2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 978-1-6654-6272-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/MICRO56248.2022.00094

DeepBurning-SEG: Generating DNN Accelerators of Segment-Grained Pipeline Architecture

Xuyi Cai[1,2,4], Ying Wang ✉[1,3,5] , Xiaohan Ma[1,2,4], Yinhe Han[1,3,5], Lei Zhang[1,2]

[1] *SKLP, Institute of Computing Technology, CAS, Beijing, China*
[2] *Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, CAS, Beijing, China*
[3] *CICS, Institute of Computing Technology, CAS, Beijing, China*
[4] *University of Chinese Academy of Sciences, Beijing, China*
[5] *Zhejiang Laboratory, Zhejiang, China*
Email: {caixuyi20b, wangying2009, maxiaohan, yinhes, zlei}@ict.ac.cn

***Abstract*—The growing complexity and diversity of *deep neural network (DNN)* applications have inspired intensive research on specialized DNN accelerators and also the design automation frameworks. Previous specialized NN acceleratos roughly fall into two categories of implementation, either the no-pipelined architecture that relies on a generic *processing unit (PU)* to sequentially execute the DNN layers in a layer-wise way, or the fully-pipelined architecture that dedicates interconnected customized PUs to the corresponding DNN layers in the model. Thus, such designs often suffer from either the resource under-utilization issue faced by no-pipelined accelerators or the resource scalability problem brought by the over-deep pipeline designs.**

In this work, we propose a novel class of design solution for DNN acceleration, *segment-grained pipeline architecture (SPA)*. In the SPA accelerator, the targeted workload of DNN models will be divided into many segments and each segment will be sequentially executed on the shared interconnected PUs in a pipeline manner, so that they will benefit from both the efficiency of pipelined execution and also the flexibility of sharing PUs across different model layers. Particularly, we found that the efficiency of the implemented SPA accelerator significantly depends on the segmentation strategies of the models and the hardware resources assignment policy for PUs. Therefore, we introduce an automated design framework, AutoSeg, that includes a parameterized SPA accelerator template and a co-design engine that will generate the efficient model segmentation solution and hardware pipeline design parameters for the acceleration workload. Experimental results show that the SPA solutions generated by the AutoSeg framework achieve 1.2× to 6.3× speedup when compared to ASIC-based general DNN processors, and the FPGA designs implemented by AutoSeg also achieve as high as 3.4× DSP efficiency and 3.6× throughput improvement.

I. Introduction

Nowadays, DNNs have been considered the most popular solution in the fields of machine learning [23], [25], [31], [61], [62], [64]. The performance gained by DNNs is accompanied by the growth of model complexity, depth and scale, which leads to a sheer rise of computation and memory pressure in the infrastructures. This trend inspired extensive researches on DNN accelerator designs [1], [5], [7], [17], [28], [50], [59], [83] and automated accelerator customization frameworks [29], [36], [67], [75], [77], [84] to improve the inference performance and energy efficiency under limited hardware resources.

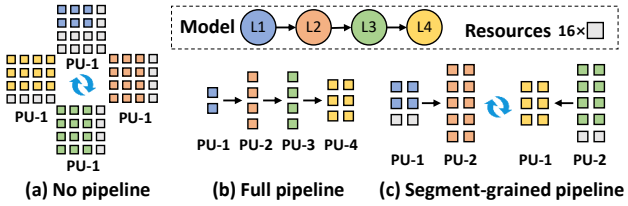


Figure 1. Specialized DNN accelerator designs. In the no-pipeline implementation, a unified PU execute the model layers one by one. In the full pipeline implementation, multiple PUs fully pipeline the entire model. In the segment-grained pipeline implementation, multiple PUs pipeline the segments one by one.

According to prior works, state-of-the-art specialized DNN accelerators and design frameworks could be roughly categorized into two popular types of implementation solutions. The first implementation methodology uses one unified processor unit (PU) to execute the DNN layers sequentially without pipelining as shown in Fig. 1(a), known as *no-pipeline implementation* [5], [7], [8], [17], [47]. When customizing towards particular workloads, the specific design-point (e.g., PE, buffer, and dataflow) of the unified PU is jointly optimized for all layers of the workload [30], [36], [54], [58], [67], [75], [78], since all the resources are available to theoretically every individual layer. However, it is difficult to optimize a unified PU for DNN models with diverse layers that exhibit vastly different characteristics in terms of the data moving, layer dimensions and etc. Besides, there is a high cost of data exchange between layers due to the lack of the pipeline that moves results directly from producer to consumer PUs, decreasing *computation to communication (CTC) ratio* and deteriorating memory

✉ Corresponding Author

978-1-6654-6272-3/22/$31.00 ©2022 IEEE
DOI 10.1109/MICRO56248.2022.00094

bandwidth bottlenecks. As a result, for each individual layer, the unified PU will lead to under-utilization of resource.

To alleviate the problem of resource utilization, an effective way is to map and pipeline the entire DNN over the full pipeline accelerator [1], [46], [69], [74], [83], [84] as shown in Fig. 1(b). The intermediate *feature maps (fmaps)* between layers are forwarded from producer-PU directly to the PUs that consume them, resulting in a higher CTC ratio and improving performance. In addition, full pipelines are much more efficient since they have a customized computation component for each layer. However, the separated customized PUs cannot flexibly share resources as the no-pipeline architectures. Thus, resources allocated to each layer can be very scarce, and sometimes it is even impossible to assign the limited resources for very deep neural network models. More importantly, the generated customized hardware can hardly be utilized across the models when design generality is desired.

Due to the inherent shortcomings of both implementation methodologies, designing accelerators for the given DNN models remains a non-trivial task. To address the challenges, we aim to design a novel segment-grained pipeline architecture that can take advantage of the high CTC ratio of hardware pipeline, while avoiding resource shortage and inflexibility caused by full pipeline architecture. As shown in Fig. 1(c), we partition the models into segments that do not contain too many layers so that each segment can be mapped onto a practical shallow pipeline architecture. In this design methodology, a flexible pipeline accelerator design shared by all network segments is required, so that a deep network can be sequentially mapped onto the accelerator in a segment-wise approach. In addition, the PUs can be assigned in a flexible way to better match PUs and the layers in each segment. For example, as shown in Fig. 1(c), L1 and L2 in the first segment are assigned to PU-1 and PU-2 respectively, while L3 and L4 in the second segment will be scheduled onto the PU-2 and PU-1 for better utilization after the first segment is completed.

In this paper, we propose a segment-grained pipeline design paradigm to divide the layers of the targeted DNN models flexibly into segments, and design a shallow **SPA** (**S**egment-Grained **P**ipeline **A**rchitecture) to achieve a high degree of specialization for all model segments, which enables high resource utilization and also scalability over very deep models. When customizing a specialized SPA accelerator for the target workloads, we need proper resource allocation for each PUs of the hardware pipeline, and an efficient model segmentation strategy that can partition various models into the segments with high utilization. Therefore, we propose AutoSeg, an automated *hardware/software (HW/SW)* co-design framework to realize optimized SPA accelerator for target DNN workloads. The main contributions are:

1) We identify the opportunities for the segment-grained pipeline of DNN models, and propose SPA, a novel segment-shared architecture that can switch between model segments at runtime.

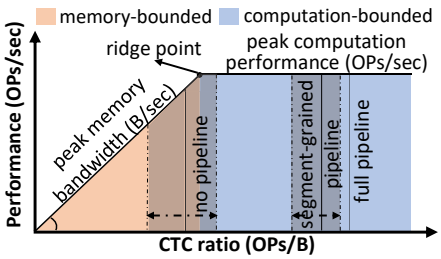


Figure 2. Roofline model.

2) We present a highly parameterized hardware template to implement the SPA. This template consists of two main components: reconfigurable inter-PU fabrics for the varied data communication between segments; and dataflow-hybrid PUs for different dataflow preferences of layers.

3) We introduce an end-to-end automation framework, AutoSeg, to customize the SPA accelerator for the given DNN model directly from its high level DNN description.

4) We propose a SPA HW/SW co-design engine which includes: the model segmentation unit based on *mixed-integer programming (MIP)* to obtain the optimized segmentation strategy; and the SPA design generation unit using a heuristic resource allocation algorithm to allocate hardware resource for each PU.

5) We evaluate the AutoSeg-generated designs for different application scenarios across a set of DNN benchmarks. The results show that the SPA accelerators generated by the AutoSeg framework achieve 1.2× to 6.3× performance speedup when compared to general DNN processors such as Eyeriss, NVDLA and EdgeTPU of the same resource budget, and the FPGA designs implemented by AutoSeg also achieve as high as 3.4× DSP efficiency and 3.6× throughput improvement over state-of-the-art accelerator solutions on both low-power and large-scale FPGA devices.

II. Motivation

In this section, we firstly characterize the computation and communication pattern of the DNN models mapped onto no-pipeline and full-pipeline implementations. The full-pipeline architecture possesses a very high CTC ratio, which is an indicator of resource efficiency and performance, but it lacks of programmability and often has a very rigid and high resource budget. Through the analysis of the CTC ratio and segment similarity in models, we identify the acceleration opportunities for segment-grained pipelining and discuss the challenges involved in designing such a SPA accelerator.

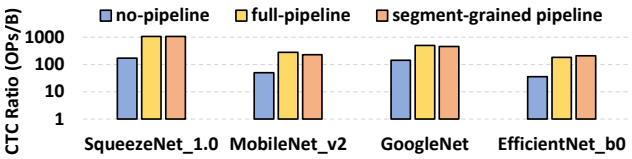


Figure 3. The CTC ratio of models running at the no-pipeline, full-pipeline and segment-grained pipeline implementations. In this example, the segment-grained pipeline evenly divides SqueezeNet, MobileNetV2, GoogleNet, and EfficientNet_b0 into 6-layer, 3-layer, 6-layer, and 5-layer segments respectively.

