Survey on Landmark DNN Accelerators and Design Processes in Academia

Victor Martin Agostinelli III, Indhu Priya Karunagaran,
Diksha Pritam Todankar, and Kazi Ahmed Asif Fuad

March 2022

1 Introduction

Machine learning solutions are increasingly popular and applicable, but with the slowing down of Moore's law comes a necessity for dedicated hardware to accelerate those solutions further. Machine learning accelerators are, as such, a hot area of both academic and commercial interest. Companies like Tesla and Google are using dedicated hardware to enable tasks such as real-time vision for autonomous vehicles or translation between disparate languages. In parallel, academic groups across the world are focused on exploring the relatively young design space for these accelerators. Developments in this space are constant, exciting, and have resulted in designs that operate quite differently from one another, in many cases with very clear trade-offs between them in terms of power consumed, area taken up, and speed.

This survey seeks to acquire a holistic view of the deep neural network (DNN) accelerator design space, some of the techniques that state of the art accelerators are being developed with, and a sense of which modern architectures are relevant and how they compare with one another. To this end, a number of papers published in relevant machine learning journals and at specific conferences must be engaged with alongside notable commercial options. Indeed, this survey will be limited to those accelerator architectures being developed by academically focused groups, as vastly more detailed literature exists on those designs and the processes that went into creating them. Several accelerator designs have already been identified as particularly promising, including Eyeriss from the Massachusetts Institute of Technology [5], FlexFlow from the Chinese Academy of Sciences [2], and MAERI from the Georgia Institute of Technology [4], and will be included in this survey's comparisons. In addition to this, some limited simulation lies within the scope of this survey, extracting performance and efficiency details from these designs using cutting edge accelerator evaluation tools [6, 7].

2 Accelerator Design Background

A few key concepts should be kept in mind that apply to most, if not all, DNN accelerators, including the typical components of a machine learning accelerator, the design stages for these accelerators, and how useful metrics can be extracted from a given piece of dedicated hardware that allow for meaningful comparisons to other architectures.

As far as components go, while this can and does vary on a case to case basis, the bulk of most accelerators is typically dedicated to a systolic array of processing engines or processing elements (PEs). These PEs are capable of accelerating typical arithmetic operations that a specific accelerator might need to often deal with, with one incredibly common example being dedicated hardware within a PE for multiply-and-accumulate operations (MACs). Tremendous energy efficiency gains can be acquired in the careful use of the array of PEs in a given accelerator, making their efficient arithmetic for some model or task extremely important [1]. In addition, many accelerators feature on-board buffers for less costly memory accesses, when necessary. These buffers allow for a given accelerator to reuse specific pieces of data when it would be too costly in terms of time or energy efficiency to pull it from off-chip memory.

The traditional design flow for a machine learning accelerator typically follows as such: the specification of a dataflow, the selection of a mapping strategy, and the allocation of hardware resources [6]. The dataflow for a given accelerator usually defines the accelerator's data reuse strategy in addition to defining some strategies for high PE array utilization that are beyond the scope of this survey [6]. Choosing a dataflow is typically done either through heuristics, brute force search of a very limited design space, or selection by experts [6]. The mapping strategy details how resources are mapped when a given machine learning model demands more PEs or buffer space than the hardware allows, which can be common for large networks or accelerators on the edge [7]. It is often wrapped together with the process of hardware resource allocation, as its design space is quite small. In line with that, hardware resource allocation is, ultimately, the partitioning of available buffer space and PEs according to the model's computational needs and is accomplished through DRL-based methods when it comes to cutting edge solutions, although such developments are still nascent [7].

When attempting to extract useful metrics, the first set of metrics that appear as fairly universal for machine learning accelerators would be power efficiency, performance for a given model and benchmark, and physical size [6]. The final metric is a simple one to measure and make comparisons with, but the first two can be more difficult to extract, especially if a team of architects is attempting to decide between suitable accelerator architectures before design-time and must simulate. Fortunately, a few academic groups have worked to provide tools that are capable of estimating one or both of these metrics, including MAESTRO [6], a novel tool developed by a group at the Georgia Institute of Technology.

3 Literature Review

3.1 ShiDianNao

Yet another group focused on highly efficient accelerator designs, the authors of ShiDianNao [8] honed in on an architecture specializing in efficient image recognition, capable of being implemented on the edge alongside a sensor array. Correspondingly, the models that they were capable of supporting were generally much more compact, with far fewer parameters expected and a requirement of all memory requests being answered on-chip as opposed to allowing access to some off-chip memory. Especially in regards to image processing on the edge, the authors justified expected drops in computational performance by considering sensor bandwidth and throughput to be a limiting factor, thus rendering larger computational blocks unnecessary [8].

