AGING-AWARE MEMRISTOR CROSSBAR FOR RELIABLE AND ENERGY-EFFICIENT DEEP LEARNING ACCELERATION

A Thesis

by

AUROSMITA KHANSAMA

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2021

Major Subject: Computer Engineering

ABSTRACT

Deep learning algorithms are highly energy and memory-intensive as their performance increases with an increasing amount of data. Moore’s law coming to an end and the ever-increasing demand for high computational power by Deep Learning algorithms are becoming a major issue. Another factor slowing down the fast Deep learning algorithms is the interconnect delay. This calls for a modern computing architecture that doesn’t physically separate memory and computation elements as done in Von Neumann's architecture. Memristor, the fourth fundamental circuit element, comes to the rescue. Owing to its less power consumption, more efficient and non-volatile nature, memristors claim to be a possible replacement for DRAM. Another advantage of memristor design is that it can be arranged in a crossbar arrangement. This makes it suitable to perform the dot-product operation and can be used in Convolutional Neural Network (CNN) architecture. BPhoton-CNN, proposed in this work, is a memristor-based CNN architecture that uses photonic Backpropagation for designing a complete analog system for training and inference. Despite showing the characteristics of a highly promising device for in-situ computing, memristive devices suffer from reliability issues given their non-linear nature. The proposed work also discusses the effect of one such non-linear characteristic called Aging. The effect of aging on the performance of deep learning accelerators and different methods to counter aging have been proposed in this work.

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I thank my thesis advisor, Dr. Rabi N. Mahapatra for his continuous guidance and support throughout my graduate studies. He has always inspired me and assisted me in smoothly carrying out all the required steps. I would also like to thank my thesis committee members, Dr. Duncan Walker, and Dr. Samuel Palermo for agreeing to be on the committee and for providing valuable suggestions.

I am thankful to Dr. Dharanidhar for his constant motivation and valuable feedback. His drive to do something impactful in life has taught me that determination is of the utmost importance for becoming a better engineer and better person. I would like to thank my amazing lab mates Dr. Jyotikrishna Dass, Syed Ali Hasnain, Karl Ott, and Jerry Yiu for helping me during the initial phases.

I am grateful to my parents, Santosh Khansama and Sasmita Khansama, and my brother Soumya Ranjan Khansama for always believing in me and motivating me to keep working hard. Thank you for always showing confidence in me and cheering me up. Whatever I am today, is because of your teachings and unconditional love.

Finally, I would like to thank my dear friends – Aman, Rohit, Flavia, Anushka, and Shreya. I would like to thank Rohit for those amazing brainstorming sessions. I learned a lot from those discussions. Flavia, Anushka, and Shreya, I had an amazing time as your flatmate. Lastly, I would like to thank Aman for always helping and motivating me.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professor Rabi N. Mahapatra[advisor] and Professor Duncan H. Walker of the Department of Computer Science & Engineering and Professor Samuel Palermo of the Department of Electrical & Computer Engineering.

The work presented in section 2 was conducted in part by Dr. Dharanidhar Dang of UCSD university.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a teaching assistantship and part-time student worker position from Computer Science and Engineering Department at Texas A&M University.

NOMENCLATURE

DNN	Deep Neural Network
DL	Deep Learning
ML	Machine Learning
AI	Artificial Intelligence
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
SOC	System-On-Chip
MDNN	Memristive Deep Neural Network

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Artificial Intelligence (AI) is ubiquitous in today's world. Deep learning (DL), a powerful AI method, supports a wide range of commercial and research aspects. Ranging from medical diagnostics to image processing to deep-sea exploration, DL has garnered special attention due to its ability to learn and predict with high accuracy and precision. The popularity of Deep learning is fueled by Big Data. The performance of a DL model increases almost exponentially with the amount of data.as seen in Fig.1.1 [1].

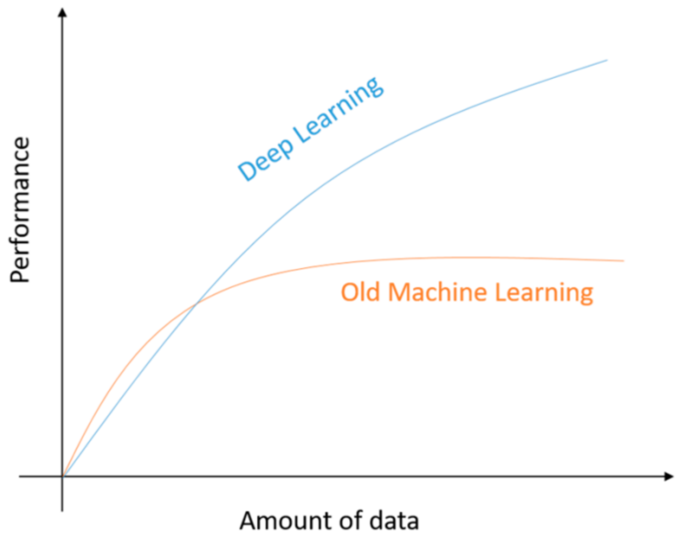


Figure 1.1: Performance of DL models w.r.t. amount of data. Reprinted from [1]

One of the most popular deep learning methods is Convolutional Neural Network (CNN) and it has been used in various application domains like computer vision and natural language processing (NLP). Convolution-based models have an exorbitant number of weight parameters. Take for instance the ResNet-50[2] architecture which has 23 million weight parameters. As the complexity of problems and demand for accurate predictions is increasing, the models are becoming larger and deeper with an appetite for a huge amount of data. In the future, we can expect to have even bigger networks that will be able to solve much more complex problems. This increases the demand for faster computational resources, additional memory storage, and faster communication between memory and computational resources. As Deep Neural Networks are becoming the driving core of a lot of critical applications, it becomes imperative to design adequate computational and memory resources to accommodate intensive applications.

1.1. Pitfalls of current modern architecture

In 1965, Gordon Moore predicted that the number of transistors was doubling every 24 months and would continue to do so [3]. But the recent decade is experiencing the end of Moore’s law. This is because as the transistor becomes smaller, leakage current increases leading to increased heat dissipation. To compensate for the performance loss due to the slowing down of device scaling, multi-core processors and GPUs have been used to perform complex computations.

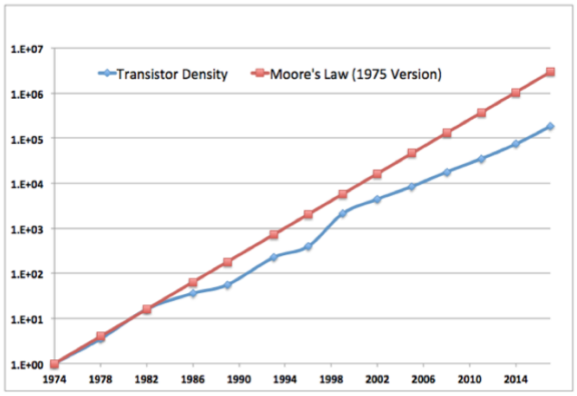


Figure 1.2: Moore’s Law slowdown in Intel Processor. Reprinted from [4]

Neural networks are trained on a large amount of data and require high bandwidth memory resources. Training a ResNet-50 model requires around 7.5 GB of local DRAM storage [59]. Over the last four decades, when Moore's law prevailed, there was significant growth in processor speeds without any substantial change in memory technology. Fig.1.3 shows an increasing gap between the processor's computational speed and memory performance.

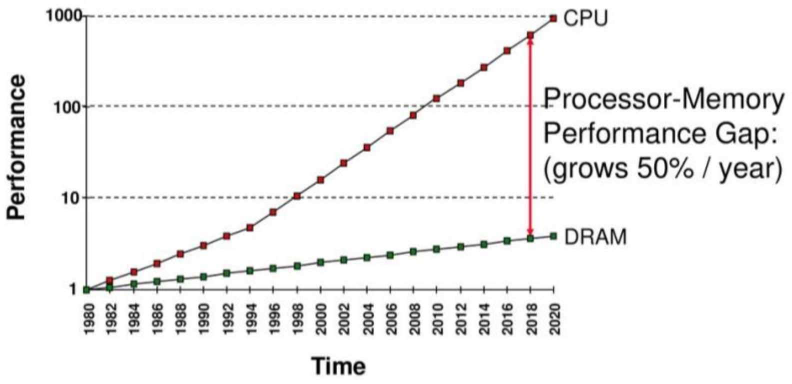


Figure 1.3: Processor-memory gap. Reprinted from [6]

To improve memory performance in modern systems many novel techniques are being adopted. Interconnects are improved to reduce data transfer latency between bandwidth agents [7,17] and CPU/GPU caches were introduced to reduce the average time and energy to access data from the main memory [8,9,18]. However, caches are also constrained as the increase in cache size causes more latency and complexity in the design. For example, a high-performance GPU usually has only 512KB of L1-cache with each core [10]. But DNNs train on huge datasets and strive for accurate predictions demanding enormous memory capacity (in the range of tens of gigabytes). Thus, data is stored in the DRAM or off-chip memory. During training, a neural network requires transferring data from off-chip memory devices to caches before it reaches the processing unit for computation. This data movement between CPU/GPU and DRAM consumes twice the energy as consumed by floating-point operations [11]. Off-chip memory is already constrained by bandwidth and latency. And now this continuous data transfer also increases the power consumption. Thus, the current architectures have design constraints that inhibit their ability to handle complex neural networks.

Gartner [12] predicts that transistor-based semiconductor technology will hit a digital wall by 2025. The research community has now started exploring new paradigms that are small in size yet energy efficient to be able to outperform traditional general-purpose hardware in power-hungry artificial intelligence applications. One such new approach is Neuromorphic or brain-inspired computing. The human brain is the most sophisticated computer in the world. It consists of about 100 billion neurons, making in the order of 100 trillion connections, and performs 100B TFlops per sec with just 20W power consumption [13]. Spiking Neural Networks (SNN)-an artificial neural network that closely mimics natural neural networks [14], TrueNorth by IBM-a brain-inspired machine using neuromorphic CMOS integrated circuit that consumes 70 milliwatts of power [15], and Loihi by Intel labs-an asynchronous spiking neural network (SNN) that is about 1000 times more energy-efficient than the conventional computation technique GPU [16] are few existing neuromorphic architectures that use analog component for computation.

However, all these architecture uses DRAM for data storage and thus incur high Analog-to-Digital (ADC) and Digital-to-Analog (DAC) power consumption and memory latency. They do solve the computational raised bottleneck, because of the decline in Moore's law, but still face the memory bottleneck.

So, the need of the hour is a system that doesn't physically separate memory and computation elements. This will reduce the latency and power consumption of moving data between memory and computing resources and will result in faster computation. This has shifted the interest towards a complete analog system that is energy-efficient and reliable.

1.2. Opportunities with Memristor

1.2.1. What is a memristor?

In 1971, Leon Chua described Memristor, arguing that it should be included along with resistor, capacitor, and inductor as the fourth fundamental circuit element [19]. Memristor is termed from Memory and Resistor because it's a resistor with the capability to store memory. It is an electrical resistance switch that can retain a state of internal resistance based on the history of applied voltage and current. Memristor was first realized by HP labs in 2008 [20].

Fig.1.4 shows the four circuit elements where the fourth element is the memristor. Memristance (M) of a memristor can be defined as [20]:

$$M(q) = d\varphi/dq = vdt/idt = v/i \qquad \text{(i)}$$

Where i is the electric current, v is voltage, q is charge and ϕ is magnetic flux.

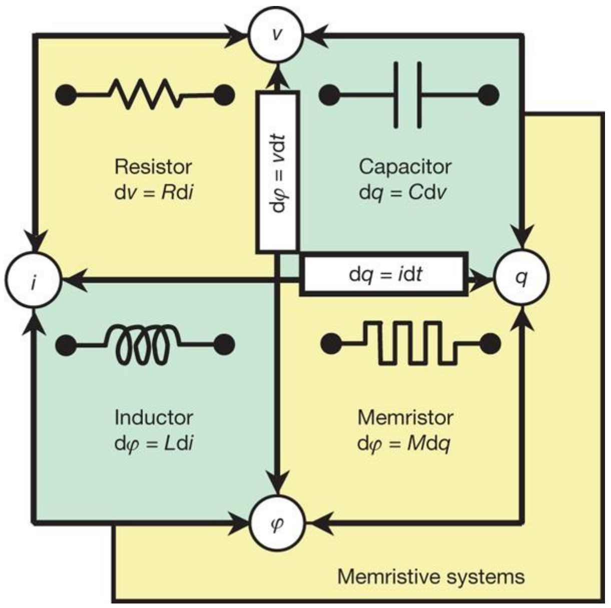


Figure 1.4: Four fundamental circuit elements. Reprinted from [20]

As proposed by HP lab in [20], memristors are made of a thin semiconductor film (TiO_2) containing two regions: One doped with oxygen vacancies called the doped region, and the other is the undoped region as seen in fig.1.5. This film is surrounded by two metal contacts. The doped region has low resistance R_{on} and the undoped region has high resistance, R_{off}. When a voltage is applied across the device, vacancies will move towards the undoped region, resulting in the shift of the boundary.

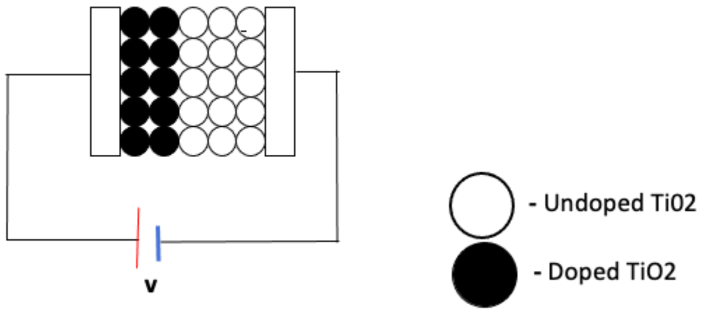


Figure 1.5: Memristor

1.2.2. Why Memristor?

Memristor devices are non-volatile in nature. So, they retain the value of resistance even if the voltage supply has been disconnected. Thus, memristor cells can be used to store information and unlike DRAM, where capacitors drain their charge after the power has been removed, memristor will store the values till another positive or negative voltage has been applied. As stated by R. Stanley Williams, a researcher from HP labs [21]:

> "Memristors can be made extremely small, and they function like synapses. Using them, we will be able to build analog electronic circuits that could fit in a shoebox and function according to the same physical principles as a brain…"

Also, Chua showed that it will take a circuit of around 15 transistors and other passive elements to emulate the behavior of a single memristor cell [21]. Various research has shown that Memristors use less power for operation and a memristor array consisting of millions of cells is 10 to 100 times faster than a GPU [22, 23].

Thus, memristor claims to be a possible replacement for DRAM, owing to its less power consumption and more efficiency. Moreover, memristors can also be used for computation along with memory storage.

