Design and Implementation of Low Complex Memristive Circuits using Machine Learning Algorithms for Signal Processing Applications

THESIS

Submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Priyanka B G

ID. No. 2017PHXF0436H

Under the Supervision of

Prof. BVVSN Prabhakar Rao

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

2023

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the thesis titled **Design and Implementation of Low Complex Memristive Circuits using Machine Learning Algorithms for Signal Processing Applications** submitted by **Priyanka B G** ID No **2017PHXF0436H** for the award of Ph.D. of the Institute embodies original work done by her under my supervision.

Signature of the Supervisor : [illegible]

Name in capital letters: Prof. BVVSN PRABHAKAR RAO

Designation: Professor,

Department of Electrical and Electronics Engineering,

BITS-Pilani, Hyderabad Campus.

Date: 20-10-23

Declaration of Academic Integrity

I, Priyanka B G, declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included; I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action as per the rules and regulations of the Institute.

[illegible]

Priyanka B G

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. BVVSN Prabhakar Rao of the Department of Electrical and Electronics Engineering, BITS Pilani, Hyderabad Campus, for giving me the opportunity to work under his guidance. My heartfelt gratitude for his valuable guidance, constant encouragement, and unwavering support throughout my research and thesis completion.

I am most grateful to Dr. Souvik Kundu for co-supervising me during his tenure at the EEE department BITS Pilani, Hyderabad Campus. His treasured guidance and relentless support have greatly facilitated my research and thesis completion.

I sincerely thank my Doctoral Advisory Committee (DAC) members, Dr. Syed Ershad Ahmed, and Dr. Chetan Kumar V, of the EEE department, for their prized suggestions and advise on my research work. My heartfelt thanks to Prof. Sanket Goel, Prof. Alivelu Manga Parimi, and Prof. Subhendu K. Sahoo, the former and the present heads of the Department of EEE, for their suggestions, help, and timely support.

I am thankful to Dr. Venkateswaran Rajagopalan, EEE department, for guiding me with his prized insights about research. I would like to thank all the faculty members, lab technicians, and office assistants of the EEE department for their extended help and support. My sincere thanks to my lab mates and scholars of the EEE department for offering a helping hand during my entire research tenure. I would also like to thank the team of undergraduate and postgraduate students who assisted me during different stages of my thesis. I also like to acknowledge the support extended by the administration, Research division, and library staff, whose student-friendly response made things easier during the Ph.D. journey.

Special thanks to Prof. Venkata Vamsi Krishna Venuganti, Associate Dean of AGSRD (Academic-Graduate Studies and Research Division) and Dr. Prashant Kumar Pattnaik, Dr. Prashant Wali, Prof. BVVSN Prabhakar Rao, and Dr. Sudha Radhika, the former and the present Conveners, Doctoral Research Committee Department of EEE, for their help with various official requirements.

I am deeply indebted to my beloved parents, Mrs. Lakshmi Bai S.N and Mr. Ganganaik N, for their immense encouragement to take up this work, untiring support, and unconditional love. I wholeheartedly dedicate this work to them.

My heartfelt gratitude to my brother, Chi. Preetham B.G. Kodavath, who has always been there. I also thank my parents-in-law, all my family members, and specially my friends, who all have been supportive and caring. This journey has been made worthwhile by my soulmate, my husband, Mr. Bharath A, I gratefully acknowledge his steadfast support and belief in me. My warmest acknowledgments to my daughter, Taneeshka K.A., who is joy of our life.

God has been always there, through thick and thin, the divine presence being my companion at the most unexpected times. I shall be eternally grateful to the divine power that guided and propelled me along the way.

Priyanka B G

Abstract

The idea of digital data calculation has become difficult as the prevalence of electronic gadgets in daily life has increased. The huge amount of data generated has caused computing to grow exponentially, necessitating the use of computational platforms in terms of design effectiveness and efficiency. Powerful approaches like machine learning and neural networks has been recognized as a suitable technology to deal with the enormous amount of data generated in day-to-day life. Contemporary computing systems, Von Neumann's architecture has drawback of having a constrained data transfer speed between the memory and processing units. This bottleneck has a negative impact on the performance of modern artificial intelligence (AI) algorithms. Also, it has significantly increased the energy consumption of these systems, especially in data-intensive computations. A promising solution to address this bottleneck is to integrate processing and memory by performing computations at the location where the data is stored. Thus, memory bottlenecks and high costs associated with data movements between primary memory and the processor are the challenges witnessed by modern-day computers and has led to the need for in-memory processing. Recently, memristor-enabled in-memory Processing has gained attraction as a possible solution to the Von Neumann bottleneck. Memristor has the inherent ability to store and process the data in the same location. It is a two-terminal passive device whose conductance depends on the amount of current that has passed through it. It is being proposed as a possible replacement for transistors because of its compact size, ultra-high packing density (~40% improvement), low power consumption (one-hundredth of transistors), large ON/OFF resistance ratio (105), and high switching speed (~100 ps). The most profound application of a memristive crossbar is a vector product accelerator of linear time complexity. It has an innate ability to carry out matrix-vector multiplication through varying weights, which fosters its usage for implementing machine learning algorithms. Also, it is notable to mention that memristive state transitions were made possible by utilizing external voltage sources. Importantly, the transitions traverse through intermediate resistive states that exist between the extreme resistive limits (ON and OFF states). Interestingly, these intermediate states can be explored for memory storage and computational applications. These two important aspects of the memristor has been explored and implemented in this work. In one of the studies about the fabrication of memristors, the fabrication of Pt/Cu:ZnO/Nb:STO based memristor was demonstrated and obtained improved electrical performances such as low *SET/RESET* voltages, high *ON/OFF* ratio, good data retention, and stability, all these are useful for signal processing applications. Thus throughout the thesis, we have used Cu:ZnO memristor.

In this regard, a systematic investigation has been initiated to design a memristor crossbar-based architecture to compute the Pearson Correlation Coefficient(PCC) within the memory array. Three different applications were demonstrated by computing PCC based on the proposed architecture. The first application is computing PCC between noisy and denoised Electrocardiogram(ECG) signals. Whereas the second application is face recognition, where we compute PCC between face images in the presence of occlusions and varying expressions. Finally, the third application is computing PCC between two models of H1N1 disease prediction to verify their similarity. Further, an effort is devoted to analyze the effect of device variations on these applications. An attempt is also made to perform detailed system-level comparisons of the proposed architecture against a Von Neumann architecture.

Considering the memristor's approach to use the crossbar for analog matrix multiplications, which is more suitable for data-intensive applications, efforts were directed towards creating an in-memory processor utilizing memristive crossbar architecture for Bayesian text classification. The approach involved utilizing memristive switches to store data required for text classification. Text classification is a critical aspect of digital media, including natural language processing, sentiment analysis, image labeling, chatbots, spam filtering, and translators. To evaluate the efficacy of the proposed circuit, it was tested on two different datasets comprising a total of 55,575 texts from Short Message Service (SMS) and Internet Movie Database (IMDb) datasets. Exploring the analog computing properties of the memristor, the state transition property of the memristor, which exhibits controllable transitions within its stable resistive states, is also studied. In this work, for the first time to the best of our knowledge, efforts are put into developing Memristive State Machine (MSM) through a simulation route for edge detection in an image. The idea of MSM was further extended to perform a tunable edge detection system for image processing applications. The obtained edge detection was compared with other popular conventional software-based edge detection systems such as Canny, Sobel, Prewitt, Log, Zerocross and Roberts.

As we continue to examine the vector product accelerator application of the memristive crossbar array, we demonstrate a memristor-enabled computing in-memory architecture of an extensively utilized Binary Particle Swarm Optimization (BPSO) algorithm which has applications in diverse sectors. To the best of our knowledge, this is the first study pertaining to the implementation of BPSO on the memristor crossbar. Otsu's function and Kapur's entropy functions are considered the objective functions or the functions left for BPSO to optimize. To validate its potential, the commonly used Lena image was segmented and the performance of the memristor-crossbar was thoroughly analyzed. Further, this (Otsu and

Kapur’s entropy with BPSO) is applied for thresholding four T2- weighted transaxial brain Magnetic Resonance Imaging (MRI) scans of the widely used online open-access medical image repository by Harvard Medical School. Later the values obtained by memristor implementation of BPSO are compared with the values obtained by brute force methods.

The growing use of image fusion for feature extraction, image segmentation, and object recognition in different fields necessitates a need for a faster and more efficient hardware architecture. Knowing the benefits of using memristive crossbar arrays in image processing applications, our work validates the multi-focus image fusion using a memristive crossbar array. In this work, a novel image fusion architecture is proposed using Pt/Cu:ZnO/Nb:STO memristor crossbar array to implement an iterative Kernel Principal Component Analysis (KPCA) algorithm. To test this algorithm, experiments were carried out using different multi-focus images as well as Infrared-Visible Images. A comparison between the software and hardware experiments results was drawn using different quantitative metrics like Structural Similarity, Entropy, and Correlation Coefficient. The proposed concepts in this thesis, including the impact of analog matrix multiplication and state transitions, alternative programming paradigms, and the construction of modern low-power and less complex memristive circuits, open up a new vista in the field of futuristic electronics and serve as an alternative potential to CMOS-based architectures.

Table of contents

List of Tables

List of Figures

List of Abbreviations

AD	Analog to Digital
ASA	Analog Switch Array
ASIC	Application Specific Integrated Circuit
BL	Bit Line
BPSO	Binary Particle Swarm Optimization
CDC	Centre for Disease Control
CMOS	Complementary Metal-Oxide-Semiconductor
CoCoPIM	Computing Correlation Parameter In-Memory
CPSO	Chaotic Particle Swarm Optimization
CPU	Central Processing Unit
CSF	Cerebrospinal Fluid
Cu:ZnO	Copper doped Zinc Oxide
DA	Digital – to – Analog
DAC	Analog Converter
DCT	Discrete Cosine Transform
DPDT	Double Pole Double Throw
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
DWT	Discrete Wavelet Transform
ECG	Electrocardiogram
fMRI	Functional Magnetic Resonance Imaging
GBs	Gigabytes
GFT	Google Flu Trends
GHA	Generalized Hebbian Algorithm
GPU	General Processing Unit
HRS	High Resistive State
ILI	Influenza-like Illness
IMDb	Internet Movie Database
IMPLY	Implementation of Implication Logic
IoT	Internet of Things
ISAAC	In-Situ Analog Arithmetic in Crossbars
KHA	Kernel Hebbian Algorithm
LERs	Line Edge Roughnesses
LRS	Low Resistive State
MARSS	Micro Architectural and System Simulator
MBs	Megabytes
McPAT	Multicore Power, Area, and Timing
MRI	Magnetic Resonance Imaging
MRIMA	MRAM-Based In-Memory Accelerator
MSE	Mean Square Error
MSM	Memristive State Machine
MSSIM	Mean Structural Similarity Index Measure
NLP	Natural Language Processing
NVM	Non-Volatile Memory
OTFs	Oxide Thickness Fluctuations
PCA	Principal Component Analysis
PCC	Pearson Correlation Coefficient
PRIME	Processing-In-Memory Architecture

PSO	Particle Swarm Optimization
RAM	Random Access Memory
RDDs	Random Discrete Dopants
Re2PIM	Reconfigurable ReRAM-Based PIM
RKHS	Reproducing Kernel Hilbert Space
S&H	Sample and Hold
SMS	Short Message Service
SMS	States of Memristor Switches
SoC	System on Chip
SRAM	Static Random Access Memory
Stt-CIM	Spin-Transfer Torque Compute In-Memory
TL	Translinear
TSMC	Taiwan Semiconductor Manufacturing Company
TSV	Through-silicon Via
VLSI	Very-Large-Scale Integrated Circuitry
VTEAM	Voltage ThrEshold Adapted Memristor
Vwrite	Write Voltage
Vread	Read Voltage
WL	Word Line

Chapter 1

Introduction

The fact that massive amount of data have been continuously generated at unprecedented and ever-increasing scales is clear evidence that we live in a data deluge era. Numerous fields, including engineering sciences, social networks, business, biomolecular research, and security, collect and analyze large amount of data [1]. Notably, the amount of digital data produced by various digital gadgets is increasing at incredible rate. State-of-the-art computing systems suffer from the drawbacks of Von Neumann's architecture, which offers a limited rate of data transfer between the processing units and the memory. This bottleneck has significantly increased the energy consumption of these systems, especially in data-intensive computations. An intuitive solution to overcome the memory bottlenecks is to place the processor as close as possible to the primary memory [2]. However, such architectures, known as near-memory processors, have low data transfer latency and higher data bandwidths. The recent advancements in 3D silicon technologies have facilitated the fabrication of near-memory computers in which the processor is placed vertically on top of the memory using Through-Silicon Via (TSV) [3]. Despite its advantages, near-memory computing suffers from several drawbacks, such as difficulties in implementing virtual memory, cache coherency, data synchronization, memory management, etc. [2].

