Thesis for The Degree of Licentiate of Engineering

Efficient Processing of Compact and Heterogeneous Deep Neural Networks

Fareed Mohammad Qararyah

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2024

Efficient Processing of Compact and Heterogeneous Deep Neural Networks

Fareed Mohammad Qararyah

Technical Report No 193L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2024.

Abstract

The unprecedented success of Deep Learning (DL) algorithms, or Deep Neural Networks (DNNs), is driving the trend toward deploying them in a variety of environments including ones with tight resources and time constraints. This has led to the emergence of *compact DNNs*. On the one hand, compact DNNs have fewer operations and lower resource requirements, which makes them the right choice for time-critical and energy-constrained applications. On the other hand, they pose new challenges for the deep learning accelerator and library design. First, these DNNs are composed of a set of operators with varying computational requirements. This makes them more *heterogeneous*. Secondly, they contain novel operators with computational requirements and bottlenecks that differ from those of the operators in traditional DNNs. These characteristics render the generic, *accelerator architectures* and *library routines* inefficient and necessitate custom designs considering these DNNs characteristics.

The constant evolution of state-of-the-art DNNs and their use in domains that have constantly changing algorithms and standards motivate deploying them on flexible hardware. That is hardware that could be programmed or reconfigured to support such variations. Moreover, the massive parallelism present in these DNNs suggests that such hardware should support parallelism. Field Programmable Gate Arrays (FPGAs), and General-Purpose Graphics Processing Units (GPGPUs) are two widely used devices that offer flexibility and support parallelism. This thesis presents hardware and software designs, *i.e.* accelerators and library routines, that enable efficient processing of the compact DNNs and their constituting operators on FPGAs, and GPGPUs.

The first contribution of the thesis is *Fixed Budget Hybrid CNN Accelerator (FiBHA)*. FiBHA is a hybrid architecture that combines both dedicated and reusable processing engines in a way that enables striking a balance between capturing DNN model heterogeneity and being resource-aware. The second contribution is proposing *Fused Convolutional Modules (FCMs)*, a set of GPU kernels fusing various combinations of two core operators used in compact vision DNNs, including convolutional neural networks (CNNs) and vision transformers (ViTs). These operations are depthwise (DW) and pointwise (PW) convolutions. FCMs alleviate these operators' performance bottlenecks leading to low-latency and energy-efficient execution.

FiBHA improves the throughput by up to 4x and 2.5x compared to the prior art. It achieves up to 2x improvement in resource utilization. Moreover, it improves the energy efficiency by up to 28%. FCMs achieve up to 3.7x speedup over standard DL libraries layer-by-layer implementations, and up to 1.8x speedup in end-to-end implementations compared to a state-of-the-art DL compiler. Moreover, FCMs-based implementations consume down to 34% of the energy per inference compared to those of a DL compiler.

Keywords:

Deep Neural Networks (DNNs), Deep Learning Accelerators, FPGA, GPU, Inter-Layer Pipelining, Layer Fusion

Acknowledgment

I would like first to thank my supervisor, Professor Pedro Petersen Moura Trancoso, for giving me the opportunity to pursue my studies and providing continuous guidance and support. And for having an open mind, and being understanding whenever there is a difference in opinions. I would also like to thank my co-supervisor, Muhammad Waqar Azhar, for his guidance and advice, feedback, assistance, and patience.

I would also like to thank my examiner Ioannis Sourdis for his support. I am also thankful for my friends and colleagues Mateo, Stavroula, Mo, Alessio, Xu, Sonia, Konstantinos, Miquel, Monica, Arne, and others who have been helpful and created a friendly environment.

Finally, I would like to give special thanks to my family for their unwavering support and continuous encouragement.

This work would not have been possible without the research grants from the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957197. This work was also partially funded by the Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project.

List of Publications

Appended publications

This thesis is based on the following publications:

I Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso "FiBHA: Fixed Budget Hybrid CNN Accelerator"
2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). (pp. 180-190). IEEE, 2022.

II Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso "An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs"
ACM Trans. Archit. Code Optim. 21, 2, Article 25 (June 2024), 26 pages.

III Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso "FCMs: Fusing Depthwise and Pointwise Convolutions for Efficient Inference on GPUs"
Under review for ICS 2024: International Conference on Supercomputing.

The papers will be referred to in the thesis using their Roman numerals.

Contents

Chapter 1

Introduction

Deep Learning (DL) algorithms' unprecedented effectiveness in a variety of domains comes usually at the cost of being resource demanding [1, 2]. However, the desire to apply DL in time-critical and resource-constrained environments has been driving the effort to design resource-efficient or compact Deep Neural Networks (DNNs). Compact DNNs have fewer weights and perform fewer operations which enables faster and less resource-demanding DL [1, 3–6]. An example from the computer vision field is the XCeption convolutional neural network (CNN) that has an accuracy that surpasses ResNet-152's [7] on the Imagenet dataset, despite being roughly three times smaller [3].

In recent years we have seen an explosion of application-specific DL accelerators [8]. These application-specific accelerators process certain classes of DNNs with low latency, high throughput, and high energy efficiency. However, designing application-specific hardware should not be the only strategy to achieve efficient DL for two reasons. The first reason is that some application domains have constantly changing algorithms or standards, or a combination of them, like telecommunication and autonomous cars [9]. Hence, these domains rely heavily on flexible accelerators, *i.e.* reconfigurable or programmable. When targeting such domains, DNNs should run efficiently on these flexible accelerators. The second reason is that DL is expected to continue to play a significant role in the future, with continuously evolving state-of-the-art DL models. This justifies the ongoing research to efficiently process DNNs on accelerators that are relatively future-proof. Flexibility to support a variety of applications is a strong predictor of being future-proof [10].