The comparison of ShiDianNao to other pieces of hardware capable of running the aforementioned DNN and CNN models that this group was focused on was somewhat lacking, unfortunately. They compared this architecture primarily to one of their previous designs, DianNao [11], in addition to some general purpose hardware and found, unsurprisingly, that when normalized compared to the resources their previous design had, ShiDianNao outperformed both comparable general purpose hardware and their previous design on nearly all workloads, failing to its predecessor only where raw throughput was concerned [8]. Energy savings abounded with this design, as expected.

3.2 Eyeriss

While a number of CNN accelerator architectures have sought out extreme energy efficiency, Eyeriss remains interesting as the first to propose a rather novel approach: making use of a row-stationary dataflow [5]. Whereas previous architectures traditionally focused on three dataflows built on minimizing weight, input, or output accesses for CNN accelerators, a row-stationary dataflow differs in that it approaches tasks adaptively based on data reuse opportunities and hardware allocation for a given model and its layers. This dataflow depends on an efficient mapping process, however, for individual computations across the systolic array of PEs. The research group at MIT that built Eyeriss, in collaboration with NVIDIA, focused on breaking down the convolutional layers of a given CNN model into what they call "1D convolution primitives" that they then mapped over the course of a two-step process [5]. The first of those steps involved mapping the convolution primitives to a logical array of PEs, with that logical array typically being significantly larger than what is physically available in hardware. Following efficient logical mapping, physical mapping was necessary, taking logical PEs and scheduling their tasks on physical PEs such that, oftentimes, a single physical PE would execute the computations of a "row" of logical PEs that would handle a given 1D convolution primitive.

The evaluation of the Eyeriss architecture and its row-stationary dataflow was focused on comparisons to other dataflow options for CNN accelerators, featuring the more traditional weight, input, and output stationary dataflows in this case, on AlexNet, a CNN that was popular during its inception for accurate object recognition in computer vision tasks [9] and with consistent hardware and parallelism opportunities. Across the board, Eyeriss and its dataflow selection was found to access DRAM less often (meaning that less energy and time was wasted on costly memory acesses) and use less energy per operation than its competitors [5].

3.3 FlexFlow

Authors of FlexFlow [2], Wenyan Lu *et al.* proposed a flexible dataflow architecture capable of mitigating the mismatch between the varying dominant parallel types of CNN and computing engine supported parallel types. There are three types of possible parallelism: synapse parallelism (SP), neuron parallelism (NP), and feature map parallelism (FP), that can be explored to design CNN accelerator efficiently. Three prominent architectures: Systolic, 2D-Mapping, and Tiling are individually efficient at supporting SP, NP, and FP respectively. However, these architectures fail to explore parallelisms efficiently as the number of feature maps, the number of output feature maps, the size of output feature map and the size of kernel varies dramatically in different layers of CNNs. Modified proposed architecture is shown in Figure 1.

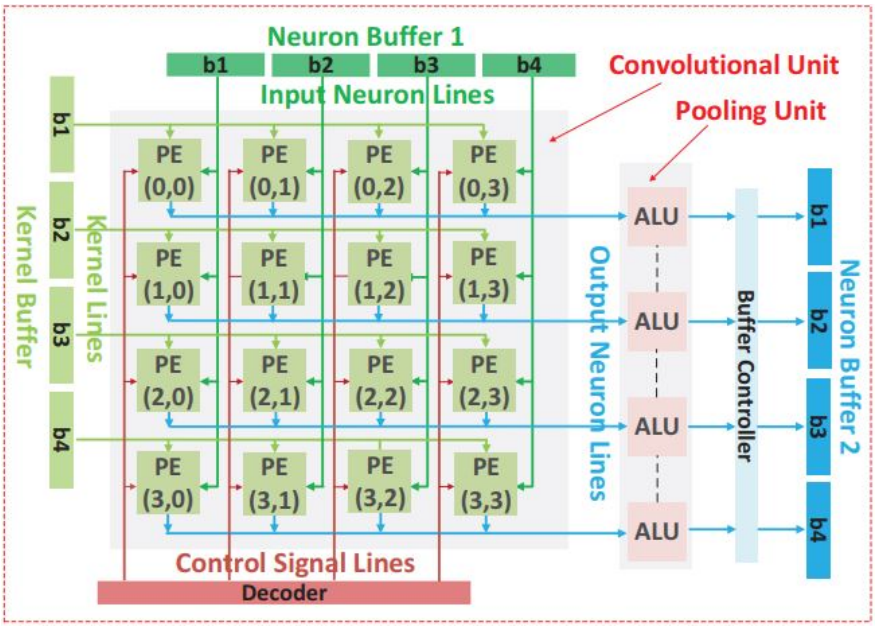

Figure 1: Flexflow Architecture [2]

