Configurable Deep Learning Accelerator with Bitwise-accurate Training and Verification

Shien-Chun Luo, Kuo-Chiang Chang, Po-Wei Chen, and Zhao-Hong Chen
Electronic and Optoelectronic System Research Laboratories,
Industrial Technology Research Institute, HsinChu, Taiwan

***Abstract*—this paper introduces an end-to-end solution to a deep neural network (DNN) inference system. We customize and enrich a family of deep-learning accelerators (DLA) based on the NVIDIA open-source deep-learning accelerator (NVDLA). Our exclusive enhancement includes hardware and software parts. The hardware part is the shared multiplier array of both high-efficient regular and depth-wise convolutions. The software part includes a new DLA toolchain that generates various specifications of DLA and provides corresponding tests. Users can either verify trained or untrained DNN graphs to simulate the accuracy and performance results. Considering the hard-to-debug accuracy loss in the integer inference, a hardware-aware and bitwise-accurate flow are proposed. The DLAs were verified using FPGA prototypes and silicon chips. The 65nm test chip has 256 convolutional multipliers, and the peak computing power is 200 GOPs, achieving 2.5TOPs/W peak energy efficiency. The average power consumption is 65mW when running an object detection DNN model.**

Keywords—Accelerator, AI, ASIC, Compiler, Deep learning, FPGA, Neural Network, SoC

I. Introduction

Optimizing the edge inference of a deep neural network (DNN) challenges from the hardware efficiency, power consumption, to the device cost. It is also difficult to let all designers of algorithms, toolchains, and hardware compromise with each other. Designing algorithms for edge inference requires extra consideration about inference speed in addition to model accuracy. Hardware (HW) architects must enhance the processing parallelism of the mass DNN parameters, reuse dependent data in the convolution, and keep updating to new operators. The toolchain developer must link diverse DNN algorithms to the converged hardware accelerator and fully utilize the hardware acceleration. However, the development of specific accelerating HW is hard to catch up with the advance of DNN models. A fast and custom synthesizer to generate various accelerators can help, but the developing toolchain, including the pre-HW analysis tools and post-HW compilers are critical.

II. End-to-end Solution

A. *Overview of the solution*

Fig. 1 overviews our end-to-end solution of an integer-based deep-learning accelerator (DLA). The solution covers four main domains: (1) the HW performance analysis of DNN algorithms, to help select a proper HW specification (SPEC), (2) the toolchain for synthesizable HW code generation, (3) the compression process of DNN models for integer inference and HW-aware fusion, and (4) the compiler, runtime, and inference system prototypes. Part of this flow is revised from the open-source deep-learning accelerator, NVDLA [1]. We fulfilled and extended its hardware synthesizer because original sources on GitHub are incomplete with various

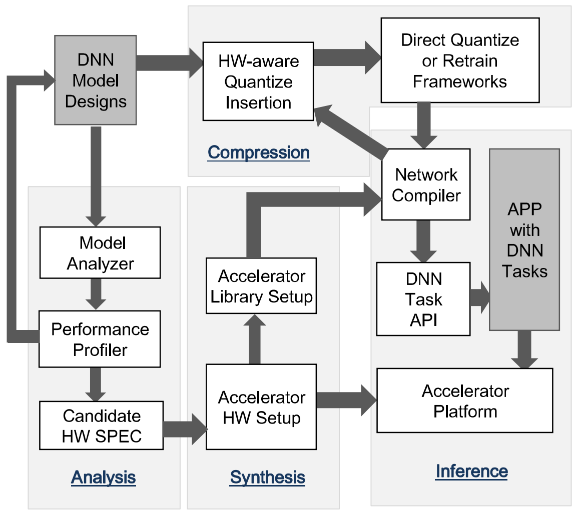

Fig. 1 Total solution to construct a DLA system.

shortages. After efforts, we make it ready to generate synthesizable Verilog codes and libraries needed by the toolchain. Besides original NVDLA hardware features, we propose a new architecture that supports both the regular and depth-wise (DW) separable convolution (CONV) [2]. The regular and DW CONV architectures share the same multiply array and CONV buffer, but they have different data sequencers and post-convolution data accumulators.