Field Programmable Gate Arrays (FPGAs), and General-Purpose Graphics Processing Units (GPGPUs) are relatively flexible and have been heavily used to develop state-of-the-art DL accelerators and libraries [11–25]. Moreover, GPUs have played a key role in the resurgence of DL [26], and are the most widely-supported accelerators by various DL frameworks [27–30]. This thesis presents hardware and software designs and techniques to process compact DNNs efficiently on FPGAs and GPUs.

Most of the proposed FPGA-based model-specific accelerators targeting compact CNNs fall into one of two categories. The first category's 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 accelerators (SEML) [22, 31–33]. But, CNN layers of the same type have various arithmetic intensities, input and filter shapes and sizes, and reuse patterns [34, 35]. Consequently, when there is a dedicated engine per layer type, this engine has to be optimized for the average case within that layer type. The second category's accelerators have a *dedicated engine per layer* [21, 36–38]. We refer to such accelerators as single-engine single-layer accelerators (SESL). They are known as streaming, or synchronous dataflow (SDF), accelerators in the literature. Such accelerators are resource-demanding and are challenging to scale for deep models [37, 39, 40]. The first contribution of this thesis is a hybrid architecture that is composed of SESL and SEML parts, namely *Fixed Budget Hybrid CNN Accelerator (FiBHA)* [41, 42]. In FiBHA, the FPGA resources and the CNN layers are partitioned among these two parts. FiBHA is more dedicated, or CNN model-aware, compared to SEML which leads to higher efficiency. At the same time, FiBHA is more resource-aware and easier to scale compared to SESL accelerators.

Standard GPU DL libraries provide sub-optimal performance in the case of compact

DNNs. Many state-of-the-art compact vision DNNs including CNNs and vision transformers (ViTs) utilize Depthwise (DW) and pointwise (PW) convolutions [1, 3–6, 43]. DW and PW convolutions have fewer operations but more memory access compared to standard convolutions. In other words, they are more memory-bound than standard convolution. Hence, to optimize their performance, the focus must be on reducing their off-chip memory access. This makes the algorithms commonly used on GPUs like general matrix multiplication (GEMM), Winograd, and Fast Fourier Transform (FFT) [44–48] inefficient. These algorithms optimize or reduce the computations at the cost of more memory accesses and higher bandwidth requirements [49, 50]. The second contribution of the thesis is presenting *Fused Convolutional Modules (FCMs)* a set of novel fused GPU kernels of DW and PW convolutions. FCMs fuse pairs of PW and DW layers processing the intermediate results at fine granularity and eliminating the need to communicate them to GPU's global memory. This reduction in global memory access alleviates the DW and PW performance bottleneck leading to low-latency and energy-efficient processing of these layers.

Extensive experiments show that the proposed work enables efficient processing of compact DNNs on FPGAs and GPUs. FiBHA improves the throughput by up to 4x and 2.5x compared to alternative representative SEML and SESL accelerators. FiBHA improves FPGA's resource utilization by up to 2x. It reduces the intermediate results memory requirements considerably, allowing better scaling and avoiding off-chip communication. Furthermore, it improves energy efficiency by up to 28%. FCMs achieve up to 3.7x speedup over standard DL libraries', cuDNN [47], layer-by-layer implementations, and up to 1.8x speedup in end-to-end implementations compared to the state-of-the-art TVM [51]. Moreover, FCM-based implementations consume down to 34% of the energy per inference compared TVM based execution.

1.1 Background

1.1.1 Convolutional Neural Networks (CNNs)

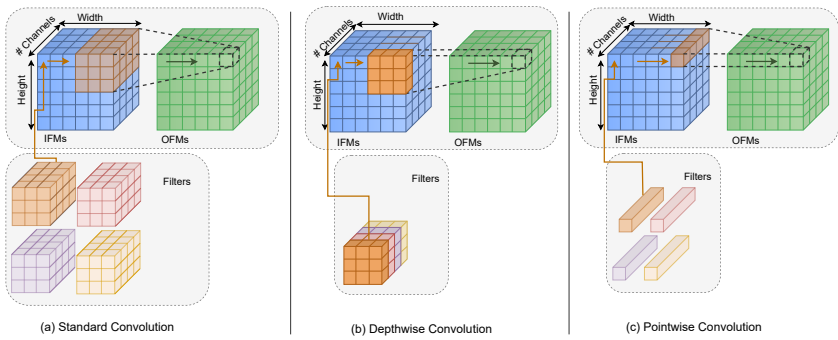


Figure 1.1: Standard, Depthwise, and Pointwise convolution.

Convolutional neural networks (CNNs) are feedforward Deep learning models designed to capture features in 1D signals like language, 2D signals like images; or 3D like video or volumetric images [52]. A CNN consists of a set of stacked layers performing feature extraction and classification [53]. The primary layers in a CNN, both in terms of functionality and computational intensity, are the convolutional layers [54]. Figure 1.1 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 Feature Maps (**FMs**), or activations, to refer to both IFMs and OFMs.

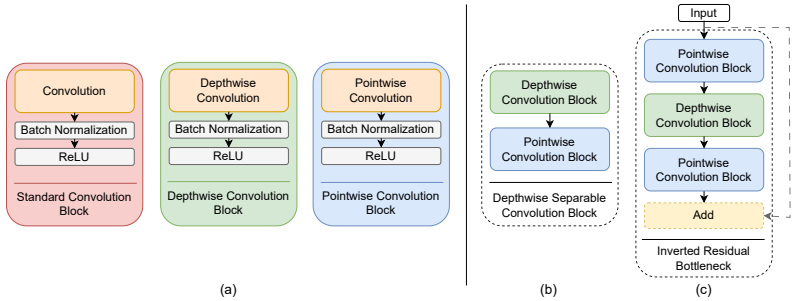


Figure 1.2: Depthwise Separable Convolution and Inverted Residual Bottlenecks.