A memristor cell can be designed in a crossbar arrangement and thus dot-product computation can be performed using Kirchhoff's current and voltage law. A dot product computation is the basis of a deep learning model as it is required while calculating the weighted summation of inputs. Further, memristor uses analog vector multiplication which is 100 X more efficient than traditional computational resources like GPU [31].

This has shifted the focus toward neuromorphic computing using memristor like ISAAC [28] and PipeLayer [30]. ISAAC uses a memristor crossbar array for each layer in the neural network in a highly pipelined manner and PipeLayer is an enhancement of ISAAC where the crossbar array size has been improved to improve the pipeline. Although this architecture shows promising results, they incur heavy area and power overhead due to inter-layer data conversions which require the use of Analog-to-Digital (ADC) and Digital-to-Analog (DAC) converters.

In earlier approaches, only one layer was made analog (mainly convolution layer in a CNN architecture), whereas the rest of the computation (in RELU and MaxPool) was still done in traditional processor and memory resources. So, the challenge is to develop a complete analog accelerator that does both training and inference.

2. BPHOTON-CNN – A COMPLETE ANALOG NEUROMORPHIC COMPUTING*

2.1. Motivation

[1]In today's era of big data, the volume of data that computing systems process is increasing exponentially. Deep learning has become state-of-the-art across a broad range of big data applications such as speech processing, image recognition, genomic prediction, etc. Convolutional neural networks (CNNs) are a popular deep learning framework with superior accuracy on applications that deal with videos and images. However, CNN's are highly energy and memory-intensive, requiring enormous computational resources. With Moore's law coming reaching its limit, traditional Von Neuman systems such as heterogeneous CPU/GPU platforms cannot offer this high computational demand, within reasonable power and processing time limitations. Therefore, several FPGA [27] and ASIC [28] approaches have been proposed to accomplish large-scale deep learning acceleration.

A CNN comprises of two stages: training and inference (i.e., validation). Most hardware accelerators for CNNs in prior literature focus only on the inference stage, while the training is done offline using GPUs. However, training a CNN is several hundred times more compute and power-intensive than its inference [29]. Moreover, for many applications, training is not a one-time activity, especially under changing environmental

*Reprinted with permission from "BPhoton-CNN: An Ultrafast Photonic Backpropagation Accelerator for Deep Learning" by Dang, D., Khansama, A., Mahapatra, R. and Sahoo, D., 2020. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pp. 27-32.

and system conditions, where re-training of CNN at regular intervals is essential to maintaining prediction accuracy for the application over time. This calls for an energy-efficient training accelerator in addition to the inference accelerator.

Training a CNN, in general, employs a backpropagation algorithm that demands high memory locality and computes parallelism. Recently, a few resistive memory (ReRAM or memristor crossbar) based training accelerators have been demonstrated for CNNs, e.g. ISAAC [28], PipeLayer [30], RCP [33], and MNN [34]. ISAAC, RCP, and MNN use highly parallel memristor crossbar arrays to address the need for parallel computations in CNNs. In addition, ISAAC uses a very deep pipeline to improve system throughput. However, this is only beneficial when a large number of consecutive images can be fed into the architecture. Unfortunately, during training, in many cases, a limited number of consecutive images need to be processed before weight updates. The deep pipeline in ISAAC also introduces frequent pipeline bubbles. Compared to ISAAC, PipeLayer demonstrates an improved pipeline approach to enhance throughput. However, RCP, MNN, ISAAC, and PipeLayer involve several analog-to-digital (AD) and digital-to-analog (DA) conversions which become a performance bottleneck, in addition to their large power consumption. Also, training in these accelerators involves sequential weight updates from one layer to another. This incurs inter-layer waiting time for synchronization, which reduces overall performance. This calls for an analog accelerator that can drastically reduce the number of AD/DA conversions, and inter-layer waiting time. It has been recently demonstrated that a completely analog matrix-vector multiplication is 100× more efficient than its digital counterpart implemented with an ASIC, FPGA, or GPU [31]. HP labs have showcased a memristor dot product engine that can achieve a speed-efficiency product of 1000× compared to a digital ASIC [31]. Vandroome et al. in [32] have demonstrated a small-scale efficient recurrent neural network using analog photonic computing. A few efficient on-chip photonic inference accelerators have also been proposed in [35], [36]. However, a full-fledged analog CNN accelerator that is capable of both training and inference has yet to be demonstrated.

In this section, we propose BPhoton, a novel silicon photonics-based backpropagation accelerator for training CNNs. BPhoton works in conjunction with a highly efficient memristor-integrated photonic feedforward CNN accelerator. We call it BPhoton-CNN which a first-of-its-kind memristor-integrated silicon photonic CNN accelerator for end-to-end analog training and inference. It is intended to perform highly energy-efficient and ultra-fast training for deep learning applications with state-of-the-art prediction accuracy.

2.2. Convolutional Neural Networks

2.2.1. Basics of CNN

Convolutional neural networks (CNNs) are a class of feed-forward neural networks commonly used for analyzing visual imagery for image classification and object detection/prediction tasks. CNN in general comprises of three types of layers: convolution layer (CONV), pooling layer (POOL), and a fully connected layer (FC). Generally, CONV is accompanied by a non-linear activation function, such as ReLU. Depending on the sequence in which these layers are arranged, there are different CNN models, such as AlexNet [37], VGG [38], LeNet [39], etc.

2.2.2. Backpropagation Algorithm

CNN operates in two stages: training and inference (testing). In the training phase, the filter weights (and biases) in CONV and FC layers are learned by using a backpropagation (BP) algorithm. The BP algorithm involves a forward and a backward pass in the deep network.

Given a training sample *x* in the forward pass, the weighted input sum (convolution) *z* is computed for neurons in each layer *l* with some initial filter weights *w* (and bias *b*) followed by neural activation $\sigma(z)$ (ReLU(*z*) in our work), and POOL. The final layer *L* computes the output label of the overall network for every forward pass. This can be summarized as follows:

Forward Pass: For each layer *l*,

$$\boldsymbol{z}^{x,l} \leftarrow \boldsymbol{w}^{l}\boldsymbol{a}^{x,l-1} + \boldsymbol{b}^{l} \qquad (1)$$

$$\boldsymbol{a}^{x,l} \leftarrow \boldsymbol{\sigma}\left(\boldsymbol{z}^{x,l}\right) \qquad (2)$$

The output error in the final prediction, $\delta^{x,L}$ is a result of errors induced by the neurons in each hidden layer during the forward pass. To compute the error contribution of a neuron in the previous layer i.e., $\delta^{x,l}$, the final error is backpropagated through the network starting from the output layer. This can be summarized as follows:

Output error: At the final layer L,

$$\delta^{x,L} \leftarrow \nabla_a C_x \odot \sigma'(z^{x,L}) \qquad (3)$$

Backward Pass: For each layer *l*,

$$\delta^{x,l} \leftarrow ((w^{l+1})^T \times \delta^{x,l+1}) \odot \sigma'(z^{x,l}) \qquad (4)$$

Here, ∇_a is the gradient of $a^{x,l}$, and $\sigma'(z^{x,L})$ is derivative of $\sigma(z^{x,L})$. These error contributions are necessary to update the filter weights *w* and biases *b* in the respective layers using a gradient descent method. In gradient descent, the forward and backward pass happen iteratively until the cost function is minimized and the network is trained. This can be summarized as follows:

Gradient Descent: For each layer l and m training samples with learning rate η,

$$w^l \leftarrow w^l - \frac{\eta}{m}\sum_x \delta^{x,l} \times (a^{x,l-1})^T \tag{5}$$

$$b^l \leftarrow b^l - \frac{\eta}{m}\sum_x \delta^{x,l} \tag{6}$$

The next section presents the details of the proposed BPhoton-CNN architecture.

2.3. BPHOTON-CNN Architecture

2.3.1. Overview

Our proposed BPhoton-CNN architecture is a fully analog, scalable, and configurable memristor-integrated photonic CNN accelerator design. Unlike previously proposed state-of-the-art CNN accelerators [28], [30], the BPhoton-CNN accelerator enables completely analog end-to-end training and testing for a CNN.

Fig.2.1 gives a high-level overview of this BPhoton-CNN architecture. As shown in the figure, BPhoton-CNN comprises of three parts: feedforward CNN accelerator architecture, backpropagation accelerator architecture, and weight update and peripheral circuitry. The analog feedforward CNN accelerator is inspired from [36]. It enables Feature Extraction (FE) through a memristive convolution layer and silicon photonics-based ReLU and pooling layers. The feedforward CNN accelerator uses memristive multiplication for Feature Classification (FC). The entire backpropagation accelerator is implemented in the photonic realm using MRMs, splitters, and multiplexers. Finally, BPhoton-CNN's weight update and peripheral circuitry are implemented through a group of memristors.

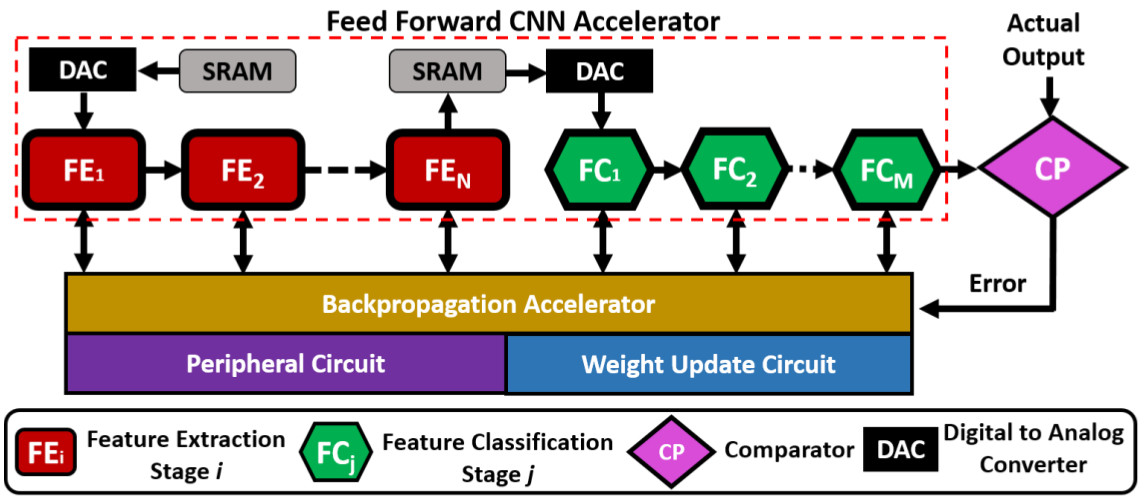

Feed Forward CNN Accelerator
DAC
SRAM
SRAM
DAC
FE₁
FE₂
FEN
FC₁
FC₂
FCM
Actual Output
CP
Error
Backpropagation Accelerator
Peripheral Circuit
Weight Update Circuit
FEi
Feature Extraction Stage i
FCj
Feature Classification Stage j
CP
Comparator
DAC
Digital to Analog Converter

Figure 2.1: An overview of *BPhoton-CNN* architecture.

2.3.1.1. Feedforward CNN Architecture

An image dataset is considered as the input data and its classification as the application to be executed with BPhoton-CNN. The CNN accelerator in BPhoton-CNN architecture (see Fig.2.1) performs feedforward feature extraction (FE) followed by feature classification of input images. The FE in the CNN architecture is carried out using multiple FE stages (FE_i). After all of the features are extracted, feature classification is performed using one or more fully-connected layers (FC).

Fig. 2.2 illustrates the microarchitecture of an FE stage. Each FE stage comprises of multiple memristor-based convolution layers (CONV), a semiconductor-optical amplifier (SOA)-based ReLU layer, an optical comparator-based max-pooling (POOL) layer, and an interface layer. BPhoton-CNN's FE adopts a completely analog computing paradigm by avoiding inter-layer A- to-D (Analog-to-Digital) and D-to-A (Digital-to-Analog) conversions compared to state-of-the-art CNN accelerators [28], [30] which use analog memristive convolution and digital CPU/GPU based ReLU and Pooling. The feedforward CNN accelerator is designed to convolve 56x56 image input at a time. The detailed working of this feedforward accelerator is not explained due to brevity; we focus completely on the backpropagation architecture which is the major contribution of this chapter.

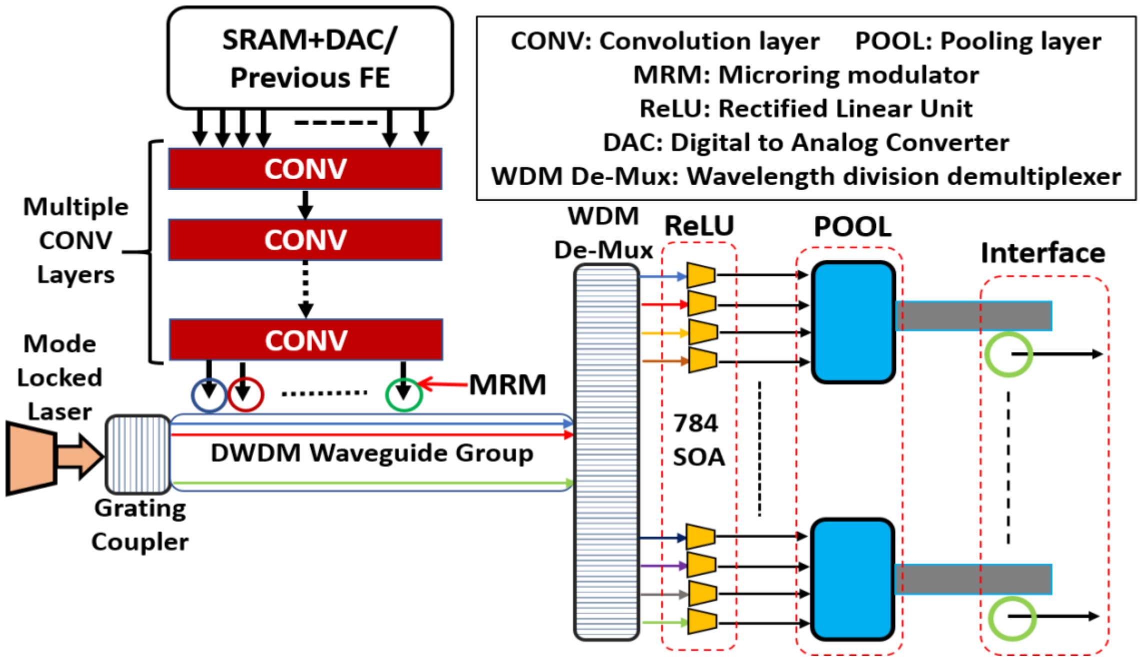


Figure 2.2: Microarchitecture of Feature Extractor (FE) in BPhoton-CNN

2.3.1.2. Backpropagation Architecture

BPhoton-CNN's backpropagation (BP) architecture employs analog microring modulators, photodiodes, multiplexers, and splitters to perform completely analog matrix-multiplication and other arithmetic operations. In contrast, previously proposed CNN accelerators [28], [30] adopt a hybrid approach by using analog memristors for matrix multiplications and digital CPU/GPU for other arithmetic operations, which requires performance hindering A-to-D and D-to-A conversions.