An alternative to near-memory computing is in-memory computing, where the computation happens within the memory unit [4]. In this architecture, there are no data movements and, as a result, do not suffer from the limitations of data latency and bandwidth. Memristor is proven to be an effective emerging device for in-memory processing, which is also a potential alternative to the CMOS technology. In concern to this, work further concentrates on the various applications of memristor cross bar array and its implementation in signal processing applications using machine learning algortithms.

1.1 Signal Processing

Signal processing is an interdisciplinary field that involves systems engineering, electrical engineering, and applied mathematics. It deals with the analysis of analog and digitized signals that represent various physical quantities. A wide range of signals can be analyzed in signal processing, including analog and digital signals that reflect physical quantities. These signals can be diverse, such as sound, electromagnetic radiation, images and videos, electrical signals acquired by different sensors, or waveforms generated by biological, control, or telecommunication systems [5]. Digital Signal Processing (DSP) is a critical field that focuses on the analysis of digitized and discrete sampled signals. It has significant applications in various areas of science and engineering, such as communications, control, computers, and economics. The common signals that are most used in all of these areas are image and text [5, 6]. The advancements in the field of signal and image processing have inspired researchers to create and deploy diverse algorithms and mathematical tools in recent times [7].

With the growing usage of electronic devices in day-to-day life, the amount of data or signals collected and its characteristics are challenging the traditional approaches of signal processing technologies as they are massive, unreliable, unstructured, and barely fit the statistical assumptions about the underlying system. Thus, the idea of digital data computation has become challenging. This has led to an exponential increase in computation which necessitated the need for computational platforms in terms of design performance and efficiency [8].

1.1.1 Image Processing

Our aspirations increased dramatically with the development of contemporary technology, which knows no limitations. The discipline of digital image processing is currently the subject of extensive research. The rate of research has continuously increased exponentially [9]. Image processing, a field of signal processing and a technique that converts an input image into a digital platform and processes it in such a way as to enhance the image or to get some helpful information out of it. In today’s modern world, it is a vast area of research with many different applications such as image morphology, neural networks, color image processing, medical image processing, image data compression, image recognition or face recognition, image fusion, edge detection, image thresholding pattern recognition, video processing, image enhancement or image sharpening, image segmentation, and so on [10], [11].

1.1.1.1 Face Recognition

Face recognition is one of the most important and trending applications of image processing. Current generation needs automatic face detection since it is crucial to robotics and artificial intelligence [10]. The essential phase of computerized face analysis is face detection. The output of a face detection system can serve as an input for various other systems, including face recognition, face tracking, face authentication, facial expression recognition, and facial gesture recognition. These systems are essential components of applications such as personal identity and access control, videophone and teleconferencing, forensic investigations, human-computer interaction, automated surveillance, cosmetology, and various other fields [12].

As the human face belongs to a dynamic object, many challenges like pose, occlusion, facial expression, existence or nonexistence of structural components, image orientation, and imaging conditions are prevalent in automatic face image analysis. Thus to overcome these challenges, algorithms with good accuracy and computational platforms with good efficiency are the need of the hour [12].

1.1.1.2 Edge Detection

Edge detection is a fundamentally important issue in picture analysis. Edge points can be viewed as places on a pixel where the gray level changes suddenly [13]. A border between two areas with comparatively unique gray-level characteristics is referred to as an edge so that only gray-level discontinuities can be used to detect where two sections merge. Edges are prominently useful for image segmentation, registration, and identifying objects in images as they define object boundaries [14].

Besides, it has several other applications in the field of computer vision, medicine, artificial intelligence, and biometric-based identification systems, which have become an essential part of our day-to-day life. Many researchers are striving to find an efficient algorithm or computational architecture for edge detection systems [15].

1.1.1.3 Image Thresholding

One of the essential initial stages of the image comprehension process, called image segmentation, tries to divide an image into areas so that each region groups adjacent pixels with comparable characteristics (intensity, color, etc.) [16]. It is crucial as the outcomes of all subsequent processes, including feature extraction, classification, and recognition, rely significantly on it. Numerous picture segmentation approaches have been documented in the literature as a result of extensive studies on the topic. These methods may generally be divided into four categories: thresholding, edge base, region growth, and clustering methods. The most fundamental technique is Image thresholding [17]. It is a straightforward and effective approach to distinguish objects from the background, as gray levels of pixels belonging to the object in most image processing applications differ significantly from those of pixels belonging to the background [18]. The thresholding operation involves dividing an image into segments that can be analyzed based on their shapes, sizes, relative positions, and other characteristics. This operation can be classified into two categories: global thresholding and local thresholding. Further, they are classified based on the gray levels, histogram, entropy, number of clusters, higher-order statistics, and local characteristics of the image [19], [20]. Known for their wide applications and different methods, many algorithms are evolved in recent years, which also has led to the challenges of implementing such algorithms without compromising the efficiency and computational complexity is the need of the hour.

1.1.1.4 Image Fusion

The usage of pictures in different applications is increasing rapidly by the day. In recent years, there has been much interest in image fusion. Image fusion is the process of combining multiple images to improve the information content in a picture [21]. It is used for object detection and has importance in many fields like military, medical, automated industry, etc. Image fusion is most important in medical imaging- combining Computed Tomography (CT) scans and MRI imaging [22]. Different fusion techniques have been proposed, primarily in remote sensing and computer vision (such as night vision). At the same time, hardware implementations have also been made to address real-time processing in many application domains.

Multiple methods can be used to perform image fusion for a broad range of applications that involve high data transfer rates. Thus, in order to eliminate the need to transfer data from different memory sources, new device opportunities are being explored, including memristors for various logic implementations and memory operations [23].

1.1.2 Text Classification

In recent times, text classification issues have received extensive study and have been addressed in numerous practical applications. Many academics are currently interested in developing applications that make use of text classification techniques, particularly in the context of recent advances in Natural Language Processing (NLP) and text mining. It can classify new documents into pre-defined classes [24]. Currently, it is a sophisticated process involving not only training of models but also numerous additional procedures, for example pre-processing data transformation and dimensionality reduction. It is possible to break down vast majority of text classification and document categorization systems into four steps such as feature extraction, dimension reduction, classifier selection, and assessments. It continues to be a significant research area, with researchers exploring various techniques and their combinations in complex systems [25].

Thus, recent developments in various signal and image processing fields have inspired researchers to design and deploy different algorithms and mathematical tools. However, as the era of big data approaches, datasets are becoming increasingly large and complex, making it challenging to process them using traditional learning methods. Also, learning from conventional datasets were not designed to handle the high volume of data, and as a result, it may not work effectively with large and complex datasets [7]. As a result, artificial intelligence (AI) techniques have emerged as a solution. In the past decade, machine learning techniques have been widely adopted in several large and complex data-intensive fields such as medicine, astronomy, biology, and more. These techniques offer potential solutions for extracting valuable information from vast amounts of data [1].

The challenges witnessed by conventional computing architectures (Von Neumann architecture) used today for processing such algorithms with huge volumes of data are memory bottlenecks and high costs associated with constant data movements between the memory and the processor, creating the need for a system that significantly reduces the memory bottleneck and improves the efficiency of the system in terms of power consumption and speed [26].

1.1.3 Signal Processing using Transistor-based Technology and its Disadvantages

The majority of signal processing hardware systems currently available use multiplexing to consolidate multiple recording channels into one or several processing units to handle complex computational tasks [27]. These systems are typically built using silicon-based Complementary Metal-Oxide Semiconductor (CMOS) technology and employ the conventional von Neumann architecture, where memory and data computing units are physically separated. Memory bottlenecks and high costs associated with data movements between the primary memory and the processor are the challenges witnessed by modern-day computers. Typically, these systems convert analog signals received into digital signals first and then compress and process them in the digital domain using various Application-Specific Integrated Circuits (ASICs) [28]–[31]. The implementation of machine learning algorithms and spiking neural networks in hardware has been an active research field, and recent publications have shown that it is feasible to apply these techniques to industrial applications, such as signal processing of complex data sets [32]–[34]. Recent advancements in nanotechnology have allowed for low power and high device integration, reigniting interest in the development of neural networks in hardware. This has led to the development of neuromorphic circuits that incorporate resistive nanoscale devices into crossbar topology with CMOS circuitry. BCrossnets is one such example that demonstrates the challenges and techniques involved in the design of neuromorphic circuitry using this approach [35], [36]. However, the design of such systems is still facing many challenges, such as power budget, delay, and scalability, mainly to catch up with the exponentially increasing number of data sets. Memristor, proposed by Chua in [37], is a promising element in this area as it may overcome the above mentioned inabilities. This inability is reduced by two factors: the small size of the memristors concerning their functionality and the ability to connect the memristors with crossbars [38].

1.1.4 Disadvantages of Transistor-based Non-Volatile Memory Devices

A Non-Volatile Memory (NVM) is a type of memory that can continue to store data in the computing system even when the power source is off [39]. Modern computing systems typically use transistor-based memory such as Static Random Access Memory (SRAM), Dynamic Random-Access Memory (DRAM), and Flash memory for data storage applications [40]. The major drawbacks of these transistor-based memories are, for instance, that SRAM is expensive, non-volatile only while powered, and has a small storage size in terms of megabytes (MBs). Although DRAM is cost-effective and highly scalable, it consumes more power, is volatile, and has a moderate storage size in gigabytes (GBs). Flash memory or hard disk is non-volatile and has a storage capacity of more than 100 GB but has poor latency [40]. Most essentially, it is vital to note that further miniaturization of the transistors would lead to undesirable device performances, such as junction breakdowns and tunneling phenomena. It was predicted, by Gordon Moore (in 2015), that Moore's law would conclude within the next decade [41]. These primary limitations of the above-mentioned existing data storage technologies have created the need for the invention of a novel data storage and computing element that has the advantages of above mentioned current technologies, consumes less power and area, and has more latency. The necessity for this kind of device, having future scope for improvements in their data storage and computational capabilities, has led to the discovery of memristors [42].

1.2 Memristor

A memristor is a short form of "memory & resistor". It is a nonlinear two-terminal passive electrical component that exhibits a unique property known as memristance. Memristance is the ability of the device to remember the history of the current that has flowed through it and to adjust its resistance accordingly. This makes the memristor a promising key element for next-generation memory and other applications, such as artificial neural networks and analog signal processing [43]. It has a smaller device dimensions of < 50% chip area per bit, compared to flash memory. It stores data in its resistance value that depends on the applied voltage's polarity, magnitude, and frequency, resulting in much simpler read and write cycles [42]. Analogous to transistors, they can be miniaturized in their device dimensions and are used in analog and digital electronic systems. Unlike transistors, memristors can store analog values in the form of device resistance [44], [45]. Memories based on memristors also have other advantages, including very high packing density, zero power dissipation during data retention, and fewer components per computation, compared to their transistor counterparts [14].

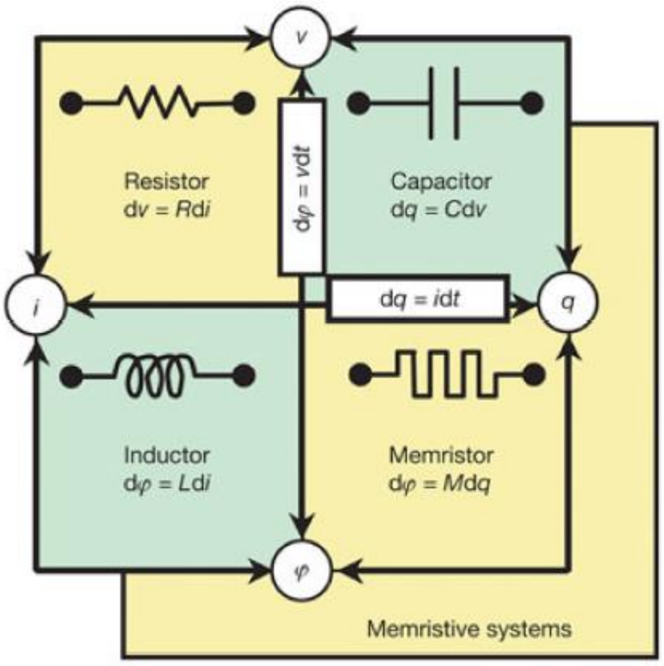


Fig 1.1: The four relations describing the essential two-terminal passive elements [40].