1.1.2 Compact CNNs and ViTs

Compact CNNs are also referred to as heterogeneous, Resource-efficient, or edge CNNs in the literature [34, 55]; hence we use these terms interchangeably throughout this paper. These CNNs are designed to balance the accuracy-efficiency trade-off. They either improve the accuracy without increasing the model weights and computations [3] or reduce the model weights and computations considerably at the cost of a negligible loss in accuracy [1, 4, 56–58]. The Depthwise separable convolution and inverted residual and linear bottlenecks are two predominant building blocks of resource-efficient CNNs [1, 3, 4, 56–58].

The Depthwise Separable Convolution (DSC) is a form of convolution that is based on decoupling the spatial and the cross-channel correlations in the feature maps [3]. This is achieved by replacing the standard convolution with two operators: (a) Depthwise convolution (**DW**) and (b) Pointwise convolution (**PW**). As Figure 1.2(b) shows, a Depthwise Separable Convolution is composed of Depthwise followed by Pointwise convolution. Figure 1.1(b) shows that the Depthwise convolution applies a filter to each feature map and sums each feature map results individually (no across-FMs summation). This is followed by a Pointwise convolution with 1×1 filters. Pointwise convolution, as shown in 1.1(c), applies a filter and sums the results across different feature maps but not within a single feature map. Depthwise Separable Convolution is a more efficient way to use the model parameters compared to the standard convolution [3, 59].

The inverted residual with linear bottlenecks is a module designed to significantly reduce the CNN model weights and operations while maintaining the accuracy [56]. As shown in Figure 1.2(c) this module combines three convolutional layers. The first layer is a 1×1 Pointwise convolution that *expands* the IFMs by increasing their depth. The second is a Depthwise convolution that processes the expanded FMs. The third is another 1×1 Pointwise 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 1.2. The reduction in weights and computations that the inverted residual with bottlenecks module permits is a result of performing heavy operators like the Pointwise convolution on a relatively low-dimensional representation, that is, the unexpanded or the squeezed FMs. And a light operation like Depthwise 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.

Transformer models are based on a self-attention mechanism that learns the relationships between elements of a sequence [60]. In vision transformers (ViTs), *self-attention* allows modeling contextual information of the full image and long-range dependencies both in space and time [61]. In this paper, we focus on convolutional ViTs that combine self-attention with convolutions [5, 6, 43].

1.1.3 GPU architecture and programming model overview

A GPU architecture consists of a scalable array of streaming multiprocessors (SMs) [62]. An SM is a Single-Instruction-Multiple-Thread (SIMT) architecture that runs groups of 32 parallel threads called *warps* in a lockstep fashion. A CUDA kernel is processed by a *grid* of threads which is composed of a set of *thread blocks*. Threads in a block run on the same SM. GPU has a memory hierarchy of multiple levels with different access constraints. Each thread has private local registers. Each SM has a low-latency L1 cache, a variable-sized portion of that cache can be configured to serve as programmer-managed *shared memory*. The shared memory is visible to all threads in a block and has the same lifetime of the block. The next level of memory is L2 cache which is shared across all threads of the entire CUDA kernel. The last level is a DRAM which is accessed when the required data is not in any of the above levels.

1.2 Problem Statements

1.2.1 Problem 1

Compact CNNs contain layers or operations of varying arithmetic intensities, reuse patterns, filters, and input/output shapes and sizes which makes them heterogeneous. This heterogeneity is present at two levels: (1) heterogeneity among layers of the same type, or intra-layer-type heterogeneity; and (2) heterogeneity among layers of different types, or inter-layer-type heterogeneity. On the one hand, ignoring this heterogeneity by processing these CNNs using a single computing engine, or a single engine per layer type, leads to sub-optimal performance. On the other hand, capturing the heterogeneity to the fullest by designing an engine per layer is resource-demanding, not scalable, and suffers from high latency. The challenge lies in striking a balance between being CNN model-aware and being resource-aware while maintaining a low latency. That is, capturing the heterogeneity in the model, given the commonly available or a reasonable resource budget.

The search for an architecture that balances being model-aware and resource-aware is done under a set of assumptions. First, the targeted hardware is an FPGA as it offers the required flexibility to generate model-specific architecture. Secondly, to support a wide range of compact CNN models that vary considerably in size, the design should not assume that the CNN model fits on-chip. Thirdly, all the engines must be mapped to the device without reconfiguration. This is because the average latency of processing many compact CNNs on common FPGAs, using - for example - a single-engine design, is in milliseconds, making reconfiguration overhead prohibitive. Finally, the design targets inference where low latency is usually crucial, hence the objective is to optimize throughput and latency simultaneously.

Problem statement 1: Given a heterogeneous CNN and FPGA resources, how to map the CNN to the FPGA in a way that captures the heterogeneity, to improve dynamic resource utilization and reduce the running time overheads.