Our analog BP architecture mainly involves computing matrix-vector multiplication in the backward pass. A photonic modulator is used for analog amplitude modulation of a light carrier. In its simplest term, analog amplitude modulation is the multiplication of a scalar input with an analog signal. The authors in [40] have demonstrated photonic modulator-based analog multipliers. Fig. 2.3 illustrates the microarchitecture of the proposed BP accelerator design. It is based on photonic matrix-vector multiplication using microring modulators (MRMs). We use MRMs for their high accuracy and quality factor.

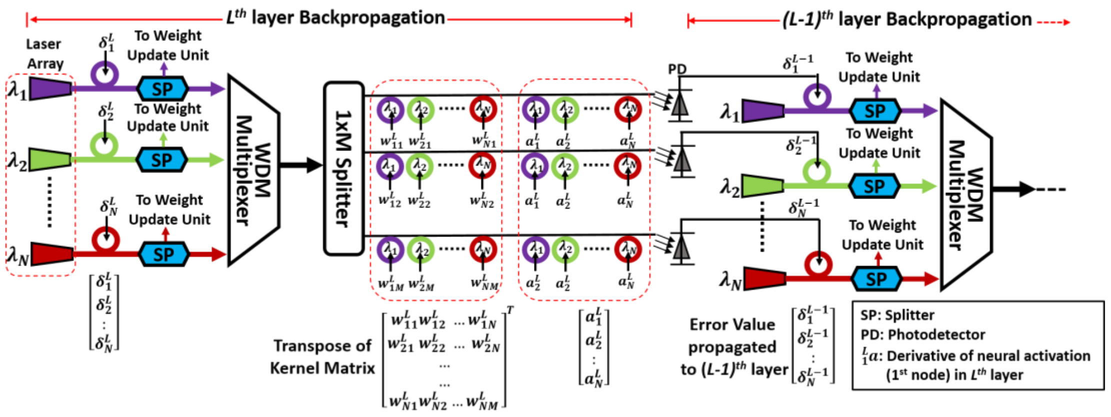

L^{th} layer Backpropagation
$(L\text{-}1)^{th}$ layer Backpropagation
Laser Array
To Weight Update Unit
SP
WDM Multiplexer
1xM Splitter
PD
Transpose of Kernel Matrix
Error Value propagated to $(L\text{-}1)^{th}$ layer
SP: Splitter
PD: Photodetector
$^{L}_{1}a$: Derivative of neural activation (1st node) in L^{th} layer

Figure 2.3: **Backpropagation architecture in *BPhoton-CNN* which presents the backpropagation between the final layer *l*=L and penultimate layer *l*=L-1**

We now describe the operation of the proposed BP architecture. As discussed in Eq. (3), the error at the final layer (l=L) of BP is $\delta^{x,L} \leftarrow \nabla_a C_x \odot \sigma'(z^{x,L})$. Here, $\nabla_a C_x$ is the rate of change of output w.r.t the output activation (i.e., the difference of actual classified output from FC of CNN architecture and the target output). $\sigma'(z^{x,L})$ is the derivative of the ReLU function in the final FC stage of the CNN architecture. Outputs from the final FC stage of the CNN architecture are fed to an analog subtraction and multiplication unit to determine $\delta^{x,L}$. Using Eq. (4) and the computed $\delta^{x,L}$, we calculate the error for the (*L-1)th* layer using the following equation:

$$\delta^{x,L-1} \leftarrow ((w^L)^T \times \delta^{x,L}) \odot \sigma'(z^{x,L-1}) \tag{7}$$

where, w^L is weight matrix obtained from Lth layer of feedforward CNN architecture through the peripheral circuit. The details of the peripheral circuit are explained in the next subsection. Fig.2.3 shows the backpropagation between the final layer *l=L* and its penultimate layer *l=L-1.* As illustrated in Fig.2.3, there is an N number of wavelength carriers coming from a mode-locked laser array. The value of N for a layer equals to the output feature size for the corresponding layer in the CNN architecture, e.g. N equals 49 (7×7) for the last layer. Each wavelength in layer *L* is modulated with error $\delta^{x,L}$ by an MRM tuned to that wavelength. In Fig.2.3, the violet MRM is tuned to modulate λ_1. Now the *jth* MRM's output is $MRM_j = \delta_j^{x,L} * A\sin(\frac{2\pi}{\lambda_j}t + \emptyset)$. Each MRM_j is split into two equal parts. The first part is sent to the weight-update circuitry to update the corresponding weights in the CNN architecture. The other part is fed to a WDM multiplexer. A WDM multiplexer is used to combine multiple light wavelengths into a single multi-wavelength carrier. After multiplexing, the combined optical signal is split into *M* parts where *M* equals the number of neurons in layer *L-1*. Each part is fed to a multi-wavelength waveguide. As a result, in each waveguide there are N wavelengths each carrying data $\delta_{j,n}^{x,L} * B\sin(\frac{2\pi}{\lambda_j}t + \emptyset)$, where $1 \leq n \leq N, B = \frac{A}{2N}$. Each weight w_{ij}^{L} of the transpose of w^{L} obtained from the peripheral circuit is modulated to a light carrier. This results in:

$$M_{i,n} = w_{ij}^{L} * \delta_{j,n}^{x,L} * A\sin(\frac{2\pi}{\lambda_j}t + \emptyset) \tag{8}$$

Now, each $M_{i,n}$ is modulated with a_n^L which is a derivative of the ReLU functions of layer *L-1* (equal to $\sigma'(z^{x,L-1})$ in Eq. (7)). Then, $M_{i,n}$ becomes,

$$M_{i,n} = w_{ij}^{L} * \delta_{j,n}^{x,L} * a_n^L * A\sin(\frac{2\pi}{\lambda_j}t + \emptyset) \tag{9}$$

Next, a photodiode is used to demodulate photonic data from each waveguide. The photodiode demodulates the combined output $M_{i,n}$ for all wavelengths in a waveguide which is nothing but the matrix-vector multiplication identical to Eq. (7). The output of each photodiode is passed through a signal conditioning and filtering circuit to remove unwanted noises. Details of the conditioning circuit are omitted for brevity. The output from the signal conditioning circuit looks as follows:

$$\delta^{x,L-1} = ((w^L)^T \times \delta^{x,L}) \odot a^L \tag{10}$$

where, $\delta^{x,L-1}$ is the error to be propagated from layer *(L-1) to (L-2).* The same procedure as above is continued until the 1st layer is reached. While doing the backpropagation, the error value in each layer is also fed to the corresponding weight-update circuit, which is discussed in more detail below.

2.3.1.3. Weight update and peripheral circuitry

For weight-update, each element of a weight kernel in any layer *l* of CNN architecture can be written as $w_{k,j}^{l}$. Please note that *l*=L for the final layer. Each $w_{k,j}^{l}$ is stored in a memristor of a memristor bank in layer *l* as $G^l_{k,j}$(which is the conductance of a memristor cell). The weight-update equation for $w^l_{k,j}$(or, $G^l_{k,j}$) can be written as per Eq. (5), as follows:

$$G^l_{new(k,j)} \leftarrow G^l_{old(k,j)} - \frac{\eta}{m} \times \delta^l_k \times O^{l-1}_j \tag{11}$$

where, O^{l-1}_j is the *j*th output from the POOL of the *(l-1)* layer of the CNN architecture. Fig.2.4 illustrates the weight-update circuitry for any layer *l*. As shown in Fig.4, δ^l_k is obtained from the BP architecture as a photonic signal. O^{l-1}_j, which is collected from the peripheral circuit, is used to modulate the light carrier carrying the error value δ^l_k. The modulated output is demodulated using a photodiode and then sent to a signal conditioning circuit. In the signal conditioning circuit, first the analog signal is filtered (from noises) and passed through a subtractor to obtain new $G^l_{k,j}$ as depicted in Eq. 9). The previous conductance or weight value $G^l_{old(k,j)}$is fed to the subtractor from the *l*th layer memristor bank. The new conductance value $G^l_{k,j}$ is now fed to the equivalent memristor control circuit to update its weight value. The conditioning circuit as well as the memristor control circuit are inspired from [29].

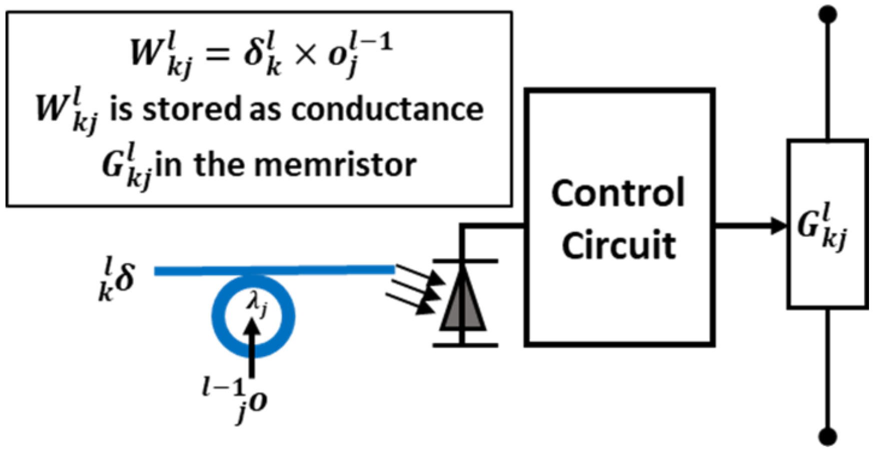

$W^l_{kj} = \delta^l_k \times o^{l-1}_j$
W^l_{kj} is stored as conductance
G^l_{kj}in the memristor
${}^l_k\delta$
λ_j
${}^{l-1}_j o$
Control Circuit
G^l_{kj}

Figure 2.4: **Weight-update circuitry for any layer *l***

The output MP_j from the POOL of a layer l can be written as MP_j^l. During the feedforward training phase, each $MP_j^l$is stored as conductance in a memristor in the peripheral circuitry. This is used in backpropagation as O_j^l, an output of the *l*th layer (as per Eq. (11)). Each MP_j^l is sent to a signal conditioning circuit and then a memristor control circuit. The resulting electronic signal is used to update the conductance (or weight value) of the memristor.

2.4. Case Study

In this section, we demonstrate the working principle of a pipelined *BPhoton-CNN* architecture for a CNN benchmark VGG [38] on the ImageNet dataset [42]. We select a particular configuration, namely, VGG-A for the case study. However, we also experiment with all variants of the VGG [38] and LeNet [39] benchmarks as shown in Table I and discussed in Section V. Using microarchitectures of the convolution layer, ReLU layer, POOL layer, interface layer, and FC layer, we configured *BPhoton-CNN* as illustrated in Fig. 2.5(a) for VGG-A application with four FE stages. The details of it are as follows.

VGG for the ImageNet dataset operates on a 224×224 image input. *BPhoton-CNN* is designed to convolve 56×56 pixels at a time, i.e., one *BPhoton-CNN* cycle. Therefore, it requires 16 *BPhoton-CNN* cycles to execute a 224×224 image. Please note that a *BPhoton-CNN* cycle is different from its clock cycle. Here, one *BPhoton-CNN* cycle refers to the complete feature extraction and feature classification of a 56×56 image. The SRAM register array in *BPhoton-CNN* is of size 2 KB to store the 56×56 input data. CONV performs feature extraction on 28×28 input data at a time in a pipelined manner. FE in *BPhoton-CNN* is performed as explained in the CONV architecture (ref: Fig.2.2).

Fig. 2.5(b) demonstrates the pipelined data flow of the feedforward operation in BPhoton-CNN. We consider a 2.5 GHz clock. Therefore, the clock cycle period$\mathrm{T_{sm}}$ = 400 ps. As shown in Fig. 2.5(b), at t=$\mathrm{T_{sm}}$, the first set of 28×28 pixels from SRAM (i.e., A) are convolved (64 filters/features) and are stored in memristors in the peripheral circuit. The other three set of 28×28 pixels are namely, B, C, and D. Note that CONV convolves a 28×28 input in one clock cycle. As $\mathrm{FE_1}$ for VGG-A consists of one convolution layer (see Table 2.1), convolved outputs of CONV-1 of $\mathrm{FE_1}$ is directly sent to the modulation phase. In the modulation phase, each convolved output is modulated by an MRR of a particular tuning wavelength to a light carrier of that wavelength in the DWDM waveguide group. The DWDM waveguide group can accommodate 784 wavelengths or in other words 4 features of size 28×28. The time required for convolved data of one FE to arrive at the next FE, T_{FE} = modulation time + ReLU time + POOL time + interface time = 20 ps + 10 ps + 10 ps + 10 ps = 50 ps. From t=T_{sm} to t=$2T_{sm}$, CONV(A) outputs from the peripheral circuit of $FE_1$are modulated, ReLU and POOL'ed, and then fed to $\mathrm{FE_2}$. There can be 8 such data movements as $\frac{T_{sm}}{T_{FE}} = 8$. In one data movement, 4 28×28 features can be processed. Therefore, at t=$2T_{sm}$, 32 CONV(A) features arrive at $\mathrm{FE_2}$. Similar to CONV(A), from t=$2T_{sm}$ to t=$3T_{sm}$, 32 CONV(B) features; from t=$3T_{sm}$ to t=$4T_{sm}$, 32 CONV(C) features; from t=$4T_{sm}$ to t=$5T_{sm}$, 32 CONV(D) features are convolved and stored in the peripheral circuit of FE_2. After this, from t=$5T_{sm}$ to t=$6T_{sm}$, the remaining 32 CONV(A) features in FE_1 are convolved in FE_2. In this way, by t=$6T_{sm}$, all the 64 CONV(A) features in $FE_1$are convolved with 128 FE_2 filters to produce 128 features and stored in the memristors of its peripheral circuit.