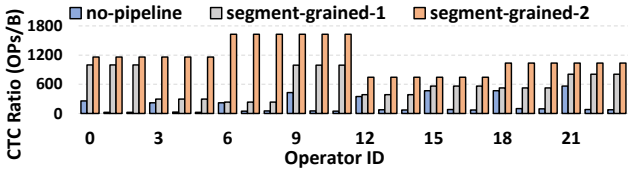


Figure 4. The impact of model segmentation on the CTC ratio of SqueezeNet. Segment-grained-1 and segment-grained-2 simply divided the model into 3-layer and 6-layer segements, respectively. Further increasing the CTC ratio is possible, but will require more advanced model segmentation techniques.

A. Analysis of Current Accelerator Architectures: A CTC Perspective

Computation and communication are the two principal factors in accelerator performance. In [73], the roofline model is introduced and it reveals the upper bound of system performance through the given peak computation performance and memory bandwidth, as visualized in Fig. 2. The CTC ratio in the roofline model measures the number of MAC operations (in OPs) per memory access (in Bytes) of an application on a specific hardware. For an accelerator, the peak computation performance forms the horizontal roof in the roofline, while the peak memory bandwidth in bytes per second is the slope of the diagonal roof, since we have (OPs/sec)/(OPs/B) = B/sec. The X-coordinate of the ridge point (where the diagonal and horizontal roof meet) gives the minimum CTC ratio required to achieve the maximum performance in the hardware. Without reaching a sufficient CTC ratio, a program becomes memory-bounded, so the computational resources are underutilized and performance is limited by the diagonal roof.

In the no-pipeline implementation [5], [7], [8], [17], [47], DNN models are executed as a sequence of layers, where each layer loads its inputs from the memory to on-chip buffers, performs computation, and stores the output back to memory. This creates unnecessary memory traffics of intermediate fmaps. Therefore, as shown in Fig. 3 and Fig. 4, many layers have low CTC ratios and their performance are memory-bounded. For example, NVDLA [47] has 5.6 TOPs/sec of int8 computation peak performance and 20 GB/s memory bandwidth. In another word, the minimum CTC ratio required to deliver the maximum performance

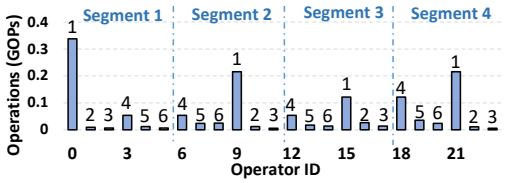


Figure 5. The number of operations (multiply and add) in SqueezeNet. With proper layer assignments, the operational distributions in different segments are similar.

in NVDLA is 280 OPs/Byte. However, in the no-pipeline implementation, the CTC ratio of most layers is lower than 280 OPs/Byte, thus the computational resources of the no-pipeline implementation are underutilized due to memory bound.

When a DNN is pipelined, the intermediate fmaps are transferred directly to the consumer PUs without additional memory accesses [1], [46], [69], [74], [83], [84]. Thus, for the same workload, the CTC ratio of the pipeline implementation could be significantly increased as in Fig. 3 over the no-pipeline baseline. However, running the entire DNN model with a full pipeline architecture as in Fig. 1(a) raises hardware cost due to deep pipeline overhead. Thereby, we are seeking a novel architecture that has a higher CTC ratio than no-pipeline designs and avoid the penalty of full pipeline hardware.

B. Opportunities for More Efficient Pipeline

To find an architecture as a compromise between no-pipeline and full pipeline architectures, we study various NN models, and discover that the model layers in a no-pipeline architecture often exhibit the alternating pattern of high and low CTC ratios as illustrated in Fig. 4. Intuitively, layers with low and high CTC ratios can be put into the same segment pipeline, thus the CTC ratio of layers in a segment can be boosted on average. In addition, by pipelining the layers in the segments, the memory accesses to intermediate fmaps within the segments can also be eliminated. As shown in Fig. 3 and Fig. 4, the CTC ratio of segment-grained pipeline implementation increases significantly over the no-pipeline counterpart, and even close to that of the full-pipeline solution. This is one of the reasons that the segment-grained pipeline achieves higher performance with the same peak computing performance and memory bandwidth as shown in Fig. 2.

On the other hand, with proper layer grouping, the distribution of the computing-operation number in different segments can be similar, as shown in Fig. 5. For example, segment 1-4 have one high-Ops layer, one medium-Ops layer and four low-Ops layers, though their positions in the segments are inconsistent. If we can properly assign the layers in the segment to the suitable PUs of the accelerator as in Fig. 1(c), all segments can achieve perfect **load-balancing**

on the accelerator with fixed resource allocation across the pipeline. At the same time, resource shortage caused by over-deep pipelines can be avoided, because the segments in the models will not contain too many layers. Thereby a shared SPA accelerator can be designed for all segments and different segments can execute efficiently in it.

C. Design Challenges to SPA

According to previous analysis, SPA is able to provide performance benefits for DNN workloads. The remainder of the section elaborates on the major challenges to generating the highly efficient SPA accelerator solution for the given models.

Challenge 1: the need for flexible SPA pipeline. A SPA accelerator should be configurable to map different model segments onto the multi-PU pipeline with high utilization. It is different from previous classic accelerators [1], [7], [28], [47], [69], [83] and requires additional design considerations to achieve efficient segment pipelining: 1) Reconfigurable inter-PU connection: Model segments can be viewed as diverse task graphs composed by layers, when they are spatially mapped onto the SPA accelerator, they result in distinct communication patterns between PUs operating in pipeline as shown in Fig. 1(c). Therefore, the accelerator needs to dynamically configure the connection between PUs according to the layer data dependencies within the segments; and 2) Flexible PU dataflow: As data reuse preferences may vary across different layers in a segment and also between segments. Thus, PUs in the pipeline can operate in different dataflow to suite their associated layers in a segment and also switch their operating dataflow between different segments of the model.

Challenge 2: the need for efficient model segmentation. In model segmentation, we need to partition the DNN model into multiple segments and sequentially execute the segments on the SPA accelerator. A successful model segmentation strategy will lead to a high CTC ratio, and also contributes to similar operational distributions among all the model segments, which translates to higher PU resource utilization. Thereby, the upper bound of the accelerator performance is determined by the quality of model segmentation. However, the optimal segmentation solution may vary significantly to different DNN tasks. Even for the same model, there is still a considerable exploration space in model segmentation, as evidenced in Fig. 4. Therefore, we need to precisely define a complete segmentation search space to contain premier segmentation options and propose a model segmentation method to efficiently search over this space.

Challenge 3: the need for efficient hardware customization method. Another important aspect to enhance the overall efficiency of a SPA design is to adjust the resource assignment of PUs according to the design goal and workload characteristics. The PUs in a SPA accelerator are shared by all segments of the given models, rather than being exclusive to a layer in the model as in traditional full pipeline architecture [16], [29], [83]. Consequently, we need to jointly optimize the resource allocation such as the arithmetic-unit numbers, and the PU buffer size for all segments, so that segments in the tasks can fully utilize the hardware resources and achieve high performance.

III. AutoSeg: Overview

In this section, we introduce AutoSeg, a HW/SW co-design framework, the workflow of which is presented in Fig. 6. The proposed framework aims to generate a specialized SPA accelerator solution for the DNN workloads, which can be expressed by *directed acyclic graphs (DAG)* $G = (L, E)$. In a workload $G = (L, E)$, node $l \in L$ represents layer l (e.g., Conv layer) and edge $\langle l_1, l_2 \rangle \in E$ represents the data dependency between layers. In the AutoSeg design flow, users can specify the design goal (i.e., minimizing latency or maximizing throughput) and the resource constraints (e.g., maximum area, power consumption, or hardware resources) as input together with the DNN model description. Accordingly, AutoSeg will generate the accelerator co-design solution including the hardware parameters and the DNN model segmentation strategy. To achieve efficient accelerator customization, as shown in Fig. 6, the framework of AutoSeg requires devising two major components as follows:

SPA Template. To address **Challenge 1**, we introduced the SPA template including multiple basic hardware primitives which are parameterizable and composable as shown in Fig. 7. Once the co-design solution of target DNN models is determined, a customized pipelined DNN accelerator can be derived from the template. The two main components of the SPA template includes the dataflow-hybrid PUs and the reconfigurable inter-PU fabrics. A dataflow-hybrid PU is composed of a systolic array of *processing elements (PEs)*, an activation buffer and a weight buffer, and it enables different dataflow, e.g. *weight stationary (WS)* and *output stationary (OS)* dataflow. The reconfigurable inter-PU fabric is responsible for data exchange between PUs in the pipeline, which is implemented as a pruned N-input, N-output Benes network. Section IV delves into the implementation of SPA.

HW/SW Co-design Engine. As the critical component of AutoSeg, HW/SW co-design engine explores the model segmentation search space and the hardware design space to generate the most suitable SPA design for the target models. The co-design process includes two key steps:

1) model segmentation. This step decide how the target models are divided into segments and how the layers in a segment are mapped into the hardware pipeline composed of interconnected PUs. To address **Challenge 2**, we define a complete segmentation search space that covers all potential design points, i.e., we allow segments to contain more layers than the number of PUs so that each PU is assigned to at least one layer as `Segment-2` in Fig. 6, and the

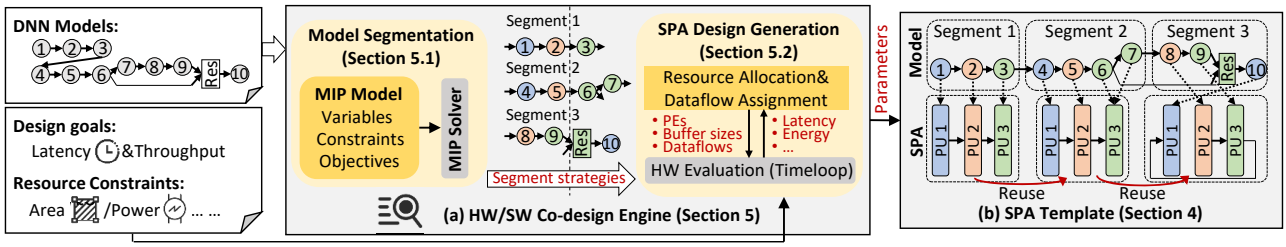


Figure 6. The workflow of AutoSeg.