1.2.2 Problem 2

Convolutions are core operators of computer vision algorithms including the DL based ones. The effort to optimize the convolutions in DNNs, to reduce their resource requirements, and to use them to design more compact DNNs, has resulted in separating them into depthwise and pointwise convolutions. Depthwise and pointwise convolutions have fewer operations and parameters (weights) compared to standard ones. However, separating a standard convolution into depthwise and pointwise increases the overall size of intermediate results. In other words, the reduction ratio in the number of operations is higher than that of the weights and intermediate results. This means these convolutions have lower arithmetic intensity and are more often memory-bound than standard ones.

Operator or layer fusion is an optimization where the fused operators process the intermediate values while being in the small and low-latency memories in the hierarchy. This avoids access to the main memory that has orders of magnitude higher latency. Hence fusing depthwise and pointwise convolutions should lead to alleviating their memory-access bottlenecks. Fusing these operators on GPUs poses a set of challenges. GPUs' low-latency memories have limited lifespan as well as capacity. But fusion increases the working set

size and requires the fused operators' weights to be used over a longer lifespan. Hence, an efficient fusion should optimize the trade-off between these limitations and requirements. Moreover, when fusing layers there is a large space of fusion parameters to explore to arrive at configurations that optimize the memory accesses while maintaining an acceptable level of compute unit utilization. This requires designing a simple and efficient cost model to estimate memory accesses as a function of the fusion configurations. Such a model should enable identifying whether fusion results in reducing memory access or not and which fusion parameters minimize the memory accesses.

Problem statement 2: Given a sequence of pointwise and depthwise convolutions and GPU architecture specifications, how to identify and implement the convolution-fusions that optimize the execution efficiency by mitigating memory access bottlenecks while maintaining compute resource utilization.

1.3 Thesis Contributions

This thesis presents techniques to improve the efficiency of processing compact DNNs and their operators on widely used and flexible hardware, *i.e.* GPUs and FPGAs. More specifically, the thesis tackles the challenges presented in Section 1.2. This section discusses the thesis contributions and how it addresses these challenges concisely.

The first contribution is an FPGA-based **hybrid CNN accelerator** for Compact and heterogeneous CNNs, namely Fixed Budget Hybrid CNN Accelerator (FiBHA). FiBHA represents a middle ground between the resource-aware and scalable accelerators and the highly dedicated streaming accelerators. The thesis focuses on two accelerator categories that target heterogeneous CNNs. The first category is referred to as single-engine multiple-layer (SEML) accelerators, which use a single engine to process multiple layers of the same type. The second is single-engine single-layer (SESL) accelerators, which process each layer using its dedicated engine. The key to designing the hybrid accelerator is to identify the layers in a CNN that benefit the most from being processed using dedicated engines and then design as many dedicated engines as the resource budget allows. Analyzing a set of representative compact CNNs has shown that their initial layers have varying input and output shapes and sizes and reuse opportunities. The rest of the layers are more homogeneous. This trend motivates a design that processes the initial layers using per-layer dedicated engines, and the rest using a minimum number of reusable engines.

Once the CNN layers are categorized, the challenge is to map them to the parts of the hybrid accelerator. To tackle this challenge, a heuristic to split CNN layers (**SplitCNN**) is proposed. SplitCNN splits the resources, more specifically the processing elements (PEs) among the CNN layers. Since FiBHA is an alternative to the SEML and SESL accelerators, it should improve over them given a practical resource budget. Moreover, as the hybrid accelerator is composed of SEML and SESL parts, SplitCNN uses an existing design of such accelerators as a starting point. Note that the proposed accelerator targets inference, where latency is crucial. Hence, a batch size of 1 is used to guarantee that the throughput and latency are optimized simultaneously. Exploring the mentioned two accelerator families shows that the SEML accelerators provide superior performance compared to SESL when using a batch size of 1. Hence, the chosen starting point to derive a FiBHA instance is a SEML baseline. The baseline accelerator is progressively transformed into a hybrid one by incorporating dedicated engines. SplitCNN allocates resources between the original and the newly added engines, considering the workloads of the CNN layers assigned to these engines, to ensure that their execution times are balanced. It then estimates the throughput using a batch size of 1. The process of adding engines, distributing the available PEs, and estimating the performance is repeated as long as the performance is improving. The process terminates with the hybrid accelerator that is estimated to deliver the best performance using the given PEs budget.

The compute engines in a SESL accelerator are pipelined to maintain a high throughput and resource utilization. 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. Meanwhile, the majority of the state-of-the-art compact CNNs are composed of inverted residual bottlenecks (Section 1.1.2). The structure of these bottlenecks gives room for reducing the mentioned pipeline overhead. **Fused Inverted Residual Bottlenecks (FIRB)** module is proposed to achieve this goal. FIRB applies fine-grained pipelining within

the inverted residual bottleneck module, resulting in a considerable reduction in the required buffers. This approach processes the intermediate results of the Depthwise (DW) layer within the bottleneck while they are in registers. This method reduces the need for the majority of buffers that would otherwise be used for communication between this layer and its neighboring layers. FIRB-based pipelines have less area, power consumption, and latency compared to traditional ones.

The second contribution of this thesis is a set of **fused convolutional kernels, or fused convolutional modules (FCMs)**, that improve the efficiency of compact DNNs' core operators on GPUs. These operators are pointwise (PW) and depthwise (DW) convolutions which are used in many state-of-the-art computer vision DNNs including CNNs and ViTs. In FCMs, unlike the traditional layer-by-layer (LBL) implementation, the intermediate values between the fused layers are processed while being in the GPU's L1 cache/shared memory avoiding frequent access to the global memory that has orders of magnitude higher latency. Fusing convolutional layers involves loading their weights simultaneously and accessing them for the duration required to compute their dependencies across both layers by each Streaming Multiprocessor (SM). This puts pressure on the limited per SM private L1 Cache/shared memory(Section 1.1.3) and could make fusion either infeasible or result in more memory accesses than the LBL implementation.

