures**; • **Computing methodologies → Machine learning**; • **Computer systems organization → Parallel architectures;**

Extension of Conference Paper: This work is an extension of “FiBHA: Fixed Budget Hybrid CNN Accelerator” [39]. In this extension, we include the following new contributions: (1) We propose FIRB, a fine-grained and memory-light pipeline building block, and evaluate its impact on implementing an SESL architecture in terms of improving energy and memory efficiency. (2) We analyze the impact of FiBHA on the memory requirements of a CNN. FiBHA reduces the intermediate results memory requirements considerably, allowing better scaling and avoiding off-chip communication, especially on memory-constrained hardware. (3) We evaluate FiBHA, comparing its performance to a dedicated accelerator on three evaluation boards representing various resource budgets in the compute continuum, namely ZC706, KCU105, and ZCU102. (4) Moreover, we evaluate FiBHA's energy efficiency, comparing it to an SESL accelerator, an SEML accelerator, and two GPUs using a set of heterogeneous CNNs.

This publication incorporates results from the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957197. The Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project also partly supported this work.

Authors' address: F. Qararyah, M. W. Azhar, and P. Trancoso, Chalmers University of Technology, Chalmersplatsen 4, Göteborg, 41296, Sweden; e-mails: qarayah@chalmers.se, waqarm@chalmers.se, ppedro@chalmers.se.

ACM 1544-3566/2024/02-ART25
https://doi.org/10.1145/3639823

Additional Key Words and Phrases: Convolutional neural networks (CNNs), hardware software co-design, deep learning, hybrid accelerator, pipelined accelerator, FPGA

ACM Reference Format:
Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso. 2024. An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs. *ACM Trans. Arch. Code Optim.* 21, 2, Article 25 (February 2024), 26 pages. https://doi.org/10.1145/3639823

1 INTRODUCTION

During the early years following the AlexNet breakthrough [24], researchers have mainly focused on designing **Convolutional Neural Networks (CNNs)** with higher accuracy [17, 47, 50, 63]. The accuracy improvements usually come at the cost of higher memory consumption and computational complexity [52]. However, as CNNs have become dominant in computer vision, there has been growing interest in improving their resource efficiency both to perform cheaper training and inference and to facilitate their deployment in resource-constrained environments [9, 18, 19, 22, 45, 49, 64]. At the same time, domain-specific accelerators that process Deep learning algorithms, including CNNs, more efficiently are being proposed [5, 8, 14, 32, 48, 49, 55, 58].

CNNs' resource efficiency is improved by reducing the model's computational complexity using modules—building blocks or layer-combinations—with reduced memory and computational requirements [9, 18, 19, 45, 52, 64]. These modules require less weight and perform lesser computations. Hence, they enable building compact, computationally inexpensive, yet accurate CNNs [6, 18, 19, 50, 52, 64]. However, these modules introduce new layer types that increase CNNs' inter-layer-type heterogeneity. We use the term "heterogeneity" to describe the diversity of the layers in a CNN, especially the computationally intensive convolutional layers. For example, we say that ProxelessNAS [6] is more heterogeneous than VGG [47] because it contains more types of convolutional layers and because its convolutional layers that are of the same type have more variations of kernel sizes, input sizes and shapes, and reuse patterns.

An in-depth analysis of monolithic accelerators, e.g., **Tensor Processing Units (TPUs)** [21], reveals that the "one-size-fits-all" heterogeneity-oblivious accelerators are very inefficient when it comes to processing heterogeneous CNNs [5, 61]. Therefore, there has been an effort to design custom accelerators for these heterogeneous CNNs. Field Programmable Gate Arrays have been heavily used to develop such custom and model-specific accelerators due to their reconfigurability, which enabled supporting the rapidly changing CNN models [2, 13, 31, 32, 48, 58, 59, 62].

Most of the proposed custom and model-specific accelerators are composed of reusable engines, where a *single engine* computes *multiple layers* of the same type. We refer to such accelerators as **single-engine multiple-layer (SEML)** accelerators. For instance, in [29, 31, 48, 58], a *single engine* processes all the **Depthwise (DW)** convolutional layers [9] and another *single engine* processes all the **Pointwise (PW)** convolutional layers. Designing dedicated DW and PW engines, each optimized for a layer type, improves the efficiency compared to a monolithic accelerator that processes layers of both types using the same engine. Hence, we say that the SEML accelerators capture inter-layer-type heterogeneity. However, layers of the same type have various arithmetic intensities, input and output shapes, reuse patterns, workloads, and filter sizes [5, 55]. Consequently, even when there is a dedicated engine per layer type, this engine has to be optimized for the average case within that layer type. Hence, we say that SEML accelerators do not capture intra-layer-type heterogeneity.

To capture the intra-layer-type heterogeneity, accelerators with a *dedicated engine per layer* are proposed [4, 30, 53, 54]. We refer to such accelerators as **single-engine single-layer (SESL)** accelerators. They are known as streaming, or **synchronous dataflow (SDF)**, accelerators in the literature. These accelerator engines are pipelined and run simultaneously to achieve high throughput. Implementing a pure SESL design where each layer has its dedicated engine and where resources are well balanced among these engines is challenging and does not scale for deep models [15, 34]. Moreover, the simultaneously running engines need to be fed with data at the same time, requiring high memory bandwidth [30].

To summarize, neither pure SEML nor pure SESL accelerator architectures offer the best performance-resource tradeoff. While SEML accelerators ignore the intra-layer-type heterogeneity, SESL accelerators are resource-hungry and unscalable. Hence, we propose ***Fixed Budget Hybrid CNN Accelerator (FiBHA),*** which combines both SESL and SEML architectures. We use the term "hybrid" to describe an accelerator that uses more than one mapping between CNN layers and the hardware, e.g., a combination of SESL and SEML mappings. FiBHA captures more heterogeneity than pure SEML accelerators and is more resource-efficient than pure SESL. To derive an FiBHA instance given a fixed resource budget, we propose a heuristic **"Split a CNN" (SplitCNN)**. SplitCNN identifies a splitting point, and it splits the CNN layers and the resources between that instance's SESL and SEML parts.

To design the SESL part of FiBHA, one option is to implement a traditional pipeline, where a tile of each pipelined layer's **feature maps (FMs)** is stored in a double-buffered or **first-in-first-out (FIFO)** fashion [30, 44, 54]. However, this introduces a non-negligible memory overhead. This overhead scales with the pipeline length since more buffers must be added between the engines. Hence, to design a memory-efficient SESL accelerator part of FiBHA, we propose **Fused-Inverted-Residual-Bottleneck (FIRB)**, a fine-grained and memory-light SESL architecture building block. This block reduces memory and energy overheads compared to traditional pipelining. The core idea of FIRB-based design is building a two-level pipeline composed of a chain of *fused engines*, or engine-groups, rather than engines. Within each *fused engine*, the communication among its engine group is done at a fine granularity, which enables processing the intermediate results in a more timely manner and eliminates the need for double buffering.

The contributions of this article are as follows:

- We propose FiBHA, a hybrid architecture composed of an SESL part and an SEML part, each computing a subset of CNN model layers.
- We propose SplitCNN, a heuristic that derives an FiBHA instance given a fixed resource budget and splits a CNN and the computational resources between its SESL and SEML parts in a way that maximizes the throughput.
- To implement the SESL part of FiBHA efficiently, we propose FIRB, a fine-grained and memory-light SESL building block, and evaluate its impact on implementing a pipelined architecture in terms of improving energy and memory efficiency.
- We analyze the impact of FiBHA on the memory requirements of a CNN. FiBHA reduces the intermediate results memory requirements considerably, allowing better scaling and avoiding off-chip communication, especially on memory-constrained hardware.
- We evaluate FiBHA by comparing its performance to a pure SEML accelerator using three different boards representing various resource budgets in the compute continuum, namely ZC706, KCU105, and ZCU102. Moreover, we evaluate FiBHA by comparing its performance to a pure SESL accelerator and demonstrate that it offers better performance and resource efficiency. We also evaluate FiBHA's energy efficiency, comparing it to a pure SEML accelerator and two GPUs using a set of heterogeneous CNNs.

In this article, we focus on accelerating inference [49]. We evaluated FiBHA using different FPGAs with varying resource budgets representing different points in the compute continuum

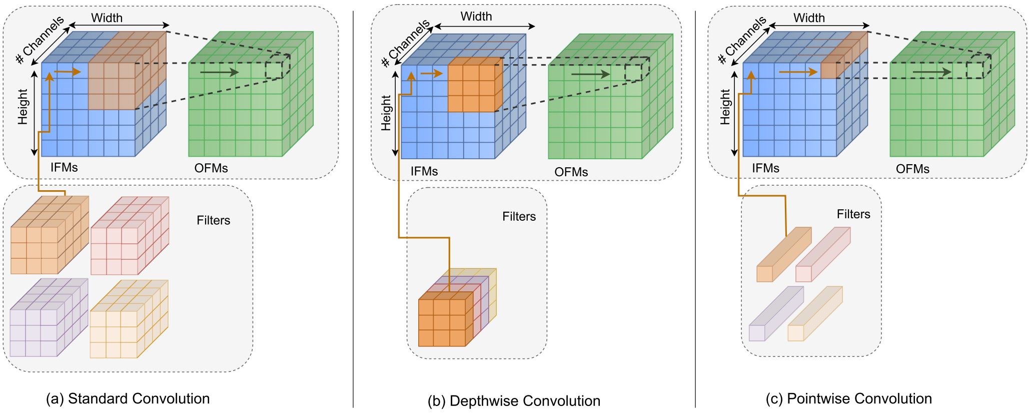


Fig. 1. Standard, Depthwise, and Pointwise convolution.

and state-of-the-art compact CNNs [6, 18, 45, 51]. FiBHA achieves 2.5x and 4x of the throughput achieved by state-of-the-art model-specific accelerators, under the same hardware budget. It also allows better scaling in cases of memory-constrained hardware compared to SEML. FIRB modules reduce the required memory by up to 54% and energy requirements by up to 35% compared to traditional inter-layer pipelining.

2 BACKGROUND

2.1 Convolutional Neural Networks (CNNs)

CNNs are feedforward deep learning models designed to capture features in 1D signals like language, 2D signals like images, or 3D signals like video or volumetric images [26]. A CNN consists of a set of stacked layers performing feature extraction and classification [27]. The primary layers in a CNN, both in terms of functionality and computational intensity, are the convolutional layers [8]. Figure 1 shows different forms of convolution that are found in CNNs. As the figure shows, a convolutional layer has a set of **filters**; these filters are composed of trainable **weights**. The filters are applied on **input feature maps (IFMs)** to extract embedded features from them, generating **output feature maps (OFMs)**. We use the term FMs, or activations, to refer to both IFMs and OFMs.