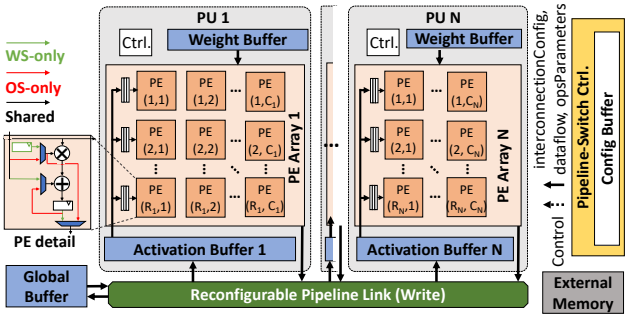


Figure 7. Architecture overview for a SPA design.

layers will not be necessarily bound onto the connected PUs exactly according to their order in the segment, which can be exemplified as `Segment-3` in Fig. 6. To obtain better segmentation solution in this space, we formulate the DNN model segmentation process as a MIP problem and conduct optimization process towards the design goal of two metrics - high CTC ratio for each segment and similar operational distributions in between segments, as introduced in Section V-A.

2) SPA design generation. On receiving the segmentation results, AutoSeg proceed to decide the hardware parameters, including the computation and memory resources assignment to the PUs, and how PUs are reconfigured during computation. To address **Challenge 3**, AutoSeg allocates hardware resources to each PU in the pipeline using the proposed heuristic resource allocation algorithm. The algorithm can directly perform global resource allocation for all PUs to increase the mapping efficiency of all segments based on the optimized CTC ratio and operational distributions, instead of performing tedious iterative joint tuning. This resource allocation algorithm is detailed in Section V-B.

IV. SPA Template Implementation

In this section, we provide a detailed description of the computation flow in SPA and the microarchitectures of the two key hardware primitives.

A. Segment-Based Computation Flow

Unlike the layer-wise computation flow which binds all layers onto a unified PU and the full pipeline computation flow which binds each layer to a dedicated PU, this computation flow decouples the layers and the PUs. Therefore, we can build a segment-grained pipeline and dynamically bind layers to the most suitable PU according to their characteristics.

The DNN execution flow is divided to multiple timeslots by segment. In a segment timeslot, each PU processes its assigned layers and pass the result to the subsequent PUs connected to it. Fig. 8 shows an example of segment-grained pipeline which contains three PUs. As presented in Fig. 8(a), the connection of PUs in different segments is varied and the layers can be flexibly bound onto the PUs, which reflects the decoupling nature of layers and PUs. Within a segment, we adopt a fine-grained execution fashion from [83], referred as piece-based execution fashion in this paper. As shown in Fig. 8(b), in the fine-grained execution fashion, each layer is divided into pieces to minimize the end-to-end latency of the model and the required on-chip memory to store fmaps. The size of layer-piece each PU processes can be determined by the data dependency between the related layers. As illustrated in Fig. 8(c), since the stride of layer Conv-2 is two, the layer Conv-1 need to produce at least two rows of output fmaps (ofmaps) in its piece-3 which will be consumed by Conv-2 in the next time block. The PU with the longest piece-time would dominate the latency of this segment. The more balanced the PU processing latency is, the higher accelerator efficiency is achieved.

B. Dataflow-Hybrid Pipeline Design

Since PUs in the pipeline process layers of different segments, the dataflow of their PE arrays should not be fixed in order to meet the data reuse preferences of layers. Thus in the SPA design, there are two goals to meet: 1) PUs in the pipeline can operate in coordination as producers and consumers even when they are running in different dataflow; 2) PUs can switch to another dataflow when the layer of the scheduled-in segment exhibits a different data reuse pattern from the completed layer.

PE Array Design. Suppose PU-$n(n = 1, ..., N)$ includes a $R_n \times C_n$ 2D systolic array of PEs, the PE muxes shown in Fig. 7 can select outputs from neighbors and enable the PE array to work in WS and OS dataflow when cooperating with the input loading mode switch shown in Fig. 9.

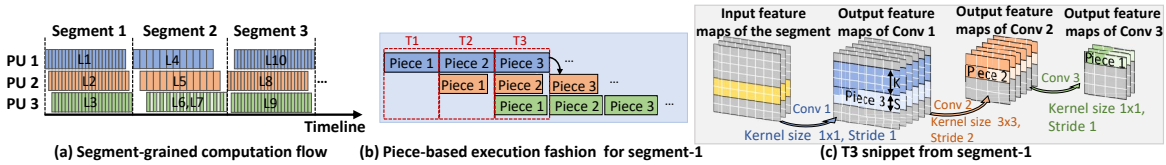


Figure 8. Computation flow of segment-grained pipeline. When multiple layers are mapped to the same PU in a segment, they are executed alternately, such as L6 and L7.

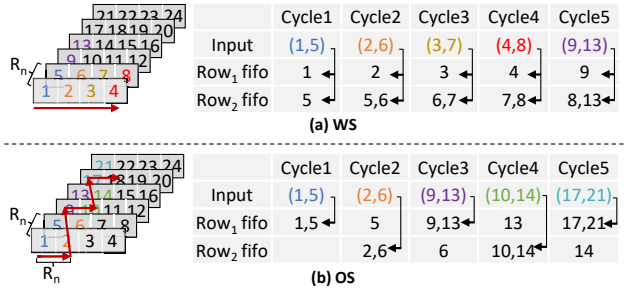


Figure 9. Input loading examples of PE array (R_n = 2).

In WS, the PE array pre-loads $R_n \times C_n$ weight along the input channel and output channel dimensions into the local registers. In each cycle, R_n inputs at different input channels are fed into the R_n input FIFOs of the systolic array in parallel, as shown in Fig. 9(a). The inputs are propagated from input FIFOs into PEs, which perform MAC operations, shift the input to right neighbor, and shift the partial result down for accumulation. In OS, R_n columns and C_n channels of ofmaps are mapped to the R_n rows and C_n columns of the PE array, respectively. The input FIFOs of the PE array access the inputs alternately, that is, one input FIFO reads R_n data from different input channels in each cycle, as shown in Fig. 9(b)). The data in the input FIFOs and the corresponding weights are propagated into PEs, which shift the input and weight to the right and down direction in the systolic array respectively.

PU Local Memory. To enable dataflow selection at PUs without having to transform the intermediate fmaps, the SPA stores the fmaps in the activation buffer in a channel-first fashion. Beside, as shown in Fig. 8(c), the active row number of the input fmaps (ifmaps) are equal to the kernel size plus stride $(K + S)$ of the layer, which provides opportunities to reuse the on-chip activation buffer. Therefore, the activation buffer is reused in a circular-shifted manner by the active rows. Eq. 1 shows how to generate the offset of the activation address from the coordinates $(c, w, r), c \in [0, C_i), w \in [0, W_i), h \in [0, H_i)$ (C_i, W_i, and H_i are channel, width, and height of the ifmaps).

$$offset = \lfloor \frac{c}{R_n} \rfloor + w \times \lceil \frac{C_i}{R_n} \rceil + h\%(K+S) \times W_i \times \lceil \frac{C_i}{R_n} \rceil \quad (1)$$

C. Reconfigurable Inter-PU Fabrics

In the pipeline, results of a PU must be streamed to the activation buffer of the subsequent PU according to the DAG topology of the running segment. Since the data dependency between layers varies to different segments, the inter-PU connection design needs to be programmable to establish the data path at run-time. To reach this goal, we prioritize three design requirements of the PU interconnect. Firstly, the interconnect should provide sufficient bandwidth to allow continuous data feeding for all PUs simultaneously. Secondly, the interconnect should support multiple data transfer patterns including unicast and multicast. Thirdly, the interconnect should have shortest possible delay between PUs to reduce pipeline stall.

In SPA, we adopt a Benes network [2], [33], [52] to support flexible interconnect between PUs. As shown in Fig. 10(a), Benes is a N-input, N-output, non-blocking network with O(NlogN) interconnection nodes, each of which consists of two 2-input muxes controlled by two selection bits, one for vertical output selection and one for diagonal output selection. The non-blocking property enabled by redundant connection in this network allows unicast and multicast transmission without data contention. Instead of clocked switches, clockless muxes transfer data from input to output in a single cycle.

Through workload analysis in the model segmentation process, we find that there is still some room for hardware overhead reduction. For target DNN models, the segments has similar structures, so interconnections between the PUs are not entirely random for a specific running model, and provide opportunity to reduce connection overhead by trimming the unnecessary freedom of interconnection. Based on predetermined data transfer patterns, we can prune links and connection nodes in the Benes network, reducing power/area overhead. Fig. 10 demonstrates an example of the pruning process. According to the segments in Fig. 6, the colored paths shown in Fig. 10(b) need to be activated to connect PUs in execution, while the connection nodes and links unused by the paths can be reduced, so that the Benes network will be reduced to a pruned instance adopted in the final customized accelerator, as shown in Fig. 10(c).

V. SPA Co-Design Engine

In the SPA co-design engine, AutoSeg uses the CTC ratio and operational distribution, which contain rich information

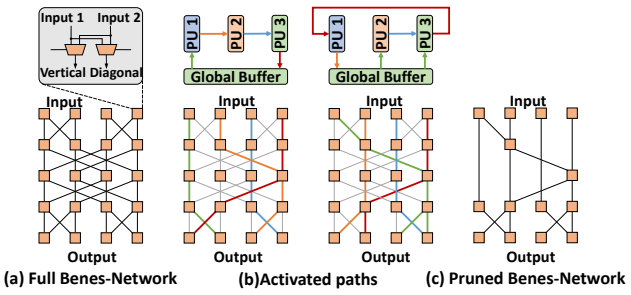


Figure 10. Example of Benes network pruning process. The switches of (b) are determined by the model segmentation example in Fig. 6.