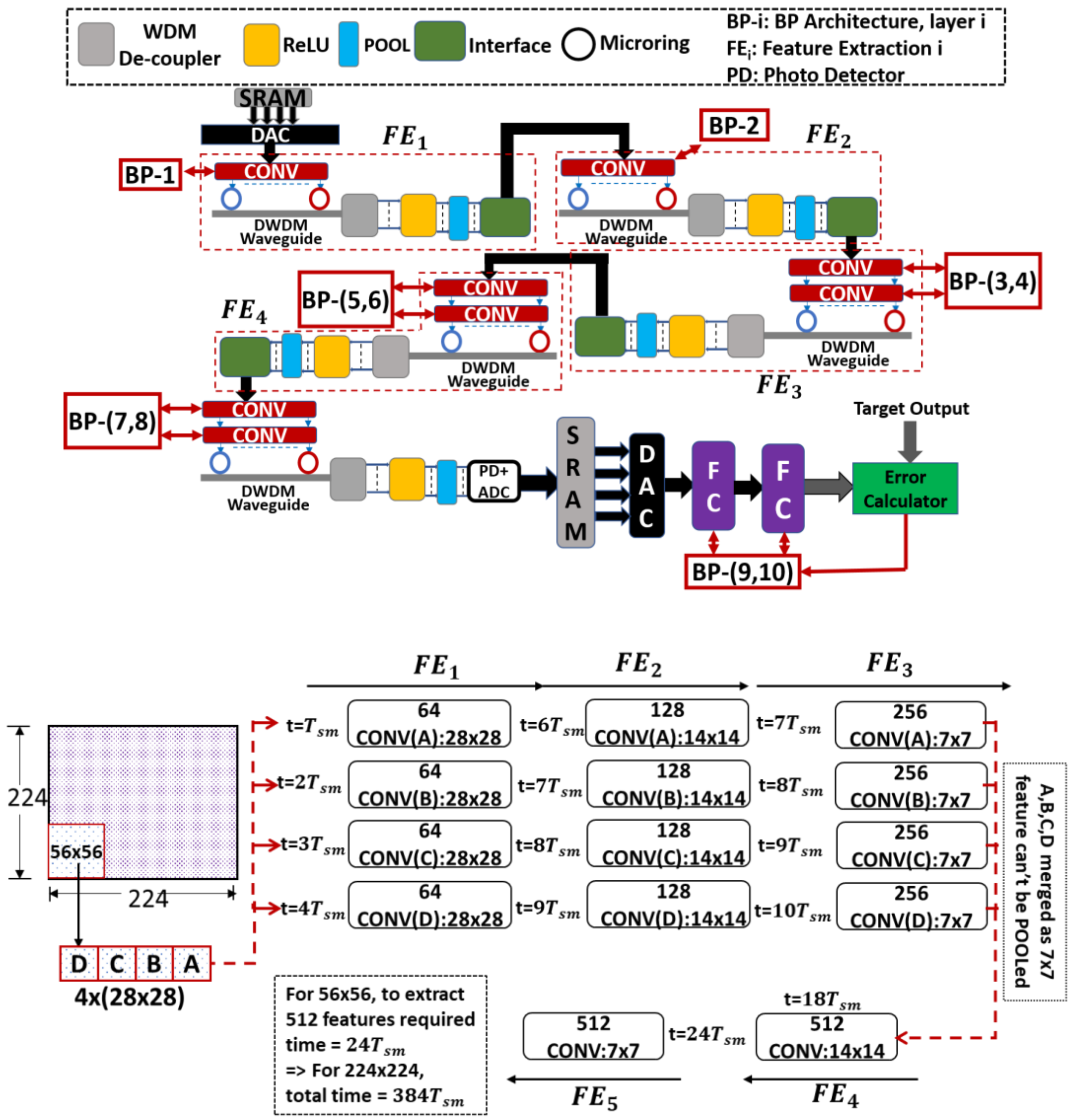


Figure 2.5: **(a) VGG-A implemented on *BPhoton-CNN* (b) Pipelined dataflow in feedforward operation in *BPhoton-CNN*.**

Similarly, remaining 32 B, C, and D features are convolved and stored (Fig. 2.5(b)) by t=7T_{sm}, t=8T_{sm}, and t=9T_{sm} respectively. FE_1 has 64 features, FE_2 has 128 features, FE_3 has 256 features, etc, as per the VGG-A configuration (Table 2.1). It is important to note that 64 CONV(A) features from FE_1 are convolved with 128 memristive WMAs (kernels/filters) to produce 128 CONV(A) features for FE_2. Similarly, 128 CONV(A) features from $FE_2$are convolved with 256 WMAs to produce 256 CONV(A) features for FE_3.

A, B, C, and D are convolved separately until $t = 10T_{sm}$ when all of them arrive at FE_3 as 256 7×7 features each. Now, all of these features are merged together to form 256 28×28 features. Therefore, it will require another $8T_{sm}$ time (i.e., t=$10T_{sm}$ *to* t=$18T_{sm}$) to send 256 28×28 features from FE_3 and convolve them as 512 14×14 features at FE_4. Similarly, convolution, ReLU, and POOL are performed in FE_4 and FE_5. As illustrated in Fig. 2.5(b), at t=$24T_{sm}$, 512 features are obtained from FE_5 for 56×56 pixels. As shown in Fig. 2.5(a), features from FE_5 are stored in SRAM until all the 224×224 pixels are extracted. For 224×224 pixels, it will take 16×24T_{sm}=384T_{sm}=153.6ns. After this, all the features are retrieved from SRAM and fed to FC for feature classification. The first FC operation requires $(T_{sm} + T)$ time as it is identical to FE.

The second FC operation requires T time as no more SRAM read is needed. This means that *BPhoton-CNN* requires 153.6 ns (for FE) +$T_{sm} + 2T$ = 154 ns, for one forward pass. After a forward pass, the FC output is sent to the BP architecture for backpropagation. Each layer in BP requires T_b units of time where T_b = (error modulation to light carrier) + (split time) + (WDM multiplexing time) + (split time) + (weight modulation time) + (ReLU function derivative modulation time) + (photodiode time) = 10 ps + 10 ps + 10 ps + 10 ps + 10 ps + 10 ps +20 ps = 80 ps. It takes $6T_b$ units of time to complete one backward pass.

	FE1	FE2	FE3	FE4	FE5	
VGG-A	**3×3, 64, 1**	**3×3, 128, 1**	**3×3, 256, 2**	**3×3, 512, 2**	**3×3, 512, 2**	**FC-4096,2 FC-1000, 1**
VGG-B	**3×3, 64, 2**	**3×3, 128, 2**	**3×3, 256, 2 1×1, 256, 1**	**3×3, 512, 2 1×1, 256, 1**	**3×3, 512, 2 1×1, 256, 1**	
VGG-C	**3×3, 64, 2**	**3×3, 128, 2**	**3×3, 256, 3**	**3×3, 512, 3**	**3×3, 512, 3**	
VGG-D	**3×3, 64, 2**	**3×3, 128, 2**	**3×3, 256, 4**	**3×3, 512, 4**	**3×3, 512, 4**	
LeNET-A	**3×3, 6,1**	**3×3, 6,1**	**3×3, 16,2**	**3×3, 16, 4**	**3×3, 120, 1**	**FC84,1**
LeNET-B	**3×3, 6,1**	**3×3, 6,1**	**3×3, 256, 1**	**3×3, 16,6**	**3×3, 120, 1**	

Table 2.1: CNN Benchmark Configuration For VGG, LeNeT

In summary, *BPhoton-CNN* requires 154 ns for one forward pass and 80 ps for a backward pass. The ultra-fast nature of photonic interconnects allows for high-speed backpropagation in *BPhoton-CNN.*

2.5. Experimental Analysis

2.5.1. CAD for Bphoton-CNN

We use IPKISS [42], a commercial optoelectronic CAD tool, to design and synthesize all of the photonic components of *BPhoton-CNN*. All of the synthesized components are integrated together to design *BPhoton-CNN*. For all of the photonics components, we consider a 32nm IPKISS library. The parametric details for *BPhoton-CNN* are obtained from [30]. We developed a C++ based architectural simulator which takes device- and link-level parameters from IPKISS, to estimate performance of *BPhoton-CNN* accelerator for several benchmarks.

2.5.1.1. Power, Area, and Performance Models

We use Caphe [42] for modeling power and area of all photonic elements such as modulators, demodulators, waveguides, lasers, etc. The energy and area parameters for memristors are adapted from [30]. We use integration and fire mechanism-based DAC identical to PipeLayer [30] in our design. The power and area models are adapted accordingly from PipeLayer. We also use power and area parameters from [29] for the ADC array used in the FC layer of *BPhoton-CNN*. We use Caffe [43], a deep learning framework, to train the datasets in conjunction with photonic component results from IPKISS. We manually map each of our benchmarks in waveguides, max-pool, buffers, and FC of *BPhoton-CNN*. This ensures zero pipeline hazards between any two layers in *BPhoton-CNN*. We compare the performance of *BPhoton-CNN* with a state-of-the-art CNN accelerator, namely PipeLayer [40]. We evaluate for the following metrics: *Computational efficiency* represents the total number of fixed point operations performed per unit area in one second (GOPS/s/mm^2); *Energy efficiency* refers to the number of fixed point operations performed per watt (Giga operations per watt or GOPS/s/W); *Throughput* is the total number of operations per unit time (GOPS/s); and lastly, *Prediction error rate* is the percentage of error in inferring any datasets

2.5.1.2. Benchmark and dataset

We use two widely used CNN benchmarks: VGG-Net [38] and LeNet [41]. We consider four variants of the VGG benchmark: VGG-A, VGG-B, VGG-C, and VGG-D and two variations of LeNet (LeNet-A and LeNet-B). The configuration of all stages of VGG-Net and LeNet benchmarks identical to [30]. For VGG, we use ImageNet dataset [41] having

224×224 images. For LeNet, we use 60,000 224×224 images of MNIST datasets [44] for training and 10,000 224×224 images for testing.

2.5.2. Performance Analysis:

Fig.2.6 demonstrates speedup (throughput) of *BPhoton-CNN* and PipeLayer [30] compared to the baseline GPU implementation results, also from [30], for four variations of the VGG and two variants of the LeNet benchmarks. The GPU-based accelerator performs with an average throughput of 310 GOPS/s. PipeLayer shows an average throughput of 87000 GOPS/s. The proposed *BPhoton-CNN* shows an average throughput of 2784000 GOPS/s. The superior performance of *BPhoton-CNN* is due to the intelligent integration of ultra-fast memristors and high-speed photonic components such as MRAs, SOAs, and comparators. The overall throughput of PipeLayer is affected by inter-layer data conversion with relatively slow ADCs. Also, PipeLayer spends most of its time in sequential weight updates during training. However, *BPhoton-CNN* has an inherent advantage due to its photonic parallel weight update mechanism. On average, *BPhoton-CNN* outperforms PipeLayer and GPU by 35× and 345× in terms of speedup, respectively. Finally, for the results presented in Fig. 2.6, the variance of speedup across benchmarks is 1650 with a standard deviation of 40.02.

Fig.2.7 illustrates the effects of weight resolution on overall speedup of *BPhoton-CNN*. With the rise in weight resolution, there is a very little degradation in speedup (5% lower for 32-bit compared to 16-bit). This is due to the additional delay in storing 32-bit data in SRAM compared to 16 or 8-bit data. However, data conversion is done either at the beginning or at the end of the forward pass in *BPhoton-CNN*. Therefore, the effect is very minimal. Furthermore, it can also be noted from Fig.2.7 that the speedup has a slightly decreasing trend from VGG-A to VGG-D. This is due to the increase in total number of convolution layers from VGG-A to VGG-D.

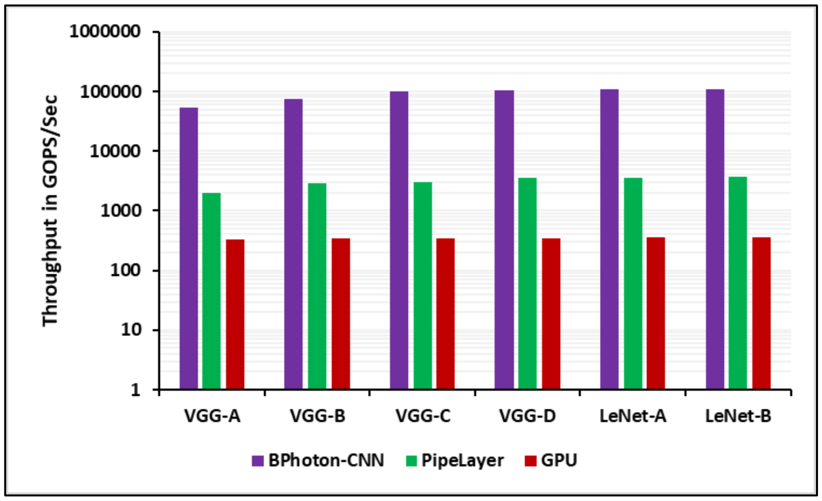


Figure 2.6: **Speedup (throughput) comparison across accelerators**

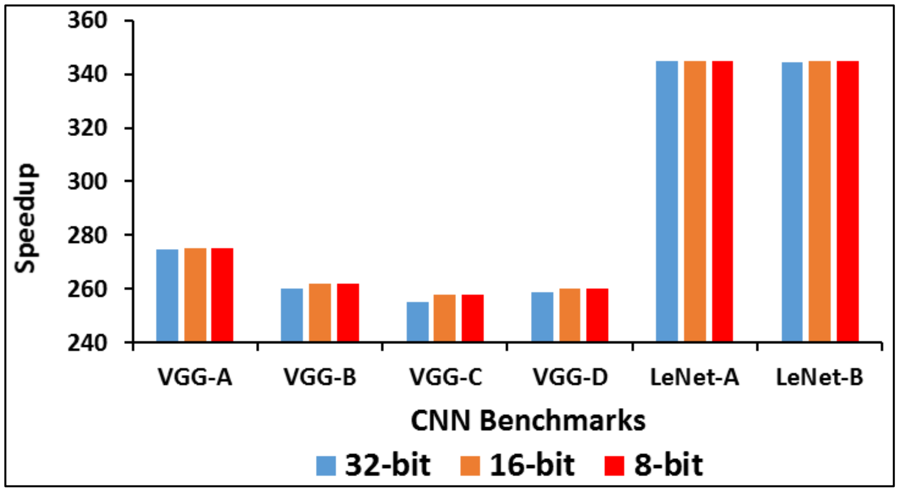


Figure 2.7: **Speedup of *BPhoton-CNN* w.r.t. weight resolution.**

Fig.2.8 illustrates the normalized computational efficiency (CE) (i.e., the total number of fixed-point operations performed per unit area in one second (GOPS/s/mm^2)) comparison of the proposed *BPhoton-CNN* and memristor crossbar based PipeLayer [30] with respect to a baseline GPU based design**.** First of all, the proposed *BPhoton-CNN* architecture shows a computational efficiency variance of 302 (17.38 GOPS/s/mm^2) which is reasonable considering its high computational efficiency. Furthermore, PipeLayer uses memristor crossbars for the bulk of its arithmetic operations. Each memristor crossbar has a CE of 1707 GOPS. However, the overall CE of PipeLayer comes down to 1485 GOPS due to its extensive usage of data conversions. Also, ReLU and POOL are performed by a digital ALU in PipeLayer. This requires more memory to store intra-layer data for synchronizing with its pipeline mechanism. The superiority of *BPhoton-CNN* comes from the fact that it is a completely analog accelerator. Therefore, *BPhoton-CNN* does not involve inter-layer data conversions or storage for synchronization. AD and DA conversions are done either at the beginning or at the end of feature extraction in *BPhoton-CNN*. In addition to the compute efficient memristor, *BPhoton-CNN* also uses high speed SOA as ReLU which has a CE in the order of 50000 GOPS/s/mm^2 [29]. As shown in Fig.7, *BPhoton-CNN* has 31× and 320× higher computational efficiency compared to PipeLayer and GPU, respectively.