The memristor is the fourth fundamental circuit element, which was theoretically formulated by Chua [46]; the other three basic circuit elements are the resistor, capacitor, and inductor, which are as shown in Fig. 1.1. It is a two-terminal electronic circuit element that exhibits controllable resistive state transitions based on the amount of charge that has passed through it. Resistance of the memristor, known as memristance, is defined by Chua as the rate of flux (ϕ) versus charge (q) passing through it (i.e., M = dϕ/dq) [47]. It is important to mention that there exists two types of memristors. If the q *and* φ relation is such that q is the independent variable, then it is charge-controlled. i.e.,

$$\varphi = f(q) \tag{1.1}$$

Differentiating both sides of the above equation with respect to time *t* results in,

$$\frac{d\varphi}{dt} = \frac{df(q)}{dq}\frac{dq}{dt} \tag{1.2}$$

The above equation can be written as:

$$v(t) = M(q)i(t) \tag{1.3}$$

where $M(q)$ is memristance in Ω.

$$M(q) = df(q)/dq \tag{1.4}$$

On the other side, if q *and* φ are related such that q is expressed in terms of φ, then it is a flux-controlled memristor [40]. Analogous to the previous case,

$$q = f(\varphi) \tag{1.5}$$

$$\frac{dq}{dt} = \frac{df(\varphi)}{d\varphi}\frac{d\varphi}{dt} \tag{1.6}$$

$$i(t) = W(\varphi)v(t) \tag{1.7}$$

where $W(\varphi)$ is the conductance of the memristor (or memductance), which has the units of *Siemen.*

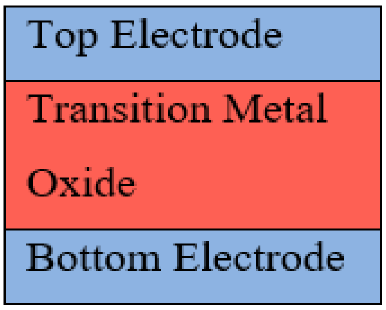
$$W(\varphi) = \frac{df(\varphi)}{d\varphi} \tag{1.8}$$

It is important to note that q and φ are expressed mathematically as the time integral of i(t) and v(t), respectively, and need not have any physical interpretations [42]. Observe that Eqn. (1.3) and Eqn. (1.4) can be interpreted as Ohm's law except that the memristance M(q) at any time t = t0 depends on the entire history of i(t) from $t = -\infty$ to t = t0. Similarly, the memductance W(ϕ) in Eqn. (1.8) depends on the entire history of v(t) from $t = -\infty$ to t = t0. It results from Eqn. (1.3) that the charge-controlled memristor defined in Eqn. (1.1) is equivalent to the charge-dependent Ohm's law. Similarly, a flux-controlled memristor corresponds to the flux-dependent Ohm's law [42]. In 2008 a solid-state memristor was developed and implemented by Hewlett-Packard (HP) Labs. It was proven that it is a novel nonlinear two-terminal nanoscale element with unique properties like memory capacity, a switching characteristic, and continuous input and output properties [48].

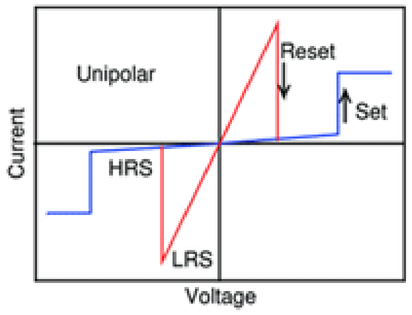
1.2.1 Characteristics of Memristor

A memristor is a device that stores information on how much charge has passed through it, and in which direction. This information is encoded in its internal state variable; the resistance of the memristor increases if current passes in one direction and decreases otherwise. It is well-known for its basic structure of two terminals and a non-volatile internal state variable resistance [40]. It consists of a transition metal oxide sandwiched between two electrodes, as shown in Fig. 1.2 (a) [49]. It has two resistive states; the Low Resistive State (LRS) of the memristor represents logic 1, whereas the High Resistive State (HRS) is read as logic 0. When the memductance is the least, the device is said to be in HRS, whereas the condition after electroforming is the LRS [40, 42]. The most significant feature of a memristor is the pinched hysteresis loop in its I-V characteristics. When a variable power source is applied to the memristor, the current through it is zero if the applied voltage is zero and vice-versa. When the voltage has gradually increased, the device, which is initially in the HRS , will change to LRS at the SET voltage. Further, if the voltage is slightly increased or decreased, the device will remain in its LRS. In order to RESET the device back to its HRS, one needs to reduce the voltage below the threshold value of the device.

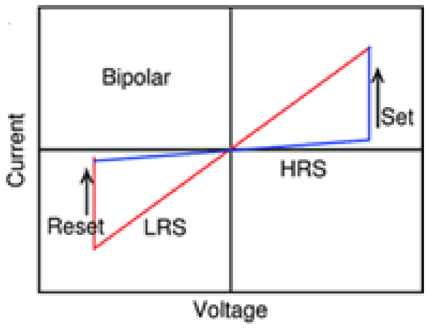
There are two kinds of resistive switching mechanisms, namely, unipolar and bipolar, as shown in Fig 1.2 (b) and Fig. 1.2 (c), respectively [49]. In the case of unipolar switching, both *set* and *reset* mechanisms occur at the same polarity of voltage; whereas in bipolar switching, logic 1 and logic 0, are written onto the memristor by passing positive and negative currents, respectively [49].



(a)



(b)



(c)

Fig 1.2: (a) Structure of a memristor, (b) Unipolar switching, and (c) Bipolar switching [49].

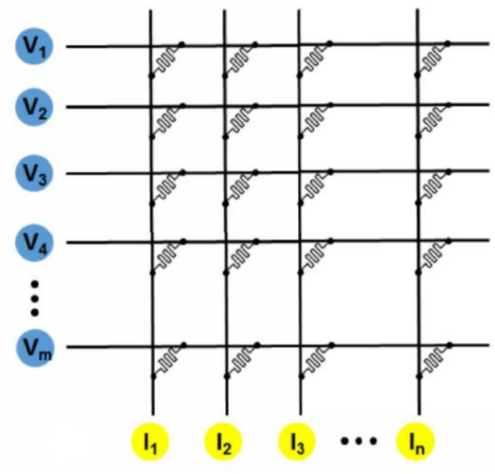


Fig 1.3: The crossbar array consists of perpendicular rows and columns with the memristor devices sandwiched at each cross point [14].

Memristive devices are customarily manufactured in a crossbar array architecture as shown in Fig. 1.3. Memristive crossbar array architecture facilitates dense memory of $4F^2$ (F is the feature size) and is less expensive because of the simplicity of the cross bar array manufacturing process [40]. The principal advantage of this crossbar array is the simplicity of its structure. This architecture is proven to be power efficient, has high-speed computation, and has a long retention time of up to ten years [40]. Thus, the memristor continues to show great potential in processing in-memory and computational applications.

In one of the recent studies, the fabrication of Pt/Cu:ZnO/Nb:STO based memristors has been demonstrated and obtained improved electrical performances such as low *SET/RESET* voltages, high *ON/OFF* ratio, good data retention, and stability, which are useful for various signal processing applications [50]. Moreover, the Cu:ZnO based memristor is a bipolar switching device, which has a stable switching attributed by retention up to 10^6 s, high endurance [50] and R_{on}/R_{off} ratio of ~ 2 X 10^3 which provides highly varied range for the state variable (ranging from 0 to 1). The fabricated Pt/Cu:ZnO/Nb:STO memristor was mimicked by utilizing The Voltage ThrEshold Adapted Memristor (VTEAM) model [51], and circuits were simulated using MATLAB Simulink. In the literature, other important models namely ion drift, Berkeley and Yakopcic's models were explored and many challenges were present to model fit the device [52]. In order to point out the important features, the ion drift model works more aptly on devices that has transition hysteresis for all voltages (eg: TiO_2 memristor). However, this model is not suitable for ferroelectric or zinc oxide memristors or for our case where the ion drift occurs only beyond certain threshold voltage. Similarly, the Berkeley model and Yakopcic's model have been prototyped based on the concept that the current variation takes place instantaneously irrespective of its previous state [53]. Unlike aforementioned models, the VTEAM model closely mimicked the fabricated device owing to the precisely controlled current variation based on applied voltage [54]. Thus as per the above mentioned reasons, in each work mentioned in this dissertation, Pt/Cu:ZnO/Nb:STO memristor is used and VTEAM model is chosed to behaviourally model the device. Details of the VTEAM modelling of memristor is as given below:

At any instant of time, a voltage-controlled memristor can be mathematically represented as:

$$\frac{dw}{dt} = f(w, v) \tag{1.9}$$

$$i(t) = G(w, v).v(t) \tag{1.10}$$

w is the internal state variable of the device. v(t) and i(t) are the voltage across the device and the current passing through the device, respectively. G is the conductance of the device. As per the VTEAM model, f (w,v) is:

$$\frac{dw}{dt} = \begin{cases} k_{off}\left(\frac{v(t)}{v_{off}} - 1\right)^{a_{off}} f_{off}(w), 0 < v_{off} < v \\ 0, v_{on} < v < v_{off} \\ k_{on}\left(\frac{v(t)}{v_{on}} - 1\right)^{a_{on}} f_{on}(w),\ v < v_{on} < 0 \end{cases} \tag{1.11}$$

Here, v_{on} and $v_{off}$are known as the on and off voltages, respectively. The parameters k_{off}, k_{on}, a_{off} and a_{on} are constants and these are device dependent. f_{off} and f_{on} are known as the window functions, and they are used to ensure that w is bounded such that w ∈ [w_{on} , w_{off}]. They are rectangular step-functions defined as follows:

$$f_{off} = \begin{cases} 1\ if\ w < 1 \\ 0\ otherwise \end{cases} \tag{1.12}$$

$$f_{on} = \begin{cases} 1\ if\ w > 0 \\ 0\ otherwise \end{cases} \tag{1.13}$$

For the model described in [53],

$$w = \frac{x}{D} \tag{1.14}$$

Where x is the length of the doped/polarized region and D is the active material thickness. The current-voltage relationship for the model is:

$$v(t) = i(t) * (R_{on} + (R_{off} - R_{on})(\frac{w - w_{on}}{w_{off} - w_{on}})) \tag{1.15}$$

As x can take values between 0 and D, w_{on}=0 and w_{off}=1. Substituting this in the above equation, it can be deduced as:

$$v(t) = i(t) * (R_{on} + (R_{off} - R_{on})w) \tag{1.16}$$

The logic state of a memristor can be changed by applying sufficient voltage across it. To change state from 0 to 1, a negative voltage, v_{set}, with magnitude greater than v_{on} is applied across the memristor. Similarly, to change the logic state from 0 to 1 a positive voltage, v_{reset}, of magnitude greater than v_{off} is applied across the memristor. Apart from these binary states, the analogue nature of memristors permit multiple states in between the LRS and HRS.

The memristor crossbar array is a structure comprising of vertical and horizontal metallic wires with memristors at the intersection of each vertical and horizontal wire [55], [56], as shown in Fig 1.3, where V_1, V_2…V_m is the input voltage, and I_1, I_2...I_n is the output current. The memristance of any memristor can be tuned by applying a suitable voltage, v_i, across at the ends of the vertical and horizontal wires which have the corresponding memristor at their intersection. To ensure that this doesn't change the memristance of the other memristors in the row or column, a voltage of v_i/2 is applied across these memristors such that $|v_i/2| < |v_{off}|$ (or $|v_{on}|$) $<|v_i|$. From Eqn. (1.11), this ensures that dw/dt=0 so the memristance won't change. One of the most important applications of memristor crossbars is the implementation of the matrix dot product, and this paves the way for various neuromorphic architectures using this structure.

For implementing matrix dot product using memristor crossbar, first, the crossbar with memconductance of all memristors proportional to the elements of the first matrix is initialized. Later, voltages with magnitude in the range (v_{on},v_{off}) and time pulse width proportional to the corresponding element of the other matrix involved for the dot-product are applied. As mentioned above, this ensures that their conductance doesn't change. From Eqn. (1.10), the output currents are obtained and integrated. Mathematically, this is equivalent to the dot product as:

$$i = g_{i,j} * v_m \tag{1.17}$$

$$Q = \int_0^{t_{i,j}} i\,dt = g_{i,j} * v_m * t_{i,j} \tag{1.18}$$

Where, $g_{i,j}$ is the memconductance of the memristor located at the intersection of the i^{th} row and j^{th} column, v_m is the magnitude of the voltage pulse, $t_{i,j}$ is the time pulse voltage width applied across the memristor, and Q is the charge obtained from integrating the current. In Eqn. (1.18), it is observed that the charge Q is directly proportional to the product $g_{i,j} * t_{i,j}$.

1.3 Literature Review

In 1971, the memristor was theoretically predicted to be a fourth fundamental circuit component relating the flux-linkage and charge [46]. Unique properties of memristors, such as simple physical structure, high density, non-volatility, high scalability, and low power consumption, have promising applications in the areas of non-volatile memory, Very-Large-Scale Integrated Circuitry (VLSI), artificial neural networks, image processing, and pattern recognition. Memristors have wide applications as a discrete device and also as an array devices [57], as these memristive devices are customarily manufactured in a crossbar array architecture, where intersections between the orthogonally oriented bottom and top lines form one storage element at each cross point yielding high scalability and efficiency [40]. The author has categorized the appropriate applications of memristors as discrete device and array device applications. Among which the crossbar array applications are classified into analog and digital applications, further the analog applications of crossbar arrays are classified as neuromorphic networks and field-programmable analog arrays, and digital applications as content addressable memory, non-volatile memory, and logic circuits [57]. It was demonstrated that memristors perform implication logic, whereas memristive circuits can play the dual role of a logic gate and a latch [58], [59]. All the logic gate functionality was successfully implemented by utilizing memristive circuits. In addition, it was shown that all Boolean functions were implementable with the least number (two) of memristors [60].