Table I
NOTATIONS USED IN MODEL SEGMENTATION PROCESS.

Indices	
l	layer-l, $l \in [1, L]$
s	segment-s, $s \in [1, S]$ segment-s_1 is executed before segment-s_2, if $s_1 < s_2$.
n	PU-n, $n \in [1, N]$
Variables	
$\lambda_{l,n,s}$	taking value 1 if layer-l is mapped to segment-s and PU-n, 0 otherwise.
$\omega_{n_1,n_2,s}$	taking value 1 if PU-n_1 to PU-n_2 have data communication in segment-s, 0 otherwise.
Constants	
$ops(l)$	the number of MACs of layer-l.
$access(l)$	the memory access of layer-l.

(e.g., bandwidth requirement and compute-load distribution), as a proxy of the system performance, e.g., latency or throughput, to bridge the HW/SW search space. The SPA model segmentation stage partitions a DNN into segments with high CTC ratios and similar operational distributions, while the design generation stage directly optimizes the hardware design-parameters on the basis of the segmentation solution as well as the corresponding metrics of CTC-ration and operational distribution. No iterative search is required between these two steps of HW/SW search process since they are decoupled, and the results of model segmentation can be repeatedly used to generate SPA designs under different hardware constraints.

A. Model Segmentation

In this section, we discuss how to formulate the model segmentation process as a MIP problem that takes the DAG-based description of the DNN model ($G = (L, E)$) as input and generates a optimized segmentation solution.

Variables and Constants. The variables and constants of the problem are summarized in Table I, where S and N represent the number of model segments and PUs, respectively. All possible (S, N) tuples will be traversed by the SPA co-design engine.

We use a binary matrix variable λ to encode the model segmentation search space, which can represent all possible segmentation results, i.e., layers (indexed by l) belong to which segment (indexed by s) and will be mapped onto which PU (indexed by n). And we use a binary auxiliary matrix variable ω to represent data movement between PUs in each segment. There are also two parameters used to describe layer l: 1) the number of operations - $ops(l)$; and 2) the memory access number - $access(l)$.

Constraints. After the segmentation solution space is defined, there are some design rules that must be specified in the problem to include only the valid segmentation options for the target hardware.

1) Layer assignment constraint: A layer can only be mapped to one PU to avoid repeated computation, while each PU is assigned to at least one layer in a segment to avoid resource idling:

$$\begin{aligned} &\forall l : \textstyle\sum_{\forall s,\forall n} \lambda_{l,n,s} = 1 \\ &\forall s, \forall n : \textstyle\sum_{\forall l} \lambda_{l,n,s} \geq 1 \end{aligned} \tag{2}$$

2) Data dependency constraint: If there exists data dependency of layer-l_2 on layer-l_1, i.e., $\langle l_1, l_2 \rangle \in E$, which represents the data dependencies in the given DNN models, layer-l_2 must not be executed before layer-l_1 when they are in different segments:

$$\begin{aligned} &\forall \langle l_1, l_2 \rangle \in E, \forall s_1 < s_2 : \\ &\textstyle\sum_{\forall n} \lambda_{l_1,n,s_2} + \sum_{\forall n} \lambda_{l_2,n,s_1} \leq 1 \end{aligned} \tag{3}$$

In a segment, layers on different PUs are executed in parallel. Thus, data movement from PU-n_1 to PU-n_2 and from PU-n_2 to PU-n_1 must not be simultaneous to avoid data dependency conflict:

$$\begin{aligned} &\forall s, \forall \langle l_1, l_2 \rangle \in E, \forall n_1 \neq n_2 : \\ &\omega_{n_1,n_2,s} \geq \lambda_{l_1,n_1,s} + \lambda_{l_2,n_2,s} - 1 \\ &\omega_{n_1,n_2,s} + \omega_{n_2,n_1,s} \leq 1 \end{aligned} \tag{4}$$

Objective Function. Then, we describe the objective function for the model segmentation problem.

1) CTC ratio driven objective: As we analyzed in Section II, a high CTC ratio helps alleviate the memory bandwidth bottleneck and enables the accelerator to have higher computation performance. Therefore, we aim to maximize the minimum CTC ratio of all segments, so that a dedicated accelerator can be designed to better meet the bandwidth requirements of all segments:

$$\begin{aligned} &\min \frac{1}{C\hat{T}C} \\ &\forall s : C\hat{T}C * \sum_{\forall l,\forall n} \lambda_{l,n,s} access(l) \leq \sum_{\forall l,\forall n} \lambda_{l,n,s} ops(l) \end{aligned} \tag{5}$$

2) Load-balance driven objective: As stated in Section IV-A, balancing the latency of PUs that run a segment in pipeline leads to higher accelerator efficiency, whilst

similar operation distributions across the segments enables balanced PU workload assignment in all segment pipelines, as illustrated in Eq. 6-9. In the equations, $L_{comp}[n][s]$ represents the computation latency of PU-n in segment-s, while $op[n][s]$ is the operational number of PU-n in segment-s and $PE[n]$ indicates the computation resource of PU-n. Thus, the overall latency of segment-s, $L_{comp}[s]$, depends on the PU with the maximum computation time (Eq. 7). When workloads for all PUs in segment-s are perfectly balanced, the process latency of PUs can be very close to each other, avoiding the bubble of resource idling (Eq. 8). Similar operation distribution between segments given by Eq. 9 is the condition for achieving workload balancing in all segments.

$$L_{comp}[n][s] \propto \frac{op[n][s]}{PE[n]} \tag{6}$$

$$L_{comp}[s] = max_n(L_{comp}[n][s]) \tag{7}$$

$$\frac{op[1][s]}{PE[1]} = \ldots = \frac{op[N][s]}{PE[N]} \Rightarrow \\ op[1][s] : \ldots : op[N][s] = PE[1] : \ldots : PE[N] \tag{8}$$

$$op[1][1] : \ldots : op[N][1] = \ldots = op[1][S] : \ldots : op[N][S] \tag{9}$$

The operational distributions are formally expressed as:

$$\forall s: \\ V_s = (\underbrace{\frac{\sum_{\forall l} \lambda_{l,1,s} ops(l)}{\sum_{\forall l,\forall n} \lambda_{l,n,s} ops(l)}, \frac{\sum_{\forall l} \lambda_{l,2,s} ops(l)}{\sum_{\forall l,\forall n} \lambda_{l,n,s} ops(l)}, \ldots}_{N}) \tag{10}$$

We use the *Manhattan* distance between operational distributions to represent segment similarity, which will be minimized in this problem:

$$\min S\hat{O}D = \textstyle\sum_{\forall s_1,s_2} |V_{s_1} - V_{s_2}| \tag{11}$$

Overall Objective. In the SPA co-design engine, the overall objective is a linear combination of $1/C\hat{T}C$ and $S\hat{O}D$, i.e., $\min(1/C\hat{T}C + S\hat{O}D)$, to maximize the CTC ratio while minimize the distance between segment operational distributions. For each possible combination of N PUs and S segments, we solve a MIP problem by using the optimization solver Gurobi [44]. By enumerating the segmentation options, the efficient solution will be discovered.

B. SPA Design Generation

After the task graph of targeted models are divided into segments and associated to the assigned PUs in the pipeline, the next step is to decide the amount of resource allocated to each PU to create the final customized SPA pipeline. In this section, we present a heuristic resource allocation algorithm, which consists of three steps:

Step 1. The operational distributions and bandwidth usage of all segments are normalized. Since the operational distributions $V[s]$ are similar to each other in our objective

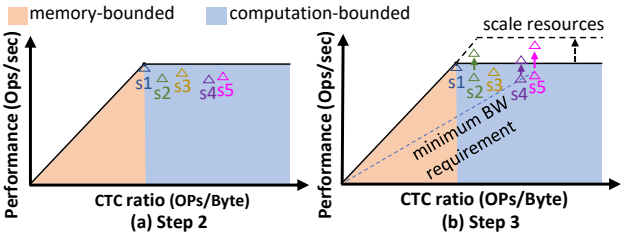


Figure 11. The procedure of scaling up the resources.

Eq. 11, the normalized $\hat{V}$ as the cluster center of all operational distributions can represent them effectively. Inspired by Eq. 7-9, the normalized operational distribution $\hat{V}$ (N-dimensional vector) are directly used as the quota indicator to allocate PE resources for N PUs in the pipeline, so that load-balance in PUs can be achieved for all segments. We then calculate the memory access bandwidth usage $B\hat{W}[s]$ according to the PE allocation ratio $\hat{V}$ for each segment as expressed in Eq. 12. In our ingenious design, $B\hat{W}[s]$ is approximate to the reciprocal of the CTC ratio expressed by Eq. 5. Then, we deem the normalized bandwidth usage as the maximum $B\hat{W}[s]$ of all segments so that we can avoid memory bandwidth hunger for any segment in Step 2. According to our CTC ratio objective in Eq. 5, the normalized bandwidth usage can be minimized, alleviating the bandwidth bottleneck effectively.

$$\begin{aligned} & B\hat{W}[s] \\ &= \sum_{n=1}^{N} (\hat{V}[n] \times \frac{\sum_{\forall l} \lambda_{l,n,s} access(l)}{\sum_{\forall l} \lambda_{l,n,s} ops(l)}) \\ &\approx \sum_{n=1}^{N} (\frac{\sum_{\forall l} \lambda_{l,n,s} ops(l)}{\sum_{\forall l,\forall n} \lambda_{l,n,s} ops(l)} \times \frac{\sum_{\forall l} \lambda_{l,n,s} access(l)}{\sum_{\forall l} \lambda_{l,n,s} ops(l)}) \\ &\leq \frac{1}{C\hat{T}C} \end{aligned} \tag{12}$$

Step 2. Based on the normalized operational distribution and normalized bandwidth usage, we allocate a proper number of PEs in the PUs to maximize the bandwidth utility, so that all the model-segment implementation will not be memory-bounded as explained in Fig. 11(a). Afterwards, we also allocate each PU with the minimum amount of on-chip memory space they needed based on the PE number.