This arrangement eliminates dependency between adjacent PEs and enables the Multiple Features Multiple Neurons Multiple Synapses (MFMNMS) style of processing as defined by the author. This processing style signifies that the architecture handles multiple feature maps, multiple neurons, and multiple synapses of each kernel at a time. The evaluation is carried out on resource utilization, power, energy, area, data scalability and data reusability. FlexFlow is compared with Systolic, 2D-Mapping and Tiling architectures. The workloads used are PV (pedestrians and vehicles recognition), FR (face recognition), LeNet-5, HG (hand gesture recognition), AlexNet and VGG. Although FlexFlow implementation requires slightly larger area, the computing resource utilization of FlexFlow is 80% compared to the baseline architectures' max 60% utilization for mixture of workloads. It achieves 2x performance and power efficiency speed up compared to Systolic and 2D Mapping whereas 10x for Tiling while maintaining high scalability [2].

3.4 Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators

DNNs are comprised primarily of MAC operations. Most of their performance contains multichannel filters with multichannel feature maps. Due to usage of MAC operations there are few dependencies between datapoints, where DNN accelerators can use parallelism at high phase to achieve high performance. This process requires a specific amount of data movement, and every MAC present performs with one write and three reads data access. Since the key goal is to achieve high energy efficiency optimization, movement of data is done when useful rather than freshly computing as the latter process often consumes more energy. To further explain relevant topics in this paper, the authors resort to comparing compilation of code between general purpose processors and DNN accelerators[1].

The difference is that a compiler is used to translate into machine readable binary codes for general purpose processors, while for a DNN accelerator the mapper is used in place of a compiler to do the translation. This operation of using a mapper is critical in that it optimizes energy efficiency. Data storage organization also varies between them. Considering evaluating energy consumption, optimal mapping is required. The authors of this paper evaluate energy consumption based on spatial architectures. These spatial architecture consist of multilevel storage hierarchies and the array of PEs in a given accelerator. With a specific mapping and given the energy system, energy consumption can be estimated by the number of times each data type is reused at each level of the entire four-level hierarchy memory[1].

In this paper there is a taxonomy of the existing DNN dataflows, including Weight-Stationary (WS) dataflows which keep the filter weight stationary in each PE. In addition, there exist Output-Stationary(OS) dataflows, which keep partial sums stationary by adding or accumulating them in the RF of individual PEs. Finally, No-Local-Reuse(NLR) dataflows are covered, where there is no stationary data so it can directly trade the RF off for a larger global buffer. An energy efficient dataflow describes DNN accelerators as they optimize data movement for few data types. A newer dataflow, mentioned previously in this survey, is called a Row-Stationary dataflow due to the fact that it divides the overall MAC operations and comprises a subset of MACs that can run on the same PE in a fixed order. Row primitives are composed for each 1D row convolution to optimize data reuse in each PE. Each row primitive is formed by a set of rules which has to be followed for processing the combination. After processing the content with the specified rules, the RS dataflow provides a high degree of flexibility in mapping, like concatenation, interleaving, duplicating, and folding of the row primitives. Dataflow comparison is done by considering two constraints. First, the size of the PE array is fixed at 256 for constant processing along dataflows. Next, the area of the total hardware is fixed. By comparison they observe RS dataflow is 1.4 to 2.5 times more efficient than other dataflows in the CONV layers of AlexNet. This process of dataflow selection is useful for energy efficiency optimization, as covered by the authors of this paper, but is also critical for any sort of optimization depending on the application and goals of a given design[1].