To evaluate the feasibility and trade-offs of fusing DW and PW convolutions, a simple and fast cost model that estimates the global memory accesses of convolution on a GPU, namely **FuseEstimator**, is proposed. FuseEstimator takes as inputs GPU specifications, mainly the number of SMs, L1 size, and the portion that could be configured as shared memory. It also takes the sequence of layers and their specifications. These specifications include weights and FMs tensors dimensions. FuseEstimator has two components, layer-by-layer LBL GM access estimator and FCMs GM access estimator. FuseEstimator does a first pass over the layer sequence and estimates their minimum GM access using the LBL GM access estimator. Then it examines the possible fusions and evaluates their GM access using the FCMs GM access estimator. Based on the LBL and FCM estimates, FuseEstimator outputs: (1) which layers are to be fused and which are not, (2) which FCMs to use, and (3) the tiling that minimizes the GM access in each case.

Chapter 2

Summary of the Papers

2.1 Summary of Paper I

Compact CNNs are heterogeneous workloads
A CNN consists of a set of stacked layers performing feature extraction and classification [53]. The core layers in a CNN, the convolutional layers, have various inputs and filter shapes and sizes, reuse patterns, and arithmetic intensities [34, 55]. We designate these variations as intra-type-heterogeneity, as they exist among layers of the same type. On top of that, compact CNNs are composed of modules containing different types of convolutions including depthwise separable convolution (DSC) and the inverted bottlenecks [3, 59, 63]. This introduces another form of heterogeneity, which we refer to as inter-layer-type heterogeneity.

Prior art model-specific accelerators
To capture compact CNNs' heterogeneity, and hence improve the performance and efficiency, custom or model-specific accelerators have been proposed. 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 [54, 64].
- **Single-Engine Multiple-Layer (SEML):** accelerators in which all the layers of a certain type are executed using the same engine [22, 31, 32, 59, 64–66].
- **Single-Engine Single-Layer (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 [21, 36–38].
- **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 [39, 67].

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 DNNs or the training phase. Figure 2.1 shows examples of both SEML and SESL mapping approaches. 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 5 convolutional layers to its engines and uses a SEML mapping technique for the rest of the layers.

Fixed budget hybrid CNN accelerator (FiBHA)
Neither SEML nor SESL accelerators address the CNN and resource awareness trade-off. While SEML accelerators do not capture intra-layer-type heterogeneity, pure SESL accelerators are resource-demanding and do not scale well. In this paper, we propose FiBHA, a *fixed budget hybrid architecture* that combines both SESL and SEML accelerators in a way that captures more heterogeneity than SEML accelerators can while being more resource-efficient than pure SESL. The hybrid mapping of FiBHA is shown in Figure 2.1. The SESL part of FiBHA computes the initial layers of a CNN, and the SEML part computes the rest. We also propose a heuristic "Split a CNN" (SplitCNN) that derives a FiBHA instance under a fixed resource budget and splits the CNN layers and the resources between that

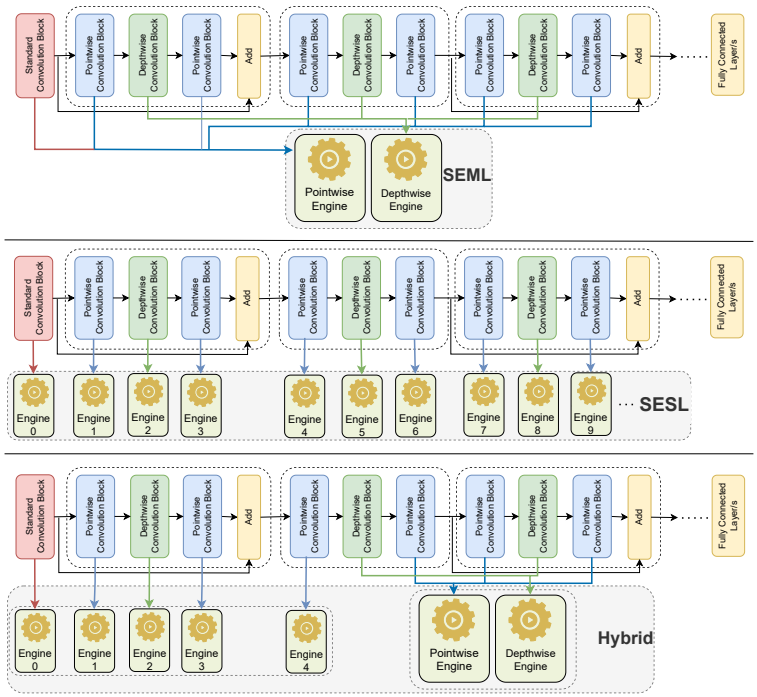


Figure 2.1: High-level overview of the SEML, SESL, and Hybrid accelerators. The figure focuses on the mapping from the convolutional layers of a CNN to the compute engines of the accelerators, the internals of the engines, their connectivity, and the memory system are omitted for simplicity.

instance SESL and SEML parts. We evaluated FiBHA instances using HLS implementation targeting an FPGA (Xilinx ZCU-102) using state-of-the-art heterogeneous CNNs [4, 56, 58]. FiBHA achieves 1.7x and 4.1x of the throughput achieved by state-of-the-art, SEML [31] and SESL [21] accelerator, respectively, under the same hardware budget. In this paper, we focus on accelerating inference [50].