The important analog applications of memristors are tunable analog circuits, filters, and chaotic circuits. Within the tuneable analog circuits, utilization of a memristor resulted in improved performance and circuit complexity [44], [61]. Interestingly, employment of a memristor in the conventional amplifier circuits led to the enhancement of linearity in tuneable gain amplifiers [62], [63]. Most importantly, memristor-based circuits were employed as filters for signal processing applications [64], [65]. It is noteworthy to mention that the chaos applications of memristor are mainly in random number generation, medical, and communication. It was identified the memristors could demonstrate chaos by utilizing Chua's circuit. This study was extended such that chaotic circuits were employed in oscillators resulting in improved nonlinearity, signal-to-noise ratio, and frequency of oscillation [66], [67]. It is important to mention that memristors have shown tremendous potential to act as operational and computing elements. These devices have been utilized in the development of programmable threshold comparators [44]. Furthermore, memristor-based circuits have closely mimicked the transfer characteristics of operational amplifier circuits [68]. Interestingly, memristors were utilized for mathematical operations such as multiplication and division through the employment of a memristive crossbar array and its analog characteristics [69], [70]. These analog computation-based operations are essential in the fields of signal processing and control systems [71].

In the present digital era, digital image processing plays the most important role in areas like medical imaging and geographical information [72]. On the other hand, Neuromorphic computing is one emerging technology post Moore's law era, where Neuromorphic computing systems are highly connected and parallel, consume relatively low power, and process in memory [73]. Artificial intelligence, image processing, and related fields require massively parallel computations, and memristors can potentially be used to solve such problems [74]. Exploring these applications and knowing the need for memristors in rising applications like image processing and neuromorphic computing, our work aims to develop optimal memristor-based circuits using machine learning algorithms for different Image processing applications. Memristive crossbars offer reconfigurable non-volatile resistance states and could remove the speed and energy efficiency bottleneck in vector-matrix multiplication, a core computing task in signal and image processing [11]. Can Li et al. has demonstrated important applications such as signal processing, image compression, and convolutional filtering, which are essential in the development of the Internet of Things (IoT) and edge computing [11].

In one of the studies [75], the author has proposed memristive cross bar arrays for the storage of image and image processing applications. In [76], the author has presented the memristor-based chaotic circuit for text or image encryption and decryption. Also, the author has shown the feasibility and practicality of memristor-based chaotic circuits for encryption and decryption from the obtained simulation results. S. Muthulakshmi et al. proposed memristor-based approximate full adder and subtractor architecture, which is verified for image addition and foreground detection, respectively [77]. The authors in [78] have demonstrated that the memristive grid is an efficient tool for achieving image smoothing and edge detection. Memristors are being considered for image fusion due to their low energy consumption and easy integration [79]. The process of fusion can be done through a pulse-coupled neural network, which has other benefits like denoising the image and edge extraction [79]. In [80], the author has proposed image edge detection using swarm intelligence and ant colony optimization using memristive devices.

Thus, it can be seen that memristor is a promising key element for image-processing applications.

1.4 Scope of the Present Investigation

One of the fundamental problems in science is processing signals. In this era of DSP, the increasing size and complexity of data have led to the development of advanced signal-processing techniques, which are more robust and efficient in handling massive and unstructured data. The use of fast computing and algorithmic developments has enabled these techniques to be applied in various fields, including image and speech recognition, natural language processing, and bioinformatics. Moreover, the emergence of machine learning and deep learning techniques which rely heavily on advancements in computing power and algorithmic development to process and analyze massive and complex datasets has revolutionized the field of signal processing. These approaches or algorithms involve a lot of data transfer during the processing. Present-day computers face the limitations of memory and the high costs of moving data between the main memory and the processor. Since the inception of using big and complex data files, Von Neumann's architecture's shortcomings have been limiting the performance of computing systems. The difference between the two peak bandwidths needed by the processor and main memory is main reason for this bottleneck.

From the extensive literature review, it is evident that memristor-based devices are the need of an hour to build high-speed, low cost and low-power-consuming electronic systems. The memristor stores and processes data at the same location. One approach is to compute digital logic on a memristor crossbar array. The other approach is to use the crossbar for analog matrix multiplications. Of these two approaches, the second approach is most suitable for data-intensive applications as matrix multiplications are the most used operations in these applications. It is also seen that image processing and machine learning are one of the most significant topics of this digital era, and memristors can be used as an active element in these applications in order to build highly efficient systems. On the other hand, marchine learning is an advanced technology that emulates the human brain and makes the process faster. Very limited studies are currently available in literature that use machine learning to develop advanced signal processing applications on memristive circuits. Thus, it is apparent that the implementation of memristive crossbar arrays for advanced signal processing applications using machine learning algorithms are vital. Therefore, in order to fulfill all the above-mentioned requirements, this work aims to design and implement low complex memristive circuits using machine learning algorithms for signal processing applications.

1.5 Objectives of the Thesis

The main objective of the proposed research is to implement low-power and less complex memristive circuits using machine learning algorithms through a simulation route for realizing advance signal processing applications.

In order to accomplish this, the specific objectives of the work are as follows:

(1) To implement an innovative memristor crossbar-based architecture, CoCoPIM (Computing Correlation Parameter In-Memory), to accelerate Pearson Correlation Coefficient computations and further demonstrate different applications based on this architecture, such as computing correlation between ECG signals, faces, and H1N1 models.

(2) To implement a devoted in-memory processor for Bayesian text classification using memristive crossbar architecture, in which memristive switches will be employed to store information required for the classification of text.

(3) To develop an edge detection system for an image processing application exploring the possible resistive states within the Memristive State Machine (MSM) of a memristor.

(4) To investigate a memristor-enabled computing in-memory architecture of the widely used Binary Particle Swarm Optimization (BPSO) algorithm for image thresholding.

(5) To explore different algorithms for image fusion and to implement that using a memristor crossbar array.

1.6 Thesis Organization

The entire work has been presented in 7 chapters.

Chapter 1: This chapter includes an introduction to signal processing and its applications in image processing and text classification. An extensive literature survey of metal oxide resistive switching devices and how beneficial the use of memristor devices as non-volatile memory is carried out. The advantages of using the memristor device for signal processing applications has been discussed. In view of the literature survey, the scope and objective of the present work have been formulated and presented.

Chapter 2: In this chapter, CoCoPIM architecheture is proposed for the computation of Pearson Correlation Parameter and computation of similarities between ECG signals, faces, and H1N1 models are demonstrated using the proposed architechture. Also, the current-voltage characteristics of the memristor and the methodology to store data in it are explained. The proposed CoCoPIM's architecture, data mapping, acceleration steps, and microarchitecture are proposed, and its applications are demonstrated. It is concluded with system-level comparisons.

Chapter 3: In this chapter, a memristor-based Bayesian text classifier for applications in digital media is developed, and the methodology is explained in detail. The efficacy of the proposed circuit in classifying the texts with high accuracy is determined. Also, the chapter is concluded with the experimental results obtained and discussed.

Chapter 4: This chapter discusses the Memristive State Machine (MSM) developed through a simulation route for edge detection systems for image processing applications. The obtained edge detections were compared with other popular conventional software-based edge detection systems such as Canny, Sobel, Prewitt, Log, Zerocross and Roberts. Later the efficacy of MSM was discussed for future generation's accurate and faster real-time image processing applications.

Chapter 5 : This chapter introduces a memristor-enabled computing in-memory architecture of the widely used Binary Particle Swarm Optimization (BPSO) algorithm is demonstrated. Then memristor model and modeling of the crossbar for implementation of BPSO for image thresholding are described. Further, the chapter is concluded with an explanation of the results for multi-thresholding on standard images as well as the Brain MR images, followed by the results with device variations.

Chapter 6: In this chapter, a novel architecture for image fusion using memristor crossbar implementation of an iterative Kernel PCA algorithm is described. Further, to test this algorithm, experiments were carried out using different multi-focus images as well as Infrared-Visible Images. Later the conclusions are drawn based on the results obtained and compared between the software implementation and memristor implementation.

Chapter 7: In this chapter, important findings are summarized, conclusions are drawn, and the contribution of the work is mentioned.

Chapter-2

Memristors Enabled Computing Correlation Parameter In-Memory System: A Potential Alternative to Von Neumann Architecture

2.1 Introduction

As discussed in the chapter 1, the Von Neumann bottleneck has been limiting the performance of computing systems since the inception of using large and complex data files [81]–[84]. This bottleneck arises from the difference between the two peak bandwidths required by the processor and main memory. On the one side, the current high-end processors, which contain multiple cores on one chip, further increase the bandwidth requirement compared to single-core processors [85]. On the other side, the continuous transistor scaling led to improvements in clock frequency but never improved the memory access speed by the same magnitude. These two trends have increased the number of idle processor clock cycles per memory access [84]. As a result, the idle clock cycles lead to an increase the energy consumption. To mitigate these issues, memristor-enabled in-memory concept is believed to be a viable solution. The memristor stores and processes data at the same location. One approach is to compute digital logic on a memristor crossbar array [86], [87]. The other approach is to use the crossbar for analog matrix multiplications [14]. Of these two approaches, the second approach is most suitable for data-intensive applications, as matrix multiplications are the most used operations in these applications [88].

Pearson correlation coefficient (PCC) is one of the data intensive computation tasks that is being used across multiple disciplines. Computing PCC on large amounts of data is computationally expensive [89]. Efforts have been devoted to speed up the PCC, which involve a General Processing Unit (GPU) implementation and Intel Xeon Phi cluster implementation [89], [90]. GPU implementation (GPU-PCC) computed PCC on a large Functional Magnetic Resonance Imaging (fMRI) dataset, while Intel Xeon implementation computes PCC to construct gene co-expression networks [89], [90]. GPU-PCC demonstrated 94.62× times speedup when compared with the CPU version, whereas Xeon cluster was found to be up to 218.2× times faster [89], [90]. Though these implementations showed significant speedup due to increased parallelization, they are still limited by the von Neumann bottleneck, which drives up their energy consumption. Therefore, there is a further need to explore non-Von Neumann architectures, which might result in lower energy consumption and also improve in gaining higher speed.

In this chapter, a systematic investigation has been initiated to design a memristor crossbar-based architecture to compute the PCC within the memory array. The crossbar has been used to perform multiplication and difference operations, where the product term in the numerator (Nr) and the difference operations in the denominator (Dr) are considered. Three different applications were demonstrated by computing PCC. The first application is computing PCC between noisy and denoised electrocardiogram (ECG) signals. The second one is face recognition, where we compute PCC between face images in the presence of occlusions and varying expressions. Finally, the third application is computing PCC between two models of H1N1 prediction to verify their similarities. Furthermore, an effort has been paid to analyze the effect of device variations on these applications. An attempt has also been made to perform detailed system level comparisons of the proposed architecture against a Von Neumann machine.

To compute PCC, the computing correlation parameter in memory (CoCoPIM) offers a few advantages when compared with a von Neumann machine. First, apart from alleviating the data transfer bottleneck as mentioned earlier, unlike von Neumann machines, the CoCoPIM computes PCC of all the input operands concurrently. This parallel task is beneficial for such applications where PCC would be computed multiple times. For example, in face recognition, PCC may need to be computed a few times to find a match. However, to perform the same task, the input operands also need to be loaded a few times in a von Neumann machine. As the operations such as square, square root, multiplication, and division (needed by PCC) require multiple cycles to complete on state-of-the art processors [91], [92]. CoCoPIM uses analog circuits to implement these operations so that these can be completed in a much shorter time. In applications such as ECG signal comparison and H1N1 model comparison, the PCC rather needs to be computed only once between two operands. Thus, the first advantage is not reflected in this case. However, the second advantage could be beneficial for improving speed up and energy savings in these applications. It is noteworthy to mention that most applications involve computation of PCC between multiple operands, and therefore both the advantages of CoCoPIM can be realized [93]–[96].

Although memristor-based Processing-In-Memory (PIM) accelerators such as In-Situ Analog Arithmetic in Crossbar (ISAAC), PIM architecture (PRIME), and reconfigurable ReRAM-based PIM (Re2PIM) have been proposed earlier, their data mapping and peripheral circuitry were specifically designed to accelerate deep learning workloads [97]–[100]. Many of the intensive arithmetic operations such as division and square root needed to compute PCC are not supported by the above-mentioned accelerators as their peripheral circuitry only computes operations (say sigmoid function) which are needed by the neural networks. On the other hand, CoCoPIM is the first accelerator for PCC which not only includes peripheral circuitry designed particularly for optimizing the throughput of these complex arithmetic operations needed to compute the PCC but also maps the input operands to exploit the parallelism in computing PCC.