3.5 MAERI

In contrast to a number of other designs, MAERI is focused on flexibility above all. Whereas CNN accelerators often have to select a single dataflow to specialize in, MAERI was built with reconfigurable interconnects, meaning a flexible network on chip (NoC), between its PEs, allowing for a flexible transition of data across the array [4]. Interestingly, this adaptability allows for mapping across highly irregular models. Should optimization for a given model require a non-traditional and unusually structured dataflow, MAERI is capable of supporting such a network. While this flexibility is important in its own right, aiding multi-purpose accelerator design, the team behind MAERI was also focused on underlining that the overhead for MAERI's NoC was minimal and that it performed competitively compared to other options. The differences in mapping structure between a more traditional DNN accelerator and MAERI is underscored by Figure 2 [4], where the multiplier switches in their interconnect are effectively mapped to as opposed to individual PEs.

The team behind MAERI evaluated the architecture by comparing it specifically to Eyeriss and even more traditional systolic array structures. They found that, generally, MAERI was a more compact architecture as its PEs tended to require fewer elements than those of Eyeriss (a simpler RF structure, separation between types of PEs for adder switches, multipler switches, simple switches, etc.). Holding the number of PEs constant or the area constant, MAERI proved to hold advantages over Eyeriss related primarily to latency,

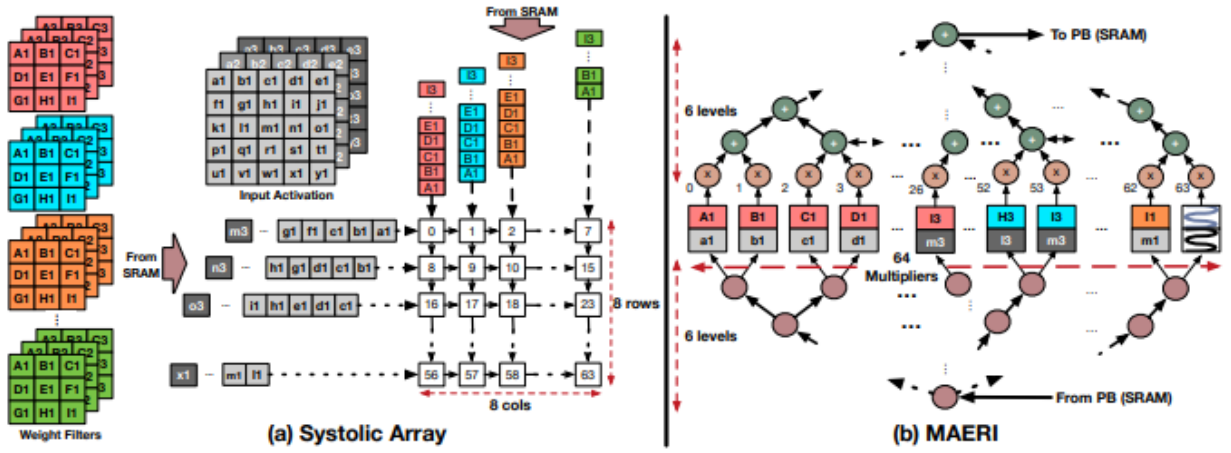

Figure 2: Comparison Between Mapping CONV Layer to Systolic Array vs. MAERI [4]

as MAERI is relatively power-hungry. The team behind MAERI attributes this primarily to the design's very high levels of utilization, as their mapping of a given model can be relatively irregular in comparison to Eyeriss and other designs.

3.6 Energy-Efficient Convolution Architecture Based on Rescheduled Dataflow

Rescheduled dataflow convolution has its hardware architecture with a focus on energy efficiency. For involving a large amount of memory access and computation, previous accelerators use parallel processing elements to overcome real-time constraints. Convolutional neural networks (CNN) have been used for various applications due to their unprecedented accuracy, particularly related to object recognition as they can discriminate between hundreds of object. This paper analyzes 1D and 2D convolution operations and on-chip memory access patterns of two state-of the-art CNN accelerators. It continues by analyzing and renovating the dataflow of aforementioned convolutions and summarizes the most suitable CNN architecture which can maximize energy efficiency by reducing redundant on-chip memory access. Since the filtering operation is repeated hierarchically, the CNN achieves a high recognition rate. According to this filtering, the convolution operation makes various effects on the image input such as blurring, sharpening, and edge detection. By using accumulation, combination, and convolution products an output image is achieved. The process that takes place is the accelerators employs parallel multipliers to compute relevant products to an output cycle. The adder tree accumulates the product to a fixed number to support limited convolution operations. By the row-stationary dataflow structure, each PE computes partial sum relevant to results and row of weights aggregated by passing to the upper PEs. A rescheduled dependence dataflow comes into the picture to reduce the energy overhead consumed by on-chip memory access. Few changes are made to reschedule the data flow. The number of steps is directly proportional to the size of the input feature by using OM. By accumulating the partial sums incrementally has the input data loaded only once from IN, which enhance the overall energy efficiency. Output features can be calculated by gradually accessing IM only once and computing the products related to the input at once.We get to know that the significant portion of the energy is mainly taken in on-chip memory access,hence the proposed accelerator adopting this rescheduled dataflow is effective in enhancing the energy efficiency. To ensure the energy efficiency was accurate, an evaluation was done using 65nm CMOS processor and designed prototype accelerator [3].