Step 3. AutoSeg re-adjustes the resource allocation to meet the given hardware constraint. If the hardware overhead is below than the budget, we select the one whose performance is the most limited by computational resources from the potentially optimizable segments, and scale up the resource of PU-n with the longest latency in the selected segment. Fig. 11 shows this procedure of resource upscaling. Since segment-s_5 is the most computation-bounded, i.e., has minimum bandwidth usage, enlarging the PU that dominates the latency of this segment will gain performance,

Algorithm 1: SPA Resource Allocation

1 Get λ, V from `ModelSegmentation(`L, E, N, S`)`;
2 Set resource constrains: ResConstr;
3 Set available memory bandwidth: $\mathrm{BW}_{\max}$;
4 $\hat{V}[1:N] \leftarrow$ normalized vertor of $V[1:N][1:S]$;
5 $\forall s \in [1:S], \hat{BW}[s] \leftarrow$ bandwidth requirement in Eq. 12;
6 $\hat{BW}_{max} \leftarrow \max(\hat{BW})$;
7 **for** $n \in [1:N]$ **do**
8 $\quad PE[n] \leftarrow \hat{V}[n] \times \mathrm{BW}_{\max} / \hat{BW}_{max} / Freq$;
9 $\quad PE[n] \leftarrow 2^{\lfloor \log_2 PE[n] \rfloor}$;
10 $\quad$ Update buffer size $AB[n]$ and $WB[n]$;
11 **for** $n \in [1:N]$ **do**
12 $\quad Res[n], L_{comp}[n][1{:}S], L_{access}[n][1{:}S], DF[n][1{:}S] \leftarrow$ `Eva(`$\lambda_{:,n,:}, PE[n], AB[n], WB[n], .cost(Timeloop)$`)`;
13 **if** *design goal is throughput* **then**
14 $\quad Batch = ResConstr / (\sum Res[n] + Link\_Res)$;
15 **else**
16 $\quad Batch = 1$;
17 **if** $(\sum_{n=1}^{N} Res + Link\_Res) \times Batch <$ ResConstr **then**
18 $\quad$ **while** $[1{:}S]/Q$ *not empty* **do**
19 $\quad\quad$ from $[1{:}S]/Q$ find $\hat{s}$ with minimum bandwidth usage;
20 $\quad\quad \hat{n} \leftarrow \mathrm{argmax}_n(L_{comp}[n][\hat{s}])$;
21 $\quad\quad$ PU-$\hat{n}$ scale to $PE[\hat{n}] * 2, WB[\hat{n}] * 2$;
22 $\quad\quad$ **if** *constraints can be satisfied* **then**
23 $\quad\quad\quad$ Update $PE, WB, Res, L_{comp}, L_{access}$;
24 $\quad\quad$ **else**
25 $\quad\quad\quad$ add $\hat{s}$ to Q;
26 **else if** $(\sum_{n=1}^{N} Res + Link\_Res) \times Batch >$ ResConstr **then**
27 $\quad$ **while** *constraints not be satisfied* **do**
28 $\quad\quad n \leftarrow$ PU with the lowest utilization;
29 $\quad\quad PE[n] \leftarrow PE[n]/2, WB[n] \leftarrow WB[n]/2$;
30 $\quad\quad$ Update $PE, WB, Res, L_{comp}, L_{access}$;

while the performance of other segments (s_2 and s_4) limited by the same PU also improves. On the contrary, when the actual resource consumption exceeds the budget, we reduce the resources of the PU with the least resource utilization.

The detailed heuristic resource allocation algorithm is given in Alg. 1. In line 4-6, AutoSeg normalizes the operational distribution and bandwidth requirement as stated in **Step 1**. In line 7-10, it maximizes the use of the available bandwidth resources as in **Step 2**. Then the engine adjusts the size of the PE array to the power of 2 for efficient hardware design (line 9), and it also update the activation buffers AB to ensure they can at least store the active rows of ifmaps, and the weight buffers WB to allow on-chip buffering of $K^2 \times PE[n]$ weights (line 10). And in line 11-12, the engine evaluates the key metrics of each PU, i.e., resource consumption Res, computation latency L_{comp} and memory access latency L_{access}, and it also select the operation dataflow with lower latency ($DF[n][1:S]$) for each segment. In line 17-30, the resource allocation will be adjusted as described in **Step 3**. When there are still resources available, the resource provision will be increased (line 17-25), wherein $Link\_Res$ is the resource consumption of the reconfigurable inter-PU fabrics (line 17). For different design scenarios, batch-level execution parallelism is also supported in the design generation stage. When the throughput goal is prioritized over latency, the batch size will be adjusted to maximize the throughput of the generated SPA solution (line 13-16).

Table II
THE HARDWARE RESOURCE BUDGET.

ASICs	#PEs	On-chip memory(KB)	Bandwidth(GB/s)
Eyeriss	192	123	25
NVDLA-Small	256	256	5
NVDLA-Large	2048	512	20
EdgeTPU	8192	8192	0.5
FPGAs	**#DSPs**	**On-chip memory(BRAM36K)**	**Bandwidth(GB/s)**
ZU3EG	360	216	3.5
7Z045	900	545	5.3
KU115	5520	2160	19.2

VI. EVALUATION

A. Experimental Setup

Acceleration Workload. We employ several popular DNN models to build the evaluation workload: AlexNet [31], VGG16 [61], MobileNetV1/V2 [23], [56], ResNet18/50/152 [19], SqueezeNet1.0 [25], and InceptionV1 [62].

Hardware Baseline Implementation. We evaluate the proposed AutoSeg framework for accelerator customization on both ASICs and FPGAs. For the ASIC platform, we set up four hardware resource constraints to represent different application scenarios: the same budget as that of Eyeriss (dense) [7], NVDLA-Small [47], NVDLA-Large [47], [48], and EdgeTPU [42], [43], respectively, as shown in Table II. To our knowledge, there is almost no published results for full-pipeline DNN accelerators implemented as ASICs in prior literature. Therefore, we consider three different FPGA boards, as shown in Table II, including two low-power FPGAs (Avnet Ultra96 with Xilinx XAZU3EG, and Xilinx ZC706 with Xilinx XC7Z045) and a large-scale FPGA (AlphaData 8K5 with Xilinx XCKU115). And we compare the proposed AutoSeg with several state-of-the-art FPGA-based accelerator designs of different architectures: 1) the no-pipeline architectures, including DPU [76], Light-OPU [81], Dynamap [45], and HybridDNN [79]; 2) the full-pipeline architectures, including DNNBuilder [83] and TGPA [69]; 3) the hybrid architecture, DNNExplorer [84], which uses full pipeline for the first several layers of the model and no pipeline for the remnants; and 4) the heterogeneous multi-PUs architecture, Multi-CLP [59].

Co-design Optimization Baseline. Since customizing a specialized SPA accelerator is a typical HW/SW co-optimization problem, we implement the following solutions that are obtained through varied combinations of hardware and software optimizers, and evaluate them as baselines to show the benefits brought by the AutoSeg co-design engine:

The first two baselines are *1) the MIP-Random search method, and 2) the MIP-Baye search method.* These two design methods partition the model by solving our introduced MIP problem as instructed in Section V-A, however the former uses a random-sampling method while the latter employs the bayesian algorithm to determine pipeline resource assignment, rather than using the proposed heuristic algorithm described in Section V-B. Both of them run 500 optimization iterations in resource assignment stage.

3) The Baye-Heuristic search method. It uses the bayesian algorithm running 2000 iterations to generate the DNN model segmentation results, and uses the AutoSeg heuristic algorithm to generate the hardware design parameters.

4) The Baye-Baye search method. It uses a two-loop bayesian algorithm inherited from [60] to search for the SPA co-design solution, wherein the outer loop optimizes the hardware pipeline implementation, and the inner loop optimizes the model segmentation strategy for a given accelerator generatd by the outer loop optimizer. In practise, this baseline runs 500 iterations for hardware search and 2000 iterations for the segmentation search.

Hardware Metrics. In the SPA design generation stage (Section V-B), we uses Timeloop [49], an open-source evaluation model, to quickly evaluate the hardware-level metrics, e.g., latency, area, etc., for each PU. In addition, we synthesize the inter-PU connection designs, and extract the energy and area profile of the interconnection nodes to estimate the inter-PU fabrics overhead. All designs are implemented and evaluated in the TSMC 28nm technology.

B. Performance

ASIC Design. Fig. 12 shows the speedup of the AutoSeg design over the baseline ASIC-based architectures of the same-level hardware resources. AutoSeg delivers $2.71\times$, $3.55\times$, $2.21\times$, and $3.89\times$ performance speedup on average when compared to Eyeriss, NVDLA-Small, NVDLA-Large and EdgeTPU, respectively. As we mentioned, the efficiency boost is mainly due to the CTC ratio increase. In the pipelined architecture, only the memory accesses to intermediate feature are reduced, while the memory access to model weights will not be reduced. Therefore, according to Amdahl's law, the improvement of CTC ratio is limited by the proportion of memory access to intermediate fmaps, which changes in different DNN architectures. As shown in Fig. 13, compared to AlexNet and ResNet18, models such as MobileNetV1 and SqueezeNet reduce more memory accesses in our customized SPA accelerators, because they have a relatively larger intermediate fmaps, which is also one of the key trends in the latest NN architectures [24], [56], [62], [63], [64]. For example, in MobileNetV1/V2, intermediate fmaps are responsible for 65% of the total memory footprint, so the SPA accelerator brings considerable performance boost compared to the no-pipeline baselines. At the same time, the performance of pipeline architecture also depends on whether the computation load is balanced across the PUs. Unbalanced load will cause bubbles. For example, depthwise Conv and pointwise Conv can be mapped onto different PUs to improve resource utilization. As a result of these two factors, AutoSeg can perform well on MobileNetV1/V2 with an average of $5\times$ speedup. As for SqueezeNet, it also has a high proportion of intermediate fmaps, and its bandwidth pressure can also be greatly reduced on the pipeline design as Fig. 13. Therefore, AutoSeg can achieve an average of $3\times$ speedup on SqueezeNet.