The end-to-end solution starts from selecting candidate DNN models, and a fast profiler estimates the hardware performance, including cycles, utilization, and frame per second (fps). Some candidate DLA SPECs are advised to the users, and the synthesizer generates hardware description Verilog codes and toolchain libraries accordingly. The toolchain supports real DNN models and pseudo, graph-only DNN models. The pseudo-DNN graphs verify the corner and edge cases of the hardware test. The toolchain consists of a DNN graph compiler, a pattern generator, and a golden result simulator. The DNN compiler optimizes the operators using remap, fusion, and partition. The pattern generator analyzes the dimension of each fusion operator and creates random test patterns. The golden result simulator, which is also a high-level simulator of DLA, accepts trained or random test patterns and calculates the results. The toolchain works with a novel bitwise-accurate quantization flow, which revises the conventional training process to have consistent accuracy with the integer DLA inference.

B. *Hardware Solution*

The DLA hardware architecture accelerates general convolutional neural network (CNN) operation, which is the convolution – normalization – pool sequence (Fig. 2). The regular and DW CONV operations share the same CONV

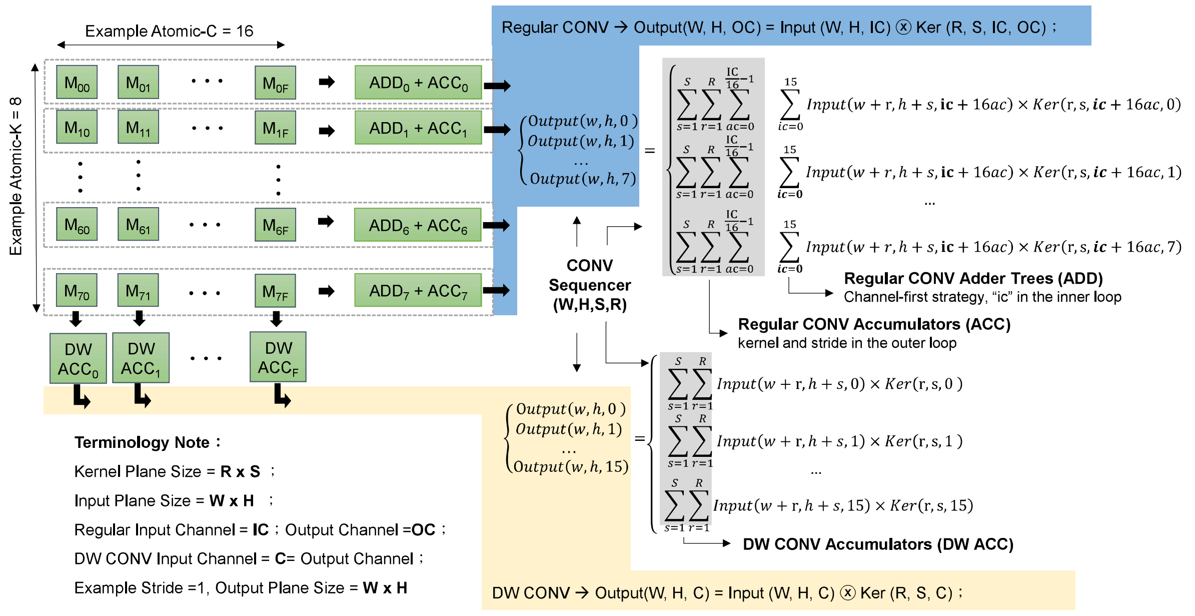
Example Atomic-C = 16
Example Atomic-K = 8
M00 M01 ... M0F ADD0 + ACC0
M10 M11 ... M1F ADD1 + ACC1
M60 M61 ... M6F ADD6 + ACC6
M70 M71 ... M7F ADD7 + ACC7
DW ACC0 DW ACC1 ... DW ACCF
Regular CONV → Output(W, H, OC) = Input (W, H, IC) ⊗ Ker (R, S, IC, OC) ;
CONV Sequencer (W,H,S,R)
Regular CONV Adder Trees (ADD)
Channel-first strategy, "ic" in the inner loop
Regular CONV Accumulators (ACC)
kernel and stride in the outer loop
DW CONV Accumulators (DW ACC)
Terminology Note :
Kernel Plane Size = R x S ;
Input Plane Size = W x H ;
Regular Input Channel = IC ; Output Channel =OC ;
DW CONV Input Channel = C= Output Channel ;
Example Stride =1, Output Plane Size = W x H
DW CONV → Output(W, H, C) = Input (W, H, C) ⊗ Ker (R, S, C) ;

Fig. 3 Regular and DW CONV architecture illustration. There are some numerical values for clarity: the atomic-c = 16, the atomic-k =8, the CONV planar size is identical in input and output, the convolution stride is one. The dimension symbols are illustrated on the bottom-left side of the figure.