3.7 ConfuciuX and MAESTRO

Deep Neural Network accelerator architecture designs are dominated by two main components, dataflow style and allocated total hardware resources. Dataflow utilizes reuse of activations, weights, and outputs in DNNs by employing computation order, parallelization order, and tiling strategies. Usual hardware resources are on-chip processing elements and memories. It is possible to allocate hardware resources in many ways to

optimize a particular dataflow architecture of a given DNN model. Recent research shows, an optimized HW resource allocation is more important for DNN performance that it's dataflow architecture [10, 7]. Authors of ConfuciuX [7], Sheng-Chun Kao et al. developed an autonomous method to allocate hardware resources for a given model and dataflow by employing a combination of reinforcement learning method, REINFORCE for global search and genetic algorithm for fine tuning. ConfuciuX takes a DNN model, the deployment scenario, the optimizing objective, and the platform constraints and it generates an optimized hardware allocation. During training, the RL agent continuously generates hardware resources: PE and buffers as "actions" which are evaluated by an analytical model such as MAESTRO [6]. MAESTRO works as an environment that outputs different statistical matrices as "rewards" which are used to train the underlying policy network to maximize the "reward". Later, platform constraints such as area or power punishes actions if those violate the constraints. This proposed method achieves optimized hardware configuration 4.7 to 24 times faster than existing other techniques at that time [7] as it uses RL to coarse-grained search for a global optimized allocation and genetic algorithm for fin-grained local optimization.

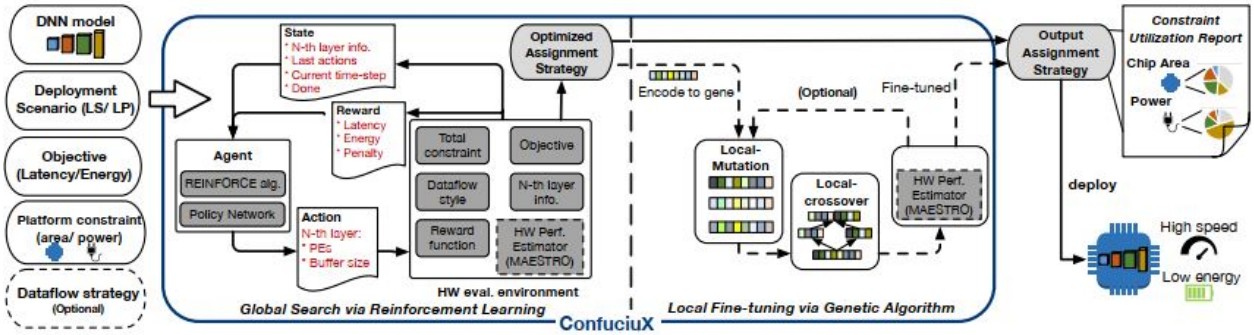

Figure 3: Overview of ConfuciuX [7]

Modeling Accelerator Efficiency via Spatio-Temporal Resource Occupancy (MAESTRO) [6] is a systematic data reuse based analytical cost model that provides more than 20 statistical metrics to model and evaluate DNN accelerator design space. This cost-benefit model is developed based on data-centric directives that allow accommodating all three crucial factors: DNN layers, hardware and mapping with precise data reuse. This framework was validated with cycle accurate RTL simulation for MAERI [4] and Eyeriss [5] accelerators and results were within 3.9% absolute error. Moreover, it provides fast estimation, taking only 493 ms to analyze the layers of Resent50 on a 256PE NVDLA style accelerator with Core-i9 CPU and 16 GB RAM.