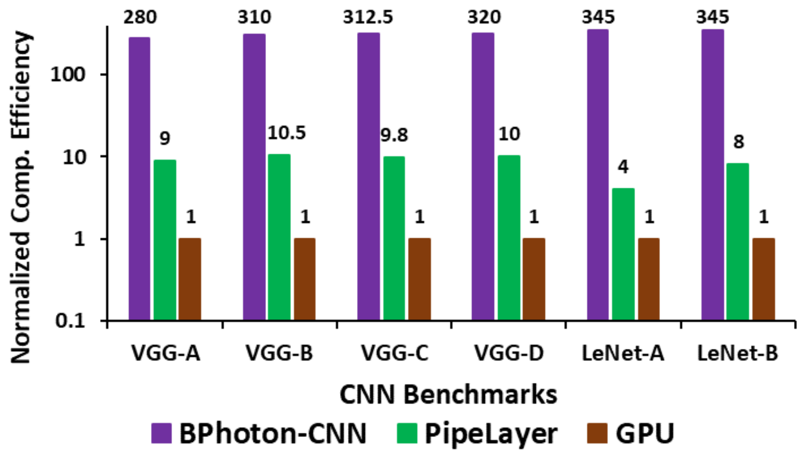


Figure 2.8: **Normalized computational efficiency across accelerators.**

2.5.3. Energy Savings

We compare the energy efficiency of BPhoton-CNN with PipeLayer and GPU as shown in Fig.2.8. The average energy efficiency for PipeLayer is 142.9 GOPS/s/W which is 7.17× higher than GPU based accelerator. BPhoton-CNN works with an average energy efficiency of 6432 GOPS/s/W. PipeLayer replicates its early feature extraction layers several times (close to 50K times) to maintain a balanced pipeline. This involves excessive use of high-power consuming data conversions. BPhoton-CNN uses passive optical components such as waveguides and comparators, in addition to energy-efficient components such as ring modulators/demodulators, SOAs, and memristor. Also, BPhoton-CNN uses very few ADCs/DACs compared to PipeLayer. As shown in Fig.8, we obtain 45× and 360× improvements in energy efficiency for BPhoton-CNN compared to PipeLayer and GPU, respectively. Overall, the variance of energy efficiency of BPhoton-CNN across benchmarks is 6.96 with a standard deviation of 2.63.

2.6. Conclusions

This work demonstrates a fully analog CNN accelerator called *BPhoton-CNN* that integrates compute-efficient memristors and ultra-fast photonic components. BPhoton-CNN comprises a completely analog photonic backpropagation architecture. Further, the proposed architecture deploys (i) a reconfigurable convolution design in each CNN layer to emulate a range of sample CNN models; (ii) a novel approach for analog signed-weight arithmetic in the memristive convolution layers. Compared to PipeLayer [30] and GPU, *BPhoton-CNN* architecture shows higher computational and energy efficiency due to the use of energy-efficient SOAs, optical comparators, and due to its use of a fully analog feature extraction method. We demonstrated that the proposed design has the potential to achieve up to 35× acceleration in training in addition to 31× improvement in computational efficiency and 45× energy saving compared to the state-of-the-art with similar accuracy. Our future work will address the issue of the broader applicability of our accelerator to other types of deep learning models, e.g., deep neural networks (DNNs).

3. ISSUE WITH MEMRISTOR

3.1. Non-ideal characteristics of a memristor

Despite showing the characteristics of a highly promising device for in-situ computing, memristor does have a few drawbacks. As memristors are non-linear in nature, they have a few reliabilities issue that needs attention before memristors can be made industry-ready.

The complexity of deep learning models is increasing to cater to the need for complex problems and to satisfy the high accuracy demand. A larger and deeper neural net will require a large memristor crossbar array to represent it. This increases the complexity of designing the crossbar. Also, large crossbar arrays have a higher chance of suffering from leakage current and other non-ideality parameters, which results in erroneous conductance values. As the dot-product computation is performed at each memristor cell, this error propagates through the crossbar array, increasing manifold in the case of a large network.

This error reduces the accuracy significantly and makes memristor-based neuromorphic computing unsuitable for applications in the field of medical diagnostic or autonomous driving, where a wrong prediction incurs a heavy loss.

Memristor cell faces modeling as well as reliability issues. Modeling a memristor is challenging because of its non-ideal characteristics [24,25]. Most of the modeling has been done in the simulator and hence it becomes important that the model depicts the behavior of the device correctly [24]. Some major reliability issues faced by the memristor devices are due to the non-ideal characteristics described in section 3.1.1 to 3.1.6.

3.1.1. Aging

Aging is an inevitable process that reflects the performance degradation of a device with time. Memristor stores values in the form of conductance states. Aging in memristor occurs due to continuous switching of the conductance value. While using memristor crossbar for deep learning applications, weight values are mapped linearly to the memristor’s conductance values [52]. During training, weight values are updated continuously. This requires continuous rewriting of the conductance values by applying appropriate pulse. With aging, the ability of memristors to hold the expected conductance values decreases as the and this affects the performance of deep learning accelerators. This occurs because the value of R_{off} decreases with time or as frequency of switching increases. As it is an irreversible process and contributes the most in deep learning performance degradation, it has been studied extensively in this work. Aging has been discussed in detail in 3.2.

3.1.2. Device variability

Memristors are arranged in a crossbar fashion to mimic the behavior of a neural network layer. Ideally, each memristor in the crossbar is expected to behave identically but the fabrication process introduces variability between devices. The doping concentration, temperature, and other physical parameters play a vital role in causing the variance. This variability affects the conductance and memresistance of the device which causes the memristor to produce a non-ideal output. Also, it has been found that the variability of R_{off} is higher than that of R_{on} [47,53]. This can cause the neural network to predict incorrect outputs which can adversely affect the deep learning application running in real-time. The device-to-device variability is the inherent nature of device manufacturing. Though fabrication methodologies have been improved [48,49] to minimize the variation, they cannot be eliminated completely. Much work has been done to simulate the memristor considering the device-to-device variability [26]. Novel techniques have been introduced to minimize the change in the output due to variability [48,49].

3.1.3. Non-linear device characteristics

Most of the memristor devices have been simulated taking into consideration the ideal linear I-V characteristics. However, non-ideal memristors have non-linear I-V device characteristics, especially at high voltage [26]. Modeling such non-ideal characteristics accurately and efficiently becomes challenging. A non-linear I/V characteristics graph for VTEAM [46] modeled using MemTorch [26] is shown in fig.3.1.

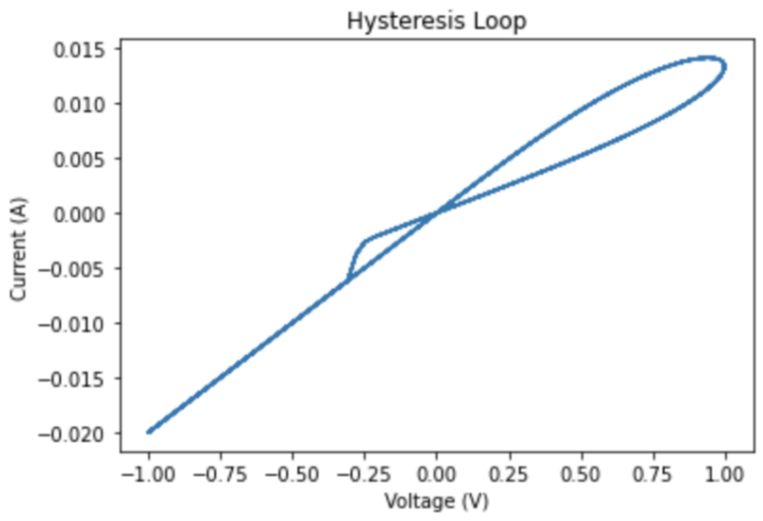


Figure 3.1: Non-linear I-V characteristics of VTEAM model

[26] determines the I/V characteristics of each device and stores them in a Lookup Tables [LUT] and uses this to calculate the output current during inference.

3.1.4. A finite number of conductance state

Memristor devices are known to have multiple levels of conductance states. These conductance states can be switched by applying an appropriate pulse to the device for a certain duration and amplitude and hence can be used to store data in the device [45]. The levels are bounded by R_{off}, the maximum resistance state during the off state, and R_{on}, the lowest resistance state [50]. Between these bounds, the conductance levels are defined. But as discussed in 3.1.1 and 3.1.2, device-device variability and aging could cause these resistances to change. This would result in non-uniform distribution of conductance state between devices. As devices age from their pristine state, this non-linearity becomes more evident. This leads to degradation of the crossbar which would adversely affect the accuracy of the model. Deterministic discretization of these conductance states is one way to overcome this condition [51]. Since discretization provides more error tolerance, it significantly improves accuracy. But inevitable factors like aging continue to degrade the crossbar and the above methods would eventually succumb to failure.

3.1.5. Device Failure

As discussed in 3.1.1., fabrication plays an important role in deciding the characteristics of a memristor device. Due to fabrication issues, and effect of external factors like temperature and increased frequency of switching voltage/pulse, the memristor device becomes susceptible to failure [45]. Device gets stuck either at low resistance state (LRS) R_{on} or at high resistance state (HRS) R_{off} or fail to electroform at pristine stage. The memristor devices stuck at a particular resistance cannot be reprogrammed and it results in loss of accuracy during a neural network implementation. In the next section we will discuss how aging of memristor crossbar affects performance of deep learning accelerators. We will also briefly describe the simulator MemTorch [26] that is used in this work.

3.2. Aging is a devil!

As discussed in the previous section, aging is a non-ideal characteristic of memristor which can significantly degrade the performance of the device over time. As memristors are nanoscale devices, fabrication process is not easy, and they show large process variation. This results in large variation in device parameters mainly, R_{off} and R_{on}. As described in 3.1.1, the conductance values of a memristor cell need to be tuned or programmed frequently to store the corresponding weight value and a high voltage is applied across the memristor cell to do this tuning. Frequent high voltage would mean high current across the memristor filament, and this changes the internal structure of the cell by increasing temperature in the filament region [45]. The constant switching reduces the range of conductance values and thus the number of conductance states. So, even if we want to map a trained weight to a desired conductance state, it will map to a conductance value different from than desired conductance. This occurs because of the decrease in maximum resistance R_{off} of the memristor cell as shown in Fig.3.2, and that causes memristors to lose their conductance levels with time. Thus, a memristor that can be programmed to level 5 when in the pristine state can only be programmed to level 3 after time t. Even after repeated try to reprogram the device, the value will not go above the aged R value. This results in wrong weight values stored in the memristor crossbar and thus generate error. As, this error propagates through the whole crossbar during computation, the accuracy is affected.

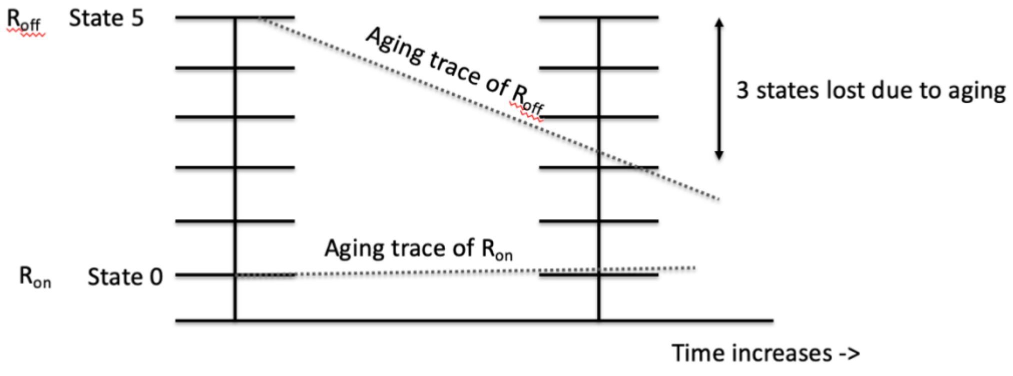


Figure 3.2: Effect of aging on R_{off}. Adapted from [52]

Experiments done by [53] concludes that the maximum value of resistance (R_{off}) ages faster than that of minimum resistance (R_{on}). We leverage this change of R_{off} as the devices ages and model an aging function. Incorporating this function in an ideal memristive deep neural network (MDNN) shows how accuracy of a deep learning model is affected by aging of a memristor device. In the next sections, we will discuss the proposed method to model aging in a MDNN.

3.2.1. Modelling Aging

Since aging plays a vital role in degrading memristor performance over time, this work tries to capture the effect of aging on a DL model. MemTorch[26] was used as a simulator to incorporate the aging function in an ideal MDNN. The basic idea behind aging is that continuously rewriting conductance values changes the internal structure of the device and thus it is not able to store the ideal R_{off} value i.e. the maximum resistance decreases as the device ages.

To model aging, first a Deep Neural Network (DNN) was defined using PyTorch. Then weights of the neural network were linearly scaled into conductance values of memristor crossbar array using the following equation:

$$g_{i,j} = \frac{(W_{i,j} - W_{min}) * (g_{max} - g_{min})}{(W_{max} - W_{min})} + g_{min} \tag{12}$$

Where, W_{min} and W_{max} is minimum and maximum weight values of a layer in the neural network. g_{max} is $1/r_{on}$ or maximum conductance and g_{min} is $1/r_{off}$ or minimum conductance value of the memristor device used.

The generated conductance values represent the weights of the MDNN equivalent model. The model is then tuned to account for the changed weight values during linear scaling by using linear regression on the generated output and the desired output for a randomly produced input. The resulting model is an ideal MDNN.

Non-ideality aging was modeled in the ideal MDNN by using the proposed function. The aging function defined in this work assumes that the R_{off} values of a memristor decrease with time and this causes the number of conductance levels to drop. Once the number of levels decreases, the percentage of devices which cannot be programmed beyond R_{off} value increases. Input to the aging function is the value of maximum resistance as it ages and the percentage of devices that will age for that resistance value. As all memristor device performs differently in a real scenario, we cannot assume that all the devices will age at the same time and in the same manner. So, the user can define the rate of decrease of maximum resistance as the device ages and the percentage of devices that will age. Fig.3.3. shows the algorithm for the proposed aging function. The conductance matrix of the ideal MDNN was changed to incorporate the updated resistance values due to aging. As not all devices age in a similar manner, the devices that will age were selected randomly from the crossbar. The resultant is an MDNN model with aging.

```
func aging($R_{age}$, device_aged,
conductance_matrix):
    g_min = 1/$R_{age}$
    length = len(conductance_matrix)
    num_select = length * device_aged
    idx = random (1, length, num_select)
    for i in idx:
        conductance_matrix[i] = g_min
    return conductance_matrix
```

Figure 3.3: Algorithm for aging function

3.2.2. Experimental Analysis

MemTorch[26] was used as a simulator for this work. MemTorch is an open-source simulation framework for memristors and is based on a well-known PyTorch library. The integration of MemTorch with PyTorch helps in the simulation of complex ML and DL models. MemTorch is programmed in C++ and python, and it inherently supports the use of GPU using CUDA which enhances the computation speed of deep neural networks.