In the literature, there are three other popular memristor based PIM architectures such as spin-transfer torque compute in-memory (Stt-CIM), MRAM-based in-memory accelerator (MRIMA), and FeFET Cim [101]–[103] which are also available. In this regard, CoCoPIM is considered as an application specific architecture, while the other three are programmable general-purpose accelerators. These three accelerators can also be integrated on a system on chip (SoC) by connecting them to a system bus similar to CoCoPIM. These three accelerators implement Boolean operations in memory, whereas CoCoPIM implements matrix multiplication in crossbars and complex arithmetic operations in the analog circuits. If one implements PCC based on these three accelerators, the overall performance will be degraded. CoCoPIM performs complex arithmetic operations using analog circuits asynchronously. When these three accelerators are concerned, it will take many instructions across multiple cycles as these complex arithmetic operations should be dilapidated into simple Boolean logic operations. In addition, the operands for these complex arithmetic operations should be aligned within the memory location for these three accelerators to perform their Boolean operations. Therefore, as a result, the memory array will have very low data reuse, and given the high write latency of memristors, the overall system will experience a huge latency and increase in energy.

The key highlights of this work are:

1) PCC Computing In-memory: memristor crossbars were employed to perform matrix multiplication to parallelize the proposed applications.

2) Throughput-optimized Data mapping and Peripheral Circuit design: the inputs to the memristor crossbar are mapped in such a way that the PCC can be computed across inputs concurrently. Attempts were also made to design the peripheral circuitry to support this throughput and minimize static power and area overheads.

3) Demonstrating three new applications: as mentioned earlier, three different applications,

such as implementing ECG signal comparison, face Recognition, and H1N1 model comparison, were developed based on the proposed architecture.

4) Variation Analysis: process variations were included in the simulations, and the efficacy of these variations on the mentioned applications were analyzed.

5) System-level Evaluation: we analyze the energy and delay of the proposed architecture for all three applications and compare it against a von Neumann machine to understand its potential for future-generation computing applications.

In the next section, the Pearson Correlation Coefficient parameter and how it is used to compute similarities between ECG signals, faces, and H1N1 models is described. In Section 2.3, the current-voltage characteristics of the memristor and the methodology to store data in it is explained. In Section 2.4, CoCoPIM's architecture, data mapping, acceleration steps, and microarchitecture of CoCoPIM architechture is explained. The applications are demonstrated in Section 2.5. The system-level comparisons is presented in Section 2.6. Section 2.7 summarizes the chapter.

2.2 Pearson Correlation Coefficient

The Pearson Correlation Coefficient (PCC) quantifies the similarity between two linearly dependent variables [89]. The PCC of two vectors *x* and *y* is expressed in Eqn. (2.1):

$$\rho_{xy} = \frac{\sum_{i=1}^{T}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{T}(x_i-\bar{x})^2\sum_{i=1}^{T}(y_i-\bar{y})^2}} \qquad (2.1)$$

where ρ_{xy} is the PCC between two *T* dimensional variables *x* and y. If the PCC is between 0.90 and 1, then both variables will have a perfect linear relationship [104]. However, they will not possess a linear relationship if the PCC is kept between 0 and 0.30 [104]. It is noteworthy that one may not conclude a confident decision about the relationship if the PCC lies between 0.30

and 0.90. PCC has many applications in different fields, such as medical research, financial market analysis, biometrics, etc. It is often used to find biomarkers, which are the gene expressions present in most cancer patients[94], [98]. In the area of speech-language pathology, PCC is used as an automatic speech processing system to compare the patient's speech and dysarthric speech to conclude if the patient suffers from dysarthria or not [95]. Another important application of PCC is to verify the efficacy of denoising algorithms. While collecting Electrocardiogram (ECG) data, the original signal gets corrupted due to noise from the environment. Therefore, there is a need to denoise those noisy signals, and this is accomplished using different techniques such as Discrete Wavelet Transform (DWT), Discrete Cosine Transform (DCT), etc. After denoising, the PCC is employed to compute the similarity check if the denoised signal resembles the original signal [105]. The PCC is also utilized for pattern recognition to evaluate the similarity between the neutral faces and occluded faces [106]. When financial markets are concerned, the PCC is used to check the similarity between the price of a stock (over a period of time) and the emotions of stock traders recorded (from social media) from a certain period before the period of analysis of the stock [107], which is then extrapolated to predict the price of the stock later.

2.3 Implementation Section

2.3.1 Memristor Modeling and Design Parameters

In this chapter, as mentioned earlier copper-doped zinc oxide (Cu:ZnO) based memristor was employed, and the device fabrications and characterizations are described in [50], [104]. Since the developed memristor is voltage controlled and exhibits non-linear current-voltage characteristics, the Voltage Threshold Adaptive Memristor (VTEAM) model has been utilized to model the device and import it into the simulation environment [53]. The parameters of the VTEAM model and their values for modeling the Cu:ZnO device are tabulated in Table 2.1.

The memristor's electrical characteristics deviate from its ideal behavior due to critical process variations such as Oxide Thickness Fluctuations (OTFs), Random Discrete Dopants (RDDs), and Line Edge Roughnesses (LERs) [48].

Table 2.1: VTEAM values for Cu:Zno memristor [51]

Parameter	**Value**
LRS Resistance (Ron)	1.2 kΩ
HRS Resistance (Roff)	1.2 MΩ
Ion Mobility in SET (kon)	250 nm/sec
Ion Mobility in RESET (koff)	200 nm/sec
Filament Physical Length (D)	10 nm
SET voltage (von)	1.35 V
RESET voltage (voff)	-1.20 V

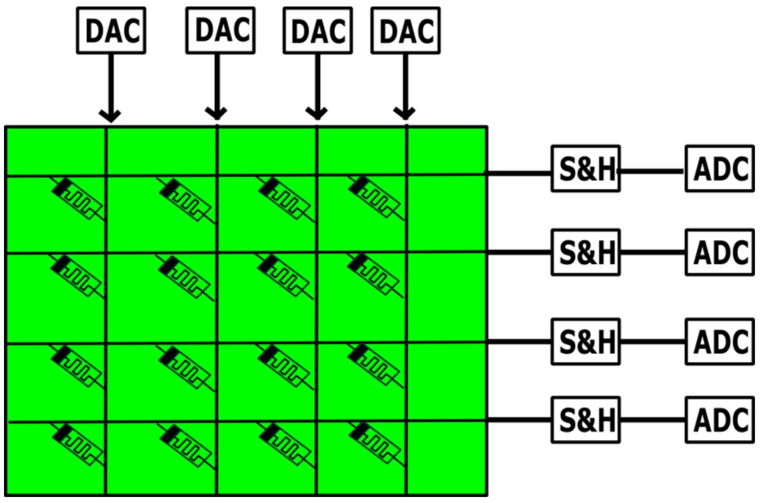


Fig. 2.1: Performing matrix multiplication in the crossbar using DAC, S&H, and ADC.

Oxide thickness is referred by the parameter D in the VTEAM model, and the deviation in dopants leads to the variations in high resistive state (HRS), low resistive state (LRS), SET voltage, RESET voltage, ion mobility in SET, and ion mobility in RESET. To model OTFs and RDDs, a normal distribution of Ron, Roff, kon, koff, von, voff, and D across the devices was introduced, where the standard deviation is 5% of the mean values of these parameters. Memristor crossbar arrays have been used to implement matrix multiplications [98], [100]. As shown in Fig. 2.1, every Bit Line (BL) of the memristor crossbar is connected to a Digital to Analog Converter (DAC), and every Word Line (WL) is connected to a Sample and Hold (S&H) circuit. Every BL is connected to every WL by a memristor. If one wants to multiply two matrices, then one matrix is initialized in the form of conductance of the memristors while the other is supplied as pulse width modulated voltage signals through the DACs. Hence, the electronic charge (accumulated in the S&H circuits at the end of every WL) represents the multiplication of both matrices. The LER arises from random uncertainties in the processes of lithography and etching [48]. It leads to random deviations in line edge print-images from their ideal patterns [108]. Hence, we add a 5% area overhead in our crossbar to account for the possible increase in the area due to LERs. However, according to the VTEAM model, the electrical characteristic of the memristor is independent of the edge length. Hence, one can conclude that LERs affect only the area of a device and not the electrical characteristics of the memristor. To analyze the effect of these process variations on CoCoPIM, these variations were included in the system-level simulations and reported the deviations observed in the proposed applications in Section 2.5.

2.4 CoCoPIM Architecture

2.4.1 Architecture

In this work, the system-on-chip (SoC) contains a processor, CoCoPIM, and main memory connected by a system bus, as shown in Fig. 2.2(a). The CoCoPIM is designed to be a Pearson Correlation Coefficient (PCC) accelerator with memristor crossbars to store the input operands (x and y variables) and compute PCC on them. This memory is visible to the programmer, and data can be written into it while programming the application (face recognition, H1N1, or ECG signals). This alleviates the data transfer between the main memory and processor, similar to the standard von Neumann machine. At a high level, CoCoPIM contains a switch matrix, a memristor crossbar, a switch array, and a peripheral circuit (Fig. 2.2(b)).

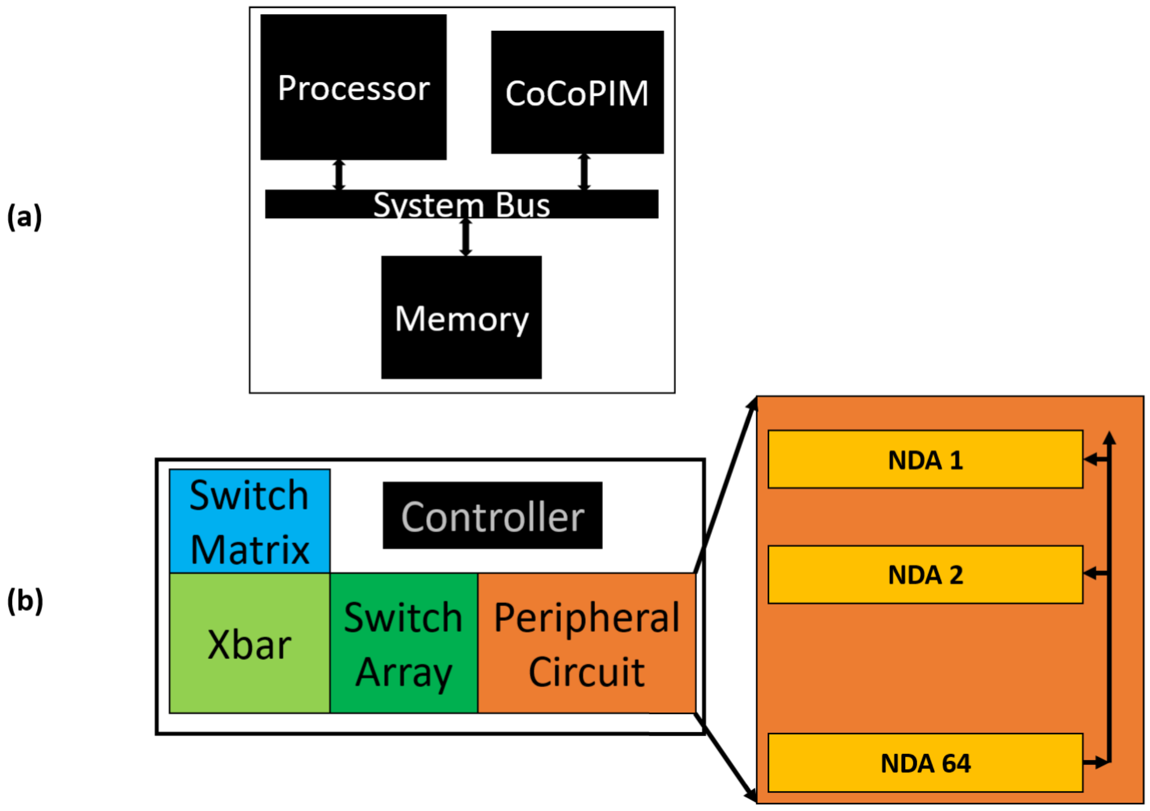


Fig. 2.2: (a) System on Chip Architecture and (b) CoCoPIM architecture.

The host CPU provides the control signals required by each of these components. On the other side, the crossbar is used to perform the matrix multiplications, while the peripheral circuit is employed to perform the other arithmetic operations required by the PCC. The peripheral circuit block, shown in Fig. 2.2(b), contains multiple blocks called Numerator, Denominator, and ADC (NDA) blocks. Where the NDA block contains circuits that compute and store the Numerator (Nr) and Denominator (Dr) of the PCC. The NDA block also contains a division and Analog-to-Digital Converter (ADC) to divide the numerator and denominator and then ultimately convert it into a digital signal ranging between 0 to 1. It is noteworthy that there are 64 NDAs inside the peripheral circuit block. The first 63 NDAs are similar, and the 64th NDA contains a different set of circuits as it needs to drive a current signal to the other 63 NDAs. This difference has been explained in Section 2.4.2. The microarchitecture of CoCoPIM is shown in Fig. 2.3. From the figure, one can notice that the switch matrix consists of transmission gates connected to the Bit Lines (BLs) of the crossbar [81] and controlled by signals B1 to B512. These control signals would decide which voltage source (V0 or V1) gets connected to the BL.