The **contributions** of this paper are as follows:

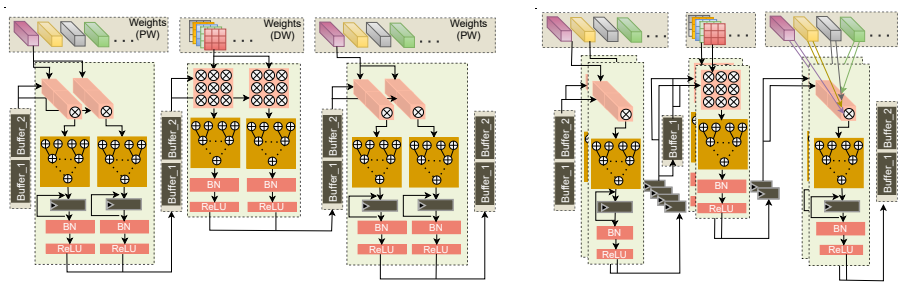
- We propose FiBHA (Fixed budget hybrid accelerator), a novel hybrid architecture composed of a SESL part and a SEML part, each computing a part of a CNN model.
- We propose SplitCNN, a heuristic that derives a FiBHA instance given a fixed resource budget and splits a CNN and the computational resources between its SESL and SEML parts in a way that maximizes the throughput.
- We evaluate FiBHA comparing its performance to pure SESL and SEML accelerators. FiBHA instances achieve throughput of 1.7x and 4.1x compared to these two accelerators, respectively.

2.2 Summary of Paper II

This paper is an extension of **Paper I**, in which we conduct more extensive experiments using the hybrid architecture presented there (FiBHA). We propose optimized FiBHA engines, named Fused Inverted Residual Bottlenecks(FIRB). FIRB reduces the required memory by up to 54%, and energy requirements by up to 35% compared to traditional inter-layer pipelining. We analyze the impact of FiBHA on the buffering requirements and show how it eliminates the intermediate results in off-chip communication when the on-chip memory is relatively small. We synthesize and use FiBHA to deploy end-to-end CNN implementations and compare and study the impact of the hybrid architecture on energy efficiency.

Fused Inverted Residual Bottlenecks(FIRB)

In a traditional implementation of pipelined SESL accelerators, a tile of the feature maps (FMs) of each layer is stored on-chip and double-buffered for concurrent access [11, 40]. This introduces a non-negligible overhead. This overhead scales with the pipeline length as more double buffers must be added. Hence, we introduce the Fused Inverted Residual Bottlenecks (FIRB) module. FIRB applies fine-grained pipelining within the inverted residual bottleneck modules that are used in most compact CNNs, resulting in a considerable reduction in the required buffer sizes.



(a) (b)

Figure 2.2: **(a)**Traditional inter-layer pipelining-based implementation of an Inverted Residual Bottleneck module. **(b)** FIRB-based implementation of an Inverted Residual Bottleneck module.

Figure 2.2a and 2.2b show two implementations of an Inverted Residual Bottleneck module. The implementation in figure 2.2a is a variant of a traditional inter-layer pipelining. The main advantage of FIRB is eliminating the need for three out of the four internal buffers in the inverted residual bottleneck module which reduces latency and energy consumption of these modules.

Minimizing FMs memory requirements

The memory consumption of Deep Learning algorithms represents a major bottleneck both in training and inference [67, 68]. In some state-of-the-art accelerators, on-chip buffers consume up to 70% to 87% of the chip area [54, 67]. As a result, reducing memory requirements is a crucial part of designing an efficient accelerator. 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. By implementing the CNN layers that have large FMs using SESL, FiBHA reduces the buffering requirements by up to 75% compared to pure SEML or monolithic accelerators.

The **contributions** of this paper are as follows:

- We propose FIRB, a fine-grained and memory-light pipeline building block and evaluate its impact on implementing a SESL 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 comparing its performance to a dedicated accelerator on three evaluation boards representing various resource budgets in the compute continuum, namely ZC706, KCU105, and ZCU102.
- We evaluate FIBHA's energy efficiency, comparing it to a SESL accelerator, a SEML accelerator, and two GPUs using a set of heterogeneous CNNs.

2.3 Summary of Paper III

Due to their relatively high accuracy-per-parameter and accuracy-per-operation, depthwise (DW) and pointwise (PW) convolutions are increasingly replacing standard convolutions leading to state-of-the-art accuracy results using smaller models [1, 3–6, 43]. However, separating a standard convolution into DW and PW increases the size of the intermediate results which results in lower arithmetic intensity and makes memory access a bottleneck. Hence, optimizing DW and PW efficiency must consider reducing memory access.

Layer-fusion is an optimization that reduces off-chip memory accesses considerably, compared to the traditional layer-by-layer (LBL) approach. The intermediate results of the fused layers are processed immediately instead of storing and loading them from the off-chip memory [18, 69–71]. On GPUs, the prior art on layer fusion focuses on fusing a single convolutional layer with element-wise layers or fusing multiple non-convolutional layers [51, 72–74].

In this paper, we propose *Fused Convolutional Modules (FCMs)* a set of novel fused GPU kernels of DW and PW convolutions. FCMs improve latency and energy efficiency by reducing global memory access. A *Fused Convolutional Module (FCM)* combines up to 6 layers, two convolutional layers, and the normalization and activation layers following each of them. We present three main types of FCMs, namely **DWPW**, **PWDW**, and **PWPW**. These combinations cover the most widely-used DNN modules that combine DW and PW convolutions, *i.e. depthwise separable convolutions (DSC)* and the *inverted residual with linear bottlenecks* [3, 59, 63]. The communication among the fused layer utilizes the GPU L1/shared memory. The limited size and lifespan of the shared memory pose a set of trade-offs when fusing convolutional layers that need to be evaluated to decide when fusion is advantageous.