4 Simulation Methodology

It is important to ground the above models in some comparative context and to give some insight into the design process for these accelerators. While the above Literature Review works to provide context for the majority of the design process, the iteration that is involved with evaluation and then adjustment of design parameters is not adequately shown. As such, two dataflows are compared using ConfuciuX and MAESTRO [7, 6] to evaluate their latency, area, and efficiency when applied to an optimized accelerator architecture. The team behind these evaluation tools supports both Eyeriss and ShiDianNao's dataflows [5, 8], providing dataflow mapping for these architectures that their tool suite is capable of reading. Comparing the fitness of other architectures would require constructing that dataflow mapping in a form that ConfuciuX and MAESTRO can comprehend, and this is beyond the scope of this survey. Nonetheless, these two serve as particular excellent test cases, as they are each rather specifically tailored for certain applications (ShiDianNao works best for compact models and is built for the edge whereas Eyeriss is a bit more general and employs a novel dataflow).

To ensure that they are fairly compared, they will each be optimized for latency, area, average throughput, and power consumption with identical constraints (max area, max power when optimizing for area). We're

Platform Constraint	Descriptions
Unlimited	No constraint. Since we set each action to 12-level, we measured the maximum constraint (power/area) consumption, $C^{max}_{power/area}$, by evaluating entire model with uniform action pair (p_{12th}, b_{12th}).
Cloud	Loose constraint. We set the constraint at *50%* $C^{max}_{power/area}$.
IoT	Tight constraint. We set the constraint at *10%* $C^{max}_{power/area}$.
Extreme small IoT (IoTx)	Extremely tight constraint. We set the constraint at *5%* $C^{max}_{power/area}$.

Figure 4: Constraint Initialization Table from ConfuciuX

setting the maximum constraint provided by our model input to 50% of its possible value, assuming some cloud-level platform constraints as provided by Figure 4 [7]. Each will see their HW resource allocation optimized over 60 epochs for the course-grained search of ConfuciuX and over 40 generations for the fine-grained genetic algorithm or until the tool suite judges that no more optimizations are possible or likely to be found. While the number of epochs and generations appears low, we found that in many runs, the difference between the fitness of 100 total epochs and 500 total epochs was no more than a 5% improvement, and the tool suite appeared to be less prone to seemingly random errors during the genetic search. Since we are not considering our simulations here to be exhaustive and complete comparisons and are instead using the results to provide insight into the design process of these accelerators, slightly less accurate results are acceptable. Three models will be used for this set of tests, VGG16, ResNet18, and ResNet50, to get a sense of how well suited each architecture is for varying models. We expect that the comparison of ResNet18 and ResNet50 will be especially interesting, given that ShiDianNao was built with the expectation that it would be applied towards compact models.

While this is unlikely to be an entirely realistic comparison, given that the two architectures were built with different technologies, resources, and applications in mind, it should still lend some insight as to why a designer would choose one or the other and how they might use that information at design time to assist in building the architecture of their accelerator.

5 Analysis of Results

Fitness Goal: Average Throughput
Constraint: Area

Model	Eyeriss	ShiDianNao
VGG16	78.9959%	99.7107%
ResNet18	78.1106%	97.2920%
ResNet50	75.6028%	90.0793%

Fitness Goal: Area
Constraint: Power

Model	Eyeriss	ShiDianNao
VGG16	4.4692%	4.4692%
ResNet18	4.5445%	4.4692%
ResNet50	5.0369%	4.4692%

Fitness Goal: Latency
Constraint: Area

Model	Eyeriss	ShiDianNao
VGG16	98.9005%	98.7847%
ResNet18	95.4865%	97.7431%
ResNet50	55.8749%	98.3109%

Fitness Goal: Power
Constraint: Area

Model	Eyeriss	ShiDianNao
VGG16	4.4692%	4.4692%
ResNet18	4.4692%	4.5445%
ResNet50	7.1658%	4.4692%

Table 1: Percentage Constraint (Area or Power) Used for Throughput Average, Area, Latency, and Power Optimization (from left to right, top to bottom.)

