# A TensorFlow and System Simulator Integration Approach to Estimate Hardware Metrics of Convolution Accelerators

Leonardo R. Juracy*, Matheus T. Moreira†, Alexandre M. Amory‡* and Fernando Gehm Moraes*
*PUCRS – School of Technology - Av. Ipiranga 6681, Porto Alegre, Brazil
†Chronos Tech, San Diego, USA
‡Scuola Superiore Sant'Anna, Pisa, Italy
leonardo.juracy@acad.pucrs.br, matheus@chronostech.com, alexandre.amory@santannapisa.it,
fernando.moraes@pucrs.br

2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS) | 978-1-7281-7670-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/LASCAS51355.2021.9459183

***Abstract*—GPUs became the reference platform for both training and inference phases of Convolutional Neural Networks (CNN), due to their tailored architecture to the CNN operators. However, GPUs are power-hungry architectures. A path to enable the deployment of CNNs in energy-constrained devices is adopting hardware accelerators for the inference phase. The design space exploration of CNNs using standard approaches, such as RTL, is limited due to their complexity. Thus, designers need frameworks enabling design space exploration that delivers accurate hardware estimation metrics do deploy CNNs. This work aims to propose a framework to explore hardware accelerators' design space, providing power, performance, and area (PPA) estimations. The heart of the framework is a system simulator with TensorFlow as front-end and as back-end performance estimations obtained from the physical synthesis. Results evaluate the energy trade-off varying the number of convolutional layers, showing that it can be necessary to spend approximately 40% more energy to increase 0.02% in the accuracy.**

***Index Terms*—CNN, Convolution Hardware Accelerator, PPA.**

## I. Introduction

One of the most common ways to deliver Machine Learning (ML) is by using Artificial Neural Networks (ANN). ANNs contain thousands of interconnected neurons. Synapses have data input samples, plus a weight that works similarly to a filter and activation function, which creates an output used in synapses of next neurons [1, 2]. Convolutional Neural Networks (CNNs) are a type of ANNs.

The deployment of CNNs comprises two phases: training and inference [1]. The training phase defines the synapses weights' value. The inference phase uses the weights previously computed to classify or predict output values using unknown inputs. The success of CNNs led to the development of frameworks that help developers to build their models. These frameworks offer mechanisms required for training and inference. Examples of frameworks include Caffe [3], Pytorch [4] and TensorFlow [5].

Classically, CPUs have been a common approach to process CNNs. Even with optimized instruction set architectures, CPUs are inefficient in terms of performance and energy (e.g., AlexNet from 2012 [6] requires billions of operations to process a single input). Thus, GPUs became the reference platform for both training and inference phases, due to their tailored architecture to the CNN operators. The main GPU drawback is its considerable energy consumption. Considering energy-constrained applications, such as IoT, autonomous driving, wearable devices, the adoption of hardware accelerators became a trend for the inference phase [7, 8].

978-1-7281-7670-3/21/$31.00 ©2021 IEEE

Accelerators are hard to implement and verify using classic design flows. The design of these blocks using register transfer level (RTL) abstraction limits the design space exploration. The RTL flow is the last design step, after defining the CNN architecture. Despite the efforts to increase the abstraction level for accelerators using high-level synthesis (HLS) [9, 10], this approach also has challenges related to performance and power estimation, once the goal of HLS is to generate an RTL description as output.

System simulators [11, 12] are a path for accelerators' design space exploration. These simulators are typically described in high-level abstraction languages, as Python and C++, reducing the design time and providing power, performance, and area (*PPA*) evaluation. The main drawback of system simulators is the PPA accuracy, estimated from the number of executed operations, as multiplier–accumulator (MAC) [11, 13].

The *goal* of the present work is to propose a flow to estimate hardware metrics, using the advantages of the TensorFlow regarding CNN's modeling, and the advantages of system simulators regarding design time and PPA evaluation. This work brings three main *contributions*: (*i*) integrate TensorFlow to a system simulator to evaluate accelerators in the inference phase; (*ii*) improve the accuracy of hardware metric measurements by providing coarse-grain PPA metrics, i.e., not MACs but CNN operators as convolution, obtained from the physical layout; (*iii*) integration of the two first contributions into a system simulator resulting in a framework enabling design space exploration from PPA estimations.