DMA, buffer, and multiplier (M) array, but they use different adder trees (ADD), accumulators (ACC), different scalar cores, and respective data sequences. The atomic operation of the DLA defines the number of multiply-accumulate (MAC) operations at the same time. The atomic dimension of the DLA uses the channels-first policy instead of the planar-first policy to process convolution. The multiplier array is constructed by the unit input feature channels (= atomic-c) multiplying the unit kernel filter channels (= atomic-k).

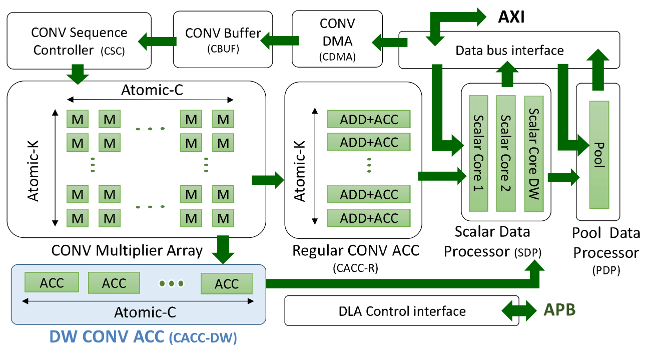

Fig. 2 DLA HW architecture. The regular and DW convolution can fuse scalar operations. The DLA is programmed through the APB interface. The scalar core (single data processor, SDP) only deals with scalar operations of the input vectors.

The benefit of DLA's channel-first strategy is to deal with variable kernel shapes and strides: regarding various kernel shapes and strides, the 1 × 1 × atomic-c operation is in the for-loop center, where the input and kernel planar dimension, the for-loop index is updated without a tight CONV data dependency. The multiplier utilization is high when regular CONV usually has 2^n channels. However, the shortage of this architecture is to deal with small channel CONV, especially the DW CONV that only has one channel. As a result, we propose to separate the accumulation direction, making DW CONV accumulate the product of atomic-k direction and each column's result is separated. The regular CONV still accumulates the products in the atomic-c direction, where each row is summed respectively.

Fig. 3 details the architecture of regular and DW CONV. Here the numbers are examples to ease reading: give the 16 × 8 multiplier array, the atomic-c is16, and the atomic-k is 8. The multiplier array will firstly access the first 8 kernels' first 16 data points that are placed in the 8 rows by 16 columns during the regular CONV. Simultaneously, the first input planar 16 channel points are shared across the atomic-k direction. The equation of the regular CONV is shown on the upper side of Fig. 3: the inner 16 adding operations are done by the adder tree (ADD), and the accumulator (ACC) then processes the full convolution loops, the times are equal to R × S × (OC/16). The sequencer before the CONV multiplier array arranges the planar input and kernel data. The DW CONV is done by the last row of the multiplier array. During the DW CONV, the other 7 multiplier rows and regular 8 ACC+ADD blocks are all clock-gated, and only the 16 DW ACC blocks serve the DW accumulation.

C. Software Solution

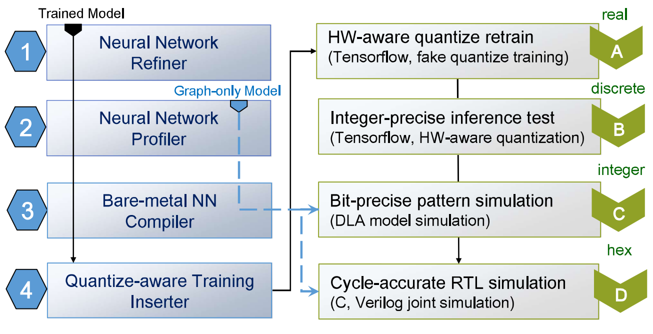
Trained Model
1 Neural Network Refiner
Graph-only Model
2 Neural Network Profiler
3 Bare-metal NN Compiler
4 Quantize-aware Training Inserter
real
HW-aware quantize retrain (Tensorflow, fake quantize training) A
discrete
Integer-precise inference test (Tensorflow, HW-aware quantization) B
integer
Bit-precise pattern simulation (DLA model simulation) C
hex
Cycle-accurate RTL simulation (C, Verilog joint simulation) D

Fig. 4 Toolchain and bitwise accurate verification flow.

