Simulation and Implementation of Novel Deep Learning Hardware Architectures for Resource Constrained Devices

Thesis submitted by

Corey Lammie, B.Eng (Hons) & B.IT

on December, 2022, for the degree of

Doctor of Philosophy

at the

College of Science and Engineering
James Cook University

Supervisors

A/Prof. Mostafa Rahimi Azghadi

Associate Professor at the College of Science and Engineering at James Cook University, Australia.

Prof. Wei Xiang

Cisco Chair of AI and Internet of Things at La Trobe University, Australia.
Adjunct Professor at the College of Science and Engineering at James Cook University, Australia.

Declaration and Statement of Access to This Thesis

Declaration

I, Corey Lammie, certify that:

The work presented in this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or institute of tertiary education.
This thesis contains a number of published research articles, all of which have been co-authored. The contribution of others is formally acknowledged in the *Contribution of Others and List of Publications Included in this Thesis* Section.

Statement of Access to This Thesis

I, the undersigned, the author of this work, understand that James Cook University will make this work available for use within the University, and via the Australian Digital Thesis Network, for use elsewhere.

I understand that as an unpublished work, a thesis has significant protection under the Copyright Act. I do not wish to place any restriction on access to this thesis. However, any use of its content must be acknowledged and could be potentially be restricted by future patents.

Signed

Corey Lammie,
December, 2022

Acknowledgments

First and foremost, I would like to thank my primary supervisor, Prof. Mostafa Rahimi Azghadi. Little did I know, after stepping put in his office for the first time, how much of an impact he would have on my life. His unwavering support and guidance has shaped me into the scientist I am today, and has made the completion of this PhD thesis possible. Mostafa is a great mentor and advocate of mine, and I'm happy to call him a friend.

I would also like to acknowledge the support of my secondary supervisor, Prof. Wei Xiang. He contributed his valuable expertise and time to help guide many of the works presented in this thesis. Having perspective from someone outside of my immediate research area was invaluable, and greatly helped me visualise the *big picture* on many occasions.

I am incredibly thankful for the many coauthors of the papers, both included in this thesis document, listed in the *List of Publications Not Included in This Thesis* Section, and under preparation for submission to be considered for publication. I have had the opportunity to collaborate with a large number of academics with diverse backgrounds. This list is continually growing, and currently includes the following great scientists: J. K Eshraghian, B. Linares-Barranco, T. J. Hamilton, G. Indiveri, M. Payvand, E. Donati, A. Olsen, T. Carrick, A. Saleh, I. H. Laradji, D. Vazquez, C. Flavell, W. D. Lu, A. Schaik, D. Ielmini, O. Krestinskaya, A. James, C. Li, R. Genov, A. Amirsoleimani, S. Yang, X. Dong, T. Zhang, I. Shipley, J. Stouffer, Y. C. Chen, A. Sebastian, M. Le. Gallo, J. Büchel, I. Boybat, H. Benmeziane, and T. Vasilopoulos. Many of these individuals, from those that have been there from the start at James Cook University, to more recent colleagues at IBM Research - Zurich, have been a great source of inspiration and encouragement through good and bad times.

Last, but certainly not least, I would like to thank my family and my life-long partner, Sheridan. Their love and has been paramount. Even if they still don't quite understand what I am doing, they have always been supportive of my endeavors - even if it does take me to many far away places around the world. Self doubt and paranoia is prevalent in academia; after all, completing a PhD is an incredibly daunting task. However, with an open mind and a strong support network, this doesn't have to be the case. If you believe in yourself, anything truly is possible.

Language Choice

American English is used throughout this thesis document. While this may be unexpected, especially considering that this thesis was conducted and written at an Australian University, American English is arguably the international language of academia. All original research articles that are included in this thesis have been published in international peer reviewed journals; all of which use American English. This decision was intentionally made with consideration to maximize the accessibility of this document for an international audience. Additionally, in all Chapters aside from the *Introduction* and *Conclusion and Future Work* Chapters, the terms *we* and *paper* are used to refer to all authors, collectively, and the corresponding chapter, respectively.

Contribution of Others and List of Publications Included in this Thesis

This thesis contains a number of original research articles which have been published during my PhD candidature. These papers have been slightly modified to improve readability and cohesion in the form of a thesis document. In this Section, a list of publications included in this thesis is presented, and the contributions of all of the coauthors who contributed to the papers included in this thesis are formally acknowledged. A complete list of all my publications during this my PhD is provided in the next Section.

1. Hardware Implementation of Deep Network Accelerators Towards Healthcare and Biomedical Applications

[1] M. R. Azghadi, **C. Lammie**, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware Implementation of Deep Network Accelerators Towards Healthcare and Biomedical Applications," in IEEE Transactions on Biomedical Circuits and Systems, vol. 14, no. 6, pp. 1138-1159, Dec. 2020, doi: 10.1109/TBCAS.2020.3036081.
Location in thesis: Chapter 2
Student contribution: 40%. Developed and ran simulations, generated figures, performed part of the literature review, and wrote large parts of the paper.

2. Modeling and Simulating In-Memory Memristive Deep Learning Systems: An Overview of Current Efforts

[2] **C. Lammie**, W. Xiang, and M. R. Azghadi, "Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts," in Array, vol. 13, Mar. 2022, doi: 10.1016/J.ARRAY.2021.100116.
Location in thesis: Chapter 3
Student contribution: 90%. Developed and ran simulations, generated figures, and wrote the paper.

3. Seizure Detection and Prediction by Parallel Memristive Convolutional Neural Networks

[3] C. Li†, **C. Lammie**†, X. Dong, A. Amirsoleimani, M. R. Azghadi and R. Genov, "Seizure Detection and Prediction by Parallel Memristive Convolutional Neural Networks," in IEEE Transactions on Biomedical Circuits and Systems, 2022, doi: 10.1109/TBCAS.2022.3185584.

†**These authors contributed equally.**

Location in thesis: Chapter 4

Student contribution: 45%. Developed and ran simulations, generated figures, and wrote most of the paper.

4. Memristive Stochastic Computing for Deep Learning Parameter Optimization

[4] **C. Lammie**, J. K. Eshraghian, W. D. Lu and M. R. Azghadi, "Memristive Stochastic Computing for Deep Learning Parameter Optimization," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 5, pp. 1650-1654, May 2021, doi: 10.1109/TCSII.2021.3065932.

Location in thesis: Chapter 5

Student contribution: 90%. Developed and ran simulations, generated figures, and wrote the paper.

5. MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems

[5] **C. Lammie**, W. Xiang, B. Barranco, and M. R. Azghadi, "MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems," in Neurocomputing, vol. 485, pp. 124-133, May. 2022, doi: J.NEUCOM.2022.02.043.

Location in thesis: Chapter 6

Student contribution: 90%. Developed the simulation framework and ran simulations, generated figures, and wrote the paper.

6. Empirical Metal-Oxide RRAM Device Endurance and Retention Model for Deep Learning Simulations

[6] **C. Lammie**, M. R. Azghadi, and D. Ielmini, "Empirical Metal-Oxide RRAM Device Endurance and Retention Model for Deep Learning Simulations," in Semiconductor Science and Technology, vol. 36, no. 6, Apr. 2021, doi: 10.1088/1361-6641/ABF29D.

Location in thesis: Chapter 7
Student contribution: 90%. Developed and ran simulations, generated figures, and wrote the paper.

7. Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge

[7] **C. Lammie**, A. Olsen, T. Carrick and M. Rahimi Azghadi, "Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge," in IEEE Access, vol. 7, pp. 51171-51184, 2019, doi: 10.1109/ACCESS.2019.2911709.
Location in thesis: Chapter 8
Student contribution: 90%. Developed and ran simulations and hardware experiments, generated figures, and wrote the paper.

List of Publications Not Included in This Thesis

8. Efficient FPGA Implementations of Pair and Triplet-Based STDP for Neuromorphic Architectures

[8] **C. Lammie**, T. J. Hamilton, A. van Schaik and M. Rahimi Azghadi, "Efficient FPGA Implementations of Pair and Triplet-Based STDP for Neuromorphic Architectures," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 4, pp. 1558-1570, April 2019, doi: 10.1109/TCSI.2018.2881753.

9. Stochastic Computing for Low-Power and High-Speed Deep Learning on FPGA

[9] **C. Lammie** and M. R. Azghadi, "Stochastic Computing for Low-Power and High-Speed Deep Learning on FPGA," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, doi: 10.1109/ISCAS.2019.8702248.

10. Accelerating Deterministic and Stochastic Binarized Neural Networks on FPGAs Using OpenCL

[10] **C. Lammie**, W. Xiang and M. R. Azghadi, "Accelerating Deterministic and Stochastic Binarized Neural Networks on FPGAs Using OpenCL," 2019 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2019, pp. 626-629, doi: 10.1109/MWSCAS.2019.8884910.

11. Variation-aware Binarized Memristive Networks

[11] **C. Lammie**, O. Krestinskaya, A. James and M. R. Azghadi, "Variation-aware Binarized Memristive Networks," 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019, pp. 490-493, doi: 10.1109/ICECS46596.2019.8964998.

12. Training Progressively Binarizing Deep Networks using FPGAs

[12] **C. Lammie**, W. Xiang and M. R. Azghadi, "Training Progressively Binarizing Deep Networks using FPGAs," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, doi: 10.1109/ISCAS45731.2020.9181099.

13. Live Demonstration: Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge

[13] **C. Lammie** and M. R. Azghadi, "Live Demonstration: Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, doi: 10.1109/ISCAS45731.2020.9180682.

14. Biologically Plausible Contrast Detection using a Memristor Array

[14] J. K. Eshraghian, **C. Lammie** and M. R. Azghadi, "Biologically Plausible Contrast Detection using a Memristor Array," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, doi: 10.1109/ISCAS45731.2020.9180914.

15. MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures

[15] **C. Lammie** and M. R. Azghadi, "MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, doi: 10.1109/ISCAS45731.2020.9180810.

16. Towards Memristive Deep Learning Systems for Real-Time Mobile Epileptic Seizure Prediction

[16] **C. Lammie**, W. Xiang and M. R. Azghadi, "Towards Memristive Deep Learning Systems for Real-Time Mobile Epileptic Seizure Prediction," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, doi: 10.1109/ISCAS51556.2021.9401080.

17. A Deep Learning Localization Method for Measuring Abdominal Muscle Dimensions in Ultrasound Images

[17] A. Saleh, I. H. Laradji, **C. Lammie**, D. Vazquez, C. A. Flavell and M. R. Azghadi, "A Deep Learning Localization Method for Measuring Abdominal Muscle Dimensions in Ultrasound Images," in IEEE Journal of Biomedical and Health Informatics, vol. 25, no. 10, pp. 3865-3873, Oct. 2021, doi: 10.1109/JBHI.2021.3085019.

18. Design Space Exploration of Dense and Sparse Mapping Schemes for RRAM Architectures

[18] **C. Lammie**, J. K. Eshraghian, C. Li, A. Amirsoleimani, R. Genov, Wei D. Lu, and M. R. Azghadi, "Design Space Exploration of Dense and Sparse Mapping Schemes for RRAM Architectures," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), 2022, doi: 10.48550/ARXIV.2201.06703.

19. Navigating Local Minima in Quantized Spiking Neural Networks

[19] J. K. Eshraghian, **C. Lammie**, M. R. Azghadi, and Wei D. Lu, "Navigating Local Minima in Quantized Spiking Neural Networks," 2022 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2022, doi: 10.48550/ARXIV.2202.07221.

20. Toward A Formalized Approach for Spike [20] Sorting Algorithms and Hardware Evaluation

[20] T. Zhang, **C. Lammie**, M. R. Azghadi, A. Amirsoleimani, and R. Genov, "Toward A Formalized Approach for Spike Sorting Algorithms and Hardware Evaluation," 2022 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2022, doi: 10.48550/ARXIV.2205.06514.

Awards and Grants

Domestic Prestige Research Training Program Scholarship (DPRTPS)

Recipient of the PhD DPRTPS; the highest paid PhD scholarship in Australia, 2019-2022.

IBM International PhD Fellowship

The only Australian recipient of the intensely competitive IBM PhD Fellowship, 2020-2021.

IEEE Circuit and System (CAS) Society Pre-Doctoral Grant

Recipient of the IEEE CAS Pre-Doctoral Grant, 2020.

Merck Innovation Cup Stipend

Recipient of the Merck Innovation Cup Stipend, 2020.

Student Travel Grant for the IEEE International Symposium on Circuits and Systems (ISCAS)

Recipient of the student travel grant for IEEE ISCAS, 2019, 2022.

Student Travel Grant for the IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)

Recipient of the student travel grant for IEEE MWSCAS, 2019.

Student Travel Grant for the IEEE International Conference on Electronics, Circuits and Systems (ICECS)

Recipient of the student travel grant for IEEE ICECS, 2019.

Regional Finalist of the InnovateFPGA Design Contest

Regional finalist of the Innovate FPGA Design Contest, 2019.

Abstract

On account of recent technological advancements, Deep Learning (DL) systems have been used to solve a large variety of challenging engineering tasks, ranging from image classification and segmentation, to biomedical signal processing for epileptic seizure prediction. In contrast to traditional Machine Learning (ML) algorithms, DL systems do not usually require features to be manually extracted prior to data processing, and can be trained with relative ease; greatly reducing the time to deployment for novel applications and scenarios. However, compared to traditional ML algorithms, DL systems commonly consume significantly more power and introduce additional latency. Consequently, they are difficult to deploy in resource-constrained environments and on resource-constrained devices; especially when they are implemented using Central Processing Units (CPUs) and Graphics Processing Units (GPUs). Of late, significant efforts have been made to study and implement low-power and high-speed DL systems suitable for deployment on resource-constrained devices. In this thesis, both traditional synchronous and brain-inspired asynchronous architectures were investigated, and novel hardware architectures were designed, simulated, and/or implemented for traditional synchronous DL systems. Two specific device technologies were investigated: memristors, and Field Programmable Gate Arrays (FPGAs). The key findings of this thesis, all of which have been disseminated in peer reviewed publications, demonstrate that novel mixed signal memristive-Complementary Metal–Oxide–Semiconductor (CMOS) and FPGA architectures can be used to reduce the power and resource requirements of DL systems, both during inference and training. Moreover, it was demonstrated through the development of an open-source simulation framework for memristive DL systems and an empirical Metal-Oxide Resistive Random-Access Memory (RRAM) device model, that specifically for mixed signal memristive-CMOS architectures, current design flows could be improved upon. Disruptive software hardware co-optimization design methodologies, such as those explored in this thesis, can be used to facilitate the design of next-generation novel hardware architectures for DL acceleration.

Contents

List of Tables

List of Figures

List of Acronyms

0T1R 0-Transistor 1-Resistor

1T1R 1-Transistor-1-Resistor

ADC Analog-to-Digital Converter

AFE Analog Front End

AI Artificial Intellegence

ANN Artifical Neural Network

API Application Programming Interface

APS Active Pixel Sensor

ASIC Application-Specific Integrated Circuit

AUROC Area Under the Receiver Operating Characteristic Curve

BCI Brain-Computer Interface

BEOL Back-End-Of-The-Line

BL Bit Line

BMI Brain-Machine Interface

BNN Binarized Neural Network

C2C Cycle-to-cycle

CA Cellular Automata

CAB Computational Analog Block

CAD Computer Automated Design

CBRAM Conductive Bridging Random-Access Memory

CF Conductive Filament

CMOS Complementary Metal–Oxide–Semiconductor

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DAC Digital-to-Analog Converter

DBS Deep Brain Stimulation

DL Deep Learning

DMNN Deep Memristive Neural Network

DNN Deep Neural Network

DRAM Dynamic Random-Access Memory

DSE Design Space Exploration

DVS Dynamic Vision Sensor

ECG Electrocardiography

ECog Electrocorticography

EDA Electronic Design Automation

EDP Energy Delay Product

EEG Electroencephalogram

ELM Extreme Learning Machine

EMG Electromyography

ESN Echo State Network

FC Fully Connected

FFT Fast Fourier Transform

FIPT Further Image Pre-processing Techniques

FIR Finite Impulse Response

FPAA Field Programmable Analog Array

FPGA Field Programmable Gate Array

FPR False Positive Rate

GPU Graphics Processing Unit

HDL Hardware Description Language

HFO High-Frequency Oscillations

HLS High Level Synthesis

HPC High Performance Computing

HPS Hard Processor System

IaaS Infrastructure-as-a-Service

IC Integrated Circuit

iEEG Intracranial Electroencephalography

IMC In-Memory Computing

IOC Intel FPGA SDK for OpenCL Offline Compiler

IoMT Internet of Medical Things

IoT Internet of Things

IPT Image Pre-processing Techniques

kNN k-Nearest Neighbor

LDMOS Laterally-Diffused Metal Oxide Semiconductor

LFP Local Field Potential

LFSR Linear-Feedback Shift Register

LIF Leaky Integrate and Fire

LSTM Long Short-Term Memory

LUT Lookup Table

MAC Multiply and Accumulate

MDLS Memristive Deep Learning Systems

MDNN Memristive Deep Neural Network

MIM Metal–Insulator–Metal

ML Machine Learning

MLP Multi-Layer Perceptron

MRAM Magnetoresistive Random-Access Memory

NLSR Nonlinear Least Squares Regression

NN Neural Network

NPU Neural Processing Unit

OSP Original Software Publication

PCA Principal Component Analysis

PCM Phase-Change Memory

PE Processing Element

PoC Point of Care

PPG Photoplethysmography

PSM Patient-Specific Modeling

PWM Pulse Width Modulation

QAT Quantization Aware Training

RAM Random-Access Memory

RF Random Forest

RMSA Root Mean Square Amplitude

RNN Recurrent Neural Network

RRAM Resistive Random-Access Memory

RTL Register Transfer Level

RTN Random Telegraph Noise

SC Stochastic Computing

SDK Software Development Kit

SDSP Spike-Driven Synaptic Plasticity

SGD Stochastic Gradient Descent

SIMD Single-Instruction-Multiple-Data

SL Select Line

SLURM Simple Linux Utility for Resource Management

SNN Spiking Neural Network

SoC System On a Chip

SOP Seizure Occurance Period

SOTA State-Of-The-Art

SPH Seizure Prediction Horizon

SPICE Simulation Program with Integrated Circuit Emphasis

SRAM Static Random-Access Memory

STDP Spike-Timing-Dependent Plasticity

STT-MRAM Spin-Transfer Torque Magnetic Random-Access Memory

SVD Singular Value Decomposition

SVM Support Vector Machine

SWAR SIMD Within a Register

SWEC Sleep-Wake-Epilepsy-Center

TDM Time-Division Multiplexing

TF TensorFlow

TFLOPS Tera Floating Point Operations Per Second

TPU Tensor Processing Unit

VLSI Very-large-scale Integration

VMM Vector-Matrix Multiplication

VRAM Video Random-Access Memory

WL Word Line

WRN Wide Residual Network

Chapter 1

Introduction

1.1 Background and Motivation

Recent technological advancements have resulted in the large scale availability of increasingly powerful CPU and GPU devices, which have revolutionized and dominated the ML domain by enabling the research and development of DL systems, that loosely mimic the functionality of the brain [36]. These systems are often comprised of millions to billions of floating-point parameters, and can be used to solve many challenging engineering tasks [37]. In contrast to traditional ML algorithms, DL systems do not usually require manual feature extraction, and they have a higher degree of task flexibility [36]. These major advantages have led to their widespread adoption across diverse research domains [38].

As the number of interconnected *smart* IoT devices continues to increase [39], the volume of data required to be processed by these systems is forecast to also increase, albeit, at an exponential rate [39]. Currently, in the IoT, most data is sent from edge devices and nodes to higher levels to be stored and/or processed using DL algorithms in large batches. This process consumes a significant amount of power and resources; in addition to introducing additional latency and privacy concerns, especially when sensitive data needs to be sent away to be processed. Consequently, there is a great demand for performant hardware capable of executing DL system workloads efficiently near the source of data collection, i.e., *near-sensor* or *in-sensor* computing systems [40]. Such systems would drastically reduce the amount of data required to be sent to and processed at higher levels of the IoT, and would reduce the amount of data required to be processed by power- and resource-hungry server farms.

Current traditional computing architectures are unsuitable to process data near- or in-sensor with low latency, due to the *von Neumann* bottleneck. The *von Neumann* bottleneck [41], or so-called *memory wall problem*, is a limitation on throughput of traditional computing architectures caused by implications of performance gaps between processor and memory units. If memory latency and bandwidth become insufficient to provide processors with enough instructions and data to continue computation, processors will effectively always be stalled waiting on memory. For conventional computing systems, this is predicted to increase in severity [42], as Moore's law, which observes that the number of transistors in a dense Integrated Circuit (IC) doubles about every two years,

inevitably fails [43].

Significant research efforts have been made of late to develop novel DL hardware architectures for IoT devices which are capable of operating on resource-constrained devices. These architectures usually offer improved resource efficiency at the cost of the reduced ability to be customized for general-purpose operation [1]. Moreover, at the cost of a marginal reduction in accuracy, using approximate and stochastic computing techniques, the resource consumption of these architectures can be further reduced [44, 45].

The primary motivation of this thesis is to contribute in-part to this collective effort [35] by leveraging software hardware co-optimization for the simulation and implementation of novel hardware architectures using two alternate technologies: memristive devices and FPGAs. Memristive devices can be arranged in crossbar architectures to avoid the memory wall issue in part by replacing Processing Elements (PEs) in the memory domain, unlike in digital In-Memory Computing (IMC) processors, whereas FPGAs can be used to deploy customized computation pipelines at reduced precisions.

Detailed motivations can be summarized as follows: (i) To reduce the power and resource consumption of DL accelerators to facilitate operation in resource-constrained environments. (ii) To reduce the amount of data required to be processed by higher-level layers of the IoT. (iii) To improve current design flows of DL systems using emerging technologies through the use of novel software hardware co-optimization methodologies. (iv) To investigate alternative computing approaches including brain-inspired neuromorphic and IMC for practical engineering tasks.

1.2 Research Questions

The background and motivation, as discussed and presented in Section 1.1, was used to formulate the following three research questions, which this thesis addresses:

1. How novel FPGA and Memristive hardware arcitectures can reduce the power and resource requirements of DL systems/accelerators?

2. Specifically, for mixed-signal hybrid CMOS-Memristive architectures, can current design flows be improved upon?

3. How can these technologies be used to solve practical engineering problems in resource-constrained environments, such as on IoT devices?

In the following sections, a summary of the original contributions of this thesis is provided. In addition, the organization of this thesis is discussed and related to the aforementioned research questions.

1.3 Original Contributions

This thesis comprises of seven significant original research contributions:

1. In [1], a tutorial was provided, which described how various technologies including emerging memristive devices, FPGAs, and CMOS can be used to develop efficient DL accelerators to solve a wide variety of diagnostic, pattern recognition, and signal processing problems in healthcare. The tutorial also explored how spiking neuromorphic processors can complement their DL counterparts for processing biomedical signals. Different hardware platforms were bench-marked by performing a sensor fusion signal processing task combining Electromyography (EMG) signals with computer vision. Novel comparisons were made between dedicated neuromorphic processors and embedded AI accelerators in terms of inference latency and energy. Finally, an analysis of the field was provided, and perspective was shared on the advantages, disadvantages, challenges, and opportunities that various accelerators and neuromorphic processors introduce to healthcare and biomedical domains.

2. In [2], a survey of existing simulation frameworks and related tools used to model large-scale MDLS was provided. Direct performance comparisons of both traditional and modernized open-source simulation frameworks were performed, and insights were provided into future modeling and simulation strategies and approaches.

3. In [3], a novel low-latency parallel CNN architecture that has between 2-2,800x fewer network parameters compared to State-Of-The-Art (SOTA) CNN architectures was proposed. The proposed system was implemented onto analog crossbar arrays comprising RRAM devices, and a comprehensive benchmark was provided by simulating, laying out, and determining hardware requirements of the CNN component of the system. The proposed system is the first to parallelize the execution of convolution layer kernels on separate analog crossbars to enable 2 orders of magnitude reduction in latency compared to SOTA hybrid Memristive-CMOS DL accelerators. Furthermore, effects of non-idealities were investigated, and QAT was used to mitigate the performance degradation due to low ADC/Digital-to-Analog Converter (DAC) resolution. In addition, a stuck weight offsetting methodology was presented to mitigate performance degradation due to stuck $R_{\text{ON}}/R_{\text{OFF}}$ memristor weights, recovering up to 32% accuracy, without requiring retraining.

4. In [4], the stochasticity during switching of probabilistic CBRAM devices was exploited to efficiently generate stochastic bit streams in order to perform DL parameter optimization, reducing the size of MAC units by 5 orders of magnitude. It was demonstrated that in using a 40-nm CMOS process, the proposed architecture occupies 1.55mm^2 and consumes approximately 167μW when optimizing parameters of a CNN while it is being trained for a character recognition task. This is a significant improvement compared to other related works in the literature; at the cost of a marginal reduction in accuracy.

5. In [5], an open-source framework, entitled *MemTorch*, was developed for customized large-scale memristive DL simulations, with a refined focus on the co-simulation of device non-idealities. MemTorch was the first open-source framework to model various non-linear device and circuit aspects, including passive crossbars, which were modeled using customized Compute Unified Device Architecture (CUDA) kernels. MemTorch adopted a modernized software engineering methodology and integrates directly with the well-known PyTorch ML library, and has been downloaded over 1,000 times.

6. In [6], a novel generalized empirical Metal-Oxide RRAM endurance and retention model for use in large-scale DL simulations was proposed. The presented model was the first to unify retention-endurance modeling while taking into account time, energy, SET-RESET cycles, device size, and temperature. The model was compared to other models in the literature, and its versatility was demonstrated by applying it to experimental data from a number of fabricated devices. Furthermore, the model was used for CIFAR-10 dataset classification using a large-scale MDLS implementing the MobileNetV2 architecture. Results demonstrated that, even when ignoring other device non-idealities, retention and endurance losses significantly affect the performance of DL networks.

7. In [7], the first FPGA-accelerated deterministically binarized DNNs, tailored toward weed species classification for robotic weed control were introduced. It was demonstrated that the presented networks significantly outperformed their GPU-accelerated counterparts when they were trained and benchmarked using a publicly available weed species dataset, named DeepWeeds, which includes close to 18,000 weed images. A $>$7-fold decrease in power consumption was reported, while performing inference on weed images 2.86 times faster compared to the best performing baseline full-precision GPU implementation. These significant benefits were gained whilst losing only 1.17% of accuracy.

It is noted that not all research outputs as a result of research conducted during my PhD candidature are included in this thesis document. A complete list of publications is included in non-numbered Section located prior to the thesis abstract entitled *List of Publications Not Included in This Thesis*.

1.4 Thesis Organization

As illustrated in Fig 1.1, this thesis is organized into nine chapters to convey all of the original research contributions in a coherent way. The current Chapter, i.e., the Introduction, highlighted in magenta, introduces the research background and motivation. In addition, research questions are formulated, and the key original contributions of this thesis are summarized.

The literature review component of this thesis is presented in Chapter 2 and Chapter 3, which are highlighted in gold in Fig 1.1. Chapter 2 investigates the use of neuromorphic asynchronous pro-

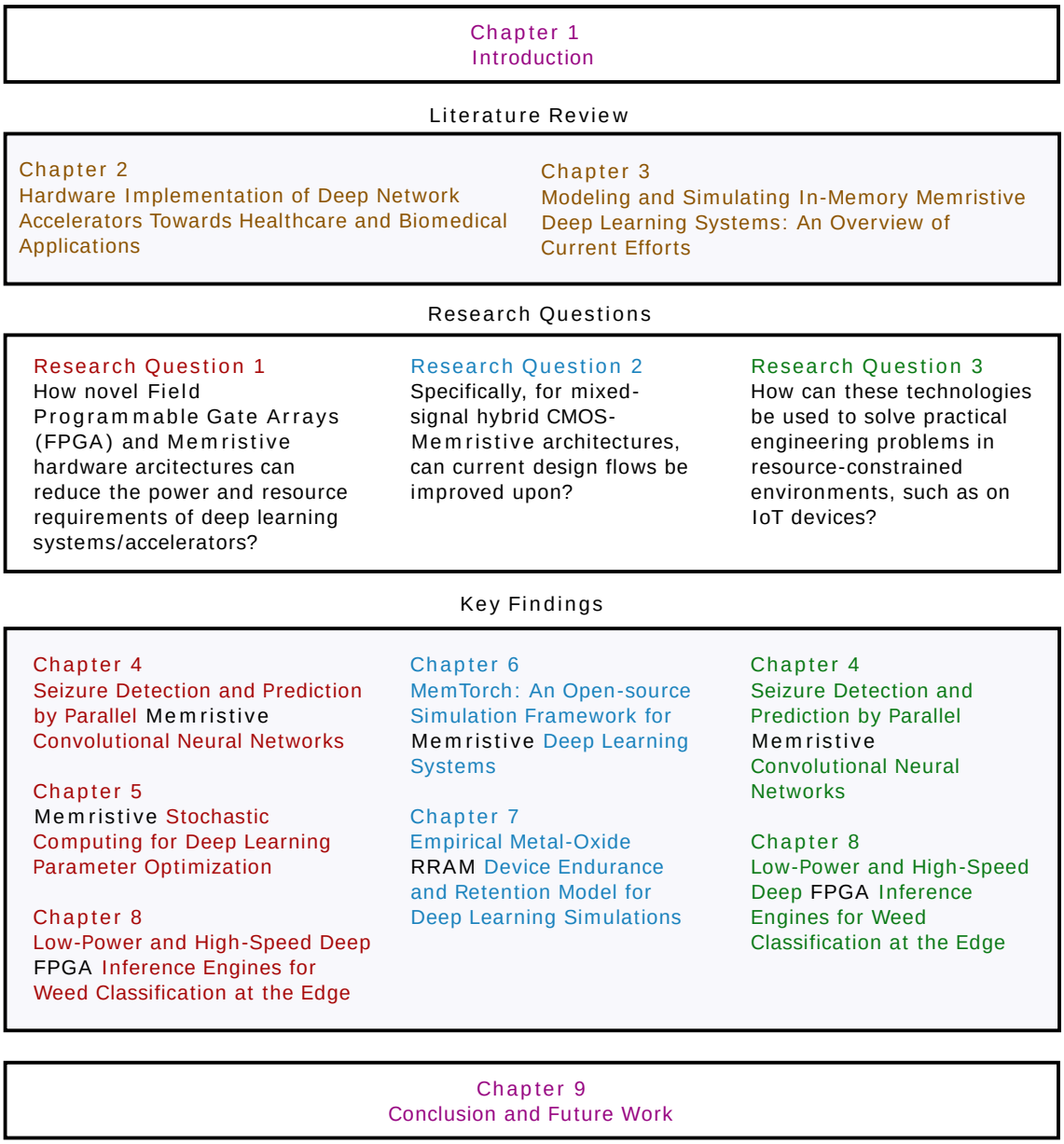


Figure 1.1: Thesis structure.

cessors, in addition to traditional synchronous DL accelerators, for healthcare and biomedical applications; a domain where IoT devices are commonly used to process data in resource-constrained environments. Different hardware technologies, including memristors and FPGAs are investigated and compared. Chapter 3 investigates research efforts in modeling in-memory MDLSs, i.e., DL systems that perform in-memory computing operations using memristors. It was determined that (i) for complex engineering tasks, currently, traditional synchronous DL accelerators outperform neuromorphic asynchronous processors; and (ii) a greater improvement in performance can be gained by using memristive technologies, rather than FPGAs, for DL systems/accelerators. This motivated the key findings of this thesis, which are presented in Chapters 4-8.

For sake of clarity, each research question is highlighted using a different color, and chapters are categorized using the aforementioned formulated research questions. In addition, the terms *FPGA*, *Memristive*, and *RRAM* are bolded. In lieu of an abstract at the beginning of each Chapter, a short text passage is included introducing the Chapter and relating it to the formulated research questions.

As can be seen in Fig 1.1, Chapters 4, 5, and 8 address the first research question. In these chapters, many different approaches are investigated and proposed for reducing the power and resource requirements of deep learning systems and architectures. These include the parallelization of operations using memristive devices, the exploitation of the stochasticity of memristive devices to perform random number generation, and the use of FPGA devices to execute DL workloads at a reduced level of precision using customized digital hardware.

Chapters 6 and 7 address the second research question. Chapter 6 investigates design flow improvements, and how the current design flows have been improved by developing a comprehensive simulation framework for MDLS, whereas Chapter 7 contributes improvements using a computationally efficient empirical device model designed for use in large DL simulations.

The third and last research question is addressed in Chapters 4 and 8. Both of these Chapters also addressed the first research question, i.e., they have novel aspects in (i) architecture and (ii) application. In Chapter 4, memristors were used within a MDLS to perform epileptic seizure prediction and detection tasks. In Chapter 8, FPGAs were used to perform low-power and high-speed DL inference for a weed species classification task.

Finally, the thesis is concluded in Chapter 9, *Conclusion and Future Work*. In Fig 1.1, this Chapter is highlighted in the same color as the Introduction to indicate a strong link/connection. In the Conclusion, the findings in other chapters are summarized with respect to the research questions formulated in the Introduction, and future research directions are discussed.

Chapter 2

Hardware Implementation of Deep Network Accelerators Towards Healthcare and Biomedical Applications

The advent of dedicated Deep Learning (DL) accelerators and neuromorphic processors has brought on new opportunities for applying both Deep and Spiking Neural Network (SNN) algorithms to a variety of healthcare and biomedical applications. This can facilitate the advancement of medical Internet of Things (IoT) systems and Point of Care (PoC) devices in resource-constrained environments, such as on IoT devices. In this Chapter, the first part of the literature review component of this thesis is presented. A tutorial is provided describing how various technologies including emerging memristive devices, Field Programmable Gate Arrays (FPGAs), and Complementary Metal Oxide Semiconductor (CMOS) can be used to develop efficient DL accelerators to solve a wide variety of diagnostic, pattern recognition, and signal processing problems in healthcare. In addition, direct comparisons of different hardware technologies are made, and insights are presented. The investigations performed and reported in this Chapter motivated the refined focus of this thesis on synchronous DL accelerators in Chapters 3-8.

2.1 Introduction

Artificial intelligence is uniquely poised to cope with the growing demands of the universal healthcare system [46]. The healthcare industry is projected to reach over 10 trillion dollars by 2022, and the associated workload on medical practitioners is expected to grow concurrently [47]. As the reliability of DL improves, it has pervaded various facets of healthcare from monitoring [48, 49], to prediction [50], diagnosis [51], treatment [52], and prognosis [53]. Fig. 2.1(a) shows how data collected from the patient, which may be a combination of bio-samples, medical images, temperature, movement, etc., can be processed using a smart DL system that monitors the patient for anomalies and/or to predict diseases. DL systems can be used to recommend treatment options and prognosis, which further affect monitoring and prediction in a closed-loop scenario.

The capacity of Artificial Intellegence (AI) to meet or exceed the performance of human experts

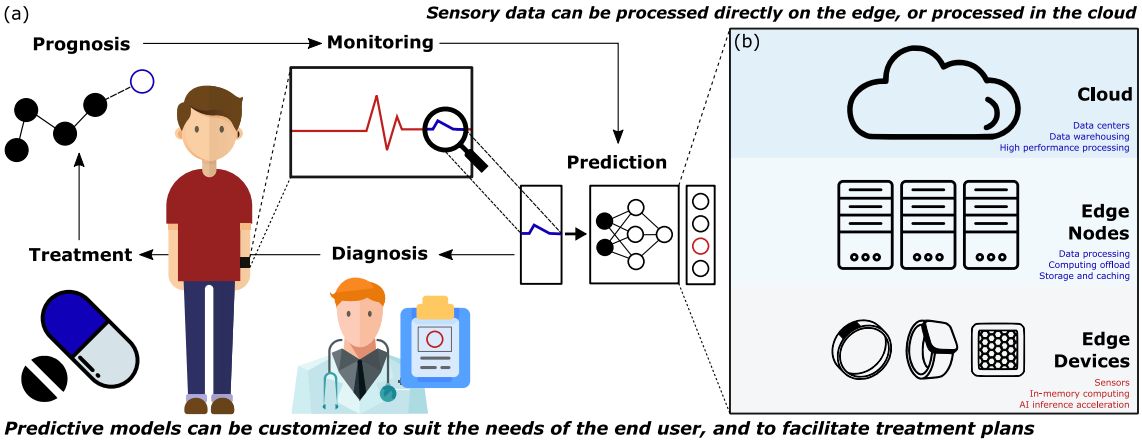


Figure 2.1: A depiction of (a) the usage of DL in a smart healthcare setting, which typically involves monitoring, prediction, diagnosis, treatment, and prognosis. The various parts of the DL-based healthcare system can run on (b) the three levels of the IoT, i.e. edge devices, edge nodes, and the cloud. However, for healthcare IoT and PoC processing, edge learning and inference is preferred.

in medical-data analysis [54–56] can, in part, be attributed to the continued improvement of high-performance computing platforms such as GPUs [57] and customized ML hardware [58]. These can now process and learn from a large amount of multi-modal heterogeneous general and medical data [59]. This was not readily achievable a decade ago.

While the field of DL has been growing at an astonishing rate in terms of performance, network size, and training run time, the development of dedicated hardware to process DL algorithms is struggling to keep up. Concretely, the compute loads of DL have doubled every 3.4 months since 2012. Moore's Law targets the doubling of compute power every 18-24 months, and appears to be slowing down [60]. The progress in hardware accelerator development currently relies on advances by a handful of technology companies, most notably Nvidia and its GPUs [61, 62] and Google and its Tensor Processing Units (TPUs) [58], in addition to new startups and research groups developing Application-Specific Integrated Circuits (ASICs) for DL training and acceleration. While there are significant advances in tailoring deep network models and algorithms for various healthcare and biomedical applications [63], most computationally expensive deep networks are trained on either GPUs or in data centers [57, 64]. The latter typically requires access to cloud computing services which is not only costly and comes with high power demands, but also compromises data privacy. This is distinct to the effective deployment of DL at the edge on an increasing number of medical IoT devices [65] and PoC systems [66], as illustrated in Fig. 2.1(b). Edge learning and inference enables the option to move processing away from the cloud. This is critical for highly sensitive medical data and offline operation. Edge-based processing must combine compactness, low-power, and rapid (high throughput) at a low-cost, to make smart health monitoring viable and affordable for integration into human life [67].

Specialized embedded DL accelerators, such as the Nvidia Jetson and Xavier series [68], and

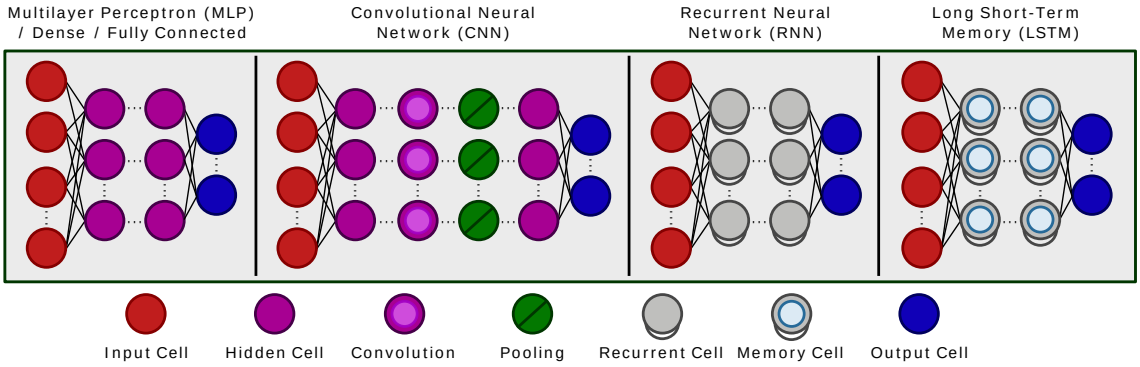


Figure 2.2: Popular ANN structures. MLP/Dense/Fully Connected are typically well-suited for cross-sectional quantitative data, whereas RNNs and LSTMs networks are optimized for sequential data. CNNs are equipped for both types.

the Movidius Neural Compute Stick [69, 70], have shown the promise of edge computing. More recently, the Nvidia Clara Embedded was released as a healthcare-specific edge accelerator. This is a computing platform for edge-enabled AI on the Internet of Medical Things (IoMT). However, embedded devices remain relatively power hungry and costly, and many state-of-the-art algorithms far exceed the memory bandwidth of resource-constrained devices. They are not yet ideal learning/inference engines for ambient-assisted precision medicine systems. There is a need for innovative systems which can satisfy the stringent requirements of healthcare edge devices to be made affordable to the community at large scales.

To that end, in this Chapter, we focus on the use of three various hardware technologies to develop dedicated deep network accelerators which will be discussed from a biomedical and healthcare application point-of-view. The three technologies that we cover here are CMOS, memristors, and FPGAs. It is worth noting that, while our focus targets edge inference engines in the biomedical domain, the techniques and hardware advantages discussed here are likely to be useful for efficient offline deep network learning, or online on-chip learning. Herein, the term DL 'accelerator' is used to refer to a device that is able to perform DL inference and potentially training.

This tutorial on DL accelerators within the biomedical sphere commences with a brief introduction to artificial and spiking neural networks. Next, we introduce the computational demands of DL by shedding light on why they are power- and resource-intensive. This will justify the need for application specific hardware platforms. After that, we discuss recent hardware advances which have led to improvements in training and inference efficiency. These improvements ultimately guide us to viable edge inference engine options.

After reviewing the literature on these DL accelerators, we quantify the performance of various algorithms on different types of DL processors. The results allow us to draw a perspective on the potential future of spike-based neuromorphic processors in the biomedical signal processing domain. Based on our analysis and perspective, we conjecture that, for edge processing, neuromorphic computing and SNNs [71] will likely complement DL inference engines, either through

signaling anomalies in the data or acting as 'intelligent always-on watchdogs' which continuously monitor the data being recorded, but only activate further processing stages if and when necessary.

We expect this tutorial, review, and perspective to provide guidance on the history and future of DL accelerators, and the potential they hold for advancing healthcare. Our contributions are summarized as follows:

1. We are the first to discuss the use of three different emerging and established hardware technologies for facilitating DL acceleration, with a focus on biomedical applications;
2. We provide tutorial sections on how one may implement a typical biomedical task on FPGAs or simulate it for deployment on memristive crossbars;
3. We are the first to discuss how event-based neuromorphic processors can complement DL accelerators for biomedical signal processing;
4. We provide open-source code and data to enable the reproduction of our results.

The remainder of this Chapter is organized as follows. In Section 2.2, we define the technical terminology that is used throughout this Chapter and cover the working principles of artificial and spiking neural networks. We also introduce a biomedical signal processing task for hand-gesture classification, which is used for benchmarking the different technologies and algorithms discussed in this Chapter. In Section 2.3, we step through the design, simulation, and implementation of DNNs using different hardware technologies. We show sample cases of how they have been deployed in healthcare settings. Furthermore, we demonstrate the steps and techniques required to simulate and implement hardware for the benchmark hand-gesture classification task using memristive crossbars and FPGAs.

In Section 2.4, we provide our perspective on the challenges and opportunities of both DNNs and SNNs for biomedical applications and shed light on the future of spiking neuromorphic hardware technologies in the biomedical domain. Section 2.5 concludes the Chapter.

2.2 Deep Artificial and Spiking Neural Networks

2.2.1 Nomenclature of Neural Network Architectures

Although most DNNs reported in literature are ANNs, DNNs refer to more than one hidden layer, independently of whether the architecture is fully connected, convolutional, recurrent, ANN or SNN, or of any other structure. For example, the most widely used DNN type in image processing, i.e. a CNN, can be physically implemented as an ANN or SNN, and in both cases it would be 'deep'. However, in this Chapter, whenever we use the terms 'deep', DL, or deep network, we refer to Deep Artificial Neural Networks. For Deep Spiking Neural Networks, we simply use the term SNN.

2.2.2 Deep Artificial Neural Networks

Traditional ANNs and their learning strategies that were first developed several decades ago [72] have, in the past several years, demonstrated unprecedented performance in a plethora of challenging tasks which are typically associated with human cognition. These have been applied to medical image diagnosis [73] and medical text processing [74], using DNNs.

Fig. 2.2 illustrates a simplified overview of the structure of some of the most widely-used DNNs. The most conventional form of these architectures is the Multi-Layer Perceptron (MLP). Increasing the number of hidden layers of perceptron cells is widely regarded to improve hierarchical feature extraction which is exploited in various biomedical tasks, such as seizure detection from EEG [75, 76]. CNNs introduce convolutional layers, which use spatial filters to encourage spatial invariance. CNNs often include pooling layers to downsample their outputs to reduce the search space for subsequent convolutional layers. CNNs have been widely used in medical and healthcare applications, as they are very well-suited for spatially structured data. Their use in medical image analysis [77] will form a major part of our discussions in subsequent sections.

RNNs are another powerful network architecture recently used both individually [78], and in combination with CNNs [79], in biomedical applications. RNNs introduce recurrent cells with a feedback loop, and are especially useful for processing sequential data such as temporal signals and time-series data, e.g. Electrocardiography (ECG) [79], and medical text [80]. The feedback loop in recurrent cells gives them a memory of previous steps and builds a dynamic awareness of changes in the input. The most well-known type of RNNs are LSTMs which are designed to mine patterns in data sequences using their short-term memory of distant events stored in their memory cells. LSTMs have been widely used for processing biomedical signals such as ECGs [78, 81]. Although there are many other variants of DNN architectures, we will focus on these most commonly used types.

Automatic hierarchical feature extraction

The above mentioned DNNs learn intricate features in data through multiple computational layers across various levels of abstraction [36]. The fundamental advantage of DNNs is that they mine the input data features automatically, without the need for human knowledge in their supervised learning loop. This allows deep networks to learn complex features by combining a hierarchy of simpler features learned in their hidden layers [36].

Learning algorithms

Learning features from data in a DNN, e.g. the networks shown in Fig. 2.2, is typically achieved by minimizing a loss function. In most cases, this is equivalent to finding the maximum likelihood using the cross-entropy between training data and the learned model distribution. Loss function minimization is achieved by optimizing the network parameters (weights and biases). This opti-

mization process minimizes the loss function from the final network layer backward through all the network layers and is therefore called backpropagation. Widely used optimization algorithms in DNNs include Stochastic Gradient Descent (SGD) and those that use adaptive learning rates [36].

Backpropagation in DNNs is computationally expensive

Despite the continual improvement of hardware platforms for running and training DNNs, reducing their power consumption is a computationally formidable task. One of the dominant reasons is the feed-forward error backpropagation algorithm, which depends on thousands of epochs of computationally intensive VMM operations [72], using huge datasets that can exceed millions of data points. These operations, if performed on a conventional von Neumann architecture which has separate memory and processing units, will have a time and power complexity of order $\mathcal{O}(N^2)$ for multiplying a vector of length N in a matrix of dimensions $N \times N$.

In addition, an artificial neuron in DNNs calculates a sum-of-products of its input-weight matrix pairs. For instance, a CNN spatially structures the sum-of-products calculation into a VMM operation. In digital logic, an adder tree can be used to accumulate a large number of values. This, however, becomes problematic in DNNs when one considers the sheer number of elements that must be summed together, as each addition requires one cycle.

Transfer learning

A major assumption when training DNNs is that both training and test samples are drawn from the same feature space and distribution. When the feature space and/or distribution changes, DNNs should be retrained. Rather than training a new model from scratch, trained parameters from an existing model can be fixed, tuned, or adapted [82]. This process of transfer learning can be used to greatly reduce the computational expense of training DNNs.

In the medical imaging domain, transfer learning from natural image datasets, particularly ImageNet [83], using standard large models and corresponding pretrained weights has become a de-facto method to speed up training convergence and to improve accuracy [84]. Transfer learning

Table 2.1: Number of weights and MAC operations in various CNN architectures for a single image and for video processing at 25 frames per second.

Network Architecture	**Weights**	**MACs**	**@ 25 FPS**
AlexNet	61 M	725 M	18 B
ResNet-18	11 M	1.8 B	45 B
ResNet-50	23 M	3.5 B	88 B
VGG-19	144 M	22 B	550 B
OpenPose	46 M	180 B	4500 B
MobileNet	4.2 M	529 M	13 B

can also be used to leverage personalized anatomical knowledge accumulated over time to improve the accuracy of pre-trained CNNs for specific patients [85], i.e., to perform patient-specific model tuning. This is an important topic in biomedical application domains, which will be further discussed in Section 2.4.6.

2.2.3 DL Accelerators

In Table 2.1, we depict some popular CNN architectures, accompanied with the total number of weights, and MAC operations that must be computed for a single image (input resolutions of 656×468 for OpenPose, 224×224 for the rest). This table highlights two key facts. Firstly, MACs are the dominant operation of DNNs. Therefore, hardware implementations of DNNs should strive to parallelize a large number of MACs to perform effectively. Secondly, there are many predetermined weights that must be called from memory. Reducing the energy and time consumed by reading weights from memory provides another opportunity to improve efficiency.

Consequently, significant research has been being conducted to achieve massive parallelism and to reduce memory access in DNN accelerators, using different hardware technologies and platforms as depicted in Fig. 2.3. Although these goals are towards general DL applications, they can significantly facilitate fast and low-power smart PoC devices [66] and healthcare IoT systems.

In addition to conventional DL accelerators, there have been significant research efforts to utilize biologically plausible SNNs for learning and cognition [86]. Spiking neuromorphic processors have also been used for biomedical signal processing [21, 22, 87]. Below, we provide a brief introduction to SNNs, which will be discussed as a method complementary to DL accelerators for efficient biomedical signal processing later in this Chapter. We will also perform comparisons among SNNs and DNNs in performing an EMG processing task.

2.2.4 Spiking Neural Networks

SNNs are neural networks that typically use Integrate-and-Fire neurons to dynamically process temporally varying signals (see Fig. 2.4(j)). By integrating multiple spikes over time, it is possible to reconstruct an analog value that represents the mean firing rate of the neuron. The mean firing rate is equivalent to the value of the activation function of ANNs. So in the mean firing rate limit, there is an equivalence between ANNs and SNNs. By using spikes as all-or-none digital events (Fig. 2.4(i)), SNNs enable the reliable transmission of signals across long distances in electronic systems. In addition, by introducing the temporal dimension, these networks can efficiently encode and process sequential data and temporally changing inputs [88]. SNNs can be efficiently interfaced with event-based sensors since they only process events as they are generated. An example of such sensors is the DVS, which is an event-based camera shown in Fig. 2.4(h). The DVS consists of a logarithmic photo-detector stage followed by an operational transconductance amplifier with a capacitive-divider gain stage, and two comparators. The ON/OFF spikes are generated every time the difference between the current and previous value of the input exceeds a

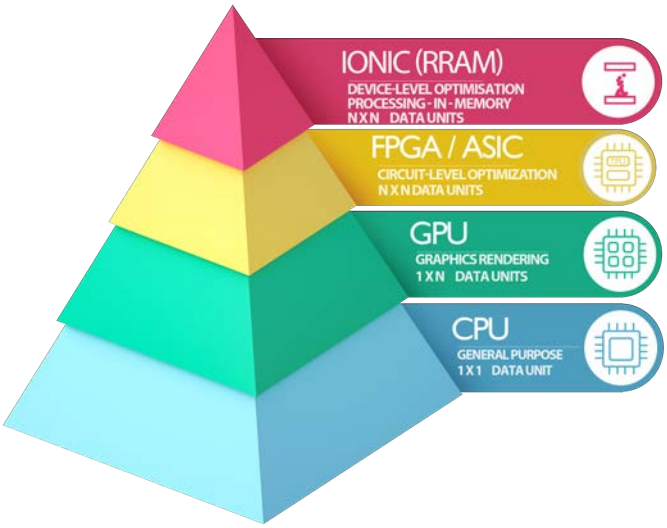


Figure 2.3: Typical hardware technologies for DNN acceleration. In this Chapter, we cover the top two layers of the pyramid, which include specialized hardware technologies for high-performance training and inference of DNNs. While the apex is labelled RRAM, this is intended to broadly cover all programmable non-volatile resistive switching memories e.g. CBRAM, MRAM, PCM, etc.

pre-defined threshold. The sign of the difference corresponds to the ON or OFF channel where the spike is produced. This is different to conventional cameras (Fig. 2.4(f)), which produce image frames (Fig. 2.4(g)). Intuitively, it makes sense to use asynchronous event-based sensor data in asynchronous SNNs, and synchronously generated frames (i.e., all pixels are given at a regular clock interval) through synchronous ANNs. But it is worth noting that conventional frames can be encoded as asynchronous spikes with frequencies that vary based on pixel intensity, and event streams can be integrated over time into synchronously generated time-surfaces [89, 90]. Event-based sensors have been used to process biomedical signals [22, 91] (Fig. 2.4(a)), which can be encoded to spike trains (Fig. 2.4(b)) to be processed by SNNs or be digitally sampled (Fig. 2.4(c)) for use in DNNs for learning and inference (Fig. 2.4(d)).

2.2.5 Benchmarking on a Biomedical Signal Processing Task

In Section 2.3 we will present a use-case of bio-signal processing where FPGA and memristive DNN accelerators are implemented and simulated. These are later compared to equivalent existing implementations[1] using DNN accelerators and neuromorphic processors from [21]. To perform comparisons, we use the same hand-gesture recognition task as in [21].

Tasks such as prosthesis control can be performed using EMG signals, hand-gesture classification, or a combination of both. Here, the adopted hand-gesture dataset [21] is a collection of

[1] `https://github.com/Enny1991/dvs_emg_fusion/blob/master/full_baseline.py`

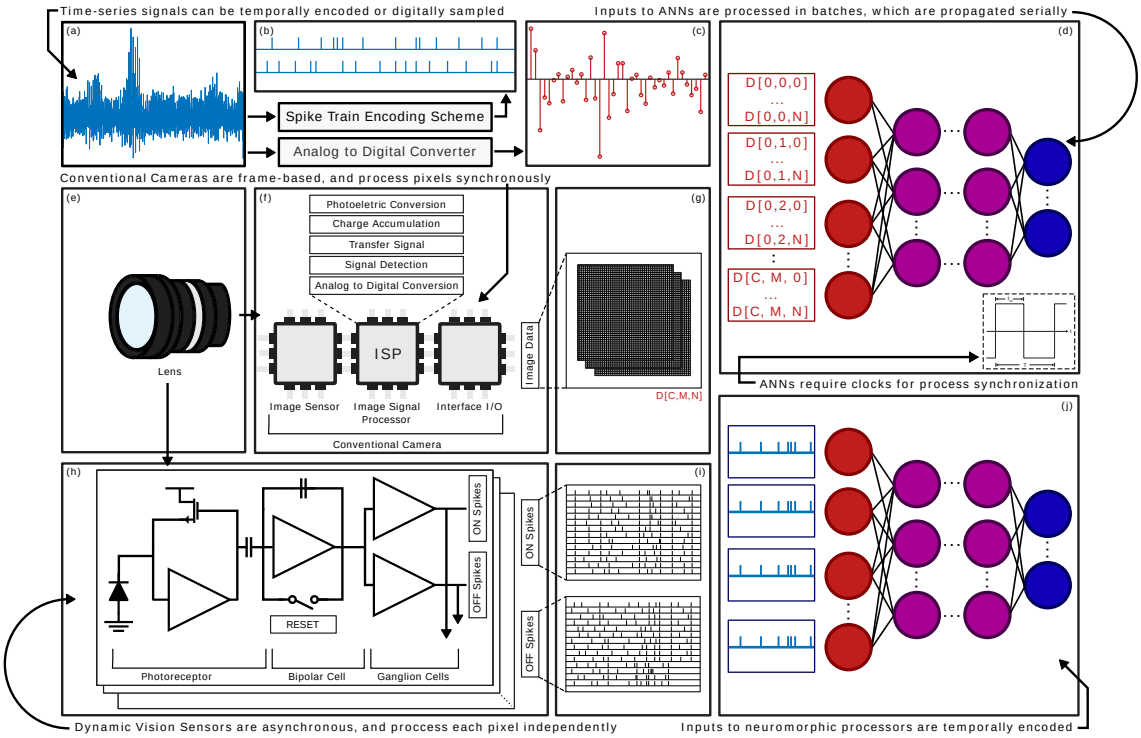


Figure 2.4: DNNs and SNN neuromorphic processors adopt different operation models. In DNNs, inputs are processed in batches which propagate serially. Consequently, they require clocks for process synchronization. SNNs are asynchronous and process temporally encoded inputs independently. Time series signals, such as the EMG signal presented in (a) can be either (b) temporally encoded using spike train encoding schemes such as [22], before being fed into (j) neuromorphic processors, or (c) digitally sampled, before being concatenated into batches, to be fed into (d) DNNs. Similarly, photographs captured from (e) lenses can be (i) temporally encoded into spike trains using (h) DVSs [23] or (f) digitally encoded using conventional cameras to build (g) image frames.

5 hand gestures recorded with two sensor modalities: muscle activity from a Myo armband that senses EMG electrical activity in forearm muscles, and a visual input in the form of DVS events. Moreover, the dataset provides accompanying video captured from a traditional frame-based camera, i.e., images from an Active Pixel Sensor (APS) to feed DNNs. Recordings were collected from 21 subjects including 12 males and 9 females between the ages 25 and 35, and were taken over three separate sessions.

For each implementation, we compare the mean and standard deviation of the accuracy obtained over a 3-fold cross validation, where each fold encapsulates all recordings from a given session. Additionally, for all implementations, we compare the energy and time required to perform inference on a single input, as well as the Energy-Delay Product (EDP), which is the average energy consumption multiplied by the average inference time.

2.3 DNN Accelerators towards healthcare and biomedical Applications

In this Section, we cover the use of CMOS and memristors in DL acceleration. We discuss how they use different strategies to achieve two of the key DNN acceleration goals, namely MAC parallelism and reduced memory access. We also discuss and review FPGAs as an alternative reconfigurable DNN accelerator platform, which has shown great promise in the healthcare and biomedical domains.

2.3.1 CMOS DNN Accelerators

General edge-AI CMOS accelerator chips can be used for DNN-enabled healthcare IoT and PoC systems. Therefore, within this Section, we first review a number of these chips and provide examples of potential healthcare applications they can accelerate. We will also explore some common approaches to CMOS-driven acceleration of AI algorithms using massive MAC parallelism and reduced memory access, which are useful for both edge-AI devices and offline data center scale acceleration.

Edge-AI DNN accelerators suitable for biomedical applications

The research and market for ASICs, which focus on a new generation of microprocessor chips dedicated entirely to machine learning and DNNs, have rapidly expanded in recent years. Table 2.2 shows a number of these CMOS-driven chips, which are intended for portable applications. There are many other examples of AI accelerator chips (for a comprehensive survey see [92]), but here we picked several prolific examples, which are designed specifically for DL using DNNs, RNNs, or both. We have also included a few general purpose AI accelerators from Google [93], Intel [94], and Huawei [95].

Although developed for general DNNs, the accelerators shown in Table 2.2 can efficiently realize portable smart DL-based healthcare IoT and PoC systems for processing image-based (medical imaging) or dynamic sequential medical data types (such as EEG and ECG). For instance, the table shows a few exemplar healthcare and biomedical applications that are picked based on the demonstrated capacity of these accelerators to run (or train [96]) various well-known CNN architectures such as VGG, ResNet, MobileNet, AlexNet, Inception, or RNNs such as LSTMs, or combined CNN-RNNs. It is worth noting that most of the available accelerators are intended for CNN inference, while only some [97–99] also include recurrent connections for RNN acceleration.

The Table shows that the total power per chip in most of these devices is typically in the range of hundreds of mW, with a few exceptions consuming excessive power of around 10 Watts [94, 95]. This is required to avoid large heat sinks and to satisfy portable battery constraints. The Table also shows the computing capability per unit time (column 'Computational Power (GOP/s)'). Regardless of power consumption, this column reveals the computational performance and consequently

the size of a network one can compute per unit time. It is demonstrated that several of these chips can run large and deep CNNs such as VGG and ResNet, which enable them to perform complex processing tasks within a constrained edge power budget.

Table 2.2: A number of recent edge-AI CMOS chips suitable for portable healthcare and biomedical applications.