The remainder of this paper is organized as follows. Section II presents the related works present in the state-of-the-art. Section III describes the proposed framework. Section IV presents the PPA results. Finally, Section V provides some conclusions.

## II. Related Works

This Section describes works that generate PPA analyses focused on ML applications. MAESTRO [14] is a framework to describe and analyze Neural Network hardware, which allows obtaining the hardware cost to implement a target architecture. It has a domain-specific language (DSL) to describe the dataflow that allows specifying the number of PEs, memory size, and NoC bandwidth parameters. The results generated by the framework are focused on performance analyses. In recent work, MAESTRO was used to estimates cost-benefit tradeoffs between execution time and energy efficiency for CNN models, such as VGG and AlexNet, and hardware configuration.

SCALE-Sim (Systolic CNN Accelerator Simulator) [15] is a systolic array cycle-accurate simulator. According to the authors, SCALE-Sim is the first simulator tuned to running DNNs. This simulator allows configuring micro-architectural features such as array size, array aspect ratio, scratchpad memory size, and dataflow mapping strategy. Also, it is possible to configure system integration parameters, such as memory bandwidth. SCALE-SIM simulates convolutions and matrix multiplications, and models the compute unit as a systolic array. Also, it allows simulation in a system context, with CPU and DMA components. The authors show detailed experiments to understand the design space and tradeoff in designing a systolic array-based CNN accelerator. A recent SCALE-Sim extension provides an analytic model to find the best accelerator configuration based on parameters like DRAM bandwidth.

Timeloop [11] is a design space exploration framework for CNNs. It can emulate a set of accelerators, such as NVDLA [16]. Its focus on the convolution layer analyses. Timeloop uses as input a workload description, such as input dimension and weight values, a hardware architecture description, such as arithmetic modules, and the hardware architecture constraints, such as how the computation can be partitioned. Instead of using a cycle-accurate simulator, Timeloop uses data transfers' deterministic behavior to perform analytic analyses. As energy models, Timeloop has memory, arithmetic units, and wire/network models based in TSMC 16nm FinFET.

STONNE [12] is a cycle-accurate architecture simulator for CNNs which allows end-to-end evaluation. It is connected with Caffe framework [3] to generate the CNNs, and models the MAERI accelerator [17]. The results are focused on performance and hardware utilization and show an average difference of 15% in total executed cycles than the original MAERI results. To estimate area and energy, STONNE uses the Accelergy energy estimation methodology [13], which considers basic modules to calculate the energy values, such as adders.

Previous works present gaps in evaluating CNN's accelerators. MAESTRO does not allow the accelerator simulation, which limits the performance evaluation (e.g., throughput). SCALE-Sim does not provide power or energy results. Timeloop provides PPA based on basic operations, such as adders and multipliers. Methods relying on operations' counting do not consider how these operators are interconnected (e.g., 1D or 2D systolic arrays or adder trees), resulting in imprecise hardware metrics. STONNE is the most similar framework to our proposal. Its main limitation is power analysis, based on a single accelerator (Accelergy), limiting the design space exploration. The proposed framework fills these gaps by integrating TensorFlow to a system simulator, with hardware metrics related to CNN operators.

## III. Proposed flow

Figure 1 presents the proposed flow. TensorFlow models the CNN, being responsible for training and inference phases, followed by the weights extraction. The weights are stored in a dictionary format, used at the inference step, which validates the weights' extraction accuracy. This work adopts an integer quantization to avoid floating-point operations in the accelerator. The last action executed in TensorFlow is exporting a header file to be used by the system simulator.

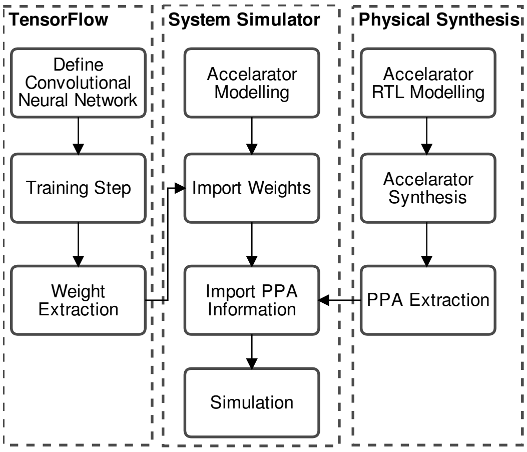

Fig. 1. Convolution Accelerator Hardware Metric Extraction Flow.