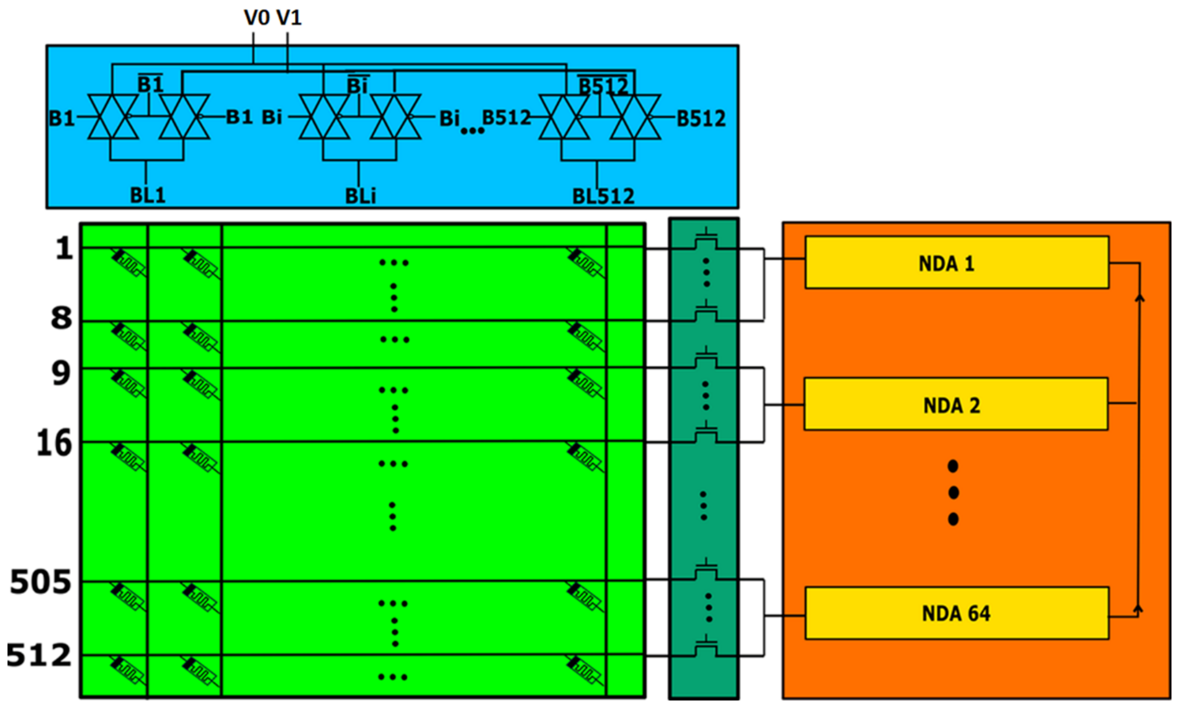


Fig. 2.3: Proposed CoCoPIM microarchitecture.

The duration of these control signals will also determine the pulse width of the voltage applied to the BLs of the crossbar. The crossbar receives input voltage signals through the BLs, and the output current signals were collected along each Word Line (WL). The NDA block consists of multiple circuits (as shown in Fig. 2.4). The main benefit of CoCoPIM over von Neumann architectures is its high throughput and latency in PCC computations.

If all the complex mathematical operations are to be performed using digital logic circuits, then one must add many ADCs to support this high throughput. As usage of ADCs increase power and area consumption significantly, it was decided to compute the complex mathematical operations using analog circuits and subsequently convert the results into digital format. The multiplication part of numerator $(x_i - \bar{x})(y_i - \bar{y})$ is computed batch wise in Xbar outside the NDA block and is given as input to miller capacitance of NDA block through switch 1(SW1). The output voltage, V2, from the Miller capacitance represents the $\sum_{i=1}^{T}(x_i - \bar{x})\,(y_i - \bar{y})$ term in the numerator of the PCC. The V2 was found to be in between 0 to 10 V. The current I5 is representative of V2 due to the transimpedance amplifier and must be in the range of nano amperes before reaching the division circuit (connected to SW3) (shown in Fig 2.4). The voltage from the miller capacitor was in positive magnitude, however, after passing through the inverting amplifier, it changed its sign magnitude into negative. In addition, initially, an inverting amplifier has been used, which divides the voltage magnitude by 1000.

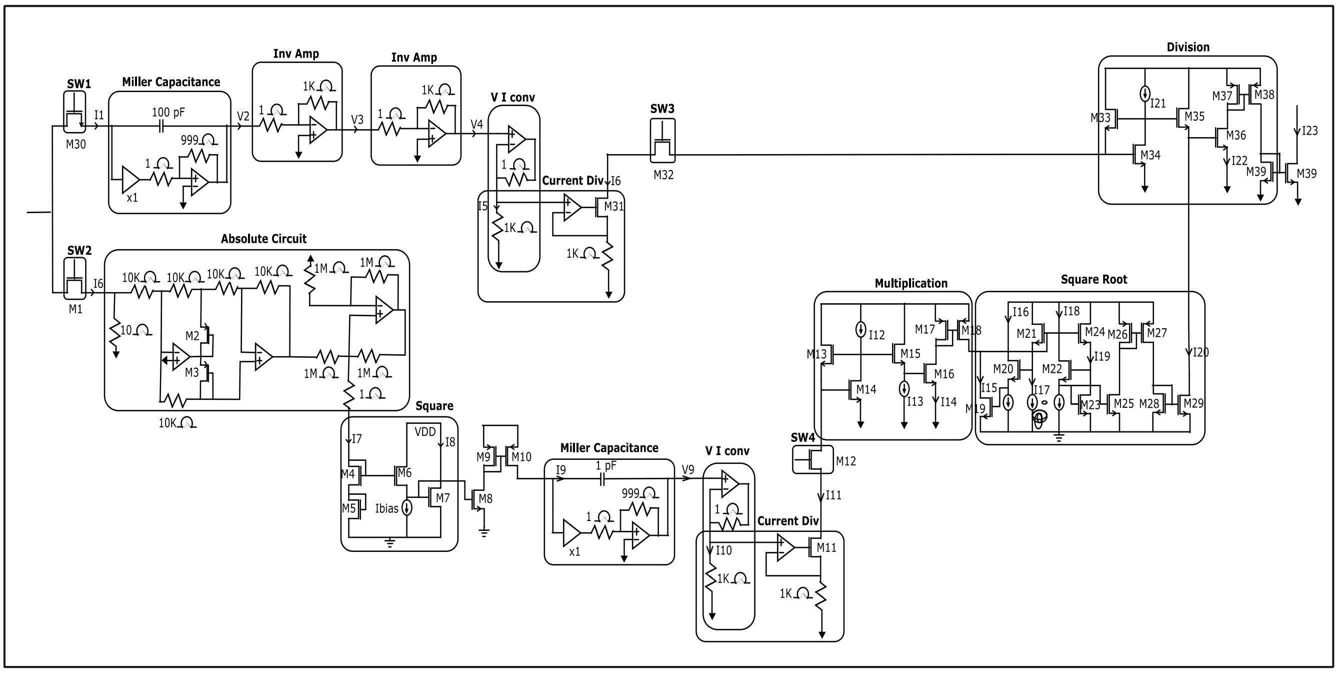


Fig. 2.4: Circuits present within each of the 63 NDA blocks.

On the other hand, the inverted voltage was passed through a VI converter circuit, which converts the input voltage into the current, while dividing its magnitude by 1000. Importantly, the output current flows through a 1 kΩ resistor, and the sign-magnitude is negative. Since the 1kΩ was also attached to a current divider, which further divides the magnitude of current by another 1000. Therefore, to adjust this, the voltage was eventually converted into current and divided by 1000 thrice to get a current in the range of Nano amperes.

The denominator was computed using the circuits connected to a switch SW2. The input current I6 ranges from -2 to 2 mA. All the circuits which were connected to SW1 are used for computing the numerator of the PCC, whereas the ones connected to SW2 are used for computing the denominator of the PCC. Once the numerator and denominator were computed, they were divided and then converted into binary digits using the ADC. The output current from the divide circuit I23 was fed into the ADC. When the SW1 was on, the current flowing through it was summed up in the Miller capacitance. A capacitance of 100 nF is required to adjust the range of the accumulated charge for further calculations. As 100 nF is not feasible to be realized on-chip, a Miller capacitance block was utilized to multiply the 100 pF capacitance by the magnitude of 1000 using the 1Ω to 999 Ω non-inverting amplifier configuration to achieve 100 nF. The crossbar size was taken as 512 × 512, and its word lines (WL) were connected to the switch array. The switch array contains 512 NMOS transistors used to connect the WLs of the crossbar to their corresponding NDA blocks. From the literature, it was found that one requires a maximum of eight rows of the crossbar to initialize one vector (in the present case, a vector is a face image or an ECG signal or an H1N1 model) [50], [92], [93]. Therefore, in Fig. 2.3, every eight rows of the crossbar were connected by the switch array to one NDA block.

The current I6 was passed through the absolute circuit to convert it into a positive current. Whereas I7 was the output of the absolute circuit, which passes through the 1Ω resistor. It is noteworthy to mention that the value of this resistance does not alter the magnitude of I7. It was observed that the I7 was the input to the square circuit, and it uses the Translinear (TL) principle to exploit transistors in the subthreshold region [109]. In the square circuit shown in

Fig. 2.4, one can notice a TL loop has been formed by transistors M1, M2, M3, and M4. The *TL* loop was expressed as follows:

$$V_{GS1} + V_{GS2} = V_{GS3} + V_{GS4} \tag{2.2}$$

where $V_{GS1}, V_{GS2}, V_{GS3}$ and V_{GS4} are the gate-source voltages of transistors M1, M2, M3 and M4 respectively. The drain current *(I_D)* in the subthreshold region of a MOS transistor is expressed below:

$$I_D = I_o . \frac{W}{L} . e^{V_{GS}/U_T} \tag{2.3}$$

where I_o is a constant, W and L are the width and length of the transistor, respectively. V_{GS} is the gate to source voltage of the transistor, and U_T is the thermal voltage. By rearranging Eqn. (2.3), the Eqn. (2.4) can be written as:

$$V_{GS} = (\ln \mathrm{I}_D - \ln W/L - \ln I_O)U_T \tag{2.4}$$

Also, by substituting Eqn. (2.4) in Eqn. (2.2), the Eqn. (2.5) can be obtained as:

$$I8 = \frac{I7^2}{Ibias} \tag{2.5}$$

where I8 is the square of the magnitude of I7. The current I8 contains a bias term in its denominator. The proposed circuit was verified, and the obtained characteristics are presented in Fig. 2.5. The input current I7, and Ibias is set at 100 nA. The output current is I8, and it matches very closely to its expected value. As shown in the Fig. 2.4, a current mirror (transistors M8 to M10) was employed to generate current I9, which has the same magnitude as I8. In the Miller capacitor, this current (I9) was summed up. A capacitance of 1 nF was required to adjust the range of the accumulated charge for further calculations. The output voltage of the Miller capacitor was V9. This voltage represents the term $\sum_{i=1}^{T}(x_i - \bar{x})^2$ in the denominator of the PCC. This must be converted into the current for further use in other arithmetic circuits. In this work, a VI converter circuit was used to convert voltage V9 into current I10.

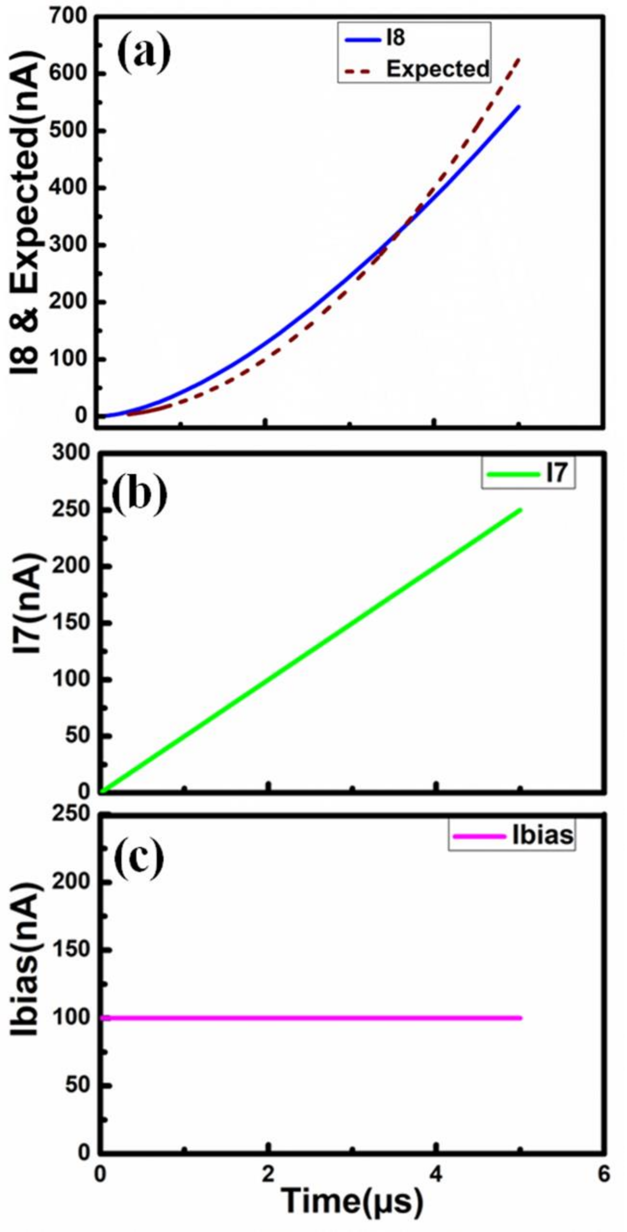


Fig. 2.5: Characteristics of the square circuit. (a) The output and expected current, (b) the input, and (c) the bias current.