CMOS Chip	**Core size (mm^2)**	**Technology (nm)**	**Computational Power (GOP/s)**	**Power (mW)**	**Power Efficiency (TOPS/W)**	**Potential Mobile and Edge-based Health-care and Medical Applications**
Cambricon -x [100]	6.38	65	544	954	0.5	Electrocorticography (ECog) analysis using a sparse VGG [101] for PoC diagnosis of cardiovascular diseases
Eyeriss [102]	12.25	65	17-42	278	0.06–0.15	- Mobile Image-based cancer diagnosis using VGG-16 [103], - Mobile diagnosis tool based on AlexNet for radiology, cardiology gastroenterology imaging [77]
Origami [104]	3.09	65	196	654	0.8	- Smart healthcare IoT edge device for heart health monitoring using a CNN-based ECG analysis [105] - Two-stage end-to-end CNN for human activity recognition [106]
ConvNet processor [107]	2.4	40	102	25-287	0.3–2.7	PoC Ultrasound processing using AlexNet [108]
Envision [109]	1.87	28	76-408	7.5-300	0.8–10	Multi-layer CNN for EEG/ECog feature extraction for epileptogenicity for epilepsy diagnosis on edge [110]
Neural processor [111]	5.5	8	1900-7000	39–1500	4.5-11.5	On edge classification of skin cancer using Inception V3 CNN [56]

LNPU [96]	16	65	600	43-367	25	- On edge learning/inference using VGG-16 for cancer diagnosis [103], - On edge AlexNet learning/inference for radiology, cardiology, gastroenterology imaging diagnosis [77]
DNPU [97]	16	65	300-1200	35-279	2.1–8.1	Parallel and Cascade RNN and CNN for acECG analysis for Brain-Computer Interface (BCI) [79]
Thinker [98]	14.44	65	371	293	1–5	- PoC conversion of respiratory organ motion ultrasound into MRI using a long-term recurrent CNN [112]
UNPU [99]	16	65	346-7372	3.2-297	3.08–50.6	- Intelligent pre-diagnosis medical support/consultation using a CNN-RNN [80] - A CNN-RNN for respiratory sound classification in wearable devices enabled by patient specific model tuning [113] - A CNN-LSTM for missing Photoplethysmographic data prediction [114]
Google Edge TPU [93]	25	-	4000	2000	2	- Low-cost and easy-to-access skin cancer detection using MobileNet V1 CNN [70] - On edge health monitoring for fall detection using LSTMs [115] - Robust long-term decoding in intracortical Brain-Machine Interfaces (BMIs) using MLP and Extreme Learning Machine (ELM) networks [116]

Intel Nervana NNP-I 1000 (Spring Hill) [94]	-	10	48000	10000	4.8	- Diagnosis using chest X-ray classification on ResNet CNN family [117] - Long term bowel sound segmentation using a CNN [118]
Huawei Ascend 310 [95]	-	12	16000	8000	2	- Cardiovascular monitoring for arrhythmia diagnosis from ECG using an LSTM [78] - Health monitoring by heart rate variability analysis using ECG analysis by a bidirectional LSTM [81]

For instance, it has been previously shown in [101] that VGG CNN (shown to be compatible with Cambricon-x [100]), can successfully analyze ECog signals. Therefore, considering the power efficiency of Cambricon-x, it can be used to implement a portable automatic ECog analyzer for PoC diagnosis of various cardiovascular diseases [119]. Similarly, Eyeriss [102] can run VGG-16, which is shown to be effective in diagnosing thyroid cancer [103]. In addition, Eyeriss can run AlexNet for several different medical imaging applications [77]. Therefore, Eyeriss can be used as a mobile diagnostic tool that can be integrated into or complement medical imaging systems at the PoC. Origami [104] is another CNN accelerator chip, which can be used for other healthcare applications based on a CNN. For instance, [105] proposes a CNN-based ECG analysis for heart monitoring, or [106] introduces a two-stage end-to-end CNN for human activity recognition for elderly and rehailitation monitoring, whereas Origami can be used to develop a smart healthcare IoT edge device. Similarly, the CNN processor proposed in [107] is shown to be able to run AlexNet, which can be deployed in a PoC ultrasound image processing system [108]. Envision [109] is another accelerator that has the capability to run large-scale CNNs. It can also be used as an edge inference engine for a multi-layer CNN for EEG/ECog feature extraction for epilepsy diagnosis [110]. Neural processor [111] is another CNN accelerator that is shown to be able to run Inception V3 CNN, which can be used for skin cancer detection [56] at the edge. LNPU [96] is the only CNN accelerator shown in Table 2.2, which unlike the others can perform both learning and inference of a deep network such as AlexNet and VGG-16, for applications including on edge medical imaging [77] and cancer diagnosis [103].

Unlike the above discussed chips that are capable of running only CNNs, DNPU [97], Thinker [98], and UNPU [99] are capable of accelerating both CNNs and RNNs. This feature makes them suitable for a wider variety of edge-based biomedical applications such as ECG analysis for BCI using a cascaded RNN-CNN [79], PoC MRI construction from motion ultrasounds using a long-

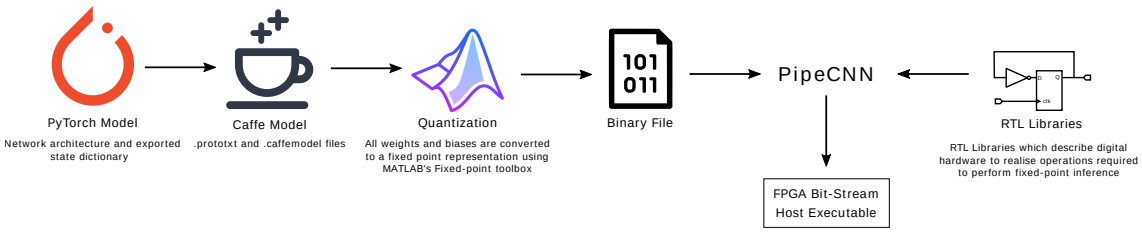


Figure 2.5: Compilation flow used to deploy an EMG classification CNN to an OpenVINO FPGA adopting fixed-point number representations using OpenCL.

term recurrent CNN [112], intelligent medical consultation using a CNN-RNN [80], respiratory sound classification in wearable devices enabled by patient specific model tuning using a CNN-RNN [113], or on-chip online and personalized prediction of missing Photoplethysmographic data [120].

Table 2.2 lists three general purpose AI accelerator chips, which have been deployed for low-cost and easy-to-access skin cancer detection using MobileNet V1 CNN [70], on edge health monitoring for fall detection using LSTMs [115], chest X-ray analysis using ResNet CNN [117], long term bowel sound monitoring and segmentation using a CNN [121], cardiovascular arrhythmia detection from ECG using an LSTM [78], or heart rate variability analysis from ECG signals through a bidirectional LSTM [81], just to name a few. These general-purpose chips have the potential to be used for other biomedical edge-based applications such as robust long-term decoding in intracortical BMIs using MLP and ELM networks in a sparse ensemble machine learning platform [122].

In addition to the edge-AI CNN and RNN acceleration chips or general ML chips mentioned in Table 2.2, there have been other works that have developed custom CMOS platforms for biomedical applications. Examples of these CMOS designs include [116] that has developed a 128-Channel ELM-based neural decoder for BMI, and [123] that has implemented an autoencoder neural network as part of a neural interface processor for brain-state classification and programmable-waveform neurostimulation.

Common approaches to CMOS-driven DL acceleration

Accelerators will typically target either data center use or embedded 'edge-AI' acceleration. Edge chips, such as those discussed above, must operate under restrictive power budgets (e.g., within thermal limits of 5 W) to cope with portable battery constraints. While the scale of tasks, input dimension capacity, and clock speeds will differ between edge-AI and modular data center racks, both will adopt similar principles in the tasks they seek to optimize.

Most of the accelerator chips, such as those discussed in Table 2.2, use similar optimization strategies involving reduced precision arithmetic [96, 99, 107, 109] to improve computational throughput. This is typically combined with architectural-level enhancements [97, 98, 100, 102, 111] to either reduce data movement (using in- or near-memory computing), heightened par-

allelism, or both. In addition, there are many other approaches commonly used to make neural network implementations more efficient. Examples of these include tensor decomposition, pruning, and mixed-precision data representation, which are often integrated in hardware with in-memory and near-memory computing. A thorough review of these approaches can be found in [124] and [125].

Sequential and combinational logic research is largely matured, so outside of emerging memory technologies, the dominant hardware benefits are brought on by optimizing data flow and architecture. An early example is the neuFlow system-on-chip (SoC) processor which relies on a grid of processing tiles, each made up of a bank of processing operators and a multiplexer based on-chip router [126]. The processing operator can serially perform primitive computation (MUL, DIV, ADD, SUB, MAX), or a parallelized 1D/2D convolution. The router configures data movement between tiles to support streaming data flow graphs.

Since the development of neuFlow, over 100 startups and companies have developed, or are developing, machine learning accelerators. The Neural Processing Unit (NPU) [127] generalizes the work from neuFlow by employing eight processing engines which each compute a neuron response: multiplication, accumulation, and activation. If a program could be partitioned such that a segment of it can be calculated using MACs, then it would be partially computed on the NPU. This made it possible to go beyond MLP neural networks. The NPU was demonstrated to perform Sobel edge detection and fast Fourier transforms as well.

NVIDIA coupled their expertise in developing GPUs with machine learning dedicated cores, namely, tensor cores, which are aimed at demonstrating superior performance over regular CUDA cores [62]. Tensor cores target mixed-precision computing, with their NVIDIA Tesla V100 GPU combining 672 tensor cores on a single unit. By merging the parallelism of GPUs with the application specific nature of tensor cores, their GPUs are capable of energy efficient general compute workloads, as well as 12 Tera Floating Point Operations Per Seconds (TFLOPSs) of matrix arithmetic.

Although plenty of other notable architectures exist (see Table 2.2), a pattern begins to emerge, as most specialized processors rely on a series of sub-processing elements which each contribute to increasing throughput of a larger processor [124, 125]. Whilst there are plenty of ways to achieve MAC parallelism, one of the most renowned techniques is the systolic array, and is utilized by Groq [128] and Google, amongst numerous other chip developers. This is not a new concept: systolic architectures were first proposed back in the late 1970s [129, 130], and have become widely popularized since powering the hardware DeepMind used for the AlphaGo system to defeat Lee Sedol, the world champion of the board game Go in October 2015. Google also uses systolic arrays to accelerate MACs in their TPUs, just one of many CMOS ASICs used in DNN processing [58].

2.3.2 FPGA DNNs

FPGAs are fairly low-cost reconfigurable hardware that can be used in almost any hardware prototyping and implementation task, significantly shortening the time-to-market of an electronic product. They also provide parallel computation, which is essential when simultaneous data processing is required such as processing multiple ECG channels in parallel. Furthermore, there exists a variety of High Level Synthesis (HLS) tools and techniques [131, 132] that facilitate FPGA prototyping without the need to directly develop time-consuming low-level Hardware Description Language (HDL) codes [133]. These tools allow engineers to describe their targeted hardware in high-level programming languages such as C to synthesize them to Register Transfer Level (RTL). The tools then offload the computational-critical RTL to run as kernels on parallel processing platforms such as FPGAs [10].

Accelerating DNNs on FPGAs

FPGAs have been previously used to realize mostly inference [11, 132, 134], and in some cases training of DNNs with reduced-precision-data [13], or hardware-friendly approaches [9]. For a comprehensive review of previous FPGA-based DNN accelerators, we refer the reader to [132].

Here, we demonstrate an example of accelerating DNNs to benchmark the biomedical signal processing task explained in Section 2.2.5. For our acceleration, we use fixed-point parameter representations on a Starter Platform for OpenVINO Toolkit FPGA using OpenCL. OpenCL [131] is an HLS framework for writing programs that execute across heterogeneous platforms. OpenCL specifies programming languages (based on C99 and C++11) for programming the compute devices and APIs to control and execute its developed kernels on the devices, where depending on the available computation resources, an accelerator can pipeline and execute all work items in parallel or sequentially.

Fig. 2.5 depicts the compilation flow we adopted. The trained DNN PyTorch model is first converted to `.prototxt` and `.caffemodel` files using Caffe. All weights and biases are then converted to a fixed point representation using MATLAB's Fixed-point toolbox using word length and fractional bit lengths defined in [135], prior to being exported as a single binary `.dat` file for integration with PipeCNN, which is used to generate the necessary RTL libraries, and to perform compilation of the host executable and the FPGA bit-stream. We used Intel's FPGA SDK for OpenCL 19.1, and provide all files used during the compilation shown in Fig. 2.5 in a publicly accessible complementary GitHub repository[2].

[2] `https://github.com/coreylammie/TBCAS-Towards-Healthcare-and-Biomedical-Applications/blob/master/FPGA/`

FPGA-based DNNs for biomedical applications

Despite the many FPGA-based DNN accelerators available [132], only a few have been developed specifically for biomedical applications such as ECG anomaly detection [136], or real-time mass-spectrometry data analysis for cancer detection [137], where the authors show that application-specific parameter quantization and customized network design can result in significant inference speed-up compared to both CPU and GPU. In addition, the authors in [138] have developed an FPGA-based BCI, in which a MLP is used for reconstructing ECog signals. In [139], the authors have implemented an EEG processing and neurofeedback prototype on a low-power but low-cost FPGA and then scaled it on a high-end Ultra-scale Virtex-VU9P, which has achieved 215 and 8 times power efficiency compared to CPU and GPU, respectively. For the EEG processing, they developed an LSTM inference engine.

It is projected that, by leveraging specific algorithmic design and hardware-software co-design techniques, FPGAs can provide $>$10 times energy-delay efficiency compared to state-of-the-art GPUs for accelerating DL [132]. This is significant for realizing portable and reliable healthcare applications. However, FPGA design is not as straightforward as high-level designs conducted for DL accelerators and requires skilled engineers and stronger tools, such as those offered by the GPU manufacturers.

2.3.3 Memristive DNNs

To achieve the two aforementioned key DNN acceleration goals, i.e. massive MAC parallelism and reduced memory access, many studies have leveraged memristors [140–143] as weight elements in their DNN and SNN [144, 145] architectures. Memristors are often referred to as the fourth fundamental circuit element, and can adapt their resistance (conductance) to changes in the applied current or voltage. This is similar to the adaptation of neural synapses to their surrounding activity while learning. This adaptation feature is integral to the brain's in-memory processing ability, which is missing in today's general purpose computers. This in-situ processing can be utilized to perform parallel MAC operations inside memory, hence, significantly improving DNN learning and inference. This is achieved by developing memristive crossbar neuromorphic architectures, which are projected to achieve approximately 2500-fold reduction in power and a 25-fold increase in acceleration, compared to state-of-the-art specialized hardware such as GPUs [140].

Memristive crossbars for parallel MAC and VMM operations

A memristive crossbar that can be fabricated using a variety of device technologies [145, 146] can perform analog MAC operations in a single time-step (see Fig. 2.6(a)). This reduces the time complexity to its minimum ($\mathcal{O}(1)$), and is achieved by carrying out multiplication at the place of memory, in a non-von Neumann structure. Using this well-known approach, VMM can be parallelized as demonstrated in Fig. 2.6(b), where the vector of size M represents input voltage

signals ($[V_1..V_M]$). These voltages are applied to the rows of the crossbar, while the matrix (of size $M \times N$), whose elements are represented as conductances (resistances), is stored in the memristive components at each cross point. Taking advantage of the basic Ohm's law ($I = V.G$), the current summed in each crossbar column represents one element of the resulting multiplication vector of size N.

Mapping memristive crossbars to DNN layers

Although implementing fully-connected DNN layers is straightforward by mapping the weights to crossbar point memristors and having the inputs represented by input voltages, implementing a complex CNN requires mapping techniques to convert convolution operations to MAC operations. A popular approach to perform this conversion is to use an unrolling (unfolding) operation that transforms the convolution of input feature maps and convolutional filters to MAC operations. We have developed a software platform named *MemTorch* [15], that will be introduced in subsequent sections, to perform this mapping as well as a number of other operations, for converting DNNs to MDNNs. The mapping process implemented in MemTorch is illustrated in the left panel in Fig. 2.7. The figure shows that the normal input feature maps and convolutional filters (shown in gray shaded area) are unfolded and reshaped (shown in the cyan shaded area) to be compatible with memristive crossbar parallel VMM operations. It is worth noting that the convolutional filters that can be applied to the input feature maps have a direct relationship with the required crossbar sizes. Furthermore, the resulting hardware size depends on the size of the input feature maps [8].

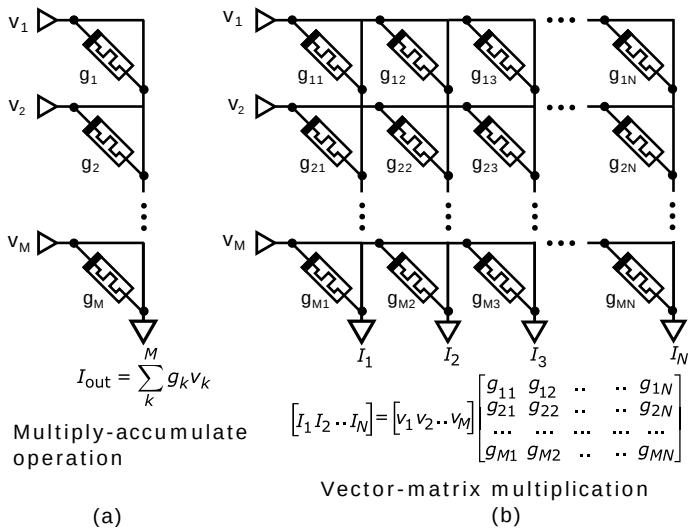


Figure 2.6: Memristive crossbars can parallelize (a) analog MAC and (b) VMM operations. Here, V represents the input vector, while conductances in the crossbar represent the matrix.

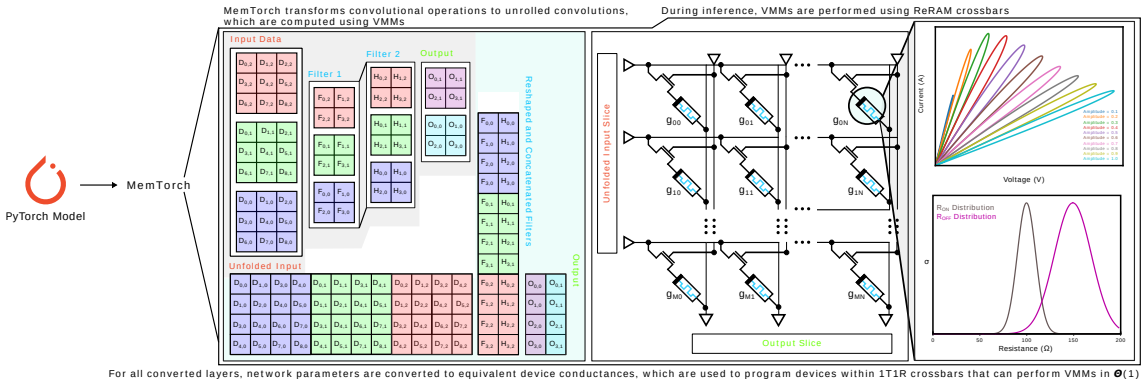


Figure 2.7: Conversion process of a DNN trained in PyTorch and mapped to a Memristive DNN using MemTorch [15], to parallelize MVMs using 1T1R memristive crossbars and to take into account memristor variability including finite number of conductance states and non-ideal R_{ON} and R_{OFF} distributions.

Peripheral circuitry for memristive DNNs

In addition to the memristive devices that are used as programmable elements in MDNN architectures, various peripheral circuitry is required to perform feed-forward error-backpropagation learning in MDNNs [142]. This extra circuitry may include: (i) a conversion circuit to translate the input feature maps to input voltages, which for programming memristive devices are usually Pulse Width Modulation (PWM) circuits, (ii) current integrators or sense amplifiers, which pass the current read from every column of the memristive crossbar to (iii) analog to digital converters (ADCs), which pass the converted voltage to (iv) an activation function circuit, for forward propagation, and for backward error propagation (v) the activation function derivative circuit. Other circuits required in the error backpropagation path include (vi) backpropagation values to PWM voltage generators, (vii) backpropagation current integrators, and (viii) backpropagation path ADCs. In addition, an update module that updates network weights based on an algorithm such as SGD is required, which is usually implemented in software. After the update, the new weight values should be written to the memristive crossbar, which itself requires Bit-Line (BL) and Word-line (WL) switch matrices to address the memristors for update, as well as a circuit to update the memristive weights. There are different approaches to implement this circuit such as that proposed in [147], while others may use software ex-situ training where the new weight values are calculated in software and transferred to the physical memristors through peripheral circuitry [142].

Memristive device nonidealities

Although ideal memristive crossbars have been projected to remarkably accelerate DNN learning and inference and drastically reduce their power consumption [140, 141], device imperfections observed in experimentally fabricated memristors impose significant performance degradation when

the crossbar sizes are scaled up for deployment in real-world DNN architectures, such as those required for healthcare and biomedical applications discussed in Section 2.3.1. These imperfections include nonlinear asymmetric and stochastic conductance (weight) update, device temporal and spatial variations, device yield, as well as limited on/off ratios [140]. To minimize the impact of these imperfections, specific peripheral circuitry and system-level mitigation techniques have been used [148]. However, these techniques add significant computation time and complexity to the system. It is, therefore, essential to take the effect of these nonidealities into consideration before utilizing memristive DNNs for any healthcare and medical applications, where accuracy is critical. In addition, there is a need for a unified tool that reliably simulates the conversion of a pre-trained DNN to a MDNN, while critically considering experimentally modeled device imperfections [15].

Conversion of DNN to MDNN while considering memristor nonidealities

Due to the significant time and energy required to train new large versions of DNNs for challenging cognitive tasks, such as biomedical and healthcare data processing [54,149], the training of the algorithms is usually undertaken in data centers [54, 58]. The pretrained DNN can then be transferred to be used on memristive crossbars. There are several different frameworks and tools that can be used to simulate and facilitate this transition [150]. In a recent study, we have developed a comprehensive tool named MemTorch, which is an open source, general, high-level simulation platform that can fully integrate any behavioral or experimental memristive device model into crossbar architectures to design MDNNs [15].

Here, we utilize the benchmark biomedical signal processing task explained in Section 2.2.5 to demonstrate how pretrained DNNs can be converted to equivalent MDNNs, and how non-ideal memristive devices can be simulated within MDNNs prior to hardware realization. The conversion process, which can be generalized to other biomedical models using MemTorch, is depicted in Fig. 2.7.

The targeted MDNNs are constructed by converting linear and convolutional layers from PyTorch pre-trained DNNs to memristive equivalent layers employing 1-Transistor-1-Resistor (1T1R) crossbars. A double-column scheme, in which two crossbars are used to represent positive and negative weight values, is used to represent network weights within memristive crossbars. The converted MDNN models are tuned using linear regression, as described in [15]. The complete and detailed process and the source code of the network conversion for the experiments shown in this Section are provided in a publicly accessible complementary Jupyter Notebook[3].

During the conversion, any memristor model can be used. For the benchmark task, a reference VTEAM model [151] is instantiated using parameters from Pt/Hf/Ti RRAM devices [152], to model all memristive devices within converted linear and convolutional layers. As already men-

[3] https://github.com/coreylammie/TBCAS-Towards-Healthcare-and-Biomedical-Applications/blob/master/MemTorch.ipynb

tioned, memristive devices have inevitable variability, which should be taken into account when implementing an MDNNs for learning and/or inference. Also, depicted in Fig. 2.7 are visualizations of two non-ideal device characteristics: the finite number of conductance states and device-to-device variability. Using MemTorch [15], not only can we convert any DNNs to an equivalent MDNNs utilizing any memristive device model, we are also able to comprehensively investigate the effect of various device non-idealities and variation on the performance of a possible MDNN, before it is physically realized in hardware.

In order to demonstrate an example which includes variability in our MDNN simulations, device-device variability is introduced by sampling R_{OFF} for each device from a normal distribution with $\bar{R}_{\text{OFF}}$ = 2,500Ω with standard deviation 2σ, and R_{ON} for each device from a normal distribution with $\bar{R}_{\text{ON}}$ = 100Ω with standard deviation σ.

In Fig. 2.8, for the converted memristive MLP and CNN that process APS hand-gesture inputs, we gradually increase σ from 0 to 500, and compare the mean test set accuracy across the three folds. As can be observed from Fig. 2.8, with increasing device-to-device variability, i.e. the variability of R_{ON} and R_{OFF}, the performance degradation increases across all networks. For all simulations, R_{ON} and R_{OFF} are bounded to be positive.

Memristive DNNs towards biomedical applications

Although some previous small-scale MDNNs have been simulated for biomedical tasks such as cardiac arrhythmia classification [153], or have been implemented on a physical programmable memristive array for breast cancer diagnosis [154], there is currently no large-scale MDNN, even at the simulation-level, which has realized any practical biomedical processing tasks.

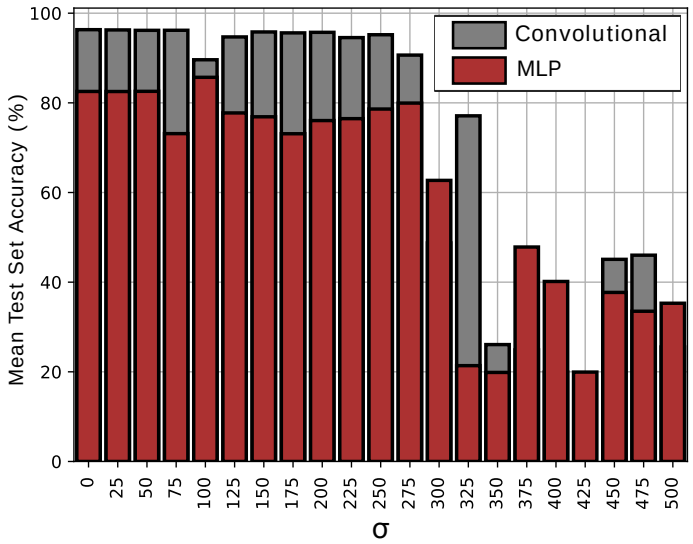


Figure 2.8: Simulation results investigating the performance of MDNNs for hand gesture classification adopting non-ideal Pt/Hf/Ti ReRAM devices. Device-device variability is simulated using MemTorch [15].

Similar to the recent advances in CMOS-driven DNN accelerator chips discussed in Section 2.3.1, there have been promises in partial [141] or full [142] realizations of MDNNs in hardware, which are shown to achieve significant energy saving compared to state-of-the-art GPUs. However, unlike their CMOS counterparts, these implementations have been only able to perform simple tasks such as MNIST and CIFAR classification. This is, of course, not suitable for implementing large-scale CNNs and RNNs, which as shown in Section 2.3.1 are required for biomedical and healthcare tasks dealing with image [77] or temporal [78] data types.

In addition, following similar optimization strategies as those used in CMOS accelerators, [155] has simulated the use of quantized and binarized MDNNs and their error tolerance in a biomedical ECG processing task and has shown their potential to achieve significant energy savings compared to full-precision MDNNs. However, due to the many intricacies in the design process and considering the peripheral circuitry that may offset the benefits gained by using MDNNs, full hardware design is required before the actual energy saving of such binarized MDNNs can be verified.

In the next Section, we provide our analysis and perspective on the use of the three hardware technologies discussed in this Section for DL-based biomedical and healthcare applications. We also discuss how SNN-based neuromorphic processors can benefit edge-processing for biomedical applications.

2.4 Analysis and Perspective

The use of ANNs trained with the backpropagation learning algorithm in the domain of healthcare and for biomedical applications such as cancer diagnosis [167] or ECG monitoring [168] dates back to the early 90s. These networks, were typically small-scale networks run on normal workstations. As they were not deep and did not have too many parameters, they did not demand high-performance accelerators. However, with the resurgence of CNNs in the early 2010s followed by the rapid spread of DNNs and large data-sets, came the need for high-speed specialized processors. This need resulted in repurposing GPUs and actively researching other hardware and design technologies including ASIC CMOS chips (see Table 2.2) and platforms [58], memristive crossbars and in-memory computing [8, 141, 142], and FPGA-based designs for DNN training [12, 13] and inference [11]. Despite notable progress in deploying non-GPU platforms for DL acceleration, similar to other data processing tasks, biomedical and healthcare tasks have mainly relied on standard technologies and GPUs. Currently, depending on the size of the required DNN, its number of parameters, as well as the available training dataset size, biomedical DL tasks are usually "trained" on high-performance workstations with one or more GPUs [57,64], on customized proprietary processors such as Google TPU [93], or on various Infrastructure-as-a-Service (IaaS) provider platforms, including Nvidia GPU cloud, Google Cloud, and Amazon Web Services, among others. This is mostly due to (i) the convenience these platforms provide using high-level languages such as Python; (ii) the availability of wide-spread and open-source

Table 2.3: Existing hardware implementations and hardware-based simulations of DNN accelerators used for healthcare and biomedical applications, and generic SNN neuromorphic processors utilized for biomedical signal processing. †Simulation-based.

Biomedical or Healthcare Task	**DNN/SNN Architecture**	**Hardware**
Image-based breast cancer diagnosis [54]	Ensemble of CNNs	CMOS (Google TPU)
Motor intention decoding [116]	ELM	CMOS
Spatial filtering and dimensionality reduction for brain-state classification [123]	Autoencoder	CMOS
Energy-efficient multi-class ECG classification [87]	Spiking RNN	CMOS
EMG signal processing [21]	Spiking CNN/MLP	CMOS
ECG signal processing [156]	Spiking RNN	CMOS
EMG signal processing [157]	Spiking RNN	CMOS
EMG signal processing [158]	Feed-forward SNN	CMOS
EMG and EEG signal processing [159]	Recurrent 3D SNN	CMOS
EEG and LFP signal processing [160]	TrueNorth-compatible CNN	CMOS
Real-time closed loop neural decoding [161, 162]	Spiking ELM	CMOS
ECG processing for cardiac arrhythmia classification [153]	MLP	Memristors†
Breast cancer diagnosis [154]	MLP	Programmable Memristor-CMOS system
ECG signal processing [155]	Binarized CNN	Memristors†
ECG arrhythmia detection for hearth monitoring [136]	MLP	FPGA
Mass-spectrometry for real-time cancer detection [137]	MLP	FPGA
ECog signal processing for BCI [138]	MLP	FPGA
Signal processing for fall detection [163]	MLP	FPGA
BCI-decoding of large-scale neural sensors [164]	LTSM	FPGA
EEG processing for energy-efficient Neurofeedback devices [139]	LTSM	FPGA and CMOS
PPG signal processing for heart rate estimation [165]	CNN/LTSM	FPGA and CMOS
Multimodal signal classification for physical activity monitoring [166]	CNN	FPGA and CMOS

DL libraries such as TensorFlow and PyTorch; and (iii) strong community and/or provider support in utilizing GPUs and IaaS for training various DNN algorithms and applications.

However, DL inference can benefit from further research and development on emerging and mature hardware and design technologies, such as those discussed in this Chapter, to open up new opportunities for deploying healthcare devices closer to the edge, paving the way for low-power and low-cost DL accelerators for PoC devices and healthcare IoT. Despite this fact, hardware implementations of biomedical and healthcare inference engines are very sparse. Table 2.3 lists a summary of the available hardware implementations and hardware-based simulations of DNNs used for healthcare and biomedical signal processing applications, using the three hardware technologies covered herein. In addition, the table shows existing biomedical signal processing tasks implemented on generic low-power spiking neuromorphic processors.

2.4.1 CMOS Technology Has Been the Main Player for DL Inference in the Biomedical Domain

Similarly to general-purpose GPUs, all other non-GPU DL inference engines at present are implemented in CMOS. Therefore, it is obvious that most of the future edge-based biomedical platforms would rely on these inference platforms. In Table 2.2, we listed a number of these accelerators that are mainly developed for low-power mobile applications. However, before the deployment of any edge-based DL accelerators for biomedical and healthcare tasks, some challenges need to be overcome. A non-exhaustive list of these obstacles include: (i) the power and resource constraints of available mobile platforms which, despite significant improvements, are still not suitable for high-risk medical tasks; (ii) the need to verify that a DL system can generalize beyond the distribution they are trained and tested on; (iii) bias that is inherent to datasets which may have adverse impacts on classification across different populations; (iv) confusion surrounding the liability of AI algorithms in high-risk environments [169]; and (v) the lack of a streamlined workflow between medical practitioners and DL. While the latter challenges are matters of legality and policy, the former issues highlight the fundamental need to understand where dataset bias comes from, and how to improve our understanding of why neural networks learn the features they do, such that they may generalize across populations in a manner that is safe for receivers of medical care.

In addition, to make the use of any accelerators possible for general as well as more complex biomedical applications, the field requires strong hardware-software co-design to build hardware that can be readily programmed for biomedical tasks. One successful co-design is the Google TPU [58], which has successfully been used to surpass human experts in medical imaging tasks [54]. Google has used a similar CMOS TPU technology to design inference engines [93], which are very promising as edge hardware to enable mobile healthcare care applications. The main reason for this promise is the availability of the established software platforms (such as TensorFlow Light) and the community support for the Google TPU.

Overall, great advancements have happened for DL accelerators in the past several years and

Table 2.4: Neuromorphic platforms used for biomedical signal processing.

Neuromorphic Chip	**DYNAP-SE**	**SpiNNaker**	**TrueNorth**	**Loihi**	**ODIN**
CMOS Technology	180 nm	ARM968, 130 nm	28 nm	14 nm FinFET	28 nm FD-SOI
Impl.	Mixed-signal	Digital	Digital ASIC	Digital ASIC	Digital ASIC
Neurons per core	256	1000 (1M cores)	256	Max 1k	256
Synapses per core	16k	1M	64k	114k-1M	64k
Energy per SOP	17 pJ @ 1.8V	Peak power 1W per chip	26 pJ @ 0.775	23.6 pJ @ 0.75V	12.7 pJ@0.55V
Size	38.5 mm^2	102 mm^2	-	60 mm^2	0.086 mm^2
Biosignal processing application	EMG [158], ECG [156], HFO [91]	EMG and EEG [159]	EEG and LFP [160]	EMG [21]	EMG [21]

they are currently stemming in various aspects of our life from self-driving cars to smart personal assistants. After overcoming a number of obstacles such as those mentioned above, we may be also able to widely integrate these DL accelerators in healthcare and biomedical applications. However, for some medical applications such as monitoring that requires always-on processing, we still need systems with orders of magnitude better power efficiency, so they can run on a simple button battery for a long time. To achieve such systems, one possible approach is to process data only when available and make our processing asynchronous. A promising method to achieve such goals is the use of brain-inspired SNN-based neuromorphic processors.

2.4.2 Towards Edge Processing for Biomedical Applications With Neuromorphic Processors

Although most of the efforts presented in this work focused on DNN accelerators, there are also notable efforts in the domain of SNN processors that offer complementary advantages, such as the potential to reduce the power consumption by multiple orders of magnitude, and to process the data in real time. These so-called neuromorphic processors are ideal for end-to-end processing scenarios, e.g., in wearable devices where the streaming input needs to be monitored in continuous time in an always-on manner.

There are already some works using both mixed analog-digital and digital neuromorphic platforms for biomedical tasks, showing promising results for always-on embedded biomedical systems. Table 2.4 shows a summary of today's large scale neuromorphic processors, used for

biomedical signal processing. The first chip presented in this table is DYNAP-SE [170], a multi-core mixed-signal neuromorphic implementation with analog neural dynamics circuits and event-based asynchronous routing and communication. The DYNAP-SE chip has been used to implement four of the seven SNN processing systems listed in Table 2.3. These SNNs are used for EMG [157, 158] and ECG [87, 156] signal processing. The DYNAP-SE was also used to build a spiking perceptron as part of a design to classify and detect High-Frequency Oscillations (HFO) in human intracranial EEG [91].

In [87, 156, 157] a spiking RNN is used to integrate the ECG/EMG patterns temporally and separate them in a linear fashion to be classifiable with a linear read-out. A Support Vector Machine (SVM) and linear least square approximation is used in the read out layer for [87, 156] and overall accuracy of 91% and 95% for anomaly detection were reached respectively. In [157], the timing and dynamic features of the spiking RNN on EMG recordings was investigated for classifying different hand gestures. In [158] the performance of a feedforward SNN and a hardware-friendly spiking learning algorithm for hand gesture recognition using superficial EMG was investigated and compared to traditional machine learning approaches, such as SVM. Results show that applying SVM on the spiking output of the hidden layer achieved a classification rate of 84%, and the spiking learning method achieved 74% with a power consumption of about $0.05\ mW$. This was compared to state-of-the-art embedded system showing that the proposed spiking network is two orders of magnitude more power efficient [34, 171].

The other neuromorphic platforms listed in Table 2.4 include digital architectures such as SpiNNaker [172], TrueNorth [173] and Loihi [174]. SpiNNaker has been used for EMG and EEG processing and the results show improved classification accuracy compared to traditional machine learning methods [159]. In [160], the authors developed a framework for decoding EEG and LFP using CNNs. The network was first developed in Caffe and the result was then used as a basis for building a TrueNorth-compatible neural network. The TrueNorth-compatible network achieved the highest classification, at approximately 76%. In [161, 162], the authors present a low-power neuromorphic platform named Spike-input Extreme Learning Machine (SELMA), which performs continuous state decoding towards fully-implantable wireless intracortical BMI. Recently, the benchmark hand-gesture classification introduced in Section 2.2.5, was processed and compared on two additional digital neuromorphic platforms, Loihi and ODIN/MorphIC [175,176]. A spiking CNN was implemented on Loihi and a spiking MLP was implemented on ODIN/MorphIC [21]. The results achieved using these networks are presented in Table 2.5.

On-chip adaptation and learning mechanisms, such as those present in some of the neuromorphic devics listed in Table 2.4, could be a game changer for personalized medicine, where the system can adapt to each patient's unique bio signature and/or drift over time. However, the challenge of implementing efficient on-chip online learning in these types of neuromorphic architectures has not yet been solved. This challenge lies on two main factors: *locality* of the weight update and *weight storage*.

Table 2.5: Comparison of conventional DNNs implemented on various hardware platforms with spiking DNN neuromorphic systems on the benchmark biomedical signal processing task of hand gesture recognition for both single sensor and sensor fusion, as explained in Section 2.2.5. The results of the accuracy are reported with mean and standard deviation obtained over a 3-fold cross validation. Loihi, Embedded GPU, and ODIN+MorphIC implementation results are from [21]. The DNN architectures adopted are as follows: $^{\diamond}$8c3-2p-16c3-2p-32c3-512-5 CNN. †16-128-128-5 MLP. ‡16-230-5 MLP. $^{\mp}4 \times$ 400-210-5 MLP. $^{\cup}$EMG and APS/DVS networks are fused using a 5-neuron dense layer.

Platform	Modality	Accuracy (%)	Energy (uJ)	Inference time (ms)	EDP (uJ * s)
Loihi (Spiking)	EMG (MLP†)	55.7 ± 2.7	173.2 ± 21.2	5.89 ± 0.18	1.0 ± 0.1
	DVS (CNN$^{\diamond}$)	92.1 ± 1.2	815.3 ± 115.9	6.64 ± 0.14	5.4 ± 0.8
	EMG+DVS (CNN$^{\cup}$)	96.0 ± 0.4	1104.5 ± 58.8	7.75 ± 0.07	8.6 ± 0.5
ODIN+ MorphIC (Spiking)	EMG (MLP‡)	53.6 ± 1.4	7.42 ± 0.11	23.5 ± 0.35	0.17 ± 0.01
	DVS (MLP$^{\mp}$)	85.1 ± 4.1	57.2 ± 6.8	17.3 ± 2.0	1.00 ± 0.24
	EMG+DVS (MLP$^{\cup}$)	89.4 ± 3.0	37.4 ± 4.2	19.5 ± 0.3	0.42 ± 0.08
Embedded GPU	EMG (MLP†)	68.1 ± 2.8	$(25.5 \pm 8.4) \cdot 10^3$	3.8 ± 0.1	97.3 ± 4.4
	EMG (MLP‡)	67.2 ± 3.6	$(23.9 \pm 5.6) \cdot 10^3$	2.8 ± 0.08	67.2 ± 2.9
	APS (CNN$^{\diamond}$)	92.4 ± 1.6	$(31.7 \pm 7.4) \cdot 10^3$	5.9 ± 0.1	186.9 ± 3.9
	APS (MLP$^{\mp}$)	84.2 ± 4.3	$(30.2 \pm 7.5) \cdot 10^3$	6.9 ± 0.1	211.3 ± 6.1
	EMG+APS (CNN$^{\cup}$)	95.4 ± 1.7	$(32.1 \pm 7.9) \cdot 10^3$	6.9 ± 0.05	221.1 ± 4.1
	EMG+APS (MLP$^{\cup}$)	88.1 ± 4.1	$(32.0 \pm 8.9) \cdot 10^3$	7.9 ± 0.05	253 ± 3.9
FPGA	EMG (MLP†)	67.2 ± 2.3	$(17.6 \pm 1.1)\, 10^3$	4.2 ± 0.1	74.1 ± 1.2
	EMG (MLP‡)	63.8 ± 1.4	$(13.9 \pm 1.8) \cdot 10^3$	3.5 ± 0.1	48.9 ± 1.9
	APS (CNN$^{\diamond}$)	96.7 ± 3.0	$(24.0 \pm 1.2)\, 10^3$	5.4 ± 0.2	130.8 ± 1.4
	APS (MLP$^{\mp}$)	82.9 ± 8.4	$(23.1 \pm 2.6) \cdot 10^3$	5.7 ± 0.2	131.4 ± 2.8
	EMG+APS (CNN$^{\cup}$)	94.8 ± 2.0	$(31.2 \pm 3.0)\, 10^3$	6.3 ± 0.1	196.3 ± 3.1
	EMG+APS (MLP$^{\cup}$)	83.4 ± 2.8	$(31.1 \pm 1.4) \cdot 10^3$	7.3 ± 0.2	228.2 ± 1.6
Memristive	EMG (MLP†)	64.6 ± 2.2	0.038	$6.0 \cdot 10^{-4}$	$2.38 \cdot 10^{-8}$
	EMG (MLP‡)	63.8 ± 1.4	0.026	$4.0 \cdot 10^{-4}$	$1.04 \cdot 10^{-8}$
	APS (CNN$^{\diamond}$)	96.2 ± 3.3	4.83	$1.0 \cdot 10^{-3}$	$4.83 \cdot 10^{-6}$
	APS (MLP$^{\mp}$)	82.4 ± 8.5	0.18	$4.0 \cdot 10^{-4}$	$7.2 \cdot 10^{-8}$
	EMG+APS (CNN$^{\cup}$)	94.8 ± 2.0	4.90	$1.2 \cdot 10^{-3}$	$5.88 \cdot 10^{-6}$
	EMG+APS MLP$^{\cup}$)	83.4 ± 2.8	0.33	$6.0 \cdot 10^{-4}$	$1.98 \cdot 10^{-7}$

Locality There is a hardware constraint that the learning information for updating the weights of any on-chip network should be locally available to the synapse, otherwise most of the silicon area would be consumed by the wires, required to route the update information to it. As Hebbian learning satisfies this requirement, most of the available on-chip learning algorithms focus on its implementation in forms of unsupervised/semi-supervised learning [175, 177]. However, local Hebbian-based algorithms are limited in learning static patterns or using very shallow networks [178]. There are also some efforts in the direction of on-chip gradient-descent based methods which implement on-chip error-based learning algorithms where the least mean square of a neural network cost function is minimized. For example, spike-based delta rule is the most common weight update used for single-layer networks which is the base of the back-propagation algorithm used in the vast majority of current multi-layer neural networks. Single layer mixed-signal neuromorphic circuit implementation of the delta rule have already been designed [179] and employed for EMG classification [158]. Expanding this to multi-layer networks involves non-local weight updates which limits its on-chip implementation. Making the backpropagation algorithm local is a topic of on-going research [180–182].

Weight storage The holy grail weight storage for online on-chip learning is a memory with non-volatile properties whose state can change linearly in an analog fashion. Non-volatile memristive devices provide a great potential for this. Therefore, there is a large body of literature in combining the maturity of CMOS technology with the potential of the emerging memories to take the best out of the two worlds.

The integration of CMOS technology with that of the emerging devices has been demonstrated for non-volatile filamentary switches [183] already at a commercial level [184]. There have also been some efforts in combining CMOS and memristor technologies to design supervised local error-based learning circuits using only one network layer by exploiting the properties of memristive devices [179, 185, 186].

Apart from the above-mentioned benefits in utilizing memristive devices for online learning in SNN-based neuromorphic chips, as discussed in Section 2.3.3, memristive devices have also shown interesting features to improve the power consumption and delay of conventional DNNs. However, as shown in Table 2.3, memristor-based DNNs are very sparse in the biomedical domain, and existing works are largely based only on simulation.

2.4.3 Why Is the Use of MDNNs Very Limited in the Biomedical Domain?

Currently there are very few hardware implementations of biomedical MDNNs that make use of general programmable memristive-CMOS, and only one programmed to construct an MLP for cancer diagnosis. We could also find two other memristive designs in literature for biomedical applications (shown in Table 2.3), but they are only simulations considering memristive crossbars. This sparsity is despite the significant advantages that memristors provide in MAC paralleliza-

tion and in-memory computing paradigm, while being compatible with CMOS technology [187]. These features make memristors ideal candidates for DL accelerators in general, and for portable and edge-based healthcare applications in particular, because they have stringent device size and power consumption requirements. To be able to use memristive devices in biomedical domain, though, several of their shortcomings such as limited endurance, mismatch, and analog noise accumulation must be overcome first. This demands further research in the materials, as well as the circuit and system design side of this emerging technology, while at the same time developing facilitator open-source software [15] to support MDNNs. Furthermore, investigating the same techniques utilized in developing CMOS-based DL accelerators such as limited precision data representation [8, 155] and approximate computing schemes can lead to advances in developing MDNNs and facilitate their use in biomedical domains.

2.4.4 Why and When to Use FPGA for Biomedical DNNs?

Table 2.3 shows that FPGA is a popular hardware technology for implementing simple DL networks such as MLPs [136–138, 188] and in a few cases, more complex LSTMs and CNNs [139, 164–166]. The table also shows that FPGAs are mainly used for signal processing tasks and have not been widely used to run complex DL architectures such as CNNs. This is mainly because they have limited on-chip memory and low bandwidth compared to GPUs. However, they demonstrate notable benefits in terms of significantly shorter development time compared to ASICs, and much lower power consumption than typical GPUs. Besides, significant power and latency improvement can be gained by customizing the implementation of various components of a DNN on an FPGA, compared to running it on a general-purpose CPU or GPU [137, 139]. For instance, in [139], EEG signals are processed on FPGAs using two customized hardware blocks for (i) parallelizing MAC operations and (ii) efficient recurrent state updates, both of which are key elements of LSTMs. This has resulted in almost an order of magnitude power efficiency compared to GPUs. This efficiency is critical in many edge-computing applications including DNN-based point-of-care biomedical devices [66] and healthcare IoT [65, 105].

Another benefit of FPGAs is that a customized efficient FPGA design can be directly synthesized into an ASIC using a nanometer-node CMOS technology to achieve even more benefits [165, 166]. For instance, [139] has shown near 100 times energy efficiency improvement as an ASIC in a 15-nm CMOS technology, compared to its FPGA counterpart.

Although low-power consumption and affordable cost are two key factors for almost any edge-computing or near-sensor device, these are even more important for biomedical devices such as wearables, health-monitoring systems, and PoC devices. Therefore, FPGAs present an appealing solution, where their limitations can be addressed for a customized DNN using specific design methods such as approximate computing [9] and limited-precision data [11, 13], depending on the cost, required power consumption, and the acceptable accuracy of the biomedical device.

Another programmable low-power device that can be used in biomedical applications are Field

Programmable Analog Arrays (FPAAs). These are constructed using programmable Computational Analog Blocks (CABs) and interconnects. Unlike FPGAs, FPAAs tend to be more application driven than general purpose as they may be current mode or voltage mode devices [189]. FPAAs have been shown to perform computation with 1000 times more power efficiency while reducing the required area by 100 times when compared to FPGAs [189]. Therefore, they are a promising candidate for accelerating biomedical signal processing if machine learning algorithms such as ANNs can be implemented using them.

In 2003, [190] explored ANNs with differential feedback, and in 2006 [191] implemented an ANN using multi-chip FPAAs. More recently, [192] have demonstrated that VMMs can be efficiently computed using FPAAs, which can be used to compute linear and unrolled convolution layers within DNNs. However, while FPAAs have been used in several biomedical applications ranging from knee-joint rehabilitation [193] to the amplification of various bio-electric signals [194], the implementation of a FPAA DNN accelerator, which can be used in biomedical and general applications, is yet to be explored.

2.4.5 Benchmarking EMG Processing Across Multiple DNN and SNN Hardware Platforms

In Table 2.5, we compare our FPGA and memristive implementations to other DNN accelerators and neuromorphic processors from [21]. In [21], the authors presented a sensor fusion neuromorphic benchmark for hand-gesture recognition based on EMG and event-based camera. Two neuromorphic platforms, Loihi [174] and ODIN+MorphIC [175, 176], were deployed and the results were compared to traditional machine learning baselines implemented on an embedded GPU, the NVIDIA Jetson Nano. Loihi and ODIN+MorphIC are digital neuromorphic platforms. Loihi is a 128-core neuromorphic chip fabricated on 14 nm FinFET process, designed by Intel Labs. It implements adaptive self-modifying event-driven fine-grained parallel computations used to implement learning and inference with high efficiency. ODIN (Online-learning Digital spiking Neuromorphic) is designed using 28 nm FDSOI CMOS technology and consists of a single neurosynaptic core with 256 neurons and 256^2 synapses that embed a 3-bit weight and a mapping table bit that allows enabling or disabling Spike-Timing-Dependent Plasticity (STDP). MorphIC is a quad-core digital neuromorphic processor with 2k Leaky Integrate and Fire (LIF) neurons and more than 2M synapses in 65 nm CMOS technology [176]. They can be either programmed with offline-trained weights or trained online with a stochastic version of Spike-Driven Synaptic Plasticity (SDSP).

For the spiking architectures shown in Table 2.5, the vision input and EMG data were individually processed using spiking CNN and spiking MLP respectively, and fused in the last layer. Loihi was trained using SLAYER [195], a backpropagation framework used for evaluating the gradient of any kind of SNN. It is a dt-based SNN backpropagation algorithm that keeps track of the internal membrane potential of the spiking neuron and uses it during gradient propagation. Both ODIN

and Morphic training was carried out in Keras with quantization-aware stochastic gradient descent following a standard ANN-to-SNN mapping approach.

The dataset used is described in Section 2.2.5. It is a collection of 5 hand gestures from sign language (e.g. ILY)[4]. In the comparison proposed in Table 2.5 the input and hidden layers are sequenced with the ReLU activation function, and output layers are fed through Softmax activation functions to determine class probabilities. Dropout layers are used in all networks to avoid overfitting. The DNN architectures are determined in the table caption.

The platforms used for each system in Table 2.5 are as follows: ODIN+MorphIC [175, 176] and Loihi [174] neuromorphic platforms were used for spiking implementations; NVIDIA Jetson Nano was used for all embedded GPU implementations; OpenVINO Toolkit FPGA was used for all FPGA implementations, and MemTorch [15] was used for converting the MLP and CNN networks to their corresponding MDNNs to determine the test set accuracies of all memristive implementations.

From Table 2.5, it can be observed that, when transitioning from generalized architectures to application specific processors, more optimized processing of a subset of given tasks can be achieved. Moving up the specificity hierarchy from GPU to FPGA to memristive networks shows orders of magnitude of improvement in both MLP and CNN processing, but naturally at the expense of a generalizable range of tasks. While GPUs are relatively efficient at training networks (compared to CPUs), the impressive metrics presented by memristor (RRAM in this simulations) is coupled with limited endurance. This is not an issue for read-only tasks, as is the case with inference, but training is thwarted by the thousands of epochs of weight updates which limits broad use of RRAMs in training. Rather, more exploration in alternative resistive-based technologies such as Magnetoresistive Random-Access Memory (MRAM) could prove beneficial for tasks that demand high endurance.

After determining the test set accuracy of each MDNN using MemTorch [15], we determined the energy required to perform inference on a single input, the inference time, and the Energy Delay Product (EDP) by adopting the metrics in [196], for a tiled memristor architecture. All assumptions made in our calculations are listed below. Parameters are adopted from those given in a 1T1R 65nm technology, where the maximum current during inference is 3μA per cell with a read voltage of 0.3V. Each cell is capable of storing 8 bits with a resistance ratio of 100, and mapping signed weights is achieved using a dual column representation. All convolutions are performed by unrolling the kernels and performing MVMs, and the fully connected layers have the fan-in weights for a single neuron assigned to one column. Each crossbar has an aspect ratio of 256×64 to enable more analog operations per ADC when compared to a 128×128 array. Where there is insufficient space to map weights to a single array, they are distributed across multiple arrays, with their results to be added digitally. Throughput can be improved at the expense of additional arrays for convolutional layers, by duplicating kernels such that multiple inputs can

[4] `https://zenodo.org/record/3663616#.X2m5GC2cbx4`. Further implementation details can be found in [21].

be processed in parallel. The number of tiles used for each network is assumed to be the exact number required to balance the processing time of each layer. The power consumption of each current-mode 8-bit ADC is estimated to be 2×10^{-4} W with an operating frequency of 40 MHz (5 MHz for bit-serial operation) [196]. The ADC latency is presumed to dominate digital addition of partial products from various tiles. The dynamic range of each ADC has been adapted to the maximum possible range for each column, and each ADC occupies a pair of columns.

The above presumptions lead to pre-silicon results that are extremely promising for memristor arrays, as shown in Table 2.5. But it should be clear that these calculations were performed for *network-specific* architectures, rather than a more general application-specific use-case. That is, we assume the chip has been designed for a given neural network model. The other comparison benchmarks are far more generalizable, in that they are suited to not only handle most network topologies, but are also well-suited for training. The substantial improvement of inference time over other methods is a result of duplicate weights being mapped to enable higher parallelism, which is tolerable for small architectures, but lends to prohibitively large ADC power consumption for computer vision tasks which rely on deep networks and millions of parameters, such as VGG-16. In addition, the area of each ADC is estimated to be $3\times10^{-3}\text{mm}^2$, which is orders of magnitude larger than the area of each RRAM cell ($1.69\times10^{-7}\text{mm}^2$). This disparity implies that pitch-matching is not viable. Instead, to achieve parallelism, weights must be duplicated across tiles which demands redundancy. This improvement in parallelism thus comes at the cost of additional area and power consumption. The use of memristors as synapses in spike-based implementations may be more appropriate, so as to reduce the ADC overhead by replacing multi-bit ADCs with current sense amplifiers instead, and reducing the reliance on analog current summation along resistive and capacitive bit-lines.

Spike-based hardware show approximately two orders of magnitude improvement in the EDP from Table 2.5 when compared to their GPU and FPGA counterparts, which highlights the prospective use of such architectures in always-on monitoring. This is necessary for enhancing the prospect of ambient-assisted living, which would allow medical resources to be freed up for tasks that are not suited for automation. In general, one would expect that data should be processed in its naturalized form. For example, 2D CNNs do not discard the spatial relations between pixels in an image. Graph networks are optimized for connectionist data, such as the structure of proteins. By extension, the discrete events generated by electrical impulses such as in EMGs, EEGs and ECGs may also be optimized for SNNs. Of course, this discounts any subthreshold firing patterns of measured neuron populations. But one possible explanation for the suitability of spiking hardware for biological processes stems from the natural timing of neuronal action potentials. Individual neurons will typically not fire in excess of 100 Hz, and the average heart rate (and correspondingly, ECG spiking rate) will not exceed 3 Hz. There is a clear mismatch between the clock rate of non-spiking neural network hardware, which tend to at least be in the MHz range, and spike-driven processes. This introduces a significant amount of wastage in processing data when there

is no new information to process (e.g., in between heartbeats, action potentials, or neural activity).

Nonetheless, it is clear that accuracy is compromised when relying on EMG signals alone, based on the approximately 10% decrease of classification accuracy on the Loihi chip and ODIN+MorphIC, as against their GPU/FPGA counterparts. This could be a result of spike-based training algorithms lagging behind in maturity compared to conventional neural network methods, or it could be an indication that critical information is being discarded when neglecting the subthreshold signals generated by populations of neurons. But when EMG and DVS data are combined, this multi-sensory data fusion of spiking signals positively reinforce upon each other with an approximately 4% accuracy improvement, whereas combining non-spiking, mismatched data representations leads to marginal improvements, and even a destructive effect (e.g., non-spiking CNN implementation on FPGA and memristive arrays). This may be a result of EMG and APS data taking on completely different structures. This is a possible indication that feature extraction from merging the same structural form of data (i.e., as spikes) proves to be more beneficial than combining a pair of networks with two completely different modes of data (i.e., EMG signals with pixel-driven images). This allows us to draw an important hypothesis: neural networks can benefit from a consistent representation of data generated by various sensory mechanisms. This is supported by biology, where all biological interpretations are typically represented by graded or spiking action potentials.

2.4.6 Deep Network Accelerators and Patient-specific Model Tuning

Given the inherent variability between patients, it is difficult to train and deploy a single model to a large group of individuals each with unique signature(s). Consequently, significant efforts are being made to facilitate patient-specific model tuning processes [113, 197, 198]. Patient-Specific Modeling (PSM) is the development of computational models of human or animal pathophysiology that are individualized to patient-specific data [197].

In the DL domain, existing ANN and neuromorphic models can be retrofitted to specific patients using transfer learning and tuning algorithms. In this approach, the network is first trained on a large dataset including data from various patients to acquire the domain-specific knowledge of the targeted task. Parts of the large network are then retrained, i.e. tuned, using patient-specific data, to produce better performance for individual patients. This way, the domain-specific features of the large network are transferred to the smaller network that is retrained to learn patient-specific features [113]. Depending on the availability of patient-specific data, PSM can be performed online (on-chip) or offline (off-chip).

Online patient-specific model tuning

Considering concerns surrounding the sensitive nature of individual patient data, and the ability of some recent edge-AI CMOS chips such as LNPU [96] to perform online training, patient-specific model tuning can be performed online on the hardware deep learning accelerator. To achieve this, a sufficient amount of patient data that is fed to the accelerator over time can be gathered to

individualize the initial generic model. An accelerator that can adapt its working to the specific needs of a patient would be highly beneficial but it may require buffering of data [199], which needs higher on-chip memory and may introduce power overheads.

Offline patient-specific model tuning

A convenient approach to tune general models, with domain-specific knowledge, to patient-specific data is offline off-chip transfer learning. However, unlike online tuning, the offline approach requires prior patient data measurements, which may not be readily available. Besides, the offline approach may require undesired remote storage and processing of private patient data to retrain and tune generic models.

2.5 Conclusion

The use of DL in biomedical signal processing and healthcare promises significant utility for medical practitioners and their patients. DNNs can be used to improve the quality of life for chronically ill patients by enabling ambient monitoring for abnormalities, and correspondingly can reduce the burden on medical resources. Proper use can lead to reduced workloads for medical practitioners who may divert their attention to time-critical tasks that require a standard beyond what neural networks can achieve at this point in time.

We have stepped through the use of various DL accelerators on a disparate range of medical tasks, and shown how SNNs may complement DNNs where hardware efficiency is the primary bottleneck for widespread integration. We have provided a balanced view to how memristors may lead to optimal hardware processing of both DNNs and SNNs, and have highlighted the challenges that must be overcome before they can be adopted at a large-scale. While the focus of this tutorial and review is on hardware implementation of various DL algorithms, the reader should be mindful that progress in hardware is a necessary, but insufficient, condition for successful integration of medical-AI.

Adopting medical-AI tools is clearly a challenge that demands the collaborative attention of healthcare providers, hardware and software engineers, data scientists, policy-makers, cognitive neuroscientists, device engineers and materials scientists, amongst other specializations. A unified approach to developing better hardware can have pervasive impacts upon the healthcare industry, and realize significant payoff by improving the accessibility and outcomes of healthcare.

Chapter 3

Modeling and Simulating In-Memory Memristive Deep Learning Systems: An Overview of Current Efforts

Memristive In-Memory Computing (IMC) systems for Deep Learning (DL), which are also known as Memristive Deep Learning Systems (MDLS), can perform repetitive Multiply and Accumulate (MAC) operations and store the results in the same physical location using emerging memory devices. This can be used to augment the performance of traditional DL architectures, massively reducing their power consumption and latency. However, memristive devices, such as Resistive Random-Access Memory (RRAM) and Phase-Change Memory (PCM), are difficult and cost-prohibitive to fabricate in small quantities, and are prone to various device non-idealities that must be accounted for. Consequently, the popularity of simulation frameworks, used to simulate MDLS prior to circuit-level realization, is burgeoning. In this Chapter, the second part of the literature review component of this thesis is presented. A survey of existing simulation frameworks and related tools used to model large-scale MDLS is provided. In addition, performance comparisons of modernized open-source simulation frameworks are made, and insights into future modeling and simulation strategies and approaches are presented.