2.2 Resource-efficient CNNs

Resource-efficient CNNs are also referred to as heterogeneous, compact, or edge CNNs in the literature [5, 61]; hence, we use these terms interchangeably throughout this article. These CNNs are designed to balance the accuracy-efficiency tradeoff. They either improve the accuracy without increasing the model weights and computations [9] or reduce the model weights and computations considerably at the cost of a negligible loss in accuracy [6, 18, 45, 52, 64]. The DW Separable Convolution and inverted residual and linear bottlenecks are two predominant building blocks of resource-efficient CNNs [6, 9, 18, 45, 52, 64].

The DW Separable Convolution is a form of convolution that is based on decoupling the spatial and the cross-channel correlations in the FMs [9]. This is achieved by replacing the standard convolution by two operations: (a) DW convolution and (b) PW convolution. As Figure 2(b) shows, a DW Separable Convolution is composed of DW followed by PW convolution. Figure 1(b) shows that the DW convolution applies a filter to each FM and sums each feature map's results individually (no across-FMs summation). This is followed by a PW convolution with 1 × 1 filters.

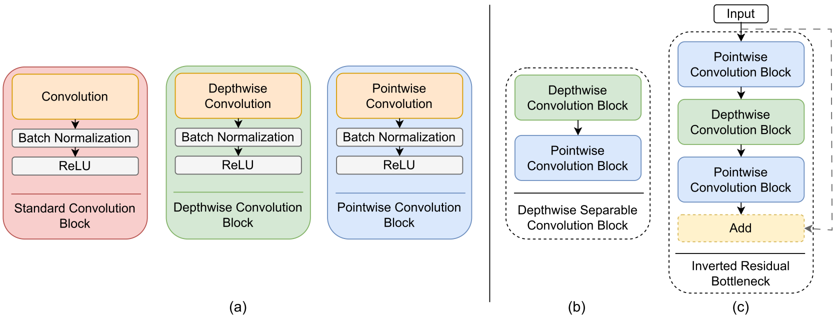

Convolution
Batch Normalization
ReLU
Standard Convolution Block
Depthwise Convolution
Batch Normalization
ReLU
Depthwise Convolution Block
Pointwise Convolution
Batch Normalization
ReLU
Pointwise Convolution Block
(a)
Depthwise Convolution Block
Pointwise Convolution Block
Depthwise Separable Convolution Block
(b)
Input
Pointwise Convolution Block
Depthwise Convolution Block
Pointwise Convolution Block
Add
Inverted Residual Bottleneck
(c)

Fig. 2. Depthwise separable convolution and inverted residual bottlenecks.

PW convolution, as shown in Figure 1(c), applies a filter and sums the results across different FMs but not within a single FM. DW Separable Convolution is a more efficient way to use the model parameters compared to the standard convolution [9, 59].

The inverted residual with linear bottlenecks is a module designed to significantly reduce the CNN model weights and operations while maintaining the accuracy [45]. As shown in Figure 2(c) this module combines three convolutional layers. The first layer is a 1 × 1 PW convolution that *expands* the IFMs by increasing their depth. The second is a DW convolution that processes the expanded FMs. The third is another 1×1 PW convolution that *squeezes* the FMs again. There could be a shortcut connection in the module that forwards the input of the module to be added to the outputs of the last convolutional layer of the module; this shortcut is represented by the dashed arrow connecting the module input to an *Add* layer in Figure 2. The reduction in weights and computations that the inverted residual with bottlenecks module permits is a result of performing heavy operations like the PW convolution on a relatively low-dimensional representation, that is, the unexpanded or the squeezed FMs, and a light operation like DW convolution on the expanded representation.

Since the modules used to design resource-efficient CNNs have different types of convolutional layers compared to the conventional CNNs that have only one type of convolution, we say that the resource-efficient CNNs have more heterogeneity. Note that even conventional CNNs have one form of heterogeneity, which is intra-layer-type. This is because even convolutional layers of the same type have variations in inputs, output, and filter shapes.

2.3 Model-aware CNN Accelerators

A plethora of custom, model-specific accelerators have been proposed aiming to improve the performance and efficiency of processing compact and heterogeneous CNNs. We categorize the custom accelerators in the literature into four categories:

- **Monolithic accelerators:** accelerators in which all the core layers are executed using the same engine [2, 8].
- **SEML:** accelerators in which all the layers of a certain type are executed using the same engine [2, 31, 32, 48, 58, 59, 62].
- **SESL:** accelerators in which each layer is mapped to a distinct engine. In these accelerators, the chain of engines is pipelined to work concurrently and maintain a high throughput [4, 30, 53, 54].
- **Multiple-Engine Multiple-Layer (MEML):** accelerators in which a single layer is processed by multiple engines, where each engine processes a tile of that single layer, and these multiple engines are reused across multiple layers [14, 34].

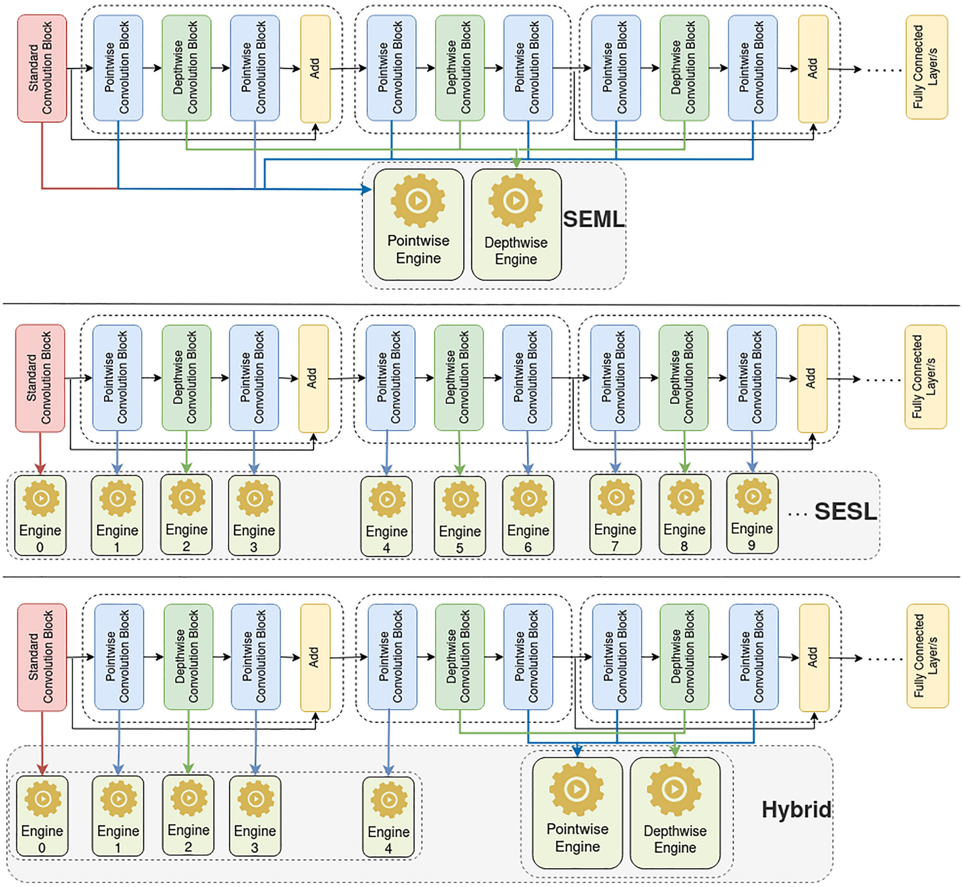


Fig. 3. High-level overview of the SEML, SESL, and Hybrid accelerators. The figure focuses on the mapping from the convolutional layers of a CNN to the compute engines of the accelerators; the internals of the engines, their connectivity, and the memory system are omitted for simplicity.

In the rest of this article, we mainly focus on the second and third categories, i.e., SEML and SESL. This is because the MEML is more tailored for resource-demanding scenarios like having large deep neural networks or the training phase. Note that in this context, the term *single layer* refers to a combination of convolution, batch normalization, and activation layers. We assume that a convolutional layer is fused with the **batch normalization (BN)** and the activation layer, e.g., **rectified linear unit (ReLU)**, following it. Note also that our taxonomy is based on the nature of the mapping between the model layers and the engines rather than the engine count. For example, considering an SEML, there could be multiple engines, but the point is that each *single engine* computes *multiple layers*. Figure 3 shows examples of both SEML and SESL mapping approaches. The top part of the figure shows an example of an SEML where all the DW layers are mapped to a CW engine, and all the PW layers and the first layer are mapped to a PW engine. This is a common mapping in custom accelerators targeting heterogeneous CNNs. In this mapping, PW and standard convolutions share the same engine. The middle part of the figure shows an SESL mapping where each layer is mapped to its own dedicated engine. Note that our taxonomy is independent of the internal design of these engines. These engines could either be custom convolution engines each highly optimized to its layer or a set of Systolic Arrays with the same architecture but with different

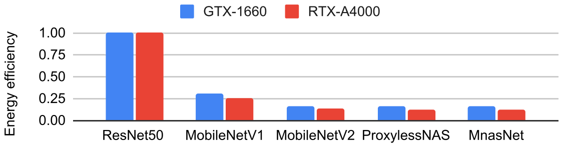


Fig. 4. Energy efficiency of a set of heterogeneous CNNs normalized to the energy efficiency of ResNet50.

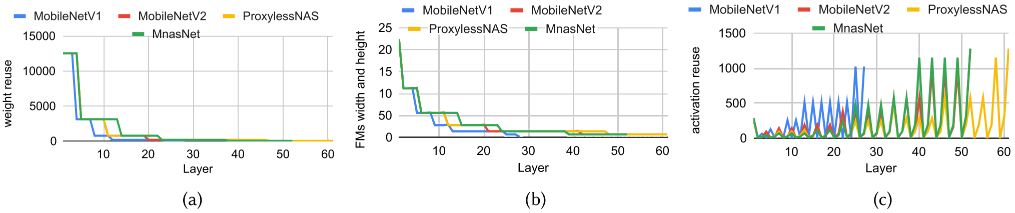


Fig. 5. (a) Weight reuse. (b) FMs spatial dimensions: width and height (note that FM width = FM height). (c) Activation (FM element) reuse.

dimensions to suit their varying layer's workloads. The bottom part of the figure depicts what we refer to as *Hybrid Accelerator*. We use the term "hybrid" to describe an accelerator that uses more than one mapping technique between CNN layers and the compute engines. The figure shows an example that maps each of the first five convolutional layers to its engines and uses an SEML mapping technique for the rest of the layers.

We refer to an SESL where all the engines have the same execution time as *perfectly balanced*. In a perfectly balanced SESL, theoretically, all engines are active all the time, eliminating any resource idleness or underutilization. We use the term **processing element (PE)** as an abstraction. It refers to hardware capable of doing a MAC operation regardless of the components that are actually used in a particular implementation (e.g., in an FPGA, a PE could be implemented using **Lookup Tables (LUTs)** or a **Digital Signal Processor (DSP)**).