The relation between I10 and V9 is given in Eqn. (2.6) as follows:

$$V9 = \frac{I10}{1K\Omega} \tag{2.6}$$

Once the V10 was converted into the current, it was further scaled up by 1000. The magnitude of the current was further reduced by using the 1kΩ resistor as the arithmetic circuits later require current to be in the nA range. Thereafter, the I10 was fed into a current divider circuit, which further fed its output current I11 into switch 4 (SW4). At this point, it was not required to further reduce the magnitude of the current I10 in the current divider as it was already in the preferred range.

The *TL* loop in this circuit is expressed by Eqn. (2.7).

$$V_{GS8} + V_{GS9} = V_{GS10} + V_{GS11} \tag{2.7}$$

where V_{GS8}, V_{GS9}, V_{GS10} and V_{GS11} are the gate-source voltages of transistors M8, M9, M10 and M11 respectively. By substituting Eqn. (2.3) in Eqn. (2.7), one gets Eqn. (2.8)

$$I11 \times I12 = I13 \times I14 \tag{2.8}$$

From Eqn. (2.8), it can be observed that the currents I11, I12, I13, and I14 are multiplied. In this case, I11 is the output current of the current divider circuit, and I12 is the output current from NDA 64.The current I13 is considered as a bias current, and I14 is the output current of the multiplication circuit, which represents the term $\sum_{i=1}^{T}(x_i - \bar{x})^2 \sum_{i=1}^{T}(y_i - \bar{y})^2$ in the denominator. Efforts were put to demonstrate the output performances, and the characteristics of this circuit are depicted in Fig. 2.6. In this figure, the bias current I13 has a magnitude of 100 nA. The output from the multiplication circuit I14 was fed to a current mirror to give the output current I15, which has the same magnitude as I14. I15 was an input to the square root circuit, which also works on the basis of *TL* principle [111].

From Fig. 2.4, one can find that the TL loop for the square root circuit is given by Eqn. (2.9)

$$V_{GS12} + V_{GS13} + V_{GS14} = V_{GS15} + V_{GS16} + V_{GS17} \tag{2.9}$$

where V_{GS12}, V_{GS13}, V_{GS14}, V_{GS15} V_{GS16} and V_{GS17} are the gate-source voltages of transistors

M12, M13, M14, M15, M16 and M17, respectively. After substituting Eqn. (2.3) in Eqn. (2.9), one gets Eqn. (2.10) as follows:

$$I15 \times I16 \times I17 = I18 \times I19 \times I19 \tag{2.10}$$

where I15, I16 (constant), I17 (constant), I18 (constant), and I19 are the currents through the transistors M19, M20, M21, M22, M23 and M24, respectively.

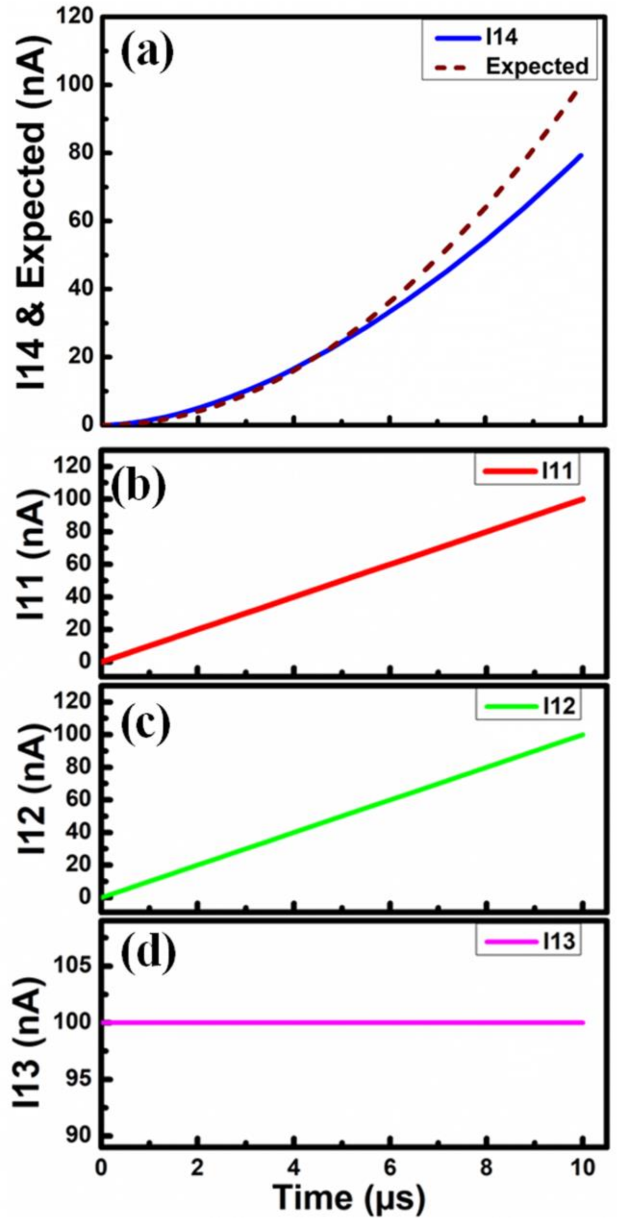


Fig. 2.6: Characteristics of the multiplication circuit. (a) The output and expected current, (b) and (c) are the inputs, and (d) the bias current.

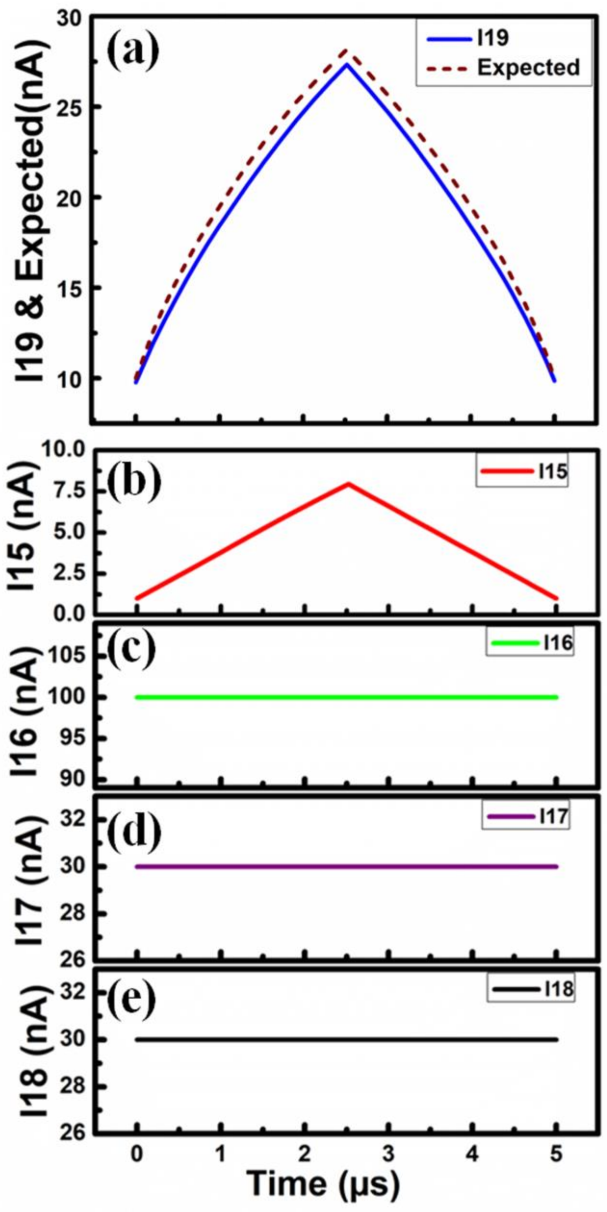


Fig. 2.7: Characteristics of the square root circuit. (a) the output current I19 and the expected output current t of inputs (b) I15, (c) I16, (d) I17, and (e) I18.

It is essential to mention that in Eqn. (2.10), the I19 has appeared twice, as the same current flows through the transistors M23 and M24. Thus, Eqn. (2.10) can be rewritten as Eqn. (2.11) as given below:

$$I19 = \sqrt{\frac{I15 \times I16 \times I17}{I18}} \tag{2.11}$$

Therefore, Eqn. (2.11) is essentially the square root of I15 and three bias currents. The output current I19 represents the term $\sqrt{\sum_{i=1}^{T}(x_i - \bar{x})^2 \sum_{i=1}^{T}(y_i - \bar{y})^2}$ in the denominator. The results from simulations of the square root circuit are shown in Fig. 2.7.

The input current I15 ranges from 20 to 80 nA and is shown in Fig. 2.7(b). The currents I16, I17, and I18, are constant current sources of magnitude 100 nA, 30 nA, -and 30 nA, respectively (Figs. 2.7(c)-(e)). It was understood that I17 and I18 would cancel each other in the division, and only I16 and I15 would remain under the root. Therefore, the square root circuit's output was found to be 10 times the square root of I15, as I16 was kept as 100 nA.

From Fig. 2.7(a), the output current I19 and the expected current overlap each other. The currents I6 and I20 represent the numerator and the denominator of the PCC, respectively. Hence, they were utilized as inputs to the division circuit. The division circuit was the same as the multiplication circuit and its *TL* and equations representing its characteristic is provided in Eqn. (2.12) and Eqn. (2.13), respectively.

$$V_{GS33} + V_{GS34} = V_{GS35} + V_{GS36} \tag{2.12}$$

$$I22 = \frac{I16 \times I21}{I20} \tag{2.13}$$

The output current of the division circuit represents the PCC. It was fed into the ADC to convert it into a digital output. The characteristics of the divide circuit are shown in Fig. 2.8. The output current and the expected output are plotted as a function of time in Fig. 2.8(a). In CoCoPIM, the NDA blocks were designed to restrict the Nr and Dr currents between 10 and 20 nA. To verify the functionality of the division circuit, in the simulation, three values 10, 15, and 20 nA were considered for I20 (in Fig. 2.8(b)); whereas I6 was found to be varied from 10 to 20 nA (in Fig. 2.8(c)), and I21 (in Fig. 2.8(d)) was found to vary as a constant at 10 nA. In Fig. 2.8(a), the expected current is given by the formula I21 × I6/I20. For all the three values of I20, one can observe that I22 deviates from the expected value by less than 1 nA, and this explains why

CoCoPIM's PCC is very close the PCC computed by software (SW). It is also important to discuss the working principle of the circuit when either of the input currents (Nr or Dr) is zero. Thus, when the Nr is zero, I22 is also zero as given by the TL principle [110]. On the other

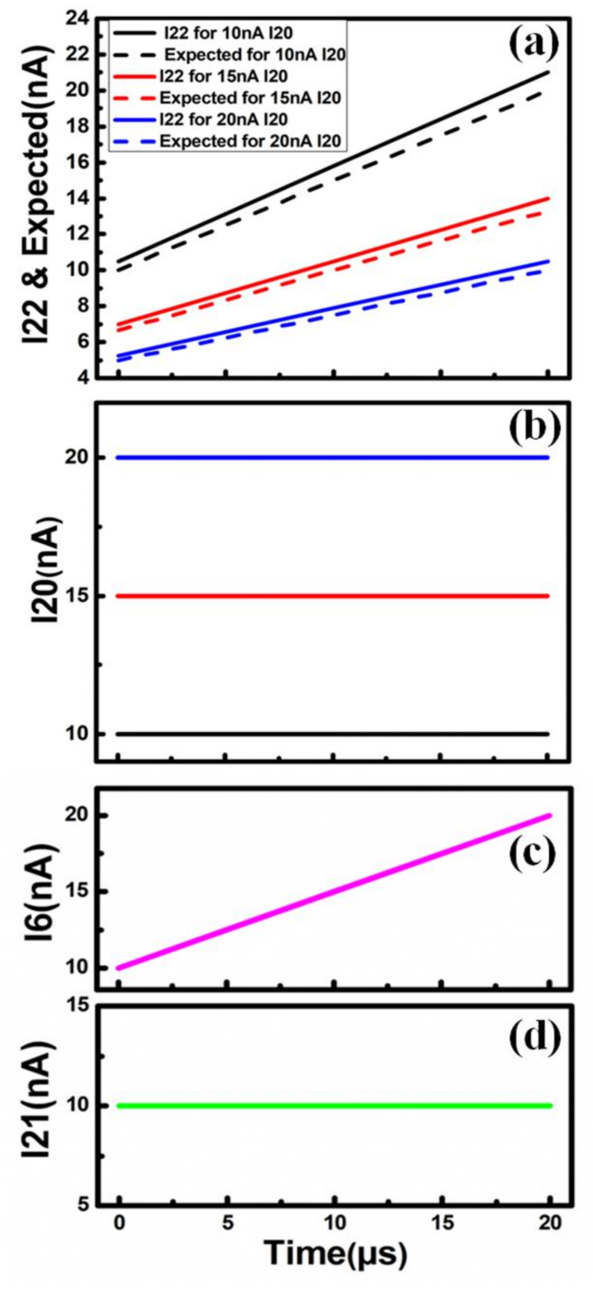


Fig. 2.8: Characteristics of the DIV: (a) output current I22, and expected Output, (b) I20, (c) I6, and (d) I21.

side, zero output current corresponds to a PCC of zero, which is essentially the expected outcome corresponding to the Nr of PCC being zero. Furthermore, when the Dr is zero, the output current was in the microamperes range indicating that the result is infinite. Interestingly, the Dr can never be zero in any dataset as it is the product of the sum of squares of the mean-shifted input operands from the dataset. NDA 64 differs from the other 63 NDAs as it is required to compute only the $\sum_{i=1}^{T}\left(y_i - \bar{y}\right)^2$ term in the Dr of the PCC. Hence, from Fig. 2.9, it was observed that NDA 64 contains the absolute circuit, square circuit, and Miller capacitance similar to other 63 NDAs.