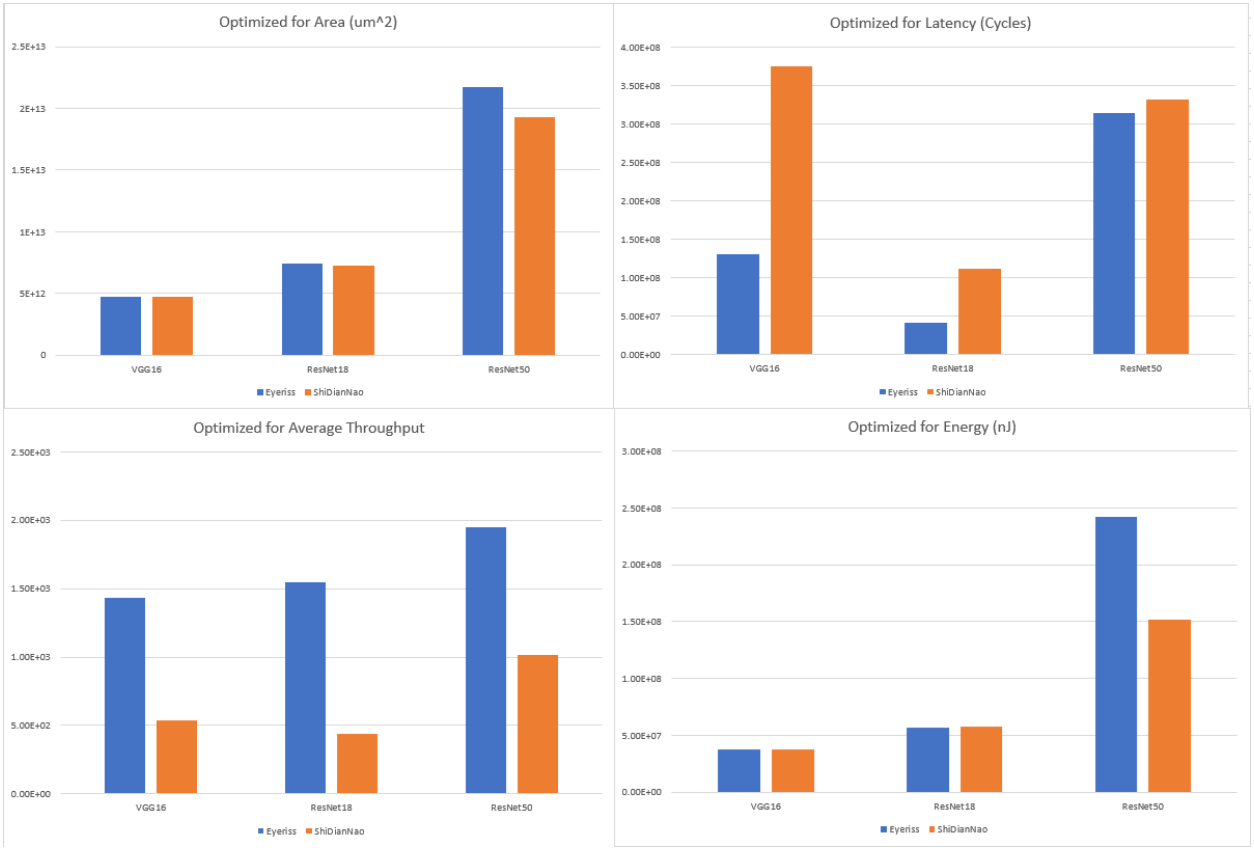

Figure 5: Results from Various Optimization Runs

As a brief note, unfortunately the units for most of these metrics are uncertain. Nowhere in the repository that contains these tools are the associated units listed, and it is not entirely clear if the results in their associated papers are formatted in the same way as our own or otherwise processed. The authors of this survey reached out to the research group behind ConfuciuX and MAESTRO for assistance on this issue, but no response was received. A best guess for each metric is included, although we have no good guess for average throughput. Additionally, it is worth noting that while the fitness goal is specified as "power" in the ConfuciuX repository, the paper that introduced it characterized an energy footprint in nJ. It is assumed that this is the result that was given by this survey's set of simulations.

From Figure 5 we can see that for essentially all models, when optimized for a metric that would commonly be associated with performance like latency and average throughput, Eyeriss tends to vastly outperform ShiDianNao. This aligns with our expectations based on academic papers for these two dataflows and architectures, as ShiDianNao is built for the edge and built to be compact, not necessarily for superior performance. This is observed in spite of the fact that the strict constraints applied to ShiDianNao don't necessarily carry over to these simulations with ConfuciuX and with MAESTRO's cost model (entire model in memory, no off-chip accesses, etc.), which is certainly interesting. Additionally, it can be observed that when optimized for area and power, while ShiDianNao is performing strictly better with these fitness goals the difference does not appear to be as steep except for models with a very large number of layers, like ResNet50. Once again, this aligns with our expectations related to these dataflows.