3.1 Introduction

Traditionally, ML and DL systems are trained and deployed using hardware platforms adopting the von Neumann computing architecture. While in recent years, GPUs have been used to massively parallelize and accelerate the performance of these workloads [37], they are still prone to performance bottlenecks caused by the amount of data being moved back and forth between physically separated memory and processing units. IMC is a novel non-von Neumann approach, where certain computational tasks are performed in the memory itself [41], which has the potential to alleviate this bottleneck.

IMC systems can be realised by arranging memory devices in crossbar architectures, where they can be used to perform various logical and arithmetic operations [200]. These memory devices

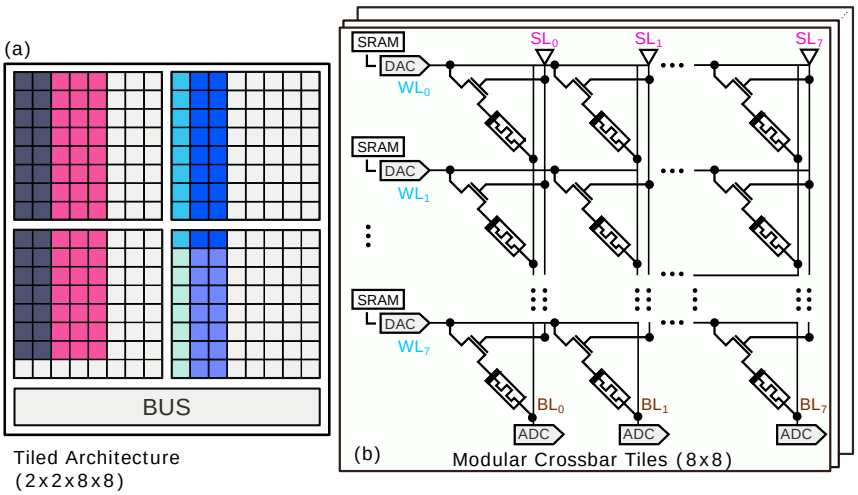


Figure 3.1: (a) A modular memristive crossbar tiled architecture containing parameters from two linear and unfolded convolutional layers; both key components of traditional CNNs. Unique colours denote mapped parameters from different layers. (b) In a modular crossbar tile that is used to perform the VMM operation in-memory, SLs can be used to isolate columns of devices (BLs), in which inputs are applied to as WL voltages. BLs currents are read out using ADCs, that can be linearly related to vector-matrix product elements. This figure is adapted from [6].

can be fabricated using legacy charge-based memory technologies, such as Static Random-Access Memory (SRAM), or emerging memristive device technologies, such as RRAM, which are introduced and discussed in Section 3.2. Memristive devices, in particular, have shown great promise to facilitate the acceleration and improve the power efficiency of ML and DL systems, as they can be passive, re-programmable, and non-volatile [1, 145, 200–203].

As depicted in Fig. 3.1, crossbar architectures constructed using memristive RRAM devices can be used to efficiently implement various in-memory computing operations, including MAC and VMMs operations. Previous works in the literature have exploited physical properties of memristive devices to realize a variety of commonly used operations and components of neuromorphic architectures [4, 42, 144, 204, 205]. Traditionally, IMC systems have been used to implement brain-inspired asynchronous neuromorphic architectures [206], realizing artificial synapses using memristive devices. However, they are also capable of accelerating VMMs, the most dominant operations in DNNs, in $\mathcal{O}(1)$, which makes them more appealing for deep learning systems [8, 207].

Currently, several memristive device technologies, including RRAM and PCM, which are depicted in Fig. 3.2, are being actively researched [200]. However, despite continuous ongoing efforts, they are prone to various device non-idealities, which limit their accuracy and reliability to use in practical engineering settings [208]. Consequently, many large-scale simulations encompassing device and circuit non-idealities have been conducted using synaptic memristive connections for brain-inspired asynchronous neuromorphic systems [209–211] and DL systems [204]. While these simulations were traditionally performed using general purpose Simulation Program

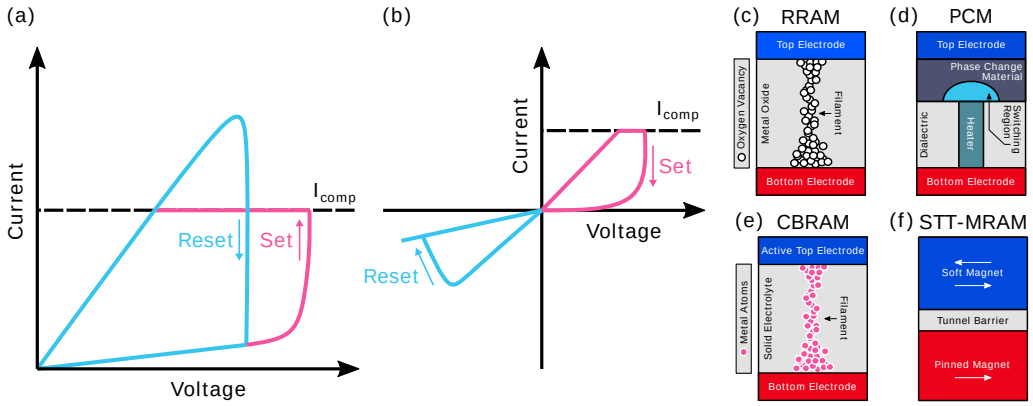


Figure 3.2: Typical (a) unipolar and (b) bipolar switching modes of memristive devices and schematics of popular device technologies: (c) RRAM, (d) PCM, (e) CBRAM, and (f) STT-MRAM.

with Integrated Circuit Emphasis (SPICE)-based simulators, as the complexity of the underlying systems and neuromorphic architectures being simulated has increased, customized simulation frameworks have been developed. These frameworks are used to rapidly prototype novel network architectures as a preliminary step prior to circuit-level validation and layout using mature CAD tools; for eventual circuit-level realization and large-scale fabrication.

In contrast to conventional SPICE-based simulation, modern CAD simulation frameworks adopt modern software engineering methodologies. Moreover, they are able to accurately model non-ideal device characteristics, peripheral circuitry, and modular crossbar tiles while being interfaceable using high-level language APIs. We confine the scope of this Chapter to MDLS, i.e., memristive IMC systems for DL system deployment, and provide a survey of existing simulation frameworks and related tools used to model large-scale MDLS.

The rest of this Chapter is structured as follows. In Section 3.2, preliminaries related to modeling and simulating in-memory MDLS are presented. In Section 3.3, existing CAD tools for in-memory MDLS are over-viewed. In Section 3.4, comparisons of modern simulation frameworks for in-memory MDLS are made, and two MDLS architectures are simulated. In Section 3.5, we provide an outlook for MDLS simulation frameworks. Finally, in Section 3.6, this Chapter is concluded.

3.2 Preliminaries

Memristors, commonly referred to as the fourth fundamental circuit element, are two-terminal passive circuit elements characterized by a relationship between the charge, $q(t) \equiv \int_{-\infty}^{t} i(\tau)d\tau$ and the flux-linkage $\varphi(t) \equiv \int_{-\infty}^{t} v(\tau)d\tau$ [212]. Memristors are capable of non-volatile storage. We depict typical unipolar and bipolar switching I-V characteristics, and schematics of popular memristive device technologies in Fig. 3.2.

Unfolded convolutional layers and linear (dense) layers within DL systems can be implemented using a series of MAC and VMM operations, which can be computed in-memory using memristive crossbar arrays, as depicted in Fig. 3.1, by encoding weights as resistance/conductance values, and inputs as WL voltages. Tiled crossbar architectures contain several modular crossbar tiles connected using a shared bus. These are also connected to additional circuitry used to realize batch-normalization, pooling, activation functions, and other computations that cannot be performed, or are not efficient, in-memory. Modular crossbar tiles consist of crossbar arrays with supporting peripheral circuitry. We refer the reader to [42] for a comprehensive description and overview of IMC accelerators for DL acceleration.

In Fig. 3.2, typical switching modes and schematics of popular memristive device technologies are depicted. Memristors differ from electrical resistors, as they have a voltage or current-dependent resistance state, which is dependent on the electric properties of the materials using which they are constructed. As depicted in Fig. 3.2(c), RRAM devices are comprised of Metal–Insulator–Metal (MIM) stacks. The resistive state of RRAM devices can be modulated by creating and disrupting Conductive Filaments (CFs), used to refer to localized concentrations of defects that allow current to flow between top and bottom electrodes.

As depicted in Fig. 3.2(d), typical PCM devices have a mushroom shape (amorphous region), where the bottom electrode confines heat and current. By crystallizing the amorphous region, different resistive states can be obtained [200]. As shown in Fig. 3.2(e), CBRAM devices are comprised of a thin solid state electrolyte layer sandwiched between oxidizable and inert electrodes. The resistive state of CBRAM devices can be modulated by driving redox reactions in the filament (solid state electrolyte layer) [213]. Finally, Fig. 3.2(f) shows the device structure of STT-MRAM, which contains two ferromagnetic layers and one tunnel barrier layer. The resistance of STT-MRAM devices can be modulated by modifying the orientation of a magnetic layer in a magnetic tunnel junction or spin valve using a spin-polarized current [214].

As memristive devices can only be programmed to positive resistance states, weights can either be represented using a dual-array scheme, a dual row scheme, where double the number of rows are required, or a current-mirror scheme, that is capable of operation using a singular device to represent each weight [196].

As can be observed in Fig. 3.1, in a 1T1R arrangement, SLs can be used to individually select memristive devices. After mapping and programming weights, to perform a MAC operation, inputs are scaled and encoded as voltages, prior to being presented to WLs. Currents from each BL are read-out using ADCs, either in parallel using one ADC per column, or sequentially, using time-multiplexing. Finally, BL currents can be correlated with desired deterministic output elements using linear regression. By time multiplexing the presentation of inputs, or duplicating modular crossbar tiles, VMMs operations can be performed in $\mathcal{O}(n)$, or $\mathcal{O}(1)$, respectively.

CAD tools can be used to convert traditional DNNs to equivalent representations using modular tiled architectures. These tools can be used to simulate the inference and training of MDLS, and

to estimate power/area/latency of end-to-end implementations when various memristive devices are integrated within CMOS processes. Models are used to simulate the behavior of peripheral circuitry and memristive devices, which can be broadly categorized as empirical or analytical (functional). Empirical models are based on, concerned with, or verified by experimental data, whereas analytical models are based on analysis or logic derived from fundamental physics of the device. In this Chapter, we do not emphasize specific memristive device and crossbar circuit models, as these have previously been surveyed in other works [215–217].

Table 3.1: Comparison of conventional simulation frameworks for MDLS simulation. †Not natively supported.

Simulation framework	**Prog. language(s)**	**GPU Pre-trained DNN conversion**	**TF/PyTorch Intg.◇**	**Inference**	**Training**	**Peripheral circuitry**	**Supported devices**	**Open-source**
NVM Spice [218]	Not specified (SPICE-like)			✓†	✓†		Non-volatile memories and legacy NAND flash.	
NVSim [219]	C++, C			✓†	✓†	✓	Non-volatile memories and legacy NAND flash.	✓
NV Main, NV Main 2.0 [220, 221]	C++, System Verilog, Python			✓†	✓†		Non-volatile memories and hybrid non-volatile plus DRAM memory systems.	✓
MN SIM [222]	Not specified			✓†	✓†	✓	Non-volatile memories.	✓

TxSim [114]	Python	✓		✓	✓	✓	✓	Non-volatile memories and legacy NAND flash.
Pipe Layer [223]	C++	✓†	✓		✓	✓	✓	Non-volatile memories.
NRIS DC [224]	Python	✓	✓		✓	✓	✓	Non-volatile memories and legacy NAND flash.
IAUR RRAM Devices [225]	Python	✓	✓		✓			RRAM.
RxNN [226]	C++	✓	✓		✓		✓	Non-volatile memories.

3.3 Overview of Existing CAD Tools

In Tables 3.1 and 3.2, we present an overview of existing *conventional* and *modernized* simulation frameworks that can be used to simulate MDLS and IMC systems utilizing non-volatile memory and legacy NAND flash devices for comparison. We categorize modernized simulation frameworks as those that support pre-trained DNN conversion and TF and/or PyTorch integration. General SPICE [227] simulation tools, such as PSPICE and LTSPICE, are not compared. Although they are the most commonly used tools for analogue circuit simulation [228], they are difficult to parallelize and prohibitively slow; even when simulating large crossbar arrays using significant approximation methodologies [229, 230]. Consequently, specialized and/or parallelizable CAD tools with direct integration with modern ML frameworks, such as PyTorch [231] and Tensorflow [232], are more commonly used to simulate MDLS.

Tables 3.1 and 3.2 demonstrate that while most mature conventional SPICE-based simulation frameworks, such as NVMSpice, NVSim, and NVMain, are CPU bound, and do not natively support pre-trained DNN conversion, inference, and training modeling, they do support a large variety of device types. In addition, they are primarily focused on the high-precision and high-speed simulation of non-volatile memories and legacy NAND flash devices. In contrast, modernized recently developed frameworks, such as DNN+NeuroSIM, MemTorch, and the IBM Analog Hardware

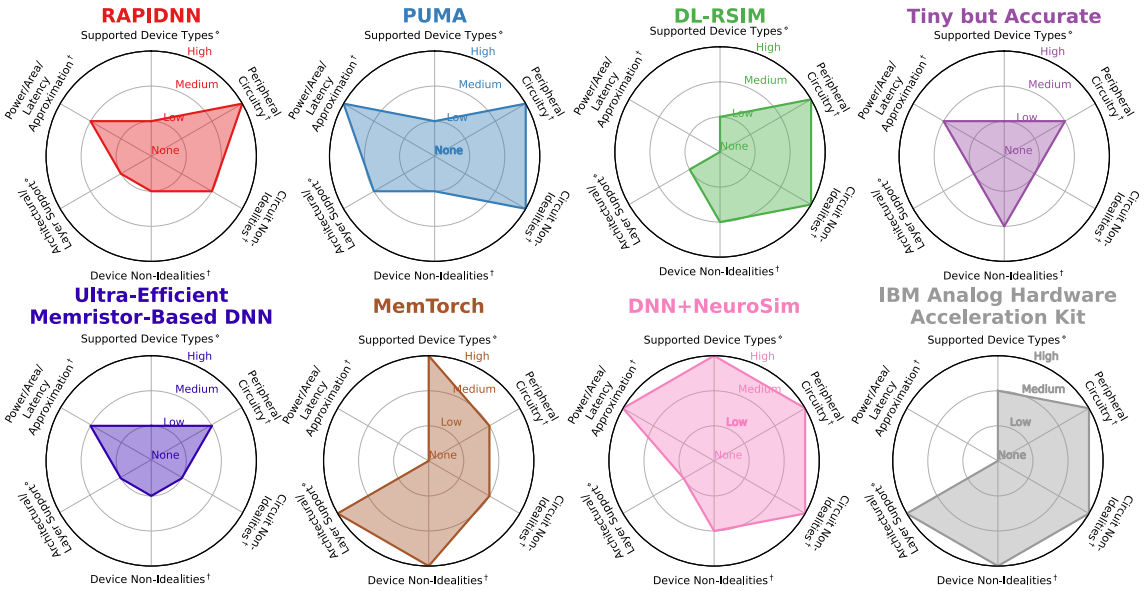


Figure 3.3: Comparison of modern simulation frameworks that support pre-trained DNN conversion and TF/PyTorch integration. †Support and Accuracy. ◇Degree of Coverage.

Acceleration Kit, abstract performance-critical operations on GPUs, integrate directly with popular ML frameworks, and have well documented APIs. Moreover, they adopt modern software engineering methodologies, and are able to accurately model non-ideal device and circuit characteristics, peripheral circuitry, and crossbar tiles. They are also directly interfaceable with other tools using accessible, general-purpose high-level programming languages; a paradigm shift from conventional SPICE-based simulation.

3.4 Comparison of Modern Simulation Frameworks

While modernized simulation frameworks superficially appear similar, upon closer inspection, they are complimentary in nature. To make this clearer, in Fig. 3.3, we compare modern simulation frameworks, i.e., those that support pre-trained DNN conversion and TF/PyTorch integration in more detail, using radar charts. As it is shown, there is not a large overlap amongst the simulation frameworks which have been compared: RAPIDNN, PUMA, DL-RSIM, Tiny but Accurate, Ultra-Efficient Memristor-Based DNN, MemTorch, DNN+NeuroSIM, and the IBM Analog Hardware Acceleration Kit.

Although many of these simulation frameworks are still under active development, and are not fully mature, they clearly adopt different design and usability approaches. For instance, both Tiny but Accurate and Ultra-Efficient Memristor-Based DNN are built upon NVSim, whereas all other simulation frameworks are either written from scratch in lower level languages, or extend upon existing high-level GPU-accelerated computing libraries to abstract performance critical operations. Moreover, while RAPIDNN, PUMA, Tiny but Accurate, Ultra-Efficient Memristor-Based

DNN and DNN+NeuroSIM can be used to generate power/area/latency reports, MemTorch and the IBM Analog Hardware Acceleration Kit support a large number of different layer types, and

Table 3.2: Comparison of modernized simulation frameworks for MDLS simulation. ‡Models are shared using Google Drive without APIs. ◇TF/PyTorch integration.

Simulation framework	**Prog. language(s)**	**GPU**	**Pre-trained DNN conversion**	**TF/PyTorch Intg.◇**	**Inference**	**Training**	**Peripheral circuitry**	**Supported devices**	**Open-source**
RAPIDNN [233]	C++, SPICE		✓	✓	✓		✓	Single-level memristive devices.	
PUMA [150]	C++		✓	✓	✓		✓	Non-volatile memories and legacy NAND flash.	
DL-RSIM [234]	Python	✓	✓	✓	✓		✓	Non-volatile memories.	
Tiny but Accurate [235]	MATLAB		✓	✓	✓		✓	Non-volatile memories.	✓‡
Ultra-Efficient Memristor-Based DNN [236]	C++, MATLAB		✓	✓	✓		✓	Non-volatile memories	✓‡
MemTorch [15, 16]	Python, C++, CUDA	✓	✓	✓	✓		✓	Non-volatile memories and legacy NAND flash.	✓
NeuroSim and derivatives [139, 139, 237–239]	C++, Python	✓	✓	✓	✓	✓	✓	Non-volatile memories and legacy NAND flash.	✓
IBM Analog Hardware Acceleration Kit [240]	C++, Python, CUDA	✓	✓	✓	✓	✓	✓	Non-volatile memories.	✓

can be used to accurately model device non-idealities in a robust and modular manner. By adopting different design and usability approaches, all simulation frameworks can be beneficial and complement each other to be used by a variety of users with different requirements.

To determine the usability and performance of each modernized simulation framework, when possible, we used each framework to simulate the training routine of the VGG-8 [241] network architecture, and the inference routine of the GoogLeNet [242] network architecture. Both training and inference routines were evaluated using the CIFAR-10 dataset. Two separate network architectures were used for evaluation, as larger and more complex networks could not be reliably trained using existing simulation frameworks with CUDA support when utilizing a single GPU, even with 32GB of Video Random-Access Memory (VRAM). Moreover, not all simulation frameworks supported convolutional layers with non-zero groups (connections between inputs and outputs), meaning that many ResNet-based architectures could not be implemented.

When possible, weights from linear and convolutional layers were mapped onto modular 1T1R crossbar tiles of size (16×16) using a differential weight mapping scheme, and device-to-device variability was modeled by sampling R_{ON} and R_{OFF} from normal distributions with mean values of $10k\Omega$ and $100k\Omega$, and standard deviation values of $1,000$ and $10,000$, respectively, i.e., $\bar{R_{\text{ON}}} = 10k\Omega$, and $\bar{R_{\text{OFF}}} = 100k\Omega$. Devices were assumed to have a finite number (6) of conductance states, and ADCs were assumed to operate at a 6-bit resolution. For inference routine simulations, 10 runs were conducted, and mean and standard deviation values were reported across all runs. For training routine simulations, mean and standard deviation values were reported across all training epochs. All codes used to perform comparisons are made publicly-accessible[1], and can be modified to perform comparisons using different hardware technologies, network architectures, and hyper-parameters.

The RAPIDNN, PUMA, and DL-RSIM simulation frameworks are not open-source, so they could not be evaluated and directly compared in more detail. Similarly, while full precision and quantized trained models are available for the DL-RSIM and Tiny but Accurate frameworks, codes used to simulate inference routines are not. Consequently, in Fig. 3.4, training routines of DNN+NeuroSim and the IBM Analog Hardware Acceleration Kit are compared, and in Fig. 3.5, inference routines of MemTorch, DNN+NeuroSim, and the IBM Analog Hardware Acceleration Kit, are compared.

3.4.1 Simulation Configurations

All simulations were conducted using a High Performance Computing (HPC) cluster with the following run-time hardware configuration set using the Simple Linux Utility for Resource Management (SLURM) workload manger: 1 node and 8 CPU cores (Intel Xeon 6132 series CPU sockets), 100GB DDR4 3200MHz Random-Access Memory (RAM), and one PCI-E 32GB Volta

[1] https://github.com/coreylammie/Modeling-and-Simulating-In-Memory-Memristive-Deep-Learning-Systems

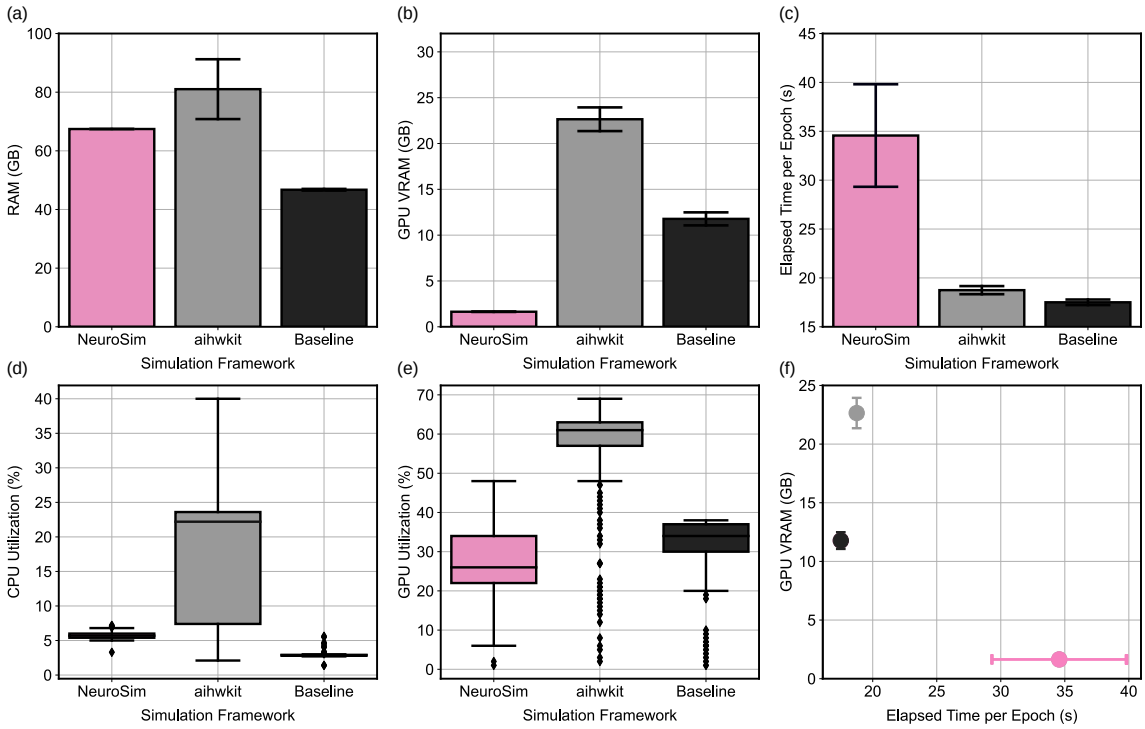


Figure 3.4: Comparison of training routines of DNN+NeuroSim and the IBM Analog Hardware Acceleration Kit, for the VGG-8 network architecture, using the CIFAR-10 dataset.

V100 GPU. `torch.cuda.Event` and `timer.time()` were used to determine the execution time of various simulation components. We reiterate that all scripts provided in [1] can be used to benchmark all simulation frameworks using different software, hardware, and environmental configurations.

MemTorch

Using `MemTorch`[2], modular crossbars tiles of (16×16) generic RRAM devices arranged using a differential weight mapping scheme were simulated. For each device, device-to-device variability was modeled by sampling R_{ON} and R_{OFF} from normal distributions with mean values of $10k\Omega$ and $100k\Omega$, and standard deviation values of $1,000$ and $10,000$, respectively. Devices were assumed to have a finite number (6) of evenly-spaced conductance states. The operating resolution of ADCs was set to 6-bits.

NeuroSim

Using `DNN_NeuroSim_V2.1`[3], modular crossbars tiles of (16×16) generic RRAM devices arranged using a differential weight mapping scheme were simulated. Each device was set to have

[2] `https://github.com/coreylammie/MemTorch`
[3] `https://github.com/neurosim/DNN_NeuroSim_V2.1`

an $\bar{R_{\text{ON}}}/\bar{R_{\text{OFF}}}$ ratio of 10, with a device-to-device variation of 10%. This was done, as NeuroSim did not have the functionality to directly set $\bar{R_{\text{ON}}}$ and $\bar{R_{\text{OFF}}}$ values. The weight precision of each device and operating resolution of ADCs were set to 6-bits.

IBM Analog Hardware Acceleration Kit

Using the IBM Analog Hardware Acceleration Kit (denoted using `aihwkit`[4] in short-form), modular crossbar tiles could not be simulated, as they were not supported. Instead, singular tiles arranged using a differential weight mapping scheme were used to map weights of linear and convolutional layers. In lieu of support for generic RRAM device modeling with arbitrary $\bar{R_{\text{ON}}}$ and $\bar{R_{\text{OFF}}}$ values and $\bar{R_{\text{ON}}}/\bar{R_{\text{OFF}}}$ ratios, devices characterised in [243] were simulated with a device-to-device variation of 10%. The weight precision of each device could not be directly set. The operating resolution of ADCs was set to 6-bits.

Baseline

In addition to simulating training and inference routines using MemTorch, DNN_NeuroSim_V2.1, and the IBM Analog Hardware Acceleration Kit, baseline training and inference routines were simulated using the native PyTorch ML library for comparison. For all baseline implementations, the exact same hyper-parameters were used. `torch.cuda.amp` was used to quantize all network parameters to 16-bits to improve performance.

3.4.2 Training Routine Comparison

In Fig. 3.4, the performance of training routines for the VGG-8 network architecture using the CIFAR-10 dataset are compared. For NeuroSim and the IBM Analog Hardware Acceleration Kit, default non-linear weight update parameters were used. All networks were trained for 256 epochs with a batch size of 128 using SGD with momentum and cross-entropy loss. An initial learning rate of 0.1 was used with fixed momentum value of 0.9. Optimizers that support adaptive learning rates were not used, as these were not supported by DNN_NeuroSim_V2.1. Instead, during training, the learning rate was decayed by one order of magnitude at epochs 100, 200, and 250 (these schedules were determined empirically), to prevent stagnation.

The functionality of each simulation framework has previously been investigated and validated [16, 239, 240]. Consequently, training and test set losses and accuracies were not reported or compared, as they have no bearing on the performance of each simulation framework. As can be seen in Fig. 3.4, the IBM Analog Hardware Acceleration Kit consumed the most RAM and GPU VRAM. While DNN_NeuroSim_V2.1 consumed more RAM than the baseline implementation, interestingly, it consumed notability less VRAM. This can be largely attributed to the large

[4] `https://github.com/IBM/aihwkit`

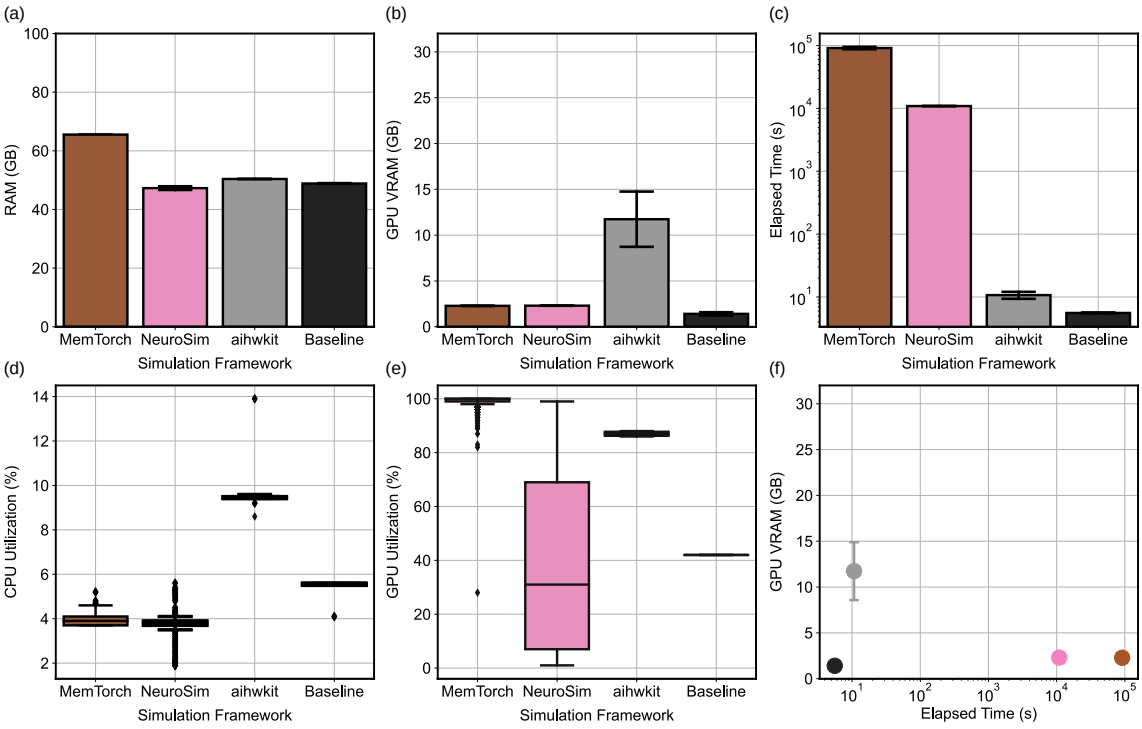


Figure 3.5: Comparison of inference routines of MemTorch, DNN+NeuroSim, and the IBM Analog Hardware Acceleration Kit, for the VGG-8 network architecture, using the CIFAR-10 dataset.

number of operations being performed on CPU and/or sequentially on GPU, rather than in parallel, and can be used to explain the relatively large elapsed time per training epoch reported by DNN_NeuroSim_V2.1, as depicted in Fig. 3.4 (c).

To quantify the performance trade-off between GPU VRAM usage and training time, Fig. 3.4 (f) was constructed. The baseline training routine clearly exhibits the best performance trade-off. Our findings suggest that DNN_NeuroSim_V2.1 is capable of simulating the training routine of larger and more complex network architectures, however, it does not fully utilize CUDA, and is much slower than other simulation frameworks. In contrast, the IBM Analog Hardware Acceleration Kit fully utilizes CUDA, and is comparable in performance to the native `torch` library. However, the IBM Analog Hardware Acceleration Kit consumes a large amount of VRAM, is unable to simulate modular crossbar tiles, and is consequently unable to simulating the training routine of larger and more complex network architectures.

3.4.3 Inference Routine Comparison

In Fig. 3.5, the performance of inference routines for the GoogLeNet network architecture using the CIFAR-10 dataset are compared. Inference was performed using a batch size of 128. As can be seen in Fig. 3.5 (c), the IBM Analog Hardware Acceleration Kit is capable of simulating

inference routines significantly faster than the MemTorch and DNN_NeuroSim_V2.1 simulation frameworks. This is while consuming more VRAM and approximately the same amount of RAM. We largely attribute this to the fact that the IBM Analog Hardware Acceleration Kit is unable to simulate modular crossbar tiles, which are difficult to parallelize using CUDA. When modular crossbar tiles are not simulated, when sufficiently small WL voltages are used to encode inputs, conventional VMMs can be used to determine output currents when 1T1R crossbars are modeled.

MemTorch and DNN_NeuroSim_V2.1 consume a similar amount of RAM and VRAM, however, MemTorch is approximately one order of magnitude slower than DNN_NeuroSim_V2.1, despite having a higher GPU utilization. We believe this is largely attrbutred to MemTorch's inefficient default weight-mapping scheme, as depicted in Fig. 3.5 (c) and Fig. 3.5 (d). This is especially evident when simulating large CNNs with many small convolutional layers, such as GoogLeNet. MemTorch stores convolutional kernels in a staggered arrangement, and does not share adjacent modular crossbar tiles between layers. DNN_NeuroSim_V2.1 utilizes proprietary weight mapping and data flow schemes [244], which significantly improves performance. We note that both DNN_NeuroSim_V2.1 and MemTorch under-utilize VRAM during inference, and both perform some operations sequentially and/or on CPU.

As can be seen in Fig. 3.5 (d), our findings suggest that the IBM Analog Hardware Acceleration Kit is able to utilize VRAM to the greatest extent, however, it is unable to simulate modular crossbar tiles. DNN_NeuroSim_V2.1 is able to simulate inference routines significantly faster than MemTorch, however, it is not as customizable, as it utilizes proprietary weight mapping and data flow schemes, which cannot be easily modified.

3.5 Outlook

It is evident that MDLS and memristive simulation frameworks are becoming increasingly useful and popular. While the reliable, large-scale operation of reconfigurable MDLS is still arguably an open problem [245], modernized simulation frameworks and tools enable researchers from a variety of disciplines to rapidly and accurately model the behavior and operation of MDLS without specialized circuit-level SPICE simulation expertise. This is in addition to the ability to work in tandem with existing modernized ML libraries. As these simulation frameworks and the models used to simulate non-ideal circuit and device characteristics mature and grow in popularity, the development cycle and production of innovative device technologies and MDLS architectures will also continue. These new devices and architectures can be conveniently integrated into the existing tools, facilitating their quick large-scale adoption.

An increasing number of simulation frameworks have been improved using measurements from experimental data, validating their reliable and accurate operation. In future, we expect CAD tools to (i) support the end-to-end characterization of memristive devices, (ii) be natively integrated within more mature and standardized MDLS design-flows, and (iii) be capable of programming

future physical re-programmable memristive circuits [133,246,247]. Such IMC simulation frameworks will be instrumental to the design of next generation of AI hardware [240].

3.6 Conclusion

In this Chapter, we presented a survey of current simulation frameworks and related tools to model and simulate IMC MDLS. In addition, we presented a detailed comparison of modern simulation frameworks that support pre-trained DNN conversion and TF/PyTorch integration. This was performed by directly comparing the training and inference routines of two popular CNN architectures using open-source modernized simulation frameworks. Furthermore, we provided an outlook/perspective into the future of CAD tools for modeling and simulating MDLS. We demonstrated that modern simulation frameworks are complimentary in nature, and can be used by a variety of users with different requirements to facilitate current research efforts in the domains of IMC and unconventional computing.

Chapter 4

Seizure Detection and Prediction by Parallel Memristive Convolutional Neural Networks

During the past two decades, epileptic seizure detection and prediction algorithms have evolved rapidly. However, despite significant performance improvements, their hardware implementation using conventional technologies, such as Complementary Metal–Oxide–Semiconductor (CMOS), in power and area-constrained settings remains a challenging task; especially when many recording channels are used. In this Chapter, the first and third research questions are addressed, and a novel low-latency parallel Convolutional Neural Network (CNN) architecture that is estimated to consume approximately 2.791W of power while occupying an area of 31.255mm^2 in a 22nm FDSOI CMOS process is presented. The proposed novel architecture achieves State-Of-The-Art (SOTA) accuracy performance across three epileptic seizure detection and prediction datasets.

4.1 Introduction

Epilepsy is a common neurological disorder that affects approximately 1% of the world's population [248]. A seizure is characterized by excessive firing of neurons in the brain, while epilepsy is a medical condition that involves recurrent seizures [249]. As the underlying occurrence mechanism of epilepsy is not well understood [250–252], it requires experimental methods of treatment that rely on accurate detection and prediction systems, as depicted in Fig. 4.1.

EEG is the most common method used to monitor the electrical activities of the brain, and can be used to detect and predict seizures. There have been numerous applications of traditional ML algorithms, such as SVMs, k-Nearest Neighbor (kNN), and Random Forest (RF) classifiers to classify ictal (seizure), preictal (prior to a seizure) and non-ictal (non-seizure) signals using EEG recordings. Despite being able to achieve high accuracies, these approaches require the manual extraction and selection of features in the time- or frequency-domain [253]. The optimal choice of such feature extractions are largely unknown, experimental, and dependant on specific patient signatures, such that there is no one-fit-all solution.

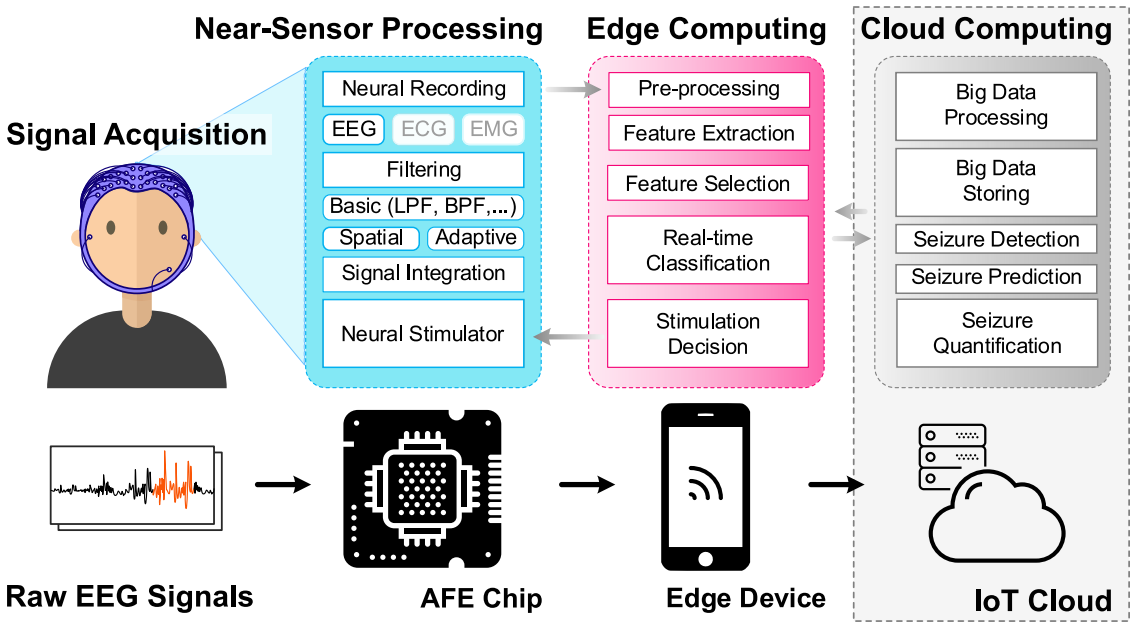


Figure 4.1: An overview of a typical epileptic seizure detection and prediction system. Acquired EEG signals are sampled and processed near-sensor using an Analog Front End (AFE), prior to being sent wirelessly to edge device(s) for real-time pre-processing and feature extraction. Features can then be fed into ML and/or DL architectures, residing either on the IoT edge or in the IoT cloud, which perform epileptic seizure detection and prediction.

Compared to traditional seizure classification algorithms, DL-based algorithms have more advantages in complex EEG signal feature extraction, as they do not require feature engineering, and are capable of outperforming traditional ML algorithms for epileptic seizure detection and prediction tasks [254]. However, when these DL systems are implemented using CMOS, there are problems such as large scale, high calculation energy consumption and high delay, which hinder their efficacy; especially in resource-constrained environments.

In order to solve this kind of problem, this Chapter proposes a neuromorphic calculation strategy based on a novel IMC RRAM architecture, which utilizes analog crossbars. Computer designers have traditionally separated the role of storage and compute units. The IMC paradigm blurs this distinction, and imposes the dual responsibility on memory substrates: storing and computing on data for massively parallel computing [255]. By exploiting the physical characteristics of emerging analog device technologies, analog crossbars can be used to perform VMMs, the most dominant operation in CNNs, in as little as $\mathcal{O}(1)$ [8,256], significantly reducing the computational complexity during inference operations. Our specific contributions are as follows:

1. To the best of our knowledge, we are the first to parallelize the execution of convolution layer kernels on separate analog crossbars to address the computational bottleneck of CNNs, enabling 2 orders of magnitude reduction in latency compared to current SOTA hybrid

Memristive-CMOS DL accelerators;

2. We reduce the number of required parameters by 2-1,600x and 5-2,800x for epileptic seizure detection and prediction tasks using deep learning models, while still achieving SOTA performance;

3. We provide a comprehensive benchmark for hardware memristor-based seizure prediction/ detection systems by simulating, laying out, and determining hardware requirements of the CNN component of our system;

4. We propose a simplified stuck weight offsetting methodology for mitigating severe degradation of system performance due to stuck $R_{\text{ON}}/R_{\text{OFF}}$ memristor weights. We demonstrate that our method is capable of achieving up to 32% performance recovery, without requiring retraining, while incurring minimal hardware and computational overhead.

To promote reproducible research, all of our simulation codes are made publicly accessible[1]. The rest of this Chapter is structured as follows: In Section 4.2, we overview and discuss related work. In Section 4.3, we present our epileptic seizure detection and prediction system. In Section 4.4, we overview and discuss our software methodology. In Section 4.5, we overview and discuss our hardware simulation methodology. In Section 4.6, we present and discuss our results. Finally, we conclude this Chapter in Section 4.7.

4.2 Related Work

In this Section, we present an overview of related work using parallel CNNs and related work using traditional and neuromorphic ML architectures for the detection and prediction of epileptic seizures using EEG and iEEG signals.

4.2.1 Parallel CNNs

Parallel CNNs are composed of one or many convolutional layers, which are executed in parallel and have been previously used in many applications. For example, in the ResNeXt [257] family of architectures, parallel blocks containing convolutional layers were used to increase network width, which can decrease the time required to train a CNN [258]. When performing multi-modal DL, parallel convolutional layers can be used to process different inputs in parallel [259], in order to improve network throughput. Specifically for epileptic seizure detection and prediction tasks, parallel convolutional layers have been used to learn high-level representations simultaneously [260]. By parallelizing convolutional operations, inference time is greatly reduced compared to current SOTA architecture that rely on sequential convolution layers, as convolution layers form the bottleneck of CNN inference.

[1] https://github.com/coreylammie/Memristive-Seizure-Detection-and-Prediction-by-Parallel-Convolutional-Neural-Networks

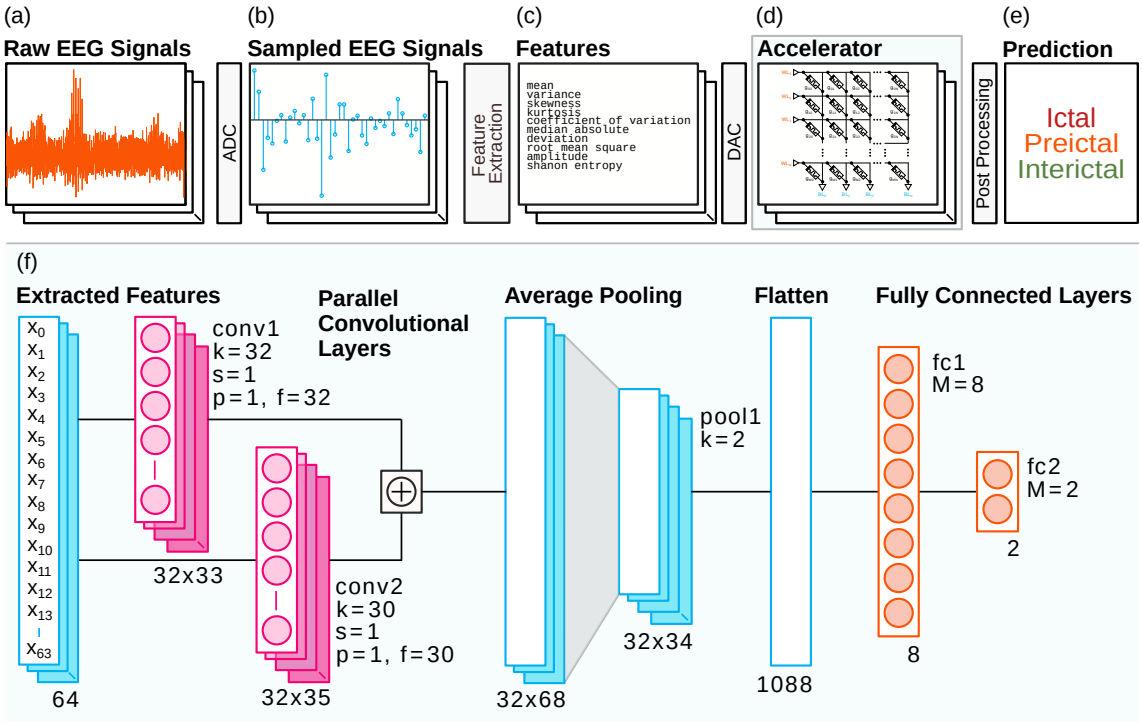


Figure 4.2: A high-level system architecture overview. (a) Raw EEG signals are sampled and digitized using ADCs. (b) Features are extracted from sampled EEG signals. (c) Extracted features are fed into a memristive DL accelerator. (d) Accelerator outputs are processed. Fig. 4.3 depicts the detailed hardware implementation of the accelerator. (e) Processed accelerator outputs are used to determine interictal, preictal, and ictal states. (f) The novel neural network architecture used consists of two parallel 1d-convolutional layers, one average pooling layer, and two fully connected (dense) layers. N is used to denote the batch size, i.e., the number of batches presented to the network in parallel. f denotes the number of filter. k determines the filter size. s denotes the stride length. p denotes the padding. M denotes the number of output neurons for each fully connected layer. Parts of this figure are derived from [6].

4.2.2 Traditional EEG-based Seizure Detection and Prediction Algorithms

As early as 1996, initial attempts were made to detect seizures using EEG signals and traditional ML approaches. Using a combination of ANNs and wavelet transforms, sensitivity values of 76% [261] and 97% [262], were reported using standardized datasets. In the late 2000s and early 2010s, SVMs encountered growing interest. Namely, when using SVMs in combination with feature extraction methods such as high-order spectra analysis, wavelet transforms, Fast Fourier Transforms (FFTs), wavelet decomposition and least-squares parameter estimators [263–272], promising sensitivity, specificity, and accuracy values $\geq$98.5% were achieved. More recently, advances in the DL domain using CNNs and RNNs, have further benefited seizure detection algorithms. Current SOTA models are capable of achieving accuracy ranging from 95-100% [118, 273–275] across multiple datasets.

Early efforts for seizure prediction started in 1970s, where seizure warning systems were designed with logic circuitry to classify extracted features from a series of filters and analog circuitry [276, 277]. To varying degrees of success, a variety of methods have been proposed, including a rule-based method using univariate measures [278], spike rate analysis [279], positive zero-crossing intervals analysis [280], statistical dispersion measures [281], multidimensional probability evolution [282], circadian concepts via probabilistic forecasting [283], and a combination of reinforcement learning, online monitoring and adaptive control theory [284]. Similarly to seizure detection, many DL techniques have also been applied. Notable contributions include the combination of CNNs and RNNs, capable of achieving 99.6% accuracy and a False Positive Rate (FPR) of 0.004 per hour [285]. Moreover, supervised deep convolutional autoencoder and bidirectional long short-term memory networks have been used to achieve accuracy, sensitivity, specificity, and precision values between 98-99%, with F1-values $\geq$0.98. More recently, augmented DL network architectures have been used to reduce computational complexity for operation in resource-constrained environments. One such approach, which employs CNNs with minimizing channels, is capable of achieving 99.47% accuracy, 97.83% sensitivity, 92.36% specificity, with a FPR of 0.0764 [286]. Finally, Siamese models have been used to achieve 88-91% accuracy on the CHB-MIT dataset [287]. We refer the reader to [288] for a comprehensive survey of EEG seizure detection and prediction algorithms.

4.2.3 Hardware Implementations of EEG-based Seizure Detection and Prediction Algorithms

Many hardware implementations of epileptic seizure detection and prediction algorithms have been reported using a variety of technologies; namely FPGA, CMOS and Very-large-scale Integration (VLSI) [1, 289]. Complementing traditional hardware implementations, IMC architectures, which use memristive crossbar arrays to perform repetitive operations in-memory, have gained increasing popularity in recent years. Kudithipudi *et al.* implemented a neuromemristive reservoir computing architecture to achieve 90% accuracy and Merkel *et al.* achieved 85% accuracy [290, 291]. Nature-inspired memristive Cellular Automata (CA) was implemented by Karamani *et al.* to emulate epilepsy-related phenomena in the brain [292].

Recent works by Liu *et al.* implemented Finite Impulse Response (FIR) filter bank on memristive crossbar array to achieve 93.46% seizure detection accuracy and obtained 95% accuracy by using a memristive crossbar based signal-processing stage combined with linear discriminant classifier [293]. Lammie *et al.* pioneered the implementation of CNNs for seizure prediction using memristor arrays, achieving 77.4% sensitivity and 0.85 Area Under the Receiver Operating Characteristic Curve (AUROC) on the CHB-MIT dataset [6].

Seizure is a chronic, recurring condition that can mostly be prevented through medication before onset [294], but even with the best medications, 30% of the patients are drug-resistant [295]. Closed-loop brain stimulation has been found to mitigate and even improve symptoms [296, 297],

but unpredictability of seizure requires a closed-loop prediction system to provide accurate warning with adequate preparation time for stimulation [298]. This calls for the need for fast, low-latency computations, as the changes within the patients can be noticed early-on, in order to start treatments early to improve safety and quality of life [299]. In doing so, symptoms and subsequent effects can be minimized, including anxiety and social exposition [300]. The major limiting factor of seizure detection and prediction algorithms is the reliance on patient specific features, leading to undesirable results when generalized to other patients in the real world [301]. With energy efficient computations, it enables the deployment of such systems within wearable devices, so that it can be coupled with the stimulation system, as well as allowing data for a patient to be collected in the long-term to further improve model's predictions by fine tuning the model to better recognize patient-specific signatures [302].

It is known that convolutional layers are the bottlenecks of CNNs. According to Cong et al., convolutions make up more than 90% of CNN inference [303]. Therefore, accelerating convolution is pivotal to efficient CNNs for future seizure detection/prediction systems. Note that all existing hardware implementations of CNN memristive accelerators focus on sequential CNNs. Memristive crossbar acceleration of parallelized convolution layers and blocks, found in many CNN architectures such as ResNeXt [257], are explored in this work to further reduce inference latency.

4.3 Seizure Detection and Prediction System

In this Section, we present our seizure detection and prediction system. As shown in Fig. 4.2, our system comprises of five stages, depicted using Fig. 4.2(a)-(e). As the same network architecture, depicted in Fig. 4.2(f), is used for both detection and prediction, and networks are bench-marked using multiple datasets, our proposed system can be reconfigured for both epileptic seizure and prediction tasks. While we briefly detail and discuss signal acquisition and pre- and post-processing stages, here-on-in, the scope of this Chapter will be largely confined to the accelerator step described in Fig. 4.2(d). We leave a detailed hardware description and evaluation of other stages to future work.

4.3.1 Parallel Convolutional Neural Network Architecture

The primary constraint put on our design was a fixed modular tile size of 64×64. Practically, passive memristor-based analog crossbar tiles of sizes up to 128×64 have been used to perform VMMs [256], however such designs have only been demonstrated using pseudo-crossbars having micron-size electrodes. Such limitations in the maximum viable size are a serious computational scalability challenge with electrodes in the tenth of nanometer range that would prevent sinking large currents through them [304]. Recently, a 4K memristor analog-grade passive crossbar circuit has been fabricated [305], which comprises several modular 64 x 64 passive crossbar tiles with

99% functional nonvolatile metal-oxide memristors. From an original exploratory investigation, it was determined that for the RRAM device being modeled, the largest feasible modular tile size which is able to be programmed using a write-verify scheme was 64×64. Consequently, this fixed modular tile size was used in our designs to minimize the power and area overhead of peripheral circuits and tile interconnects, which are much larger when smaller fixed modular tiles are used.

4.3.2 Model Search and Selection

Most current state-of-the-art CNNs employ sequential convolution layers, whereby subsequent convolution operations are dependent on results from previous layers. However, in parallel CNNs, convolution layers can be processed simultaneously, enabling the use of multiple crossbars at the same time. In addition, parallel convolution layers with different kernel sizes enable the network to extract features of varying receptive fields, providing the fully connected layers a diverse and yet compact representation of the features for classification; enabling a reduction in network parameters required.

As shown in Fig. 4.2, our proposed CNN architecture consists of two parallel convolution kernels. Algorithm 1 formalizes the methodology used to search for and select the employed model. For our selected model, latency was minimized using OBJ_{min}. L, D, and $\boldsymbol{\beta}$ were fixed to values determined empirically using a preliminary exploratory analysis, and $\boldsymbol{\alpha}$ was optimized as per Algorithm 1. The following additional hardware design constraints were imposed for our design: all convolutional layers must be capable of fitting onto one modular crossbar tile, and the total number of required modular crossbar tiles must not exceed 8.

As the convolution operation bottlenecks CNN inference, the size of kernels used in parallel convolution layers need to be carefully considered to optimize both network performance and latency. In our proposed architecture, shown in Fig. 4.2(f), we have two parallel convolution layers and one average pooling layer, comprising one convolutional block. To parallelize the two convolution layers, it would be necessary to map the weights of the two convolution layers onto two separate crossbars. As a design choice, we wanted to retain the flexibility of mapping both convolution layers onto the same crossbar, if space complexity is prioritized over latency. Therefore, during the kernel size search, we imposed a constraint of 62, i.e., $m - 2$, for the sum of convolution kernels, as 2 additional rows are designated for implementing the bias for both parallel convolution layers.

When denoting the kernel size of the first parallel convolutional layer as α, the kernel size of the second parallel convolutional layer can be expressed as $62 - \alpha$. To determine the optimal network architecture, the University of Bonn's EEG seizure dataset [306] was used. Specifically, a 80:20 train validation split was employed, and EVAL(Net) was used to determine the 5-fold cross validation accuracy. Seed values of 32 and 8 were arbitrarily set for the network architecture search, to ensure reproducibility of results, and to reduce bias between search and validation.

Empirically, $L = 1$, $D = 2$, and $\boldsymbol{\beta}$=[8,] achieved substantial performance. For the single convo-

Algorithm 1 Model Search and Selection Methodology.

Input: Fixed modular crossbar tile size ($m \times n$), OBJ_{max}, objectives to minimize, OBJ_{min}, additional hardware design constraints, $\mathbf{w}$.

Output: Optimized network architecture $(L, D, \boldsymbol{\alpha}, \boldsymbol{\beta})$, where L is the number of convolutional layer blocks, D is the number of fully connected layers, $\boldsymbol{\alpha}$ is a vector containing the sizes of the first kernel for each convolutional layer when parallel convolutional layer execution is performed, and $\boldsymbol{\beta}$ is a vector containing the number of output neurons for each fully connected layer.

minimize $\text{OBJ}(m, n, L, D, \boldsymbol{\alpha}, \boldsymbol{\beta})$ subject to $\mathbf{w}$.

procedure NETWORK_ARCHITECTURE($m, n, L, D, \boldsymbol{\alpha}, \boldsymbol{\beta}$)
  for $l = 0$ to $L - 1$ **do** ▷ For each convolutional layer
    $\mathbf{C_{in}}_l$ = m ▷ Input channels
    $\mathbf{C_{out}}_l$ = floor(n / 2) ▷ Output channels
    if parallel convolutional layer execution **then**
      $\mathbf{k}_{l0} = \boldsymbol{\alpha}_l$, $\mathbf{k}_{l1} = m - 2 - \boldsymbol{\alpha}_l$ ▷ Set kernel sizes
    else
      $\mathbf{k}_l = m - 1$ ▷ Set kernel size
    end if
  end for
  for $d = 0$ to $D - 2$ **do** ▷ For each fully connected layer
    $\mathbf{m}_d = \boldsymbol{\beta}_l$ ▷ Set number of output neurons
  end for
  $\mathbf{m}_{D-1} = 2$ ▷ Last layer
end procedure

function $\text{OBJ}(m, n, L, D, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{w})$
  maximize EVAL(Net) and **minimize** PARAMS(Net), ▷ i.e., determine L, D, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$, where EVAL determines the validation accuracy, and PARAMS determines the total number of network parameters
  where,
  Net = NETWORK_ARCHITECTURE($m, n, L, D, \boldsymbol{\alpha}, \boldsymbol{\beta}$) **return** OBJ_{min}(Net)
end function

lutional block, $\boldsymbol{\alpha}_0$ was varied between 31 and 60. A validation accuracy of 100% was achieved for all values of $\boldsymbol{\alpha}_0$, except for $\boldsymbol{\alpha}_0 = 60$, which achieved an optimal validation accuracy of 99.375%. This is not surprising, as the window size of input data is only 64. Therefore, convolution kernel sizes of 60 and 2 provides two extreme and dramatically different receptive fields. In particular, a kernel size of 2, which corresponds to around 10ms of data at 173.61Hz, is likely insufficient to capture local correlation and learn seizure characteristics. The final model was chosen using Occam's razor principle, whereby the simplest model is the best model. Consequently, a kernel size of 32 was selected, as a kernel size 31 would be the simplest to implement due to symmetric convolution kernel sizes; however 32 provides a more diverse receptive field. To further demonstrate the advantage of varied kernel size, a 5-fold cross-validation was performed using a) 64 filters of kernel size 31 b) two parallel convolution layers each with 32 filters of kernel size 30 and 32 (see

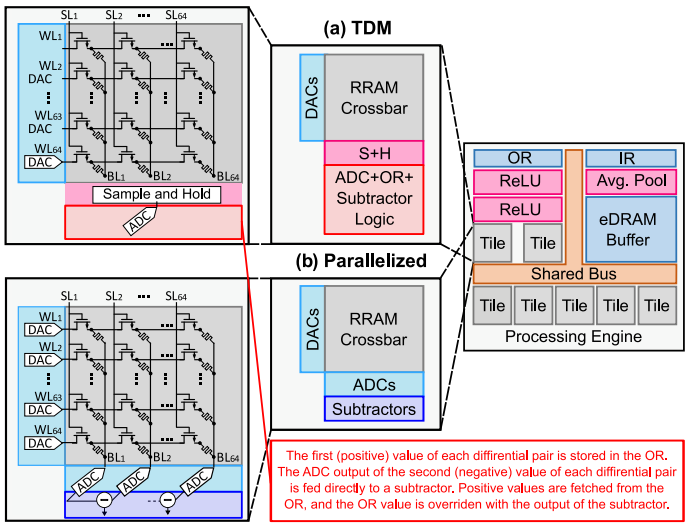


Figure 4.3: Architecture hierarchy of our memristive DL accelerator with (a) TDM and (b) Parallelized Implementation.

Fig. 4.2). It was observed that both networks are capable of achieving accuracy varying between 99.61% to 99.83%, but varied kernel size leads to +0.03%, -0.01%, +0.02% change in performance on Bonn, SWEC-ETHZ and CHBMIT datasets, respectively, compared to using 64 filters of kernel size 31. Although a small degradation in performance is observed for SWEC-ETHZ dataset, improvements are observed for both Bonn and CHBMIT dataset. A net improvement is observed for both seizure detection and prediction using a varied kernel size, while both experiments employ an identical number of weights.

4.3.3 Hardware Architecture Hierarchy

In Fig 4.3, we present our hardware architecture hierarchy. The processing engines comprises 7 memristive crossbar array tiles, as well as I/O registers, eDRAM buffers, and peripheral circuits for ReLU, subtract, and average pooling. We present two configurations for our tile, Time-Division Multiplexing (TDM), and parallelized. In the TDM case, each tile contained a S+H and an ADC for reading out column currents, and one DAC per row for reading inputs in parallel, as shown in Fig. 4.3(a). In the parallelized case, each tile contains 64 ADCs, as shown in Fig. 4.3(b).

4.4 Software Methodology

To train and evaluate our epileptic seizure detection and prediction system, we benchmarked our system using one epileptic seizure detection task and two epileptic seizure prediction tasks. For epileptic seizure detection, the University of Bonn's EEG seizure dataset [306] was used. For epileptic seizure prediction, the CHB-MIT Scalp EEG [307], and the long-term SWEC-ETHZ iEEG [308] datasets were used.

To perform epileptic seizure detection and prediction, EEG and iEEG samples can be categorized as either ictal, interictal or preictal. Ictal samples indicate the presence of a seizure, interictal samples are periods between seizures, and preictal samples can be used to detect the onset of a seizure. For epileptic seizure detection, binary classification is performed between ictal and interictal samples. For epileptic seizure prediction, binary classification is performed between preictal and interictal samples. For both epileptic seizure detection and prediction tasks, on account of unbalanced classes, 5-fold cross validation was used to train and validate our network architecture.