The physical synthesis corresponds to the synthesis of NVDLA modules [16]. This step generates CNN operators' layout, e.g., a 3x3 convolution, and a netlist with extracted parasitic capacitances. The simulation of this netlist enables extracting the switching activity to characterize the accelerator dynamic power. The result of the physical synthesis is PPA reports.

The ORCA cycle-accurate system simulator [18] models the hardware accelerator, integrated with the CNN model generated by TensorFlow and the PPA reports generated by the physical synthesis. The simulator captures information related to the CNN execution, presenting a summary with accelerator performance, area, and energy results considering minimum, average, and maximum dynamic power dissipation at the end of the simulation. Next sections detail the framework.

### A. TensorFlow

TensorFlow [5] is a Google framework providing libraries to implement ML applications. TensorFlow allows implementing CNNs, including the training and inference phases. It is possible to use CNN's functions such as 2D convolution, max pooling, and ReLU. This work uses the TensorFlow for:

1) Modeling the CNN, exploring its architecture;
2) Extracting the weight values of the selected network;
3) Evaluating the weights' quantization from 16-bits floating-point to 16-bits integer.

Figure 2 shows an example of a TensorFlow code. The environment allows exploring CNN architectures and their accuracy regarding the network depth, stride dimension, activation functions, and the number of filters. Thus, it is possible to tune the CNN architecture based on an accuracy target. This example shows a CNN with four convolution layers with 16, 8, 3, and 1 filters, a fully connected layer, and strides with dimensions 2x2 and 1x1.

```
# Clearup everything before running
keras.backend.clear_session()

# Create model
model = keras.models.Sequential()

# Add layers
model.add(keras.layers.Conv2D(16, (3,3), strides=(2, 2), activation='relu',
↪  input_shape=(28, 28, 1)))
model.add(keras.layers.Conv2D(8,  (3,3), strides=(1, 1), activation='relu'))
model.add(keras.layers.Conv2D(3,  (3,3), strides=(2, 2), activation='relu'))
model.add(keras.layers.Conv2D(1,  (3,3), strides=(1, 1), activation='relu'))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))

# Build model and print summary
model.build(input_shape=featureShape)
model.summary()
```

Fig. 2. TensorFlow Environment Code Example.

TensorFlow also allows extracting the weights' values after reaching the target accuracy – training phase. A post-extraction quantization happens after the training phase by converting the floating-point weights to 16-bit integers by multiplying the weights' values by a power of two. Adopting integer values avoids floating-point arithmetic in the accelerator, reducing its area and power consumption.

It is necessary to simulate the CNN after quantization to evaluate if there is accuracy loss. If the quantization presents a small accuracy penalty, TensorFlow exports the weights in a header format to the system simulator.

### B. PPA Extraction

NVIDIA Deep Learning Accelerator (NVDLA) [16] is an open-source framework from NVIDIA to implement machine learning applications, providing RTL codes. This work uses NVDLA for building the accelerator RTL description by configuring multiplier, adders, and ReLU activation function units. Figure 3 illustrates an instance of the accelerator array (3x3 convolution). The accelerator array parameters is a function of the CNN configuration generated in TensorFlow, considering the filter dimension.

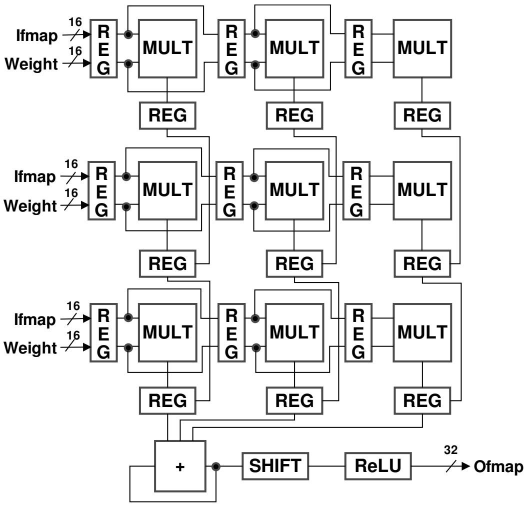

Fig. 3. Proposed hardware accelerator architecture based on the NVDLA modules [16].