3 MOTIVATION

The Depthwise Separable Convolution and the inverted bottlenecks (Section 2) reduce CNNs' storage and computational requirements. However, they increase the CNN model heterogeneity as these modules include layers of varying arithmetic intensities, reuse opportunities, FMs and weight shapes, and memory requirements. This makes the generic accelerators unable to fully achieve the desired performance gains out of this reduction. Figure 4 shows an example of that; it depicts the energy efficiency of four compact and heterogeneous CNNs normalized to that of ResNet50 [17]. In contrast to ResNet50, these heterogeneous CNNs have much lower energy efficiency.

Figures 5(a) and 5(c) depict some aspects of the heterogeneity in a set of compact CNNs, namely MobileNetV1 [18], MobileNetV2 [45], ProxylessNAS [6], and MnasNet [51]. Figure 5(a) depicts the weight reuse of the models on the y-axis and layers on the x-axis. We use the term *reuse* to indicate the number of times, or operations, where a single element (weight/input/output) is used. To give an example, in Figure 1(c), each of the PW filters slides over the width and height of the IFMs and is used to produce one output feature map. Since the number of elements in one output feature map equals *OFMs width * OFMs height*, a single weight or filter could be loaded from memory once and used to do that many computations. The reuse indicates the compute-to-memory access ratio of an element. Capturing reuse is important as it reduces data movement, which in turn reduces the latency [8, 33] and energy. Figure 5(a) shows that the initial layers have high and dynamic weight reuse; the reuse is low and stable in the rest. Figure 5(b) shows the width and height of the FMs in the targeted CNNs. The figure shows that the FMs of the initial layers have relatively large widths and heights. As a result, the weight reuse in these layers is the highest. The figure also suggests that in the first layers, the width and height of the FMs change, more specifically decrease, frequently and after that, they change less frequently. As a result of the frequent changes in the FM's width and height, the weight reuse is more dynamic in these first layers. As a result of the less frequent change in the rest of the layers, the reuse does not vary as much.

Figure 5(c) depicts the activation reuse. For DW layers, activation reuse is calculated as *filter width* × *filter height*, except for the activation on IFM borders and in case of strides of greater than one. For PW layers, activation reuse is equivalent to the *number of filters*. Activation reuse follows an opposite trend compared to weight reuse, low in the initial layers and high in the rest. Moreover, unlike weight reuse, activation reuse fluctuates. This is a result of the inverted bottleneck module and the DW convolutions. The peaks of the activation reuse represent the reuse values of all the PW layers in the MobileNetV1 case and the expansion layers (Section 2) in the rest. The valleys represent the DW layers. Note that activation reuse is negligible in the case of DW layers. Hence, regarding activation reuse, we focus on the higher and more dynamic reuse of the PW layers.

Reuse heterogeneity is one of many forms of heterogeneity in the studied CNNs. SEML accelerators fail to capture the intra-layer-type form of heterogeneity. SESL accelerators could address the two forms, but they are impractical for deep CNNs and do not scale [34, 43]. Researchers proposed several techniques to alleviate SESL bottlenecks [3, 4, 54], but each of these techniques has its own shortcomings.

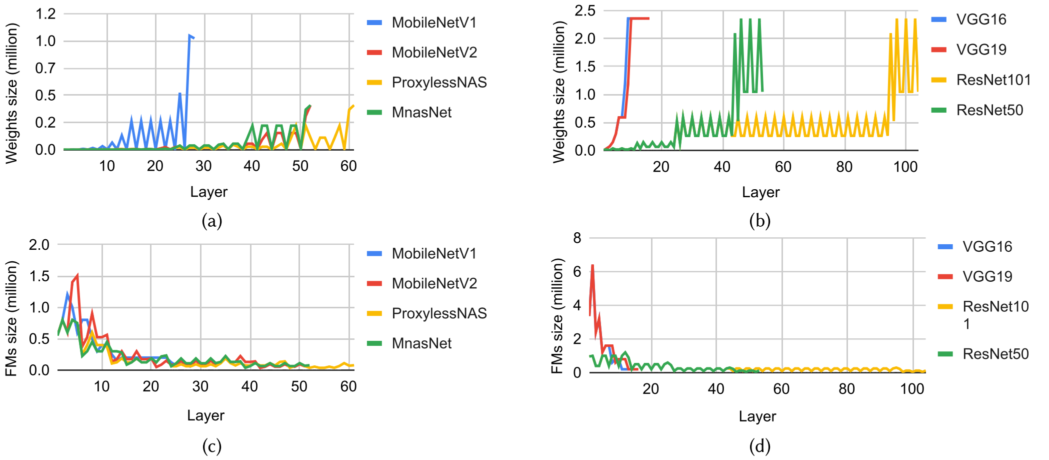
Motivated by the observation that neither SEML nor SESL accelerators address the tradeoff between addressing heterogeneity and maintaining resource efficiency well enough, we propose our hybrid architecture (FiBHA). Based on Figures 5(a) and 5(c), we can generalize by saying that the layers in the targeted CNNs can be split into two parts. The first part has high weight reuse and low input reuse, while the second has low weight reuse and high input reuse (considering input reuse only in the PW layers). This trend motivates designing a hybrid accelerator that uses two forms of architectural design to handle these two parts.

4 FIBHA

4.1 Design Choice: First SESL Then SEML

The primary design goal of FiBHA is to improve efficiency by capturing the two forms of heterogeneity present in resource-efficient CNNs as much as the available resource budget permits. The first form of heterogeneity is the inter-layer-type heterogeneity, which is a result of having different types of layers including DW, PW, and Standard convolution. The second is heterogeneity among layers of the same type, represented in variations in input and output shapes, sizes, filter shapes, and reuse patterns, as discussed in Section 3. When designing FiBHA, we mainly consider the PE budget. However, we also target minimizing memory consumption, which leads to energy savings, and we avoid the off-chip memory bandwidth bottleneck. In the rest of this section, we discuss why using SESL to process the initial layers of an CNN and SEML to process the rest is the option that fulfills these design goals. It is important to note that while this arrangement is the best for CNNs, as we show in the rest of this section, our design is not restricted by it, meaning that in other workloads, it could be beneficial to use other arrangements.

4.1.1 Maximizing the Captured Heterogeneity. SESL captures the two levels of heterogeneity; hence, it is preferred over SEML when there is more heterogeneity. The initial layers exhibit more heterogeneity than the rest. Three forms of heterogeneity are much more visible in the initial layers:



(a)

(b)

(c)

(d)

Fig. 6. Weight size per layer of (a) heterogeneous and (b) less heterogeneous CNNs. FM size per layer of (c) heterogeneous and (d) less heterogeneous CNNs, respectively.

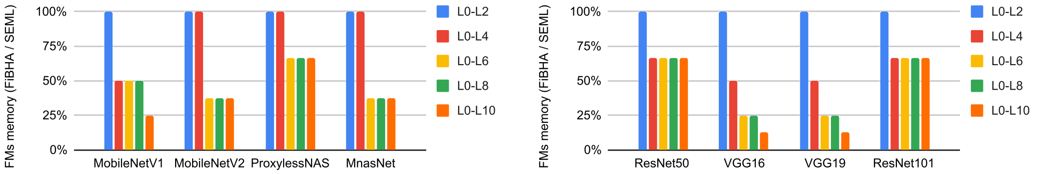
- **Weight reuse heterogeneity:** Figure 5(a) shows that the weight reuse in the initial layers is changing dramatically. However, in the rest of the layers, the weight reuse either is stable or changes negligibly.
- **Parallelism pattern heterogeneity:** The initial layers, especially the first layer, have shallower inputs compared to the rest, which means they exhibit different parallelism patterns. As a result, these layers have low resource utilization when computed using an engine that is suitable for the majority of the layers. Previous work suggested computing the first layer on the CPU [7, 35].
- **Inter-layer-type heterogeneity:** Another form of heterogeneity that the first layer introduces, considering all the compact CNN models we have evaluated, is inter-layer-type heterogeneity. This layer is the only standard convolution, making its filters unique.

4.1.2 Minimizing FM Memory Requirements. The memory consumption of Deep Learning algorithms represents a major bottleneck both in training and in inference [14, 40]. In some state-of-the-art accelerators, on-chip buffers consume up to 70% to 87% of the chip area [8, 14]. As a result, reducing memory requirements is a crucial part of designing an efficient accelerator.

Figure 6(c), which depicts layers on the x-axis and the size of FMs on the y-axis, shows that the initial layers have much larger FMs than the rest. CNNs' initial layers extract many simple features from an input, e.g., edges and curves in an image. As we go deeper, the layers combine the many simple into fewer more abstract features. That is why FMs' size shrinks as we go deeper. The shrinking is done in practice by sub-sampling through pooling layers or skipping through strided convolutions.

Unlike SEML, where the outputs need to be fully stored in memory and then loaded when the next layer is computed, in SESL data flows between the pipelined engines and is processed almost immediately. As a result, if SESL is used to process the initial layers, smaller on-chip buffers are needed to store the FMs. On the other hand, the rest of the layers have small FMs, meaning that storing them fully is relatively cheap.

Figure 7(a) shows the FMs' memory requirement of FiBHA normalized to SEML or a monolithic accelerator. The figure shows that up to 75% of the required FMs' memory could be saved. An SEML accelerator requires an FM's buffer size that can accommodate the maximum IFMs plus the OFMs of a layer. This assumes that two buffers are allocated to hold the IFMs and OFMs of a layer



(a) (b)

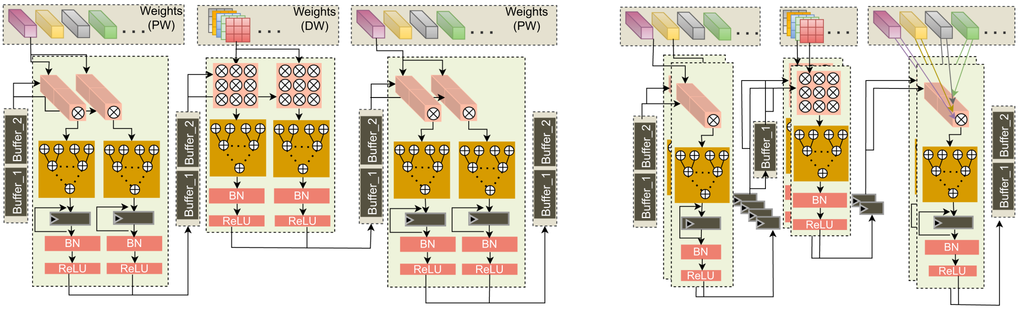
Fig. 7. Maximum FM memory requirement of FiBHA normalized to SEML or monolithic accelerator considering various SESL part lengths of (a) heterogeneous and (b) less heterogenous CNNs. L0-Lx means that the SESL part of FiBHA spans the layers starting from layer 0 up to layer x - 1.