4.4.1 Training and Evaluation Methodologies

Epileptic Seizure Detection

The University of Bonn's EEG seizure dataset is comprised of 5 sets (A-E), where set A is normal with open eyes, set B is normal with closed eyes, set C and D is seizure free intervals, and set E is seizure only activities. Each set contains 100 single-channel EEG time series of 23.6 seconds, with 4,096 samples in each time series. All data were collected at 173.61 Hz, at a resolution of 12 bits. To perform binary classification between ictal and interictal samples, all samples from sets A and E were used.

Both sets (A and E) were divided into samples of 64 seconds periods and randomly shuffled. No augmentation and pre-processing techniques, such as normalization, were performed, as CNNs are capable of automatic feature extraction from time-series data and are robust to noise. The lack of need for pre-processing steps implies reduced hardware complexity to perform such operations. Using the network model (with optimal kernel sizes determined in Section 4.3.2), a 5-fold cross-validation strategy was used to determine network's performance. To determine performance, the mean of left out set accuracy, sensitivity, specificity, false-positive rate and the AUROC across folds of 5-fold cross-validation were reported.

Epileptic Seizure Prediction

The CHB-MIT Scalp EEG, and the long-term SWEC-ETHZ iEEG datasets were used. The CHB-MIT Scalp EEG dataset comprises of 23 cases, which were collected from 22 subjects (5 males, ages 3–22; and 17 females, ages 1.5–19). The last case was obtained 1.5 years after the first, from one of the female subjects [307]. All signals were sampled at 256Hz with 16-bit resolution, using 23-26 electrodes. During data acquisition, no augmentation steps were performed.

The long-term SWEC-ETHZ iEEG dataset comprises of 18 patients with pharmaco-resistant epilepsy, who were evaluated for surgery at the Sleep-Wake-Epilepsy-Center (SWEC) of the University Department of Neurology at the Inselspital Bern [308]. All signals were sampled at either 512Hz or 1025Hz with 16-bit resolution, using 26-100 electrodes. During data acquisition, after analog-to-digital conversion, a digital band-pass filter was used to filter signals between 0.5 and 150Hz using a fourth-order Butterworth filter. Moreover, forward and backward filtering was

Table 4.1: Overview of cases used to perform epileptic seizure prediction from the CHB-MIT Scalp EEG (CHB-MIT) and the long-term SWEC-ETHZ iEEG (SWEC-ETHZ) datasets.

Patient	Seizures	Interictal Hrs.*	Preictal Hrs.*	Interical Smp.†	Preictal Smp.◇	Synthetic Preictal Smp.◇
CHB-MIT						
1	7	33.74	0.43	1,898	24	42
2	3	32.85	0.14	1,848	8	14
3	7	30.86	0.39	1,736	22	37
5	5	33.85	0.30	1,904	17	30
8	5	14.93	0.36	840	20	3
SWEC-ETHZ						
1	2	19.91	1.00	1,120	56	108
2	2	19.91	1.00	1,129	56	108
3	4	29.87	1.99	1,680	112	216
5	4	29.87	1.99	1,680	112	216
6	8	69.69	3.48	3,920	196	430

*Hours. †Samples.

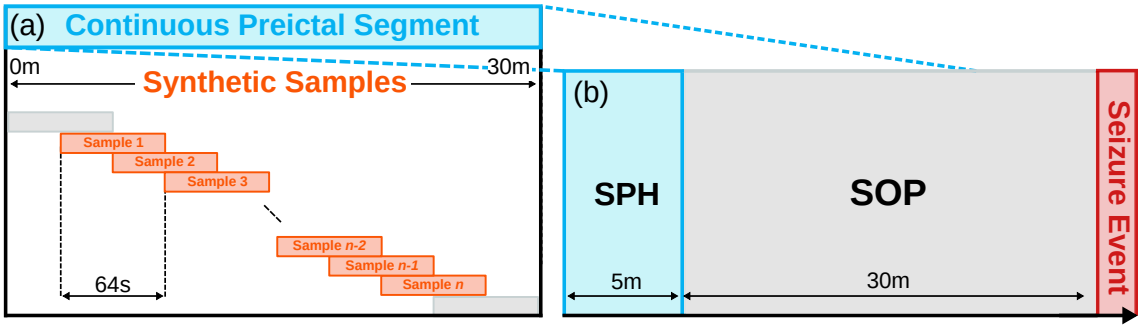


Figure 4.4: Depiction of (a) our adopted overlapped sampling technique extracting n samples from a continuous preictal segment, and (b) the SPH and SOP terms. As can be seen, continuous preictal segments are extracted during the SPH. All preictal samples that occur during the SOP period are discarded.

applied to minimize phase distortion.

Due to computation burden of crossbar simulation, we report the performance using the first 5 viable cases of the the CHB-MIT Scalp EEG and long-term SWEC-ETHZ iEEG datasets, reducing the computation required, similar to [260, 309]. In Table 4.1, we present an overview of all cases used to perform binary classification between preictal and interictal samples. A case was categorized as viable if it contained valid labels (namely time-stamps) and data files (i.e., no recording files were missing or corrupt). For both datasets, the first 22 channels of each patient were extracted and used. All signals were down-sampled to 256Hz, and a window size (batch size) of 64s was used when extracting samples. After discarding seizures that occur in the first 20-minute monitoring period, a Seizure Occurance Period (SOP) of 30m and a Seizure Prediction Horizon (SPH) of 5m were used to extract and label preictal samples for all cases; both of which

have previously demonstrated significant performance [309]. These terms are defined visually in Fig. 4.4. Interictal samples were extracted from one hour recording segments containing no seizures (ictal samples) to reduce class inbalance during training.

Next, 176 features per sample were extracted (8 per channel per window/batch interval): the mean, variance, skewness, kurtosis, coefficient of variation, median absolute deviation of EEG amplitude and Root Mean Square Amplitude (RMSA), and the shannon entropy. Since the input size of the proposed network is 64, the dimensionality of the input data needed to be reduced. A correlation analysis was first performed across the 176 extracted features, but no particular channel could be removed as no strongly correlated channels were discovered. Using Principal Component Analysis (PCA), linear dimensionality reduction via Singular Value Decomposition (SVD) enabled the projection of data to lower dimensional space of 64 principal axes. During training, synthetic preictal samples were generated using an overlapped sampling technique inspired by [288], by sliding a 64s window with a stride of 32s across continuous preictal segments extracted during the SPH period, as depicted in Fig. 4.4. The same cross-validation training and evaluation strategy and metrics as described in Section 4.4.1 was employed.

4.5 Hardware Methodology

In this Section, we discuss our device technology selection, memristor crossbar array implementations of CNNs, and present our adopted hardware simulation methodology.

4.5.1 Device Technology Selection

Computing with charge-based computing devices is attractive due to their technological maturity, even though they have a relatively large area footprint even at advanced technology nodes and face severe scaling challenges [200]. Resistance-based memory, in contrast, can be scaled to the nanometer scale, and has the potential of forming cross-point structures without using access devices, achieving ultra high density. RRAM devices are used in our design, as they are widely considered to be the most promising emerging resistance-based memory technology- they operate faster than PCM, have a simpler and smaller cell structure than MRAM and CBRAM devices, and are made of materials that are common in semiconductor manufacturing [200].

4.5.2 Memristor Crossbar Array Implementations of Parallel CNNs

Consider the conductance values of a crossbar array as a matrix and input voltages to a crossbar as a vector. The output current from the crossbar, determined using Kirchoff's and Ohm's Law represents the result of the VMM. Such operations form the core of CNNs. Being able to accelerate and parallelize them would facilitate the real-time operation of deeper and heavier neural networks for epileptic seziure detection and prediction in resource-constrained hardware [145].

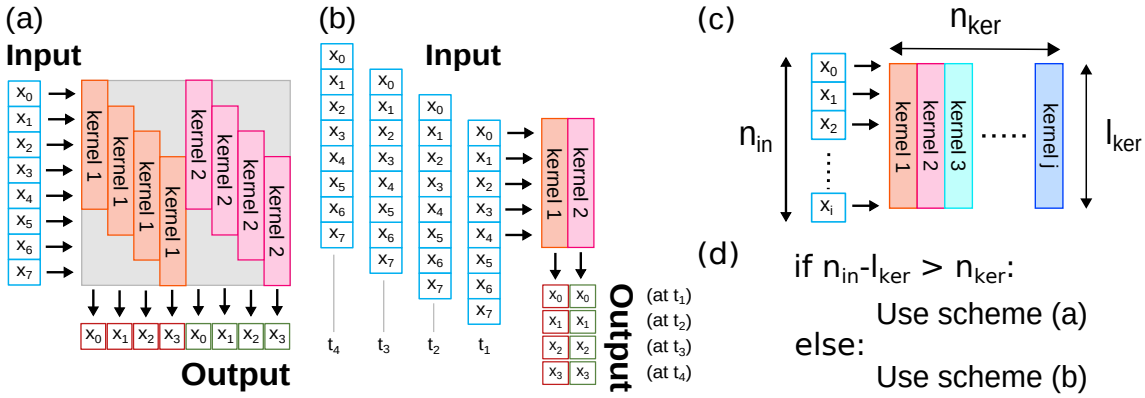


Figure 4.5: A comparison of possible mapping schemes. (a) visualizes the staggering mapping of convolution weights, which is commonly adopted due to its ability to produce all results within a single pass through the crossbar array. (b) visualizes our proposed mapping scheme, without staggering of convolution weights and sparsity in crossbar, at the cost of increased read/write operations. (c) provides a comparison of methods (a) and (b), visualizing when one method should be chosen over the other.

Table 4.2: Crossbar mapping comparison for space and computation trade-off using schemes (a) and (b) in Fig. 4.5.

Layer	Number of Memristor Cell Required				Number of Memristor Cell Required Inc. Sparsity			
	Scheme (a)	Scheme (b)	Area Reduction	Computation Increase	Scheme (a)	Scheme (b)	Area Reduction	Computation Increase
conv1	69,696	2,112	33x	33x	133,184	2,112	63x	33x
conv2	69,440	1,984	35x	35x	145,600	1,984	73x	35x
fc1	17,424	17,424	None	None	17,424	17,424	None	None
fc2	36	36	None	None	36	36	None	None

To represent signed weight matrices on memristive crossbar arrays, as negative conductance values cannot be expressed using analog memristive devices, a differential mapping scheme was adopted, where two columns of memristors are chosen to represent positive and negative weights, respectively. The signed output is thus the arithmetic difference of current from both columns. In the case of 1D CNNs, fully connected and convolutional layers can be decomposed into a series of dot products between inputs, represented as voltages, and weights, represented as memristive conductance. For convolutional layers, the `im2col` algorithm [310] can be used to map convolutional kernels onto separate crossbar columns. With a single pass, m 1D convolutions can be performed simultaneously, where m represents the number of columns. Average pooling and ReLU operations are performed using additional digital circuitry.

4.5.3 Hardware Simulation Methodology

Based on existing literature from Section 4.2.3, all mapping of convolution kernels onto crossbars are sparse, whereby the convolution kernels form a sparse diagonal matrix, as depicted in Fig. 4.5(a). This naive approach is extremely space demanding, as the kernels are staggered multiple times throughout the crossbar array, rendering a lot of memristive cells unused. To reduce

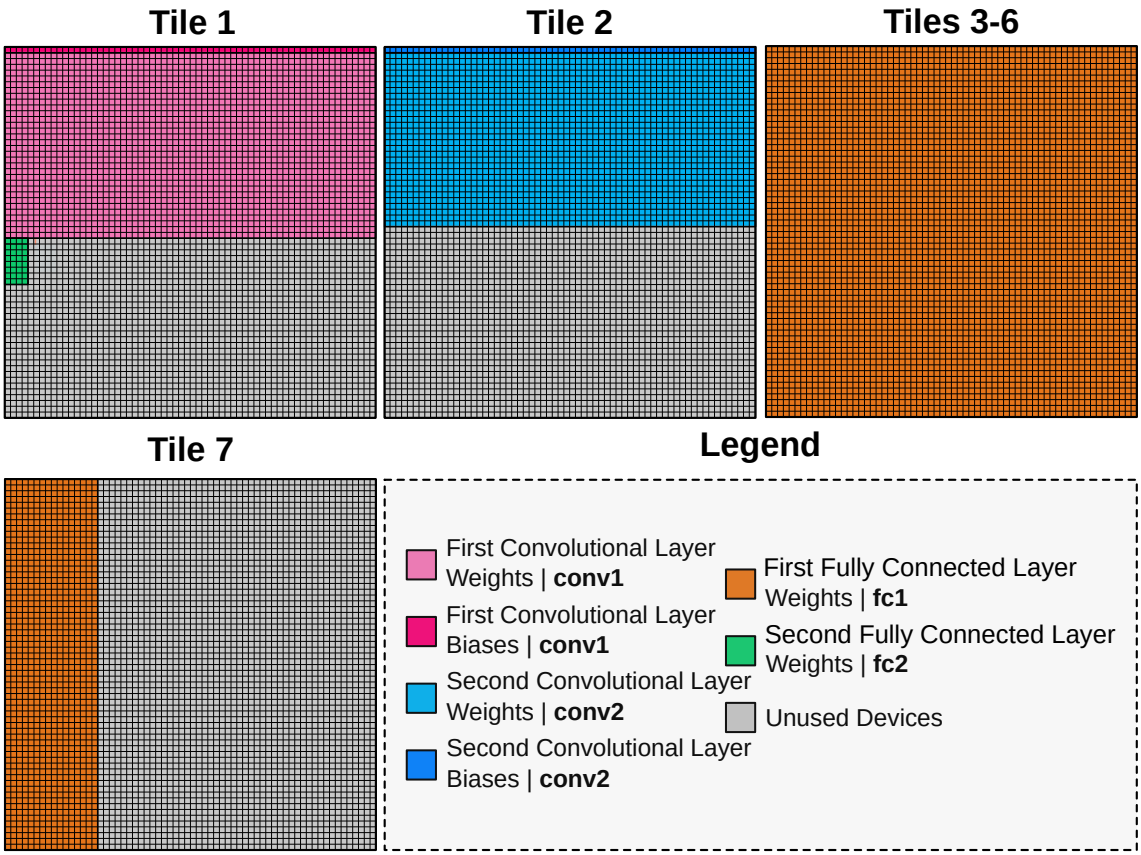


Figure 4.6: The crossbar parameter mapping layout adopted. Seven 64 $\times$ 64 modular crossbar tiles are utilized. Bias terms of fully connected layers, and the single pooling layer, `pool1`, are computed using additional digital circuitry. To reduce the number of unused devices, parameters of different layers are shared between tiles.

the space requirement of mapping scheme (a), one possible approach is to build upon the input-stationary concept. One may remap the crossbar weights during inference and replace them with different kernel weights, while reusing the input fetched from memory.

On the other hand, one may build upon the weight-stationary concept, as depicted in Fig. 4.5(b). In this scheme, convolution kernels can be mapped without staggering before inference. For kernels to convolve against different parts of the signal, the input signal slides. The bottleneck of this approach now lies within fetching input data, requiring additional read/write operations on the peripheral of the crossbar compared to mapping scheme (a). The weight-stationary approach is more efficient compared to the input-stationary approach, as crossbar weight writes can be very time and energy consuming, compared to fetching of inputs and staggering them with shifting circuitry. Fig. 4.5(c) provides visualization of when one scheme should be adopted over the other.

A comparison of the naive approach and our proposed weight-stationary approach is performed for our network architecture in Table 4.2. As can be observed, the number of memristor cells required for scheme (b) (depicted in Fig. 4.5 (b)) is significantly smaller, due to the compact nature

Table 4.3: 5-Fold cross-validation result for epileptic seizure detection and prediction using our network architecture.

Dataset	**Bonn**	**CHB-MIT**					**SWEC-ETHZ**				
Partition	**Set A vs. E**	**Patient 1**	**Patient 2**	**Patient 3**	**Patient 5**	**Patient 8**	**Patient 1**	**Patient 2**	**Patient 3**	**Patient 5**	**Patient 6**
Accuracy	99.84 ± 0.37	99.50 ± 0.89	99.95 ± 0.11	99.95 ± 0.13	99.73 ± 0.57	98.96 ± 2.33	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.86 ± 0.22	100.00 ± 0.00
Sensitivity	99.87 ± 0.28	98.64 ± 2.79	100.00 ± 0.00	100.00 ± 0.00	99.62 ± 0.70	99.76 ± 0.54	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
Specificity	99.80 ± 0.45	99.73 ± 0.37	100.00 ± 0.00	99.93 ± 0.15	99.77 ± 0.52	97.38 ± 5.85	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.77 ± 0.39	100.00 ± 0.00
FP per Hour	N/A	0.13 ± 0.17	0.00 ± 0.00	0.03 ± 0.07	0.10 ± 0.22	0.53 ± 1.19	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.08 ± 0.13	0.00 ± 0.00
AUROC	99.84 ± 0.37	99.31 ± 1.06	100.00 ± 0.00	99.82 ± 0.39	99.63 ± 0.79	99.04 ± 2.15	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.84 ± 0.25	100.00 ± 0.00

of the mapping. This comes, however, at the cost of 33x increase in computation. When taking sparsity, i.e. unused memristors depicted by the gray background in Fig. 4.5 (a), into consideration, scheme (b) demonstrates even more significant reduction, i.e. 63x-73x fewer memristors required, while the computation increase remains constant. Unlike convolutional layers, fully connected layers do not involve sliding of signals, so VMMs for fully connected layers were implemented using the naive scheme (a). Using scheme (b), we mapped convolutional kernels within our trained network onto crossbars tiles of 64×64. While scheme (b) was chosen for our hardware design, if scheme (a) were chosen with different n_{ker} and l_{ker} values, or the added space complexity is not of concern, the staggered weights of scheme (a) would enable all rows of the crossbars to be employed simultaneously. By choosing the input size of our network to be 64, we maintain the flexibility of mapping with scheme (a) to make use of all crossbar rows simultaneously.

As Fig. 4.6 demonstrates, for parallel convolution layers to be accelerated simultaneously, it was necessary to map the weights of the `conv1` and `conv2` onto two separate crossbar tiles. The weight of the `fc1` layer is a matrix of 1088×8, and using a differential weight scheme, would require 1088×16 memristors. The weight matrix can be further divided into 17 sections of 64×16 weights. To maximize the usage of each 64×64 crossbar array, 4 sections of 64×16 weights can be stacked horizontally onto each crossbar, requiring a total of 5 crossbar tiles.

Since there are unused memristors on the convolution tiles and `fc2` layer operations are not performed immediately after convolution operations, we decided to map the weights of `fc2` onto the convolution layer tile, instead of using another tile. Note that since the simulation serves as a validation for proof-of-concept, we decided to use the same dimensions for all 7 crossbar tiles. We do recognize that tile 1, 2 and 7 have many unused memristor devices, as a result, performing small VMMs on a large switch matrix. This leads to large power overhead due to high amortized ADC/DAC power over a small matrix and charge/discharge of long row and column wires without using full length for computation. To address such problem in a real medical device, instead of using square tiles, tile 1, 2 and 7 can be easily mapped onto rectangular tiles of the exact required dimensions.

4.5.4 Impact of Device and Crossbar Non-Idealities

Memristors and memristive crossbar arrays are prone to numerous device and circuit non-idealities which have been demonstrated to severely impact the performance of memristive DL accelera-

tors [147]. Consequently, they should be comprehensively simulated prior to circuit-level realization. In this Chapter, preliminary simulations were performed using the MemTorch [15] simulation framework, and comprehensive simulations of the system using passive crossbar arrays were performed using the crossbar array model provided by [311]. Non-idealities considered include input and output resolutions, weight write resolution, weight write deviation, stuck $R_{\text{ON}}/R_{\text{OFF}}$ devices, line and source resistance, and conductance range variation.

Other memristive phenomena, such as the dynamic behavior of switched memristive neural networks after programming [312], and read disturbance [313], are not accounted for, as practical metal-oxide memristors are endurance-limited, during programming a write-verify scheme is used, and during inference, all BL voltages are constrained to have a maximum absolute amplitude of 0.3V [313].

4.5.5 Stuck Weight Offsetting Methodology

Stuck $R_{\text{ON}}/R_{\text{OFF}}$ weights are known to cause significant network performance degradation in memristive crossbar arrays. Existing works have demonstrated performance recovery through a variety of techniques. In 2014, Kannan et al. took inspiration from SRAM/DRAM technologies and repaired crossbar defects using redundant rows and columns [314]. In 2017, Liu et al. proposed to identify significant weights before applying a retraining and remapping algorithm [315]. In 2018, Xia et al. proposed a mapping algorithm with inner fault tolerance to leverage the differential mapping scheme of crossbar arrays to tolerate faults [316]. In 2019, Zhang et al. proposed the use of matrix transformations to reduce the magnitude of error introduced by stuck-at-fault devices [317]. Also in 2019, Yeo et al. modified conventional transimpedance amplifiers to detect when abnormal current is detected at a particular column due to stuck-at-fault devices and repair by retraining the network with the known defects [318]. Among those works, significant hardware or software overhead is introduced through rewriting and tuning of weights, retraining of networks, or using additional circuitry.

To minimize the overhead, we propose stuck weight offsetting, which improves upon the inner fault tolerance method. Inner fault tolerance first identifies all available (non stuck-at-fault) devices and initializes them to default values. Then, the scheme goes through all available devices and adjusts each value such that the represented values cannot be made any closer to the target matrix parameter. Intuitively, this serves to minimize the incorrect contribution of the $R_{\text{ON}}/R_{\text{OFF}}$ weight. We propose to bypass the initialization of available devices to default values and to focus on the complementary weight of stuck-at-fault devices only. Before writing any weights to the crossbar, all stuck-at-fault devices are identified. For each stuck-at-fault device, if the complementary weight is not stuck-at-fault, we calculate its complementary weight to minimize the difference between represented value and target value. All calculated values, along with normal weights, are then written onto the crossbar. This modification reduces overhead by two means. First, all crossbar weights are only required to be written once, as opposed to twice in the inner

Table 4.4: Comparison of our baseline software model against SOTA for Seizure Detection using the University of Bonn dataset.

Paper	Pre-processing	Method	Parallelization	Parameters	Accuracy (%)
Ullah *et al.* (2018)	✓	1D-CNN	✗	21,436	99.90
We *et al.* (2018)	✓	1D-CNN	✗	16,778,144	92.00
Abdelhameed *et al.* (2018)	✓	2D-CNN	✗	106,388	98.00
Liu *et al.* (2019)	✓	2D-CNN	✗	N/R*	99.60
Turk *et al.* (2019)	✓	2D-CNN	✗	1,603,080	99.45
Abdelhameed *et al.* (2021)	✓	2D-CNN	✗	10,304,467	**100.00**
Ours	✗	1D-CNN	✓	**10,778**	99.84

*Not reported.

Table 4.5: Comparison against SOTA for Seizure Prediction using the SWEC-ETHZ and CHB-MIT datasets.

Paper	Method	Parallelized	Parameters	Sensitivity (%)	Specificity (%)	Accuracy (%)	FPR†
CHB-MIT							
[309]	2D-CNN	✗	N/R◇	81.20	N/R◇	N/R◇	**0.16**
[319] *	2D-CNN	✗	N/R◇	N/R◇	N/R◇	92.00	N/R◇
[320]	2D-CNN	✗	49,560	82.71	88.21	98.19	N/R◇
[321] *	2D-CNN	✗	N/R◇	88.80	88.60	88.70	N/R◇
[322] *	3D-CNN	✗	28,459,615	96.66	99.14	98.33	N/R◇
[323] *	2D-CNN	✗	9,695,012	84.00	99.00	99.00	0.2
[260]	1D-CNN	✓	105,538	95.55	**99.68**	**99.64**	N/R◇
Ours	1D-CNN	✓	**10,778**	**99.24**	98.68	99.01	0.47
SWEC-ETHZ							
[324] *	Ensemble HD	✗	N/R◇	96.38	97.31	96.85	N/R◇
[260]	1D-CNN	✓	105,538	94.57	**99.86**	**99.81**	N/R◇
Ours	1D-CNN	✓	**10,778**	**98.22**	97.02	97.54	**0.99**

*Indicates the results are reported across the entire dataset and patient-wise performance was not reported. †False positive rate (per hour). ◇Not reported.

fault tolerance method (from default to adjusted). Second, our method focuses on complementary weights for stuck-at-fault devices only, as opposed to all available devices for all target parameters. This method incurs minimum additional computational cost, and does not require retraining.

4.5.6 Quantization Aware Training for Lower Resolution Systems

A high resolution system is often not feasible to deploy on edge devices, given power consumption constraints and sampling frequency requirements, which are fundamental tradeoffs for resolution in DACs and ADCs. However, lower resolution systems with improved power and frequency performance can exhibit performance degradation. This effect was observed for some patients, and more details can be found in Section 4.6.3. For significant performance degradation (a degradation

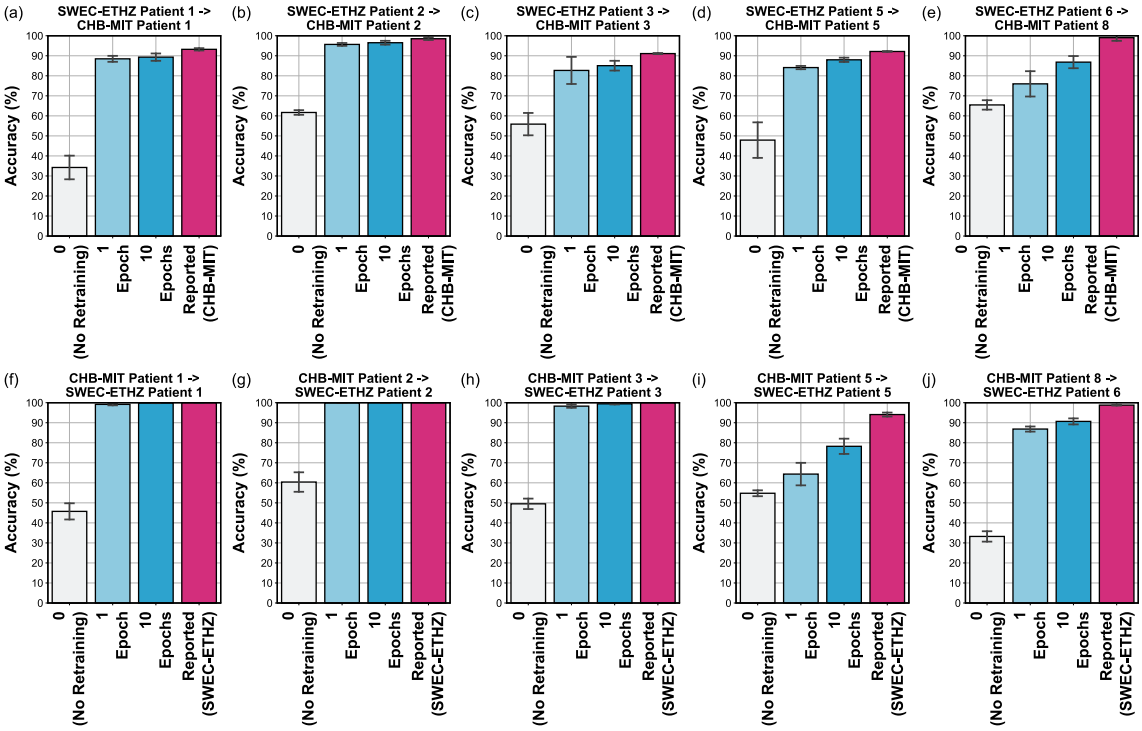


Figure 4.7: The ability of our trained networks to generalize between different datasets when performing epileptic seizure prediction. The cross validation accuracy is reported for networks which have not been retrained, and for networks that have been retrained after 1 and 10 training epochs, respectively, when transfer learning was performed. In addition, the standard evaluation accuracy is reported for each dataset and patient, to facilitate comparisons.

of 5% or more compared to full resolution system), we propose to perform Quantization Aware Training (QAT) prior to mapping the weights onto memristive crossbar arrays [325]. During QAT, we quantized the convolutional and fully connected layers of the network to the resolution equivalent to or even lower than that of the resolution of the crossbar weights and ADC/DAC resolution. Quantized layers are implemented using the Brevitas library [325], which provides PyTorch-compatible convolution and fully connected layers of specified weight resolutions. In addition, inputs to the network were quantized, while intermediate outputs remained not quantized. Network architecture and other training parameters remained unchanged.

4.6 Results and Discussion

Prior to the investigation of device and crossbar non-idealities, we report baseline software results for epileptic seizure detection and prediction using our network architecture, in Table 4.3. 5-fold cross-validation was performed using a different seed to eliminate bias on the first fold. To demonstrate the generalizability of the designed network to different domains and patients, the

same architecture was applied for seizure detection and prediction. Unlike the Bonn dataset, both the CHB-MIT and SWEC-ETHZ datasets are multi-channel EEG datasets with larger memory and computation requirements within the time domain. In order to reduce the time and memory complexity, pre-processing steps as described in Section 4.4.1 were applied to transform the dataset into frequency domain. The shown results suggest that the proposed network is sufficient and can generalize well for both detection and prediction.

4.6.1 Comparisons Against SOTA Software Implementations

In Tables 4.4 and 4.5, we compare our baseline software implementations that use full precision (32-bit) floating-point parameters against other software implementations in literature for epileptic seizure detection and prediction, respectively. As shown in the Tables, for epileptic seizure detection we achieve SOTA performance in 3/4 criteria, while for prediction we obtain SOTA performance in 3/6 criteria. Specifically, for detection, our network architecture is able to achieve an accuracy of 99.84% across all samples without any pre-processing steps, while requiring only 10,778 parameters. This is $\sim$2x fewer parameters than the smallest model in [273], which achieved a slightly higher accuracy of 99.90%, while employing various pre-processing steps. Except for the model used in [326], which achieves a 100% accuracy, but requires over 10M parameters, all the other models shown in Table 4.4, achieve lower accuracy values despite significantly higher number of network parameters.

For epileptic seizure prediction, pre-processing is performed. Across both datasets, our network architecture achieves the highest sensitivity while requiring the fewest number of parameters. We report close specificity and accuracy values to [260], which has also used a 1D-CNN architecture with parallelization, but needs $\sim$10x more parameters. Finally, we report the highest FPR across both datasets, however, unlike previous works, we performed no post-processing steps, which may cause this. Also, only two out of the nine previous works have reported their FPR, which makes the comparison incomplete. When mapping trained parameters to ideal crossbars with fully analog devices without any device or circuit non-idealities, the same results were achieved.

4.6.2 Generalization Between Datasets

To determine whether or not our trained networks have the ability to generalize, we evaluated the performance of networks trained using the CHB-MIT dataset on the SWEC-ETHZ dataset, and vice-versa in Fig. 4.7. In addition, we report the cross validation accuracy for networks which have been retrained using transfer learning. To perform transfer learning, parameters were frozen for all layers except the last two fully connected layers, and the weights and biases of the last two fully connected layers were re-trained using the training set of the evaluation dataset. Direct evaluations to/from either of these datasets and the University of Bonn dataset were not made, as the University of Bonn dataset is used for epileptic seizure detection and not prediction, and it is structured differently.

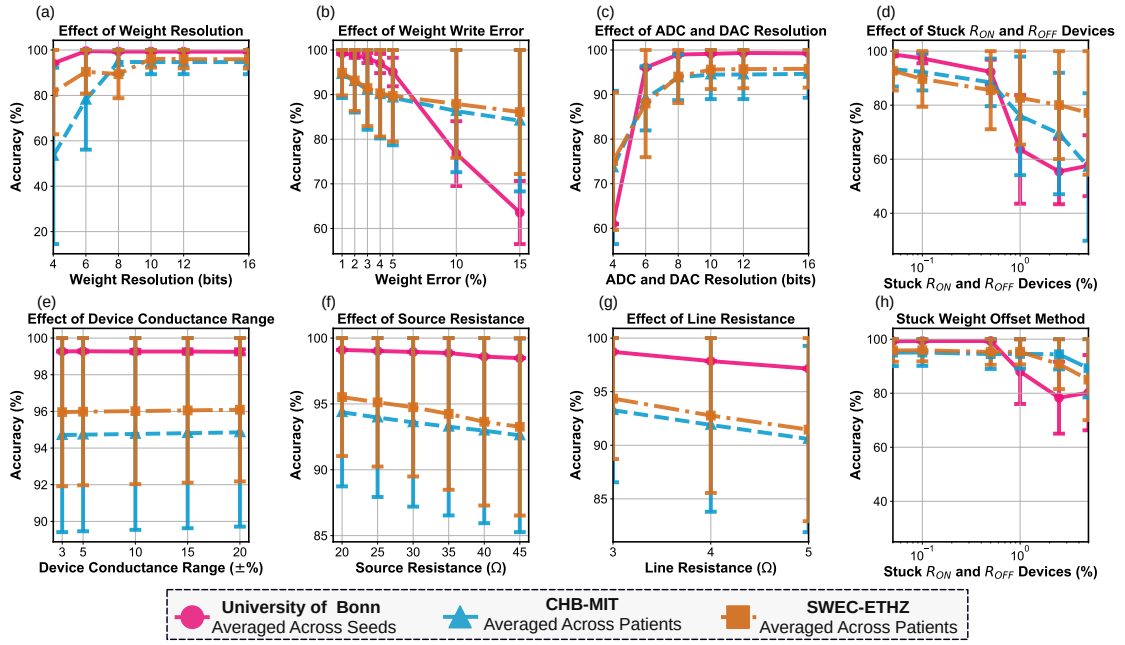


Figure 4.8: The impact of all (a-g) non-idealities on the University of Bonn, CHB-MIT, and SWEC-ETHZ datasets. (h) summarizes performance recovery by applying our proposed stuck weight offsetting to address the performance degradation of stuck-at fault devices. For the University of Bonn dataset, each data-point shows the mean and standard deviation across five arbitrary seed values: 5, 6, 7, 8, and 9.

4.6.3 Quantization-Aware Training

To demonstrate the effectiveness of QAT, we evaluated the performance of our network architecture when trained with and without QAT. Comparisons are made in Fig. 4.9. During QAT training, inputs and network weights were reduced to 6-bit resolution, while network architecture and other training parameters were held constant, as described in Fig. 4.2(f). The accuracy, sensitivity, specificity, AUROC, and FPR metrics were all reported and compared. When using 6-bit ADCs and DACs, it can be observed that for all patients and metrics, except for specificity of patient 5 from the CHB-MIT dataset, QAT network yields significant performance improvements.

4.6.4 Effects of Non-Idealities on System Performance

Fig. 4.8 provides a summary of the impact of non-idealities on our system for epileptic detection and prediction. For the University of Bonn dataset, as samples between patients are not explicitly distinguished, the mean and standard deviation of test set accuracy is reported across samples using five arbitrarily chosen seed values. For the CHB-MIT and SWEC-ETHZ datasets, the mean and standard deviation of test set accuracy is reported across samples for the first five viable patients of each dataset, respectively. Across datasets, some patients were observed to be more robust to non-idealities than others. This was observed in our investigations for patients 1, 2, 3 from

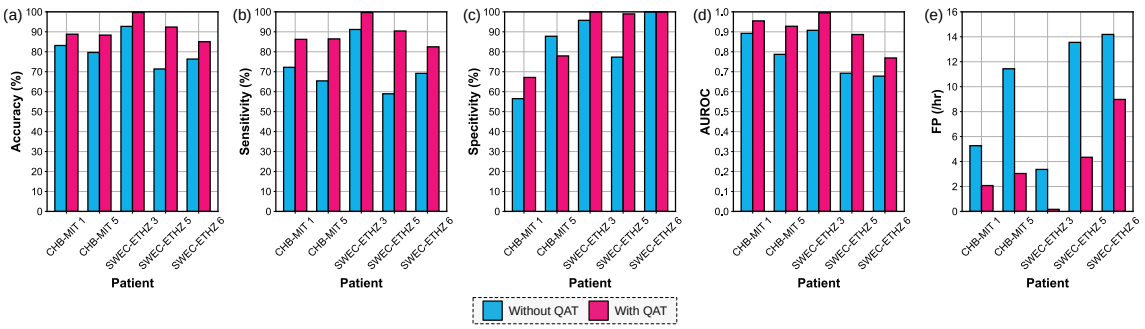


Figure 4.9: The impact of QAT on our network architecture tasked for epileptic seizure prediction (a-e) evaluated using the CHB-MIT and SWEC-ETHZ datasets when network parameters are quantized to 6-bit fixed-point resolution. Only patients that exhibited a degradation of 5% or more when quantized to 6-bit fixed-point resolution (from full-precision floating-point) were investigated.

Table 4.6: Power, area, and latency metrics for the simulated memristive DL accelerator using a 22 nm CMOS process. Using our TDM architecture, VMMs are performed in $\mathcal{O}(n)$, where n is the number of columns of the output vector. Using our parallelized architecture, VMMs are performed in $\mathcal{O}(1)$.

Component	Params.	Time-Division Multiplexing (TDM)						Parallelized					
		Specification	Area (mm^2)	Power (mW)	Latency (us)*	Total Latency (us)	Energy (uJ)	Specification	Area (mm^2)	Power (mW)	Latency (us)*	Total Latency (us)	Energy (uJ)
DAC	Resolution Number	6 bits 7x64	2.58E+01	2.69E+03	8.00E-04	2.15E+00	5.78E+00	6 bits 7x64	2.58E+01	2.69E+03	8.00E-04	3.36E-02	9.03E-02
ADC	Resolution Number Frequency	6 bits 7 10MHz	4.62E+00	7.00E+01	1.00E-01	2.69E+02	1.88E+01	6 bits 7x64 10MHz	2.96E+02	4.48E+03	1.00E-01	6.00E-01	2.69E+00
ReLU	Number	2	9.60E-03	3.28E-02	9.80E-02	9.80E-02	3.22E-06	2	9.60E-03	3.28E-02	9.80E-02	9.80E-02	3.22E-06
Average Pool	Number	1	3.83E-04	1.59E+00	8.49E-05	8.49E-05	1.35E-07	1	3.83E-04	1.59E+00	8.49E-05	8.49E-05	1.35E-07
Adder	Number	10	5.34E-03	1.74E-02	3.06E-04	6.13E-04	1.06E-08	10	5.34E-03	1.74E-02	3.06E-04	6.13E-04	1.06E-08
Subtractor	Number	7	2.46E-04	2.87E-01	3.34E-04	1.28E-01	3.69E-05	7x32	7.88E-03	9.20E+00	3.34E-04	2.01E-03	1.85E-05
S+H†	Number	7x64	8.98E-06	3.81E-03	8.33E-04	5.00E-03	1.90E-08	7x64	8.98E-06	3.81E-03	8.33E-04	5.00E-03	1.90E-08
eDRAM Buffer	Size Bus Width	2KB 128	4.72E-03	1.81E+01	1.15E-04	2.30E-04	4.17E-06	2KB 128	4.72E-03	1.81E+01	1.15E-04	2.30E-04	4.17E-06
eDRAM-Tile Bus	Number	192	4.50E-03	3.5E+00	9.02E-05	9.02E-05	3.16E-07	192	4.50E-03	3.5E+00	9.02E-05	9.02E-05	3.16E-07
IR†	Size	1KB	8.10E-01	6.74E-01	8.21E-05	1.64E-04	1.11E-07	1KB	8.10E-01	6.74E-01	8.21E-05	1.64E-04	1.11E-07
OR†	Size	512B	8.70E-04	4.18E-01	8.21E-05	1.64E-04	6.87E-08	512B	8.70E-04	4.18E-01	8.21E-05	1.64E-04	6.87E-08
Scenario: R_{ON}^{-}													
Crossbar	Number Size Bits per cell	7 64x64 32	2.87E-04	8.67E+00	2.03E-03	5.82E+01	5.06E-01	7 64x64 32	2.87E-04	8.69E+00	2.03E-03	1.30E-01	1.13E-03
Total			**3.13E+01**	**2.79E+03**		**3.29E+02**	**9.19E+02**		**3.22E+02**	**7.21E+03**		**8.70E-01**	**6.27E+00**
Scenario: $(R_{\text{ON}}^{-} + R_{\text{OFF}}^{-})/2$													
Crossbar	Number Size Bits per cell	7 64x64 32	2.87E-04	4.35E+00	6.07E-03	1.74E+02	7.58E-01	7 64x64 32	2.87E-04	4.35E+00	6.07E-03	3.88E-01	1.69E-03
Total			**3.13E+01**	**2.79E+03**		**4.45E+02**	**1.24E+03**		**3.22E+02**	**7.21E+03**		**1.13E+00**	**8.12E+00**

*The latency is listed as individual element. †S+H = Sample and Hold, IR = Input Register, OR = Output Register.

the SWEC-ETHZ dataset, and patient 2 from the CHB-MIT dataset, for which non-idealities have minimal impact. For the rest of the patients, however, no clear pattern was established with regards to robustness against non-idealities. We attribute the varying degree of effectiveness between patients to underlying patient specific signatures.

4.6.5 Stuck Weight Offsetting

As observed in Fig. 4.8(d), stuck $R_{\text{ON}}/R_{\text{OFF}}$ devices lead to severe performance degradation. At 1% stuck-at fault and above, system performance can drop below 50% accuracy, rendering the system ineffective. In response to such degradation, we apply our proposed simplified stuck weight offsetting method. Comparing Fig. 4.8(h) against (d), it is evident that the stuck weight offsetting method improves the average accuracy across all stuck device percentages and datasets. At 1% stuck-at fault, the average accuracy improved by as much as 20% for the Bonn dataset and more than 10% for SWEC-ETHZ and CHB-MIT. The largest improvement was found for the CHB-MIT dataset at 5% stuck-at fault, improving accuracy by 32.11%. At higher stuck device percentages, reduced accuracy recovery is observed. This can be explained by the fact that at higher stuck device percentages, more network information cannot be recovered. Minimizing the contribution of stuck weight cannot fully retrieve the missing information, thereby leading to reduced accuracy recovery. In addition, the proposed method greatly reduces the standard deviation across patients and seeds, thanks to reduced contribution of stuck $R_{\text{ON}}/R_{\text{OFF}}$ devices to final output.

The limitation of this method lies within its inability to deal with both elements of the complementary weight being stuck R_{ON} and R_{OFF} simultaneously. If a positive (negative) weight is stuck R_{ON} and negative (positive) weight is stuck R_{OFF}, stuck weight offsetting cannot provide any further adjustment to minimize the error. Meanwhile, if both weights are stuck R_{ON} or R_{OFF}, the lost weights cannot be recovered, contributing nothing to the final output.

4.6.6 Power, Area, and Latency Requirements

The following assumptions, all supported by SOTA DL accelerators, are made when estimating the power, area and latency requirements of our proposed memristive DL accelerator depicted in Fig. 4.3, targeting a 22nm CMOS process with device integration at the Back-End-Of-The-Line (BEOL). A memristive device has a fixed area of 100×100 nm^2 [335, 336] and the device read latency is 6 ns [208]. An ADC operating frequency is 10 MHz [208], with a power consumption of 10 mW [208] and a device area of 1.1×0.6 mm^2 [336, 337]. A DAC operating frequency is 1.25 GHz, with per unit power consumption of 6 mW and a device area of 0.0576 mm^2 [338]. Other peripheral circuitry with different purposes, including the activation function [339], average pooling layer made up from 4-to-1 multiplexers [340, 341], Sample and Hold (S+H) [342], subtractor [343], and adder [344] circuits, were listed with more detail in Table 4.6.

All the peripheral components are scaled to 22nm technology by factors introduced in [345] and all buffers with their associated connections have energy, area and latency estimated by CACTI 7.0 [346]. For all calculations, the source resistance and line resistance of 20 Ω and 2 Ω are used respectively. To account for RC delays within crossbars when signals are propagated, the methodology presented in [347] was used, with C_{SA}, $T_{settling}$, and C_{write} parameters from [348]. The largest total device latency was used for all devices.

In Table 4.6, four scenarios are considered: two where the resistance of all active (utilized)

Table 4.7: Performance summary and comparison of our simulated system and existing seizure detection/prediction system implementations in the literature.

Paper	Technology	Algorithm(s)	No. Channels	Analog Front-End*	Feature Extract.†	Area (mm^2)	Latency (s)	Power (mW)	Energy (uJ)	Pred.◇	Eval. Task(s)
ML-Based											
[199]	CMOS (180nm)	BPF, LSVM	8	✓	✓	25.00	2.00	N/R°	N/R°	✗	CHB-MIT
[327]	CMOS (180nm)	BPF, NL−SVM	8	✓	✓	25.00	2.00	N/R°	N/R°	✗	CHB-MIT
[328]	CMOS (130nm)	NL−SVM	18	✗	✓	N/R°	4.80	N/R°	N/R°	✗	CHB-MIT
[329]	CMOS (180nm)	FFT, ApEn, LLS	8	✓	✓	13.47	0.8	2.80	2.24E +03	✗	In Vivo
[198]	CMOS (180nm)	BPF, D^2A−LSVM	16	✓	✓	25.0	1.0	N/R°	N/R°	✗	CHB-MIT
[330]	CMOS (180nm)	BPF, NL−SVM	8	✓	✓	25.0	2.0	0.23	460.00	✗	CHB-MIT
[289]	CMOS (130nm)	FIR, PLV	64	✓	✓	3.86	N/R°	1.07	N/R°	✓	In Vivo
[123]	CMOS (130nm)	FIR, PLV/ SE/CFC	32	✓	✓	7.59	0.25	0.71	177.50	✓	In Vivo
[?]	CMOS (180nm)	DWT, KDE, SVM	8	✓	✓	5.83	N/R°	0.67	N/R°	✓	CHB-MIT
[331]	CMOS (40nm)	FFT, NL−SVM	14	✗	✓	4.50	0.71	1.90	1.35E +03	✗	CHB-MIT
[332]	CMOS (65nm)	CHT, XGBoost−DT	16	✓	✓	0.38	N/R°	0.40	N/R°	✗	CHB-MIT, iEEG.org
[234]	CMOS (180nm)	FFT	1	✓	✓	N/R°	N/R°	✗.89	N/R°	✗	CHB-MIT
[?]	CMOS (90nm)	ICA	8	✗	✓	0.4	0.1	8.16E -02	8.16	✗	In Vivo
[311]	CMOS (180nm)	LLS	1	✓	✓	10.41	0.72	2.86E -02	20.59	✗	In Vivo
DL-Based											
[139]	CMOS (65nm)	RNN	8	✗	✗	10.15	N/R°	1✗.80	N/R°	✗	N/R°
[333]	FPGA (M2GL 025-VF256)	MLP	1	✗	✓	N/R°	N/R°	159.70	N/R°	✗	Bonn
[334]	CMOS (180nm)	SNN	1	✗	✓	0.15	64.98E -03	5.40E -03	0.35	✓	In Vivo
Ours (TDM)	**CMOS (22nm)/ RRAM (BEOL)**	**Manual feature extraction, CNN**	**22**	✗	✗	**31.25**	**4.45E -04**	**2.79E +03**	**1.24E +03**	✓	**Bonn, CHB-MIT, ETHZ-SWEC**
Ours (Par.)						**322.31**	**1.13E -06**	**7.20E +03**	**8.12**		

*Reported power, area, and latency requirements include the analog front end/signal acquisition component. †Reported power, area, and latency requirements include feature extraction component(s). ◇Denotes whether systems are able to perform epileptic detection and/or prediction. °Not reported.

devices was fixed to $\bar{R_{\text{ON}}} \approx 10\ k\Omega$, while considering either TDM or parallel use of ADC, and two where the average resistance of all active devices was assumed to be $(\bar{R_{\text{ON}}} + \bar{R_{\text{OFF}}})/2 \approx 55\ k\Omega$, again for either TDM or parallelized ADC. These resistance values are representative of two weight distributions: uniform, where all weights are zero, and normal, where all weights are centered around zero. The first distribution was used to report the maximum possible power consumption of our system, and the second distribution was used to report the power consumption of a typical CNN trained using L2-regularization. Considering the marginal impact on total power consumption, (0.16% and 0.06% for TDM and parallelized configurations, respectively), the power of each individual trained CNN was not determined or reported.

For all scenarios, constant operation at 0.3V per cell [313] was assumed. Neither RRAM crossbar tiles nor peripheral circuitry was assumed to be stacked vertically. Consequently, the circuit area consumption was computed as the summation of all individual elements. Both ADCs and

DACs were assumed to operate at 6-bit resolution, as stated in Section 4.6.3, for the best performance with QAT.

As can be observed in Table 4.6, TDM implementations consume significantly less power than parallelized implementations due to the smaller number of required ADCs. For the worst case TDM scenario, i.e, when all active devices are programmed to R_{ON}^{-} with a constant 0.3V read voltage, our proposed memristive DL accelerator has a latency of 445.22 μs, and consumes approximately 2.79W and 31.255 mm^2 of power and area. This is fairly low power consumption for a DL accelerator to reside on a separate chip from the neural implant, whereby the implant uses thermal energy to wirelessly communicate with the accelerator [349], for reduced latency.

It is noted that we have chosen to optimize the latency of our system at the cost of higher power consumption for multiple reasons. Firstly, analog crossbars which are used to perform IMC operations, in particular VMMs, require peripheral circuitry which is power- and area-hungry. Consequently, independent of the latency of the system, when inference is being performed, a large proportion of the total system's area and power is consumed by peripheral circuitry, registers, and buffers. While TDM ADCs can be used to reduce the total power consumption by increasing latency, other peripheral circuits, registers, and buffers, are still required for operation. Counterintuitively, in certain instances, the energy of the system can be reduced by minimizing system latency during active operation. In other instances, the performance of the system can greatly be improved at the cost of increased power consumption.

Secondly, RRAM devices suffer from conductance drift induced by read disturbances, which may aggregate, as the analog current is summed up along each WL during inference [313]. To mitigate this behavior, we have constrained the absolute amplitude of BL voltages to 0.3V and minimized the duration in which a voltage is applied to each device, i.e., latency is minimized to avoid read disturbances, and to prolong the lifespan of RRAM devices, at the cost of increased power consumption. Lastly, as RRAM devices are non-volatile, gating circuitry can be used to reduce the energy consumption of both TDM and parallelized architectures, as both of our architectures have a critical delay path which is much shorter than typical signal acquisition sampling rate periods. This also allows for input buffering to be performed, so that constant operation is not required.

4.6.7 Comparison to Existing Hardware Implementations

In Table 4.7, we compare the performance of hardware implementations of notable epileptic seizure detection and/or prediction hardware systems in the literature. As many different evaluation tasks were used, we did not report performance metrics. Hardware implementations are broadly categorized as either ML- or DL-based. As can be observed, both of our implementations (reported for the $(R_{\text{ON}}^{-} + R_{\text{OFF}}^{-})/2$ scenario in Table 4.6) have significantly reduced inference latency, at the cost of higher power consumption, compared to traditional CMOS and FPGA-based implementations. It is worth noting that, most of the previous designs have not reported a com-

plete power consumption analysis, are not capable of seizure prediction, and use fewer channels, which can lead to lower power consumption and silicon area.

While, compared to prior work, our proposed system is competitive in resource-constrained environments, it consumes a relatively large area and more power. This results in significantly higher power consumption compared to neuromorphic processors, such as that presented in [334]. It is noted that our design is primarily intended to be used as a reference design for future works implementing epileptic seizure detection and prediction systems using CMOS and memristors. Using analog SRAM, vertical stacking of crossbars and CMOS components, and partial sensing approaches, the power and area requirements of our simulated system could be greatly reduced. We aim to investigate these in our future research.

4.7 Conclusion

We proposed a parallel CNN architecture that can be used to perform both epileptic seizure detection and prediction rapidly. Compared to other works in literate, our architecture requires significantly fewer parameters, and demonstrates competitive performance on the University of Bonn, CHB-MIT, and SWEC-ETHZ datasets. Using emerging memristive devices and software-hardware optimization methodologies, we demonstrated, through comprehensive simulations, that our memristive DL accelerator is capable of performing real-time operation, and consuming reasonable power in real-world conditions. We also proposed and investigated a new simplified stuck weight offsetting method to improve the robustness of our system to non-idealities. This Chapter sets a clear path towards the eventual circuit-level realization of a memristive epileptic seizure detection and prediction system.

Chapter 5

Memristive Stochastic Computing for Deep Learning Parameter Optimization

Stochastic Computing (SC) is a computing paradigm that allows for the low-cost and low-power computation of various arithmetic operations using stochastic bit streams and digital logic. In contrast to conventional representation schemes used within the binary domain, the sequence of bit streams in the stochastic domain is inconsequential, and computation is usually non-deterministic. In this Chapter, the first research question is addressed. The stochasticity during switching of probabilistic Conductive Bridging Random-Access Memory (CBRAM) devices is exploited to efficiently generate stochastic bit streams in order to perform Deep Learning (DL) parameter optimization, reducing the size of Multiply and Accumulate (MAC) units by 5 orders of magnitude. It is demonstrated that in using a 40-nm Complementary Metal–Oxide–Semiconductor (CMOS) process, the proposed architecture occupies 1.55mm^2 and consumes approximately 167μW when optimizing parameters of a Convolutional Neural Network (CNN) while it is being trained for a character recognition task, observing no notable reduction in accuracy post-training.

5.1 Introduction

Embedded RRAM-based neuromorphic and DL accelerators have attracted significant attention due to their promise to revolutionize computing [145]. Such devices are capable of in-memory computation, and can be used to perform near-sensor high-speed and low-power computation at the IoT edge [350].

Current research efforts are primarily focused towards the realization of scalable and reliable memristive architectures for in-memory computing applications [42, 145]. For example, memristive crossbar architectures can be used to efficiently implement MAC or dot-product accelerators to perform 2D VMMs, which are prominent in both neuromorphic and DL systems, in $\mathcal{O}(1)$ [8]. Resistive memories, however, are still considered an emerging technology [351] that are prone to device-to-device variability, endurance challenges, and stochastic behavior [256], which make reaching the maximum gain of RRAM technology, currently infeasible.

While the commercial long-term viability of using RRAM devices for such purposes is yet to

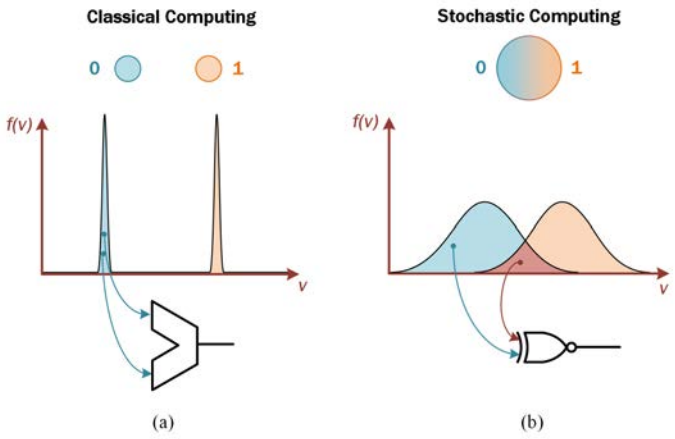


Figure 5.1: Conceptual difference between classical and stochastic computing in terms of probability density functions. (a) Classical or deterministic computing requires well-defined margins. (b) Stochastic computing is error-tolerant, and can thus exploit quantum models of uncertainty that occur at the particle level. This simplifies the hardware required for downstream processing. In our case, a large arithmetic logic unit containing a multiplier can be replaced by a single logic gate.

be properly determined [352], rather than treating the non-ideal stochastic behavior of RRAM devices as a hindrance, some researchers have sought to exploit the well-characterized stochasticity of RRAM devices for alternative applications such as chaos [353,354] and random number generation [101,355]. Traditionally, large hardware costs associated with stochastic number generation have hindered stochastic processors, as they require large bit stream lengths to mitigate undesirable computational errors [356], rendering them largely ineffective for most applications. In [357], a hybrid CMOS-memristor stochastic processor was proposed, which used digital CBRAM devices for efficient stochastic number generation. It was demonstrated that when using large bit-stream lengths, the processor was able to perform gradient descent optimization and k-means clustering in a low-power and high-speed mode of operation.

In this Chapter, we expand upon efforts in [357], and exploit the well characterized switching stochasticity of probabilistic CBRAM devices to efficiently generate stochastic bit streams in order to perform deep learning parameter optimization using a hybrid CMOS-memristor stochastic processor. By nature, such a processor is highly tolerant to external noise and relaxes many of the stringent hardware requirements needed in generating distinct voltage levels (Fig. 5.1). While most prior DL-based SC research applies SC to the feed-forward processing (inference) stage, as it is known to be tolerant to noise, we instead explore if it is at all possible to perform parameter optimization, that typically requires high precision, using probabilistic bits. Our specific contributions are as follows:

1. We are the first to exploit the switching stochasticity of CBRAM devices to perform deep learning parameter optimization using SC;

2. We evaluate our architecture by training a DNN for the MNIST character recognition task using Mini-Batch SGD and Mini-Batch SGD with Momentum;

3. We investigate the tradeoff between latency, area and power consumption, and demonstrate that our architecture can be used to reduce the size of MAC units.

5.2 Preliminaries

5.2.1 Stochastic Computing

In SC, numbers are represented using bit streams. The frequency of 0s and 1s in stochastic bit streams determines their value depending on a range, R, denoted as the priori. The priori can be unipolar, i.e., $R \in [0, 1]$ to represent unsigned operands, or bipolar to represent signed operands, i.e., $R \in [-1, 1]$. The value represented by unipolar bit streams can be determined using the mean, x, whereas the value represented by bipolar bit streams can be determined using $2x - 1$. Conventional SC circuits use Linear-Feedback Shift Registers (LFSRs) to generate pseudo-random numbers for stochastic bit stream generation, and popcount and up-down counter circuits to decode bit streams. SC blocks for bit stream generation, addition, multiplication, and stochastic bit stream to binary converters are depicted in Fig. 5.2.

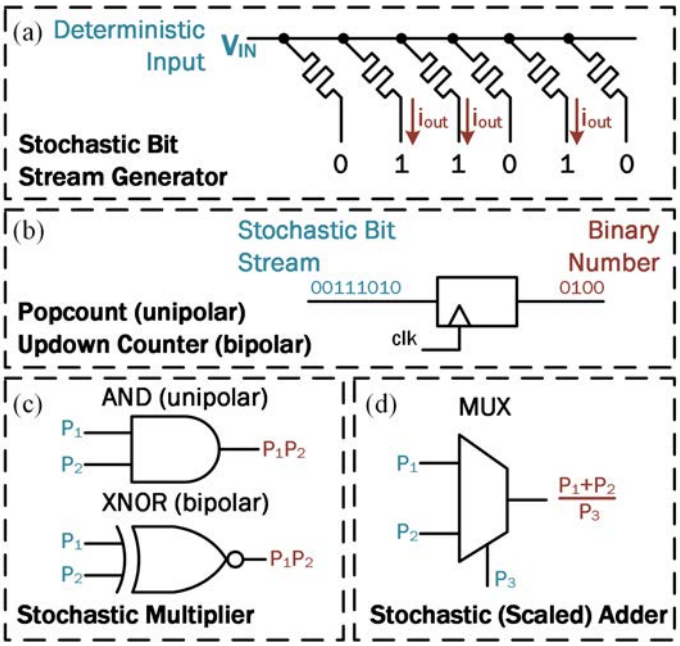


Figure 5.2: Stochastic computing blocks for (a) bit stream generation, (b) stochastic bit stream to binary converter, (c) multiplication, and (d) scaled addition.

5.2.2 Memristor-based Native Stochastic Computing

Digital memristor devices have a large dynamic range between logic levels that is associated with the formation and rupture of a dominant, nanoscale conducting filament [24]. On account of device-device variation, experimental studies have demonstrated that the switching time between logic levels is stochastic, and that the time-to-switch can be accurately modelled using a Poisson distribution [224, 358], as depicted in Fig. 5.3. By exploiting this property, both unipolar and bipolar stochastic bit streams can be efficiently generated using digital memristors by applying programming pulses with variable pulse widths to memristor cells. Memristor-based native SC systems can be realized by using memristor cells for stochastic bit stream generation, and using CMOS for stochastic arithmetic circuits.