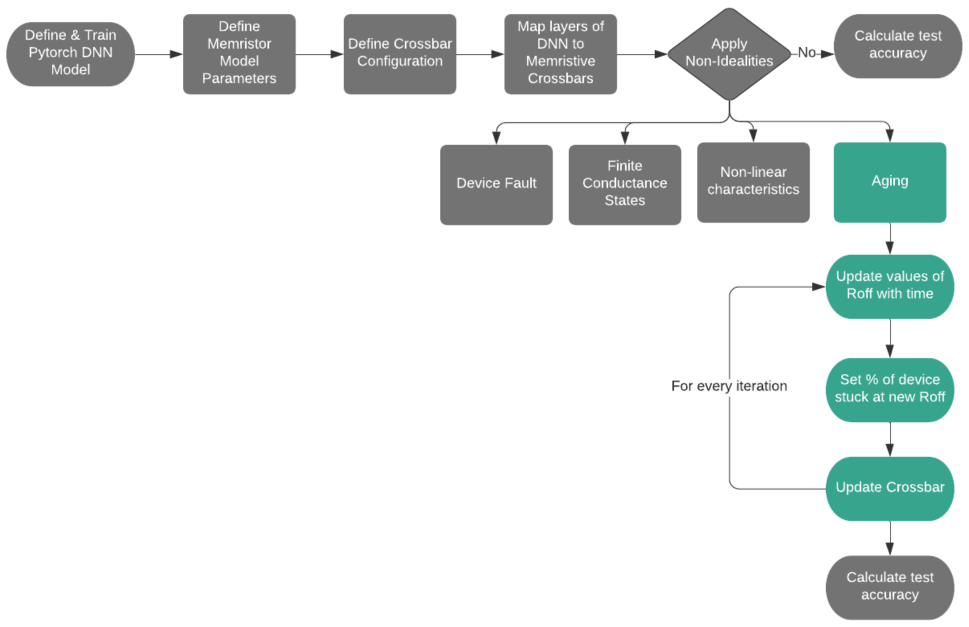
First, a Deep neural network (DNN) with two conv2d layers and two Linear layers is defined using PyTorch. The model was trained on the MNIST dataset [44]. A training accuracy of 99.25% and a testing accuracy of 99% was achieved for the DNN model. Then the memristor model to be considered for the experiment was defined. This model is used as a base for generating the crossbar arrays. Currently, MemTorch supports 4 different models for Memristor. These models are based on VTEAM [46], Stanford PKU RRAM model [54], linear ion drift model [20], and data-driven Verilog-A RRAM model [55]. VTEAM or Voltage Threshold Adaptive Memristor model is used in this work as it has a threshold voltage of 0.2V and a value less than that will not make any changes in the conductance values of the device. The parameters used for the VTEAM model are defined in table 3.1.

Model	VTEAM
Device Length	3nm
$V_{threshold}$	0.2 V
Mapping Routine	Linear
Column	Double
ADC Resolution	8

Table 3.1 : Memristor model parameters

A test accuracy of 98.46% was obtained when the test dataset was executed on the ideal MDNN model. MemTorch is also capable of modeling non-ideal characteristics discussed in section 3.1. Though Device variability, nonlinear device characteristics, device failure, and the number of conductance states have been modeled by the

MemTorch, the aging is not considered while modeling. This work takes advantage of the current MemTorch framework and extends it by modeling the aging of memristors. A complete flow chart of MemTorch including the proposed aging function is described in fig.3.3.



Define & Train Pytorch DNN Model
Define Memristor Model Parameters
Define Crossbar Configuration
Map layers of DNN to Memristive Crossbars
Apply Non-Idealities
No
Calculate test accuracy
Device Fault
Finite Conductance States
Non-linear characteristics
Aging
Update values of Roff with time
For every iteration
Set % of device stuck at new Roff
Update Crossbar
Calculate test accuracy

Figure 3.4: Flowchart describing MemTorch along with proposed Aging function

The MDNN model with the incorporated aging function was tested on the test dataset for different values of R_{aged} and different percentages of aging. Two cases were considered.

Case 1: R_{off} was aged exponentially and for every value of R_{aged}, the aging percentage was increased linearly. It was observed that as more and more device ages the accuracy decreases significantly as shown in Fig.3.5. It was also observed that the decrease in accuracy as aging increases is steeper when the R-value is lower. For example, the accuracy with R value set at 1.95E5 ohms decreased to around 50% when only 20% of devices were aged whereas the accuracy remained close to 98% when R was 5E7 ohms. The value of R_{on} or minimum resistance was kept the same for all the cases. Thus, we can conclude that when the difference between maximum and minimum resistance is less, the effect of aging is more profound. This can be explained as the number of conductance states of a memristor device is bounded by R_{off} and R_{min}. If the difference between R_{off} and R_{min} is small, we will have a smaller number of conductance states. And as aging affects the R_{off} value, an aged device will have even fewer conductance states and thus accuracy is affected adversely.

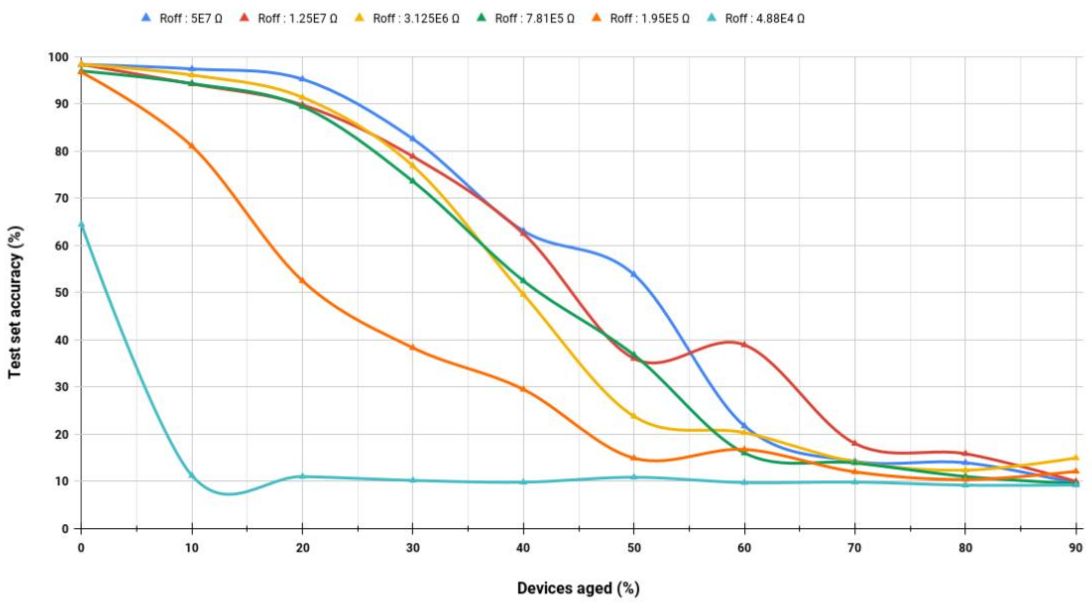


Figure 3.5:Test accuracy vs percentage of device stuck at R_off for different R_off

Case 2: R_{off} was aged linearly and as the value of R decreases, the aging percentage was increased. This represents a more realistic simulation as when a device ages, the value of R decreases and with time more and more devices will age. As R ages, the aging percentage was selected randomly between 8-12%. It was kept lower to observe a gradual degradation in performance with aging. The crossbar array was then updated to reflect this aged behavior and was tested on test dataset. This was then repeated till the percentage of aging reaches 100%. Fig.3.6. shows the graph obtained for this experiment. The graph shows that as device ages, the accuracy decreases. This is because, aging changes the value of R_{off} and thus decrease the number of conductance states. This affects the weights stored in the crossbar as conductance as they cannot reach the desired value. This generated error which degrades the performance of MDNN.

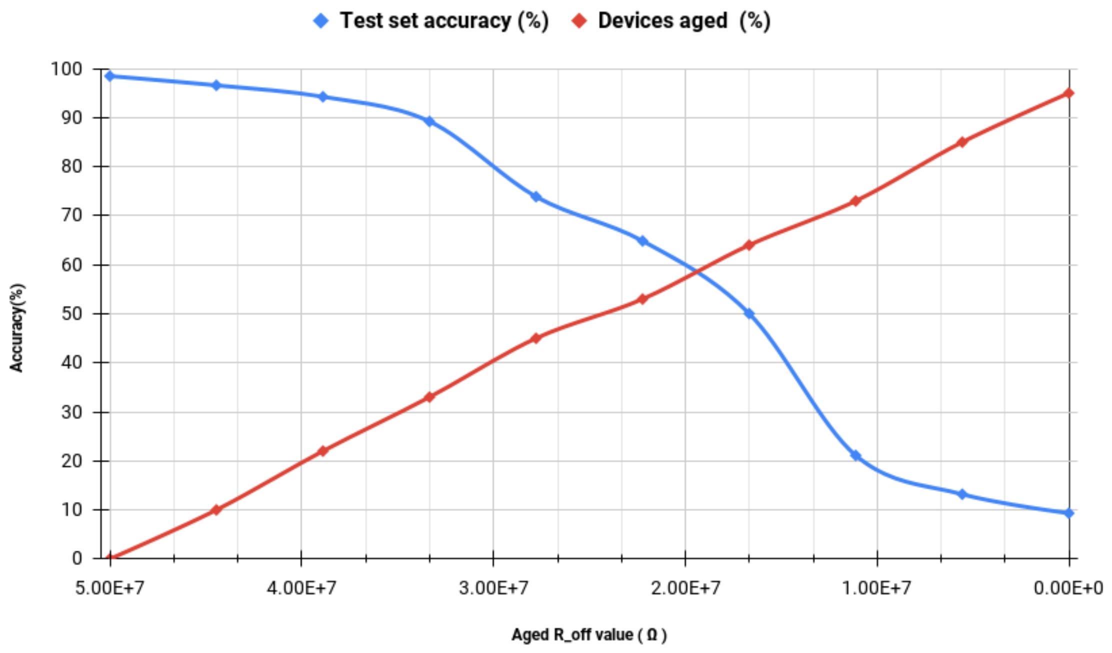


Figure 3.6: Test accuracy VS aging

The above results prove that accuracy of a deep neural network is significantly hampered as the memristor ages. Although various works have been done to counter aging, such an extensive study on the effect of aging on accuracy is limited. Now that we have established that aging of memristor crossbar plays a significant role in the development of memristor based neuromorphic computing for deep learning models, we will see some approaches to counter aging.

3.2.3. Existing approaches to counter aging

The effect of aging has been studied extensively as it is a major factor affecting the reliability of memristors. As aging decreases the number of conductance states of the memristor device, [57] uses binary weighted memristive devices which have only two states 0 and 1. Some also use an extra row of memristors which can be used to substitute the row that is aged [58]. The addition of these redundant rows increases the lifetime of the memristor crossbar but incurs extra hardware costs. Aging occurs because of frequent rewriting of conductance values by applying high voltage. High voltage results in high current across the filament and thus changes the internal structure of the device. If the current through the filament is decreased, then aging also decreases. This technique has been adopted by various researchers to reduce the effect of aging. [56] uses sinusoidal pulses as programming voltage instead of DC voltage as the average is reduced. We will also use this technique but from software perspective. In the next section we will discuss about the proposed method to counter aging.

4. AGING AWARE MEMRISTOR CROSSBAR

In this section we will discuss how to counter aging to develop an aging aware memristor crossbar array for reliable deep learning application. As discussed in previous section, aging occurs due to changes made in the memristor filament because of current flowing through it. High current increases the temperature across the device and over time this degrades performance or memristor cell. So, if the current across the filament decrease, then process of aging can be slowed down.

Various work has been done to reduce the current flow and thus reduce aging. [52] proposed one such method called skewed weight training. Skewed weight training is done by concentrating the weight values to a smaller value during training. Training is done using software, and then weights are linearly mapped into the conductance of memristor crossbar. So, if we reduce the value of weight, it will result in lower conductance values and high resistance values. High resistance will mean that less current will flow through the memristor cell and thus slows down the advance of aging. In our proposed work, we will incorporate skewed weight training to generate a DNN model which will be converted into a MDNN using the simulator memristor. Then effect of aging on a skewed weight trained memristive model will be studies using our propose aging function.

4.1. Skewed weight trained Deep Neural Network

Skewed weight training means concentrating the weights to a smaller region while training the model. Fig.4.1. shows the weight distribution during weight initialization. Skewed weight training will reduce the variance of the weight distribution graph.

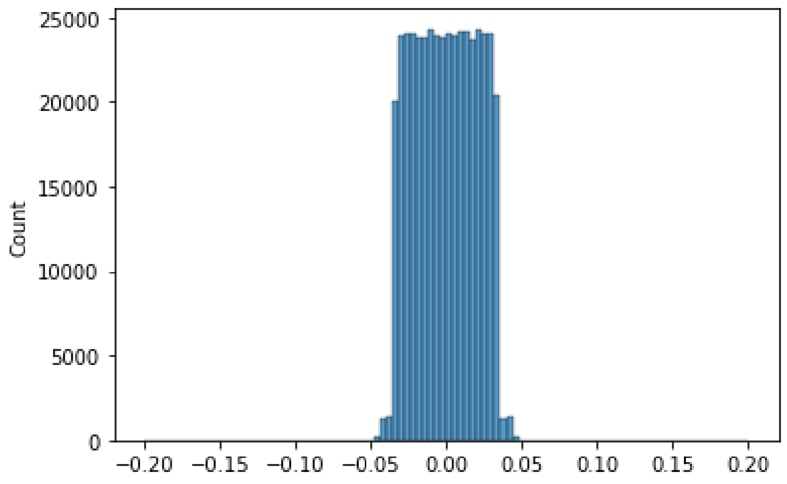


Figure 4.1: Weight distribution during initialization

Before understanding how skewed weight training is done, it is important to understand how neural network update weight values during training.

4.1.1. Neural Network training

Training a fully connected neural network consist of forward and backward propagation. In forward propagation, the provided input is multiplied with weight values and intermediate variables are calculated in forward direction that is from input layer to output layer and finally output is generated. First step is to multiply weight vector(W^1) with given input vector (X).

$$z = W^1 X \tag{13}$$

Second step is to pass the generated output through an activation function $\emptyset$.

$$h = \emptyset(Z) \tag{14}$$

The generated output serves as input for the next layer where both steps are repeated till output layer is reached.