and are reused in an alternating fashion throughout an inference. FiBHA requires an FM buffer size that can accommodate the maximum IFMs plus the OFMs of a layer in the SEML part (starting from the last SESL layer + 1) plus the SESL part overhead. The SESL part overhead depends on the granularity of the pipeline and is negligible for all the considered SESL lengths. Note that in case of having skip connections, as in the case of inverted bottlenecks (Section 2), a layer has more than one input or output. For models that have a linear structure with no skip connections like MobileNetV1 or VGG, one could make a rough estimate of the FMs' buffer size reduction, by looking at Figure 6(c) or 6(d). The size of the required buffer in the case of a pure SEML accelerator is the sum of the two highest peaks. In the case of FiBHA, the required buffer size is roughly the sum of the two highest peaks starting from FiBHA's SEML part first layer. Equation (1) gives the memory reduction for any CNN ignoring the SESL part small buffering overheads, where *sl* is the splitting layer, which is the first layer processed by the second part of FiBHA, and *ll* is the last layer in the CNN:

$$\begin{aligned} FMs'\ memory\ reduction = max&\left(\sum_{IFMs \in l_x\ inputs} size(IFMs) + \sum_{OFMs \in l_x\ outputs} size(OFMs)\right), l_x \in [0, ll] \\ -max&\left(\sum_{IFMs \in l_y\ inputs} size(IFMs) + \sum_{OFMs \in l_y\ outputs} size(OFMs)\right), l_y \in [sl, ll]. \end{aligned} \tag{1}$$

This reduction in FMs' memory requirements has two benefits. First, when using on-chip memory to store FMs, relying on large buffers is expensive in terms of latency, area, and power [14]. Second, embedded accelerators may not have sufficient on-chip memory; in this case, FiBHA helps to fit the FMs on a smaller on-chip memory and avoid relying solely on the costly-to-access DRAM. In Section 6.1, we demonstrate experimentally the benefits of FMs' memory reduction. Note that the results of the analysis *in this section* are generalizable and apply to other more homogeneous CNNs, like ResNet [17] and VGG [47]. Figures 6(b) and 6(d) show that these CNNs' FMs and weights follow similar trends: while the weights grow bigger, FMs shrink as we go deeper. Hence, the advantage of FiBHA is not limited to compact CNNs; it applies to other CNNs as shown in Figure 7(b).

4.1.3 Avoiding Off-chip Memory Bandwidth Bottleneck. An SESL architecture suffers from the off-chip memory bandwidth bottleneck since all the pipelined engines must be supplied with weights concurrently. As Figure 6(a) shows, the initial layers have small weights that can be stored on-chip locally. As a result, these layers could be processed with an SESL architecture while avoiding the off-chip memory bandwidth being a bottleneck. On the other hand, the layers toward the end of a model have much larger weights that usually have to be stored off-chip. Note that the increase in the size of the weights as we move from the first toward the last layers happens because the depth of FMs channels increases, requiring more and larger filters.



(a)

(b)

Fig. 8. (a) Traditional inter-layer pipelining-based implementation of an Inverted Residual Bottleneck module. (b) FIRB-based implementation of an Inverted Residual Bottleneck module.

Based on this analysis, to capture more heterogeneity, minimize memory requirements, and avoid bandwidth bottlenecks, the SESL part is used to process the initial layers. The SEML is used to compute the rest.

4.2 Fused Inverted Residual Bottlenecks (FIRB)

As discussed in Section 4.1.1, the motivation behind using SESL to process the first layers is minimizing memory requirements. However, in a traditional implementation of the pipeline, a tile of the FMs of each layer needs to be stored on-chip and double-buffered for concurrent access, introducing a non-negligible overhead. This overhead scales with the pipeline length as more double buffers must be added.

To reduce the buffering required by a pipelined architecture, Gao et al. have proposed a novel dataflow named **Alternate Layer Loop Ordering (ALLO)** [15]. However, it is only possible to apply ALLO to alternate pairs of adjacent layers. Hence, it requires to fully delay and completely buffer the FMs of half of the pipelined layers. While ALLO improves over naive coarse-grained pipelining where the whole FMs need to be entirely stored between each pair of layers, it still is very expensive when considering a limited on-chip memory. Hence, we introduce the FIRBmodule. FIRB applies a fine-grained pipelining within the inverted residual bottleneck modules (Section 2), resulting in a considerable reduction in the required buffers.

Figures 8(a) and 8(b) show two implementations of an Inverted Residual Bottleneck module. The implementation in Figure 8(a) is a variant of a traditional inter-layer pipelining. This implementation maximizes weight reuse across the three layers of the module. Figure 8(b) shows an implementation of FIRB; this implementation maximizes input reuse in the first layer (the expansion layer), sacrifices the reuse in the second (the DW layer), and maximizes the output reuse in the third (the projection layer). The main advantage of FIRB is eliminating the need for three out of the four buffers connecting the expansion to the DW and the DW to the projection layer (Figure 8(b)). Note that the four intermediate buffers are three to six times bigger than the four outer buffers because they hold *expanded* FMs (Section 2), and the usually used expansion ratios are three and six times. One of the four intermediate buffers has to be kept to hold the FM values that are shared between two consecutive passes of the DW layer filters.

The advantage of reducing buffers allowed by FIRB comes at the cost of sacrificing the reuse in the DW layer and imposing an extra constraint on the parallelism patterns. Sacrificing the reuse in the initial DW layers, as the SESL part targets the initial layers of a CNN, is not costly as they hold hundreds of weights that are negligible compared to the millions of weights in the whole model.

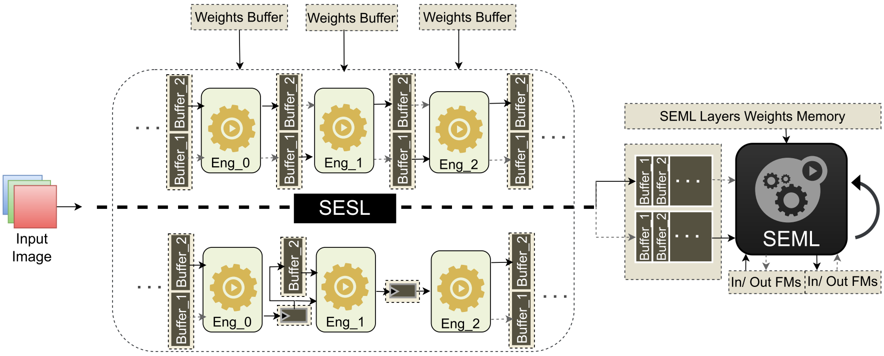


Fig. 9. FiBHA architecture: a high-level overview. The SESL part implementation is FIRBbased or traditional tiled pipelining based depending on the CNN.

The parallelism patterns constraint requires that the three layers target the three dimensions of the FMs in a way that permits having a common blocking factor across them (the three layers). Common blocking is important to balance the computational loads and avoid leaving any of the engines idle. One challenge that this constraint introduces is the need to gather the weights from multiple filters in the projection layer, as shown in Figure 8; this is required in case two or more levels of memory are used. However, this challenge can be resolved by simply rearranging and storing the layer weights in a round-robin fashion offline. Note that the weights do not change during the inference, so the rearranging is done only once.

We implement a relatively coarse-grained pipelining across FIRB's modules. This is because the fine-grained pipelining cannot be extended to cover multiple modules as the expansion and the projection layers have to work in an alternating fashion to maintain the common blocking factor. This means that a FIRB module produces a set of *d_squeezed* elements each *d_expanded* cycles, where *d_squeezed* is the depth of the squeezed FMs and *d_expanded* is the depth of the expanded FMs, rather than a single element per cycle. However, compared to ALLO [15], our course-grained pipelining has finer granularity and does not require stalls or storing the whole FMs.

FIRB supports models that use the Inverted Residual Bottleneck as a building block; those represent the majority of the compact CNNs [6, 45, 51, 64]. However, the older families of compact models, like MobileNetV1, do not have this module. Hence, to support both, we construct the SESL part of FiBHA using FIRB or traditional pipelining based on the targeted model as shown in Figure 9.

4.3 Architecture Overview

Figure 9 shows a high-level overview of FiBHA architecture. A FiBHA instance is composed of two main parts: an SESL and an SEML. Double buffering is used to bridge these two parts such that both can work concurrently. For instance, while the SEML part is processing the input n, the SESL could process the next input $(n + 1)$. The SESL part is composed of a chain of FIRB modules if they support the targeted CNN; otherwise, it is composed of a tiled traditional inter-layer pipelining. The SEML part is represented as a black box to indicate that any SEML design, e.g., the two-engine design presented in Figure 3, could be used as a part of FiBHA. This is explained in detail in the next section (Section 4.4).

In the SESL part, the engines are pipelined, and double buffering is used between the engines when necessary. Note that these buffers are much smaller than SEML buffers. In addition to the buffers between the engines, each engine has its own weight buffer. The input image is processed

ALGORITHM 1: SplitCNN

```
In: number_of_PEs, CNN_model, SEML_overhead
Out: FiBHA_throughput, FiBHA_PEs_distribution
split_layer ← 2
SESL_PEs ← []
FiBHA_PEs_distribution ← [[], num_of_PEs]
FiBHA_throughput ← GET_THROUGHPUT(number_of_PEs, 0, CNN_model, SEML_overhead)
while split_layer < number_of_layers do
    SESL_PEs ← GET_BALANCED_SESL_PEs(split_layer, CNN_model)
    SESL_max_PEs, _ ← GET_PE_RATIOS(split_layer, CNN_model)
    while SESL_PEs ≤ SESL_max_PEs do
        tmp_FiBHA_throughput ← GET_THROUGHPUT(number_of_PEs, SESL_PEs, CNN_model, SEML_overhead)
        FiBHA_throughput, FiBHA_PEs_distribution ← MAXIMIZE(FiBHA_throughput, tmp_FiBHA_throughput)
        SESL_PEs ← AUGMENT_SESL_PEs(SESL_PEs)
    end while
end while
```

in chunks. Assuming we have l engines in the SESL part, while Engine_0 is processing chunk $l + n$, Engine_1 handles chunk $l + n - 1$, and so forth. Once Engine_$l - 1$ completes a chunk, it pushes it to one of the two buffers connecting the SESL part to the SEML part. Each of these engines is assigned a number of PEs such that the execution times of the engines are the same, which makes the SESL part perfectly balanced.

Layers $l + 1$ up to the last layer are executed using the SEML part. This part could be implemented using any SEML architecture [2, 31, 32, 48, 58, 59, 62]. This part processes its layers one by one and has two buffers used in an alternating fashion to load and store the inputs and the outputs.