5.2.3 Deep Learning Parameter Optimization

SGD [359] is a parameter optimization method that is described by (5.1), which is commonly used to train DNNs.

$$\theta_n = \theta_{n-1} - \eta \nabla_\theta J(\theta, y_i', y_i), \tag{5.1}$$

where, η is the learning rate, θ denotes a trainable parameter, y_i represents the class label, y_i' denotes the predicted class label, and $J(\theta, y_i', y_i)$ describes the objective function. Momentum [360], or SGD with momentum, described by (5.2), is a method that improves SGD and accelerates gradients in relevant directions towards a local minima by dampening oscillations using γ and v, i.e.,

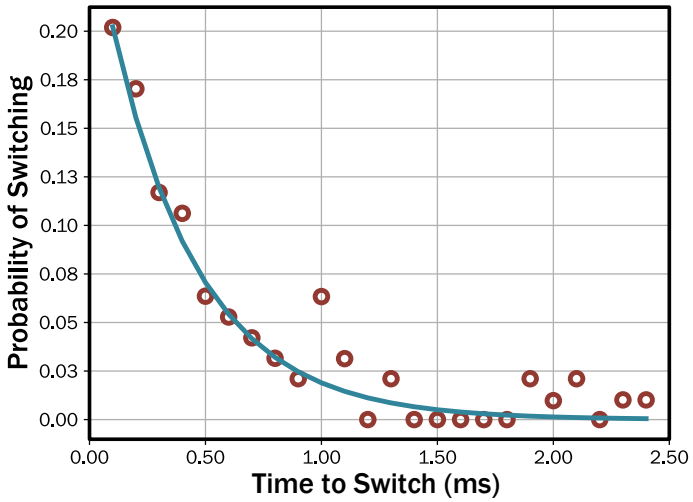


Figure 5.3: Distribution of switching time of a fabricated CBRAM device [24] under an applied voltage of 4.5 V. Switching events (red circles) fit a Poisson distribution (blue line), $P(t) = (\Delta t/\tau)e^{-t/\tau}$ for Δt = 0.5, $V = 0.4$ and $\tau = 0.38$ms.

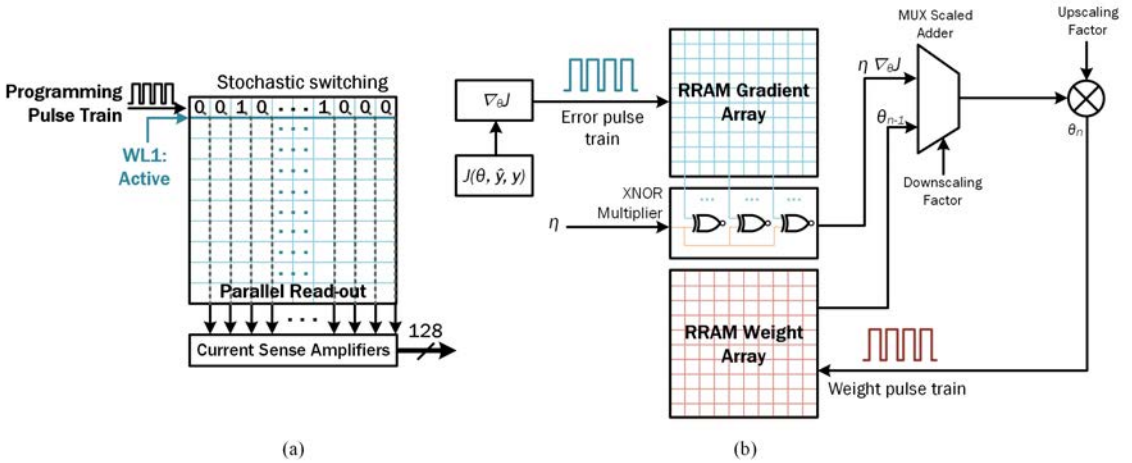


Figure 5.4: Native stochastic parameter update process. (a) The word line (WL) is activated to open one row at a time. A pulse train is applied to induce stochastic switching in the RRAM. The current sense amplifiers are used to sense which devices are drawing current, indicating which cells have been switched on. (b) The parallel-generated bit stream is XNOR multiplied by the negative learning rate and added to the previous weights to perform a single parameter update. The overhead of requiring an additional array to compute the weight bit stream is partially offset by removing the need for ADCs.

momentum and velocity parameters.

$$\begin{aligned} v_n &= \gamma v_{n-1} + \eta \nabla_\theta J(\theta, y_i', y_i), \\ \theta_n &= \theta_{n-1} - v_n \end{aligned} \tag{5.2}$$

Mini-Batch SGD and Mini-Batch Momentum are variations of the SGD and Momentum optimization algorithms which split training datasets into small batches that are used to perform parameter optimization. From this stage forward, we implicitly refer to mini-batch variations.

Table 5.1: Adopted network architecture.

Layer	**Output Shape**
Convolutional, $f = 10, k = 5, s = 1, p = 0$	$(10 \times 24 \times 24)$
Max Pooling, $k = 2, p = 2$	$(10 \times 12 \times 12)$
Convolutional, $f = 20, k = 5, s = 1, p = 0$	$(20 \times 8 \times 8)$
Max Pooling, $k = 2, p = 2$	$(20 \times 4 \times 4)$
Fully Connected, $N = 50$	(50)
Fully Connected, $N = 10$	(10)

5.3 Proposed Architecture

A block diagram of our proposed architecture is depicted in Fig. 5.4. We confine operating in the stochastic domain to the parameter update stage during training. The architecture consists of scalable crossbar tiles of probabilistic CBRAM devices (RRAM input and weight arrays) and CMOS systolic stochastic arithmetic circuits (multipliers and scaled adders). Stochastic bit streams are generated by writing pulse trains to columns of 2D crossbar architectures. Given an applied voltage, V, and a programming pulse width, t, the switching probability can be expressed using (5.3)

$$P(t, V) = 1 - e^{(-te^{V/V_0})/\tau_0}, \tag{5.3}$$

where V_0 and τ_0 are fitting parameters [357]. By fixing the applied voltage and varying the programming pulse width applied to each column, bipolar bit streams can be efficiently generated in-memory. The bits are read out from the array in parallel using current sense amplifiers (Fig. 5.4(a)). Depending on task-specific requirements, crossbar tiles can either be duplicated to increase throughput and to reduce latency, or time-multiplexing can be used to decrease power and area.

5.4 Training and Validation Methodologies

To evaluate our architecture, we trained a CNN described in Table 5.1 using MNIST. All convolutional, and the first fully connected layer were sequenced with batch normalization layers, and the ReLU activation function was used for all layers. All networks were trained for 10 epochs with a batch size of 256 and a fixed learning rate. Gradients were clipped between -1.0 and 1.0. Cross entropy loss was used in conjunction with SGD and SGD with momentum. For all networks trained using momentum, Nesterov was disabled, and a fixed momentum value of 0.9 was used, which has demonstrated significant performance for a variety of DL tasks [360].

5.5 Results

5.5.1 Performance

The MNIST test set accuracy for all training epochs, and the MNIST test set accuracy and training loss for all mini-batches during the first training epoch are reported in Fig. 5.5, for network architectures trained using our native SC-based parameter optimization with SGD and SGD with momentum. These are also shown for a baseline implementation using conventional training. The learning rate was varied from 0.01 – 0.5 and the stochastic bit stream length (N_{bit}) was varied from 2-Kbits – 64-Kbits, near the range investigated in [357]. Each training instance was repeated

[0]For each convolutional and pooling layer, f denotes the number of filters, k determines the filter size, s is the stride length, and p denotes the padding. N is the number of output neurons for each fully connected layer.

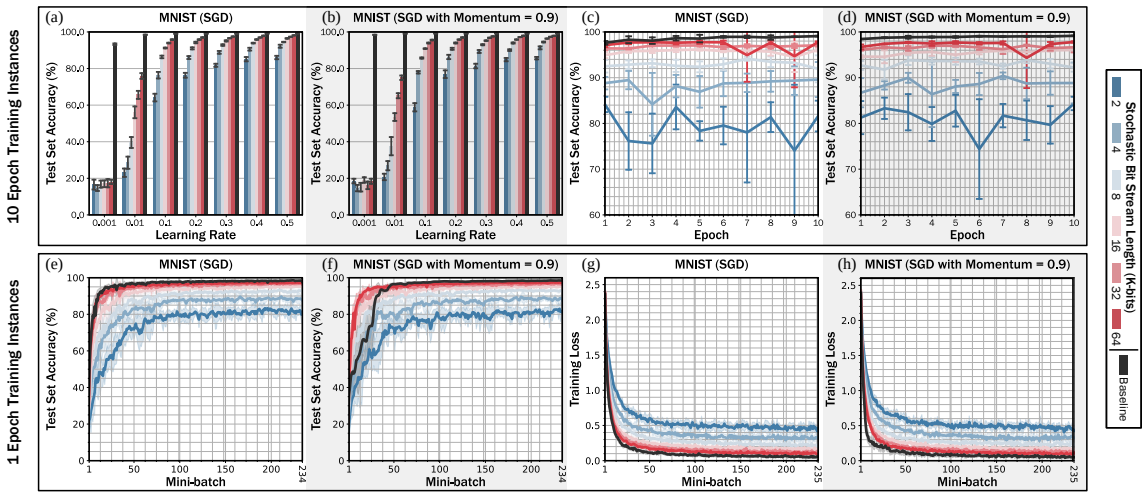


Figure 5.5: The MNIST test set accuracy across seperate training instances of 10 epochs (a–d) and 234 mini-batches (1 epoch) (e–h), respectively, for CNNs trained using our native SC-based parameter optimization with different learning rates for (a) SGD, and (b) SGD with momentum; for a fixed learning rate with (c) SGD, and (d) SGD with momentum; for a fixed learning rate with (e) SGD, and (f) SGD with momentum. The training loss during the first training epoch for (g) SGD, and (h) SGD with momentum. In (c–h) a learning rate of 0.5 was used. Each baseline implementation uses 32-bit floating point weights.

10 times, and the mean and standard deviation across instances were determined. From Fig. 5.5, it can be observed that for both SGD and SGD with momentum a bit stream length larger than 8-Kbit and a learning rate of $\geq$0.1 is required for stable training performance. We attribute the performance degradation when smaller learning rates are used to the fact that positive values near zero can be encoded negatively in the stochastic domain, and note that such implementations may eventually converge if trained for $\gg$10 epochs.

5.5.2 Power and Area Requirements

Power and area estimates were calculated based on a 40 nm CMOS process integrated with RRAM in the back end of the line [196]. Specifically, a pre-configured IP is used for the encoder and decoder, and a full-custom approach was taken to laying out the RRAM array, sense-amplifiers, registers, MUX-based scaled adder, and XNOR accumulator, with verified DRC/LVS compliance. Power draw is modified to suit the device distribution in Fig. 5.3. The area of each 128$\times$128 RRAM tile is $2.77\times10^3\mu$m^2. Assuming that the entire bit stream must be accessible at once, and that only one row may be read out at a time, a bit stream length of 16-Kbits requires 2^7 tiles. The gradient bit stream is multiplied by a learning rate bit stream of equivalent length and subtracted from the weight. To improve throughput, the number of arrays is doubled to allocate half of the tiles to be reset while the other half are generating bit streams. This gives a total of 2^9=512 RRAM tiles, occupying a total area of 1.42mm^2.

Static power consumption is estimated based on the presumption that half of the tiles are generating bit-streams while the other half are being reset, one row at a time. Balancing the latency between reading and resetting is feasible because reading requires stochastic programming. As a conservative estimate, we assume an input voltage of 4.5V, which is tolerable at 40nm using Laterally-Diffused Metal Oxide Semiconductor (LDMOS) transistors, that draw approximately 10nA from a device that is on, and 100pA from a device that is off [361]. The corresponding static power dissipation is 45nW and 0.45nW, respectively. The power consumed for one read out can be measured by (5.4):

$$P_{\text{read}} = N_{\text{bit}}[P_{\text{on}}\mathbf{E}(\nabla_\theta \mathbf{J}) + P_{\text{off}}(1 - \mathbf{E}(\nabla_\theta \mathbf{J}))], \tag{5.4}$$

where N_{bit} is the bit stream length, P_{on} and P_{off} are the power dissipated from an on and off device, and $\mathbf{E}(\nabla_\theta \mathbf{J})$ is the expected value of the gradient $\nabla_\theta J(\theta, y_i', y_i)$. We measured $\mathbf{E}(\nabla_\theta \mathbf{J})$=0.5367 at the start of the training process. Gradient updates are generally larger at the start of training than at convergence. Therefore, more cells are switched on for steeper gradients. This means that worst-case power consumption should be measured at the start of training. For normalized bipolar weights, $\mathbf{E}(\theta)$=0.5. Power dissipation from the RRAM array is estimated to be 43.0 μW per gradient, and 40.6 μW per weight. These estimates are doubled to account for additional tiles used for resetting and enhancing throughput, giving a total result of 167 μW.

The layout of the peripheral CMOS circuitry was used to generate accurate area and power estimates. The area of a single XNOR gate is 670nm$\times$355nm. The pitch of the RRAM array is approximately 410nm. The XNOR multiplier can be oriented such that it is pitch-matched to the RRAM columns. It can then fit under the bottom row without additional area overhead. The total area occupation of the XNOR gate of 0.031mm^2 can thus be merged with the RRAM area. The total area overhead from accumulation consists of the MUX-based scaled adder and a register which consume 0.0735mm^2. Each sense-amplifier built from a pair of cross-coupled inverters occupy 0.41μm^2. Although the shorter dimension can be optimized to be pitch-matched to the array, a reference signal is required to distinguish on and off cells. This reference must be obtained from an adjacent column, which requires time multiplexing the bit stream generation process into two steps. With pairwise column sharing, the total area of all current sense amplifiers is 0.0267mm^2. Time multiplexing also halves the maximum static power dissipation to 83.5 μW. Total static power dissipation of the CMOS elements is from subthreshold current draw, and thus dominated by resistive dissipation.

5.6 Discussion

The length of the parallel-generated bit stream may cause large power draw as throughput is increased. But this drawback is offset by three factors. Firstly, the MAC process of non-quantized weights can now be completed in one single step using an XNOR gate and a multiplexer (with a

simulated post-layout delay of 58 ps). A 16-bit MAC unit designed in a comparable process [362] is approximately five orders of magnitude larger and two orders slower than an XNOR gate and a MUX. Secondly, the power estimates are derived from the peak value at the end of bit stream generation. All devices are initialized to be off and only switch on during the course of the time to switch (Fig. 5.3). This peak only occurs at the end of the read out process. Finally, sampling inputs and weights from a normal distribution tracks gradient descent pathways that avoid saddle points that riddle high-dimensional problems. While not evident on a task as simple as MNIST, we expect this advantage to manifest on more practical problems with deeper networks.

Numerous options are available to further optimize the power dissipation. Significant resources have been dedicated to developing low-voltage memristors well below the supply limitation of the 40nm process used. At one extreme end, the work in [53] uses 100 mV programming pulses. This would reduce resistive dissipation by a larger factor, though must be balanced with the average switching time. Our work has shown that lengthy bit streams are required to obtain acceptable convergence in the training process. This is on par with literature [357], however, we expect that the use of quantization-aware training can effectively place a constraint upon the set of permissible values, thus reducing the bit length.

Alternatively, the generation of long bit streams can be time multiplexed to free up arrays to increase throughput. Although this slows down bit stream generation, we have eliminated the large delays associated with MAC operations by one order of magnitude [362]. SC also uses noise to avoid being trapped by saturated gradients during training, known to significantly slow down the training process in conventional computing. Endurance concerns can be overcome by substituting RRAM for MRAM, which exhibits endurance of over 10^{15} cycles, compared to that of RRAM ($\approx 10^5 - 10^{12}$ cycles) with faster write times, at the cost of increased fabrication complexity.

5.7 Conclusion

In this Chapter, we demonstrated that it is indeed possible to perform DL parameter optimization using stochastic bits by exploiting the stochasticity during switching of probabilistic CBRAM devices to efficiently generate stochastic bit streams. This new insight to stochastic computing is valuable for the following reasons:

1. For an end-to-end stochastic computing system, the gradient update step can share resources with the feed-forward step; the alternative to long bit-streams would be a floating point unit, which offsets this disadvantage;

2. The multiply-and-accumulate steps now rely only on two combinational logic gates. This means the propagation delay for MAC is reduced by orders of magnitude. Therefore, the number of registers can be significantly reduced by distributing computation over additional cycles, without increasing the total computation time.

While in this Chapter, memristive SC was solely used for DL parameter optimization, SC has been proposed and projected at almost all levels of the computing stack, due to its ability to be used to perform basic arithmetic operations. Practical engineering applications that could make use of the proposed architecture include, but are not limited to image processing, vector quantization, machine control, and error correction decoding. Investigation surrounding these alternative applications, switching probability variation on account of device-to-device variation, and an end-to-end timing analysis at the circuit and system level for a variety of configurations to train more complex networks using larger datasets, forms the basis of future work.

Chapter 6

MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems

Memristive devices have shown great promise to facilitate acceleration and improve the power efficiency of Deep Learning (DL) systems. Crossbar architectures constructed using these Resistive Random-Access Memory (RRAM) devices can be used to efficiently implement various in-memory computing operations, such as Multiply and Accumulate (MAC) and unrolled-convolutions, which are used extensively in Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs). However, memristive devices face concerns of aging and non-idealities, which limit the accuracy, reliability, and robustness of Memristive Deep Learning Systems (MDLS). These limitations should be considered prior to circuit-level realization. This Chapter, which addresses the second research question, presents *MemTorch*, an open-source[1] framework for customized large-scale memristive DL simulations, with a refined focus on the co-simulation of device non-idealities. It is noted that this Chapter arguably presents the most significant contribution of this thesis. MemTorch enabled, and was used to conduct, many studies, which are presented in other Chapters.

6.1 Introduction

Memristive crossbar architectures [363] have been used to reduce the time complexity of VMMs used in DNNs from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, and in extreme cases to $\mathcal{O}(1)$ [11], facilitating the acceleration and improving the power efficiency of DL systems [145]. However, memristors are still considered an emerging technology, where their reliable manufacturing processes are yet to be achieved. As a result, DL architectures realized using memristor crossbars are putative to be prone to severe errors due to a number of device limitations including: finite discrete conductance states, device I/V non-linearity, failure, aging, cycle-to-cycle and device-to-device variability [352, 364]. Consequently, significant research efforts are being made to improve the reliability and robust-

[1] `https://github.com/coreylammie/MemTorch`

ness of memristive, or RRAM crossbars, used to perform *in-situ* learning [4, 147, 223, 365] and inference [11, 45, 150, 223, 225, 366] in DL systems. A general cross-platform, heterogeneous, high-level, customizable and open-source simulation framework with a refined focus on the co-simulation of device non-idealities could be used to conveniently build, rapidly prototype, and investigate device non-idealities in customized large-scale MDNNs and MDLSs. In this Chapter, we present such a framework, entitled *MemTorch*, for deep memristive learning using crossbar architectures. MemTorch is an open-source [367] simulation framework that integrates directly with the open-source PyTorch ML library that:

1. Facilitates the cross-platform development and distribution of large-scale passive 0-Transistor 1-Resistor (0T1R) and active 1T1R memristive deep learning systems;
2. Places a large emphasis on modeling non-ideal, but inevitable, device characteristics in arbitrary and customizable device models;
3. Supports heterogeneous platforms such as CPUs and GPUs;
4. Has a high-level API, which is able to abstract performance-critical tasks described in various low-level languages.

6.2 Related Work

We compare MemTorch to other memristor-based DNN frameworks and inference accelerators, which are software-based and do not rely on SPICE modeling, in Table 6.1. More exhaustive comparisons are performed in [2]. Software-based frameworks and inference accelerators use a combination of programming languages to simulate the behavior of memristive devices. Among previous works, DNN+NeuroSim [237,238] and the IBM Analog Hardware Acceleration Kit [240] are the most similar offerings, which integrate with both PyTorch and/or Tensorflow, and can be used to account for non-ideal device characteristics. However, they are largely concerned with algorithm-to-hardware mapping, and are designed to evaluate training and inference accuracy with hardware constraints. They are not designed to model any arbitrary device non-idealities for any behavioral device model. MemTorch, on the other hand, emphasizes the co-simulation of non-ideal device characteristics and generic behavioral device models with stochastic parameters for higher flexibility to simply account for process variance.

6.3 Software Framework

The MemTorch simulation framework is programmed in C++, CUDA and Python, with a Python interface. Performance critical tasks are performed using either C++ or CUDA, for CPU or GPU execution, respectively; otherwise Python is used. MemTorch relies heavily on the open source

Table 6.1: Comparison of MemTorch to other memristor-based DNN simulation frameworks and inference accelerators.*Does not support GPU-accelerated inference and/or parameter mapping.†Models are shared using Google Drive without Application Programming Interfaces (APIs).

Simulation framework	Open-source	GPU	Pretrained DNN conversion	Programming language(s)
RAPIDNN [368]		✓*	✓	C++
MNSIM [222]			✓	Not Specified
PUMA [150]			✓	C++
DL-RSIM [234]		✓	✓	Python
PipeLayer [223]		✓*	✓	C++
Tiny but Accurate [235]	✓†		✓	MATLAB
Ultra-Efficient Memristor-Based DNN Framework [236]	✓†		✓	C++, MATLAB
Non-ideal Resistive Synaptic Device Characteristic Simulation Framework [224]		✓	✓	Python
Neurosim [139], NeuroSim+ [369], and DNN+NeuroSim [237, 238]	✓	✓	✓	C++, Python
IBM Analog Hardware Acceleration Kit [240]	✓	✓	✓	Python, C++, CUDA
MemTorch	✓	✓	✓	**Python, C++, CUDA**

PyTorch [231] ML framework, and uses the C++ and Python PyTorch APIs extensively to abstract low-level operations. Consequently, it supports native CPU and GPU operations.

6.3.1 Software Architecture

MemTorch is made up of seven distinct sub-modules. General utility functions, such as data loaders or generic functions, are grouped within `memtorch.utils`. The `memtorch.bh` sub-module encapsulates all crossbar models, crossbar mapping and programming methods, crossbar tile mapping and programming methods, memristor models, memristor model window functions, models for all non-ideal device characteristics, quantization methods, and methods to generate stochastic parameters. The `memtorch.mn` sub-module mimics `torch.nn` and defines equivalent memristive`torch.nn.Module` layers. `memtorch.mn` currently extends `torch.nn.Linear`, `torch.nn.Conv1d`, `torch.nn.Conv2d`, and `torch.nn.Conv3d`. `memtorch.mn.Module.patch_model` can be used to either instantiate new layers, or to patch existing instances. `memtorch.mn.Module.patch_model()` iterates through and patches all named modules within classes extending from `torch.nn.Module` and adds a `self.tune_()` method, in addition to other helper methods, to the class instance of the model that automatically patches each selected named module.

The `memtorch.cpp` sub-module encapsulates all Python-wrapped C++ extensions, whereas the `memtorch.cu` sub-module encapsulates all Python-wrapped CUDA extensions. Currently, MemTorch uses C++ and CUDA bindings to perform inference for both active and passive modular tiled architectures, and to parallelize quantization operations. The use of bindings can be disabled, and legacy python methods (developed in previous versions of MemTorch) can be used instead using the `use_bindings` argument, when patching `torch.nn.Module` instances. `memtorch.examples` sub-module encapsulates general-usage examples and supporting scripts. The `memtorch.map` sub-module encapsulates all mapping and tuning algorithms used when programming and tuning memristive crossbar arrays. Finally, the `memtorch.submodule` sub-

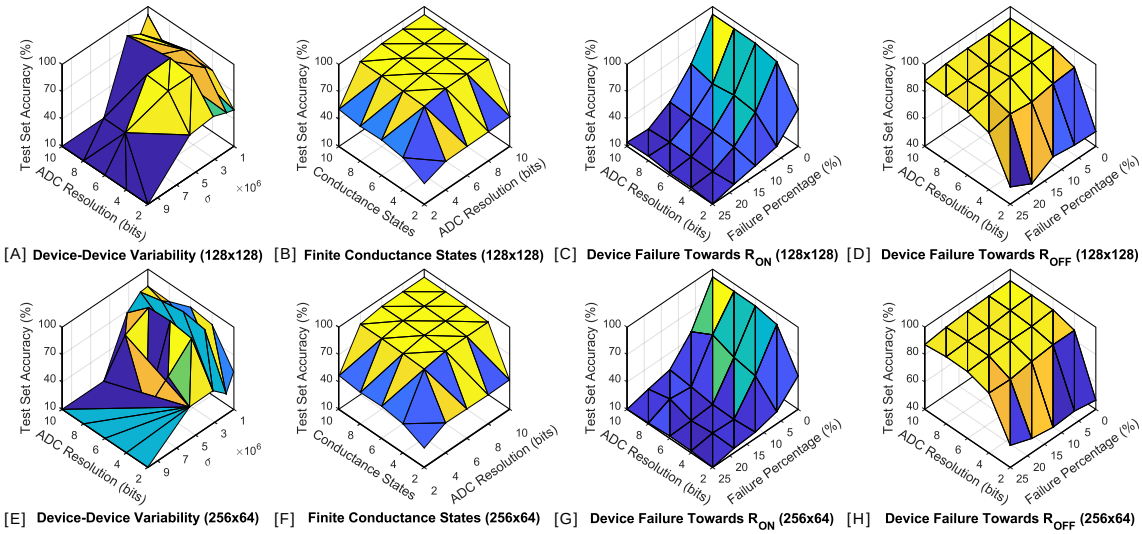


Figure 6.1: Simulation results when exemplar device non-idealities are considered for classifying CIFAR-10 dataset for two different modular crossbar tile sizes, 128x128 and 256x64. While MemTorch can be used to simulate both passive and active architectures, for demonstration purposes, in this figure, only active architectures are considered.

module encapsulates all external git sub-modules that MemTorch uses. We review and present the algorithms and models that are currently built into MemTorch in Section 6.6, and our approach to modeling non-ideal device characteristics in Section 6.7.

6.3.2 Software Functionalities

Complete examples demonstrating the functionality of MemTorch are publicly accessible[2]. ReadTheDocs[3] is used to explain all functionalities.

6.4 Implementation and Empirical Results

In Fig. 6.1, we present exemplar large-scale deep learning simulations to investigate the performance degradation due to device-device variability, finite number of conductance states, and device failure when two different modular crossbar sizes are considered. A separate case study is not presented, as we have previously used MemTorch in other works to perform hand gesture classification [1], epileptic seizure prediction [6], to develop an empirical metal-oxide device endurance and retention model [4], and to develop an extended Design Space Exploration (DSE) methodology for RRAM architectures [5]. Prior to simulation, all convolutional and linear layers from a pre-trained MobileNetV2 for CIFAR-10 were converted to memristive equivalent layers,

[2] https://github.com/coreylammie/MemTorch/tree/master/memtorch/examples
[3] https://memtorch.readthedocs.io/en/latest/

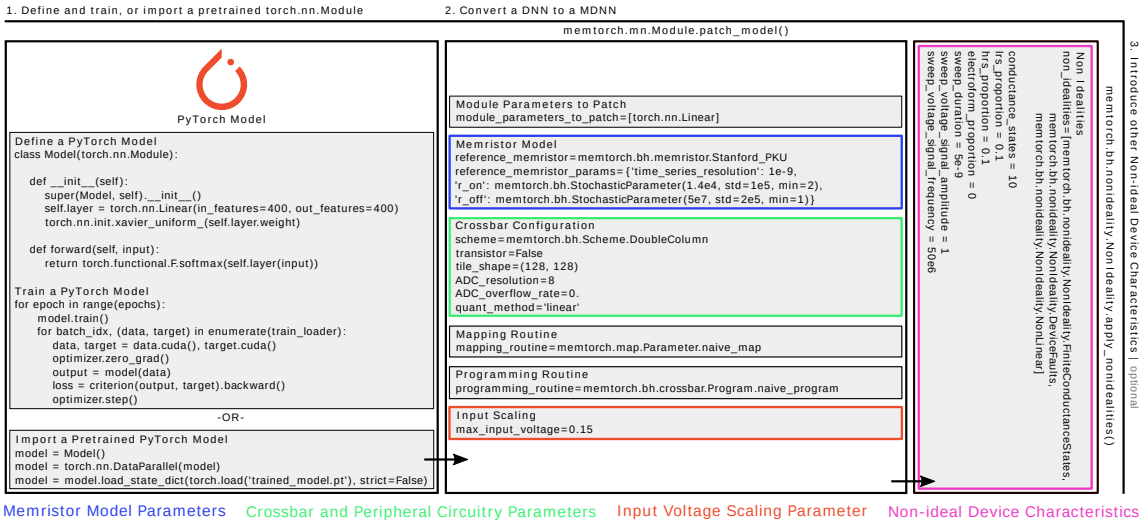


Figure 6.2: Illustration of a typical use-case workflow in MemTorch.

as explained in detail in [4], and documented in Section 6.8.

6.5 Illustrative Example

To demonstrate MemTorch's intuitive design, we depict a typical use-case work flow in Fig. 6.2. Here, `torch.nn.Linear` layers are converted to equivalent memristive layers constructed using modular crossbar tiles, that each contain (128×128) devices, which represent weights using a double-column parameter representation scheme. Inputs are scaled between $\pm 0.15V$, and 8-bit ADCs are used to read out column currents. The Stanford PKU RRAM model [370] is used to model TiN/Hf(Al)O/Hf/TiN devices from [27]. Three other non ideal device characteristics were also accounted for including a finite number (10) of discrete conductance states, device faults, and non-linear I/V device behavior.

6.6 Algorithms and Models

This Section reviews and presents the algorithms and models that are currently built into MemTorch.

6.6.1 Memristive Device Models

Within MemTorch, we use five base memristive device models that extend the `memtorch.bh.Memristor.Memristor` base class for our simulations. These include the linear ion drift model by [371]; the VTEAM model by [151], which is a general model for voltage-controlled

[4] `https://github.com/coreylammie/MemTorch/tree/master/memtorch/examples/Exemplar_Simulations.ipynb`

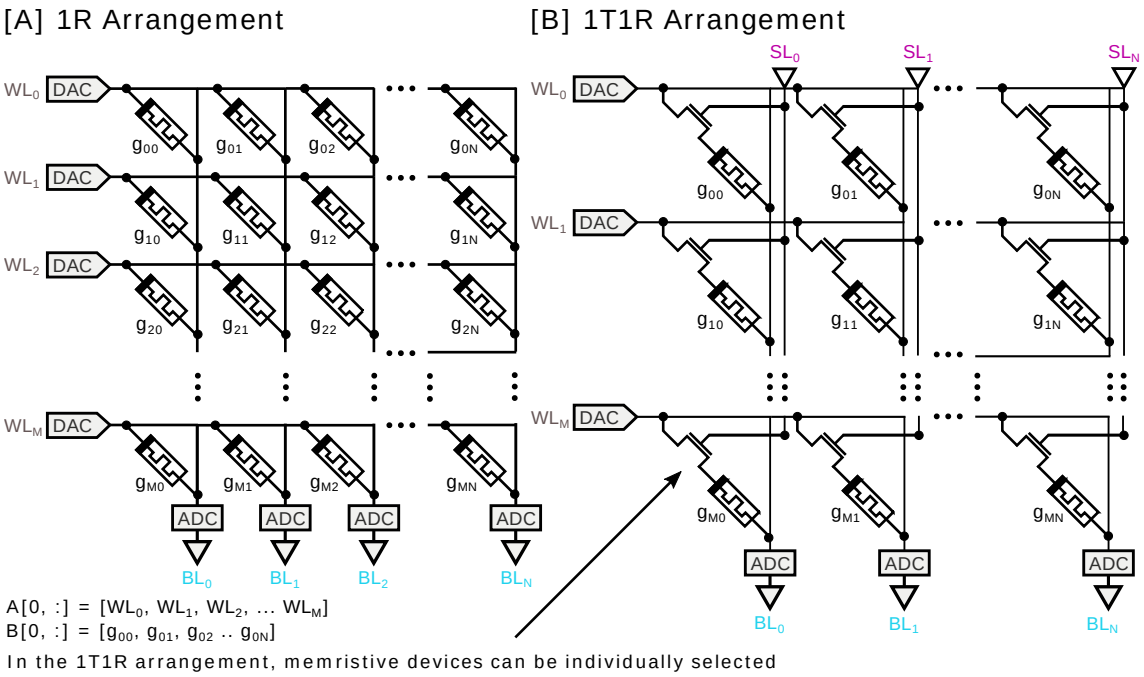


Figure 6.3: Depiction of an $M \times N$ [A] 1R (0T1R) crossbar architecture and a [B] 1T1R crossbar architecture. Matrix-vector and matrix-matrix multiplication can be performed by encoding and presenting a scaled input vector or matrix $\boldsymbol{A}$ as voltage signals to each row of the crossbar's WLs. As shown in [A], assuming a linear I/V relationship, the total current in each column's BL is linearly proportional with the sum of the multiplication of the WL voltages and conductance values in that column, i.e., $\boldsymbol{BL}[0, :] \propto \boldsymbol{A}[0, :] \times \boldsymbol{B}$. In the 1T1R arrangement [B], individual memristive devices can be selected using SLs.

memristors that can be used to fit a large range of experimental device data; the Stanford PKU RRAM model [370], which describes switching performance for bipolar metal oxide RRAM; and two versions of the data-driven Verilog-A RRAM model [372], which expresses device current-voltage characteristics and resistive switching rate as a function of the bias voltage and the initial resistive state of each device. For each base model, finite differences is used to obtain a numerical solution for each discretized time-step, dt. While only five base memristive models are currently supported natively, others, which can model the equivalent conductance of a memristive device for an arbitrary applied voltage signal, such as those modeling PCM or other device technology behavior, can easily be integrated modularly by extending `memtorch.bh.Memristor.Memristor`.

6.6.2 Window Functions

Within memristive device models, window functions are widely employed to restrict the changes of the internal state variables to specified intervals [373]. MemTorch currently natively supports

the Biolek [374], Jogelkar [375], and Prodromakis [376] window functions, and can easily be extended to support others.

6.6.3 Memristive Crossbar Architectures

Memristive devices can be arranged within crossbar architectures to perform VMMs, which are used extensively in forward and backward propagations within DNNs. There are two commonly used crossbar architecture configurations, namely 1T1R, and 1-Resistor (1R or /0T1R), which are both depicted in Fig. 6.3. In 1T1R arrangements, one transistor is used to select and control each memristive device, whereas in 1R arrangements, rows and columns of memristive devices are positioned perpendicular to each other, with memristive devices sandwiched in-between.

The product of a vector and a matrix or, in a more general form, two matrices, $\boldsymbol{A}$ of size $(M \times C)$ and $\boldsymbol{B}$ of size $(C \times N)$, can be computed using a crossbar-architecture, as illustrated in Fig. 6.3, where $\boldsymbol{A}$ represents input voltage signals and $\boldsymbol{B}$ is encoded within the crossbar as memristor conductances. Separate ADCs can be used to read out the current of each column in parallel, as depicted, or sample and hold circuits can be used in conjunction with a single ADC per crossbar, that can be used to read out the current of each column sequentially using TDM. As the output current of each column is linearly proportional to the elements of $\boldsymbol{AB}$, a linear constant, K, is used to correlate the ADC readout of each column accordingly. By separately presenting each row of $\boldsymbol{A}$ to the crossbar through WLs, all rows of $\boldsymbol{AB}$ can be computed.

Because memristors cannot be programmed to have negative conductances, within MDNNs, weight matrices can either be represented using two devices per weight [377], as described by (6.1),

$$\boldsymbol{AB} = K \sum_{i=0}^{C} \boldsymbol{A}[i,:](g_{\text{pos}}[i,j] - g_{\text{neg}}[i,j]), \text{for } j = 0 \text{ to } N, \tag{6.1}$$

or using a single device per weight [96, 378] using complex weight mapping algorithms or current mirrors, as described by (6.2),

$$\boldsymbol{AB} = K \sum_{i=0}^{C} \boldsymbol{A}[i,:](g[i,j] - g_m), \text{for } j = 0 \text{ to } N. \tag{6.2}$$

For the single-column case, the current through g_m, used to mirror a current $-2V/(\bar{R}_{\text{ON}} + \bar{R}_{\text{OFF}})$ to each crossbar, is copied to each column and subtracted from all memristor columns. This current can be realized using a diode-connected NMOSFET by adjusting the NMOSFET channel width so that it has a passive resistance g_m. From this stage forward, we refer to the weight matrix representation methodology adopted, that is, whether two devices are used to encode each weight, i.e, differential weight mapping, one device is used to encode each weight, or another configuration is used to encode weight matrices, as the parameter representation scheme.

Modular memristive crossbar tiles

Mapping complete unrolled neural network layers into large memristive crossbar architectures often results in poor performance. This is due to non-ideal device characteristics that introduce substantial current variability when accumulated currents from columns with a large number of devices are read out. When one or two large crossbars are used, for single-column and double-column parameter representation schemes, respectively, they cannot easily be modularized because customized crossbar shapes are required to represent each individual layer. Instead of using large crossbars, modular crossbar tiles [379] can be used that map layers into multiple uniformly sized crossbars, commonly referred to in literature as *crossbar tiles*.

One large crossbar of size $(M \times N)$ can be mapped using ceil$(M/S_0)\times$ ceil(N/S_1) crossbar tiles, each with a size of $(S_0 \times S_1)$, where the total utilization, ρ, of all crossbar tiles can be determined using (6.3),

$$\rho = \frac{MN}{\text{ceil}(M/S_0)\text{ceil}(N/S_1)S_0S_1}. \tag{6.3}$$

Duplication of crossbar tiles and TDM can be used to regulate mapping to improve the array utilization and computation time by balancing latency among layers [196].

Memristor crossbar programming

The conductance of memristive devices can be altered between a low resistance state R_{ON} and a high resistance state R_{OFF}, by applying programming voltage pulses with different intervals and amplitudes. While individual devices within crossbars can be selected and programmed within 1T1R cells, in 1R arrangements, when a voltage is applied to a specific device, a non-zero voltage (usually half that of the nominal programming pulse amplitude) is applied to all other devices in the same row and column. Consequently, various multistage programming [380–383] and corrective methods [11, 363, 384], which can use analog voltage wave-forms, are often used to ensure the difference between the programmed conductance states and the conductance states-to-program are within an acceptable tolerance.

Memristor crossbar tuning

The total current of each column in an ideal memristive crossbar is linearly proportional to the output elements of the VMM resultant vector. Consequently, after each DNN layer's weights are programmed into a crossbar or group of tiles, linear regression can be used to correlate the output current of each column with any desired output to determine K for the crossbar or group of tiles, given a randomly generated input matrix that is sufficiently large. On account of device-device variations and device failures, further tuning is often required to recover accuracy loss and mitigate variances between intended and actual device conductance values. Tuning methods can

Algorithm 2 Memristor crossbar programming algorithm.

Input: Array containing all continuous weights in a given layer, $\mathbf{w}$, HRS/LRS ratio, p_L.
Output: Equivalent memristive crossbars conductance values, $\mathbf{g}$, indexed using i and j.

$\mathbf{w}$ = abs($\mathbf{w}$)
$\mathbf{w}$ = descending_order($\mathbf{w}$)
s = size($\mathbf{w}$)
index = int($p_L \cdot s$)
$\mathbf{w}_{\text{max}} = \mathbf{w}$[index]
$\mathbf{w}_{\text{min}} = \mathbf{w}_{\text{max}}/(\mathbf{R}_{\text{OFF}}/\mathbf{R}_{\text{ON}})$
$\mathbf{w} = \text{clip}(\mathbf{w}, \mathbf{w}_{\text{min}}, \mathbf{w}_{\text{max}})$
$\boldsymbol{g}[i,j] = \frac{(R_{\text{ON}}-R_{\text{OFF}})\cdot(\sigma(\boldsymbol{w})[i,j]-\mathbf{w}_{\text{min}})}{|\boldsymbol{w}|_{\text{max}}-\mathbf{w}_{\text{min}}} + R_{\text{OFF}}$

either be used pre-programming [317], to improve robustness and reduce susceptibility to error, or post-programming by retraining device-specific conductance values [235].

Memristor crossbar weight mapping

Weights, denoted using $\boldsymbol{w}$, within unrolled convolutional layers [310] and linear layers can be mapped to equivalent conductance values, $\boldsymbol{g}$, using (6.4).

$$\boldsymbol{g}[i,j] = \frac{(g_{\text{ON}} - g_{\text{OFF}})(\sigma(\boldsymbol{w})[i,j] - \mathbf{w}_{\text{min}})}{|\boldsymbol{w}|_{\text{max}} - \mathbf{w}_{\text{min}}} + g_{\text{OFF}}, \tag{6.4}$$

where $\mathbf{w}_{\text{min}}$ represents the minimum weight value to encode, and $\mathbf{w}_{\text{max}}$ represents the maximum weight value to encode. $g_{\text{ON}} = 1/R_{\text{ON}}$ and $g_{\text{OFF}} = 1/R_{\text{OFF}}$. When two crossbars are used to represent weight, crossbars containing positive components will have $\sigma(\boldsymbol{w}) = \boldsymbol{w}[\boldsymbol{w} \geq 0]$, while crossbars containing negative components will have $\sigma(\boldsymbol{w}) = \boldsymbol{w}[\boldsymbol{w} \leq 0]$. When a single crossbar is used to represent weights, $\sigma(\boldsymbol{w}) = \boldsymbol{w} - g_m$.

To reduce the inner weight gap in a given device, Algorithm 2 in [225] can be used to exclude a small proportion, p_L, of weights with the absolute largest values to reduce the variability effect of non-ideal memristive devices.

Passive memristor crossbar architectures

When modeling passive memristor crossbar architectures, source and line resistances should be accounted for. MemTorch utilizes a comprehensive crossbar array model with solutions for source and line resistances, as described in [311], to solve for node voltages $\boldsymbol{V}$, within passive crossbar architectures in each simulation time-step. Specifically, linear matrix algebra is used to solve (6.5)

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \boldsymbol{V} = \boldsymbol{E}, \tag{6.5}$$

where for a crossbar of size $(M \times N)$, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ are all $(MN \times MN)$ matrices, and $\boldsymbol{E}$ is a $(2MN \times 1)$ vector. All matrices and vectors in (6.5) are derived and defined in [311]. As the concatenated $\boldsymbol{ABCD}$ matrix is sparse, traditional linear matrix algebra methods cannot be easily used to solve (6.5), because they require a prohibitive amount of memory. Instead, sparse supernodal LU factorization with partial pivoting for general matrices [385] is used, as it is parallelizable, and has demonstrated the best empirical numerical performance, compared to related techniques, e.g., QR decomposition [386].

6.6.4 C++ and CUDA Bindings

Numerous performance critical operations, including tiled VMMs, linear matrix algebra, and quantization, are accelerated using C++ and CUDA bindings. `PYBIND11_MODULE()` is a method within the pybind11 [5] python library [387] that exposes C++ types in Python to enable seamless operability between C++11, CUDA, and Python. This library is used within MemTorch to overload method pointers and to expose C++ and CUDA functions to the developed Python API. The Eigen [388] C++ template library is used extensively to perform various linear algebra operations, as many Eigen functions can be compiled for use within CUDA kernels using `__device__ __host__` function type qualifiers.

6.7 Modeling Non-Ideal Device Characteristics

Non-ideal device characteristics can either be encapsulated within device-specific memristive models, or introduced to base (generic) models using the `memtorch.bh.nonideality` sub-module. This sub-module can currently be used to introduce four non-ideal device characteristics to memristive device models: device-device variability, finite number of discrete conductance states, device failure, and non-linear I/V device characteristics. We leave native support for modeling other non-ideal device characteristics, such as programming non-linearity and asymmetry, resistance state drift in time, Random Telegraph Noise (RTN), and thermal noise, to future releases. Three of the non-ideal device characteristics that are currently supported by MemTorch are shown in Fig. 6.4. Fig. 6.4[A] depicts typical non-linear I/V device characteristics using a set-reset curve and an inset hysteresis loop. Fig. 6.4[B] demonstrates gradual switching, which is used to achieve a finite number of stable conductance states, and Fig. 6.4[C] shows overlapping distributions of R_{ON} and R_{OFF}, which is caused by device-to-device variability.

6.7.1 Device-to-device Variability

Device-to-device variability is modeled stochastically using `memtorch.bh.StochaticParameter`. Stochastic parameters are generated using the `memtorch.bh.StochaticParameter. Stochat-`

[5] `https://github.com/pybind/pybind11`

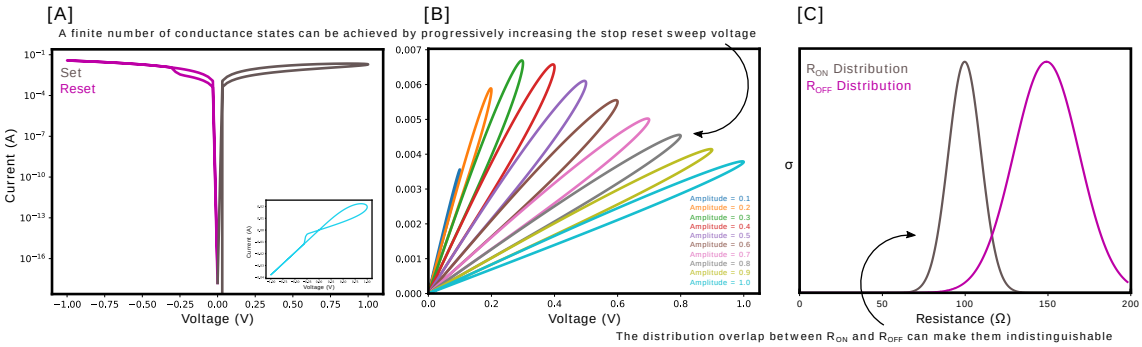


Figure 6.4: Depiction of [A] device I/V characteristics, and [B] reset voltage double-sweeps demonstrating gradual switching from R_{ON} to R_{OFF}, which can be used to achieve 10 finite stable conductance states for the VTEAM model using the TEAM [25] model's parameters, with a linear dependence on w, achieved using sinusoidal signals with a fixed frequency of 50 MHz. [C] shows distributions of R_{ON} and R_{OFF}, which are caused by device-device variability, for a memristive device with $\bar{R}_{\text{ON}} = 100\Omega$ and $\bar{R}_{\text{OFF}} = 150\Omega$. In [C], overlapped regions are indistinguishable from each other.

`icParameter()` method, which accepts an arbitrary number of keyword arguments, that are used to sample from a `torch.distributions` each time a device model is instantiated. To model device-device variability, we use stochastic parameters to sample R_{ON} and R_{OFF} from a normal distribution with $\sigma R_{\text{ON}} = \sigma$ and $\sigma R_{\text{OFF}} = 2\sigma$. $\sigma R_{\text{OFF}} > \sigma R_{\text{ON}}$, as the variability of R_{OFF} has been demonstrated to be larger than R_{ON} [389]. As depicted in Fig. 6.4[C], device-device variability can cause the distribution of R_{ON} and R_{OFF} to overlap, resulting in R_{ON} and R_{OFF} occupying the same conductance regions.

6.7.2 Cycle-to-cycle Variability

Cycle-to-cycle (C2C) variability [390] is modeled stochastically, similarly to device-to-device variability, using stochastic parameters for R_{ON} and R_{OFF}. `memtorch.bh.nonideality.DeviceFaults.apply_cycle_variability()` is used to sample R_{ON} and R_{OFF} from a normal distribution with $\sigma R_{\text{ON}} = \sigma$ and $\sigma R_{\text{OFF}} = 2\sigma$ after each SET RESET cycle.

6.7.3 Finite Number of Discrete Conductance States

Realistic memristive devices are non-ideal and have a finite number of stable discrete electrically switchable conductance states, bounded by a low-conductance semiconducting state R_{OFF}, and a high-conductance metallic state, R_{ON} [391]. Previous works have investigated evenly spaced conductance or resistance states, and have demonstrated that, assuming they are relatively uniformly distributed, the spacing between states is not critical [225].

Therefore, deterministic discretization [392] can be used to represent a finite number of electrically switchable conductance states, as depicted in Fig. 6.4[B]. In order to efficiently quantize a

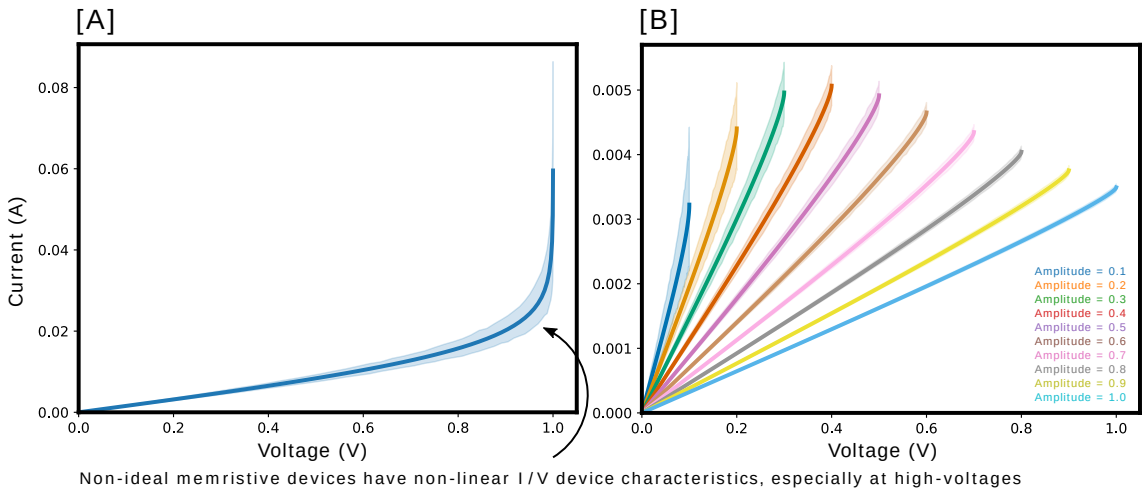


Figure 6.5: Non-linear I/V characteristics for 100 devices (instances) of the VTEAM model using the TEAM [25] model's parameters, with a linear dependence on w, achieved using sinusoidal signals with a fixed frequency of 50 MHz. R_{ON} and R_{OFF} were stochastically sampled from a normal distribution with $\bar{x} = 50, \sigma = 25$, and $\bar{x} = 1000, \sigma = 50$, respectively. [A] depicts I/V characteristics for devices with an infinite number of discrete conductance states. [B] depicts I/V characteristics for devices with a finite number of discrete conductance states.

tensor to a defined finite number of quantization states, in which each element can have a different range, CUDA kernels are used to perform a binary search on sorted tensors (generated using the `linspace` algorithm in C++) containing defined quantization states in $\mathcal{O}(n\log(n))$, where n is the number of quantized states.

6.7.4 Device Failure

Memristive devices are susceptible to failure, by either failing to eletroform at a pristine state, or becoming stuck at high or low resistance states [225]. MemTorch incorporates a specific function for accounting for device failure in simulating DL systems. Given a `nn.Module`, `memtorch.bh.nonideality.DeviceFaults.apply_device_faults()` sets the conductance of a proportion of devices within each crossbar to R_{ON} or R_{OFF}. It is assumed that the total proportion of devices set to R_{OFF} is equal to the proportion of devices that fail to eletroform at pristine states plus the proportion of devices stuck at a high resistance state. However, these proportions and the ratio of device failures can be manipulated as desired. Devices are chosen at random using `np.random.choice()`.

6.7.5 Non-linear I/V Characteristics

Non-ideal memristive devices have non-linear I/V device characteristics, especially at high voltages, which are difficult to accurately and efficiently model [225]. We demonstrate these charac-

teristics using Fig. 6.5[A], by depicting the I/V curve of the VTEAM model between 0–1V using the TEAM [25] model's parameters. The `memtorch.bh.nonideality.NonLinear.apply_non_linear()` method can be used to efficiently model non-linear device I/V characteristics during inference for devices with an infinite number of discrete conductance states, and for devices with a finite number of conductance states. For cases where devices are not simulated using their internal dynamics, it is assumed that the change in conductance during read cycles is negligible.

Devices with an infinite number of discrete conductance states

The `memtorch.bh.nonideality.NonLinear.apply_non_linear()` method uses two methods to efficiently model non-linear device I/V characteristics for devices with an infinite number of discrete conductance states during inference:

1. During inference, each device is simulated for a single timestep, `device.time_series_resolution`, using `device.simulate()`;
2. Post weight mapping and programming, the I/V characteristics of each device are determined using a single reset voltage sweep. The I/V characteristics of each device are stored, and used as Lookup Tables (LUTs) to compute device output currents during inference.

Devices with a finite number of discrete conductance states

The `memtorch.bh.nonideality.NonLinear.apply_non_linear()` method effectively models non-linear I/V characteristics for devices with a finite number of discrete conductance states by determining the I/V characteristics of each device post weight mapping and programming during several single reset voltage sweeps. Fig. 6.5[B] depicts sweeps for 100 stochastic devices with 10 finite discrete conductance states. These are stored and used as LUTs to compute device output currents during inference, where each I/V curve corresponds to each finite discrete conductance state. In Fig. 6.5[B], the smallest voltage amplitude corresponds to the finite conductance state closest to R_{ON}, whereas the largest voltage amplitude corresponds to the finite conductance state closest to R_{OFF}.

6.8 Exemplar Simulation Details

For all simulations performed to obtain the results presented in Fig. 6.1, we followed the following training and test procedure. We first augmented a pretrained MobileNetV2 CNN trained using the CIFAR-10 training set. All convolutional and linear layers within the network were sequenced with batch-normalization layers with fixed affline parameters to normalize outputs. The network was trained until improvement on the validation set was negligible (for 100 epochs) with a batch size of $\Im = 256$. The initial learning rate was $\eta = 1e - 1$, which was decayed by an order of

magnitude every 40 training epochs. SGD was used to optimize network parameters and Cross Entropy [79] was used to determine network losses. The network achieved $> 90\%$ accuracy on the CIFAR-10 test set.

When implementing the MDNNs, each memristive layer's weights were mapped to a double column line crossbar architecture adopting a 1T1R arrangement. Linear regression was used to correlate the output current of each column and its corresponding output to determine K for each crossbar, given a randomly generated input matrix sampled from a uniform distribution between ± 1.0. For linear layers, the random inputs had a size of ($8 \times$ `in_features`), while for convolutional layers the random inputs had a size of ($8 \times$ `in_channels` $\times\ 32 \times 32$). Unless otherwise stated, inputs to memristive layers were scaled from -0.3 to 0.3, to emulate voltage signals between $\pm$0.3V, which were applied to the word-lines of each memristive crossbar. All device models originated from the VTEAM model, with $\bar{R}_{\text{ON}}$ = 1.4e4Ω and $\bar{R}_{\text{OFF}}$ = 5e7Ω, to model TiN/Hf(Al)O/Hf/TiN devices from [27].

Implementations are investigated using modular crossbar tiles of size 128$\times$128 and 256$\times$64, as these have previously been demonstrated to be effective in terms of utilization and power efficiency [196]. While power and latency balancing is beyond the scope of MemTorch 1.1.5, 256$\times$64 tile size enables higher operation throughput and more analog operations per ADC compared to 128$\times$128 tile size [196]. However, the area utilization may be lower for arrays with more than 64 columns considering the number of output channels.

6.9 Conclusion

We presented an open-source simulation framework, entitled *MemTorch*, for large-scale deep memristive crossbar architectures. We showed that MemTorch is designed with a focus to integrate any desired behavioral or experimental device model, and introduce arbitrary device non-idealities, while co-simulating crossbar and peripheral circuitry. We compared MemTorch to similar works, detailed its package structure, and performed exemplar simulations to demonstrate its functionality. We hope that MemTorch will be adopted and expanded by the community to advance memristive DL research and development endeavours.

Chapter 7

Empirical Metal-Oxide RRAM Device Endurance and Retention Model for Deep Learning Simulations

In this Chapter, the second research question is addressed. A novel generalized empirical Metal-Oxide Resistive Random-Access Memory (RRAM) endurance and retention model for use in large-scale Deep Learning (DL) simulations is presented. The developed model is the first to unify retention-endurance modeling while taking into account time, energy, SET-RESET cycles, device size, and temperature. It is demonstrated that, even when ignoring other device non-idealities, retention and endurance losses significantly affect the performance of DL networks.

7.1 Introduction

RRAM devices have attracted significant attention for use in next generation DL and neuromorphic architectures to perform in-memory computing operations, which can reduce power usage and time complexity, massively augmenting performance [145, 256, 352, 393]. However, RRAM is an emerging technology with a number of limitations including endurance and retention losses, as depicted in Fig. 7.1. Consequently, significant research efforts are being made to efficiently and accurately model device limitations to improve the reliability and robustness of RRAM-based DL architectures [394–396].

In this Chapter, we propose a generalized empirical Metal-Oxide RRAM device endurance and retention model. We compare our model to related works and demonstrate its versatility by using it to fit experimental data from several devices. We then deploy the model within large-scale DL simulations to implement the MobileNetV2 CNN architecture to investigate how device endurance and retention losses affect inference performance using the CIFAR-10 dataset.

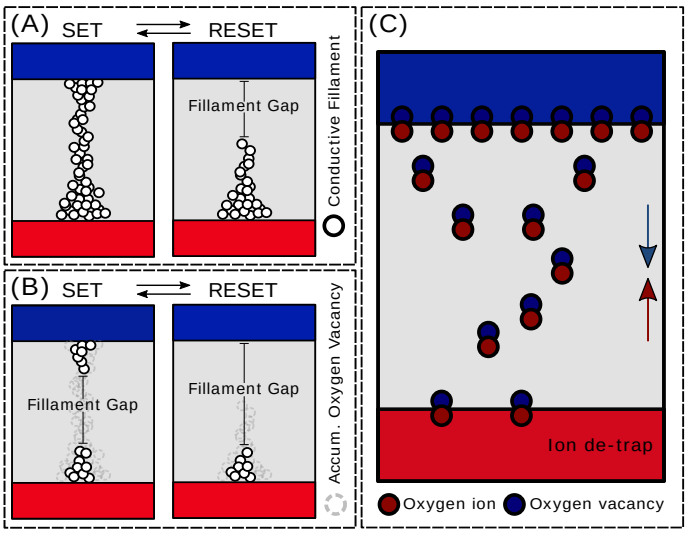


Figure 7.1: (A) The formation of a conductive filament within metal-oxide RRAM devices results in low resistive states, whereas its partial destruction increases the resistivity to high resistive states. (B) When a voltage is applied, defects are gradually created within the conductive filament [26], which cause endurance losses. (C) Oxygen ions return to the previous thermal equilibrium state during the baking process, which causes retention losses.

7.2 Related Work

Previous works have investigated Metal-Oxide RRAM endurance and retention losses experimentally [29, 397–403], numerically [404], and analytically [34, 139, 224, 237, 405–407]. Table 7.1 compares the proposed model with previous numerical and analytical RRAM device-level endurance and retention models. Given the increasing popularity of RRAM-based Deep Memristive Neural Networks (DMNNs), a number of works [29, 139, 224, 237, 399–401, 405] specifically consider endurance and retention loss effects on DMNNs performance. While most models [34, 405–407] are inherently physics-based and model various phenomena and internal device mechanics using fundamental physics principals, others [139, 224, 237, 404] adopt a generalized high-level approach, and model device behavior empirically. Our model fits into the latter group, and is the first to:

1. Accurately model device endurance and retention behavior, before and after the conductance window begins to collapse;
2. Model both gradual and sudden window collapse;
3. Model temperature, cell size, and when modeling endurance, the voltage dependence, V_{stop};
4. Model endurance and retention interchangeably using a unified methodology.

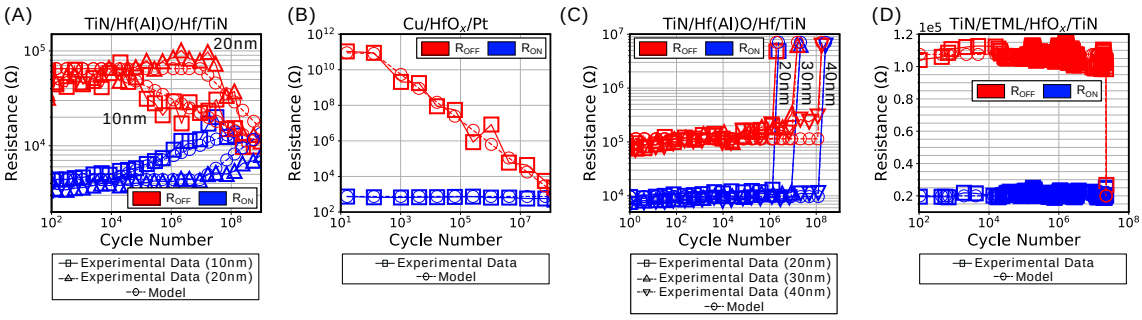


Figure 7.2: Experimental endurance data from various Metal-Oxide RRAM device types, and the behavior of our proposed model. (A) TiN/Hf(Al)O/Hf/TiN [27] devices with different cell sizes, (B) Cu/HfO$_x$/Pt [28] devices, and results from the proposed model in gradual resistance convergence operation mode; (C) TiN/Hf(Al)O/Hf/TiN [27] devices with different cell sizes, (D) TiN/Electro-thermal Modulation Layer (ETML)/HfO$_x$/TiN [29] devices, and results from the proposed model in sudden resistance convergence operation mode.