Fig. 4 shows the toolchain and flow that facilitates the bitwise-accurate verification. Four functional tools construct the toolchain, comprising the neural network (NN) refiner, profiler, bare-metal compiler, and the QAT inserter. The NN refiner parses the NN graph and gives a modification guide, including merging redundant operations and removing redundant data reshaping. The profiler will fuse the refined

NN layers into a practical DLA inference queue, and then estimate the executing cycle count of the DLA inference queue under various system clock and data memory bandwidth settings.

The bare-metal compiler will first arrange the NN inference queue according to the DLA SPEC, then allocate the DLA resource and storage space, and finally generate the scrips of DLA driving codes in the C language. These scripts will be further compiled on the host machine, wrapping to an application interface (API) that includes the DLA register header file and the linking addresses of the system. The QAT inserter will insert quantization and precision control nodes in the NN graph in the TensorFlow, which is equivalent to the fused NN layers and hardware precision, respectively. The trained models with existing parameters will go through the entire flow. The graph-only models, called pseudo-models here, will go through the short-cut flow that begins from the profiler and ends in the python/Verilog model simulations. Random weights are generated for the pseudo models, and the purpose is to test edge/corner functionalities of the DLA hardware. The graph-only models may be either extracted from the trained model or handcrafted, where the DLA are verified with a testing set consisting of single operators, fused operators, and sequences of fused operators.

There are four verification flows that cover three precision conversion steps: real-to-discrete, discrete-to-integer, and integer-to-hex. The real-to-discrete conversion is simply inside the TensorFlow. The discrete-to-integer conversion is verified by comparing TensorFlow and DLA python simulator results. Finally, the integer-to-hex conversion is confirmed by the DLA python and Verilog simulations. The python DLA model is imported to the TensorFlow so that bitwise inference accuracy is inspected during the HW-aware QAT.

III. FPGA AND CHIP VERIFICATION

This section introduces the DLA implementation in FPGA and chips. The FPGA prototype uses the Xilinx ZCU102 platform, which has an ARM CPU that hosts the DLA and the operating system (Linux). The drive of the DLA uses a bare-metal method, where the entire NN model is wrapped into a standard C-language function. A function call comprises the entire NN layers, and each layer has an individual register setup, memory allocation, and status (interruption) notification. The DLA toolchain compiles the NN model and outputs the drive codes. The host CPU toolchain compiles the main program that includes the DLA drive and CPU host codes. The CPU hosts the live image capture, data format conversion, post-NN process, and display augments, for example, class labeling or bonding box drawing.

Two FPGA prototypes of DLA256 and DLA2048 are compared in Table I. DLA2048 was specified to implement multipliers using DSP slices because of the limited FPGA gate count. The operating frequency is slow in FPGA, which is 200MHz of the DLA256 and 150MHz of the DLA2048. This slow frequency limits the data bandwidth on the AXI bus, so the DLA2048 multiplier utilization cannot be optimized. The peak memory bandwidth is equivalent to 1.6GB/s in the DLA256 and 4.8GB/s in the DLA2048. The abovementioned memory bandwidth is insufficient to support a high hardware utilization in DLA2048. Therefore, the physical performance (fps) ratio of DLA2048/DLA256 is less than 4.4, not proportional to the computation power ratio 8 due to the memory bandwidth limitation.

TABLE I DLA IMPLEMENTATION INFORMATION

DLA SPEC	DLA 256	DLA 2048
DW Option	Turn ON	Turn OFF
MUL#	256 (8×32)	2048 (32×64)
CBUF Size	128KB	512KB
SDP number	1×	8×
PDP number	1×	4×
AXI width	64 bits	256 bits
Platform	Xilinx ZCU102 43% Logic, 3%DSP	Xilinx ZCU102 78% Logic, 86% DSP
CLK Rate	200 MHz	150 MHz
Reference Performance	Tiny YOLO v1 = 14.4 fps Tiny YOLO v2 = 7.5 fps Tiny YOLO v3 = 12.3 fps Full YOLO v3 = 1.07 fps Full YOLO v4 = 1.04 fps Resnet50 = 6.4 fps Mobilenet v1 =33.4 fps	Tiny YOLO v1 = 28.7 fps Tiny YOLO v2 = 28.7 fps Tiny YOLO v3 = 32.3 fps Full YOLO v3 = 4.5 fps Full YOLO v4 = 4.41 fps Resnet50 =17.6 fps Mobilenet = DW turn-off