The implementation details of the SEML part engines are not a part of the core contribution of this work. However, in Section 5, we provide the details of an FPGA-based implementation example. To be as generic as possible, FiBHA works at a higher level of abstraction, making the following assumptions:

(1) Any SEML accelerator, e.g., $SEML_x$, is not capable of capturing the intra-layer-type heterogeneity. This results in inefficiencies like execution time/throughput overhead.
(2) Each engine in an SESL accelerator should have a negligible overhead. This is because it is designed and optimized for a specific layer. If the workloads are perfectly balanced among all the SESL engines, we could generalize by saying that the SESL part has a negligible overhead as well.

From (1) and (2), given $SEML_x$ and an available PE budget and a CNN (CNN_x), we argue that a hybrid architecture that is composed of an SESL part and $SEML_x$-like part outperforms the pure $SEML_x$ if (a) CNN_x can be split and PEs can be redistributed such that the ratio of $SEML_x$ PEs to its workload remains the same or increases and (b) the SESL part execution time is upper-bounded by the SEML execution time. Note that we apply the second assumption regardless of the implementation of the SESL part, which could be FIRBbased or traditional pipelining.

4.4 SplitCNN: A Heuristic to Derive FiBHA Instance

To optimize FiBHA's performance, a CNN and the PE budget should be split between the two parts of FiBHA such that (a) the execution times of the SESL engines are balanced to avoid any within-pipeline idleness (to guarantee assumption 2 in Section 4.3), and (b) as the slower of the two parts forms a bottleneck, the maximum of the SESL part execution time and the SEML part execution times should be minimized to maximize the overall throughput. We have designed a heuristic named **split CNN (SplitCNN)** to split a CNN and the PEs' budget in a way that meets these two objectives. The high-level pseudocode of the main part of SplitCNN is shown in Algorithm 1.

$$SESL\_PEs = \frac{1}{GCD(comp(l_x))} \times \sum_{l=0}^{sl} comp(l), l_x \in [0, sl) \tag{2}$$

$$SESL\_throughput \propto \frac{1}{max(comp(l)/PEs(l))}, l \in [0, sl) \tag{3}$$

$$SEML\_throughput \propto \frac{1}{\sum_{l=sl}^{ll}(1 + Overhead(l)) \times \frac{comp(l)}{PEs(l)}} \tag{4}$$

The starting point or the baseline throughput that SplitCNN aims to improve is that of a certain SEML, or a monolithic architecture. This starting point could be any of the state-of-the-art compact model-specific accelerators, or even an implementation of generic DL accelerators like Eyeriss [8] or Systolic Array [57]. SplitCNN takes as input the number of PEs, the CNN model, and the baseline SEML overhead. The CNN model is a data structure that contains, for each CNN layer, the IFMs' and OFMs' shape, weights' shape, strides, and layer type (DW/PW/Standard convolution). The SEML overhead is the difference between the theoretical and the practical SEML execution time. The theoretical execution time is the result of Equation (4) when substituting 0 for the *Overhead*. SplitCNN produces an FiBHA instance description of the best-performing instance given the PEs' budget. The instance description contains the splitting layers and the PEs' distribution on each of the SESL part engines and the SEML part. It also provides the estimated throughput of that instance.

SplitCNN explores the available splitting layer options and tries to find a hybrid architecture that improves over the baseline SEML throughput. Initially, it is assumed that all the PEs are assigned to the SEML part; the throughput is obtained accordingly using Equation (4) (line 6). The main loop, starting at line 7, explores the viable splitting options (*split_layer*) starting from the second layer up to the last. Note that “splitting at the last layers” is actually not splitting; it is rather a pure SESL architecture. This means that if the number of PEs is sufficient to implement a pure and perfectly balanced SESL architecture, it could be suggested by SplitCNN.

In each iteration of the main loop (for each *split_layer*), there could be multiple viable PE distributions. Initially, the SESL part is assigned the minimum number of PEs that is needed to form a balanced pipeline of depth *split_layer*; this is obtained at line 8 using Equation (2) (where *comp(l)* is the operation count in layer *l* and *sl* is the *split_layer*). The SEML part is assigned the rest of the PEs. Then the maximum amount of PEs that could be assigned to the SESL part is obtained (line 9); this depends on the ratio between the workloads of the two splits. For each PE distribution between the SESL minimum and maximum, in the inner loop (lines 10–14), the throughput of FiBHA is estimated, and the combination that maximizes it is kept as the best option of *split_layer*. Throughput estimation is described in Equations (3) and (4), where *ll* is the last layer in the CNN. Figure 10 shows examples of the splittings suggested by splitCNN under different PE budgets; these budgets correspond to the number of DSP slices on the used evaluation boards (Table 2). Table 1 shows the PEs' distribution for a FiBHA instance examples.

4.5 FiBHA Instance Generation

To facilitate generating an FiBHA instance given a CNN model and PE budget, we developed a set of utilities and an HLS engine repository. Currently, the engine's repository contains HLS implementations of the Standard, PW, and DW convolution; fused ReLU-BN-quantization; and pooling layers. These utilities map a model description written in a high-level framework using the engine's repository to an HLS-based hardware description. The mapping steps are described in

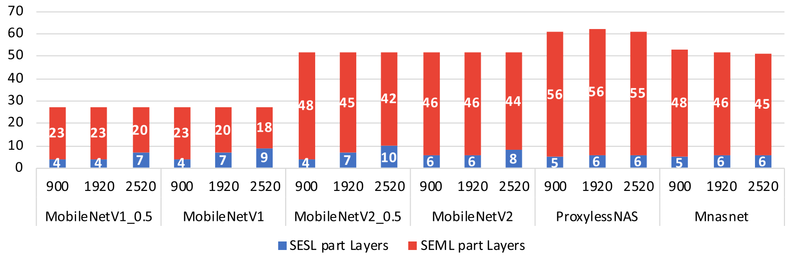


Fig. 10. Number of layers assigned to each of the FiBHA parts. The numbers on the x-axis represent the PEs' budget.

Table 1. SplitCNN Suggested FiBHA Architecture for MobileNetsV1_0.5 and 2,048 PEs

Split layer	7						
SEML PEs	1,693						
SESL PEs	355						
SESL Engines	Engine_0	Engine_1	Engine_2	Engine_3	Engine_4	Engine_5	Engine_6
PEs/Engine	54	18	64	9	64	18	128
MAC/Engine	5,419,008	1,806,336	6,422,528	903,168	6,422,528	1,806,336	12,845,056

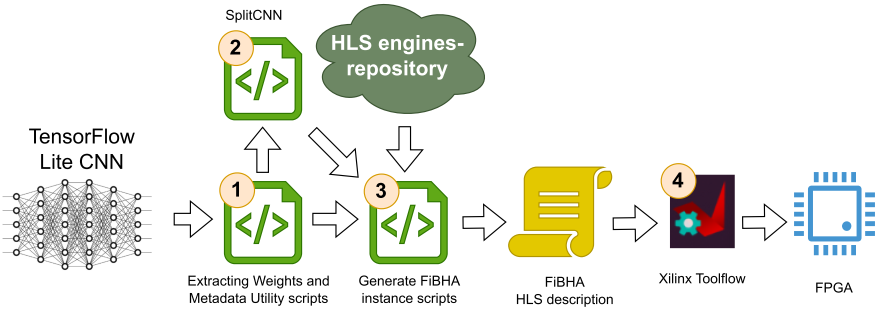


Fig. 11. FiBHA instance generation flow.

Figure 11. Our utilities take a CNN model description written in a high-level framework—we support Tensorflow lite [20]—and extract the weights and metadata. The metadata is fed to SplitCNN, which in turn generates a recommended PE distribution and splitting point. The data generated by SplitCNN and the CNN weights are used to generate the HLS code of the FiBHA instance. This HLS code is fed to the Vitis tool flow to generate the accelerator back-end. All the evaluated FiBHA accelerators' HLS implementations are generated using this flow. This set of utilities is being developed and extended aiming to support more CNNs and provide a fully automated end-to-end mapping flow.

5 EXPERIMENTAL SETUP

5.1 Synthesis Tools and Hardware Platform

We used Vitis-IDE, Vitis-HLS, and Vivado (2021.2) to implement and evaluate our accelerator instances. The evaluation is done using three FPGA boards representing different points of resource budgets. These boards are ZCU102, ZC706, and KCU105; their details are listed in Table 2. Moreover, we deployed FiBHA on a physical board, Zynq UltraScale+ MPSoC ZCU102, with an XCZU9EG

Table 2. The used Evaluation Boards

Board	LUT (K)		DSP Slices		Block RAM (Mb)	
	available	max utilization	available	max utilization	available	max utilization
ZC706	218.6	82%	900	89%	19.1	92%
KCU105	242.4	72%	1920	68%	21.1	80%
ZCU102	274.1	86%	2520	83%	32.1	68%

Note that the memory size is reported is in megabits.

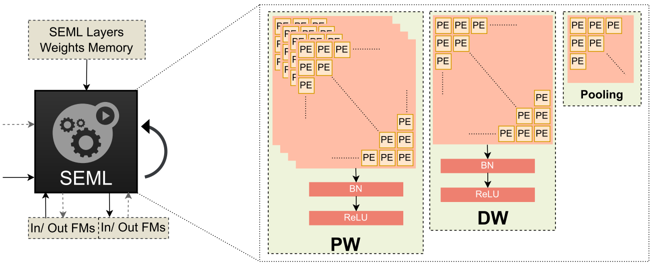


Fig. 12. B_SEML: The baseline SEML.

FPGA and quad-core ARM Cortex-A53. The used clock frequency for FiBHA instances is 180 MHz, as it is the highest supported by most of the designs.

A resource budget determines the combined degrees of parallelism (mainly expressed as loop unrolling in HLS) of all the engines of the accelerator under evaluation. For example, in the FiBHA instance that uses a budget of 1,024 PEs, the parallelism in the SESL part engines and the SEML part engines combined is less than or equal to 1,024.

Our designs' power is measured with Vivado Power Analyzer using the post-route netlist. This approach is commonly used to produce accurate power dissipation estimations [12, 25]. We have compared the energy efficiency of our designs with two generic NVIDIA GPUs, namely GTX-1660 and RTX-A4000. To measure the power consumption of these GPUs, we use the *nvidia-smi* monitoring utility. The frequency is set to the maximum supported for both GPUs.

5.2 Baseline Accelerators

5.2.1 B_SEML. Figure 12 shows the main engines of the baseline SEML we used in our implementation and evaluations. It has two separate convolution engines, one for PW convolution and another for DW convolution. It also has a pooling engine, as all the models we evaluate contain a pooling layer. This design is similar to many state-of-the-art compact CNN model-specific accelerators [31, 48, 58]; we refer to this accelerator as *B_SEML*. The accelerator designs that we based B_SEML on were mainly proposed to accelerate MobileNets variants, as they are the most heavily targeted compact CNNs. To support different models and resource budgets:

- B_SEML supports different DW filter sizes (e.g., 3×3 filters in MobileNets and 3×3 and 5×5 in Mnasnet and ProxylessNAS).
- The PEs are distributed among the engines depending on their workloads. For each engine, the used parallelism on each dimension is set to a factor of the available parallelism (input/output dimensions) to fully utilize the PEs [33]. When scaling the PEs' budget, the number of PEs in each engine is scaled accordingly as long as the rest of the resources suffice.