In backward propagation, weight values are reassigned moving backward from output layer to input layer. The generated output(z_i) after forward propagation is compared with the expected output(y_i) to calculate loss using loss function.

$$L = \text{cost}\ (z_i, y_i) \tag{15}$$

Then gradient of weight function with respect to weight function is calculated for each weight value in a layer W_{ij}^{k} where k is the layer.

$$gradient = \delta L/\delta W_{ij}^{k} \tag{16}$$

The gradient is subtracted from the weight value at that layer to get the new weight value. This gradient is backpropagated till the input layer and all weights are updated.

4.1.2. Skewed weight training

An extra term is added to the loss calculated in eq.15 which increases the value of gradient and thus the weight value becomes small. The updated loss can be represented as:

$$L = \text{cost}\ (z_i, y_i) + \text{skew}(W_{ij}^{k},\ W^{k\prime}) \tag{17}$$

where W_{ij}^{k} is weight value for kth layer and $W^{k\prime}$ is the reference weight for kth layer around which the weights are skewed.

The reference weight is selected in the range of weights of the model and the weight distributions are skewed around this reference weight. Original weight values that lie in the left and right side of the reference weight are penalized. This can be done as represented in equation:

$$skew\left(W_{ij}^{k},\, W^{k'}\right) = \begin{cases} \sum_{k=1}^{no.\,of\ layers} \lambda_1 \,.\ \left||W^k - W^{k'}|\right|^2 & \text{when } W^k < W^{k'} \\ \sum_{k=1}^{no.\,of\ layers} \lambda_2 \,.\ \left||W^k - W^{k'}|\right|^2 & \text{when } W^k > W^{k'} \end{cases} \tag{18}$$

Where, λ_1 ,λ_2 are the penalty factors for the weights on the left and right side of reference weight respectively.

This updated loss function is then used to calculate the weight values to make the weight distribution graph skewed around the reference weight.

4.1.3. Experimental setup and Results

The PyTorch model defined in section 3.2.2. is used for this experiment. In the model, we used ADAM optimizer[60] to calculate the gradient. To incorporate skewed weight learning during training, we designed a revised version of the Adam algorithm, where the above loss function was implemented. The reference weight was selected to be a factor of the standard deviation of the original initialized weight for each layer i.e.

$$W^{k'} = \sigma_{\mathrm{I}} * \text{delta\_scale}. \tag{19}$$

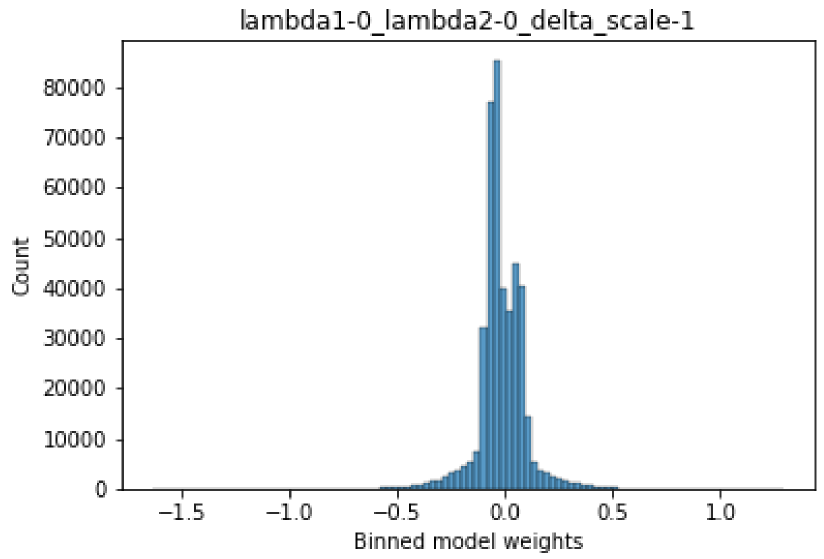
where delta_scale represents the factor with which standard deviation was multiplied to obtain the reference weight.

Different values of $\lambda_{1\,,}\, \lambda_1$ and delta_scale was considered to find the best-suited value with no performance degradation with training a neural network with skewed weight. Table 4.1. represents different values considered for the experiment.

Fig.4.2(a) represents the weight distribution graph of the original PyTorch model without skewed weight training and fig.4.2(b) shows the weight distribution graph after skewed weight training.

$\boldsymbol{\lambda_1}$	0,1,0.5,0.1,0.08,0.05,0.01,0.005,0.001
$\boldsymbol{\lambda_2}$	0,1,0.5,0.1,0.08,0.05,0.01,0.005,0.001
delta_scale	0.75,1,1.5,2,-0.75,-1,-1.5,-2

Table 4.1: Different values of $\boldsymbol{\lambda_1}$, $\boldsymbol{\lambda_1}$ and delta_scale considered



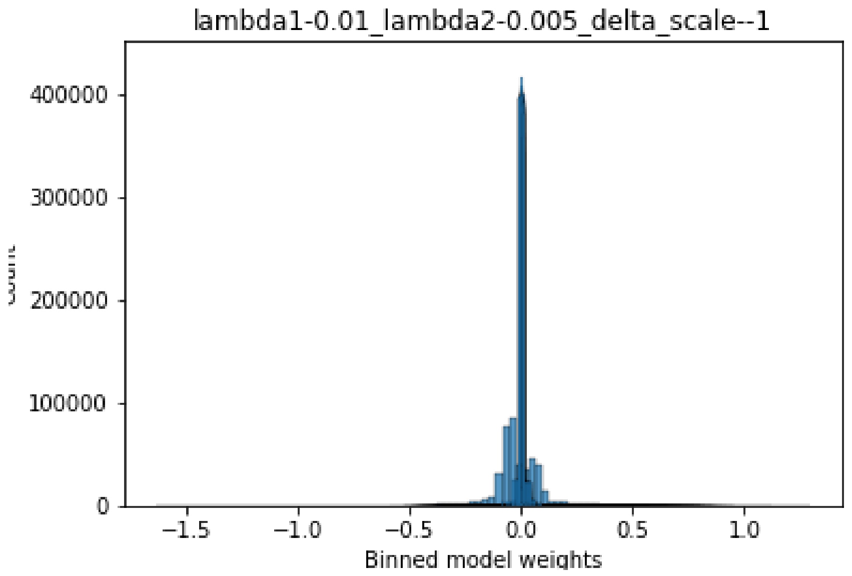


Figure 4.2: (a) Weight distribution graph before skewed weight training (b) Weight distribution graph after skewed weight training

In Fig.4.2(a) the variance of weight distribution is higher than in fig.4.2(b) where skewed weight training is implemented. Thus, it can be deduced that the skewed weight training results in weights being skewed around the reference weight.

Different values of λ_1 , λ_2 and delta_scale, where delta_scale represents the factor with which standard deviation was multiplied to obtain the reference weight, were considered to find the best range for which the weight values can be skewed without compromising on the accuracy.

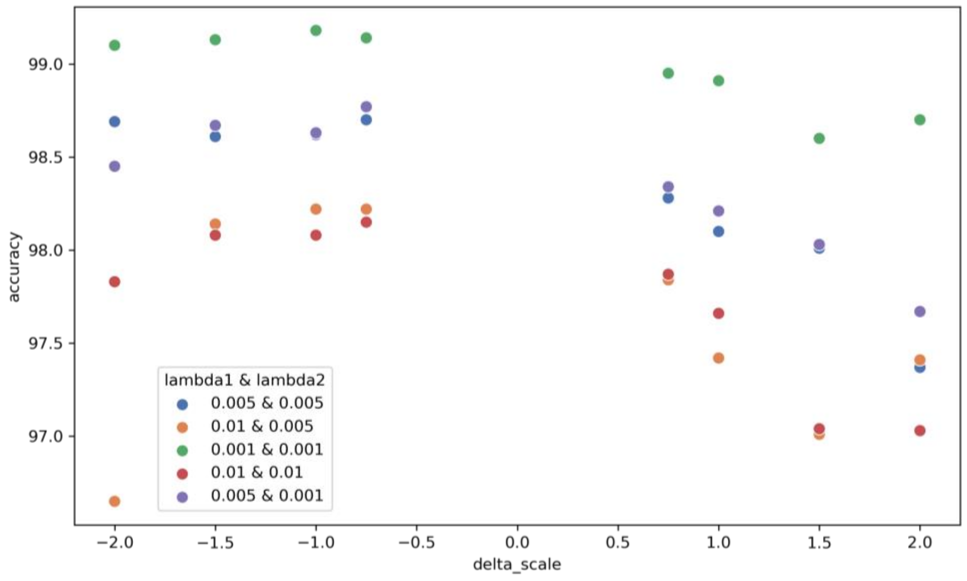


Figure 4.3: Scatter Plot for different values of λ_1 , λ_2 and delta_scale

It can be observed from fig.4.3 that lower value of λ_1 and λ_2 gives better accuracy. When λ_1 and λ_2 are 0.001, the accuracy is above 99% which is comparable with the accuracy of an unskewed model. Whereas λ_1 and λ_2 was increased to 0.01, the accuracy dropped below 98%. This is because, higher λ_1 and λ_2 means more penalization or increased loss value. This will increase the gradient and thus weight value decreases. As weight value decreases, the learning during forward propagation will be affected which will result in less accuracy. Also, accuracy is better when the delta-scale is in between -1 to 0.75 i.e., in the middle region. This is because the weight distribution follows a nearly normal distribution curve. And in a normal distribution majority of value (around 68%) is located in between (-1 * σ) and (1 * σ). As delta_scale represents the factor with which standard deviation was multiplied to obtain the reference weight, the performance of the model will be better when the reference weight is selected to be in the 68% range.

The above graph showed how different parameters affect the accuracy of a model. Now, we will consider how they affect the reduction in minimum and maximum weight values or variance of the weight distribution graph.

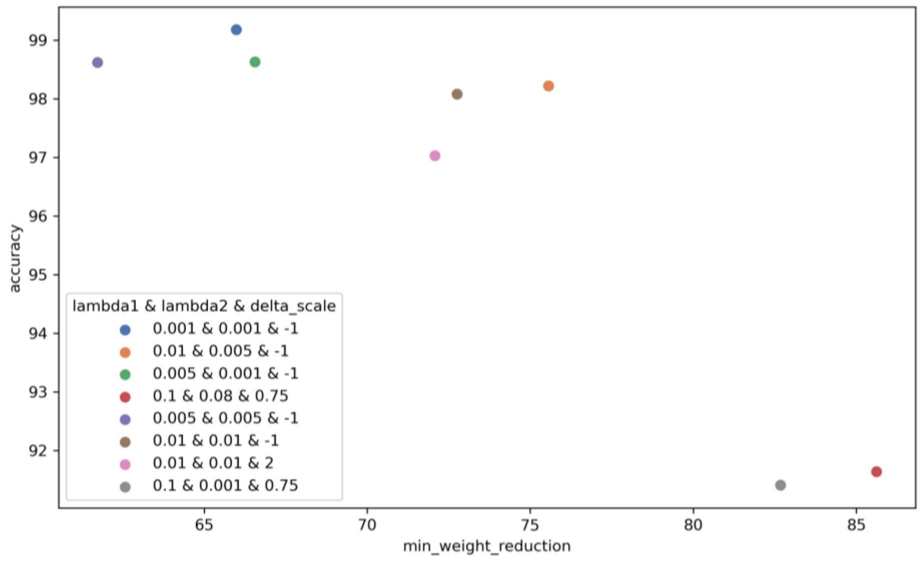


Figure 4.4: Accuracy Vs Min_weight reduction for different values of λ_1 ,λ_2 and delta_scale

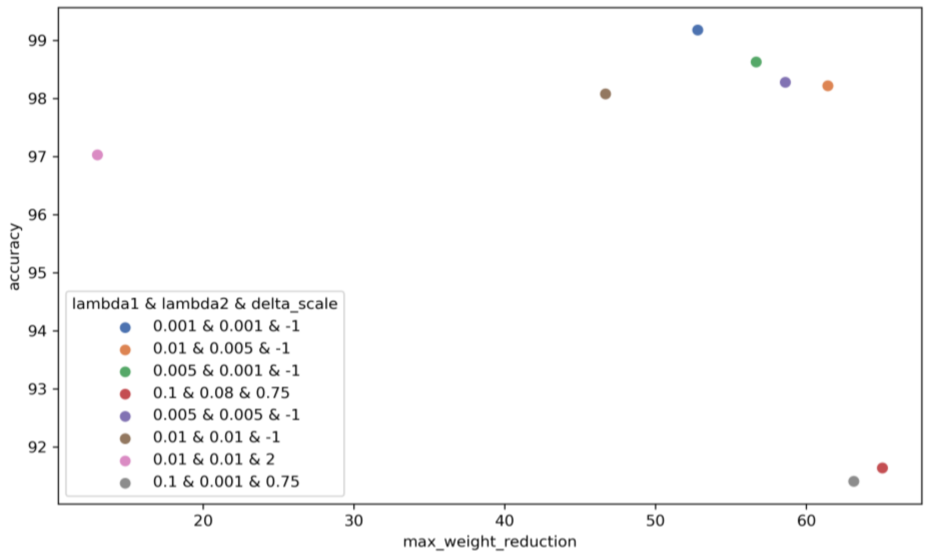


Figure 4.5: Accuracy Vs Max_weight reduction for different values of λ_1 ,λ_2 and delta_scale

Fig.4.4 and fig.4.5 shows the plot of accuracy vs reduction in minimum and maximum weight for different values of λ_1 , λ_2 and delta_scale in a skewed weight trained model. It can be observed that higher value of λ_1 , and λ_2 results in the maximum percentage of reduction in weight values but at the cost of accuracy. When λ_1 , λ_2 values were set at 0.1 and 0.08, around 85% reduction in minimum weight value was observed but accuracy was only 91.64%. This can be explained as higher λ_1 and λ_2 means weights will be penalized with a higher value and thus, more weights will be skewed around reference weight.

The model was trained for different values of λ_1 , λ_2 and delta_scale to find the best tradeoff between accuracy and skewed percentage. Table 4.2. shows a few results.

λ_1	λ_2	delta_scale	Max_weight reduction %ge	Min_weight reduction %ge	Accuracy
0.001	0.001	-1	32.8%	46.8%	99.18%
0.01	0.005	-1	52%	61.4%	98.22%
0.1	0.08	0.75	65%	85%	91.64%

Table 4.2: Accuracy and weight reduction percentage for different values of λ_1 , λ_2 and delta_scale

The best range of λ_1 , λ_2 and delta_scale that reduces the minimum and maximum weight value without significant compromise on accuracy was found to be $\lambda_1 = 0.01, \lambda_2 = 0.005 \; to \; 0.001$ and delta_scale = -1. Also, for $\lambda_1 > 0.1$ and $\lambda_2 > 0.08$, the model performs very poorly.