The proposed model is well suited toward DL modeling using memristors, as the behavior of new devices can easily be modeled using tools provided in our supplementary materials ‡, it is highly integrable ‡, and it is able to capture a large range of Metal-Oxide RRAM device behavior, as depicted in Fig. 7.2, Fig. 7.3, and Fig. 7.4.

7.3 Proposed Model

The proposed model has two modes of operation. The first mode assumes that resistance states gradually converge after a device-specific threshold energy level is exceeded, and can be used to model device endurance and retention, as depicted in Fig. 7.2 (A,B) and Fig. 7.3 (B,C). The second mode, on the other hand, assumes sudden failure, and can be used to model device endurance, as

Table 7.1: Comparison of RRAM endurance and retention models. †Models are defined independently.

Model	Models	
	Endurance	Retention
Endurance Statistical [404]	✓	
Statistical State Instability and Retention [405]		✓
Reliability Perspective [139, 224, 237]	✓†	✓†
Endurance, Retention and Window Margin [406]	✓	
Retention Model for High-Density RRAM [407]		✓
Voltage-Controlled Cycling Endurance [34]	✓	
Proposed	✓	✓

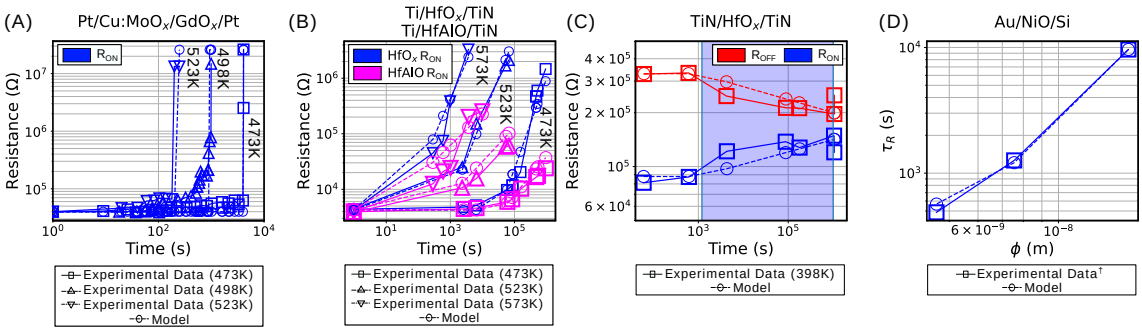


Figure 7.3: Experimental retention data from various Metal-Oxide RRAM device types, and the behavior of our proposed model. (A) Pt/Cu:MoO$_x$/GdO$_x$/Pt [30] devices operating at different temperature points, and results from the proposed model in sudden resistance convergence operation mode; (B) TiN/HfO$_x$/TiN and Ti/HfAlO/TiN devices [31] operating at different temperature points, and results from the proposed model in sudden resistance convergence operation mode; (C) TiN/HfO$_x$/TiN [32] devices and results from the proposed model in gradual operation mode, where the temperature was elevated from room temperature (25°C) to 125°C depicted using a blue background segment between 1200s and 10^6s; (D) Relationship between the retention time to failure, τ_R, and conductive filament diameter, ϕ, of Au/NiO/Si [33] devices, and results from the proposed model, where ϕ is substituted for the cell size, and $\tau_R \propto e_{th}$, i.e., $\tau_R = p_0 e^{p_1\phi + p_2 T_c}$.† The conductive filament size, which can be representative of device dimension, was obtained using a piecewise linear fit of the mean activation energy, E_{AC}, as done in [33].

depicted in Fig. 7.2 (C,D) and Fig. 7.3 (A). The gradual convergence of resistance states is modeled as

$$R(x,s,T) = \begin{cases} R_0 & x \leq e_{th} \\ 10^{p_3(p_1 s + p_2 T_c)\log(x) + \log(R_0) - p_3(p_1 s + p_2 T_c)\log(e_{th})} & \text{otherwise,} \end{cases} \tag{7.1}$$

and the sudden convergence of states is modeled as

$$R(x,s,T) = \begin{cases} R_0 & x \leq e_{th} \\ R_\infty & \text{otherwise,} \end{cases} \tag{7.2}$$

where the device-specific threshold energy level, e_{th}, if exceeded, causes the resistance window of a device to collapse either gradually or suddenly, is modeled using

$$e_{th} = p_0 e^{p_1 s + p_2 T_c}. \tag{7.3}$$

The temperature constant, T_c, is expressed as

$$T_c = \min(\frac{T_{th}}{T}, 1), \tag{7.4}$$

which is used to introduce temperature dependence to the model. Using (7.1)–(7.4), the resistance state of a device, R, is determined using four parameters, x, s, T, and T_{th}, and various fitting parameters, $\mathbf{p}$. Here, R_∞, the collapsed resistive state to which R_{ON} and R_{OFF} converge, is bounded to the range $[R_{\text{ON}}, R_{\text{OFF}}]$ [27, 29, 32, 34, 403, 408]. x denotes either the time (s), the energy (J), or the number of SET-RESET cycles, s denotes the device cell size (nm), when the depth and width are fixed, or the filament volume (nm^3) when they are not, T denotes the operating temperature (K), and T_{th} denotes the temperature threshold, that if exceeded, accelerates device failure. For both modes of operation, p_0 modulates the magnitude of e_{th}, and p_1 and p_2 modulate the strength of the dependence on s and T, respectively. For instances where s is fixed, p_1 = 0, and for instances where T is fixed, p_2 = 0. When modeling the gradual convergence of resistance states, p_3 is used to modulate the rate of failure once e_{th} is exceeded. We believe that, given sufficient data, all fitting parameters could be related to physical device parameters, such as those determined using ab initio calculations in [406], including formation enthalpy energies, ΔH, migration barriers, E_d, and hopping distances between sites during ion migration, d_h.

The parameter p_0 in (7.3) can be modulated using (7.5) when modeling endurance to introduce dependence to V_{stop}, the most negative voltage in the negative voltage sweep during the RESET

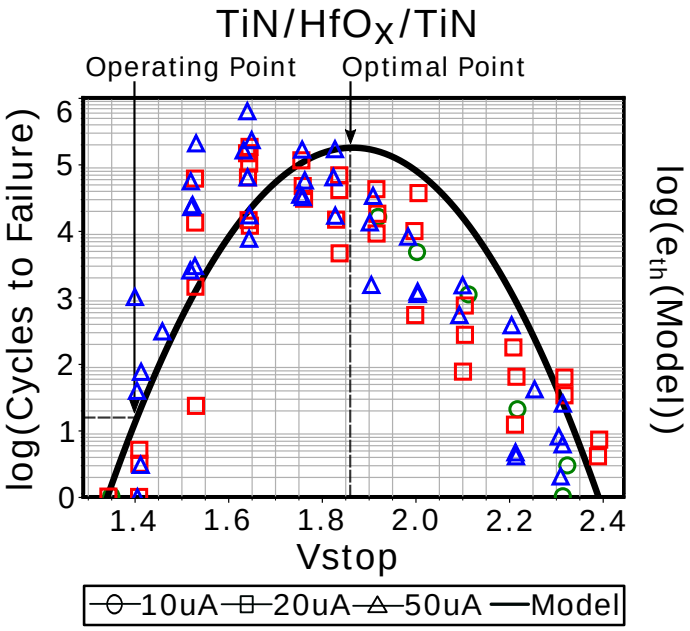


Figure 7.4: The window function used to model v_{stop} dependence. Experimental data is extracted from TiN/HfO_x/TiN devices [34].

cycle [34].

$$p_0 = \frac{10^{K(1-(2\bar{V}_{\text{stop}}-1)^2)}}{e^{p_1 s + p_2 T_c}} \tag{7.5}$$

K is used to modulate the amplitude, and V_{stop} is mapped to $\bar{V}_{\text{stop}} \in [0, 1]$. Fig. 7.4 demonstrates the inclusion of V_{stop} dependence in the proposed model, where x is assumed to denote the number of SET-RESET cycles, and the model to be used will operate in sudden resistance convergence mode. In this figure, the optimal point corresponds to the optimal V_{stop} value, i.e., the V_{stop} value for a given device that maximizes e_{th}. Given sufficient experimental data observing the relationship between V_{stop} and e_{th}, the mapping bounds of V_{stop} can be determined, and the K parameter can be determined using Nonlinear Least Squares Regression (NLSR).

The following assumptions are made in our modeling:

1. The waveform used to program each device is constrained, and only V_{stop} is mutable; and
2. The impacts of the compliance current, I_c, and the maximum set voltage magnitude are considered negligible [34]; and
3. Resistance states converge to R_∞ when device failure occurs; and
4. Resistance states are stable until a device-specific threshold energy level, e_{th}, is exceeded; and
5. (7.5) is constrained to be symmetrical around the optimal point.

7.4 Model Validation

To validate the proposed model, we fit it to experimental data from various fabricated devices, indicative of a variety of use cases, as shown in Fig. 7.2, Fig. 7.3, and Fig. 7.4. NLSR is used to fit the model empirically to each device type. In Fig. 7.2 and Fig. 7.3 (C), two sets of parameters are used to model R_{OFF} and R_{ON}, respectively, for each simulated device. In Fig. 7.3 (A,B), one set of parameters are used to model R_{ON}. To the best of our knowledge experimental data for R_{OFF} is currently not available in literature.

In Fig. 7.3 (D), we model the relationship between the retention time to failure, τ_R, and the conductive filament diameter, ϕ, of Au/NiO/Si [33] devices. The conductive filament size, which can be representative of device dimension, was obtained using a piecewise linear fit of the mean activation energy, E_{AC}, which accounts for metallic and semiconductor-like behavior, as done in [33]. $\bar{V}_{\text{stop}}$ dependence is validated in Fig. 7.4 using TiN/HfO_x/TiN devices [34]. We note that variations between experimental data and the behavior of the proposed model could be further reduced by simulating other device non-idealities, such as conductance drift, evident in Fig. 7.2 (C) and Fig. 7.3 (A), however, this is beyond the scope of this Chapter. In favour of reproducible

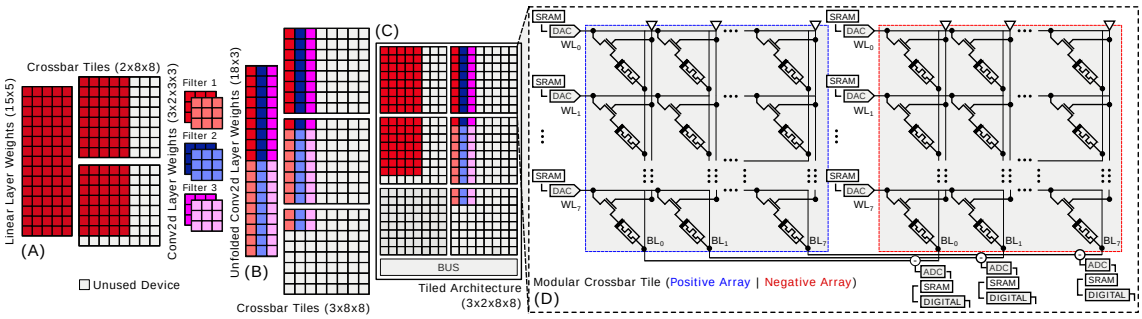


Figure 7.5: An overview of the mapping process of Linear (dense) and Conv2d (convolutional) layers onto a 3×2 tiled architecture with tiles constructed using 128 ($2\times8\times8$) devices. (A) Linear layers are mapped directly onto crossbar tiles. (B) Convolutional layers are unfolded before being mapped onto crossbar tiles. (C) Tiled architectures contain several modular crossbar tiles connected using a shared bus. (D) Modular crossbar tiles consist of crossbar arrays with supporting peripheral circuitry, and can represent weights using a dual-array scheme (as depicted), a dual row scheme, where double the number of rows are required, or a current-mirror scheme, that is capable of operation using a singular device to represent each weight [15].

research, our model, its fitting parameters, and all of the information required to reproduce the reported results are made publicly available [1].

7.5 Large-scale Deep Learning Simulations

Exemplar large scale DL simulations were performed that modeled the gradual and sudden resistance state convergence on account of endurance and retention losses of TiN/Hf(Al)O/Hf/TiN, TiN/ETML/HfO_x/TiN, and TiN/Hf_x/TiN RRAM devices using the VTEAM model [151] within layers of a DMNN employing 1T1R crossbars. These crossbars were constructed by converting linear and unfolded convolutional layers from a pre-trained MobileNetV2 CNN that achieved 91.93% accuracy on the CIFAR-10 test set. In Fig. 7.5, we overview the mapping process of linear and convolutional layers onto a modular tiled architecture. Batch-normalization, pooling, and activation functions, which are simulated in our experiments, should be implemented using additional circuitry to realize the other computations required for a DL task. Inputs are unfolded and scaled, prior to being presented to the network. By generalizing this approach, modular crossbar tiles and digital circuitry can be used to perform inference of any arbitrary DNN [196].

Algorithm 3 details our simulation methodology, in which a double-column scheme is used to represent network weights within memristive crossbars, i.e., a dual-array scheme is adopted. All RRAM devices are assumed to operate as fully analog devices, and other device non-idealities are ignored. ADCs are assumed to have a bit-length of 8, and modular crossbar tiles are constructed using two arrays of 128×128 devices, representing positive and negative parameters, respectively.

[1] https://github.com/coreylammie/SST-Reproducibility

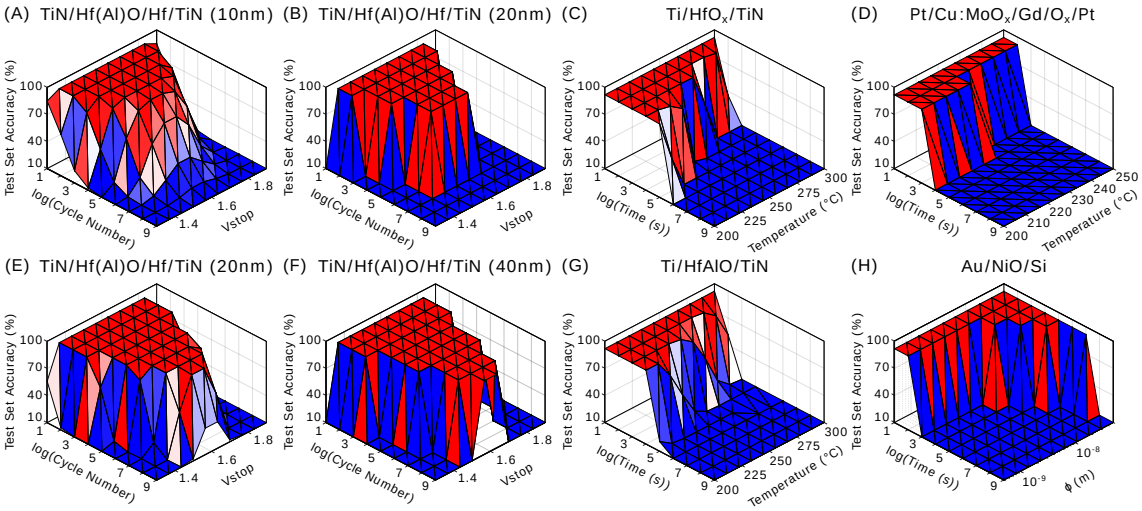


Figure 7.6: Large-scale DL simulations of TiN/Hf(Al)O/Hf/TiN, TiN/HfO$_x$/TiN, Pt/Cu:MoO$_x$/GdO$_x$/Pt, TiN/HfAlO/TiN, and Au/NiO/Si devices. (A,E) gradual endurance failure; (B,F) sudden endurance failure; (C,G) gradual retention failure; (D,H) sudden retention failure.

Unfolded inputs are scaled and encoded using voltage signals between $\pm 0.3V$ [313].

After network parameters are mapped, to tune each memristive layer, random inputs of variable size that are sampled from uniform distributions between ± 1.0 are presented to each layer. The readout currents of each column associated with each layer are linearly related to each layer's desired output. Prior to endurance and retention losses, our RRAM-based networks achieved 91.69% accuracy on the CIFAR-10 test set. We attribute the small performance degradation to quantization noise introduced from ADCs and the non-ideal mapping and tuning methodologies employed. The results from six exemplar large-scale DL simulations are presented in Fig. 7.6. Each surface plot is constructed from the results of 100 individual simulations (one per point).

In Fig. 7.6 (A,B,E,F), the CIFAR-10 test set accuracy is reported after each SET-RESET cycle to investigate the performance degradation on account of endurance losses, i.e., we assume massive reprogramming in the DNN accelerator is performed. v_{stop} was extrapolated using (7.5), where $\max(v_{\text{stop}})$ was arbitrarily chosen to be 1.6, due to the unavailability of experimental data on v_{stop}. K and the mapping bounds of v_{stop} were determined using operational points from each device. In Fig. 7.6 (C,D,G,H), the CIFAR-10 test set accuracy is reported at each time-step to investigate the performance degradation on account of retention losses. TiN/Hf(Al)O/Hf/TiN devices from Fig. 7.2 (A) are modeled to achieve the results in Fig. 7.6 (A) and Fig. 7.6 (E), for devices with cell sizes of 10nm and 20nm, respectively; TiN/Hf(Al)O/Hf/TiN devices from Fig. 7.2 (C) are modeled to achieve the results in Fig. 7.6 (B) and Fig. 7.6 (D), for devices with cell sizes of 20nm and 40nm, respectively; Ti/HfO$_x$/TiN devices from Fig. 7.3 (B) are modeled to achieve the results in Fig. 7.6 (C); Pt/Cu:MoO$_x$/GdO$_x$/Pt devices from Fig. 7.3 (A) are modeled to achieve the results in Fig. 7.6 (D); Ti/HfAlO/TiN devices from Fig. 7.3 (B) are modeled to achieve the results in Fig. 7.6 (G),

Algorithm 3 Adopted simulation methodology.

1. Map Network Parameters
for each convolutional and linear layer **do**
$\boldsymbol{W}_{\text{max}} = \text{descending\_order}(\text{abs}(\boldsymbol{W})[\text{size}(\boldsymbol{W})])$
$\boldsymbol{W}_{\text{min}} = \boldsymbol{W}_{\text{max}}/(R_{\text{OFF}}/R_{\text{ON}})$
$\boldsymbol{W}_{\text{pos}} = \boldsymbol{W}[\boldsymbol{W} \geq 0]$, $\boldsymbol{W}_{\text{neg}} = \boldsymbol{W}[\boldsymbol{W} < 0]$
for each device, $\boldsymbol{R}_{\text{pos}}[i,j]$, $\boldsymbol{R}_{\text{neg}}[i,j]$ in $\boldsymbol{W}_{\text{pos}}$, $\boldsymbol{W}_{\text{neg}}$ **do**
$\boldsymbol{R}_{\text{pos}}[i,j] = \frac{(R_{\text{ON}}-R_{\text{OFF}})(\boldsymbol{W}_{\text{pos}}[i,j]-\mathbf{w}_{\text{min}})}{|\boldsymbol{w}|_{\text{max}}-\mathbf{w}_{\text{min}}} + R_{\text{OFF}}$
$\boldsymbol{R}_{\text{neg}}[i,j] = \frac{(R_{\text{ON}}-R_{\text{OFF}})(\boldsymbol{W}_{\text{neg}}[i,j]-\mathbf{w}_{\text{min}})}{|\boldsymbol{w}|_{\text{max}}-\mathbf{w}_{\text{min}}} + R_{\text{OFF}}$
end for
end for
2. Tune Memristive Layers
for each converted memristive layer **do**
if the layer is convolutional **then**
$P = (8 \times \text{in\_channels} \times 32 \times 32)$
else if the layer is linear **then**
$P = (8 \times \text{in\_features})$
end if
determine β_0 for $\tilde{Y} = \beta_0 \tilde{X}$, where $\tilde{Y}$ denotes the legacy layer's output and $\tilde{X}$ denotes the converted layer's output when a randomly generated tensor of size P is propagated.
end for
3. Model Device Endurance and Retention
for each value of x to simulate **do**
for each converted memristive layer **do**
for each device, $\boldsymbol{R}_{\text{pos}}[i,j]$, $\boldsymbol{R}_{\text{neg}}[i,j]$ in $\boldsymbol{W}_{\text{pos}}$, $\boldsymbol{W}_{\text{neg}}$ **do**
$\boldsymbol{R}_{\text{pos}}[i,j], \boldsymbol{R}_{\text{neg}}[i,j] = R(x, s, T, \bar{V}_{\text{stop}})$
end for
end for
determine the test set accuracy for the given x value
end for

and Au/NiO/Si devices from Fig. 7.3 (D) are modeled to achieve the results in Fig. 7.6 (H). From Fig. 7.6, it can be observed that the proposed model is capable of robustly modeling endurance and retention losses of Metal-Oxide RRAM devices within large-scale DL simulations.

7.6 Discussion and Conclusion

We proposed a novel generalized empirical Metal-Oxide RRAM device endurance and retention model for use in large-scale simulations. We demonstrated its versatility by fitting it to experimental data from various devices, and using it for large DL simulations. Our findings show that, even when other device non-idealities are ignored, endurance and retention losses significantly affect the reprogrammability of DMNNs, degrading their learning and inference accuracy. A limitation of the proposed model is the lack of a clear link between its parameters and physical device

parameters. This is mainly due to unavailability of experimental data, which resulted in developing an empirical, rather than a physics-based model. Additionally, while this work only focuses on endurance and retention and their impact on memristive DL networks performance, future improvements of our model can account for modeling a finite number of conductance states and other device non-idealities [5, 225].

Chapter 8

Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge

With recent advancements in High Level Synthesis (HLS) techniques, new methods for accelerating Deep Learning (DL) systems using Field Programmable Gate Arrays (FPGAs) are emerging. FPGA-based Deep Neural Networks (DNNs) present substantial advantages in energy efficiency over conventional Central Processing Unit (CPU)- and Graphics Processing Unit (GPU)-accelerated networks. Using the Intel FPGA Software Development Kit (SDK) for OpenCL development environment, networks described using the high-level OpenCL framework can be accelerated targeting heterogeneous platforms including CPUs, GPUs, and FPGAs. These networks, if properly customized on GPUs and FPGAs, can be ideal candidates for learning and inference in resource-constrained portable devices such as robots and resource-constrained Internet of Things (IoT) devices, where power is limited and performance is critical. In this Chapter, the first and third research questions are addressed, and the first GPU- and FPGA-accelerated deterministically binarized DNNs, tailored toward weed species classification for robotic weed control, are presented.

8.1 Introduction

The promise of robotic weed control to provide a step change in the productivity of primary industry is widely coveted [409, 410]. The rationale is clear - replace human involvement in this time-consuming and labor-intensive undertaking with more efficient autonomous machines. In addition, improving the efficacy of weed control would have enormous economic impact [411]. The majority of current works in this arena focus on the four core technologies of robotic weed control: detection, mapping, guidance, and control [412]. The robust and efficient detection of weed species remains a major obstacle to the widespread uptake of robotic weed control technologies [411].

The use of DNNs specifically tasked for plant classification has demonstrated incredible perfor-

mance in recent works [413–415]. Using the Intel FPGA SDK for OpenCL development environment, networks described using the high-level OpenCL framework can be accelerated by targeting heterogeneous platforms with CPUs, GPUs, and FPGAs. These developed networks are ideal candidates for edge computing applications, where low-power consumption and high performance are critical.

In this Chapter, we investigate the acceleration of binarized DNNs on GPUs and FPGAs using the high-level OpenCL framework for weed species classification targeted toward robotic weed control, as depicted in Fig. 8.1. We demonstrate that our FPGA implementations, employing an `Intel DE1-SoC` FPGA development board, customized for edge processing, exhibit comparable performance to state-of-the-art hardware implementations, employing an `NVIDIA Titan V` GPU and an `AMD Ryzen 2700X @ 4.10 GHz Overclocked` CPU, which are typically used for conventional desktop-based processing. Our specific contributions are five-fold:

1. We implement and present the first FPGA-accelerated binarized DNN specifically tasked for weed species classification;

2. Our complete FPGA-accelerated DNN runs on a standalone System On a Chip (SoC), requiring no host computer or extra device for partial computation;

3. We investigate the effect of down-sampling input images on the DNN classification accuracy, and demonstrate that significantly reducing the image resolution has a marginal effect on accuracy;

4. We thoroughly compare our efficient implementations on GPU and FPGA platforms and demonstrate that our new binarized FPGA-accelerated DNNs offer significantly reduced

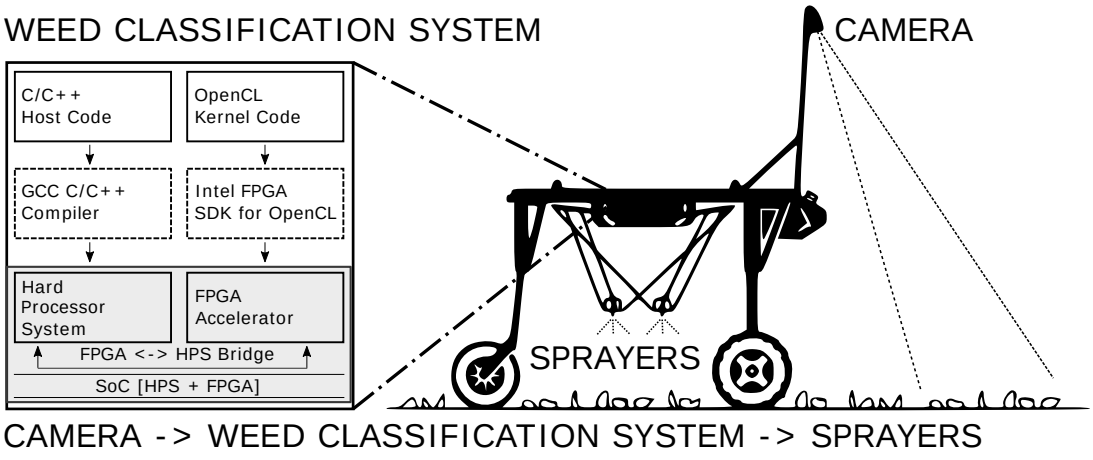


Figure 8.1: A block diagram detailing a complete robotic weed management system, where our developed weed classification inference engine is used for low-power and real-time weed classification, that can trigger herbicide sprayers for the detected and classified weed species.

power usage while lowering per-image inference times compared to their conventional GPU-accelerated counterparts;

5. We make our software code publicly available, to provide the community with the opportunity to replicate our experimental results and to adapt our utilized techniques for their various applications.

This Chapter is structured as follows: Section 8.2 presents related works. Section 8.3 presents an overview of the algorithms and methods used in our designed networks. Section 8.4 presents our new labeled version of the DeepWeeds dataset [416], *DeepWeedsX*. Our image pre-processing techniques used are detailed in Section 8.5. Section 8.6 presents our developed software and hardware network architectures. Section 8.7 reports the effect of image down-sampling on the network performance. Section 8.8 presents and discusses our software and hardware results, while Section 8.9 provides further discussions on classification and real-time performance of the proposed design. This Chapter is concluded in Section 8.10.

8.2 Related Works

A variety of techniques have been explored to automatically detect and classify target plant life. Means of detection can be categorised into one of three representations of the light spectrum: image-based [413–415, 417–423], spectrum-based [290, 424] and spectral image-based [425, 426].

In [416], we have achieved and demonstrated 95.7% weed classification accuracy on our large multiclass weed species image dataset, named DeepWeeds, using the ResNet-50 [427] CNN architecture on a single NVIDIA GTX 1080Ti GPU.

Performing real-time learning and inference, targeted for plant classification, on high-performance GPUs such as the NVIDIA GTX 1080Ti, is putative to consume large amounts of power, and hence, is ill-suited to deployment in portable resource-constrained smart devices and robotic systems, which are becoming commonplace. Consequently, devising methods and hardware synthesis techniques for reducing power consumption and for improving throughput of DNN hardware, becomes formidable.

While GPU-based implementations targeted towards image classification tasks are plentiful [241, 428–432], only one recent work [433] has detailed the implementation of custom hardware accelerators specifically tasked for agricultural purposes. [433] demonstrates real-time performance by implementing an FPGA-based DNN on a Terasic DE1-SoC for plant detection in organic farming. The system developed in [433] can classify a target dataset with accuracy matching that of a state-of-the-art GPU while running at up to 42 frames per second with only 4 W of power consumption, which is 45 times lower than an NVIDIA GTX 1080Ti.

One solution proffered by developers such as NVIDIA, are embedded GPUs for mobile applications. The Jetson family of compute modules and the NVIDIA TensorRT library provide a power-efficient platform for accelerating deep learning architectures, some of which have already started

being used in agriculture [416, 434–436]. Although embedded GPUs offer substantial power improvements over conventional GPUs, they are still relatively power hungry when compared to FPGAs. Moreover, they are usually cost prohibitive and are not suitable for developing products that require mass production, such as inference engines on IoT edge devices [437, 438]. Functionally verified HDL implementations developed for FPGA platforms can easily be synthesized to target ASICs, yielding considerable reductions in production costs. With recent advancements in HLS techniques, the development of FPGA-accelerated DNNs has been greatly expedited. FPGA-based DNNs present substantial advantages in energy efficiency over conventional GPUs accelerated networks, while exhibiting marginal performance degradation [133, 439–444].

8.3 Preliminaries

This Section briefly reviews the algorithms and methods used in our developed networks for the DeepWeeds classification benchmark.

8.3.1 Softmax Regression & Cross Entropy Loss

The Softmax model is commonly used to apply logistic regression to multinomial problems. Softmax regression [445] determines a discrete probability distribution of each class, ρ_i, for the output of the final layer in our developed deep networks using (8.1).

$$\rho_i = \frac{e^{y'_i}}{\sum_{i=1}^{N} e^{y'_i}}, \tag{8.1}$$

where, y'_i represents the predicted class. In addition, $\sum_{i=1}^{N} \rho_i = 1$ where N is the number of classes to be distinguished. Softmax regression is commonly used in tandem with cross-entropy loss, presented in (8.2), to enable the network to learn different classes during backward propagation where y_i represents the class label.

$$\frac{-1}{N} \sum_{i=1}^{N} y'_i \cdot \log(y_i) + (1 - y'_i) \cdot \log(1 - y_i) \tag{8.2}$$

8.3.2 Binary Weight Regularization

Since the solution space of DNNs is very broad, networks adopting gradient descent optimization algorithms, such as stochastic gradient descent with momentum, are susceptible to overfitting to training data. Overfitting significantly affects the generalization ability of the network and can lead to poor performance on test data. Regularization is any modification made to a learning algorithm to reduce its generalization error, but not its training error.

Binary weight regularization, as proposed in [446], constrains network weights to either $+1$ or -1 during forward and backward propagations. The binarization operation transforms the full-

precision weights into binary values. Deterministic binarization is based on the sign function presented in (8.3).

$$w_b = \begin{cases} -1 & \text{if } w \leq 0 \\ +1 & \text{otherwise,} \end{cases} \tag{8.3}$$

where w_b is the binarized weight and w is the real-valued full-precision weight.

8.3.3 L2 and Binarynet Regularization Loss Terms

Regularization is usually added as a term to the learning loss function, to introduce another degree of control to the network parameter growth and help avoid the network over- and under-fitting to the training data.

Several different regularization techniques have been proposed in literature such as $\ell 1$ and $\ell 2$ regularization [447], dropout [448], and data augmentation [449]. Here, we utilize $\ell 2$ regularization for implementations employing full resolution weights, and BinaryNet-regularization for our implementations employing binarized weights. The $\ell 2$-regularization term is defined in (8.4).

$$J(\boldsymbol{w}, \boldsymbol{b}) = L(\boldsymbol{w}, \boldsymbol{b}) + D\|\boldsymbol{w}\|_2^2 = L(\boldsymbol{w}, \boldsymbol{b}) + \frac{D}{2}\sqrt{\sum_{i=1}^{N} \boldsymbol{w}_i^2} \tag{8.4}$$

Here, $J(\boldsymbol{w}, \boldsymbol{b})$ shows the overall loss function that includes a task-related loss term, $L(\boldsymbol{w}, \boldsymbol{b})$, summed with a regularization term, in which D determines the relative significance of the regularization, N denotes the total number of trainable network parameters, w represents a weight, and b is a bias. This regularization technique penalizes the large network weights by adding the sum of their square to the loss function, which helps prevent over-fitting. However, as (8.4) is differentiable at $w_i = 0$, $\ell 2$-regularization has a non sparse solution, and does not perform feature selection.

BinaryNet-regularization [450] can be defined as (8.5),

$$J(\boldsymbol{w}, \boldsymbol{b}) = L(\boldsymbol{w}, \boldsymbol{b}) + D\sum_{i=1}^{N}(1 - \boldsymbol{w}_i^2), \tag{8.5}$$

where all the parameters are similar to (8.4). Using BinaryNet-regularization, the weights are piloted to +1 or -1, rather than to 0 as in a $\ell 2$-regularization, which is suitable for binary networks.

8.3.4 Training Algorithm

Algorithm 4 provides a high-level overview of the training algorithm of our accelerated binarized network architectures. Here, w, b, and η represent the weights, biases, and learning rate, while C denotes the cost function for each mini-batch. Furthermore, w_b represents binary weights and a_k

Algorithm 4 Training algorithm of the accelerated Binarized Neural Networks.

Input: a mini-batch of (inputs, targets), previous parameters w_{t-1} and b_{t-1}, and a learning rate η.

Output: updated parameters w_t and b_t.

1. Forward Propogation

$w_b \leftarrow$ binarize (w_{t-1}).

for $k = 1$ to L **do**

Compute a_k knowing a_{k-1}, w_b, b_{t-1}.

end for

2. Backward Propogation

Initialize output layer's activation gradient $\frac{\partial C}{\partial a_L}$

for $k = L$ to 2 **do**

Compute $\frac{\partial C}{\partial a_{k-1}}$ using $\frac{\partial C}{\partial a_k}$ and w_b.

end for

3. Parameter Update

Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{\partial db_{t-1}}$, using $\frac{\partial C}{\partial a_k}$ and a_{k-1}.

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$.

4. Weight Normalization

$w \leftarrow \text{clip}(w)$

represents the kth layer activation function, while `binarize()` implements (8.3), and `clip()` clips values between -1 and $+1$.

8.4 DeepWeedsX Dataset

The images used to construct the DeepWeedsX dataset have previously been made openly accessible [416], however, they have not been labeled as test and training images. Instead, they have been used in a 5-fold cross validation configuration for training and validation in [416]. Here, we present a labeled variant of DeepWeeds, DeepWeedsX, with clearly defined training and test datasets. We use a splitting ratio of 6:1 (train: test), similar to the popular MNIST [451], CIFAR-10 [452], and CIFAR-100 [452] image classification datasets.

The class distribution of the DeepWeedsX dataset is presented in Table 8.1. A validation subset may be constructed for parameter optimization using a subset of the labeled training data. In order to facilitate future comparisons to this work, DeepWeedsX, including all its labels and images, is made publicly available. In addition, we have developed data-loaders for PyTorch and TensorFlow, which are intended to assist utilizations in new deep learning experiments[1]. It is worth noting that, one image was removed from the original DeepWeeds dataset 2019 to ensure the training and test subsets have the same class distributions.

Table 8.1: DeepWeedsX class distribution.

Class	Species Label	Training	Test	Total
0	Chinee Apple	964	161	1,125
1	Lantana	912	152	1,064
2	Parkinsonia	884	147	1,031
3	Parthenium	876	146	1,022
4	Prickly Acacia	910	152	1,062
5	Rubber Vine	865	144	1,009
6	Siam Weed	921	153	1,074
7	Snake Weed	871	145	1,016
8	Negatives	7,804	1,301	9,105
Total		**15,007**	**2,501**	**17,508**

8.5 Image Pre-processing

Images available from our DeepWeedsX dataset have a resolution of 256×256 pixels. In order to enhance the testing accuracy, several pre-processing steps can be undertaken before the presentation of the images to the network. All of our implementations adopt one of two image pre-processing techniques that are denoted using Image Pre-processing Techniques (IPT), and Further Image Pre-processing Techniques (FIPT).

8.5.1 Image Pre-processing Techniques (IPT)

IPT down-samples input images from the native resolution of 256×256 to a resolution of 224×224, 64×64, or 32×32 pixels. No further image pre-processing techniques are performed.

8.5.2 Further Image Pre-processing Techniques

Further Image Pre-processing Techniques, similarly to [83], randomly crops all images from a resolution of 256×256 to a resolution of 224×224 pixels. This is to ensure that the input images have the same size as the images in well-known datasets such as ImageNet [83], which facilitates using networks developed for those datasets to be deployed for our DeepWeeds images. After all images are cropped, they are down-sampled to a resolution of 64×64, or 32×32 pixels. Their

Table 8.2: Color channel mean and standard deviation values used.

Image Channel	Mean	Standard Deviation
Red	0.485	0.229
Green	0.456	0.224
Blue	0.406	0.225

orientation is randomly rotated between -15 and +15 degrees. Their image brightness, contrast, and saturation are also randomly varied by 10%, and normalized between the values of 0 and 1. Finally, the color channels of each image are normalized using distinct mean and standard deviation values as seen in Table 8.2, which have demonstrated significant performance on the ImageNet dataset.

8.6 Network Architecture

The complete structure of the implemented networks consists of two main components: the software and the hardware. The software architecture defines the targeted neural network structures, which are described in C++ and OpenCL kernels. The hardware architecture describes the integration of the hardware used to run the OpenCL kernels, and a host controller, which is the program executed on a host processor to launch OpenCL kernels and to manage available memory.

8.6.1 Software Architecture

Three popular deep network architectures, i.e. VGG-16 [241], DenseNet-128-10 [453], with 128 layers and a growth rate of 10, and Wide Residual Network (WRN)-28-10 [258], with 28 layers and a growth rate of 10, were trained using full resolution and deterministically binarized weights for comparison. Although networks with tuned hyper-parameters would be expected to achieve higher validation accuracies, we present baseline implementations on our labeled dataset without any hyper parameter tuning. The mentioned networks are chosen as they demonstrate significant performance on the ImageNet dataset, which we believe to be indicative of high performance on the DeepWeedsX dataset. In favour of reproducible research, we have made the specific code level implementations of all these networks publicly available through a Github repository[1].

More details on our developed software network architectures are as follows. The output of each network's last layer is fed through a Softmax activation layer [445], and each network's loss is minimized using cross-entropy. Stochastic gradient descent with momentum [454] is used to optimize network parameters. For all networks, the momentum variable m, is set to 0.8. In our implementations, Binarized Neural Networks (BNNs) employ a smaller initial learning rate, $\eta[0] = 0.001$, to avoid frequent sign changes, while for conventional networks using full resolution weights a larger initial learning rate of $\eta[0] = 0.01$ is used to speed-up learning convergence. In addition, the regularization coefficient, $D = 5e^{-7}$ for our BinaryNet-regularization terms and $D = 1e^{-5}$ for our $\ell 2$-regularization terms. For all BNNs, ReLU activation functions, used in conventional networks, are replaced with the hyperbolic tangent function, $tanh$.

Furthermore, in order to maximize the networks' performance, a decaying learning rate is used during training for all networks. This learning rate, η, is decayed by a factor of ten when learning

[1] https://github.com/coreylammie/Low-Power-and-High-Speed-Deep-FPGA-Inference-Engines-for-Weed-Classification

falls stagnant, i.e. does not increase for a period of 10 epochs. Finally, the weights are randomly uniformly distributed using the He initialization technique presented in [455] to accelerate learning convergence for full resolution weights used for the parameter update stage.

8.6.2 Hardware Architecture

After functionally verifying our implementations using the `PyTorch` [231] ML library with a state-of-the-art *Titan V* GPU and an `AMD Ryzen 2700X @ 4.10 GHz Overclocked` CPU, we developed hardware architectures consisting of C++ host controllers and multiple OpenCL kernels, which were accelerated using FPGAs and GPUs. For x86-based systems, OpenCL accelerated kernels using FPGAs typically reside on an FPGA development board, which is connected to a separate independent host system through the PCIe express interface [440]. For ARM-based systems, the FPGA is typically connected to a Hard Processor System (HPS) on a SoC through specialized bridges – as in the case of the Intel DE1-SoC development board used herein. This allowed our proposed FPGA-accelerated networks to run completely independently using the SoC, without using a separate device for computation.

Our OpenCL implementations originate from the publicly available *DeepCL* OpenCL ML library[2]. All convolutional, inner-product, activation, pooling, regularization, and batch-normalization operations are described using single work-item kernels with an NDRange size of (1, 1, 1). Multi-mode 3D NDRange kernels are used to fetch and store data to and from the global memory for all computation pipelines, similarly to [440]. Consequently, our implementations operate with minimal controller computation. We make these efforts toward an eventual implementation that avoids controller computation completely, with motivations to make our future designs applicable to both GPUs and non-SoC FPGAs, avoiding the power overhead of the host controller, that is required for the OpenCL programming model. In the following Sections, we detail the different synthesis constructs, such as the loop unroll factor (#pragma unroll for GPU), and Single-Instruction-Multiple-Data (SIMD) vectorization factor, used for our developed single work-item kernels targeted for acceleration on heterogeneous platforms.

Convolutional Kernel

Convolutional operations were implemented by mapping 3-D convolutions as matrix multiplication operations, by flattening and rearranging the input features, similarly to [441]. Each work-item performs either fused MAC operations, or accumulate operations, on the local memory data, depending on whether weights are quantized to binary states or not. This process is accomplished using the loop unrolling technique, and is repeated by sliding the convolutional window to get the corresponding elements in the product matrix. We further detail the XNOR kernels, described using RTL modules, utilized in our FPGA BNN implementations for convolutional and Fully Connected (FC) kernels in Section 8.6.3.

[2] `https://github.com/hughperkins/DeepCL`

Inner-product Kernel

Inner-product operations were implemented for FC layers using single work-item kernels, where, similarly to our convolutional kernel, fused MAC, or accumulate operations are performed, depending on whether weights are quantized to binary states or not.

Activation Kernel

All activation functions were computed at the output of convolution and inner product implementations using single work-item kernels.

Pooling, Regularization, & Batch-normalization Kernels

Pooling, regularization, optimization, and batch normalization operations were implemented using single work-item kernels, where acceleration is achieved by unrolling the loop to generate parallel outputs in a single cycle.

8.6.3 Platform Specific Compilation & Performance Enhancing Techniques

In this Section we detail the platform specific compilation details, libraries, and resources required for compilation and deployment of our developed host controller and OpenCL kernels, for FPGA and GPU platforms.

Both our GPU and FPGA implementations of BNNs are accelerated using XNOR kernels, which enables SIMD within a register (SWAR) [446]. Here, full precision 32-bit floating point weights are concatenated into groups of 32 binary variables into 32-bit registers, resulting in a 32-time speed-up on bitwise operations.

FPGA

To compile OpenCL kernels for FPGA, the IOC was used, as part of the *Intel FPGA SDK for OpenCL* and *Quartus Prime Design Suite* 18.1. IOC fully supports version 1.0 of the OpenCL specification, and has some preliminary support for newer features from version 2.0. Fig. 8.2 presents the compilation flow for the IOC. Here, inputs are a set of OpenCL kernels, and the output is a singular $.aocx$ image file containing the necessary information to program the FPGA at runtime containing the FPGA image. The host application loads data that is used to create program objects, to program the target FPGA, as required for all kernel launch operations.

SIMD Within a Register (SWAR) was also implemented using a digital logic approach, similarly to [8], to accelerate our FPGA BNN implementations. Fig. 8.3 illustrates SIMD using 32-bit registers. Using RTL modules, the given size of each register to store binary weights is arbitrary and not constrained by the computer architecture. These instructions, therefore, take only four clock cycles on FPGAs.

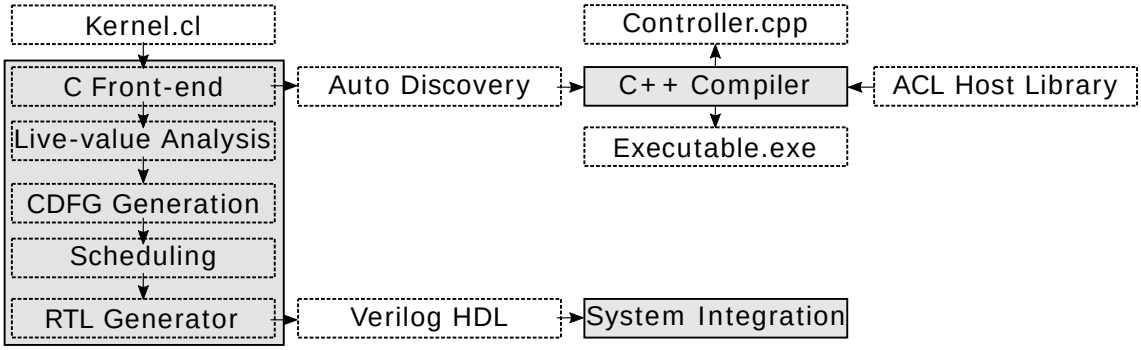


Figure 8.2: Compilation flow for the IOC for our OpenCL implementations on FPGA.

We integrate our developed XNOR RTL module into the Intel FPGA SDK for OpenCL Pipeline using the IOC. Our RTL module has a balanced latency, where its threads match the number of pipeline stages in our design. This allows the threads of the RTL module to execute without stalling the SDK's pipeline and bottlenecking operational performance.

GPU

In addition to accelerating the targeted DeepWeedsX recognition networks on the Intel DE1-SoC FPGA development board, the networks were also accelerated on a state-of-the-art *Titan V* GPU to execute OpenCL kernels and an *AMD Ryzen 2700X @ 4.10 GHz Overclocked* CPU to drive the host controller. We use version 419.35 of the Titan V GPU driver to launch compute kernels. Using SWAR OpenCL kernels, on GPUs, it is possible to evaluate 32 network connections with only 3 instructions (accumulation, popcount, and XNOR), as described in (8.6).

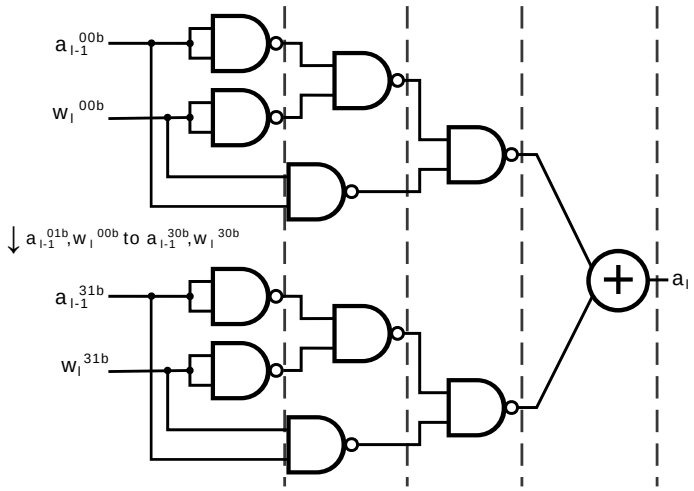


Figure 8.3: Schematic of the XNOR kernel implementation on FPGA with 32-bit registers containing 32 binary variables. Here, vertical lines represent clock cycles. a_{l-1} contains binary activations, and w_l contains binary weights.

$$a_l+ = \text{popcount}(\text{xnor}(a_{l-1}^{32b}, w_l^{32b})). \tag{8.6}$$

Here, l denotes the layer number, a_l is the resulting weighted sum, and a_{l-1}^{32b}, w_l^{32b} are the concatenated inputs and weights. These instructions take 6 overall clock cycles, including 1 for accumulation, 4 for popcount, and 1 for XNOR, on the *NVIDIA Titan V* GPU used [456].

While NVIDIA's CUDA compiler is, presently, much more efficient and mature than their OpenCL compiler, to advocate fair comparison, we use OpenCL implementations across FPGA and GPU platforms. We note that enhanced timing performance is expected on NVIDIA GPUs using CUDA alongside with NVIDIA's Deep Neural Network library (cuDNN).

8.7 Investigating The effect of Image Down-sampling and Preprocessing on Performance

Before gathering implementation results, VGG-16 [241], DenseNet-128-32 [453], and WRN-28-10 [258] were trained using full resolution weights with a batch size, $\Im = 32$, for input images down-sampled to three different sizes. We restrict the batch size to 32 and below due to the harsh real-time constraints presented by our specific edge computing use case for robotic weed control, which is discussed further in Section 8.9.2. These sizes include dimensions of 32×32, 64×64, and the original size of 224×224. This was performed to investigate the down-sampling effect on the degradation of our chosen deep networks validation accuracy. Figs. 8.4, 8.5, and 8.6 demonstrate the validation accuracy of all networks over 200 training epochs. While other works report results over a 500 epoch training routine, we observed that learning on the training set was saturated at around 150 epochs and therefore report results over 200 epochs. The maximum validation accuracy achieved for each implementation is presented in Table 8.3. It is worth noting

Table 8.3: Maximum validation accuracy achieved during 200 epochs of training for all baseline implementations used to investigate down-sampling performance degradation.

Network Architecture	**(3, 32, 32)**	**(3, 64, 64)**	**(3, 224, 224)**
	IPT		
VGG-16	**86.72**	89.48	91.08
DenseNet-128-32	**90.08**	91.52	89.40
WRN-28-10	**88.88**	93.36	94.82
	FIPT		
VGG-16	**81.45**	89.12	93.04
DenseNet-128-32	**85.89**	86.05	94.24
WRN-28-10	**85.97**	90.72	95.85

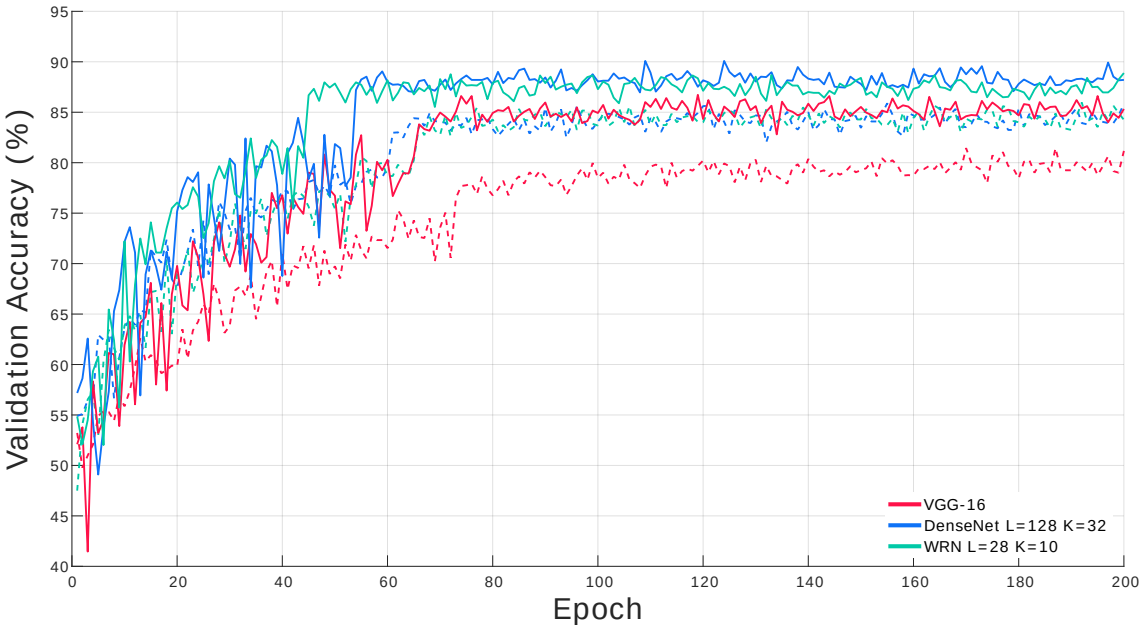


Figure 8.4: Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 32, 32). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

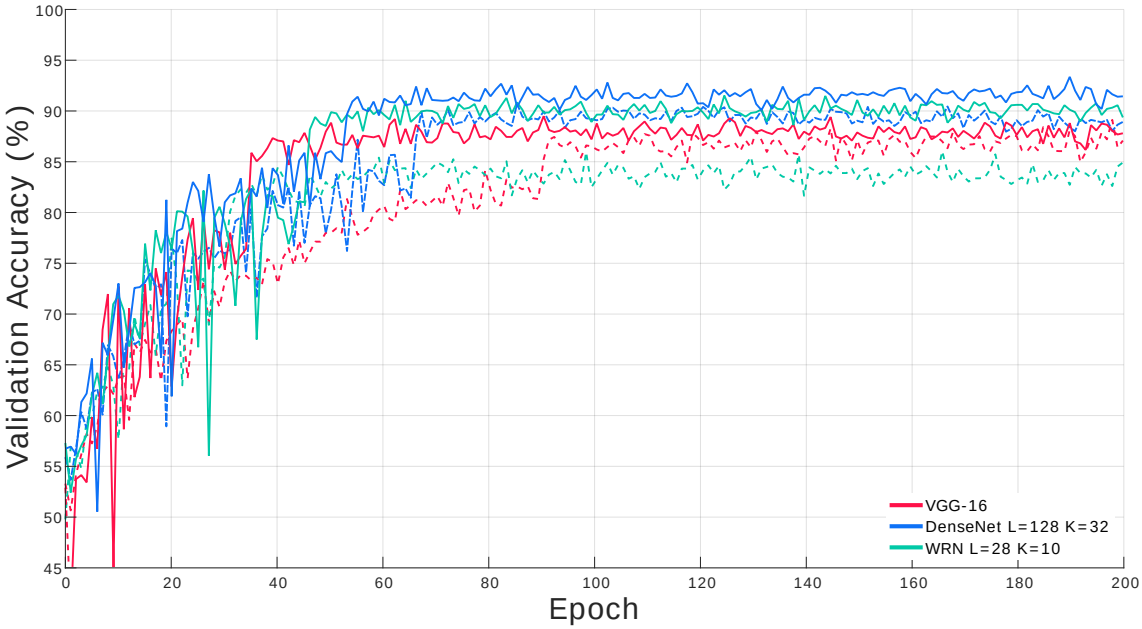


Figure 8.5: Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 64, 64). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

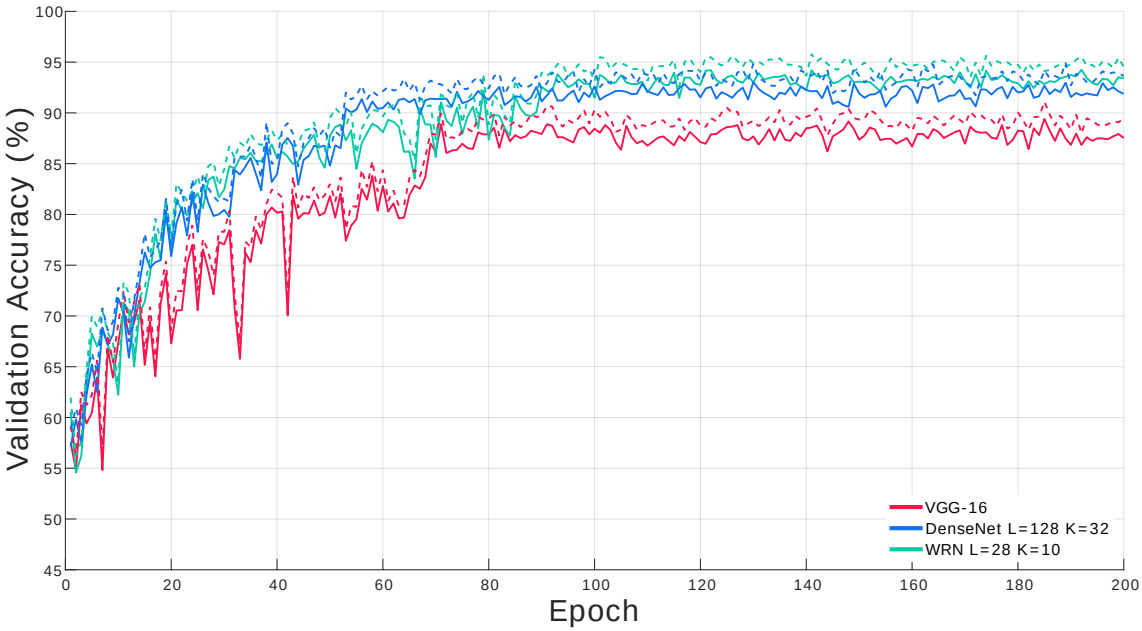


Figure 8.6: Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 224, 224). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

that, for the networks shown in Table 8.3, no hyper-parameter optimization is performed. In addition, for the FIPT cases, we replicated our image pre-processing steps originally proposed in our previous work [416], which demonstrated significant performance for input images with a resolution of 224×224. It is expected that, hyper-parameter optimization, different types or amounts of image pre-processing techniques, or a combination of them could lead to validation accuracy improvement.

From Figs. 8.4, 8.5, and 8.6, it can be observed that down-sampling input images to (3, 32, 32) leads to performance degradation, compared to using images with (3, 64, 64) or those with a native resolution of (3, 224, 224). In addition, the figures show that for low-resolution images, i.e. (3, 32, 32) and (3, 64, 64), performing FIPT leads to lowering accuracy. FIPT is useful in the case of the native resolution images and leads to the highest accuracy achievable.

Interestingly, it is possible to achieve >85% validation accuracy using down-sampled input images with IPT at (3, 32, 32), which can significantly improve processing speed and reduce the total required memory utilization during inference, compared to the use of higher resolution images. Therefore, we down-sample all images to (3, 32, 32) to obtain all our implementation results reported in the following sections. This is expected to drastically reduce the training time required, as well as the resource utilization for all networks, which have previously been demonstrated to be governed by the total number of trainable network parameters, input image size, and network configuration [457].

Table 8.4: Total number of trainable network parameters for each network architecture.

Network Architecture	(3, 32, 32)	(3, 64, 64)	(3, 224, 224)
VGG-16	33,642,569	39,934,025	49,225,881
DenseNet-128-32	7,727,065	7,757,737	8,217,817
WRN-28-10	36,478,553	36,478,553	36,478,553

Table 8.5: Implementation results obtained from our baseline implementations using full resolution weights, trained over 200 epochs with down-sampled (3, 32, 32) images. During inference $\Im = 32$.

Baseline Network Architecture	Total Kernel Power Usage (W)	Inference Time per Image (ms)	Validation Accuracy (%)
	$\Im = 32$		
VGG-16	41.00	5.038	86.72
DenseNet-128-32	41.00	4.398	90.08
WRN-28-10	44.67	2.535	88.88

Table 8.4 presents the total trainable network parameters for each network architecture used to obtain our implementation results. For VGG-16 and DenseNet-128-32 architectures, additional parameters are required for larger resolution input images to implement additional max pooling and fully connected layers. However, the WRN-28-10 network parameter size shows no dependency to the input image resolution, which could be attributed to its wide (not deep) structure.

8.8 Implementation Results

Initially, conventional networks trained using full resolution weights were all implemented on the GPU platform (see Table 8.3). This was to have as a baseline for validation accuracy comparison to both previous works and our binarized implementations. Direct comparison to the relevant previous work [416] is not possible using the given labeled dataset (DeepWeedsX), because, in [416] five-fold cross-validation is used to report accuracy, while here we obtained accuracy using a train-test dataset split. Nonetheless, the best validation accuracy achieved here (95.85%) using WRN-28-10 with FIPT, is marginally better than the best accuracy previously achieved using ResNet-50 with FIPT (95.7%) as reported in [416].

In order to validate and investigate the performance of the proposed FPGA- and GPU-accelerated BNN architectures on the DeepWeedsX dataset, the validation error rate, power consumption, and Inference Time Per Image (ITPI) were analyzed. On both FPGA and GPU platforms, all aforementioned performance metrics were determined during inference, i.e. after importing trained weights, for different batch sizes $\Im \in [4, 8, 16, 32]$ to investigate the effect of the batch size on inference performance. As discussed earlier, we restrict the batch size to between 4 and 32 due to the harsh real-time constraints presented by our specific edge computing use case for robotic weed

control (see Section 8.9.2). For FPGA implementations we also report the resource utilization, which is an important parameter in identifying hardware cost.

The total kernel power usages were determined using the *Post Place & Route Estimator* for FPGA post-synthesis, and *NVIDIA-SMI* for GPU. To ensure all reported power usage readings are accurate for our GPU implementations, we artificially elongate kernel execution times when measuring GPU kernel power utilization.

It is worth noting that, there is no need to consider the time required for image pre-processing because for all (3, 32, 32) cases, networks employing IPT outperform their FIPT counterparts (see Table 8.3). It is expected that during real-time inference, images are fed directly to the inference engine after undergoing pipelined real-time image resizing similar to [458], hence, requiring no down-sampling (IPT). There is also no need for FIPT, because FIPT is only useful for (3, 224, 224). We used images of size (3, 32, 32).