To evaluate fusion tradeoffs and decide when to use FCMs and when not to, we propose FuseEstimator. FuseEstimator is a cost model that estimates the global memory accesses of the DW and PW kernels on GPUs. FuseEstimator takes as inputs GPU specifications, mainly the number of SMs, L1 size, and the portion that could be configured as shared memory. It also takes the sequence of layers and their specifications. These specifications include weights and FMs tensors dimensions. This sequence could be a model topology, we support generating model typologies from Tensorflow. FuseEstimator has two components, layer-by-layer *LBL GM access estimator* and *FCMs GM access estimator*. FuseEstimator does a first pass over the layer sequence and estimates their minimum GM access using

the LBL GM access estimator. Then it examines the possible fusions and evaluates their GM access using the FCMs GM access estimator. Based on the LBL and FCM estimates, FuseEstimator outputs: (1) which layers are to be fused and which are not, (2) which FCMs to use, and (3) the tiling that minimizes the GM access in each case.

Our FCM kernels achieve up to 1.8x speedup over a custom layer-by-layer (LBL) implementation and up to 3.7x over the best cuDNN implementations using representative CNNs and ViTs. End-to-end implementations of four CNNs using the proposed kernels achieve up to 1.8x speedup compared to TVM implementations and consume as little as 34% of the energy consumed by TVM-optimized models.

The **contributions** of this paper are as follows:

- We propose a set of novel fused convolutional modules (FCMs), which are GPU kernels comprising PW and DW convolutions. Our kernels serve as low-latency and energy-efficient building blocks of DNNs that use these convolutions.
- We propose *FuseEstimator*, a cost model that estimates global memory accesses of LBL and FCM implementations given a target GPU architecture. *FuseEstimator* decides which layer to fuse and which not to, and if to be fused which FCMs to use.
- We evaluate the proposed FCMs comparing them to both custom and standard DL library-based (cuDNN) implementations using representative CNNs and ViTs on multiple GPUs. We also compare end-to-end implementations of the CNNs built with modules to TVM-optimized models.

Chapter 3

Conclusion and Future Work

Compact DNNs, which have fewer weights and perform fewer operations are being increasingly used. While they offer faster and less resource-demanding training and inference, these DNNs are composed of a variety of novel operators with characteristics that differ from those of the operators of traditional DNNs. As a result, processing these DNNs using generic, accelerator architectures and library routines offers sub-optimal performance. This motivates the designing of custom hardware and software considering these DNNs characteristics. Compact DNNs, as DNNs in general, are continuously evolving and are used in domains with constantly changing standards and constraints. This motivates optimizing their performance on devices that could be programmed or reconfigured to support such variations as Field Programmable Gate Arrays (FPGAs), and General-Purpose Graphics Processing Units (GPGPUs).

This thesis presents both *i.e.* accelerators and library routines, that enable efficient processing of the compact DNNs and their operators on FPGAs, and GPGPUs. The first presented contribution in the thesis is *Fixed Budget Hybrid CNN Accelerator (FiBHA)*. FiBHA is a hybrid architecture that targets the balance between capturing DNN model heterogeneity and being resource-aware. The second contribution is *Fused Convolutional Modules (FCMs)*, a set of GPU kernels that fuse different combinations of depthwise (DW) and pointwise (PW) convolutions, two core operators utilized in compact vision DNNs, including convolutional neural networks (CNNs) and vision transformers (ViTs). FCMs mitigate these operators' memory access bottlenecks leading to low-latency and energy-efficient inference.

There are several research directions to explore following up on the work done so far. One direction is to widen the scope of the presented hybrid architecture to support non-compact CNNs. FiBHA, support is limited to compact CNNs. However, the presented analysis shows the potential impact of such architecture on the required buffering and data movement of CNNs in general. This requires an exhaustive analysis of the prior art and exploring if such hybrid architecture could offer a more optimal CNN-to-FPGA mapping in some scenarios. A second direction is exploring the possibility of reusing modules of the presented architecture to augment the state-of-the-art and offer higher efficiency in processing specific operations. A third direction is integrating the optimized Fused Convolutional Modules (FCMs) into a state-of-the-art DL compiler like TVM. FCMs offer additional design points for such compilers to explore. They could either be used as a backend when they offer a performance advantage or used as starting points and auto-tuned to achieve even higher efficiency. Another direction is exploring other DNNs including graph neural networks (GNNs) and large language models. This could lead either to generalizing some of the presented modules to support a wider variety of applications, or designing new modules and ending up having a library of efficient DL modules for programmable and reconfigurable devices.

Bibliography

[1] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[3] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[5] D. R. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le, "Primer: Searching for efficient transformers for language modeling," *arXiv preprint arXiv:2109.08668*, 2021.

[6] K. Yuan, S. Guo, Z. Liu, A. Zhou, F. Yu, and W. Wu, "Incorporating convolution designs into visual transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 579–588.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] S. R. Sukumar, J. A. Balma, C. Xu, and S. Serebryakov, "Survival of the fittest amidst the cambrian explosion of processor architectures for artificial intelligence," in *2021 IEEE/ACM Programming Environments for Heterogeneous Computing (PEHC)*. IEEE, 2021, pp. 34–43.

[9] O. Mencer, D. Allison, E. Blatt, M. Cummings, M. J. Flynn, J. Harris, C. Hewitt, Q. Jacobson, M. Lavasani, M. Moazami *et al.*, "The history, status, and future of fpgas: Hitting a nerve with field-programmable gate arrays," *Queue*, vol. 18, no. 3, pp. 71–82, 2020.