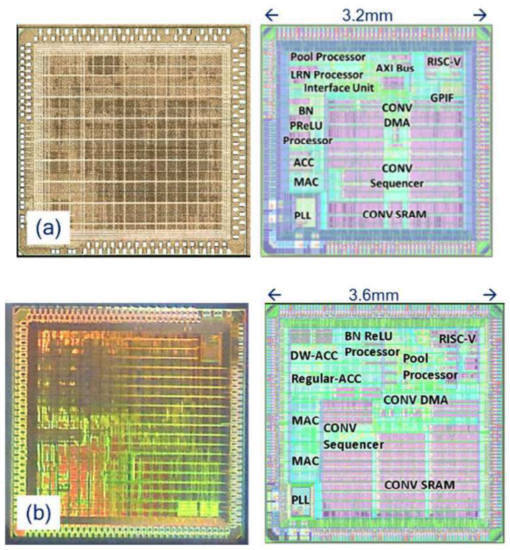

Fig. 5 (a) DLA64, with 128KB SRAM. The silicon area is 3200μm square. Peak performance is 50 GOPs with 60 mW consumption. (b) DLA256, with 128KB SRAM and DW function. The silicon area is 3600 μm square. Peak performance is 200 GOPs with 80mW consumption. The manufacturing technology is TSMC 65nm. The power consumption is limited to the DLA only, not including the PLL, RISC-V, and DRAM interface.

Fig. 5 shows two DLA chips implemented by TSMC-65GP technology. Fig. 5 (a) is the DLA64 that is close to the original open-source nv-small design, and Fig. 5 (b) is the DLA256 with the proposed DW CONV engine. Both DLA64 and DLA256 have a 128KB CONV buffer (CONV SRAM). The floorplan of the chips shows the CONV buffer occupies the most area, about 50% of the chip area. The DW CONV engine is placed on the accumulator (ACC) region, and therefore the DLA256's ACC area is larger than DLA64's. The chip floorplan is designed according to the convolutional data flow, which will go through a clockwise route as described below. First, the CONV DMA requests the outside data through the AXI interface at the top side of the chips. Second, the CONV DMA stores the data into the CONV SRAM on the bottom-left side. Third, the CONV sequencer reads the data out from CONV SRAM and feeds them to the multipliers and ACC units on the right-hand side. Finally, the ACC outputs the results to the batch normalization (BN), ReLU, and pool processors, whose position is close to the AXI interface.

Fig. 6 introduces the demo system and the functional blocks of the DLA256 chip. There is a GPIO port with a peak

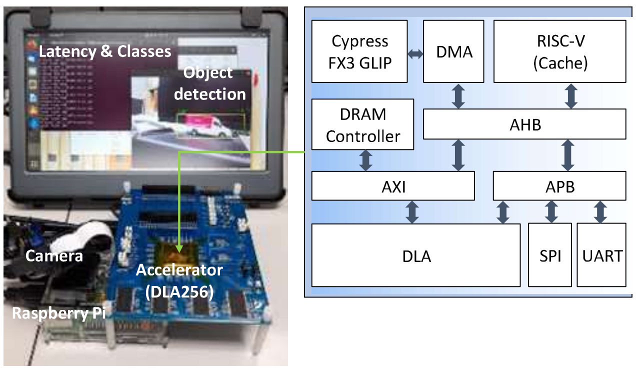

Fig. 6 Object detection test system using DLA256 USB evaluation board. The host is a Raspberry Pi 4 running Ubuntu Linux. The camera interface is MIPI.

data rate of 3200 bps to communicate with the host machine. The GPIO signals are bridged by a Cypress FX3-EZUSB card and then converted to USB 3.0 sequences. The host machine is Raspberry Pi here. Inside the chip, a corresponding GLIP (generic logic interfacing project) data controller bridges the data interface to the board DRAM. An initialization process loads the control codes from the Raspberry Pi to the DLA accelerator. The control codes include the MCU routines, DLA scripts, and DNN weights. The DRAM chips on the board are single-data-rate DRAMs so that the data rate is relatively slow to the DLA requirement. Therefore, we design a cycling convolution test case to let the DLA utilization be above 95% and measure the peak power consumption.