B_SEML is used to implement the SEML part of FiBHA (Figure 9). Note that the SEML part of FiBHA has the architecture of B_SEML, but usually with a smaller PE budget since some of the

Table 3. Evaluated CNN's Convolutional Layers and Operation Counts

CNN	Standard Conv Layers	DW Layers	PW Layers	Parameters (Million)	MAC (Billion)
MobileNetV1	1	13	13	4.2	0.57
MobileNetV1_0.5	1	13	13	1.3	0.15
MobileNetV2	1	17	34	3.4	0.3
MobileNetV2_0.5	1	17	34	1.9	0.07
ProxylessNAS	1	20	40	4.1	0.27
MnasNet	1	17	34	4.4	0.26

PEs are used to implement the SESL part. We argue that replacing B_SEML with another SEML accelerator will not change the trends of the evaluation. This is because the performance difference between B_SEML and FiBHA comes from capturing more heterogeneity rather than how optimized each of the designs is (Section 6.1).

5.2.2 FINN. FINN is an open-source framework that generates a model-specific streaming-dataflow accelerator. In the generated accelerator, each layer has its own dedicated engine [4]. FINN gives the ability to specify the desired throughput and generates an accelerator design that guarantees that throughput as long as the available resources can support it. In our experiments, we report the maximum supported throughput by FINN and the corresponding resource utilization. This means that the PE budget, in this case, is the maximum available on the FPGA, and that FINN cannot produce an accelerator with higher throughput using more resources. We used only MobileNetV1 variants in our comparison with FINN as they are the only models in our set with an available FINN end-to-end open-source implementation. It was not possible to generate FINN accelerators for the other CNN models in our experiments. For these models, FINN fails at the streamlining stage, which is one of its transformation stages. More specifically, it fails to convert the addition and scaling operations in these CNNs to *thresholding operations* [4].

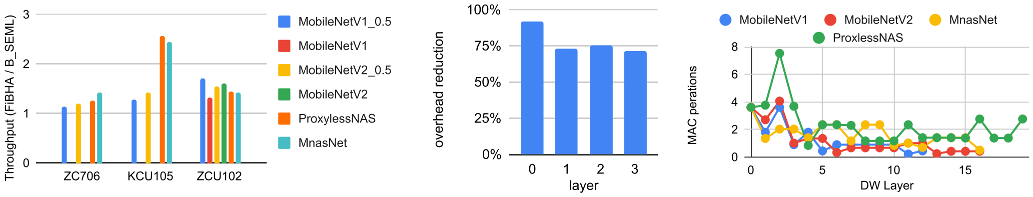
5.3 Evaluated CNNs

Our evaluation used a representative set of compact heterogeneous and hardware-efficient CNNs. They are MobileNetV1, MobileNetV2, ProxylessNAS, and MnasNet [6, 18, 45, 51]. CNN descriptions are listed in Table 3. We also used MobileNetV1_0.5 and MobileNetV2_0.5, which are lighter variants having a *width multiplier* of 0.5 [18], especially in the cases where the original variants do not fit due to resource constraints. We picked MobileNets as they are the most targeted resource-efficient CNNs in the literature (Section 7). We picked ProxylessNAS and MnasNet models as they represent a class of compact CNNs that are designed using NAS algorithms. Inputs of size 224 × 224 × 3 were used (ImageNet input size [10]). We used 8-bit quantization (INT8) for weights and FMs and 32-bit for partial sums. We implemented the quantization scheme used in Tensorflow Lite [20]. Hence, the accuracy is proportional to that of the quantized model provided by Tensorflow Lite. When comparing to FINN, we used 8 bits for the first layer and 4 bits for the rest to match the quantization offered in the FINN-supported open-source models [37].

6 EVALUATION

6.1 FiBHA vs. SEML (B_SEML)

Figure 13(a) shows the throughput of FiBHA compared to B_SEML using various boards and CNN models. For each evaluation board and CNN, we report the throughput of FiBHA normalized to that of B_SEML. The figure shows that there is always an instance of FiBHA that outperforms B_SEML. This demonstrates that the hybrid architecture has an advantage under different resource budgets.



(a) (b) (c)

Fig. 13. (a) FiBHA throughput normalized to B_SEML throughput; the missing bars represent cases where B_SEML FMs did not fit on-chip. (b) Overhead reduction in the first four layers of MobileNetV2 using FiBHA compared to B_SEML on ZCU102. (c) Number of MAC operations per DW layer.

The figure depicts three scenarios; we discuss these scenarios in detail in the following paragraphs. The first scenario is where the throughput improvement is between $1x$ and $2x$. The second scenario includes cases where there are missing bars. The third scenario is where the improvements are greater than $2x$.

In the first scenario, the throughput improvements are a result of FiBHA capturing more heterogeneity than B_SEML. Unlike B_SEML, FiBHA processes the initial layers that are heterogeneity-rich (Section 4.1) using their own specifically optimized engines (Sections 4.3). Figure 13(b) shows the overhead reduction in the first four layers of MobileNetV2 when implemented, each using its own dedicated engine (FiBHA's SESL part), where the overhead is the difference between the measured and the ideal execution time of a layer. Similar trends are found in the rest of the evaluated CNNs. We report the results for the first four layers as they are common in all SESL parts of FiBHA in our experiments (Figure 10). This reduction in overhead is the highest for the first layer, which is a standard convolution layer. This is because this layer suffers the lowest resource utilization with B_SEML mainly due to its shallow IFMs (parallelism pattern heterogeneity). The first layer forms a bottleneck in other state-of-the-art accelerators as well [7, 35]. The overhead reduction in the rest is a result of, in part, capturing heterogeneity in activation and weight reuse (Section 3). Another form of heterogeneity is the variation in the computational load, or the number of MAC operations, across the DW layers. This variation can be categorized as intra-layer-type heterogeneity. Figure 13(c) shows the number of operations in a DW layer for all the DW layers of the evaluated CNNs. The first few DW layers have much larger operation counts than the rest. Since the B_SEML design uses a single DW engine to execute all the DW layers, the PEs assigned to that engine must be proportional to the average DW layer operation count. This results in a mismatch between the workload and the engine resources in the initial layers, making them a bottleneck.

In summary, the considerable heterogeneity among the initial layers explains why processing them using dedicated engines is crucial. As FiBHA processes these layers using dedicated engines that suit their parallelism patterns and their workloads (e.g., Table 1), the inefficiencies are minimized. Note that these dedicated engines are resource inexpensive (e.g., Table 1). This is because their execution time could be relaxed to fill all the time needed by the SEML part to execute the rest of the layers. The double buffering between the two parts of FiBHA (Section 4.3) allows the few layers of the SESL to take as much time as the majority of the layers executed by the SEML part take without forming a bottleneck.

In the second scenario, MobileNetV1 and MobileNetV2 on ZC706 and KCU105 (Figure 13(a)), only FiBHA instances worked successfully, but there are no results for B_SEML instances because their FMs of a single layer did not fit in the on-chip BRAM (FiBHA reduces the FM buffers requirements considerably, Figure 7). This gives FiBHA a qualitative advantage over the SEML-based

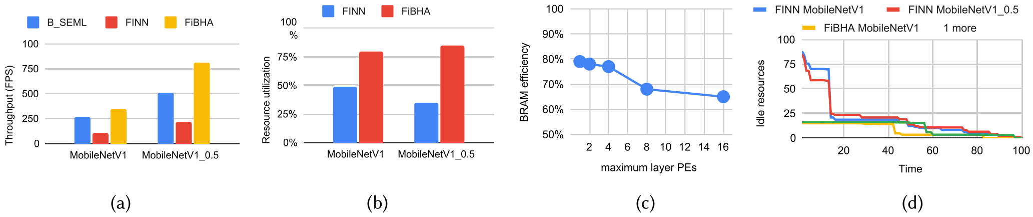


Fig. 14. (a) B_SEML, FINN, and FiBHA throughput normalized to B_SEML throughput. (b) Maximum LUT utilization achieved by FiBHA and FINN on ZCU102. (c) FINN BRAM efficiency: the ratio of the actually used to the allocated BRAM blocks. (d) FINN vs. FiBHA weighted computing resource idleness.

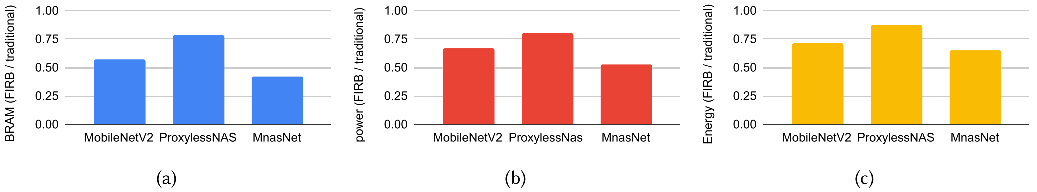
accelerator. Note that in such scenarios, an SEML-based design would work if the FMs are stored off-chip. However, we do not consider such implementation since it results in a huge amount of access to the external memory, which would lead to a considerable energy overhead. To give an example, an SEML-based implementation of MobileNetV2 that stores FMs off-chip requires roughly 13.4 million DRAM accesses compared to *zero* when using FiBHA in these scenarios.

In the third scenario, B_SEML , unlike FiBHA, did not scale because the available on-chip BRAM imposed a scaling bottleneck. Hence, FiBHA provides > 2*x* throughput improvement in this scenario. KCU015 has roughly twice the number of PEs compared to ZC706, but both of them have roughly similar on-chip memory (Table 2). When scaling an engine by adding more PEs, more elements need to be accessed in one cycle to keep up with the parallelism and not form a bottleneck. To enable accessing more elements in one cycle, the data has to be partitioned on a sufficient number of independently accessible banks or blocks. This results in some form of fragmentation where even though the FMs are slightly smaller than the available BRAM, they cannot be stored in a way that satisfies the PEs' scaling requirements. Since FiBHA reduces the required FM buffering, it increases the free space, which enables partitioning the data without being bottlenecked by the on-chip memory. As a result, FiBHA scaled up to utilize the additional PEs available on KCU015, while B_SEML did not.

6.2 FiBHA vs. SESL (FINN)

Figure 14(a) shows the throughput of FiBHA, B_SEML, and FINN measured in **frames per second (FPS)**. Both FiBHA and B_SEML outperform FINN under the same resource budget. FiBHA provides up to 4*x* throughput improvement compared to FINN. For all three approaches the resource utilization is maximized for the given budget. As such, neither of the three approaches can scale further to achieve higher throughput given this budget. FINN's relatively low throughput is caused by the general SESL scalability issue and by FINN's pipeline balancing technique. We explain these issues in detail in the following paragraphs.