8.8.1 Baseline Implementations

Baseline implementations of VGG-16 [241], DenseNet-128-32 [453], and WRN-28-10 [258], were trained using full resolution weights with $\Im = 32$, on GPU for images of size (3, 32, 32). The collected performance metrics are reported in Table 8.5. It can be observed that DenseNet L=128, K=32 achieves the highest validation accuracy of 90.08%. It requires 0.64 ms less inference time per image, compared to its VGG-16 variant, while consuming the same amount of power and yielding an increase 0f 3.36% in validation accuracy. Compared to WRN-28-10 network, the

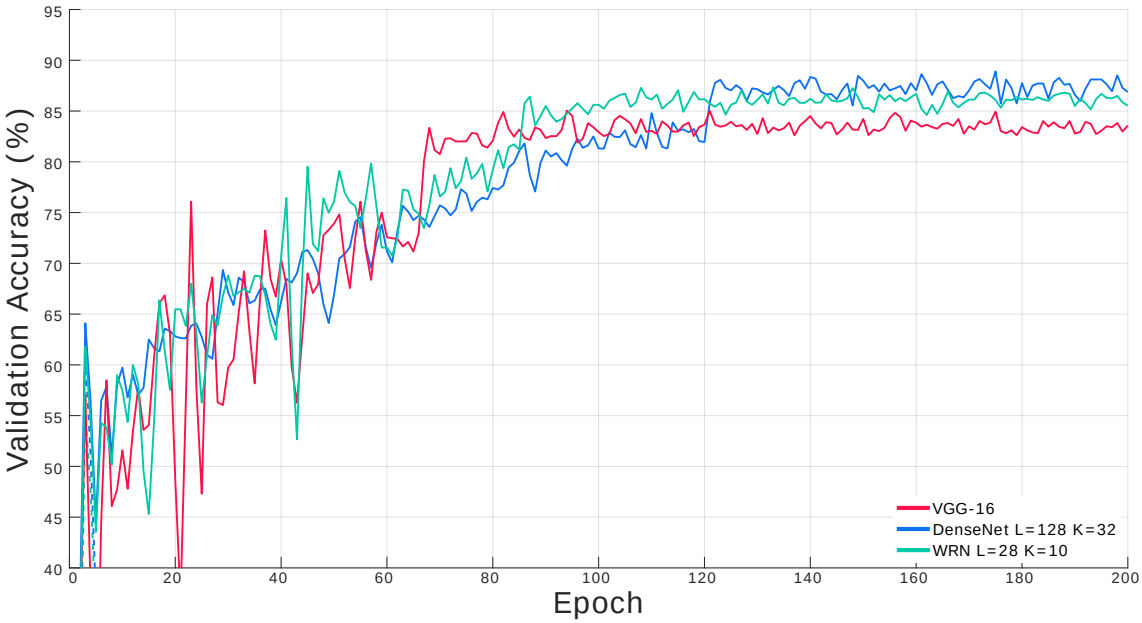


Figure 8.7: Validation accuracy after each epoch during 200 training epochs for our new binarized implementations with down-sampled (3, 32, 32) images.

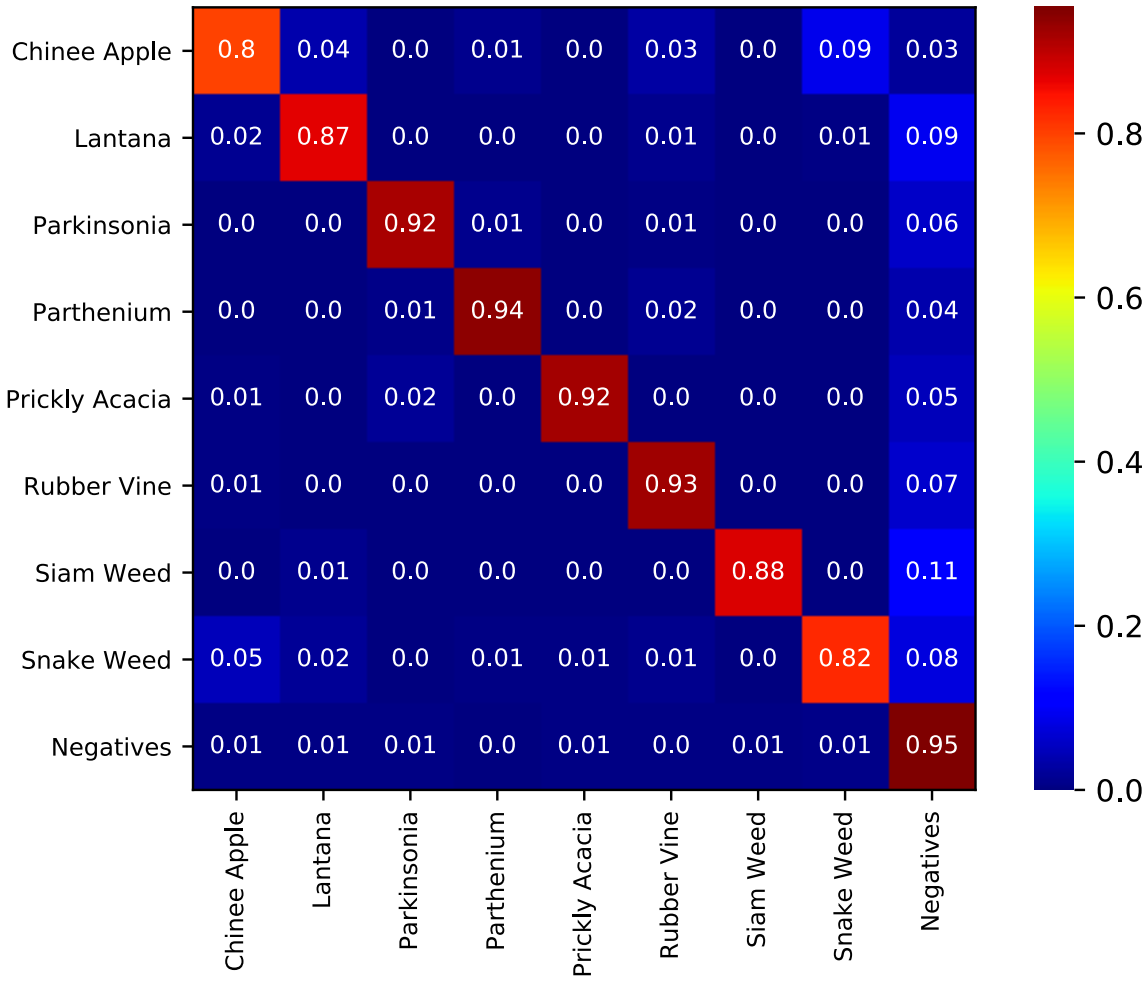


Figure 8.8: Confusion matrix obtained using the DenseNet L=128, K=32 with baseline implementation. Performance is described for each individual class. Values are normalized from 0 to 1 corresponding to 0% and 100% validation accuracy, respectively.

DenseNet L=128, K=32 achieves 1.2% improvement in accuracy, while being 1.86 ms slower in image inference, but consuming 3.67 W less power.

Furthermore, in order to determine class specific performance, a confusion matrix for the top performing baseline implementation, DenseNet L=128, K=32, is presented in Fig. 8.8. It can be observed that the individual class accuracy is weakly correlated to the class distribution in the DeepWeeds dataset. Further species-specific performance discussion is presented in Section 8.9.1.

8.8.2 Binary Implementations

We implement and train our new binary networks of the selected deep architecture across GPU and FPGA platforms. The validation accuracy for each epoch during training for each implementation is presented in Fig. 8.7. As all networks are trained using the same `NVIDIA Titan V` GPU, only one plot for each network is presented. Consequently, the validation accuracies and corresponding

Table 8.6: Implementation results obtained from our new binarized implementations with down-sampled (3, 32, 32) images, over 200 epochs.

Binary Implementation		VGG-16				DenseNet-128-32				WRN-28-10			
		$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$	$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$	$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$
Total Kernel Power Usage (W)	GPU	46.16	47.10	47.39	47.69	56.78	56.83	57.47	58.71	55.64	56.68	56.72	58.07
	FPGA	4.19	4.58	5.31	6.26	4.48	4.72	5.40	6.72	4.01	4.49	5.26	6.04
Inference Time per Image (ms)	GPU	9.87	5.55	4.5	3.3	7.47	3.93	2.88	2.43	4.83	2.98	2.38	1.49
	FPGA	6.647	3.843	2.974	2.199	4.762	2.689	2.045	1.539	3.268	2.02	1.539	1.059
Validation Accuracy (%)		85.05				88.91				87.33			

Table 8.7: Device utilization (%) comparison for the implemented FPGA-accelerated BNNs adopting IPT. All reported hardware utilization numbers are expressed as percentages of the total available resources on the FPGA. [1]The maximum frequency for each implementation was extracted from *acl_quartus_report.txt* report, generated by the Quartus Design Studio.

Binary Implementation	VGG-16				DenseNet-128-32				WRN-28-10			
	$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$	$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$	$\Im = 4$	$\Im = 8$	$\Im = 16$	$\Im = 32$
Flip Flops	33.34%	48.49%	68.72%	96.17%	28.09%	36.82%	71.49%	74.67%	30.17%	43.07%	58.63%	88.96%
ALMs	58.32%	61.91%	87.46%	96.30%	35.22%	46.94%	78.49%	92.92%	50.26%	53.29%	79.74%	88.89%
DSPs	74.11%	80.14%	85.42%	95.20%	34.73%	47.26%	53.53%	68.81%	55.92%	75.29%	84.13%	90.35%
Frequency [MHz][1]	52	46	43	39	58	49	43	41	51	45	44	41

confusion matrices are identical across platforms.

From Fig. 8.7, it can be observed that, similar to our baseline implementations adopting full resolution weights, binary DenseNet L=128, K=32 achieves the highest validation accuracy, with a degradation of only 1.17% compared to its full-resolution counterpart.

In addition, Table 8.6 reports the total kernel power usages and inference times for both GPU and FPGA implementations for various batch sizes of $\Im \in [4, 8, 16, 32]$. Table 8.6 demonstrates that as the batch size, $\Im$, is increased, the inference time per image is notably decreased. We believe this is a a direct result of increased parallelism.

For all networks, the FPGA implementations require less time to perform inference despite operating at a much lower operational frequency. Our DenseNet L=128, K=32 implementation on FPGA reduces the inference time per image compared to its GPU counterpart by 0.89 ms for $\Im = 32$ while achieving the same validation accuracy, and by 2.86 ms compared to its baseline implementation time reported in Table 8.5. These results and their implications are discussed in Section 8.9.2.

Furthermore, in order to determine class specific performance of binarized networks on GPU and FPGA, a confusion matrix for the top performing implementation, DenseNet L=128, K=32 is presented in Fig. 8.9. Interestingly, the class specific performance for binarized DenseNet varies significantly across various weed species, while degrading only 1.17% overall. Section 8.9.1 provides further discussion on species-specific performance.

In addition, to investigate the hardware complexity of the implemented binarized networks on FPGA and compare it to the achieved inference and power consumption figures, the device uti-

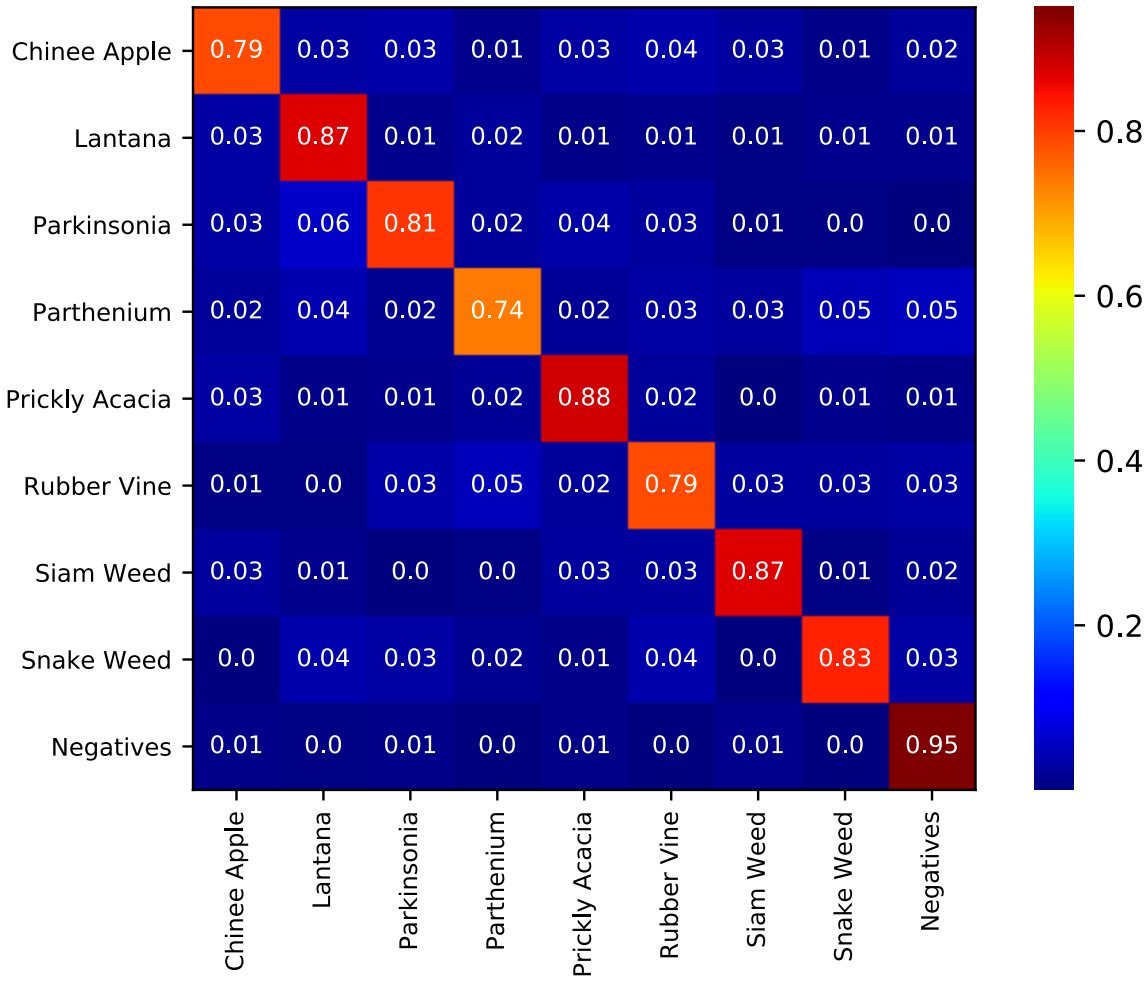


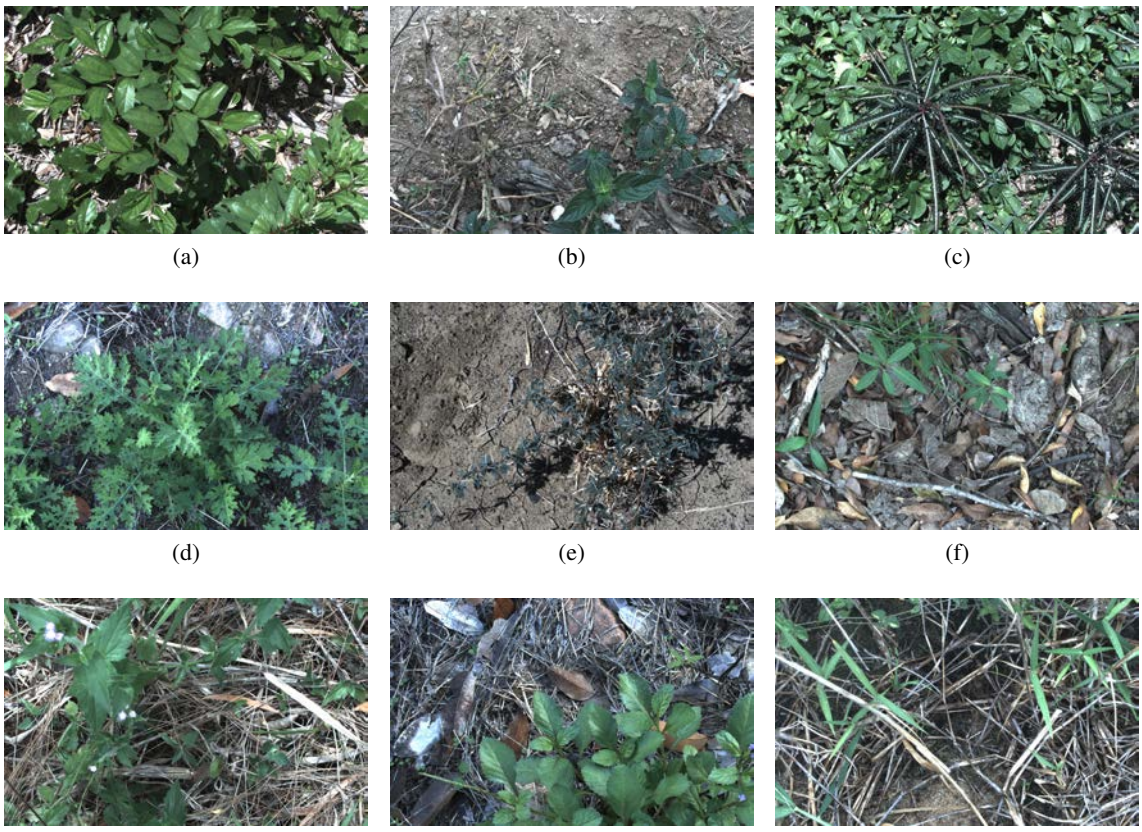
Figure 8.9: Confusion matrix obtained using the binary DenseNet L=128, K=32 with IPT. Performance is described for each individual class. Values are normalized from 0 to 1 corresponding to 0% and 100% validation accuracy, respectively.

lization of all FPGA accelerated networks were measured and presented in Table 8.7.

From Table 8.7 it can be observed that the device utilization for each network architecture is weakly correlated with the batch size during inference. For all our implementations, as $\Im$ is increased, the Flip Flops, ALMs, DSPs required for synthesis are increased and the maximum synthesizable frequency is decreased. Our best performing implementation, DenseNet L=128, with $\Im = 32$, requires 46.58% more Flip Flops, 57.7% more ALMs, and 34.08% more DSPs than its $\Im = 4$ counterpart.

8.9 Further Discussion

In this Section, we provide further discussion of our implementation results and how they benefit the application of robotic weed control.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 8.10: Example images from each class of the *DeepWeeds* dataset, including: (a) Chinee apple, (b) Lantana, (c) Parkinsonia, (d) Parthenium, (e) Prickly acacia, (f) Rubber vine, (g) Siam weed, (h) Snake weed and (i) Negatives. Chinee apple and Snake weed are prone to high levels of confusion due to their very similar features.

8.9.1 Classification Performance

Tables 8.5 and 8.6 show that our baseline and binary implementations of the DenseNet architecture consistently outperform VGG-16 and WRN-28-10. With images drastically down-sampled to (3, 32, 32), our full precision baseline DenseNet-128-32 implementation offers a validation accuracy of 90.1% on the DeepWeedsX dataset. This compares well with the ResNet-50 architecture used in [416] to achieve 95.7% performance on much larger images, 224×224 pixels in size.

However, validation accuracy is an unreliable sole metric due to the imbalanced nature of the DeepWeeds and DeepWeedsX datasets. In the application of robotic weed control, the goal is to maximize coverage of weed targets at the cost of collateral off-target damage. As such, it is vitally important to consider the performance on each individual species. The confusion matrix of the baseline implementation of DenseNet, presented in Fig. 8.8, is analogous to the classification performance in [416]. The species with the highest recall accuracy, ranging from 87-95%, include: Lantana, Parkinsonia, Parthenium, Prickly acacia, Rubber vine, Siam weed and negative plant life. The species presenting the most difficult challenge for the network, ranging from 80-82% recall accuracy, are Chinee apple and Snake weed. With the model confusing 9% of Chinee apple as Snake weed, and 5% vice versa, we reason their respective poor performance is due to the inherent similarity in the two plants image features. The similarity between these two classes is made evident in Fig. 8.10, which presents an example image of each class.

Fortunately, confusing one weed target for another is inconsequential in the application of robotic weed control. However, missed targets (i.e. false negatives) and off-target damage (i.e. false positives) are of great consequence so we must examine the performance on the negative class. The baseline DenseNet architecture confuses between 3-11% of each species with the negative class. This constitutes a significant number of missed targets. Also, 5% of the negative class is falsely classified as positive species. The existence of false positives is assumed to be the result of the massive variation of plant life in the negative class.

Table 8.6 reveals that the best performing binarized implementation is again the DenseNet-128-32 architecture, which offers 88.9% average classification accuracy. Interestingly, the inter-species performance is vastly different to the baseline implementation, as shown by the confusion matrix in Fig. 8.9. Our binarized implementation of the DenseNet architecture appears to have generalized the classification performance of the network across species. Confusions between species are more scattered and are no longer owing to visibly discernible characteristics. The species offering the best performance, ranging from 83-88%, include: Lantana, Prickly acacia, Siam weed and Snake weed. While the species presenting the most difficult challenge, ranging from 74-81% are: Chinee apple, Parkinsonia, Parthenium and Rubber vine. We believe that the extreme quantization of weights to binary states prevented less-distinctive features to be extracted, causing a degradation in validation accuracy for weed species with a large number of features. Species with distinctive features, or lack thereof, demonstrated similar accuracies to our baseline implementations. For example, in both our best performing implementations the negative class is strongly classified at

95% with 5% false positives.

Tables 8.5 and 8.6 also show that the performance of all binarized implementations are slightly worse than their full precision baseline counterparts. This result confirms what is seen in literature. However, the real-time performance of these binarized networks are far greater than their full precision counterparts and present a fruitful tradeoff for the application of robotic weed control, as discussed below.

8.9.2 Real Time Performance

Tables 8.5 and 8.6 show that every novel binarized implementation presented here significantly outperforms its GPU baseline implementation in terms of inference time and required power, with only a slight degradation in validation accuracy. The WRN architecture offers the fastest inference engine with a 1.018 ms inference time when implemented on an FPGA. While DenseNet-128-32, our most accurate architecture, also offers a significant increase in speed, with an average inference time of 1.539 ms per image.

In addition to the binarization of full precision architecture and acceleration using FPGA hardware, two further methods of pushing the real-time performance were investigated: presenting a smaller image size to the network, and reducing the amount of pre-processing or image augmentations performed. Table 8.3 reveals that down-sampling the image size from (3, 224, 224) to (3, 32, 32) only slightly decreases the average validation accuracy from 94.24% to 90.1% for DenseNet, 93.04% to 86.7% for VGG-16 and 95.85% to 88.9% for WRN. The significantly smaller image size of (3, 32, 32) is a major reason these networks perform inference faster, compared to the (3, 224, 224) shaped architectures in [416].

Table 8.3 also shows that applying further image pre-processing techniques offers no advantage in classification performance for the utilized down-sampled images. In fact, we observed that validation accuracy is degraded by an average of over 4% for the binary implementations when images are down-sampled below (3, 224, 224) when further image augmentations are applied. This suggests that our FIPT are ill-suited to low resolution images and can only be beneficial to large-resolution images, which we do not use here. However, as discussed in Section 8.7, these results are expected to be significantly improved if further image preprocessing technique investigation, and/or hyper-parameter optimization is performed, for each network, and each input image size.

Let us consider the use case of the prototype agricultural robotic spot-sprayer, AutoWeed, first introduced in [416]. Its optical system comprises four FLIR Blackfly 23S6C high-resolution cameras, each providing a 450 x 280 mm field of view with a maximum data rate of 41 fps. This allows a threshold of at most 100 ms total processing time per image per camera for the selective spot sprayer to operate at the target speed of 10 km/hr. This imposes a required frame rate of at least 10 fps per camera to achieve target real-time performance. With simultaneous image acquisition from four cameras, batch sizes of between 4 and 32 were considered for our above implementations. The lower limit of 4, considers processing one frame at a time from each

camera. While the upper limit of 32, considers waiting for the acquisition of up to 8 frames from each camera before processing them in a batch for the apparent per image inference speed increase.

With an average inference time of 1.539 ms implemented on an FPGA, the 450×280 mm field of view of our specific edge device can be processed fast enough to achieve a frame rate of over 600 fps. Far exceeding the real time requirement for one camera at 10 km/hr. This efficiency would also allow the robot to operate at a much higher vehicle speed to yield more efficient performance in the agricultural domain.

Compared to existing full precision architectures and their power-hungry GPU implementation [416], the low-power and high-speed inference engines presented here offer an attractive tradeoff with slightly worse classification performance for greatly increased speed and power efficiency. This tradeoff will allow researchers and developers to solve the speed and power inefficiencies in applications of precision agriculture, like robotic weed control, with software instead of hardware. The proposed binarized solutions can also be generalized for improving the efficiency of edge computing in general, where slight amounts of accuracy can be traded off for great amounts of speed and power improvement.

8.10 Conclusion

In this Chapter, we are the first to investigate the acceleration of binarized DNNs on GPUs and FPGAs using the high-level OpenCL framework for weed species classification targeted toward edge computing applications and robotic weed control. We investigated the performance degradation exhibited when down-sampling input images, and demonstrated that significantly reducing the image resolution has a marginal effect on validation accuracy. After thoroughly comparing efficient implementations on GPU and FPGA platforms, we were able to achieve a $>$7-fold decrease in power consumption, while performing inference on weed images 2.86 times faster while degrading validation accuracy by only 1.17% on our newly labeled and publicly available dataset. Finally, we provided further discussion pertaining to species-specific classification performance and real time performance implications for robotic weed control. The binarized DNNs in this Chapter represent ideal candidates for future implementation in precision agricultural robots using edge computing devices which operate at reduced precision, including existing devices, such as Google's TPU [93] or IBM's Artificial Intelligence Unit (AIU) [459], and future devices capable of reduced precision operation, which are envisioned to burgeon in popularity [35].

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, simulations and implementations of novel DL hardware architectures for resource-constrained devices were presented. Two specific technologies were investigated: memristors, and FPGAs. In Chapter 2, both neuromorphic brain-inspired asynchronous and traditional synchronous DL architectures were investigated for healthcare and biomedical applications. It was determined that novel hardware architectures could greatly reduce the power consumption and latency of neuromorphic processors and DL inference accelerators for numerous signal processing, and image classification and segmentation tasks.

Additionally, it was determined that neuromorphic processors and traditional DL accelerators are complimentary in nature, and that for some specific tasks, especially those for which data is collected asynchronously, neuromorphic processors can outperform DL accelerators. However, for many complex practical engineering tasks, DL accelerators are required to achieve significant performance. For DL accelerators, in particular, memristors and FPGAs were identified to be strong candidates for deployment in novel DL system architectures. Finally, an outlook and future perspectives were provided.

In Chapter 3, modeling and simulation efforts related to emerging memristive devices were over-viewed, and simulation frameworks were investigated and compared. It was determined that modernized frameworks, which adopt a sophisticated software engineering approach, offer significant benefits over conventional SPICE simulation techniques. The run-time and memory usage of all modernized open-source frameworks were compared when possible; both for inference and training simulations. Finally, similarly to as done in Chapter 2, an outlook and future perspectives were provided.

In Chapter 4, the first and third research questions were addressed. A novel hybrid CMOS-memristive architecture was presented, which was capable of executing convolutional layers using multiple crossbar tiles. In addition, a novel mitigation strategy was proposed for stuck at level RRAM device faults. Comprehensive simulations of this architecture in a 22nm FDSOI CMOS process were conducted in two different modes of operation with different latency and resource requirements, and it was determined that SOTA performance could be achieved for both epileptic

seizure detection and prediction tasks.

In Chapter 5, the first research question was addressed. Non-idealities of CBRAM devices were exploited to generate pseudo-random numbers efficiently by characterizing the stochastic switching behavior between low and high resistance states, in order to perform DL parameter optimization using SC. It was demonstrated that the size of MAC units could be reduced by 5 orders of magnitude. Comprehensive simulations of this architecture in a 40nm CMOS process were conducted, which determined that the architecture was capable of performing parameter optimization for a CNN while it is being trained for a character recognition task and consuming less than 100μW.

In Chapter 6, the second research question was addressed. MemTorch, a novel open-source simulation framework for MDLSs, was developed with a refined focus on modeling device- and circuit-level non-idealities. Exemplar simulations were conducted using the MobileNetV2 architecture for the CIFAR-10 image classification task, and it was demonstrated that MemTorch could be used to perform design exploration and quantify trade-offs between resource usage and performance during development, prior to exhaustive transistor-level simulation and validation.

In Chapter 7, the second research question was also addressed; albeit with a refined focus on two specific device characteristics: endurance and retention. A novel generalized empirical Metal-Oxide RRAM endurance and retention model designed for use in large-scale DL simulations was presented. This model was the first to unify retention and endurance modeling while taking into account time, energy SET-RESET cycles, device size, and temperature. It was compared to other related models and used in large-scale simulations using the MobileNetV2 architecture for the CIFAR-10 image classification task; the same task used to demonstrate the operation of MemTorch in Chapter 6. It was determined that, even when ignoring other device non-idealities, retention and endurance losses significantly affect the performance of DL networks. Consequently, they should be accounted for during the development of hybrid CMOS-memristive architectures for DL systems.

Finally, in Chapter 8, the first and third research questions were addressed. Novel FPGA deterministic BNN accelerators were presented, which were tailored towards weed species classification. It was demonstrated that these networks significantly outperformed equivalent GPU accelerated networks, with minimal reduction in accuracy. Specifically, a $>$7-fold decrease in power consumption was reported, with a 2.86 times improvement in processing speed using the DeepWeeds dataset, which includes close to 18,000 weed images.

Overall, this thesis has investigated many different novel hardware architectures for DL system acceleration on resource-constrained devices using software hardware co-optimization. A variety of different computing approaches were investigated. Contributions have been made in many areas, ranging from the development of customized RTL designs for FPGA architectures, device-level modeling for RRAM devices, to DL parameter optimization using SC by exploiting the stochastic behavior of CBRAM devices, which has traditionally been treated as a hindrance.

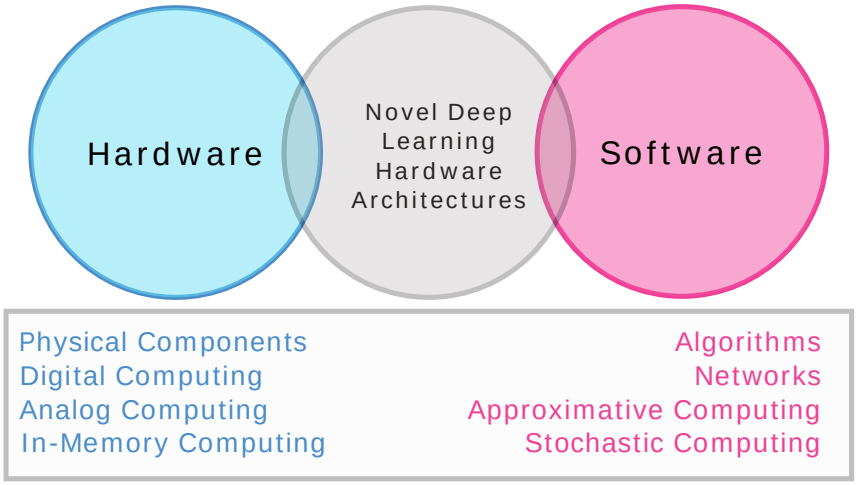


Figure 9.1: Both hardware and software innovations can be used synergically to develop novel DL hardware architectures. This figure was inspired by [35].

Clearly, there is much more work needed to be done in this exciting interdisciplinary area.

9.2 Future Work

As stated in the Introduction, the primary motivation of this thesis was to contribute, in-part, to the collective effort of reducing the power consumption and latency of DL accelerators to enable their use in resource-constrained devices. Both hardware and software improvements were investigated, and used to develop novel hardware architectures using two alternate technologies: memristive devices, and FPGAs, which were simulated and/or physically realized.

In this thesis, two types of memristive devices were investigated in detail: Metal-Oxide RRAM and CBRAM. Many other memristive device types, such as PCM and STT-MRAM, are also actively being researched [460], which could potentially be the key focus of future work. In addition to more fundamental device-level, circuit-, architectural-, and simulation-level research is being performed [42]. FPGAs, which were also investigated in this thesis, are being actively used by many researchers to reduce the power consumption and latency of DL accelerators in resource-constrained environments [440].

HLS, which was investigated in Chapter 8, has recently attracted significant attention in particular, as it can be used to significantly reduce the development time of digital circuits described using HDLs [133]. FPGAs are commonly used to prototype digital circuits prior to eventual implementation using ASICs. Due to limited resources, in this thesis, synthesis (using a specific CMOS process) and fabrication of novel digital hardware architectures was not performed. The implementation of digital circuits presented in this thesis using ASIC could potentially be an additional key focus of future work.

While both top-down and bottom-up design approaches can be used to develop hardware ar-

chitectures for efficient hardware implementation [461], in this thesis, a top-down approach was used, so bottom-up design approaches could also be an additional key focus. All research findings, and future research directions of thesis, can be broadly categorized using Fig. 9.1. As Electronic Design Automation (EDA) tools continue to improve, and hardware architectures become increasingly complex, software-hardware co-optimization approaches are becoming more popular and widely adopted [462]. As can be seen in this figure, both hardware and software innovations can be used synergically to develop novel DL hardware architectures. To conclude, some specific future directions include, but are certainly not limited to:

- The synthesis and eventual tape-out of the presented novel hybrid Memristive-CMOS hardware architectures using ASIC with BEOL device integration. This would enable the simulations of different hardware architectures to be validated for a specific CMOS process and for the architectures to be physically deployed;
- Further investigation of neuromorphic brain-inspired asynchronous processors for online and continuous learning applications. While in this thesis, the scope was confined to accelerating DL architectures, recent breakthroughs in the neuromorphic engineering domain, which approximate the back-propagation algorithm, have made neuromorphic processors a viable candidate for many of the investigated applications;
- Further investigation of novel computing architectures for more complex DL systems, such as recurrent-based networks and transformer architectures. Recent advancements in the DL domain, such as the attention mechanism, have greatly improved the efficacy of these systems. However, they require more complex circuitry to implement using ASIC;
- Further architectural improvements using software-hardware co-optimization. While some quantization and approximate computing techniques such as parameter quantization, alternative parameter mapping schemes, and stochastic computing were explored, they were not done so exhaustively.

Bibliography

[1] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1138–1159, Dec. 2020.

[2] C. Lammie, W. Xiang, and M. Rahimi Azghadi, "Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts," *Array*, vol. 13, p. 100116, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005621000540

[3] C. Li, C. Lammie, X. Dong, A. Amirsoleimani, M. R. Azghadi, and R. Genov, "Seizure detection and prediction by parallel memristive convolutional neural networks," *IEEE Transactions on Biomedical Circuits and Systems*, 2022.

[4] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1650–1654, 2021.

[5] C. Lammie, W. Xiang, B. Linares-Barranco, and M. Rahimi Azghadi, "MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems," *Neurocomputing*, vol. 485, pp. 124–133, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222002053

[6] C. Lammie, M. Rahimi Azghadi, and D. Ielmini, "Empirical metal-oxide RRAM device endurance and retention model for deep learning simulations," *Semiconductor Science and Technology*, vol. 36, no. 6, p. 065003, Jun. 2021.

[7] C. Lammie, A. Olsen, T. Carrick, and M. R. Azghadi, "Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge," *IEEE Access*, vol. 7, pp. 51 171–51 184, 2019.

[8] C. Lammie, T. J. Hamilton, A. van Schaik, and M. R. Azghadi, "Efficient FPGA implementations of pair and triplet-based STDP for neuromorphic architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1558–1570, apr 2019. [Online]. Available: https://doi.org/10.1109%2Ftcsi.2018.2881753

[9] C. Lammie and M. R. Azghadi, "Stochastic Computing for Low-Power and High-Speed Deep Learning on FPGA," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, Japan., May 2019.

[10] C. Lammie, W. Xiang, and M. R. Azghadi, "Accelerating Deterministic and Stochastic Binarized Neural Networks on FPGAS using OpenCL," in *Proceedings of the IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Dallas, TX., Aug. 2019, pp. 626–629.

[11] C. Lammie, O. Krestinskaya, A. James, and M. R. Azghadi, "Variation-aware Binarized Memristive Networks," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, Nov. 2019, pp. 490–493.

[12] C. Lammie, W. Xiang, and M. R. Azghadi, "Training progressively binarizing deep networks using fpgas," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[13] C. Lammie and M. R. Azghadi, "Live demonstration: Low-power and high-speed deep fpga inference engines for weed classification at the edge," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[14] J. K. Eshraghian, C. Lammie, and M. R. Azghadi, "Biologically plausible contrast detection using a memristor array," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[15] C. Lammie and M. R. Azghadi, "MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Oct. 2020, pp. 1–5.

[16] C. Lammie, W. Xiang, and M. R. Azghadi, "Towards memristive deep learning systems for real-time mobile epileptic seizure prediction," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021.

[17] A. Saleh, I. H. Laradji, C. Lammie, D. Vazquez, C. A. Flavell, and M. R. Azghadi, "A deep learning localization method for measuring abdominal muscle dimensions in ultrasound images," *IEEE Journal of Biomedical and Health Informatics*, 2021.

[18] C. Lammie, J. K. Eshraghian, C. Li, A. Amirsoleimani, R. Genov, W. D. Lu, and M. R. Azghadi, "Design Space Exploration of Dense and Sparse Mapping Schemes for RRAM Architectures," 2022.

[19] J. K. Eshraghian, C. Lammie, M. R. Azghadi, and W. D. Lu, "Navigating local minima in quantized spiking neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2202.07221

[20] T. Zhang, C. Lammie, M. R. Azghadi, A. Amirsoleimani, M. Ahmadi, and R. Genov, "Toward a formalized approach for spike sorting algorithms and hardware evaluation," 2022. [Online]. Available: https://arxiv.org/abs/2205.06514

[21] E. Ceolini, C. Frenkel, S. B. Shrestha, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Hand-gesture recognition based on EMG and event-based camera sensor fusion: a benchmark in neuromorphic computing," *Frontiers in Neuroscience*, no. 520438, p. 36, 2020.

[22] F. Corradi and G. Indiveri, "A Neuromorphic Event-Based Neural Recording System for Smart Brain-Machine-Interfaces," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 699–709, 2015.

[23] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128 ×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[24] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–5878, 2013.

[25] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, Jan. 2013.

[26] E. Ambrosi, A. Bricalli, M. Laudato, and D. Ielmini, "Impact of Oxide and Electrode Materials on the Switching Characteristics of Oxide ReRAM Devices," *Faraday Discuss*, vol. 213, p. 87, 2019.

[27] A. Fantini, L. Goux, A. Redolfi, R. Degraeve, G. Kar, Y. Y. Chen, and M. Jurczak, "Lateral and Vertical Scaling Impact on Statistical Performances and Reliability of 10nm TiN/Hf(Al)O/Hf/TiN RRAM Devices," in *Symposium on VLSI Technology*, 2014.

[28] L. Hangbing *et al.*, "Evolution of Conductive Filament and Its Impact on Reliability Issues in Oxide-Electrolyte Based Resistive Random Access Memory," *Scientific Reports*, vol. 5, no. 1, p. 7764, Jan. 2015.

[29] M. Zhao *et al.*, "Characterizing Endurance Degradation of Incremental Switching in Analog RRAM for Neuromorphic Systems," in *IEEE International Electron Devices Meeting*, 2018.

[30] J. Park, M. Jo, E. M. Bourim, J. Yoon, D. Seong, J. Lee, W. Lee, and H. Hwang, "Investigation of State Stability of Low-Resistance State in Resistive Memory," *IEEE Electron Device Letters*, vol. 31, no. 5, pp. 485–487, May 2010.

[31] B. Traoré, P. Blaise, E. Vianello, H. Grampeix, S. Jeannot, L. Perniola, B. De Salvo, and Y. Nishi, "On the Origin of Low-Resistance State Retention Failure in HfO2-Based RRAM and Impact of Doping/Alloying," *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4029–4036, Dec. 2015.

[32] S. Ambrogio, S. Balatti, Z. Q. Wang, Y.-S. Chen, H.-Y. Lee, F. T. Chen, and D. Ielmini, "Data Retention Statistics and Modelling in HfO 2 Resistive Switching Memories," in *IEEE International Reliability Physics Symposium*, 2015.

[33] D. Ielmini, F. Nardi, C. Cagli, and A. L. Lacaita, "Size-Dependent Retention Time in NiO-Based Resistive-Switching Memories," *IEEE Electron Device Letters*, vol. 31, no. 4, pp. 353–355, Apr. 2010.

[34] S. Balatti, S. Ambrogio, Z. Wang, S. Sills, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Voltage-controlled cycling endurance of hfox-based resistive-switching memory," *IEEE Transactions on Electron Devices*, vol. 62, no. 10, p. 3365, 2015.

[35] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821003335

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[37] R. Chellappa, S. Theodoridis, and A. van Schaik, "Advances in machine learning and deep neural networks," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 607–611, 2021.

[38] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, p. 100379, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000198

[39] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[40] "Take it to the edge," *Nature Electronics*, vol. 2, no. 1, 2019.

[41] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and P. Deaville, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.

[42] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, and A. J. Kenyon, "Memristors—From In-Memory Computing, Deep Learning Acceleration, and

Spiking Neural Networks to the Future of Neuromorphic and Bio-Inspired Computing," *Advanced Intelligent Systems*, vol. 2, no. 11, p. 2000085, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202000085

[43] E. Track, N. Forbes, and G. Strawn, "The end of Moore's law," *Computing in Science Engineering*, vol. 19, no. 2, pp. 4–6, 2017.

[44] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1737–1746.

[45] H. Tsai, S. Ambrogio, P. Narayanan, R. M. Shelby, and G. W. Burr, "Recent Progress in Analog Memory-based Accelerators for Deep Learning," *Journal of Physics D: Applied Physics*, vol. 51, no. 28, p. 283001, Jun. 2018.

[46] G. Rong, A. Mendez, E. B. Assi, B. Zhao, and M. Sawan, "Artificial Intelligence in Healthcare: Review and Prediction Case Studies," *Engineering*, vol. 6, no. 3, pp. 291–301, 2020.

[47] T. Arevalo, "The State of Health Care Industry- Statistics & Facts," Tech. Rep., Apr. 2020.

[48] V. Jindal, "Integrating Mobile and Cloud for PPG Signal Selection to Monitor Heart Rate during Intensive Physical Exercise," in *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESOFT)*, Austin, TX., May 2016, pp. 36–37.

[49] P. Sundaravadivel, K. Kesavan, L. Kesavan, S. P. Mohanty, and E. Kougianos, "Smart-Log: A Deep-Learning Based Automated Nutrition Monitoring System in the IoT," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 3, pp. 390–398, 2018.

[50] B. Shi, L. J. Grimm, M. A. Mazurowski, J. A. Baker, J. R. Marks, L. M. King, C. C. Maley, E. S. Hwang, and J. Y. Lo, "Prediction of occult invasive disease in ductal carcinoma in situ using deep learning features," *Journal of the American College of Radiology*, vol. 15, no. 3, pp. 527–534, 2018.

[51] X. Liu, L. Faes, A. U. Kale, S. K. Wagner, D. J. Fu, A. Bruynseels, T. Mahendiran, G. Moraes, M. Shamdas, C. Kern *et al.*, "A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis," *The Lancet Digital Health*, vol. 1, no. 6, pp. e271–e297, 2019.

[52] F. Liu, P. Yadav, A. M. Baschnagel, and A. B. McMillan, "MR-based Treatment Planning in Radiation Therapy Using a Deep Learning Approach," *Journal of Applied Clinical Medical Physics*, vol. 20, no. 3, pp. 105–114, 2019.

[53] W. Zhu, L. Xie, J. Han, and X. Guo, "The Application of Deep Learning in Cancer Prognosis Prediction," *Cancers*, vol. 12, no. 3, p. 603, 2020.

[54] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. C. Corrado, A. Darzi *et al.*, "International evaluation of an AI system for breast cancer screening," *Nature*, vol. 577, no. 7788, pp. 89–94, 2020.

[55] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature medicine*, vol. 25, no. 1, p. 65, 2019.

[56] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[57] T. Kalaiselvi, P. Sriramakrishnan, and K. Somasundaram, "Survey of using GPU CUDA programming model in medical image analysis," *Informatics in Medicine Unlocked*, vol. 9, pp. 133–144, 2017.

[58] N. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. 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 International Symposium on Computer Architecture (ISCA)*, Toronto, Canada., Jun. 2017.

[59] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Medicine*, vol. 25, no. 1, pp. 24–29, 2019.

[60] R. Perrault, Y. Shoham, E. Brynjolfsson, J. Clark, J. Etchemendy, B. Grosz, T. Lyons, J. Manyika, S. Mishra, and J. C. Niebles, "The ai index 2019 annual report," *AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA*, 2019.

[61] N. G. Peter *et al.*, "NVIDIA Fermi: The First Complete GPU Computing Architecture," *A White Paper of NVIDIA*, 2009.

[62] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[63] R. Zemouri, N. Zerhouni, and D. Racoceanu, "Deep Learning in the Biomedical Applications: Recent and Future Status," *Applied Sciences*, vol. 9, no. 8, p. 1526, 2019.

[64] E. Smistad, T. L. Falch, M. Bozorgi, A. C. Elster, and F. Lindseth, "Medical Image Segmentation on GPUs–A Comprehensive Review," *Medical Image Analysis*, vol. 20, no. 1, pp. 1–18, 2015.

[65] B. Farahani, F. Firouzi, and K. Chakrabarty, "Healthcare IoT," in *Intelligent Internet of Things*. Springer, 2020, pp. 515–545.

[66] Q. Xie, K. Faust, R. Van Ommeren, A. Sheikh, U. Djuric, and P. Diamandis, "Deep Learning for Image Analysis: Personalizing Medicine Closer to the Point of Care," *Critical Reviews in Clinical Laboratory Sciences*, vol. 56, no. 1, pp. 61–73, 2019.

[67] M. Hartmann, U. S. Hashmi, and A. Imran, "Edge computing in smart health care systems: Review, challenges, and research directions," *Transactions on Emerging Telecommunications Technologies*, p. e3710, 2019.

[68] I. Azimi, A. Anzanpour, A. M. Rahmani, T. Pahikkala, M. Levorato, P. Liljeberg, and N. Dutt, "Hich: Hierarchical Fog-Assisted Computing Architecture for Healthcare IoT," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–20, 2017.

[69] K. Sethi, V. Parmar, and M. Suri, "Low-Power Hardware-Based Deep-Learning Diagnostics Support Case Study," in *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Cleveland, OH., Oct. 2018.

[70] P. Sahu, D. Yu, and H. Qin, "Apply lightweight deep learning on internet of things for low-cost and easy-to-access skin cancer detection," in *Proceedings of Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications*, vol. 10579, Houston, TX, Feb. 2018, p. 1057912.

[71] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri, "Neuromorphic electronic circuits for building autonomous cognitive systems," *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367–1388, 2014.

[72] D. E. Rumelhart, G. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[73] T. C. Hollon, B. Pandian, A. R. Adapa, E. Urias, A. V. Save, S. S. S. Khalsa, D. G. Eichberg, R. S. DAmico, Z. U. Farooq, S. Lewis *et al.*, "Near real-time intraoperative brain tumor

diagnosis using stimulated Raman histology and deep neural networks," *Nature Medicine*, pp. 1–7, 2020.

[74] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis," *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 5, pp. 1589–1604, 2017.

[75] M. A. Sayeed, S. P. Mohanty, E. Kougianos, and H. P. Zaveri, "Neuro-Detect: A Machine Learning-Based Fast and Accurate Seizure Detection System in the IoMT," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 359–368, 2019.

[76] J. Yang and M. Sawan, "From seizure detection to smart and fully embedded seizure prediction engine: A review," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 5, pp. 1008–1023, 2020.

[77] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?" *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.

[78] J. Gao, H. Zhang, P. Lu, and Z. Wang, "An Effective LSTM Recurrent Network to Detect Arrhythmia on Imbalanced ECG Dataset," *Journal of Healthcare Engineering*, vol. 2019, 2019.

[79] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," *CoRR*, vol. abs/1805.07836, 2018. [Online]. Available: http://arxiv.org/abs/1805.07836

[80] X. Zhou, Y. Li, and W. Liang, "CNN-RNN Based Intelligent Recommendation for Online Medical Pre-Diagnosis Support," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[81] J. Laitala, M. Jiang, E. Syrjälä, E. K. Naeini, A. Airola, A. M. Rahmani, N. D. Dutt, and P. Liljeberg, "Robust ECG R-peak detection using LSTM," in *Proceedings of the ACM Symposium on Applied Computing (SAC)*, Brno, Czech Republic., Mar. 2020, pp. 1104–1111.

[82] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[83] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-scale Hierarchical Image Database," in *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami Beach, FL., Jun. 2009.

[84] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio, "Transfusion: Understanding Transfer Learning for Medical Imaging," in *NeurIPS*, Vancouver, Canada., Dec. 2019.

[85] M. S. Elmahdy, T. Ahuja, U. A. van der Heide, and M. Staring, "Patient-specific finetuning of deep learning models for adaptive radiotherapy in prostate ct," in *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, 2020, pp. 577–580.

[86] G. Indiveri and S.-C. Liu, "Memory and Information Processing in Neuromorphic Systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[87] F. Corradi, S. Pande, J. Stuijt, N. Qiao, S. Schaafsma, G. Indiveri, and F. Catthoor, "ECG-based Heartbeat Classification in Neuromorphic Hardware," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary., Jul. 2019.

[88] G. Indiveri and Y. Sandamirskaya, "The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 16–28, 2019.

[89] J. K. Eshraghian, K. Cho, C. Zheng, M. Nam, H. H.-C. Iu, W. Lei, and K. Eshraghian, "Neuromorphic vision hybrid rram-cmos architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2816–2829, 2018.

[90] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "Hots: a hierarchy of event-based time-surfaces for pattern recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1346–1359, 2016.

[91] M. Sharifshazileh, K. Burelo, T. Fedele, J. Sarnthein, and G. Indiveri, "A Neuromorphic Device for Detecting High-Frequency Oscillations in Human iEEG," in *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genova, Italy., Nov. 2019, pp. 69–72.

[92] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," *arXiv preprint arXiv:1908.11348*, 2019.

[93] "Edge TPU," https://coral.ai/docs/edgetpu/faq/.

[94] J. Hruska, "Intel Nervana Inference and Training AI Cards," https://www.extremetech.com/computing/296990-intel-nervana-nnp-i-nnp-t-a-training-inference.

[95] P. Kennedy, "Huawei Ascend 310," https://www.servethehome.com/huawei-ascend-910-provides-a-nvidia-ai-training-alternative/.

[96] J. Lee, J. K. Eshraghian, K. Cho, and K. Eshraghian, "Adaptive Precision CNN Accelerator Using Radix-X Parallel Connected Memristor Crossbars," *arXiv e-prints*, p. arXiv:1906.09395, Jun. 2019.

[97] D. Shin, J. Lee, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture," *IEEE Micro*, vol. 38, no. 5, pp. 85–93, 2018.

[98] 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.

[99] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2018.

[100] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Taipei, Taiwan., Oct. 2016.

[101] J. Zhang, S. Gajjala, P. Agrawal, G. H. Tison, L. A. Hallock, L. Beussink-Nelson, E. Fan, M. A. Aras, C. Jordan, K. E. Fleischmann *et al.*, "A Computer Vision Pipeline for Automated Determination of Cardiac Structure and Function and Detection of Disease by Two-Dimensional Echocardiography," *arXiv preprint arXiv:1706.07342*, 2017.

[102] 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.

[103] Q. Guan, Y. Wang, B. Ping, D. Li, J. Du, Y. Qin, H. Lu, X. Wan, and J. Xiang, "Deep Convolutional Neural Network VGG-16 Model for Differential Diagnosing of Papillary Thyroid Carcinomas in Cytological Images: A Pilot Study," *Journal of Cancer*, vol. 10, no. 20, p. 4876, 2019.

[104] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, 2016.

[105] I. Azimi, J. Takalo-Mattila, A. Anzanpour, A. M. Rahmani, J.-P. Soininen, and P. Liljeberg, "Empowering Healthcare IoT Systems with Hierarchical Edge-Based Deep Learning," in *Proceedings of the International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Washington, DC., Sep. 2018, pp. 63–68.

[106] J. Huang, S. Lin, N. Wang, G. Dai, Y. Xie, and J. Zhou, "TSE-CNN: A Two-Stage End-to-End CNN for Human Activity Recognition," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 1, pp. 292–299, 2020.

[107] B. Moons and M. Verhelst, "An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS," *IEEE Journal of Solid-state Circuits*, vol. 52, no. 4, pp. 903–914, 2016.

[108] M. Blaivas and L. Blaivas, "Are All Deep Learning Architectures Alike for Point-of-Care Ultrasound?: Evidence From a Cardiac Image Classification Model Suggests Otherwise," *Journal of Ultrasound in Medicine*, 2019.

[109] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA., Feb. 2017, pp. 246–247.

[110] M.-P. Hosseini, T. X. Tran, D. Pompili, K. Elisevich, and H. Soltanian-Zadeh, "Deep Learning with Edge Computing for Localization of Epileptogenicity Using Multimodal rs-fMRI and EEG Big Data," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, Columbus, OH., Jul. 2017, pp. 83–92.

[111] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "An 11.5TOPS/W 1024-MAC Butterfly Structure Dual-Core Sparsity-Aware Neural Processing Unit in 8nm Flagship Mobile SoC," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA., Feb. 2019, pp. 130–132.

[112] F. Preiswerk, C.-C. Cheng, J. Luo, and B. Madore, "Synthesizing Dynamic MRI Using Long-Term Recurrent Convolutional Networks," in *Proceedings of the International Workshop on Machine Learning in Medical Imaging (MLMI)*. Granada, Spain.: Springer, Sep. 2018, pp. 89–97.

[113] J. Acharya and A. Basu, "Deep Neural Network for Respiratory Sound Classification in Wearable Devices Enabled by Patient Specific Model Tuning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 3, pp. 535–544, 2020.

[114] S. Roy, S. Sridharan, S. Jain, and A. Raghunathan, "TxSim:Modeling Training of Deep Neural Networks on Resistive Crossbar Systems," *arXiv:2002.11151 [cs, eess, stat]*, Jan. 2021.

[115] J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Edge-AI in LoRa-based Health Monitoring: Fall Detection System with Fog Computing and LSTM Recurrent Neural Networks," in *Proceedings of the International Conference on Telecommunications and Signal Processing (TSP)*, 2019, pp. 601–604.

[116] Y. Chen, E. Yao, and A. Basu, "A 128-Channel Extreme Learning Machine-Based Neural Decoder for Brain Machine Interfaces," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 3, pp. 679–692, 2016.

[117] I. M. Baltruschat, H. Nickisch, M. Grass, T. Knopp, and A. Saalbach, "Comparison of Deep Learning Approaches for Multi-Label Chest X-Ray Classification," *Scientific Reports*, vol. 9, no. 1, pp. 1–10, 2019.

[118] W. Zhao, W. Zhao, W. Wang, X. Jiang, X. Zhang, Y. Peng, B. Zhang, and G. Zhang, "A Novel Deep Neural Network for Robust Detection of Seizures Using EEG Signals," *Computational and Mathematical Methods in Medicine*, vol. 2020, p. 9689821, Apr. 2020, publisher: Hindawi.

[119] G. Zamzmi, L.-Y. Hsu, W. Li, V. Sachdev, and S. Antani, "Harnessing Machine Intelligence in Automatic Echocardiogram Analysis: Current Status, Limitations, and Future Directions," *IEEE Reviews in Biomedical Engineering*, 2020.

[120] M. S. Roy, B. Roy, R. Gupta, and K. D. Sharma, "On-device reliability assessment and prediction of missing photoplethysmographic data using deep neural networks," *IEEE transactions on biomedical circuits and systems*.

[121] K. Zhao, H. Jiang, Z. Wang, P. Chen, B. Zhu, and X. Duan, "Long-term bowel sound monitoring and segmentation by wearable devices and convolutional neural networks," *IEEE Transactions on Biomedical Circuits and Systems*, 2020.

[122] S. Shaikh, R. So, T. Sibindi, C. Libedinsky, and A. Basu, "Sparse Ensemble Machine Learning to Improve Robustness of Long-Term Decoding in iBMIs," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 2, pp. 380–389, 2020.

[123] G. O'Leary, D. M. Groppe, T. A. Valiante, N. Verma, and R. Genov, "NURIP: Neural Interface Processor for Brain-State Classification and Programmable-Waveform Neurostimulation," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 11, pp. 3150–3162, 2018.

[124] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

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

[126] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proceedings of the IEEE International*

Midwest Symposium on Circuits and Systems (MWSCAS), Fort Collins, CO., Aug. 2012, pp. 1044–1047.

[127] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, Minneapolis, MN., Jun. 2014, pp. 13–24.

[128] D. Abts, J. Ross, J. Sparling, M. Wong-VanHaren, M. Baker, T. Hawkins, A. Bell, J. Thompson, T. Kahsai, G. Kimmell *et al.*, "Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads," Valencia, Spain., May 2020.

[129] H. Kung and C. E. Leiserson, "Systolic Arrays (for VLSI)," in *Proceedings of Sparse Matrix*, vol. 1. Society for industrial and applied mathematics, 1979, pp. 256–282.

[130] H.-T. Kung, "Why systolic architectures?" *Computer*, no. 1, pp. 37–46, 1982.

[131] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[132] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-Based Neural Network Inference Accelerator," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[133] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, apr 2011. [Online]. Available: https://doi.org/10.1109%2Ftcad.2011.2110592

[134] M. Carreras, G. Deriu, L. Raffo, L. Benini, and P. Meloni, "Optimizing Temporal Convolutional Network inference on FPGA-based accelerators," *arXiv preprint arXiv:2005.03775*, 2020.

[135] D. Wang, K. Xu, and D. Jiang, "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks," in *Proceedings of the International Conference on Field Programmable Technology (ICFPT)*, Melbourne, Australia., Dec. 2017, pp. 279–282.

[136] M. Wess, P. S. Manoj, and A. Jantsch, "Neural network based ECG anomaly detection on FPGA and trade-off analysis," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD., May 2017.

[137] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks," *BMC Bioinformatics*, vol. 19, no. 18, p. 490, 2018.

[138] R. R. Shrivastwa, V. Pudi, and A. Chattopadhyay, "An FPGA-Based Brain Computer Interfacing Using Compressive Sensing and Machine Learning," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Hong Kong, China., Jul. 2018, pp. 726–731.

[139] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.

[140] G. Burr, P. Narayanan, R. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, Washington, DC., Dec. 2015.

[141] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, p. 60, 2018.

[142] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.

[143] J. K. Eshraghian, S.-M. Kang, S. Baek, G. Orchard, H. H.-C. Iu, and W. Lei, "Analog weights in reram dnn accelerators," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 267–271.

[144] M. R. Azghadi, B. Linares-Barranco, D. Abbott, and P. H. W. Leong, "A hybrid cmos-memristor neuromorphic synapse," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 2, pp. 434–445, 2017.

[145] M. Rahimi Azghadi, Y. Chen, J. Eshraghian, J. Chen, C. Lin, A. Amirsoleimani, A. Mehonic, A. Kenyon, B. Fowler, J. Lee, and Y. Chang, "Complementary Metal-Oxide Semiconductor and Memristive Hardware for Neuromorphic Computing," *Advanced Intelligent Systems*, vol. 2, no. 5, p. 1900189, 2020.

[146] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature Materials*, vol. 18, no. 4, pp. 309–323, Apr. 2019.

[147] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in Memristive Neural Network Architectures Using Analog Backpropagation Circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 719–732, Feb. 2019.

[148] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, Washington, DC., Dec. 2015.

[149] N. Bien, P. Rajpurkar, R. L. Ball, J. Irvin, A. Park, E. Jones, M. Bereket, B. N. Patel, K. W. Yeom, K. Shpanskaya *et al.*, "Deep-learning-assisted diagnosis for knee magnetic resonance imaging: development and retrospective validation of MRNet," *PLoS Medicine*, vol. 15, no. 11, p. e1002699, 2018.

[150] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 715–731.

[151] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A General Model for Voltage-Controlled Memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, p. 786, 2015.

[152] E. Yalon, A. Gavrilov, S. Cohen, D. Mistele, B. Meyler, J. Salzman, and D. Ritter, "Resistive switching in HfO_2 probed by a metal–insulator–semiconductor bipolar transistor," *IEEE Electron Device Letters*, vol. 33, no. 1, p. 11, 2012.

[153] A. M. Hassan, A. F. Khalaf, K. S. Sayed, H. H. Li, and Y. Chen, "Real-Time Cardiac Arrhythmia Classification Using Memristor Neuromorphic Computing System," in *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, HI, Jul. 2018, pp. 2567–2570.

[154] F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn, and W. D. Lu, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, no. 7, pp. 290–299, 2019.