FPGA Design. Table III shows the throughput comparison between the AutoSeg solutions and the baseline FPGA-based accelerators. For each evaluated benchmark, we select the designs that achieve the best results we can find on the FPGA platform and use the AutoSeg solutions with close resource consumption for comparison. All designs are worked in 8-bits by packing two activations together (i.e., double batch size compared to 16-bits) according to [11], thus the shown throughput of TGPA and DNNExplorer are doubled over the original data reported in literature for fairness. The results show that our designs achieve up to $1.6\times$ higher performance on large-scale FPGA, and $3.6\times$ higher performance on low-power FPGAs of equivalent resources provision. The speedups of AutoSeg over ASIC designs are more significant than that over FPGA designs, because the FPGA designs with pipeline, i.e., DNNBuilder, TGPA and DNNExplorer, have higher resource utilization, and the FPGA designs without pipeline have also made some optimizations to improve utilization, for example, Light-OPU is optimized for lightweight networks, Dynamap and HybridDNN support different algorithm (e.g., Winograd) for high-performance. In order to analyze the FPGA resource utilization of different schemes, we use the metric of DSP efficiency from DNNExplorer to evaluate whether the AutoSeg-generated accelerators maximize the usage of allocated DSPs. For AlexNet, VGG16 and ResNet152, our designs outperform the full pipeline designs DNNBuilder and TGPA in DSP efficiency, because our shallow segment-grained pipeline designs can be reused flexibly by the partitioned segments and have fewer bubbles in the pipeline. When compared to the generic architectures DPU, Light-OPU and Dynamap, our SPA designs deliver $3.4\times$, $1.4\times$, and $1.8\times$ higher DSP efficiency in MobileNetV2, GoogleNet, and SqueezeNet respectively, because our designs match the pipeline stages with the DNN layers and they reduce the bandwidth-induced computational resource idling due to higher CTC ratios. When compared to advanced design proposed in DNNExplorer, which directly

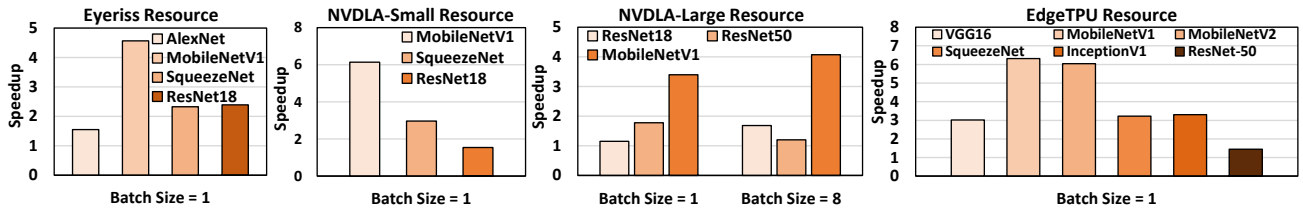


Figure 12. Performance comparison of AutoSeg with the baseline ASIC designs.

Table III
PERFORMANCE COMPARISON TO STATE-OF-THE-ART FPGA-BASED DESIGNS.

Model	**AlexNet**					**VGG16**							**ResNet152**		-
Design	DNNBuilder		TGPA	**ours**		HybridDNN		DNNBuilder	TGPA	DNNExplorer	**ours**		TGPA	**ours**	-
Device	7Z045	KU115	VU9P	7Z045	KU115	7Z020	VU9P	KU115	VU9P	KU115	ZU3EG	KU115	VU9P	KU115	-
Freq.(MHz)	200	220	200	200	200	100	167	235	210	200	200	235	200	200	-
DSP	808 (90%)	4854 (88%)	4480 (66%)	840 (93.3%)	5192 (94.1%)	220 (100%)	5163 (75.9%)	4318 (78%)	4096 (60%)	4444 (80.5%)	264 (73.3%)	5128 (92.9%)	4096 (60%)	4390 (79.5%)	-
BRAM	303 (56%)	986 (46%)	1682 (78%)	509 (93.3%)	1834 (84.9%)	NA	NA	1578 (81%)	1690 (78%)	1648 (76.3%)	209 (96.5%)	1486 (76.9%)	2960 (68%)	2136 (98.9%)	-
Perf.(GOP/s)	494	3265	2864	635	3955	83.3	3376	4022	3020	3405	203	4778	2926	3166	-
DSP Effi.	76.4%	76.4%	80%	94.5%	95.2%	94.6%	97.9%	99.1%	87.7%	95.8%	96.1%	99.2%	89.3%	90.1%	-
Model	**MobileNetV2**					**InceptionV1**				**SqueezeNet**					
Design	DPU	Light-OPU	**ours**			DPU	Dynamap	**ours**		DPU	Light-OPU	Multi-CLP	**ours**		
Device	ZU3EG	K325T	ZU3EG	7Z045	KU115	ZU3EG	U200	ZU3EG	KU115	ZU3EG	K325T	KU115	ZU3EG	7Z045	KU115
Freq.(MHz)	287	200	300	200	200	287	286	300	250	287	200	170	300	200	200
DSP	282 (78.3%)	704 (83.8%)	312 (86.7%)	744 (82.7%)	4776 (86.5%)	282 (78.3%)	6239 (91.0%)	336 (93.3%)	5192 (94.1%)	282 (78.3%)	704 (83.8%)	3238 (58.7%)	340 (94.4%)	832 (92.4%)	5192 (94.1%)
BRAM	123 (56.9%)	194 (43.5%)	188 (86.8%)	380 (69.7%)	2125 (98.4%)	123 (56.9%)	2000 (93.0%)	205 (94.9%)	1896 (87.8%)	123 (56.9%)	193.5 (43.48%)	524 (24.3%)	158 (72.9%)	245 (45.0%)	1054 (48.8%)
Perf.(GOP/s)	80	197	290	483	3188	226	3568	327	5053	128	267	1216	270	461	2711
DSP Effi.	24.7%	35%	77.5%	81.2%	83.4%	69.8%	50.0%	81.1%	97.3%	39.5%	47.5%	55.2%	66.2%	69.3%	65.3%

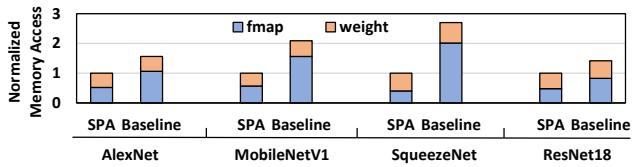


Figure 13. Memory access reduction of SPA compared to Eyeriss.

combines both generic and pipelined architectures onto a implementation, the generated SPA solutions can still gain significant performance improvement since DNNExplorer is not flexible enough to perform different model efficiently, especially for emerging DNN models, such as MobileNet, SqueezeNet, etc. We also implement an accelerator using the template provided in Multi-CLP, which performs frame-based parallelism in multiple heterogeneous PUs to achieve higher PE utilization, and the intermediate data is still written back to the off-chip memory without bandwidth saving. Compared with it, SPA design deliver 1.26× higher DSP efficiency in SqueezeNet. This is because the double buffer cannot fully overlap the data transfer with the computation.

C. Detailed Comparison and Case Study

To understand the essential differences between no-pipeline, full-pipeline and segment-grained pipeline architectures, we present a detailed comparison of the AlexNet-dedicated designs (only Conv layer) on Xilinx ZC706@200MHz. Table IV presents the design parameters chosen by [29] for no-pipeline architecture while Table V describes the exemplary full-pipeline architecture chosen by [83]. The AutoSeg-generated SPA accelerator are presented in Table VI. All accelerators use the same number arithmetic-unit (768 PEs).

Table IV, V, and VI also shows the assignment of DNN

Table IV
THE CUSTOMIZED NO-PIPELINE ACCELERATOR.

	PU-1					**Overall**
#PE ($C_n \times R_n$)	96×8					768
Layers	conv1_a/b	conv2_a/b	conv3_a/b	conv4_a/b	conv5_a/b	
Latency(ms)	1.83	2.19	0.97	0.73	0.73	6.45

Table V
THE CUSTOMIZED FULL-PIPELINE ACCELERATOR.

	PU-1/2	**PU-3/4**	**PU-5/6**	**PU-7/8**	**PU-9/10**	**Overall**
#PE ($C_n \times R_n$)	16×4(×2)	16×8(×2)	8×8(×2)	8×8(×2)	8×8(×2)	768
Layers	conv1_a/b	conv2_a/b	conv3_a/b	conv4_a/b	conv5_a/b	
Operational distribution	0.15	0.32	0.21	0.16	0.16	
Latency(ms)	4.90	4.37	**5.83**	4.38	4.38	5.83

Table VI
THE CUSTOMIZED SPA ACCELERATOR.

	PU-1	**PU-2**	**PU-3**	**PU-4**	**Overall**
#PE ($C_n \times R_n$)	32×4	32×8	32×8	32×4	768
Segment1	conv1_a/b	conv2_a/b	conv3_a/b,conv4_a/b	conv5_a/b	
Operational distribution	0.15	0.32	0.37	0.16	
Latency(ms)	4.90	4.37	**5.11**	4.38	5.11

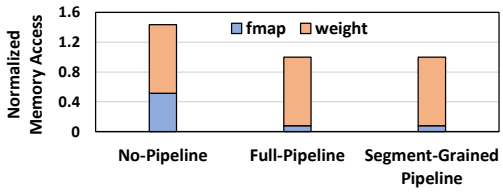


Figure 14. The memory access for different customized designs of AlexNet on ZC706@200MHz.