SESL-based accelerators do not scale well in general. Figure 14(b) shows the maximum compute resource utilization to the overall resources available on the used FPGA for both FINN and FiBHANote. When comparing with FINN, LUTs are used as the computing resources rather than DSPs; this is because all layers but the first use 4-bit quantization, for which FINN offers LUT-based optimized implementation. FINN utilized less than half of the LUTs, which are used as compute resources or PEs. This is because, as with any pipelined SESL accelerator, scaling the performance requires scaling all the engines (Figure 3). If only one is not scaled, it forms a bottleneck, forcing the rest to stay idle until it finishes its layer. Moreover, to maintain the utilization and improve the throughput, the engines should be scaled by a common factor of the available parallelism in the layers [4]. Note that in the MobileNetV1_0.5 case (Figure 14(b)), the resource utilization is 35%. So ideally, FINN should scale by a factor of 2× and utilize roughly 70% of the resources, but it did not.



(a) (b) (c)

Fig. 15. (a) Required on-chip buffers of FIRB normalized to a traditional tiled inter-layer pipeline. (b) Power consumption of FIRB buffers normalized to a traditional tiled inter-layer pipeline. (c) Energy consumption per inference of FIRB normalized to a traditional tiled inter-layer pipeline.

This is because the on-chip memory forms a bottleneck. Figure 14(c) shows the BRAM efficiency reported by FINN, which is the ratio of the actually used memory to the allocated BRAM blocks. BRAM efficiency drops when scaling up as a result of fragmentation similar to SEML (Section 6.1). This is because when an engine is scaled up, it needs to access more weights and FM elements per cycle to keep its PEs busy. Since each BRAM has a limited number of ports, accessing more elements per cycle requires partitioning these elements on more BRAMs to scale the bandwidth. This partitioning reduces BRAM efficiency. With more scaling, the result is using more BRAM blocks with fewer elements in each block, which at some point becomes a scaling bottleneck. But unlike SEML, where the reason beyond fragmentation is storing the large FMs of the initial layers on-chip, in FINN the reason is storing the large weights of the last layers on-chip.

The FINN scaling algorithm starts by setting the parallelism to 1 for all engines; they refer to this step as minimal dataflow compute implementation. As the name suggests, this is the minimum number of PEs that could be used. After that initial assignment, and based on the compute requirements of all the layers, the FINN scaling algorithm systematically gives more resources to the most pressing bottlenecks. When giving more resources to these bottlenecks, which are the engines with the highest latencies, other engines become the new bottlenecks. Hence, this process is repeated by identifying the new bottleneck engines and giving them more resources until the available resources are exhausted. Even though this gives the best balance for the available PE budget, the achieved balance could be far from optimal. This is because an optimal balance requires that all the engines have equivalent ratios between their layers' computational workloads and the PEs that they are assigned. Figure 14(d) depicts the impact of FINN assignment on idleness, which is a direct result of not having optimal balance among the engines. The x-axis shows time, and the y-axis shows weighted idleness, which is the product of the idleness of an engine (the difference between its execution time and the slowest engine execution time), and the ratio of that engine's resources to the overall used resources. The figure shows that up to $\approx 60\% - 75\%$ of the resources are idle for more than 10% of the time. Note that this ratio is out of FINN's utilized resources, not the overall resources.

FiBHA does not suffer from any of these issues; FiBHA's pipeline is much shorter than a pure SESL (e.g., FINN), and this has two advantages. First, scaling FiBHA's SESL part is much less expensive. Second, FiBHA's SESL engines are easier to balance, and hence it experiences lower resource idleness than a pure SESL.

6.3 FIRB vs. Traditional Inter-layer Pipelining

Figure 15 quantifies the advantage of using FIRB pipelining compared to traditional inter-layer pipelining to implement the SESL part of FiBHA. Figure 15(a) shows the amount of on-chip buffers (BRAM) required by FIRB compared to the traditional pipelining. FIRB saves up to 55% of the SESL buffering. This reduction is a result of the fine-grained pipelining inside the inverted bottlenecks (Section 4.2). Figure 15(a) shows the power consumption of the BRAM required by FIRB compared

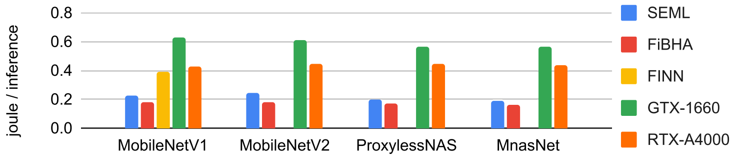


Fig. 16. Joules per inference of FiBHA, B_SEML, GTX-1660, and RTX-A4000. FiBHA and B_SEML measurements are of ZCU012.

to the traditional pipelining. It can be seen that the reduction in power is almost proportional to the reduction in the needed BRAM count. ProxylessNas has the lowest memory and power reduction; this is due to the fact that its initial inverted bottlenecks have a smaller expansion factor (Sections 2 and 4.2) than the others. This means that the buffers inside the bottlenecks, which are eliminated by FIRB, are small compared to the other models. Figure 15(c) shows the energy consumption of FIRB compared to the traditional pipelining. FIRB constantly improves energy consumption due to power savings.

6.4 Energy Efficiency

Figure 16 shows the energy consumption of FiBHA, B_SEML, FINN, and two GPUs (NVIDIA GTX-1660 and RTX-A4000) using four heterogeneous CNNs. Note that in our evaluation we use a batch size of 1 as usually done when evaluating inference. For latency-critical applications, e.g., inference in vision applications for autonomous driving, a batch size of 1 could be the only practical option. However, for applications that are not latency critical, GPUs could exploit larger batch sizes to achieve better performance.

FiBHA constantly improves the energy efficiency over B_SEML. This reduction is caused by capturing heterogeneity, which is translated to better resource utilization (Section 6.1). In addition to that, FiBHA reduces the FMs' buffer sizes considerably. Smaller buffers contain fewer BRAM blocks leading to power savings (Section 6.3). Both FiBHA and B_SEML outperform FINN; FINN consumes more energy per inference due to its relatively high execution time (Section 6.2). The three dedicated accelerators have better energy efficiency than the GPUs. The RTX-A4000 is more energy-efficient than GTX-1660. This can be explained by the fact that RTX-A4000 has third-generation tensor cores, while GTX-1660 does not have any [36].

MobileNetV2, ProxylessNAS, and MnasNet have similar energy consumption levels. This is expected as they have similar operation counts (Table 3) and the same building block—the Inverted Residual Bottleneck. Even though MobileNetV1 has roughly double the number of operations compared to the rest, it has similar energy consumption. However, note that MobileNetV2, ProxylessNAS, and Mnasnet have $2\%-5\%$ higher accuracy compared with MobileNetV1 [6, 45, 51], meaning they offer higher accuracy without increasing energy consumption.

Table 4 shows the energy efficiency of several accelerators targeting compact and heterogeneous CNNs. Many of these accelerators are highly dedicated and target one model, which is often a variant of MobileNets. Different approaches use different optimizations and different quantization levels and are deployed on different boards. Hence, this table is meant to show efficiency in the context of prior art rather than presenting a direct comparison. Note that FiBHA, as we discuss in Sections 4.3 and 4.4, is not a specific implementation, and any state-of-the-art SEML accelerator could be used as the SEML part of FiBHA. For example, [62] is likely to outperform our FiBHA implementation of MobileNetV2 using the same board, as it is designed specifically and highly optimized only for MobileNetV2. But [62] could then be used instead of B_SEML to generate a new and more optimized FiBHA instance targeting MobileNetV2 following the steps described in Section 4.4. [1] also outperforms FiBHA using the same board,

Table 4. Energy Efficiency of Prior Art Accelerators and FiBHA Targeting Heterogeneous CNNs (Lower Is Better)

Paper	Platform	Quantization (Bit-width, Type)	Energy Efficiency (Joules/Inference)			
			MobileNetV1	MobileNetV2	ProxylessNAS	MnasNet
[62]	XC7V690t	8, integer	–	**0.113**	–	–
[1]	XCZU7EG	channel-wise, integer	–	0.125	–	–
[56]	Arria10 GX1150	–	0.79	–	–	–
[31]	XC7Z045	16, fixed-point	18.4	–	–	–
[46]	XCZU7EV	8, integer	–	0.514	–	–
[60]	XC7Z020	16, fixed-point	–	0.198	–	–
[38]	XCZU7EV	dynamic, fixed-point	-	4.85	–	–
[41]	XC7V690T	16-bit fixed-point	1.23	–	–	–
[65]	XC7VX690T	–	–	–	–	0.73
FiBHA	XCZU7EG	8, integer	**0.179**	0.177	**0.168**	**0.16**

since it uses channel-wise quantization with extremely low bit-widths. Note that [1] as well could be fed to SplitCNN 4.4 and be used to derive a more optimized FiBHA instance.

7 RELATED WORK

To improve the resource efficiency of DNN processing, at both the training and inference phases, researchers have been proposing model-specific accelerators. FPGAs are heavily used in developing such custom accelerators owing to their reconfigurability [4, 16, 30, 34, 42, 43, 48, 53].

MobileNets are the most commonly targeted resource-efficient CNNs by model-specific SEML accelerators. This is because the modules or the building blocks of MobileNets have been used to design most of the state-of-the-art efficient models [6, 51, 52, 64]. Bai et al. [2] proposed an accelerator that processes all the CNN layers using a single engine. But they designed a configurable adder tree that supports both DW and PW convolutions. In [31, 32, 48, 56, 58], the authors proposed to use two separate convolution engines, one for PW and another for DW. These two engines are pipelined to maintain high throughput. Su et al. [48] also utilized two separate engines to process the convolutional layers. Moreover, they optimized the model using structured kernel-level pruning and quantization to reduce further the model parameter count producing **Redundancy-Reduced MobileNet (RR-MobileNet)**. In RR-MobileNet, the model parameters and intermediate results are stored on-chip. Wu et al. [58] proposed a channel augmentation technique that improves the efficiency of the first layer of MobileNets. Yan et al. [62] optimized the bottleneck module, and they proposed dedicated hardware to reduce the module data transmission latency.

SESL accelerators were proposed to capture CNNs' heterogeneity to the fullest [3, 4, 11, 30, 44, 53, 54]. In an SESL architecture, all the model layers are mapped to the hardware, each layer having its own dedicated engine. These engines are pipelined and work simultaneously. The SDF model of computation [28] is usually used to map CNNs to the hardware [53, 54]. To reduce the latency and buffering overheads among the pipelined engines, Gao et al. [15] have proposed an ALLO dataflow that enables forwarding the intermediate results in a relatively timely manner between pairs of adjacent layers.