The accelerator, Figure 3, is an array of 3x3 multipliers (**MULT**), an accumulator, and a ReLU module (**ReLU**). The **SHIFT** is a module responsible for quantization, in such a way to normalize the convolution result to 16 bits. The accelerator inputs are shifted horizontally through the array, while the multipliers' output is shifted vertically until the accumulator. The accumulator output passes through the **SHIFT** and sets the **ReLU** input, which produces a zero value if the input value is negative, else by-pass the value to the output. The system simulator models the accelerator in a high-level abstraction description, using the same architecture.

After generating the RTL array description, Cadence Genus and Innovus tools are employed to execute synthesis and place and route (P&R). The accelerator area includes gates and wires, and not only cell counting. The simulation of the post-P&R netlist furnishes the accelerator performance (operating frequency) and the switching activity (value change dump (VCD) file). The VCD file is the source for power estimation. The PPA (power, performance, and area) metrics are exported to the system simulator.

### C. System Simulator - ORCA

ORCA [18] is a system simulator that allows modeling hardware using a high-level language and execute simulations with cycle accuracy. This work uses the ORCA for:

1) Model and simulate the accelerator;
2) Validate the CNN accuracy in hardware;
3) Generate PPA evaluation of the CNN.

ORCA uses C++ to describe circuits architectures, making modeling easier than the RTL approach that uses VHDL/Verilog. Also, it allows simulating the circuit behavior faster than other languages such as SystemC/SystemVerilog.

The CNN is modeled into the ORCA simulator, using the header generated in Section III-A and the accelerator array. Thus, it is possible to validate the accuracy obtained after the quantization in the hardware model. The simulator reports the CNN power and performance estimation at the end of the simulation according to the number of executed convolutions.

This first version of the framework only supports CNN architectures composed by convolution layers. To overcome this limitation, it is necessary to include accelerators responsible for executing other CNN operations.

## IV. Results

Results use CNNs generated by TensorFlow. Three networks were generated using convolution operations, changing the network depth by 2, 3, and 4 layers, with 4, 12, and 38 filters respectively. All three CNNs were trained based on the MNIST dataset using 3x3 filters with strides between 1x1 and 2x2, ReLU as activation function, and a fully-connected layer with softmax activation function. The training step was performed in TensorFlow for 5 epochs. The fully-connected layer is not accelerated in hardware and is executed in software in the system simulator.

Six VCD files were created from a post P&R simulation. Each VCD represents a simulation scenario that generates one pixel of a convolution output feature map. Scenarios include a real convolution operation, two scenarios with random values, and three scenarios with constant input values (x"AAAA",

TABLE I
PPA RESULTS FOR NVDLA-BASED ACCELERATOR RUNNING A MNIST APPLICATION.

| Tech. | Freq. | Area | # Conv. | # Conv. | Accu. (%) | | Exec. | Enegy (mJ) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MHz | $\mu m^2$ | Layers | Oper. | TensorFlow | ORCA | Time (ms) | Min. | Avg. | Max. |
| 28nm | 1.6 GHz | 35003 | 2 | 62800 | 0.95 | 0.90 | 0.5 | 11 | 14 | 17 |
| | | | 3 | 174000 | 0.95 | 0.92 | 1.4 | 32 | 38 | 47 |
| | | | 4 | 375600 | 0.96 | 0.93 | 3.1 | 69 | 84 | 102 |
| 65nm | 1.0 GHz | 97890 | 2 | 62800 | 0.95 | 0.90 | 0.8 | 24 | 31 | 39 |
| | | | 3 | 174000 | 0.95 | 0.92 | 2.4 | 68 | 87 | 109 |
| | | | 4 | 375600 | 0.96 | 0.93 | 5.2 | 147 | 189 | 236 |

x“5555”, x“FFFF”). The simulation of these scenarios generates minimal, average, and maximum power dissipation values for a convolution operation. The power values are used to compute energy consumption.