That being said, it is worth also keeping in mind how much of our constraint range we're using for each of these dataflows when optimized for different goals across different models. An architecture that performs slightly better in regards to latency, for example, may still be evaluated as inferior to an architecture that, when optimized for latency, is slightly slower but much smaller. In line with that, the constraint usage found in Table 1 related to average throughput and latency are fascinating. These are already areas where the dataflow associated with Eyeriss already showcases superior performance, but these metrics were also extracted with, in many cases, a severely reduced area footprint compared to ShiDianNao's dataflow. In regards to latency, most models resulted in similar constraint usage but ResNet50 was particularly notable, with Eyeriss' dataflow only using approximately 56% of its constraint and ShiDianNao's dataflow using approximately 98%. While the difference was less stark for average throughput, the differences are still significant. It's possible that the number of epochs for the design space search could have caused this disparity, but it is unlikely that these differences would be observed across so many categories if they did not represent the strengths and weaknesses of each dataflow to some degree.

6 Conclusion

In this survey, a number of design methodologies, dataflows, accelerator architectures, and evaluation procedures were showcased that the authors of this survey consider to be landmarks within the field of DNN accelerator design. This survey spoke to designs that were critical to the development of the space, such as FlexFlow [2], Eyeriss [5], ShiDianNao [8], and MAERI [4]. Each offered important design point options, all with their own trade-offs for specific applications. ShiDianNao and Eyeriss established themselves with novel dataflows and tight, energy efficient architectures. FlexFlow and MAERI focused on flexibility in different ways, opening up avenues for accelerators that were a bit more generalizable.

Somewhat similarly, the papers this survey covered related to dataflow selection and rescheduling [1, 3] describe important design techniques. Energy efficiency and performance gains are heavily related to the execution of an efficient and appropriate dataflow. Approaching accelerator design with these techniques in mind, and, more importantly, the associated attitude towards achieving efficiency gains that these aforementioned papers highlight is critical for executing on architectural strategies.

Finally, the performance evaluation tools that were covered by this survey, namely MAESTRO and ConfuciuX [6, 7], are critical within a given designer's flow to efficiently iterate on their design point selection. Relying on expert impressions of how an accelerator should perform or simple heuristics is an outdated methodology, and heavy simulation is simply too slow, not to mention that the HW resource allocation space can be massive depending on the selected model and dataflow, making it inefficient to comb through by hand. While exhaustive testing with tools like MAESTRO or ConfuciuX can be impractical, they serve to provide a useful performance profile.

All of the above tools, dataflows, and techniques have served to widen the design space of DNN accelerators and accelerate performance and efficiency gains across the board. It is critical for any designer to be aware of the aforementioned papers as they are moving through their design flow, choosing their models carefully for their application, selecting a dataflow that best suites the needs of their model, and engaging with the mapping strategy and HW resource allocation space efficiently while using capable tool suites to extract useful performance details. The design space of DNN accelerators is so vast and swiftly growing that this survey is in no way comprehensive. The authors of this survey hope that a possible knowledge base has been highlighted in addition to an incredible number of opportunities within the DNN accelerator space.

References

[1] Y. Chen, J. Emer and V. Sze, "Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators," in IEEE Micro, vol. 37, no. 3, pp. 12-21, 2017, doi: 10.1109/MM.2017.54.

[2] W. Lu, G. Yan, J. Li, S. Gong, Y. Han and X. Li, "FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 553-564, doi: 10.1109/HPCA.2017.29.

[3] J. Jo, S. Kim and I. Park, "Energy-Efficient Convolution Architecture Based on Rescheduled Dataflow," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4196-4207, Dec. 2018, doi: 10.1109/TCSI.2018.2840092.

[4] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. SIGPLAN Not. 53, 2 (February 2018), 461–475. DOI:https://doi.org/10.1145/3296957.3173176.

[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. SIGARCH Comput. Archit. News 44, 3 (June 2016), 367–379. DOI:https://doi.org/10.1145/3007787.3001177.

[6] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer and A. Parashar, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," in IEEE Micro, vol. 40, no. 3, pp. 20-29, 1 May-June 2020, doi: 10.1109/MM.2020.2985963.

[7] S. -C. Kao, G. Jeong and T. Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 622-636, doi: 10.1109/MICRO50266.2020.00058.

[8] 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 ISCA, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in IEEE CVPR, 2016.

[10] X. Yang, M. Gao, J. Pu, A. Nayak, Q. Liu, S. E. Bell, J. O. Setter, K. Cao, H. Ha, C. Kozyrakis et al., "DNN dataflow choice is overrated," arXiv preprint arXiv:1809.04070, 2018.

[11] T. Chen, Z. Du, N. Sun, J. Wang, and C. Wu, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machinelearning," in Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Salt Lake City, UT, USA, 2014, pp. 269–284.

[12] 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.