4.2. Skewed weight trained memristive model

The idea behind skewed weight training was to reduce conductance value across memristor cells to reduce the current and hence slow down aging. In the previous section, we discussed how skewed weight training reduces the weight variance by a significant amount without much loss in accuracy. Now, in this section, we will discuss how skewed weight training also helps to counter aging.

The skewed weight trained neural network model was converted into a memristive model using steps described in section 3.2.2. This results in a Skewed weight-trained

MDNN. The proposed aging function was applied to the MDNN and a test dataset was used to see the effect of aging on a skewed weight-trained memristive model.

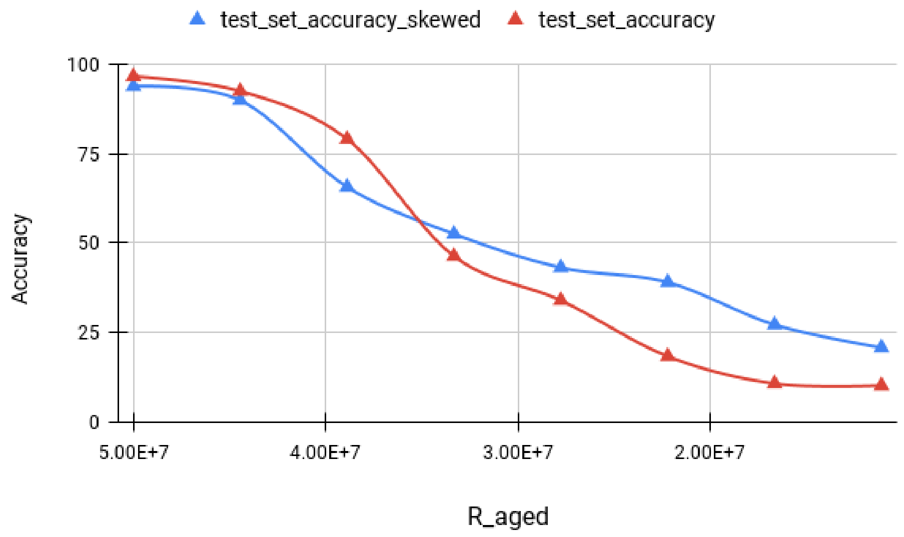


Figure 4.6: Effect of aging on Skewed weight trained model and traditional weight trained model

The effect of aging on a skewed weight trained MDNN and a traditional weight trained MDNN was compared as shown in Fig.4.6. When all the memristor were at pristine state, the accuracy achieved by the skewed weight trained model was comparable to the traditional model accuracy. But as the devices age, the decrease in accuracy of the unskewed model is steeper than the skewed model. This shows that when a model is skewed, the effect of aging is slowed down.

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

Memristor is a fundamental non-linear device that has a huge potential in Deep Learning applications. The property of the memristor to store the data and perform compute operations can be exploited in performing the dot operations in deep learning algorithms. The weights in a DL model can be stored as the conductance of the device and current can be converted to dot sum product of the matrix. This provides the benefits of reducing the latency of fetching the data from memory cache or DRAM and executing it on the processor. Like any electrical device memristor also has many non-linear characteristics and is prone to degrade with aging.

In this work, we designed a completely analog memristor-based photonic CNN architecture called BPHOTON-CNN. It integrates an efficient memristor with fast photonic components. Compared to state-of-the-art architecture, the proposed BPhoton-CNN shows improvement in energy and computational efficiency.

Then we studied how different non-idealities affect the performance of a memristor device. Aging, which is a non-reversible and inevitable process, challenges the reliability of a memristor crossbar. We modeled an aging function to consider the effect of aging in a memristive device. This function was incorporated as an extension to an existing simulator MemTorch[26]. Performance of the memristive neural network after introducing the non-ideality aging was studied and a decrease in accuracy was observed as the memristor device ages.

Then we demonstrated skewed weight training which is a software approach to counter aging. A skewed weight trained memristive network was introduced with the proposed aging function and its performance was studied. We showed that skewed weight trained MDNN ages slower than traditional weight trained.

5.2. Future work

In this work, a trained deep neural network was converted into a memristive neural network. The effect of non-idealities was studied on the memristive model for inference. This can be extended to study the effect of aging on a memristor model during training. Also, the effect of skewed weight training on a memristor model can be extended for training.

Along with aging, other non-idealities also affect the reliability of a memristor device. This calls for a function that considers all the non-idealities and designs a reliability-aware model to give the best performance in the long run.

REFERENCES

[1] M. Z. Alom *et al.*, “A State-of-the-Art Survey on Deep Learning Theory and Architectures,” *Electronics*, vol. 8, no. 3, Mar. 2019, doi: 10.3390/electronics8030292.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

[3] G. E. Moore, “Cramming More Components onto Integrated Circuits.”

[4] John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

[5] Dang, D., Khansama, A., Mahapatra, R. and Sahoo, D., 2020, September. BPhoton-CNN: An Ultrafast Photonic Backpropagation Accelerator for Deep Learning. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*(pp. 27-32).

[6] Hennessy, John L. and David A. Patterson. Computer Architecture: A Quantitative Approach. 4th ed., p. 289. Elsevier, 2007.

[7] Grecu, C., Ivanov, A., Saleh, R. and Pande, P.P., 2006, October. NoC interconnect yield improvement using crosspoint redundancy. In *2006 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems* (pp. 457-465). IEEE.

[8] Zukowski, M., Heman, S., Nes, N. and Boncz, P., 2006, April. Super-scalar RAM-CPU cache compression. In *22nd International Conference on Data Engineering (ICDE'06)* (pp. 59-59). IEEE.

[9] Smith, A.J., 1987. Line (block) size choice for CPU cache memories. *IEEE transactions on computers*, *100*(9), pp.1063-1075.

[10] Agarwal, N., Nellans, D., Ebrahimi, E., Wenisch, T.F., Danskin, J. and Keckler, S.W., 2016, March. Selective GPU caches to eliminate CPU-GPU HW cache coherence. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 494-506). IEEE.

[11] Farmahini-Farahani, A., Ahn, J.H., Morrow, K. and Kim, N.S., 2015, February. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA) (pp. 283-295). IEEE.

[12] Cooney, M., 2021. Gartner: Top strategic predictions for 2022 and beyond. *Network World*. Available at: https://www.networkworld.com/article/3637951/gartner-top-strategic-predictions-for-2022-and-beyond.html.

[13] Moravec, H., 1998. When will computer hardware matches the human brain. *Journal of evolution and technology*, *1*(1), p.10.

[14] Xin, J. and Embrechts, M.J., 2001, July. Supervised learning with spiking neural networks. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)* (Vol. 3, pp. 1772-1777). IEEE.

[15] Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y. and Brezzo, B., 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), pp.668-673.

[16] Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S. and Liao, Y., 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, *38*(1), pp.82-99.

[17] Li, J. and Cheng, C.K., 1995, April. Routability improvement using dynamic interconnect architecture. In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines* (pp. 61-67). IEEE.

[18] Nugteren, C., Van den Braak, G.J., Corporaal, H. and Bal, H., 2014, February. A detailed GPU cache model based on reuse distance theory. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)* (pp. 37-48). IEEE.

[19] Chua, L., 1971. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, *18*(5), pp.507-519.

[20] Strukov, D.B., Snider, G.S., Stewart, D.R. and Williams, R.S., 2008. The missing memristor found. *nature*, *453*(7191), pp.80-83.

[21] Williams, R.S., 2021. How we found the missing memristor. *IEEE Spectrum*. Available at: https://spectrum.ieee.org/how-we-found-the-missing-memristor.

[22] Cai, F., Correll, J.M., Lee, S.H., Lim, Y., Bothra, V., Zhang, Z., Flynn, M.P. and Lu, W.D., 2019. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nature Electronics*, *2*(7), pp.290-299.

[23] Nichols, B.M., Mazzoni, A.L., Chin, M.L., Shah, P.B., Najmaei, S., Burke, R.A. and Dubey, M., 2016. Advances in 2D materials for electronic devices. In *Semiconductors and Semimetals* (Vol. 95, pp. 221-277). Elsevier.

[24] James, A.P. ed., 2019. *Deep Learning Classifiers with Memristive Networks: Theory and Applications* (Vol. 14). Springer.

[25] James, A., 2020. Introductory Chapter: Challenges in Neuro-Memristive Circuit Design. *Memristors: Circuits and Applications of Memristor Devices*, p.3.

[26] Lammie, C., Xiang, W., Linares-Barranco, B. and Azghadi, M.R., 2020. MemTorch: An open-source simulation framework for memristive deep learning systems. *arXiv preprint arXiv:2004.10971*.

[27] Zhang, C., Sun, G., Fang, Z., Zhou, P., Pan, P. and Cong, J., 2018. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *38*(11), pp.2072-2085.

[28] Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S. and Srikumar, V., 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, *44*(3), pp.14-26.

[29] Gokmen, T. and Vlasov, Y., 2016. Acceleration of deep neural network training with resistive cross-point devices: Design considerations. *Frontiers in neuroscience*, *10*, p.333.

[30] Song, L., Qian, X., Li, H. and Chen, Y., 2017, February. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 541-552). IEEE.

[31] Hu, M., Strachan, J.P., Li, Z., Grafals, E.M., Davila, N., Graves, C., Lam, S., Ge, N., Yang, J.J. and Williams, R.S., 2016, June. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (pp. 1-6). IEEE.

[32] Vandoorne, K., Dambre, J., Verstraeten, D., Schrauwen, B. and Bienstman, P., 2011. Parallel reservoir computing using optical amplifiers. *IEEE transactions on neural networks*, *22*(9), pp.1469-1481.

[33] Gokmen, T., Onen, M. and Haensch, W., 2017. Training deep convolutional neural networks with resistive cross-point devices. *Frontiers in neuroscience*, *11*, p.538.

[34] Li, C., Belkin, D., Li, Y., Yan, P., Hu, M., Ge, N., Jiang, H., Montgomery, E., Lin, P., Wang, Z. and Song, W., 2018. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications*, *9*(1), pp.1-8.

[35] Shen, Y., Harris, N.C., Skirlo, S., Prabhu, M., Baehr-Jones, T., Hochberg, M., Sun, X., Zhao, S., Larochelle, H., Englund, D. and Soljačić, M., 2017. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, *11*(7), pp.441-446.

[36] Dang, D., Dass, J. and Mahapatra, R., 2017, December. ConvLight: A convolutional accelerator with memristor integrated photonic computing. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)* (pp. 114-123). IEEE.

[37] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, pp.1097-1105.

[38] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations (ICLR), 2015

[39] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), pp.2278-2324.

[40] Long, Y., Zhou, L. and Wang, J., 2016. Photonic-assisted microwave signal multiplication and modulation using a silicon Mach–Zehnder modulator. *Scientific reports*, *6*(1), pp.1-6.

[41] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), pp.211-252.

[42] IPKISS-Photonic Framework. (2018) [online]. Available: www.lucedaphotonics.com

[43] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T., 2014, November. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 675-678).

[44] The MNIST Database. (2018). [online]. Available: http://yann.lecun.com/exdb/mnist/

[45] Fadeev, A.V. and Rudenko, K.V., 2021. To the Issue of the Memristor’s HRS and LRS States Degradation and Data Retention Time. *Russian Microelectronics*, *50*(5), pp.311-325.

[46] Kvatinsky, S., Friedman, E.G., Kolodny, A. and Weiser, U.C., 2012. TEAM: Threshold adaptive memristor model. *IEEE transactions on circuits and systems I: regular papers*, *60*(1), pp.211-221.

[47] Krestinskaya, O., Irmanova, A. and James, A.P., 2019, May. Memristive non-idealities: Is there any practical implications for designing neural network chips?. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1-5). IEEE.

[48] Pickett, M.D., Strukov, D.B., Borghetti, J.L., Yang, J.J., Snider, G.S., Stewart, D.R. and Williams, R.S., 2009. Switching dynamics in titanium dioxide memristive devices. *Journal of Applied Physics*, *106*(7), p.074508.

[49] Kavehei, O., Cho, K., Lee, S., Kim, S.J., Al-Sarawi, S., Abbott, D. and Eshraghian, K., 2011, August. Fabrication and modeling of Ag/TiO 2/ITO memristor. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)* (pp. 1-4). IEEE.

[50] Yi, W., Savel'Ev, S.E., Medeiros-Ribeiro, G., Miao, F., Zhang, M.X., Yang, J.J., Bratkovsky, A.M. and Williams, R.S., 2016. Quantized conductance coincides with

state instability and excess noise in tantalum oxide memristors. *Nature communications*, *7*(1), pp.1-6.

[51] Yu, S., 2018. Neuro-inspired computing with emerging nonvolatile memorys. *Proceedings of the IEEE, 106*(2), pp.260-285.

[52] Zhang, S., Zhang, G.L., Li, B., Li, H.H. and Schlichtmann, U., 2019, March. Aging-aware lifetime enhancement for memristor-based neuromorphic computing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*(pp. 1751-1756). IEEE.

[53] Uddin, M., Majumder, M.B., Rose, G.S., Beckmann, K., Manem, H., Alamgir, Z. and Cady, N.C., 2016, July. Techniques for improved reliability in memristive crossbar PUF circuits. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 212-217). IEEE.

[54] Jiang, Z., Yu, S., Wu, Y., Engel, J.H., Guan, X. and Wong, H.S.P., 2014, September. Verilog-A compact model for oxide-based resistive random access memory (RRAM). In *2014 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)* (pp. 41-44). IEEE.

[55] Messaris, I., Serb, A., Stathopoulos, S., Khiat, A., Nikolaidis, S. and Prodromakis, T., 2018. A data-driven verilog-a reram model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(12), pp.3151-3162.

[56] Chen, B., Lu, Y., Gao, B., Fu, Y.H., Zhang, F.F., Huang, P., Chen, Y.S., Liu, L.F., Liu, X.Y., Kang, J.F. and Wang, Y.Y., 2011, December. Physical mechanisms of

endurance degradation in TMO-RRAM. In *2011 International Electron Devices Meeting* (pp. 12-3). IEEE.

[57] Krestinskaya, O. and James, A.P., 2018, July. Binary weighted memristive analog deep neural network for near-sensor edge processing. In *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)* (pp. 1-4). IEEE.

[58] Cai, Y., Lin, Y., Xia, L., Chen, X., Han, S., Wang, Y. and Yang, H., 2018, June. Long live time: improving lifetime for training-in-memory engines by structured gradient sparsification. In *Proceedings of the 55th Annual Design Automation Conference* (pp. 1-6).

[59] Hanlon, J., 2020. How to solve the memory challenges of deep neural networks. *TOPBOTS*. Available at: https://www.topbots.com/how-solve-memory-challenges-deep-learning-neural-networks-graphcore/.

[60] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.