Table I presents results for 28nm and 65nm technology nodes. The accuracy was extracted using 100 inputs from MNIST dataset. Results show that the quantization causes a small penalty in the hardware accelerator’s accuracy compared to the TensorFlow results. Also, the Table shows that accuracy increases together with the CNN depth. On the other side, the execution time and energy increase. Although these results are expected, the goal is to show the framework’s potential to explore different CNN configurations for different technologies, obtaining PPA metrics. Thus, these results demonstrate the potential of the environment for the CNN design space exploration.

Figure 4 presents energy values in the x-axis and the CNN accuracy in the y-axis. It is possible to note that it is necessary to spend approximately 40% more energy in the 28nm technology node to increase 0.02% in the accuracy. This increase is more pronounced in the 65nm technology node, reaching approximately 60%.

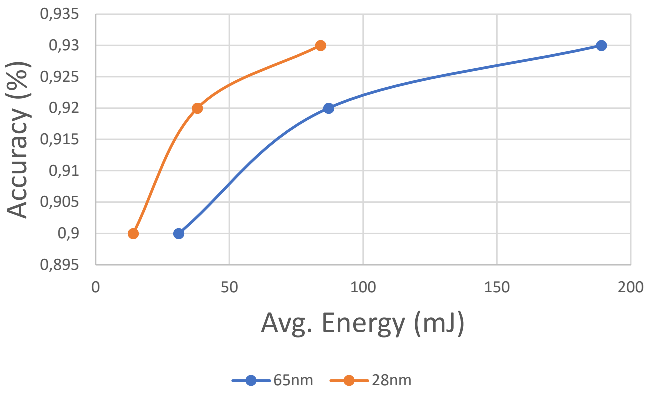

Fig. 4. Accuracy and Average Energy Trade-off.

## V. Conclusion

This work proposed a framework to analyze hardware metrics regarding convolution accelerators. The framework allows to: $(i)$ build CNNs with TensorFlow; $(ii)$ extract their weights; $(iii)$ execute the network using a high-level accelerator model in a system simulator; $(iv)$ estimate PPA results and to perform design space exploration. Thus, it is possible to analyze the trade-offs to increase the accuracy regarding power, performance, and area to a given hardware configuration.

We propose to analyze power regarding the entire accelerator, not only isolated models, once it results in more accurate PPA values. Future works include extending the framework to support other CNN operations, like max-pooling, to explore different CNNs’ configurations.

## References

[1] S. S. Haykin *et al.*, “Neural networks and learning machines/Simon Haykin,” 2009.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[3] Caffe, “Caffe,” 2020. [Online]. Available: https://caffe.berkeleyvision.org/

[4] PyTorch, “PyTorch,” 2020. [Online]. Available: https://pytorch.org/

[5] TensorFlow, “TensorFlow,” 2020. [Online]. Available: https://www.tensorflow.org/

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] Y.-H. 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, 2016.

[8] Apple, “iPhone 11,” 2019. [Online]. Available: https://www.apple.com/iphone-11/

[9] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, “ESP4ML: Platform-based design of systems-on-chip for embedded machine learning,” *arXiv preprint arXiv:2004.03640*, 2020.

[10] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. R. Pinckney, P. Raina *et al.*, “MAGNet: A Modular Accelerator Generator for Neural Networks,” in *ICCAD*, 2019, pp. 1–8.

[11] 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 *ISPASS*, 2019, pp. 304–315.

[12] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, “STONNE: A Detailed Architectural Simulator for Flexible Neural Network Accelerators,” *arXiv preprint arXiv:2006.07137*, 2020.

[13] Y. N. Wu, J. S. Emer, and V. Sze, “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” in *ICCAD*, 2019, pp. 1–8.

[14] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[15] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, “A systematic methodology for characterizing scalability of dnn accelerators using scale-sim,” in *2020 IEEE International Symposium on Performance Analysis of Systems and Software*, 2020.

[16] NVIDIA, “NVDLA,” 2019. [Online]. Available: http://nvdla.org/

[17] H. Kwon, A. Samajdar, and T. Krishna, “Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects,” *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[18] A. R. P. Domingues, “ORCA: A Self-Adaptive, Multiprocessor System-On-Chip Platform,” Ph.D. dissertation, Pontifical Catholic University of Rio Grande do Sul, 2020.