[155] T. Hirtzlin, M. Bocquet, B. Penkovsky, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Digital Biologically Plausible Implementation of Binarized Neural Networks With Differential Hafnium Oxide Resistive Memory Arrays," *Frontiers in Neuroscience*, vol. 13, 2019.

[156] F. C. Bauer, D. R. Muir, and G. Indiveri, "Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor," *IEEE Transactions on Biomedical Circuits and Systems*, 2019.

[157] E. Donati, M. Payvand, N. Risi, R. Krause, K. Burelo, G. Indiveri, T. Dalgaty, and E. Vianello, "Processing EMG signals using reservoir computing on an event-based neuromorphic system," in *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Cleveland, Ohio., Oct. 2018.

[158] E. Donati, M. Payvand, N. Risi, R. Krause, and G. Indiveri, "Discrimination of EMG Signals Using a Neuromorphic Implementation of a Spiking Neural Network," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 795–803, 2019.

[159] J. Behrenbeck, Z. Tayeb, C. Bhiri, C. Richter, O. Rhodes, N. Kasabov, J. I. Espinosa-Ramos, S. Furber, G. Cheng, and J. Conradt, "Classification and regression of spatio-temporal signals using NeuCube and its realization on SpiNNaker neuromorphic hardware," *Journal of Neural Engineering*, vol. 16, no. 2, p. 026014, 2019.

[160] E. Nurse, B. S. Mashford, A. J. Yepes, I. Kiral-Kornek, S. Harrer, and D. R. Freestone, "Decoding EEG and LFP Signals using Deep Learning: Heading TrueNorth," in *Proceedings of the ACM International Conference on Computing Frontiers (CF)*, Como, Italy., May 2016, pp. 259–266.

[161] S. Shaikh, R. So, T. Sibindi, C. Libedinsky, and A. Basu, "Real-time Closed Loop Neural Decoding on a Neuromorphic Chip," in *Proceedings of the IEEE/EMBS International Conference on Neural Engineering (NER)*, San Francisco, CA., Mar. 2019, pp. 670–673.

[162] ——, "Towards Intelligent Intracortical BMI (i^2BMI): Low-Power Neuromorphic Decoders That Outperform Kalman Filters," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1615–1624, 2019.

[163] P. Škoda, T. Lipić, Srp, B. M. Rogina, K. Skala, and F. Vajda, "Implementation framework for Artificial Neural Networks on FPGA," in *Proceedings of the International Convention (MIPRO)*, Opatija, Croatia., May 2011, pp. 274–278.

[164] C. Heelan, A. V. Nurmikko, and W. Truccolo, "FPGA implementation of deep-learning recurrent neural networks with sub-millisecond real-time latency for BCI-decoding of large-scale neural sensors (104 nodes)." in *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, HI., Jul. 2018.

[165] L. G. Rocha, D. Biswas, B. Verhoef, S. Bampi, C. Van Hoof, M. Konijnenburg, M. Verhelst, and N. Van Helleputte, "Binary CorNET: Accelerator for HR Estimation From Wrist-PPG," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 4, pp. 715–726, 2020.

[166] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin, "SensorNet: A Scalable and Low-Power Deep Convolutional Neural Network for Multimodal Data Classification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 274–287, 2019.

[167] L. Ohno-Machado and D. Bialek, "Diagnosing breast cancer from fnas: variable relevance in neural network and logistic regression models," *Studies in Health Technology and Informatics*, vol. 52, pp. 537–540, 1998.

[168] Y. Ku, W. Tompkins, and Q. Xue, "Artificial neural network for ECG arrhythmia monitoring," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, vol. 2. Baltimore, MD.: IEEE, Jun. 1992, pp. 987–992.

[169] J. K. Eshraghian, "Human ownership of artificial creativity," *Nature Machine Intelligence*, pp. 157–160, 2020.

[170] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[171] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "PULP-HD: Accelerating Brain-inspired High-dimensional Computing on a Parallel Ultra-low Power Platform," in *Proceedings of the ACM/ESDA/IEEE Design Automation Conference (DAC)*, San Francisco, CA., Jun. 2018.

[172] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[173] P. Merolla and K. Boahen, "A Recurrent Model of Orientation Maps with Simple and Complex Cells," in *Proceedings of Advances in Neural Information Processing Systems 17 (NIPS)*, Vancouver, Canada., Dec. 2004, pp. 1995–2002.

[174] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[175] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 145–158, 2019.

[176] C. Frenkel, J.-D. Legat, and D. Bol, "MorphIC: A 65-nm 738k-Synapse/mm ^{2} Quad-Core Binary-Weight Digital Neuromorphic Processor With Stochastic Spike-Driven Online Learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 999–1010, 2019.

[177] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers in neuroscience*, vol. 9, p. 141, 2015.

[178] M. R. Azghadi, S. Moradi, D. B. Fasnacht, M. S. Ozdas, and G. Indiveri, "Programmable spike-timing-dependent plasticity learning circuits in neuromorphic VLSI architectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 2, p. art. no. 17, 2015.

[179] M. Payvand and G. Indiveri, "Spike-based Plasticity Circuits for Always-on On-line Learning in Neuromorphic Systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, Japan., May 2019.

[180] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[181] G. Bellec, F. Scherr, E. Hajek, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Eligibility traces provide a data-inspired alternative to backpropagation through time," Vancouver, Canada., Dec. 2019.

[182] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn, "Dendritic cortical microcircuits approximate the backpropagation algorithm," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, Montreal, Canada., Dec. 2018, pp. 8721–8732.

[183] A. Valentian, F. Rummens, E. Vianello *et al.*, "Fully Integrated Spiking Neural Network with Analog Neurons and RRAM Synapses," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA., Dec. 2019, pp. 14.13.1–14.13.4.

[184] Y. Hayakawa, A. Himeno, R. Yasuhara, W. Boullart *et al.*, "Highly reliable TaOx ReRAM with centralized filament for 28-nm embedded application," in *VLSI Technology*, 2015, pp. T14–T15.

[185] T. Dalgaty, M. Payvand, B. De Salvo *et al.*, "Hybrid cmos-rram neurons with intrinsic plasticity," in *IEEE ISCAS*. IEEE, 2019, pp. 1–5.

[186] M. Payvand, Y. Demirag, T. Dalgaty, E. Vianello, and G. Indiveri, "Analog weight updates with compliance current modulation of binary rerams for on-chip learning," in *To appear in the Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020.

[187] E. Chicca and G. Indiveri, "A recipe for creating ideal hybrid memristive-cmos neuromorphic processing systems," *Applied Physics Letters*, vol. 116, no. 12, p. 120501, 2020.

[188] Z. Chen, A. Howe, H. T. Blair, and J. Cong, "CLINK: Compact LSTM Inference Kernel for Energy Efficient Neurofeedback Devices," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Bellevue, WA., Jul. 2018.

[189] T. S. Hall, C. M. Twigg, P. Hasler, and D. V. Anderson, "Application Performance of Elements in a Floating-gate FPAA," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Vancouver, Canada., May 2004.

[190] R. Manjunath and K. S. Gurumurthy, "Artificial Neural Networks as Building Blocks of Mixed Signal FPGA," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT)*, Tokyo, Japan., Dec. 2003.

[191] Puxuan Dong, G. L. Bilbro, and Mo-Yuen Chow, "Implementation of Artificial Neural Network for Real Time Applications Using Field Programmable Analog Arrays," in *Proceedings of the IEEE International Joint Conference on Neural Network Proceedings (IJCNN)*, Vancouver, Canada., Jul. 2006.

[192] C. R. Schlottmann and P. E. Hasler, "A Highly Dense, Low Power, Programmable Analog Vector-Matrix Multiplier: The FPAA Implementation," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 3, pp. 403–411, 2011.

[193] S. Shah, H. Toreyin, O. T. Inan, and J. Hasler, "Reconfigurable Analog Classifier for Knee-joint Rehabilitation," in *Proceedings of the IEEE International Conference in Engineering in Medicine and Biology Society (EMBC)*, Orlando, FL., Aug. 2016.

[194] A. Zbrzeski, P. Hasler, F. Kölbl, E. Syed, N. Lewis, and S. Renaud, "A programmable Bioamplifier on FPAA for In Vivo Neural Recording," in *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Paphos, Cyprus., Nov. 2010.

[195] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1419–1428.

[196] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A Deep Neural Network Accelerator Based on Tiled RRAM Architecture," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA., Dec. 2019, pp. 14–4.

[197] A. Shoeb, H. Edwards, J. Connolly, B. Bourgeois, T. Treves, and J. Guttag, "Patient-specific Seizure Onset Detection," in *Proceedings of IEEE Engineering in Medicine and Biology Society Conference (EMBC)*, San Francisco, CA., 2004.

[198] M. A. Bin Altaf, C. Zhang, and J. Yoo, "A 16-channel patient-specific seizure onset and termination detection soc with impedance-adaptive transcranial electrical stimulator," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 11, pp. 2728–2740, 2015.

[199] J. Yoo, L. Yan, D. El-Damak, M. A. B. Altaf, A. H. Shoeb, and A. P. Chandrakasan, "An 8-channel scalable eeg acquisition soc with patient-specific seizure classification and recording processor," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 214–228, 2013.

[200] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, Jul. 2020. [Online]. Available: https://doi.org/10.1038/s41565-020-0655-z

[201] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, no. 1, p. 2473, May 2020.

[202] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[203] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, Jun. 2018.

[204] X. Liu and Z. Zeng, "Memristor crossbar architectures for implementing deep neural networks," *Complex & Intelligent Systems*, Jul. 2021. [Online]. Available: https://doi.org/10.1007/s40747-021-00282-4

[205] M. Shahsavari, *Unconventional Computation From Digital to Brain-like Neuromorphic: Memristive Computing*. Éditions universitaires européennes, 2017.

[206] M. A. Zidan, A. Chen, G. Indiveri, and W. D. Lu, *Memristive Computing Devices and Applications*. Cham: Springer International Publishing, 2022, pp. 5–32. [Online]. Available: https://doi.org/10.1007/978-3-030-42424-4_2

[207] Z. Sun and R. Huang, "Time Complexity of In Memory Matrix Vector Multiplication," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[208] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electronics*, vol. 1, no. 1, pp. 22–29, Jan. 2018.

[209] O. Bichler, D. Roclin, C. Gamrat, and D. Querlioz, "Design exploration methodology for memristor-based spiking neuromorphic architectures with the xnet event-driven simulator,"

in *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2013, pp. 7–12.

[210] Y. Demirag, C. Frenkel, M. Payvand, and G. Indiveri, "Online training of spiking recurrent neural networks with phase-change memory synapses," *CoRR*, vol. abs/2108.01804, 2021. [Online]. Available: https://arxiv.org/abs/2108.01804

[211] P. Boulet, P. Devienne, P. Falez, G. Polito, M. Shahsavari, and P. Tirilly, "N2s3, an open-source scalable spiking neuromorphic hardware simulator," 2017.

[212] L. Chua, "Memristor-The missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.

[213] M. Kund, G. Beitel, C.-U. Pinnow, T. Rohr, J. Schumann, R. Symanczyk, K. Ufert, and G. Muller, "Conductive bridging ram (cbram): an emerging non-volatile memory technology scalable to sub 20nm," in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, 2005, pp. 754–757.

[214] A. V. Khvalkovskiy, D. Apalkov, S. Watts, R. Chepulskii, R. S. Beach, A. Ong, X. Tang, A. Driskill-Smith, W. H. Butler, P. B. Visscher, D. Lottis, E. Chen, V. Nikitin, and M. Krounbi, "Basic principles of STT-MRAM cell operation in memory arrays," vol. 46, no. 7, p. 074001, feb 2013. [Online]. Available: https://doi.org/10.1088/0022-3727/46/7/074001

[215] M. Khalid, "Review on Various Memristor Models, Characteristics, Potential Applications, and Future Works," *Transactions on Electrical and Electronic Materials*, vol. 20, no. 4, pp. 289–298, Aug. 2019.

[216] W. Woods, M. M. A. Taha, S. J. Dat Tran, J. Bürger, and C. Teuscher, "Memristor panic — A survey of different device models in crossbar architectures," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH ´15)*, Jul. 2015, pp. 106–111.

[217] B. Mohammad, D. Homouz, and H. Elgabra, "Robust Hybrid Memristor-CMOS Memory: Modeling and Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 11, pp. 2069–2079, Nov. 2013.

[218] W. Fei, H. Yu, W. Zhang, and K. S. Yeo, "Design Exploration of Hybrid CMOS and Memristor Circuit by New Modified Nodal Analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1012–1025, Jun. 2012.

[219] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[220] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*, Aug. 2012, pp. 392–397.

[221] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, Jul. 2015.

[222] L. Xia, B. Li, T. Tang, P. Gu, P. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "MNSIM: Simulation Platform for Memristor-Based Neuromorphic Computing System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1009–1022, May. 2018.

[223] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2017, pp. 541–552.

[224] X. Sun and S. Yu, "Impact of Non-Ideal Characteristics of Resistive Synaptic Devices on Implementing Convolutional Neural Networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 570–579, Sep. 2019.

[225] A. Mehonic, D. Joksas, W. H. Ng, M. Buckwell, and A. J. Kenyon, "Simulation of Inference Accuracy Using Realistic RRAM Devices," *Frontiers in Neuroscience*, vol. 13, p. 593, 2019.

[226] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 326–338, Feb. 2021.

[227] L. W. Nagel and D. Pederson, "SPICE (simulation program with integrated circuit emphasis)," Tech. Rep. UCB/ERL M382, Apr. 1973.

[228] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, Dec. 2000.

[229] L. Song, J. Zhang, A. Chen, H. Wu, H. Qian, and Z. Yu, "An efficient method for evaluating RRAM crossbar array performance," *Solid-State Electronics*, vol. 120, pp. 32–40, Jun. 2016.

[230] R. Uppala, C. Yakopcic, and T. M. Taha, "Methods for reducing memristor crossbar simulation time," in *2015 National Aerospace and Electronics Conference (NAECON)*, Jun. 2015, pp. 312–319.

[231] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[232] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.

[233] M. Imani, M. Samragh Razlighi, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Deep Learning Acceleration with Neuron-to-Memory Transformation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2020, pp. 1–14.

[234] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, and M.-F. Chang, "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, Nov. 2018, pp. 1–8.

[235] X. Ma, G. Yuan, S. Lin, C. Ding, F. Yu, T. Liu, W. Wen, X. Chen, and Y. Wang, "Tiny but Accurate: A Pruned, Quantized and Optimized Memristor Crossbar Framework for Ultra Efficient DNN Implementation," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2020, pp. 301–306.

[236] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, "An Ultra-Efficient Memristor-Based DNN Framework with Structured Weight Pruning and Quantization Using ADMM," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2019, pp. 1–6.

[237] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies," in *2019 IEEE International Electron Devices Meeting (IEDM)*, Dec. 2019, pp. 32.5.1–32.5.4.

[238] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators for On-chip Training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.

[239] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, "NeuroSim Simulator for Compute-in-Memory Hardware Accelerator: Validation and Benchmark," *Frontiers in Artificial Intelligence*, vol. 4, 2021.

[240] M. J. Rasch, D. Moreda, T. Gokmen, M. Le Gallo, F. Carta, C. Goldberg, K. El Maghraoui, A. Sebastian, and V. Narayanan, "A Flexible and Fast PyTorch Toolkit for Simulating Training and Inference on Analog Crossbar Arrays," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021.

[241] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[242] 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.

[243] N. Gong, T. Idé, S. Kim, I. Boybat, A. Sebastian, V. Narayanan, and T. Ando, "Signal and noise extraction from analog memory elements for neuromorphic computing," *Nature Communications*, vol. 9, no. 1, p. 2102, May 2018. [Online]. Available: https://doi.org/10.1038/s41467-018-04485-1

[244] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1333–1343, 2020.

[245] Y.-F. Qin, H. Bao, F. Wang, J. Chen, Y. Li, and X.-S. Miao, "Recent Progress on Memristive Convolutional Neural Networks for Edge Intelligence," *Advanced Intelligent Systems*, vol. 2, no. 11, p. 2000114, 2020.

[246] P. W. C. Ho, H. A. F. Almurib, and T. N. Kumar, "Configurable memristive logic block for memristive-based FPGA architectures," *Integration*, vol. 56, pp. 61–69, Jan. 2017.

[247] M. F. Tolba, M. E. Fouda, H. G. Hezayyin, A. H. Madian, and A. G. Radwan, "Memristor FPGA IP Core Implementation for Analog and Digital Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 8, pp. 1381–1385, Aug. 2019.

[248] E. Beghi, G. Giussani, E. Nichols, F. Abd-Allah, J. Abdela, A. Abdelalim, H. N. Abraha, M. G. Adib, S. Agrawal, F. Alahdab *et al.*, "Global, regional, and national burden of epilepsy, 1990–2016: a systematic analysis for the global burden of disease study 2016," *The Lancet Neurology*, vol. 18, no. 4, pp. 357–375, 2019.

[249] C. E. Stafstrom and L. Carmant, "Seizures and epilepsy: An overview for neuroscientists," *Cold Spring Harbor Perspectives in Medicine*, vol. 5, no. 6, 2015.

[250] D. C. Patel, B. P. Tewari, L. Chaunsali, and H. Sontheimer, "Neuron-glia interactions in the pathophysiology of epilepsy," *Nature Reviews Neuroscience*, vol. 20, no. 5, pp. 282–297, May 2019.

[251] G. P. Brennan and D. C. Henshall, "micrornas in the pathophysiology of epilepsy," *Neuroscience Letters*, vol. 667, pp. 47–52, 2018, epilepsy: Advances in Genetics and Pathophysiology.

[252] S. Gasparini, E. Ferlazzo, C. Sueri, V. Cianci, M. Ascoli, S. M. Cavalli, E. Beghi, V. Belcastro, A. Bianchi, P. Benna, R. Cantello, D. Consoli, F. A. De Falco, G. Di Gennaro, A. Gambardella, G. L. Gigli, A. Iudice, A. Labate, R. Michelucci, M. Paciaroni, P. Palumbo, A. Primavera, F. Sartucci, P. Striano, F. Villani, E. Russo, G. De Sarro, U. Aguglia, and O. behalf of the Epilepsy Study Group of the Italian Neurological Society, "Hypertension, seizures, and epilepsy: a review on pathophysiology and management," *Neurological Sciences*, vol. 40, no. 9, pp. 1775–1783, Sep. 2019.

[253] M. K. Siddiqui, R. Morales-Menendez, X. Huang, and N. Hussain, "A review of epileptic seizure detection using machine learning classifiers," *Brain informatics*, vol. 7, no. 1, pp. 5–5, May 2020, publisher: Springer Berlin Heidelberg.

[254] F. E. Ibrahim, H. M. Emara, W. El-Shafai, M. Elwekeil, M. Rihan, I. M. Eldokany, T. E. Taha, A. S. El-Fishawy, E.-S. M. El-Rabaie, E. Abdellatef, and F. E. Abd El-Samie, "Deep Learning-based Seizure Detection and Prediction from EEG Signals," *International journal for numerical methods in biomedical engineering*, p. e3573, Jan 2022.

[255] R. Das, "Special Issue on In-Memory Computing," *IEEE Micro*, vol. 42, no. 1, pp. 87–88, 2022.

[256] B. Li, L. Song, F. Chen, X. Qian, Y. Chen, and H. H. Li, "ReRAM-based Accelerator for Deep Learning," in *Design, Automation Test in Europe Conference Exhibition*, 2018.

[257] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5987–5995.

[258] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[259] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal Deep Learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, p. 689–696.

[260] X. Wang, X. Wang, W. Liu, Z. Chang, T. Kärkkäinen, and F. Cong, "One dimensional convolutional neural networks for seizure onset detection using long-term scalp and intracranial eeg," *Neurocomputing*, vol. 459, pp. 212–222, 2021.

[261] W. Webber, R. P. Lesser, R. T. Richardson, and K. Wilson, "An approach to seizure detection using an artificial neural network (ann)," *Electroencephalography and clinical Neurophysiology*, vol. 98, no. 4, pp. 250–272, 1996.

[262] N. Pradhan, P. Sadasivan, and G. Arunodaya, "Detection of seizure activity in eeg by an artificial neural network: A preliminary study," *Computers and Biomedical Research*, vol. 29, no. 4, pp. 303–313, 1996.

[263] A. M. Chan, F. T. Sun, E. H. Boto, and B. M. Wingeier, "Automated seizure onset detection for accurate onset time determination in intracranial eeg," *Clinical Neurophysiology*, vol. 119, no. 12, pp. 2687–2696, 2008.

[264] T. Netoff, Y. Park, and K. Parhi, "Seizure prediction using cost-sensitive support vector machine," in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2009, pp. 3322–3325.

[265] K. Chua, V. Chandran, U. R. Acharya, and C. Lim, "Automatic identification of epileptic electroencephalography signals using higher-order spectra," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 223, no. 4, pp. 485–495, 2009.

[266] T. L. Sorensen, U. L. Olsen, I. Conradsen, J. Henriksen, T. W. Kjaer, C. E. Thomsen, and H. B. Sorensen, "Automatic epileptic seizure onset detection using matching pursuit: a case study," in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*. IEEE, 2010, pp. 3277–3280.

[267] L. Chisci, A. Mavino, G. Perferi, M. Sciandrone, C. Anile, G. Colicchio, and F. Fuggetta, "Real-time epileptic seizure prediction using ar models and support vector machines," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 5, pp. 1124–1132, 2010.

[268] E. B. Petersen, J. Duun-Henriksen, A. Mazzaretto, T. W. Kjær, C. E. Thomsen, and H. B. Sorensen, "Generic single-channel detection of absence seizures," in *2011 Annual Interna-*

tional Conference of the IEEE Engineering in Medicine and Biology Society. IEEE, 2011, pp. 4820–4823.

[269] A. Temko, E. Thomas, W. Marnane, G. Lightbody, and G. Boylan, "Eeg-based neonatal seizure detection with support vector machines," *Clinical Neurophysiology*, vol. 122, no. 3, pp. 464–473, 2011.

[270] U. R. Acharya, S. V. Sree, and J. S. Suri, "Automatic detection of epileptic eeg signals using higher order cumulant features," *International journal of neural systems*, vol. 21, no. 05, pp. 403–414, 2011.

[271] A. Kharbouch, A. Shoeb, J. Guttag, and S. S. Cash, "An algorithm for seizure onset detection using intracranial eeg," *Epilepsy & Behavior*, vol. 22, pp. S29–S35, 2011.

[272] Y. Liu, W. Zhou, Q. Yuan, and S. Chen, "Automatic seizure detection using wavelet transform and svm in long-term intracranial eeg," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 20, no. 6, pp. 749–755, 2012.

[273] I. Ullah, M. Hussain, E. ul Haq Qazi, and H. Aboalsamh, "An automated system for epilepsy detection using eeg brain signals based on deep learning approach," *Expert Systems with Applications*, vol. 107, pp. 61 – 71, 2018.

[274] R. Abiyev, M. Arslan, J. Bush Idoko, B. Sekeroglu, and A. Ilhan, "Identification of epileptic eeg signals using convolutional neural networks," *Applied Sciences*, vol. 10, no. 12, 2020.

[275] P. Boonyakitanont, A. Lek-uthai, K. Chomtho, and J. Songsiri, "A comparison of deep neural networks for seizure detection in eeg signals," *bioRxiv*, 2019.

[276] S. Liss, "Method and apparatus for monitoring and counteracting excess brain electrical energy to prevent epileptic seizures and the like," *US3850161A*, 1973.

[277] S. Viglione, V. Ordon, W. Martin, and C. Kesler, "Epileptic seizure warning system," *US3863625A*, 1973.

[278] A. Aarabi and B. He, "A rule-based seizure prediction method for focal neocortical epilepsy," *Clinical Neurophysiology*, vol. 123, no. 6, pp. 1111–1122, 2012.

[279] S. Li, W. Zhou, Q. Yuan, and Y. Liu, "Seizure prediction using spike rate of intracranial eeg," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 21, no. 6, pp. 880–886, 2013.

[280] A. S. Zandi, R. Tafreshi, M. Javidan, and G. A. Dumont, "Predicting epileptic seizures in scalp eeg based on a variational bayesian gaussian mixture model of zero-crossing intervals," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 5, pp. 1401–1413, 2013.

[281] M. Bedeeuzzaman, T. Fathima, Y. U. Khan, and O. Farooq, "Seizure prediction using statistical dispersion measures of intracranial eeg," *Biomedical Signal Processing and Control*, vol. 10, pp. 338–341, 2014.

[282] P. E. McSharry, T. He, L. A. Smith, and L. Tarassenko, "Linear and non-linear methods for automatic seizure detection in scalp electro-encephalogram recordings," *Medical and Biological Engineering and Computing*, vol. 40, no. 4, pp. 447–461, 2002.

[283] B. Schelter, H. Feldwisch-Drentrup, M. Ihle, A. Schulze-Bonhage, and J. Timmer, "Seizure prediction in epilepsy: From circadian concepts via probabilistic forecasting to statistical evaluation," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2011, pp. 1624–1627.

[284] S. Wang, W. A. Chaovalitwongse, and S. Wong, "A novel reinforcement learning framework for online adaptive seizure prediction," in *2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2010, pp. 499–504.

[285] H. Daoud and M. A. Bayoumi, "Efficient epileptic seizure prediction based on deep learning," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 5, pp. 804–813, 2019.

[286] R. Jana and I. Mukherjee, "Deep learning based efficient epileptic seizure prediction with eeg channel optimization," *Biomedical Signal Processing and Control*, vol. 68, p. 102767, 2021.

[287] T. Dissanayake, T. Fernando, S. Denman, S. Sridharan, and C. Fookes, "Patient-independent epileptic seizure prediction using deep learning models," *arXiv preprint arXiv:2011.09581*, 2020.

[288] T. N. Alotaiby, S. A. Alshebeili, T. Alshawi, I. Ahmad, and F. E. Abd El-Samie, "Eeg seizure detection and prediction algorithms: a survey," *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, p. 183, Dec. 2014.

[289] H. Kassiri, M. T. Salam, M. R. Pazhouhandeh, N. Soltani, J. L. Perez Velazquez, P. Carlen, and R. Genov, "Rail-to-rail-input dual-radio 64-channel closed-loop neurostimulator," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 11, pp. 2793–2810, 2017.

[290] D. Kudithipudi, Q. Saleh, C. Merkel, J. Thesing, and B. Wysocki, "Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing," *Frontiers in Neuroscience*, vol. 9, p. 502, 2016.

[291] C. Merkel, Q. Saleh, C. Donahue, and D. Kudithipudi, "Memristive reservoir computing architecture for epileptic seizure detection," *Procedia Computer Science*, vol. 41, pp. 249 –

254, 2014, 5th Annual International Conference on Biologically Inspired Cognitive Architectures, 2014 BICA.

[292] R.-E. Karamani, I.-A. Fyrigos, V. Ntinas, I. Vourkas, G. C. Sirakoulis, and A. Rubio, "Memristive cellular automata for modeling of epileptic brain activity," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[293] Z. Liu, J. Tang, B. Gao, P. Yao, X. Li, D. Liu, Y. Zhou, H. Qian, B. Hong, and H. Wu, "Neural signal analysis with memristor arrays towards high-efficiency brain–machine interfaces," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[294] S. M. Usman, M. Usman, and S. Fong, "Epileptic seizures prediction using machine learning methods," *Computational and mathematical methods in medicine*, vol. 2017, 2017.

[295] K. Fujiwara, M. Miyajima, T. Yamakawa, E. Abe, Y. Suzuki, Y. Sawada, M. Kano, T. Maehara, K. Ohta, T. Sasai-Sakuma *et al.*, "Epileptic seizure prediction based on multivariate statistical process control of heart rate variability features," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 6, pp. 1321–1332, 2015.

[296] M. Zanghieri, A. Burrello, S. Benatti, K. Schindler, and L. Benini, "Low-latency detection of epileptic seizures from ieeg with temporal convolutional networks on a low-power parallel mcu," in *2021 IEEE Sensors Applications Symposium (SAS)*. IEEE, 2021, pp. 1–6.

[297] C. N. Heck, D. King-Stephens, A. D. Massey, D. R. Nair, B. C. Jobst, G. L. Barkley, V. Salanova, A. J. Cole, M. C. Smith, R. P. Gwinn *et al.*, "Two-year seizure reduction in adults with medically intractable partial onset epilepsy treated with responsive neurostimulation: final results of the rns system pivotal trial," *Epilepsia*, vol. 55, no. 3, pp. 432–441, 2014.

[298] M. Nasseri, T. Pal Attia, B. Joseph, N. M. Gregg, E. S. Nurse, P. F. Viana, G. Worrell, M. Dümpelmann, M. P. Richardson, D. R. Freestone *et al.*, "Ambulatory seizure forecasting with a wrist-worn device using long-short term memory deep learning," *Scientific reports*, vol. 11, no. 1, pp. 1–9, 2021.

[299] Y. Yang, M. Zhou, Y. Niu, C. Li, R. Cao, B. Wang, P. Yan, Y. Ma, and J. Xiang, "Epileptic seizure prediction based on permutation entropy," *Frontiers in computational neuroscience*, vol. 12, p. 55, 2018.

[300] M. Pinto, A. Leal, F. Lopes, A. Dourado, P. Martins, C. A. Teixeira *et al.*, "A personalized and evolutionary algorithm for interpretable eeg epilepsy seizure prediction," *Scientific reports*, vol. 11, no. 1, pp. 1–12, 2021.

[301] R. E. Stirling, D. B. Grayden, W. D'Souza, M. J. Cook, E. Nurse, D. R. Freestone, D. E. Payne, B. H. Brinkmann, T. Pal Attia, P. F. Viana *et al.*, "Forecasting seizure likelihood with wearable technology," *Frontiers in neurology*, p. 1170, 2021.

[302] P. Peng, Y. Song, and L. Yang, "Seizure prediction in eeg signals using stft and domain adaptation," *Frontiers in Neuroscience*, p. 1880, 2021.

[303] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Artificial Neural Networks and Machine Learning – ICANN 2014*, S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and A. E. P. Villa, Eds. Cham: Springer International Publishing, 2014, pp. 281–290.

[304] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. R. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, and D. Drouin, "In-memory vector-matrix multiplication in monolithic complementary metal–oxide–semiconductor-memristor integrated circuits: Design choices, challenges, and perspectives," *Advanced Intelligent Systems*, vol. 2, no. 11, p. 2000115, 2020.

[305] H. Kim, M. R. Mahmoodi, H. Nili, and D. B. Strukov, "4k-memristor analog-grade passive crossbar circuit," *Nature Communications*, vol. 12, no. 1, p. 5198, Aug. 2021. [Online]. Available: https://doi.org/10.1038/s41467-021-25455-0

[306] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of Nonlinear Deterministic and Finite-dimensional Structures in Time Series of Brain Electrical Activity: Dependence on Recording Region and Brain State," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 64, p. 061907, Dec. 2001.

[307] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet," *Circulation*, vol. 101, no. 23, pp. e215–e220, Aug. 2021.

[308] A. Burrello, L. Cavigelli, K. Schindler, L. Benini, and A. Rahimi, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 752–757.

[309] N. D. Truong, A. D. Nguyen, L. Kuhlmann, M. R. Bonyadi, J. Yang, S. Ippolito, and O. Kavehei, "Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram," *Neural Networks*, vol. 105, pp. 104–111, Sep. 2018.

[310] K. Chellapilla, S. Puri, and P. Y. Simard, "High Performance Convolutional Neural Networks for Document Processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, G. Lorette, Ed., Université de Rennes 1. Suvisoft, Oct. 2006.

[311] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Transactions on Electron Devices*, vol. 60, no. 4, pp. 1318–1326, 2013.

[312] J. Cheng, L. Liang, J. H. Park, H. Yan, and K. Li, "A Dynamic Event-Triggered Approach to State Estimation for Switched Memristive Neural Networks With Nonhomogeneous Sojourn Probabilities," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 12, pp. 4924–4934, 2021.

[313] W. Shim, Y. Luo, J. Seo, and S. Yu, "Impact of Read Disturb on Multilevel RRAM based Inference Engine: Experiments and Model Prediction," in *IEEE International Reliability Physics Symposium (IRPS)*, Dallas, TX, Apr. 2020.

[314] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Detection, diagnosis, and repair of faults in memristor-based memories," in *2014 IEEE 32nd VLSI Test Symposium (VTS)*. IEEE, 2014, pp. 1–6.

[315] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[316] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in rram computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, 2017.

[317] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-faults in memristor crossbar arrays using matrix transformations," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, USA: ACM, 2019, pp. 438–443. [Online]. Available: https://doi.org/10.1145/3287624.3287707

[318] I. Yeo, M. Chu, S.-G. Gi, H. Hwang, and B.-G. Lee, "Stuck-at-fault tolerant schemes for memristor crossbar array-based neural networks," *IEEE Transactions on Electron Devices*, vol. 66, no. 7, pp. 2937–2945, 2019.

[319] T. Wen and Z. Zhang, "Deep convolution neural network and autoencoders-based unsupervised feature learning of eeg signals," *IEEE Access*, vol. 6, pp. 25 399–25 410, 2018.

[320] M. S. Hossain, S. U. Amin, M. Alsulaiman, and G. Muhammad, "Applying deep learning for epilepsy seizure detection and brain mapping visualization," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 15, no. 1s, pp. 1–17, 2019.

[321] J. Cao, J. Zhu, W. Hu, and A. Kummert, "Epileptic signal classification with deep eeg features by stacked cnns," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 12, no. 4, pp. 709–722, 2019.

[322] X. Tian, Z. Deng, W. Ying, K.-S. Choi, D. Wu, B. Qin, J. Wang, H. Shen, and S. Wang, "Deep multi-view feature learning for eeg-based epileptic seizure detection," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 10, pp. 1962–1972, 2019.

[323] W. Liang, H. Pei, Q. Cai, and Y. Wang, "Scalp eeg epileptogenic zone recognition and localization based on long-term recurrent convolutional network," *Neurocomputing*, vol. 396, pp. 569–576, 2020.

[324] A. Burrello, S. Benatti, K. Schindler, L. Benini, and A. Rahimi, "An ensemble of hyperdimensional classifiers: Hardware-friendly short-latency seizure detection with automatic ieeg electrode selection," *IEEE journal of biomedical and health informatics*, vol. 25, no. 4, pp. 935–946, 2020.

[325] A. Pappalardo, "Xilinx/brevitas," 2021.

[326] A. Abdelhameed and M. Bayoumi, "A deep learning approach for automatic seizure detection in children with epilepsy," *Frontiers in Computational Neuroscience*, vol. 15, p. 29, 2021.

[327] M. A. B. Altaf, J. Tillak, Y. Kifle, and J. Yoo, "A 1.83$/micro$j/classification nonlinear support-vector-machine-based patient-specific seizure classification soc," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2013, pp. 100–101.

[328] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, 2013.

[329] W.-M. Chen, H. Chiueh, T.-J. Chen, C.-L. Ho, C. Jeng, M.-D. Ker, C.-Y. Lin, Y.-C. Huang, C.-W. Chou, T.-Y. Fan, M.-S. Cheng, Y.-L. Hsin, S.-F. Liang, Y.-L. Wang, F.-Z. Shaw, Y.-H. Huang, C.-H. Yang, and C.-Y. Wu, "A fully integrated 8-channel closed-loop neural-prosthetic cmos soc for real-time epileptic seizure control," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 232–247, 2014.

[330] M. A. Bin Altaf and J. Yoo, "A 1.83μj/classification, 8-channel, patient-specific epileptic seizure classification soc using a non-linear support vector machine," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 1, pp. 49–60, 2016.

[331] S.-A. Huang, K.-C. Chang, H.-H. Liou, and C.-H. Yang, "A 1.9-mw svm processor with on-chip active learning for epileptic seizure control," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 2, pp. 452–464, 2020.

[332] A. Uran, K. Ture, C. Aprile, A. Trouillet, F. Fallegger, A. Emami, S. P. Lacour, C. Dehollain, Y. Leblebici, and V. Cevher, "A 16-channel wireless neural recording system-on-chip with cht feature extraction processor in 65nm cmos," in *2021 IEEE Custom Integrated Circuits Conference (CICC)*, 2021, pp. 1–2.

[333] H. G. Daoud, A. M. Abdelhameed, and M. Bayoumi, "Fpga implementation of high accuracy automatic epileptic seizure detection system," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 407–410.

[334] M. Ronchini, M. Zamani, H. A. Huynh, Y. Rezaeiyan, G. Panuccio, H. Farkhani, and F. Moradi, "A CMOS-based neuromorphic device for seizure detection from LFP signals," *Journal of Physics D: Applied Physics*, vol. 55, no. 1, p. 014001, oct 2021.

[335] S. Lv, J. Liu, and Z. Geng, "Application of memristors in hardware security: A current state-of-the-art technology," *Advanced Intelligent Systems*, vol. 3, no. 1, p. 2000127, 2021.

[336] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, "Compute-in-memory chips for deep learning: Recent trends and prospects," *IEEE Circuits and Systems Magazine*, vol. 21, no. 3, pp. 31–56, 2021.

[337] M. Yoshioka, M. Kudo, K. Gotoh, and Y. Watanabe, "A 10 b 125 ms/s 40 mw pipelined adc in 0.18 /spl mu/m cmos," in *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005.*, 2005, pp. 282–598 Vol. 1.

[338] J. Jung, K.-H. Baek, S.-I. Lim, S. Kim, and S.-M. Kang, "Design of a 6 bit 1.25 gs/s dac for wpan," in *2008 IEEE International Symposium on Circuits and Systems*, 2008, pp. 2262–2265.

[339] M. Giordano, G. Cristiano, K. Ishibashi, S. Ambrogio, H. Tsai, G. W. Burr, and P. Narayanan, "Analog-to-digital conversion with reconfigurable function mapping for neural networks activation function acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 367–376, 2019.

[340] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 678–681.

[341] P.-E. Gaillardon, M. H. Ben-Jamaa, F. Clermidy, and I. O'Connor, "Evaluation of a crossbar multiplexer in a lithography-based nanowire technology," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, 2011, pp. 2930–2933.

[342] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 14–26.

[343] K. Govindarajan and V. S. K. Bhaaskaran, "Borrow Select Subtractor for Low Power and Area Efficiency," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 518–523.

[344] S. Ganesan *et al.*, "Area, delay and power comparison of adder topologies," Ph.D. dissertation, 2015.

[345] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[346] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, jun 2017.

[347] A. Dozortsev, I. Goldshtein, and S. Kvatinsky, "Analysis of the row grounding technique in a memristor-based crossbar array," *International Journal of Circuit Theory and Applications*, vol. 46, no. 1, pp. 122–137, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2399

[348] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memristor-based rram cross-point structures," in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6.

[349] M. ElAnsary, J. Xu, J. Sales Filho, G. Dutta, L. Long, C. Tejeiro, A. Shoukry, C. Tang, E. Kilinc, J. Joshi *et al.*, "Bidirectional peripheral nerve interface with 64 second-order opamp-less adcs and fully integrated wireless power/data transmission," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 11, pp. 3247–3262, 2021.

[350] O. Krestinskaya, A. P. James, and L. O. Chua, "Neuromemristive circuits for edge computing: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 1, pp. 4–23, 2020.

[351] Y. Chen, "Reram: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[352] S. Mittal, "A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, p. 75, 2018.

[353] B. Muthuswamy and P. Kokate, "Memristor-based chaotic circuits," *IETE Technical Review*, vol. 26, 11 2009.

[354] C. Zheng, J. K. Eshraghian, A. James, and H. H.-C. Iu, "Chaotic oscillator using coupled memristive pairs," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 462–465.

[355] H. Jiang, D. Belkin, S. E. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, Q. Wu, J. J. Yang, and Q. Xia, "A novel true random number generator based on a stochastic diffusive memristor," *Nature Communications*, vol. 8, no. 1, p. 882, 2017. [Online]. Available: https://doi.org/10.1038/s41467-017-00869-x

[356] J. M. de Aguiar and S. P. Khatri, "Exploring the viability of stochastic computing," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 391–394.

[357] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283–293, 2014.

[358] R. Naous, M. Al-Shedivat, and K. N. Salama, "Stochasticity modeling in memristors," *IEEE Transactions on Nanotechnology*, vol. 15, no. 1, pp. 15–28, 2016.

[359] S. Ruder, "An overview of gradient descent optimization algorithms," 2016. [Online]. Available: https://arxiv.org/abs/1609.04747

[360] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[361] S. Gaba, F. Cai, J. Zhou, and W. D. Lu, "Ultralow sub-1-na operating current resistive memory with intrinsic non-linear characteristics," *IEEE Electron Device Letters*, vol. 35, no. 12, pp. 1239–1241, 2014.

[362] M. Shoba and R. Nakkeeran, "Energy and area efficient hierarchy multiplier architecture based on vedic mathematics and gdi logic," *Engineering Science and Technology, an International Journal*, vol. 20, no. 1, pp. 321–331, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098616303202

[363] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor Crossbar-Based Neuromorphic Computing System: A Case Study," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 10, pp. 1864–1878, Oct. 2014.

[364] G. C. Adam, A. Khiat, and T. Prodromakis, "Challenges Hindering Memristive Neuromorphic Hardware from Going Mainstream," *Nature communications*, vol. 9, no. 1, p. 5267, 2018.

[365] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip Training of Memristor Based Deep Neural Networks," in *Proc. International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, May. 2017, pp. 3527–3534.

[366] H. Jeong and L. Shi, "Memristor devices for neural networks," *Journal of Physics D: Applied Physics*, vol. 52, no. 2, p. 023003, Oct. 2018. [Online]. Available: https://doi.org/10.1088%2F1361-6463%2Faae223

[367] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "coreylammie/MemTorch: Initial Release," 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3760696

[368] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "RAPIDNN: In-Memory Deep Neural Network Acceleration Framework," *CoRR*, vol. abs/1806.05794, 2018. [Online]. Available: http://arxiv.org/abs/1806.05794

[369] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *2017 IEEE International Electron Devices Meeting (IEDM)*, Dec. 2017, pp. 6.1.1–6.1.4.

[370] Z. Jiang, S. Yu, Y. Wu, J. H. Engel, X. Guan, and H. . P. Wong, "Verilog-a compact model for oxide-based resistive random access memory (rram)," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Yokohama, Japan., Sep. 2014, pp. 41–44.

[371] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.

[372] I. Messaris, A. Serb, S. Stathopoulos, A. Khiat, S. Nikolaidis, and T. Prodromakis, "A Data-Driven Verilog-A ReRAM Model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3151–3162, Dec. 2018.

[373] V. A. Slipko and Y. V. Pershin, "Importance of the Window Function Choice for the Predictive Modelling of Memristors," *CoRR*, vol. abs/1811.06649, 2018. [Online]. Available: http://arxiv.org/abs/1811.06649

[374] Z. Biolek, D. Biolek, and V. Biolková, "Spice Model of Memristor With Nonlinear Dopant Drift," *Radioengineering*, pp. 210–214, 2009.

[375] Y. N. Joglekar and S. J. Wolf, "The Elusive Memristor: Properties of Basic Electrical Circuits," *European Journal of Physics*, vol. 30, no. 4, pp. 661–675, May. 2009.

[376] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A Versatile Memristor Model With Nonlinear Dopant Kinetics," *IEEE Transactions on Electron Devices*, vol. 58, no. 9, pp. 3099–3105, Sep. 2011.

[377] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern Classification by Memristive Crossbar Circuits using ex situ and in situ Training," *Nature Communications*, vol. 4, no. 1, p. 2072, 2013.

[378] S. N. Truong and K.-S. Min, "New Memristor-based Crossbar Array Architecture with 50-% Area Reduction and 48-% Power Saving for Matrix-vector Multiplication of Analog Neuromorphic Computing," *Journal of semiconductor technology and science*, vol. 14, no. 3, pp. 356–363, 2014.

[379] D. J. Mountain, M. R. McLean, and C. D. Krieger, "Memristor Crossbar Tiles in a Flexible, General Purpose Neural Processor," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 137–145, 2018.

[380] I. E. Ebong and P. Mazumder, "Self-Controlled Writing and Erasing in a Memristor Crossbar Memory," *IEEE Transactions on Nanotechnology*, vol. 10, no. 6, pp. 1454–1463, Nov. 2011.

[381] M. S. Tarkov, "Mapping Neural Network Computations onto Memristor Crossbar," in *Proc. International Siberian Conference on Control and Communications (SIBCON)*, Omsk, Russia, May 2015, pp. 1–4.

[382] R. Hasan, C. Yakopcic, and T. M. Taha, "Ex-situ Training of Dense Memristor Crossbar for Neuromorphic Applications," in *Proc. IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Beijing, China, 2015, pp. 75–81.

[383] K. Jo, C. Jung, K. Min, and S. Kang, "Self-Adaptive Write Circuit for Low-Power and Variation-Tolerant Memristors," *IEEE Transactions on Nanotechnology*, vol. 9, no. 6, pp. 675–678, Nov. 2010.

[384] B. Feinberg, S. Wang, and E. Ipek, "Making Memristive Neural Network Accelerators Reliable," in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018, pp. 52–65.

[385] X. S. Li and M. Shao, "A Supernodal Approach to Incomplete LU Factorization with Partial Pivoting," *ACM Trans. Math. Softw.*, vol. 37, no. 4, feb 2011. [Online]. Available: https://doi.org/10.1145/1916461.1916467

[386] P. Matstoms, "Sparse QR Factorization in MATLAB," *ACM Trans. Math. Softw.*, vol. 20, no. 1, p. 136–159, mar 1994. [Online]. Available: https://doi.org/10.1145/174603.174408

[387] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 — Seamless Operability Between C++11 and Python," 2016, https://github.com/pybind/pybind11.

[388] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[389] O. Krestinskaya, A. Irmanova, and A. P. James, "Memristive Non-Idealities: Is there any Practical Implications for Designing Neural Network Chips?" in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, Japan, May. 2019, pp. 1–5.

[390] E. Miranda, A. Mehonic, W. H. Ng, and A. J. Kenyon, "Simulation of Cycle-to-Cycle Instabilities in SiO $_x$ -Based ReRAM Devices Using a Self-Correlated Process With Long-Term Variation," *IEEE Electron Device Letters*, vol. 40, no. 1, pp. 28–31, 2019.

[391] W. Yi, S. E. Savel'ev, G. Medeiros-Ribeiro, F. Miao, M.-X. Zhang, J. J. Yang, A. M. Bratkovsky, and R. S. Williams, "Quantized Conductance Coincides with State Instability and Excess Noise in Tantalum Oxide Memristors," *Nature Communications*, vol. 7, no. 1, p. 11142, Apr. 2016. [Online]. Available: https://doi.org/10.1038/ncomms11142

[392] S. Yu, "Neuro-inspired Computing with Emerging Nonvolatile Memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, Feb. 2018.

[393] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ACM/IEEE International Symposium on Computer Architecture*, 2016.

[394] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Optimizing Latency, Energy, and Reliability of 1T1R ReRAM Through Cross-Layer Techniques," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 3, p. 352, 2016.

[395] M. Valad Beigi and G. Memik, "THOR: THermal-aware Optimizations for extending ReRAM Lifetime," in *IEEE International Parallel and Distributed Processing Symposium*, 2018.

[396] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Programming Strategies to Improve Energy Efficiency and Reliability of ReRAM Memory Systems," in *IEEE Workshop on Signal Processing Systems*, 2015.

[397] M. Zhao, B. Gao, Y. Xi, F. Xu, H. Wu, and H. Qian, "Endurance and Retention Degradation of Intermediate Levels in Filamentary Analog RRAM," *IEEE Journal of the Electron Devices Society*, vol. 7, p. 1239, 2019.

[398] M. Zhao *et al.*, "Impact of Switching Window on Endurance Degradation in Analog RRAM," in *Electron Devices Technology and Manufacturing Conference*, 2019.

[399] Y. Xiang, P. Huang, Y. Zhao, M. Zhao, B. Gao, H. Wu, H. Qian, X. Liu, and J. Kang, "Impacts of State Instability and Retention Failure of Filamentary Analog RRAM on the Performance of Deep Neural Network," *IEEE Transactions on Electron Devices*, vol. 66, no. 11, p. 4517, 2019.

[400] M. Zhao *et al.*, "Investigation of statistical retention of filamentary analog RRAM for neuromophic computing," in *IEEE International Electron Devices Meeting*, 2017.

[401] A. Grossi *et al.*, "Resistive RAM Endurance: Array-Level Characterization and Correction Techniques Targeting Deep Learning Applications," *IEEE Transactions on Electron Devices*, vol. 66, no. 3, p. 1281, 2019.

[402] Y. Sharma, P. Misra, and R. S. Katiyar, "Unipolar Resistive Switching Behavior of Amorphous YCrO3 Films for Nonvolatile Memory Applications," *Journal of Applied Physics*, vol. 116, no. 8, p. 084505, 2014.

[403] Y. Y. Chen *et al.*, "Balancing SET/RESET Pulse for $> 10^{10}$ Endurance in HfO_2/Hf 1T1R Bipolar RRAM," *IEEE Transactions on Electron Devices*, vol. 59, no. 12, p. 3243, 2012.

[404] D. Alfaro Robayo, G. Sassine, Q. Rafhay, G. Ghibaudo, G. Molas, and E. Nowak, "Endurance Statistical Behavior of Resistive Memories Based on Experimental and Theoretical Investigation," *IEEE Transactions on Electron Devices*, vol. 66, no. 8, p. 3318, 2019.

[405] P. Huang, Y. C. Xiang, Y. D. Zhao, C. Liu, B. Gao, H. Q. Wu, H. Qian, X. Y. Liu, and J. F. Kang, "Analytic Model for Statistical State Instability and Retention Behaviors of Filamentary Analog RRAM Array and Its Applications in Design of Neural Network," in *IEEE International Electron Devices Meeting*, 2018.

[406] C. Nail *et al.*, "Understanding RRAM Endurance, Retention and Window Margin Tradeoff using Experimental Results and Simulations," in *IEEE International Electron Devices Meeting*, 2016.

[407] Z. Wei *et al.*, "Retention Model for High-Density ReRAM," in *IEEE International Memory Workshop*, 2012.

[408] T. Cabout *et al.*, "Temperature Impact (up to 200 °C) on Performance and Reliability of HfO2-based RRAMs," in *IEEE International Memory Workshop*, 2013.

[409] P. Gonzalez-de Santos, A. Ribeiro, C. Fernandez-Quintanilla, F. Lopez-Granados, M. Brandstoetter, S. Tomic, S. Pedrazzi, A. Peruzzi, G. Pajares, G. Kaplanis, M. Perez-Ruiz, C. Valero, J. del Cerro, M. Vieri, G. Rabatel, and B. Debilde, "Fleets of robots for environmentally-safe pest control in agriculture," *Precision Agriculture*, vol. 18, no. 4, pp. 574–614, 2017. [Online]. Available: https://doi.org/10.1007/s11119-016-9476-3

[410] C. Fernández-Quintanilla, J. M. Peña, D. Andújar, J. Dorado, A. Ribeiro, and F. López-Granados, "Is the current state of the art of weed monitoring suitable for site-specific weed management in arable crops?" *Weed Research*, vol. 58, no. 4, pp. 259–272, May 2018. [Online]. Available: https://doi.org/10.1111%2Fwre.12307

[411] D. L. Shaner and H. J. Beckie, "The future for weed control and technology," *Pest Management Science*, vol. 70, no. 9, pp. 1329–1339, jan 2014. [Online]. Available: https://doi.org/10.1002%2Fps.3706

[412] D. Slaughter, D. Giles, and D. Downey, "Autonomous robotic weed control systems: A review," *Computers and Electronics in Agriculture*, vol. 61, no. 1, pp. 63–78, apr 2008. [Online]. Available: https://doi.org/10.1016%2Fj.compag.2007.05.008

[413] A. dos Santos Ferreira, D. M. Freitas, G. G. da Silva, H. Pistori, and M. T. Folhes, "Weed detection in soybean crops using ConvNets," *Computers and Electronics in Agriculture*, vol. 143, pp. 314–324, dec 2017. [Online]. Available: https://doi.org/10.1016%2Fj.compag.2017.10.027

[414] S. H. Lee, C. S. Chan, S. J. Mayo, and P. Remagnino, "How deep learning extracts and learns leaf features for plant classification," *Pattern Recognition*, vol. 71, pp. 1–13, nov 2017. [Online]. Available: https://doi.org/10.1016%2Fj.patcog.2017.05.015

[415] S. H. Lee, C. S. Chan, P. Wilkin, and P. Remagnino, "Deep-plant: Plant identification with convolutional neural networks," in *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2015. [Online]. Available: https://doi.org/10.1109%2Ficip.2015.7350839

[416] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, B. Calvert, M. R. Azghadi, and R. D. White, "DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning," *Scientific Reports*, vol. 9, no. 1, p. 2058, feb 2019. [Online]. Available: https://doi.org/10.1038/s41598-018-38343-3

[417] A. Bakhshipour and A. Jafari, "Evaluation of support vector machine and artificial neural networks in weed detection using shape features," *Computers and Electronics in Agriculture*, vol. 145, pp. 153–160, feb 2018. [Online]. Available: https://doi.org/10.1016%2Fj.compag.2017.12.032

[418] M. Dyrmann, R. Jørgensen, and H. Midtiby, "RoboWeedSupport - detection of weed locations in leaf occluded cereal crops using a fully convolutional neural network," *Advances in Animal Biosciences*, vol. 8, no. 2, pp. 842–847, 2017. [Online]. Available: https://doi.org/10.1017%2Fs2040470017000206

[419] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," in *2007 IEEE International Symposium on Signal Processing and Information Technology*. IEEE, dec 2007. [Online]. Available: https://doi.org/10.1109%2Fisspit.2007.4458016

[420] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. B. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in *Computer Vision – ECCV 2012*. Springer Berlin Heidelberg, 2012, pp. 502–516. [Online]. Available: https://doi.org/10.1007%2F978-3-642-33709-3_36

[421] D. Hall, C. McCool, F. Dayoub, N. Sunderhauf, and B. Upcroft, "Evaluation of features for leaf classification in challenging conditions," in *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE, jan 2015. [Online]. Available: https://doi.org/10.1109%2Fwacv.2015.111

[422] C. Kalyoncu and Önsen Toygar, "Geometric leaf classification," *Computer Vision and Image Understanding*, vol. 133, pp. 102–109, apr 2015. [Online]. Available: https://doi.org/10.1016%2Fj.cviu.2014.11.001

[423] J. Carranza-Rojas, H. Goeau, P. Bonnet, E. Mata-Montero, and A. Joly, "Going deeper in the automated identification of herbarium specimens," *BMC Evolutionary Biology*, vol. 17, no. 1, aug 2017. [Online]. Available: https://doi.org/10.1186%2Fs12862-017-1014-z

[424] A. Shirzadifar, S. Bajwa, S. A. Mireei, K. Howatt, and J. Nowatzki, "Weed species discrimination based on SIMCA analysis of plant canopy spectral data," *Biosystems Engineering*, vol. 171, pp. 143–154, jul 2018. [Online]. Available: https://doi.org/10.1016%2Fj.biosystemseng.2018.04.019

[425] M. Louargant, G. Jones, R. Faroux, J.-N. Paoli, T. Maillot, C. Gée, and S. Villette, "Unsupervised classification algorithm for early weed detection in row-crops by combining spatial and spectral information," *Remote Sensing*, vol. 10, no. 5, p. 761, may 2018. [Online]. Available: https://doi.org/10.3390%2Frs10050761

[426] F. Lin, D. Zhang, Y. Huang, X. Wang, and X. Chen, "Detection of corn and weed species by the combination of spectral, shape and textural features," *Sustainability*, vol. 9, no. 8, p. 1335, aug 2017. [Online]. Available: https://doi.org/10.3390%2Fsu9081335

[427] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2016. [Online]. Available: https://doi.org/10.1109%2Fcvpr.2016.90

[428] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural*

Information Processing Systems - Volume 1, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[429] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, p. 1237–1242.

[430] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *2011 International Conference on Document Analysis and Recognition*. IEEE, sep 2011. [Online]. Available: https://doi.org/10.1109%2Ficdar.2011.229

[431] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," *ArXiv*, vol. abs/1102.0183, 2011.

[432] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," 2012. [Online]. Available: https://arxiv.org/abs/1202.2745

[433] F. J. Knoll, M. Grelcke, V. Czymmek, T. Holtorf, and S. Hussmann, "CPU architecture for a fast and energy-saving calculation of convolution neural networks," in *SPIE Proceedings*, B. C. Kress, W. Osten, and H. P. Urbach, Eds. SPIE, jun 2017. [Online]. Available: https://doi.org/10.1117%2F12.2270282

[434] A. Milioto, P. Lottes, and C. Stachniss, "REAL-TIME BLOB-WISE SUGAR BEETS VS WEEDS CLASSIFICATION FOR MONITORING FIELDS USING CONVOLUTIONAL NEURAL NETWORKS," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-2/W3, pp. 41–48, aug 2017. [Online]. Available: https://doi.org/10.5194%2Fisprs-annals-iv-2-w3-41-2017

[435] I. Sa, Z. Chen, M. Popovic, R. Khanna, F. Liebisch, J. Nieto, and R. Siegwart, "weedNet: Dense semantic weed classification using multispectral images and MAV for smart farming," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 588–595, jan 2018. [Online]. Available: https://doi.org/10.1109%2Flra.2017.2774979

[436] A. Milioto, P. Lottes, and C. Stachniss, "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2018. [Online]. Available: https://doi.org/10.1109%2Ficra.2018.8460962

[437] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs suitable for edge computing?" in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA:

USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/biookaghazadeh

[438] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, feb 2017. [Online]. Available: https://doi.org/10.1145%2F3020078.3021740

[439] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2, pp. 1–27, sep 2013. [Online]. Available: https://doi.org/10.1145%2F2514740

[440] H. Abdelkrim, S. B. Othman, and S. B. Saoud, "Reconfigurable SoC FPGA based: Overview and trends," in *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. IEEE, jan 2017. [Online]. Available: https://doi.org/10.1109%2Faset.2017.7983723

[441] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, may 2016. [Online]. Available: https://doi.org/10.1109%2Ffccm.2016.22

[442] 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*. ACM, jun 2017. [Online]. Available: https://doi.org/10.1145%2F3061639.3062244

[443] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "FPGA-based CNN inference accelerator synthesized from multi-threaded c software," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*. IEEE, sep 2017. [Online]. Available: https://doi.org/10.1109%2Fsocc.2017.8226056

[444] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, jan 2018. [Online]. Available: https://doi.org/10.1109%2Ftcad.2017.2705069

[445] K. Duan, S. S. Keerthi, W. Chu, S. K. Shevade, and A. N. Poo, "Multi-category classification by soft-max combination of binary classifiers," in *Multiple Classifier Systems*, T. Windeatt and F. Roli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 125–134.

[446] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *ArXiv*, vol. abs/1602.02830, 2016.

[447] A. Y. Ng, "Feature selection, il/isub1/sub vs. il/isub2/sub regularization, and rotational invariance," in *Twenty-first international conference on Machine learning - ICML '04*. ACM Press, 2004. [Online]. Available: https://doi.org/10.1145%2F1015330.1015435

[448] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, jan 2014.

[449] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *ArXiv*, vol. abs/1712.04621, 2017.

[450] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, p. 2625–2631.

[451] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109%2F5.726791

[452] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[453] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[454] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166

[455] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[456] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6869–6898, jan 2017.

[457] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1501.02876

[458] G. Mishra, Y. L. Aung, M. Wu, S.-K. Lam, and T. Srikanthan, "Real-time image resizing hardware accelerator for object detection algorithms," in *2013 International Symposium on Electronic System Design*. IEEE, dec 2013. [Online]. Available: https://doi.org/10.1109%2Fised.2013.26

[459] J. Burns and L. Chang, "Meet the ibm artificial intelligence unit," Oct. 2022. [Online]. Available: https://research.ibm.com/blog/ibm-artificial-intelligence-unit-aiu

[460] M. Lanza, A. Sebastian, W. D. Lu, M. L. Gallo, M.-F. Chang, D. Akinwande, F. M. Puglisi, H. N. Alshareef, M. Liu, and J. B. Roldan, "Memristive technologies for data storage, computation, encryption, and radio-frequency communication," *Science*, vol. 376, no. 6597, Jun. 2022. [Online]. Available: https://doi.org/10.1126/science.abj9979

[461] C. Frenkel, D. Bol, and G. Indiveri, "Bottom-up and top-down neural processing systems design: Neuromorphic intelligence as the convergence of natural and artificial intelligence," *CoRR*, vol. abs/2106.01288, 2021. [Online]. Available: https://arxiv.org/abs/2106.01288

[462] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.