Pure SESL accelerators where all the layers are mapped to dedicated engines are resource demanding and unscalable [34, 43]. One FPGA-based workaround is based on partitioning the CNN model along its depth and mapping each partition to a bitstream [54]. This incurs a considerable overhead, which grows as the models get deeper. Another proposal is to use aggressively quantized, even binarized CNNs [4, 53], where fewer bits are needed to represent the weights, the activations, and the partial sums. As a result, computations are performed using simpler hardware, and the bandwidth requirements are reduced. However, the accuracy drop caused by aggressive quantization may not be acceptable. Another bottleneck of a pure SESL accelerator is the need to feed weights and inputs to all the simultaneously running engines, which increases the bandwidth requirements. Li et al. [30] proposed a mechanism to alleviate the bandwidth bottleneck that helps when a CNN is shallow, e.g., AlexNet [23].

Unlike the related work, FiBHA's hybrid design captures as much heterogeneity as the PEs' budget allows without sacrificing resource efficiency [39]. Hence, it achieves better scalability and higher throughput compared to the alternatives. In this extension, we propose, implement, and evaluate FIRB, a fine-grained and memory-light SESL building block. We analyze and demonstrate the memory efficiency and scalability of FiBHA using multiple FPGA boards. Finally, we demonstrate FiBHA's energy efficiency compared to both custom accelerators and GPUs.

8 CONCLUSION AND FUTURE WORK

In this article, we presented FiBHA, a hybrid architecture that captures CNNs' heterogeneity within a fixed resource budget. We proposed FIRB, a memory-light and fine-grained module that improves the efficiency of pipelining the Inverted Residual Bottlenecks that are common in recent compact CNNs. We implemented instances of FiBHA and demonstrated its advantages over the two commonly used alternative custom and model-aware accelerator families (SEML and SESL) using different FPGAs representing different resource budgets. Our experiments demonstrated that capturing CNN heterogeneity translates to better resource utilization and higher throughput. For a fixed hardware budget, FiBHA achieved up to $2.5x$ and $4x$ of the throughput achieved by the state-of-the-art accelerators of the two categories. Moreover, FIRB modules reduce the required memory by up to 54% and energy requirements by up to 35% compared to traditional inter-layer pipelining. Finally, we compare and analyze the energy efficiency when processing such CNNs using FiBHA, an SEML-based accelerator, and generic GPUs.

We plan to extend our cost model and heuristics by designing multi-objective optimizations that address the tradeoffs between performance, memory requirements, and energy efficiency. In our future work, we plan to extend the targeted models from compact to general CNNs.

REFERENCES

[1] Mohammadreza Baharani, Ushma Sunil, Kaustubh Manohar, Steven Furgurson, and Hamed Tabkhi. 2021. Deepdive: An integrative algorithm/architecture co-design for deep separable convolutional neural networks. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. 247–252.

[2] Lin Bai, Yiming Zhao, and Xinming Huang. 2018. A CNN accelerator on FPGA using depthwise separable convolution. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 10 (2018), 1415–1419.

[3] Chaim Baskin, Natan Liss, Evgenii Zheltonozhskii, Alex M. Bronstein, and Avi Mendelson. 2018. Streaming architecture for large-scale quantized neural networks on an FPGA-based dataflow platform. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'18)*. IEEE, 162–169.

[4] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'Brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[5] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. 2021. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT'21)*. IEEE, 159–172.

[6] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. 578–594.

[8] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 367–379.

[9] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1251–1258.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 248–255.

[11] Zhen Dong, Yizhao Gao, Qijing Huang, John Wawrzynek, Hayden K. H. So, and Kurt Keutzer. 2021. Hao: Hardware-aware neural architecture optimization for efficient inference. In *2021 IEEE 29th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'21).* IEEE, 50–59.

[12] Naveen Kumar Dumpala, Shivukumar B. Patil, Daniel Holcomb, and Russell Tessier. 2017. Energy efficient loop unrolling for low-cost FPGAs. In *2017 IEEE 25th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'17).* IEEE, 117–120.

[13] Jingbo Gao, Yu Qian, Yihan Hu, Xitian Fan, Wai-Shing Luk, Wei Cao, and Lingli Wang. 2021. LETA: A lightweight exchangeable-track accelerator for Efficientnet based on FPGA. In *2021 International Conference on Field-programmable Technology (ICFPT'21).* IEEE, 1–9.

[14] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3D memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems.* 751–764.

[15] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems.* 807–820.

[16] Cong Hao and Deming Chen. 2018. Deep neural network model and FPGA accelerator co-design: Opportunities and challenges. In *2018 14th IEEE International Conference on Solid-state and Integrated Circuit Technology (ICSICT'18).* IEEE, 1–4.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861.*

[19] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint arXiv:1602.07360.*

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2704–2713.

[21] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture.* 1–12.

[22] Okan Kopuklu, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. 2019. Resource efficient 3D convolutional neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops.* 0–0.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25 (2012).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 60, 6 (2017), 84–90.

[25] Sanmukh R. Kuppannagari, Ren Chen, Andrea Sanny, Shreyas G. Singapura, Geoffrey Phi C. Tran, Shijie Zhou, Yusong Hu, Stephen P. Crago, and Viktor K. Prasanna. 2014. Energy performance of FPGAs on perfect suite kernels. In *2014 IEEE High Performance Extreme Computing Conference (HPEC'14).* IEEE, 1–6.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[27] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. 2010. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 253–256.

[28] Edward A. Lee and David G. Messerschmitt. 1987. Synchronous data flow. *Proceedings of the IEEE* 75, 9 (1987), 1235–1245.

[29] Guoqing Li, Jingwei Zhang, Meng Zhang, Ruixia Wu, Xinye Cao, and Wenzhao Liu. 2022. Efficient depthwise separable convolution accelerator for classification and UAV object detection. *Neurocomputing* 490 (2022), 1–16.

[30] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. 2016. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. IEEE, 1–9.

[31] Jiawen Liao, Liangwei Cai, Yuan Xu, and Minya He. 2019. Design of accelerator for MobileNet convolutional neural network based on FPGA. In *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC'19)*, Vol. 1. IEEE, 1392–1396.

[32] Bing Liu, Danyin Zou, Lei Feng, Shou Feng, Ping Fu, and Junbao Li. 2019. An FPGA-based CNN accelerator integrating depthwise separable convolution. *Electronics* 8, 3 (2019), 281.

[33] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 45–54.

[34] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. 2016. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. IEEE, 1–8.

[35] Thierry Moreau, Tianqi Chen, and Luis Ceze. 2018. Leveraging the VTA-TVM hardware-software stack for FPGA acceleration of 8-bit resnet-18 inference. In *Proceedings of the 1st on Reproducible Quality-efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*. 1.

[36] Nvidia. 2022. NVIDIA RTX A4000. https://www.nvidia.com/content/dam/en-zz/Solutions/gtcs21/rtx-a4000/nvidia-rtx-a4000-datasheet.pdf datasheet.

[37] Alessandro Pappalardo. 2021. *Xilinx/Brevitas*. https://doi.org/10.5281/zenodo.3333552

[38] Ignacio Pérez and Miguel Figueroa. 2021. A heterogeneous hardware accelerator for image classification in embedded systems. *Sensors* 21, 8 (2021), 2637.

[39] Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso. 2022. FiBHA: Fixed budget hybrid CNN accelerator. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'22)*. IEEE, 180–190.

[40] Fareed Qararyah, Mohamed Wahib, Doğa Dikbayır, Mehmet Esat Belviranli, and Didem Unat. 2021. A computational-graph partitioning method for training memory-constrained DNNs. *Parallel Computing* 104 (2021), 102792.

[41] Wenqiang Qin, Zhongcheng Wu, Jun Zhang, and Fang Li. 2022. Design and optimization of MobileNet neural network acceleration system based on FPGA. In *2022 3rd International Conference on Electronics, Communications and Information Technology (CECIT'22)*. IEEE, 7–12.

[42] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 26–35.

[43] Atul Rahman, Jongeun Lee, and Kiyoung Choi. 2016. Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE'16)*. IEEE, 1393–1398.

[44] Youki Sada, Masayuki Shimoda, Akira Jinguji, and Hiroki Nakahara. 2019. A dataflow pipelining architecture for tile segmentation with a sparse MobileNet on an FPGA. In *2019 International Conference on Field-Programmable Technology (ICFPT'19)*. IEEE, 267–270.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[46] Nazariy K. Shaydyuk and Eugene B. John. 2020. FPGA implementation of MobileNeTv2 CNN model using semi-streaming architecture for low power inference applications. In *2020 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom'20)*. IEEE, 160–167.

[47] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[48] Jiang Su, Julian Faraone, Junyi Liu, Yiren Zhao, David B. Thomas, Philip H. W. Leong, and Peter Y. K. Cheung. 2018. Redundancy-reduced MobileNet acceleration on reconfigurable logic for Imagenet classification. In *International Symposium on Applied Reconfigurable Computing*. Springer, 16–28.

[49] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE* 105, 12 (2017), 2295–2329.

[50] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1–9.

[51] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2820–2828.

[52] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.

[53] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-programmable Gate Arrays.* 65–74.

[54] Stylianos I. Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'16).* IEEE, 40–47.

[55] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture.* 13–26.

[56] Liu Wei and Lv Peng. 2021. An efficient OpenCL-based FPGA accelerator for MobileNet. *Journal of Physics: Conference Series* 1883 (2021), 012086.

[57] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017.* 1–6.

[58] Di Wu, Yu Zhang, Xijie Jia, Lu Tian, Tianping Li, Lingzhi Sui, Dongliang Xie, and Yi Shan. 2019. A high-performance CNN processor based on FPGA for MobileNets. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL'19).* IEEE, 136–143.

[59] Jiale Xiao, Yonghao Chen, and Tao Su. 2021. A MobileNet accelerator with high processing-element-efficiency on FPGA. In *2021 China Semiconductor Technology International Conference (CSTIC'21).* IEEE, 1–3.

[60] Xiaofei Xie, Guodong Zhao, Wei Wei, and Wei Huang. 2022. MobileNetV2 accelerator for power and speed balanced embedded applications. In *2022 IEEE 2nd International Conference on Data Science and Computer Application (ICDSCA'22).* IEEE, 134–139.

[61] Rui Xu, Sheng Ma, Yaohua Wang, and Yang Guo. 2021. HESA: Heterogeneous systolic array architecture for compact CNNs hardware accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE'21).* IEEE, 657–662.

[62] Shun Yan, Zhengyan Liu, Yun Wang, Chenglong Zeng, Qiang Liu, Bowen Cheng, and Ray C. C. Cheung. 2021. An FPGA-based MobileNet accelerator considering network structure characteristics. In *2021 31st International Conference on Field-programmable Logic and Applications (FPL'21).* IEEE, 17–23.

[63] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146.*

[64] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 6848–6856.

[65] Tong Zhao, Lufeng Qiao, Qinghua Chen, Qingsong Zhang, and Na Li. 2020. A hardware accelerator based on neural network for object detection. *Journal of Physics: Conference Series* 1486 (2020), 022045.

Received 27 February 2023; revised 13 October 2023; accepted 21 December 2023