The measurement results of the DLA64 and DLA256 chips are summarized here. The peak computational power of the chips is 200 GOPs of DLA256 and 50 GOPs of DLA64, both operating at 1V and 400 MHz. When running the cycling convolution test case, DLA64 consumes 60mW and DLA256 consumes 80mW, so that the peak power efficiency is 0.83 tera-operations per second per Watt (TOPs/W) of DLA64 and 2.5 TOPs/W of DLA256. When running a Tiny YOLO v1object detection, the average power consumption is 50mW of DLA64 and 65mW of DLA256, and the utilization drops to about 75% and 50%, respectively, because the fully-connected layer has low multiplier utilization but high data transport time.

IV. Comparison and Conclusion

This section concludes this work with comparisons and discussions. Table II compares several 65-nm silicon-proven CNN accelerators. A fair comparison is difficult because each CNN accelerator may have specific operating assumptions, such as the voltage scaling, data sparsity, or specific kernel shapes [5]. The processing element (PE) array shape implies the convolution structure of each accelerator is different. The inference data precision also varies. We leave the voltage scaling and sparsity issues open, and therefore Table II only compares the results at the standard operating voltage. The DLA chips provide a higher frequency and computation performance in Giga-operations per second (GOPs) than the reference chips. A larger multiplier number in our DLA structure will have higher energy efficiency in TOPs/W. Besides the measurement results in Table II, DLA2048 achieved about 6 TOPs/W under the same technology and frequency in the gate-level simulation. However, the DLA2048 requires high bandwidth memories to exert the computation power, and this is also our current work about stacking the high-bandwidth memory with the DLAs.

TABLE II Comparisons among CNN accelerator chips

	Eyeriss Series[6][7]	DNPU[8]	DCNNIP[9]	PD-CNNP[10]	**This Work**
Inner PE Array	12 x 14	16 x 3 x 4	4 x 2 x 8 x 8	13 x 3 x 3	8 x 8 8 x 32
SRAM	192 KB	272 KB	848 KB	72 KB	128 KB
Precision	16-bits	4/16-bit	10-bit	8-bit	8-bits
Tech.	65 nm	65 nm	65 nm	65 nm	65 nm
Case 1: PE#, RES#	168-PE, 16-bit	768-PE, 16-bit,	512-PE, 10-bit	117-PE, 8-bit	64-PE, 8-bit
Voltage, Perf, Energy Efficiency	1.17V, 84 GOPs, 450mW	1.1V, 200 MHz, 1.0TOPs/W	0.4V, 60MHz, 1.15TOPs/W	1.0V, 90GOP/s, 1.25TOPS/W	1.0V, 400MHz, 60mW, 0.8TOPs/W
Case 2: PE#, RES#	384-PE, 8-bit	3072-PE, 4-bit			256-PE, 8-bit
Voltage, Perf, Energy Efficiency	153.6 GOPs, 200 to 900GOPs/W	1.1V, 100 MHz, 3.9 TOPs/W			1.0V, 400MHz, 80mW, 2.5TOPs/W

To briefly conclude, this paper introduced an end-to-end solution of DLA design. The toolchains provide hardware generation, fast DNN profiling, DNN graph-based verification, and DNN model compile. The series of DLAs are realized in FPGA prototypes and silicon chips. The measured energy efficiency of DLA256 is 2.5TOPs/W at 1V and 400 MHz, using TSMC 65-nm technology.

References

[1] NVDLA open source project [Online] Available: http://nvdla.org/

[2] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. “Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR,” *arXiv: 1704.04861[cs.CV]*, Apr 2017.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779-788.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection” *arXiv:2004.10934 [cs.CV],* Apr 2020.

[5] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1-9.

[6] Y. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127-138, Jan. 2017.

[7] Y. Chen, T. 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, June 2019.

[8] D. Shin, J. Lee, J. Lee and H. Yoo, “14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2017, pp. 240-241.

[9] J. Sim, S. Lee and L. Kim, “An Energy-Efficient Deep Convolutional Neural Network Inference Processor With Enhanced Output Stationary Dataflow in 65-nm CMOS,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 87-100, Jan. 2020.

[10] Yue et al., “A 3.77TOPS/W Convolutional Neural Network Processor With Priority-Driven Kernel Optimization,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 2, pp. 277-281, Feb. 2019.