[10] M. Horowitz, "Reconfigurable future," *ACM Queue vol1*, no. 7, 2003.

[11] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535–547, 2017.

[12] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.

[13] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao, "Flowgnn: A dataflow architecture for real-time workload-agnostic graph neural network inference," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1099–1112.

[14] M. Véstias, "Processing systems for deep learning inference on edge devices," *Convergence of Artificial Intelligence and the Internet of Things*, pp. 213–240, 2020.

[15] A. C. Elster and T. A. Haugdahl, "Nvidia hopper gpu and grace cpu highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.

[16] J. Hao, P. Subedi, L. Ramaswamy, and I. K. Kim, "Reaching for the sky: Maximizing deep learning inference throughput on edge devices with ai multi-tenancy," *ACM Transactions on Internet Technology*, vol. 23, no. 1, pp. 1–33, 2023.

[17] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.

[18] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[19] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "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*, 2017, pp. 45–54.

[20] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[21] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[22] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 136–143.

[23] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu *et al.*, "Compressing deep neural networks on fpgas to binary and ternary precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, p. 015001, 2020.

[24] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo *et al.*, "Fast convolutional neural networks on fpgas with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045015, 2021.

[25] Y.-C. Lin, B. Zhang, and V. Prasanna, "Hp-gnn: Generating high throughput gnn training implementation on cpu-fpga heterogeneous platform," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022, pp. 123–133.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[27] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron *et al.*, "Theano: Deep learning on gpus with python," in *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3, no. 0. Citeseer, 2011.

[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[31] J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. Leong, and P. Y. Cheung, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2018, pp. 16–28.

[32] J. Liao, L. Cai, Y. Xu, and M. He, "Design of accelerator for mobilenet convolutional neural network based on fpga," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1. IEEE, 2019, pp. 1392–1396.

[33] G. Li, J. Zhang, M. Zhang, R. Wu, X. Cao, and W. Liu, "Efficient depthwise separable convolution accelerator for classification and uav object detection," *Neurocomputing*, vol. 490, pp. 1–16, 2022.

[34] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "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)*. IEEE, 2021, pp. 159–172.

[35] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 13–26.

[36] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.

[37] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–9.

[38] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 40–47.

[39] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th international conference on field programmable logic and applications (FPL)*. IEEE, 2016, pp. 1–8.

[40] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 807–820.

[41] F. Qararyah, M. W. Azhar, and P. Trancoso, "Fibha: Fixed budget hybrid cnn accelerator," in *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2022, pp. 180–190.

[42] ——, "An efficient hybrid deep learning accelerator for compact and heterogeneous cnns," *ACM Transactions on Architecture and Code Optimization*.

[43] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "Cvt: Introducing convolutions to vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 22–31.

[44] V. Podlozhnyuk, "Fft-based 2d convolution," *NVIDIA white paper*, vol. 32, no. 1, 2007.

[45] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International conference on artificial neural networks*. Springer, 2014, pp. 281–290.

[46] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4013–4021.

[47] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[48] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[49] G. Lu, W. Zhang, and Z. Wang, "Optimizing depthwise separable convolution operations on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 70–87, 2021.

[50] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[51] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[52] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[53] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 2010, pp. 253–256.

[54] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[55] R. Xu, S. Ma, Y. Wang, and Y. Guo, "Hesa: Heterogeneous systolic array architecture for compact cnns hardware accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 657–662.

[56] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[57] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[58] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[59] J. Xiao, Y. Chen, and T. Su, "A mobilenet accelerator with high processing-element-efficiency on fpga," in *2021 China Semiconductor Technology International Conference (CSTIC)*. IEEE, 2021, pp. 1–3.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[61] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[62] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?" *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[63] X. Xie, G. Zhao, W. Wei, and W. Huang, "Mobilenetv2 accelerator for power and speed balanced embedded applications," in *2022 IEEE 2nd International Conference on Data Science and Computer Application (ICDSCA)*. IEEE, 2022, pp. 134–139.

[64] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[65] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, 2019.

[66] S. Yan, Z. Liu, Y. Wang, C. Zeng, Q. Liu, B. Cheng, and R. C. Cheung, "An fpga-based mobilenet accelerator considering network structure characteristics," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 17–23.

[67] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[68] F. Qararyah, M. Wahib, D. Dikbayır, M. E. Belviranli, and D. Unat, "A computational-graph partitioning method for training memory-constrained dnns," *Parallel Computing*, vol. 104, p. 102792, 2021.

[69] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 27–42.

[70] X. Cai, Y. Wang, and L. Zhang, "Optimus: towards optimal layer-fusion on deep learning processors," in *Proceedings of the 22nd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2021, pp. 67–79.

[71] H.-J. Jeong, J. Yeo, C. Bahk, and J. Park, "Pin or fuse? exploiting scratchpad memory to reduce off-chip data transfer in dnn accelerators," in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, 2023, pp. 224–235.

[72] L. Jia, Y. Liang, X. Li, L. Lu, and S. Yan, "Enabling efficient fast convolution algorithms on gpus via megakernels," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 986–997, 2020.

[73] C. Li, Y. Yang, M. Feng, S. Chakradhar, and H. Zhou, "Optimizing memory efficiency for deep convolutional neural networks on gpus," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 633–644.

[74] S. Dong, X. Gong, Y. Sun, T. Baruah, and D. Kaeli, "Characterizing the microarchitectural implications of a convolutional neural network (cnn) execution on gpus," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 96–106.