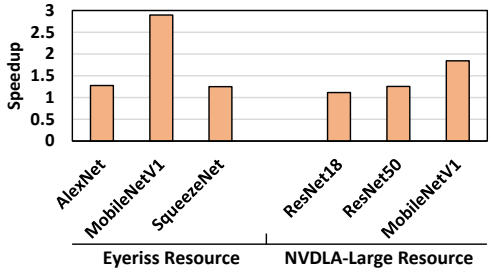


Figure 15. Performance comparison with software layer fusion optimization.

layers and the execution latency on each PU. Compared to the no-pipeline and full-pipeline accelerator, SPA accelerator achieves $1.26\times$ and $1.14\times$ speedup, respectively, because of the improvement of PE utilization. A unified PUs cannot be effectively utilized by all layers, so the PE utilization of the no-pipeline accelerator in Table IV is only 71.0%. As for full pipeline architectures, the layer is fixed to its customized PU. However, data transmitted between consumer and producer PUs needs to be aligned in real case for efficient hardware design, resulting in the discontinuous allocation of PEs between PUs (e.g., in our paper and [83], [84], the number of PEs in the PU is limited to a power of 2). Therefore, it is difficult to allocate balanced resources for each PU under a limited resource budget, resulting in many bubbles in the pipeline, and the PE utilization is only 78.1% for the example shown in the Table V. On the contrary, in addition to the heuristic design generation algorithm proposed to efficiently allocate resources for each PU, SPA co-design engine can also flexibly assign the layers using the model segmentation method, which allows us to obtain more balanced latency as exemplified in Table VI. Consequently, the PE utilization of the generated SPA accelerator is as high as 89.6%.

We also analyzed the impact of different architectures on the memory overhead. Compared to no-pipeline accelerators, full-pipeline and SPA accelerators reduce the memory access to DNN fmaps significantly as shown in Fig. 14, which lowers the energy consumed by high-cost main-memory access and also eases the memory bandwidth bottleneck.

D. Comparison with Fusion Optimization

To reduce the memory accesses of the intermediate fmaps, software layer fusion optimization [4], [85] has also been proposed to execute multiple layers in cascade on a single PU. In order to better understand the effectiveness of AutoSeg design, we apply the layer fusion optimization from Optimus [4] to the baseline designs, and compare the AutoSeg design with them. Fig. 15 shows the speedup of AutoSeg design over the baseline designs with layer fusion optimization. Layer fusion can bring performance improvement to the no-pipeline architectures. However, compared with AutoSeg, it is still inferior, which is mainly due to two factors. First, adjacent fmap tiles of the same layer are usually overlapped with each other, but the layer fusion optimization makes adjacent tiles of the same layer not perform consecutively. Thus, the overlapping data, which is inactive and not reused until all the corresponding tiles of other layers in the cascade have been calculated, is usually cached in the on-chip buffer, which reduces the actual available on-chip buffer area for the active data. As a result, the on-chip buffer resources are not efficiently utilized, which limits the optimization of memory accesses. Instead, there are multiple PUs in AutoSeg design that execute multiple layers in parallel, allowing overlapping data to be reused in time. Therefore, AutoSeg design reduces more memory accesses than the no-pipeline design with layer fusion as shown in Fig. 16. Second, layer fusion cannot solve the problem that a unified PU cannot be efficiently utilized by all layers.

E. Energy Efficiency

In this section, we compare the energy efficiency of the AutoSeg design with that of the baseline designs and the baseline designs with layer fusion (marked as *Fusion*). Fig. 16 illustrates the breakdown of energy. And we also quantify the additional energy overhead of the inter-PUs connections and dataflow-hybrid muxes (*Others* in Fig. 16) in AutoSeg, which accounted for less than 3% of the energy overhead. On the whole, AutoSeg achieves an average energy efficiency improvement of $1.65\times$ and $1.32\times$ over baselines and *Fusion*, respectively. The energy efficiency gains come from the performance speedup and the memory access reduction.

F. Generality Analysis

In this experiment, we demonstrate the generality of the SPA. We design a dedicated SPA accelerator for a model using the AutoSeg co-design engine (for example, AlexNet-dedicated design is customized for the AlexNet model), and mapped other models to the dedicated SPA accelerator. We perform model mapping by changing the target of the MIP-based model segmentation method in Section V-A to the direct latency target and adding the connection constraints of the Pruned Benes-Network. Fig. 17 shows that the performance of a model on its non-dedicated accelerators is slightly lower than that of its model-dedicated accelerator,

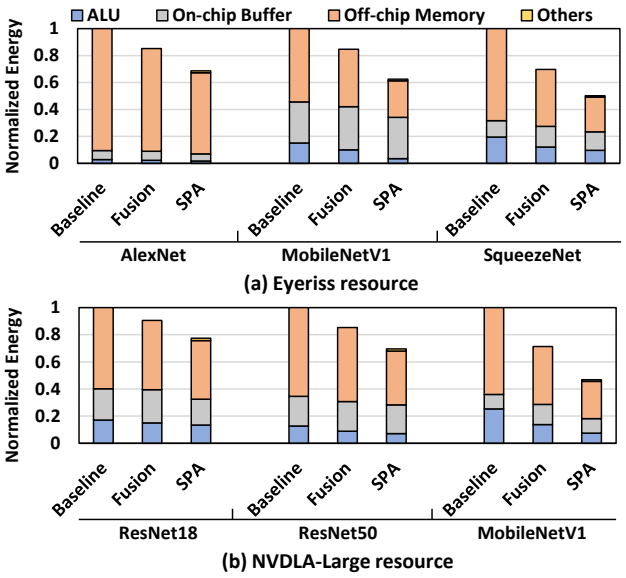


Figure 16. Energy breakdown.

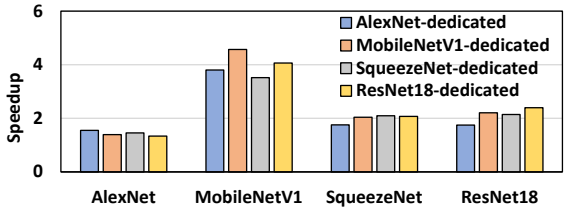


Figure 17. Performance comparison over Eyeriss baseline for generality analysis.

because the resource allocation and network connection pattern of the non-dedicated accelerators cannot match the model as perfectly as its model-dedicated accelerator. However, the performance of a model on its non-dedicated accelerators still exceeds that of baseline. This is because, when running models in SPA accelerators, the pipeline architecture can still reduce the off-chip memory access of the intermediate data, and multiple PUs are available for different layers of the models, resulting in higher utilization of computing units.

G. The Impact of AutoSeg Co-design Method

In Fig. 18, we run the baseline co-design methods for AlexNet and MobileNetV1, and label the best results of each method. AutoSeg can consistently find the best solution among all the methods. The premier performance of AutoSeg is mainly attributed to three factors:

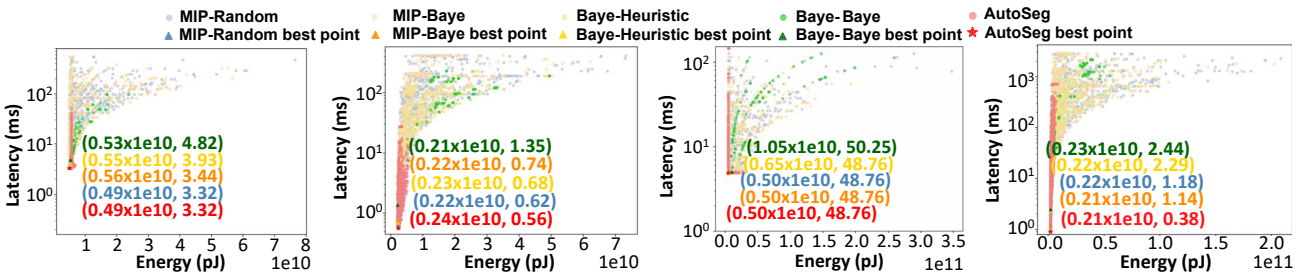
1) The heuristic resource allocation algorithm in Section V-B ensures that the resources (including bandwidth, on-chip memory and computational resources) occupied by each design point are fully utilized, and it ensures that the execution delay is always smaller for the same amount of total resource budget, further guaranteeing low energy consumption. Therefore, compared to "MIP-Random" and "MIP-Baye", the AutoSeg co-design engine, which can be described as "MIP-Heuristic", alway generates much-lower-energy solution. For example, in Fig. 18(a), the maximum energy consumption of the design points obtained by "MIP-Random" and "MIP-Baye" are $7.7 \times 1e10pJ$ and $5.7 \times 1e10pJ$, respectively. However, the maximum energy consumption of the design points discovered by AutoSeg is only $0.8 \times 1e10pJ$.

2) The MIP-based model segmentation method is very fast and efficient, and it guarantees the best segmentation solution for all the targeted models. However, it is difficult to obtain the results with the same-level performance by running other algorithms. For example, to fully show the strength of MIP-based segmentation method, we also tested the "Baye-Heuristic" method, which entails running many iterations of search in the model segmentation design space but still receives inferior solutions than the AutoSeg framework. As shown in Fig. 18(d), the best design point of our co-design engine achieves 6× speedup than that of "Baye-Heuristic".

3) In our co-design engine, model segmentation returns the segmentation strategy as well as the according CTC ratio and operational distributions, which contains richer system-level information than the direct latency/throughput feedback only, and feed them to guide the hardware parameters search. On the contrary, prior works [36], [60], [75] generally treat the HW/SW co-search as a two nested optimization loops. For example, in the "Baye-Baye" method, which is a bi-loop bayesian algorithm modified from [60], the outer bayesian loop generates hardware parameters, while the inner bayesian loop generates model segmentation strategies according to the hardware parameters from the outer loop. The model segmentation loop feeds back evaluation results, e.g., latency, to the hardware generation loop which takes the next batch of samples based on the feedback. Due to insufficient feedback information, this bi-loop method will easily fall into local optimization. As shown in Fig. 18 our co-design engine achieves 1.03-6.42× speedup than "Baye-Baye" method.

H. The Impact of Dataflow-Hybrid PUs

We also collect the on-chip memory energy consumed by DNN models running different dataflows, since it is known that the biggest difference in dataflows is their data reuse patterns and the induced moving overhead. Fig. 19 shows the evaluation results for two baseline SPA designs. The "WS-only" bars show the results of using WS dataflow consistently across all the layers while the "OS-only" bars show the results of using OS dataflow only. The "Hybird" bars show the results of using reconfigurable dataflow-hybrid PUs as instructed by Alg. 1, which is observed to outperform the "WS-only" and "OS-only" solutions on all



(a) AlexNet, NVDLA-Large. (b) MobileNet, NVDLA-Large. (c) AlexNet, EdgeTPU. (d) MobileNet, EdgeTPU.

Figure 18. The performance of discovered solutions across a suite of optimization methods on different hardware constraints. Each point represents a SPA solution including a hardware design and the related segmentation strategy.

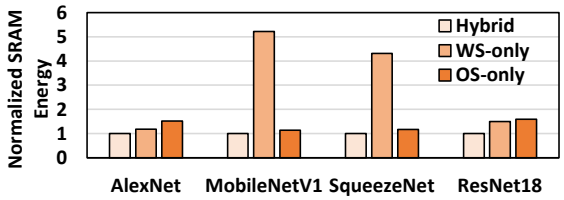


Figure 19. On-chip data moving cost of different dataflows.

models. The layers with large-size weights usually prefer the WS dataflow while layers with larger fmaps usually prefer the OS dataflow. Therefore, "OS-only" performs well in MobileNetV1 and SqueezeNet, while "WS-only" performs better in AlexNet and ResNet18. Our dataflow-hybrid design can embrace the strengths of both WS and OS at run-time.

VII. Related Work

DNN Accelerator. Generic DNN accelerators [5], [7], [8], [9], [22], [28], [37], [39], [50], [51], [80] rely on one uniform PUs to execute all layers of the DNN models sequentially. In theory, all the resources are available to each layer to minimize the execution latency, but they have low resource utilization due to various of layers. Furthermore, for small buffers, intermediate data needs to be written back to off-chip memory resulting in worse bandwidth bottlenecks. [27], [32], [35], [59] proposed to use heterogeneous PUs to process different layers for higher utilization. However, they perform coarse-grained frame-based parallelism instead of piece-based flow. From the perspective of a single image frame, different layers still run in non-overlapping order as in the no-pipeline architectures, and intermediate data is still written back to the off-chip memory with no bandwidth and energy saving. Fused-layer [1] practises fine-grained pyramid-based pipelined parallelism to reduce the off-chip memory access to intermediate data, but requires additional on-chip memory overhead for overlapped features. [46], [69], [74], [83], [84] use the circular line buffer to store fmaps for each layer to avoid buffer space consumption and special efforts on data management as [1]. [72] proposed a pipeline architecture as the fixed-weight feature extractor to run the front-end layers of a MobileNet-based model and then used transfer learning for different tasks. However, such pipelines lack of programmability and scalability, and their implementation often leads to a very high resource budget. [12], [14], [57], [65], [66] proposes homogeneous multi-cores for building large-scale systems that achieve coarse-grained parallelism in NNs. [12] realize inter-layer pipelines over a customized NoC through customized dataflow. However, they are not customized computational pipelines and also suffer from low pipeline resource utilization. Sparse DNN accelerators, e.g., [7], [15], [17], [41], [50], improved NN efficiency by avoiding redundant operations and memory footprints, which are orthogonal to our design.

HW/SW Exploration for DNN Acceleration. [13], [26], [54], [58], [70], [71], [77], [78], [82], etc., automatically generate no-pipeline DNN accelerators with a unified PU for all layers of the targeted model. [6], [16], [27], [34], [83], etc., propose hardware exploration methods to generate fully pipelined accelerators dedicating one specialized PU to each layer of the model. ConfuciuX [29] search for the best resources assignment for different DNN layers for both no-pipeline and full-pipeline architectures. For co-design automation, [30], [36], [60], [67], [75], etc., co-optimizes the hardware and software (layer-wise mapping) solution. However, there is no hardware exploration method that can generate the shared pipeline accelerator for all segments of the model, neither the co-optimization approach for hardware customization and segment-wise mapping.

Efficient Neural Network. Previous works proposed model pruning [18], [20], [21], [40] and quantization [18], [53], [55], [68] to improve model efficiency by compressing an model, or used neural architecture search (NAS) [3], [10], [38], [86] to design efficient models automatically. They are orthogonal to our design, and their insights should be useful for SPA as well.

VIII. Conclusion

Though customized DNN accelerators can dramatically improve the performance and energy efficiency of DNN

workloads, they face fundamental trade-offs: the resource under-utilization for no-pipeline architectures and the resource scalability for full pipeline architecture. To address this issue, we propose Segment-grained Pipeline Architecture (SPA) and an end-to-end automation framework, AutoSeg, which includes a parameterized SPA accelerator template and a co-design engine that generate the optimized segmentation strategies and SPA accelerators for the acceleration workloads. When compared to both SOTA no-pipeline and full pipeline accelerators, the accelerator solution generated by AutoSeg achieves significant performance boost.

ACKNOWLEDGMENT

This paper is supported in part by the National Natural Science Foundation of China (NSFC) under grant No.(61874124, 61876173, 62222411), Zhejiang Lab under Grants 2021PC0AC01. The corresponding author is Ying Wang.

REFERENCES

[1] 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.

[2] S. Arora, F. T. Leighton, and B. M. Maggs, "On-line algorithms for path selection in a nonblocking network," *SIAM Journal on Computing*, vol. 25, no. 3, pp. 600–625, 1996.

[3] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021.

[4] 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.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[6] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[7] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.

[9] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[10] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on xilinx devices," *White Paper*, 2016.

[12] 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.

[13] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[14] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 681–697.

[15] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019, pp. 151–165. [Online]. Available: https://doi.org/10.1145/3352460.3358291

[16] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Maloc: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2601–2612, 2018.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[19] 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.

[20] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[21] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[22] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 319–333.

[23] 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.

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[26] L. Jia, Z. Luo, L. Lu, and Y. Liang, "Tensorlib: A spatial accelerator generation framework for tensor algebra," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 865–870.

[27] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, "Heterogeneous fpga-based cost-optimal design for timing-constrained cnns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2542–2554, 2018.

[28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. ACM New York, NY, USA, 2017, pp. 1–12.

[29] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 622–636.

[30] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[32] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[33] T. T. Lee and S.-Y. Liew, "Parallel routing algorithms in benes-clos networks," in *Proceedings of IEEE INFOCOM'96. Conference on Computer Communications*, vol. 1. IEEE, 1996, pp. 279–286.

[34] 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.

[35] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[36] Y. Lin, M. Yang, and S. Han, "Naas: Neural accelerator architecture search," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1051–1056.

[37] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 369–381, 2015.

[38] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[39] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 393–405.

[40] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3296–3305.

[41] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 573–586.

[42] G. LLC, "Edge tpu," accessed on 2021-07-20. [Online]. Available: https://cloud.google.com/edge-tpu

[43] ——, "Edge tpu benchmarks," accessed on 2021-07-20. [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/

[44] G. O. LLC, "Gurobi optimizer reference manual," accessed on 2021-04-30. [Online]. Available: https://www.gurobi.com

[45] Y. Meng, S. Kuppannagari, R. Kannan, and V. Prasanna, "Dynamap: Dynamic algorithm mapping framework for low latency cnn inference," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 183–193.

[46] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in fpga design for cnn-based object detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450–2464, 2020.

[47] NVIDIA Corporation, "Nvidia deep learning accelerator," accessed on 2021-07-20. [Online]. Available: http://nvdla.org/

[48] ——, "Nvidia developer," accessed on 2021-07-20. [Online]. Available: https://developer.nvidia.com/blog/nvdla/

[49] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[50] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.

[51] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 389–402.

[52] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[53] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[54] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 267–278.

[55] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 326–335, 2020.

[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] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, and C. T. Gray, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[58] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[59] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 535–547.

[60] Z. Shi, C. Sakhuja, M. Hashemi, K. Swersky, and C. Lin, "Learned hardware/software co-design of neural accelerators," *arXiv preprint arXiv:2010.02075*, 2020.

[61] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[62] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[63] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

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

[65] Z. Tan, H. Cai, R. Dong, and K. Ma, "Nn-baton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1013–1026.

[66] 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.

[67] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[68] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[69] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "Tgpa: tile-grained pipeline architecture for low latency cnn inference," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[70] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[71] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 268–281.

[72] P. Whatmough, C. Zhou, P. Hansen, S. Venkataramanaiah, J.-s. Seo, and M. Mattina, "Fixynn: Energy-efficient real-time mobile computer vision hardware acceleration via transfer learning," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 107–119, 2019.

[73] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[74] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[75] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, "Hasco: Towards agile hardware and software co-design for tensor computation," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1055–1068.

[76] Xilinx, "Dpu," accessed on 2021-07-20. [Online]. Available: https://github.com/Xilinx/Vitis-AI

[77] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 40–50.

[78] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[79] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "Hybrid-dnn: A framework for high-performance hybrid dnn accelerator design and implementation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[80] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, 2017.

[81] Y. Yu, T. Zhao, K. Wang, and L. He, "Light-opu: An fpga-based overlay processor for lightweight convolutional neural networks," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 122–132.

[82] D. Zhang, S. Huda, E. Songhori, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack accelerator search technique for vision applications," *arXiv preprint arXiv:2105.12842*, 2021.

[83] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[84] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[85] S. Zheng, X. Zhang, D. Ou, S. Tang, L. Liu, S. Wei, and S. Yin, "Efficient scheduling of irregular network structures on cnn accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3408–3419, 2020.

[86] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.