However, the output of the Miller capacitor must be fed as input to the other 63 NDAs. Thus, it was converted into current as in the other 63 NDAs, and then current mirrors with 63 current sources so that the output from each of the current sources can be used as an input for the

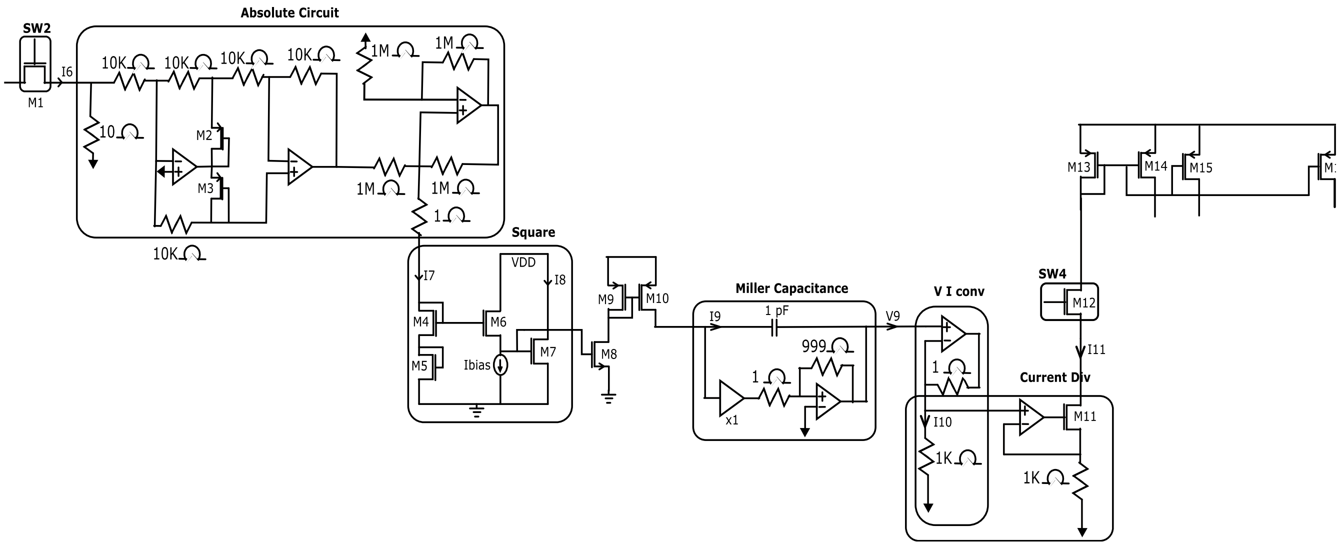


Fig. 2.9: Circuits present within the NDA 64 block. It contains absolute circuit, square circuit, and Miller capacitance.

multiplication mirrors with 63 current sources so that the output from each of the current sources can be used as an input for the multiplication circuit in each of the 63 NDAs.

2.4.2 Data Encoding and Mapping

In this work, based on the repeatability, it was restricted to employ an on/off ratio of 100 so that the device variations do not lead to the wrong value being read or written into the device.

The relation between the data that should be stored within the memristor and the conductance of the memristor are given by Eqn. (2.14) and Eqn. (2.15).

$$s = \frac{(a-LL)*100}{(UL-LL)} \tag{2.14}$$

$$g = \frac{(s)}{(100)} * \left(\left(G_{on} - G_{off}\right) + G_{off}\right) \tag{2.15}$$

In Eqn. (2.14), *a is* the data that should be stored in the memristor, *UL* and *LL* are the Upper Limit and Lower Limit of the data, respectively. *s* represents the state of the memristor. As there are 100 states in the memristor, a 100 was multiplied by the numerator of Eqn. (2.14). Hence, Eqn. (2.14) offers basically a linear relation between the state and the data, whereas Eqn. (2.15) provides a linear relation between the state and the conductance of the memristor. *g* represents the conductance of the memristor. Using Eqn. (2.14) and Eqn. (2.15), each element of the *x* and *y* vectors was encoded into the memristor in the form of conductance. After *g* is computed, the write operation was performed using the Switch Matrix. The pulse width of the write signal was provided by the CPU as the signals (B1 to B512) to the switch matrix, and the value of this signal was calculated according to the VTEAM model [53]. The elements of the *y* variable were encoded as the voltage signals and used for calculating the numerator. The relation between the data and the pulse width is given by Eqn. (2.14) and Eqn. (2.16). Using Eqn. (2.14), the state (between 0 to 100) corresponding to the data was found, and then the pulse width is calculated as follows:

$$t = s * t_0 \tag{2.16}$$

In Eqn. (2.16), t_0 is a constant value of 10 ns, which is basically the read time of the memristor [53]. The PCC should be computed between two *T* dimensional variables *x* and *y*, where *x* will have multiple instances. For example, in the face recognition task, one should compare one test image (*y*) with 15 known images ($x_1, x_2, \ldots, x_{15}$). In this application, each of the 15 images was an instance of *x*. The size of the Analog Switch Array (ASA) imposes a limit on the number of rows available per instance of *x*. As the switch array has a dimension of 8×8, the number of rows per instance of *x* was restricted to 8. As shown in Fig. 2.10, all instances of *x* were stored in the crossbar. For the first instance x_1, the elements were stored from the first column to the 511th column. In the 512th column, the average of the corresponding instance was stored, and the 512th to 1022nd elements of x_1 were stored in the second row. In a similar way, the first 511 elements of x_2 were stored in the 9th row, and the 10th row contains the 512th to 1022nd elements of x_2. On the other side, the entire memristor crossbar was divided into groups of 8 rows each, and one instance of *x* was written in that group. In the last 8 rows (505-512) of the memristor crossbar, *y* was stored. From the literature, it was found that very few applications required a T greater than 4088 [50], [89]–[94], [106]. Hence, 8 rows of the memristors were connected to one NDA block. If one intends to modify CoCoPIM for T greater than 4088, then more than eight rows of the crossbar should be connected to one NDA. This type of connection requires a much larger switch array, and we have left this important work to continue as a part of our future research.

2.4.3 Computation Steps

CoCoPIM computes the PCC in two steps. In the first step, the numerator in Eqn. (2.1) is computed, followed by the second step, wherein the denominator in Eqn. (2.1) is computed. The numerator in Eqn. (2.1) is expanded, as shown in Eqn. (2.17) below.

$$\sum_{i=1}^{T}(x_i - \bar{x})\,(y_i - \bar{y}) = \sum_{i=1}^{T} x_i * (y_i - \bar{y}) - \bar{x} * (y_i - \bar{y}) \tag{2.17}$$

The first term on the right-hand side of Eqn. (2.17) was obtained by supplying the $(y_i - \bar{y})$ the term as voltage pulses (Fig. 2.10 (a)) along the first 511 columns of the memristor crossbar, while the x_i terms are stored in the crossbar. In this case, the pulse width is proportional to the magnitude of $(y_i - \bar{y})$ and the voltage magnitude is either 0.50 or -0.50, depending on the sign magnitude of $(y_i - \bar{y})$. The second term was obtained by supplying a voltage pulse to the 512th column, which has a time period proportional to the sum of $(y_i - \bar{y})$; where i range from 1 to 511. The magnitude of this voltage pulse is either 0.50 or -0.50, depending on the sign-magnitude of the corresponding term $(\sum_{i=1}^{T}(y_i - \bar{y}))$.

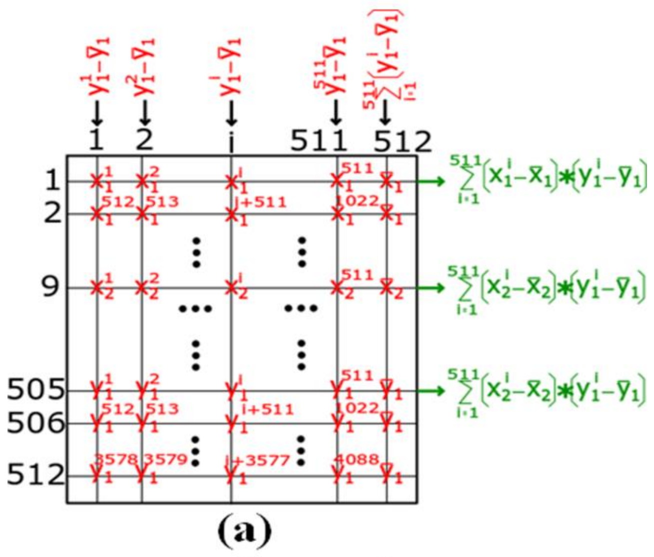
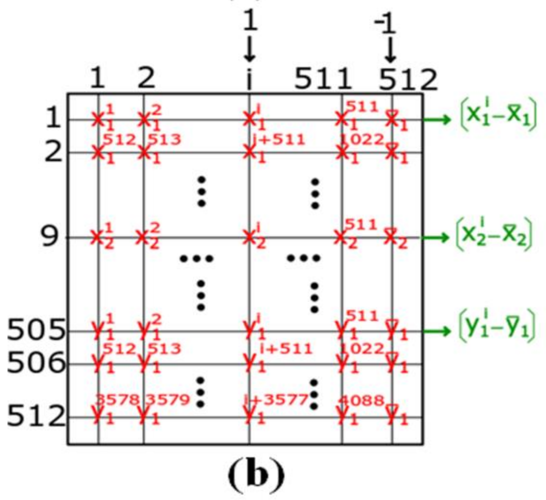



Fig. 2.10: Data Mapping and Computation in CoCoPIM. (a) Numerator Computation, and (b) Denominator computation.

In this regard, the switch array decides which row's output must be connected to the NDA block for further processing. In Fig. 2.10(a), the output from the rows of 1st, 9th, etc., were transmitted to the NDA block by the ASA. These outputs are a part of the numerator terms of the PCC between y and x_1, x_2, etc. In the next cycle, the voltage pulses were applied to the columns corresponding to elements 512 to 1022 of the y variable.

In this cycle, the outputs from rows 2nd, 10th, etc., were connected to the NDA block by the ASA, and all the other rows' outputs were avoided. It is noteworthy that when the numerator is being computed, the last eight rows, which store the variable y were not considered, and the computation of the denominator is shown in Fig. 2.10(b). In Eqn. (2.1), one can observe that the denominator has two terms involving $(x_i - \bar{x})$ and $(y_i - \bar{y})$. Each of these terms requires many operations, such as square, sum, square root, and multiplication. However, for these operations to be performed, one must first obtain $(x_i - \bar{x})$ and $(y_i - \bar{y})$ from the memristor crossbar. To achieve this, +1V was applied to the ith column, and -1V was employed to the 512th column. This results in an output of $(x_i - \bar{x})$ in the rows of 1st, 9th, etc.,. This operation was performed for every column from 1 to 511, and the outputs of the 1st, 9th, etc. rows were connected to the NDA block. Thereafter, the same operation was repeated; however, different rows' outputs, such as 2nd, 10th, etc., were linked to the NDA block. This operation was continued until all the T terms of $(x_i - \bar{x})$ and $(y_i - \bar{y})$ were obtained. In this case, the last 8 rows of the memristor crossbar were connected to get the terms of $(y_i - \bar{y})$.

2.5 Applications

2.5.1 Face Recognition

Face Recognition is ubiquitous in modern-day life. It is used to track attendance, identification of criminals, prevent fraud voters, fraud detection of visas, etc., [106]. Due to the pervasiveness of Face Recognition, a memristor-based solution is adopted to accelerate its progress. In this work, the Yale Face dataset was utilized, which has images from 15 subjects recorded in the presence of occlusion and varying expressions [93]. One of those 15 subjects is depicted in Fig. 2.11. Fig. 2.11(a) is the normal image, while Fig. 2.11(b) and Fig. 2.11(c) represent the images with occlusion (glass) and facial expression variation (happiness), respectively.

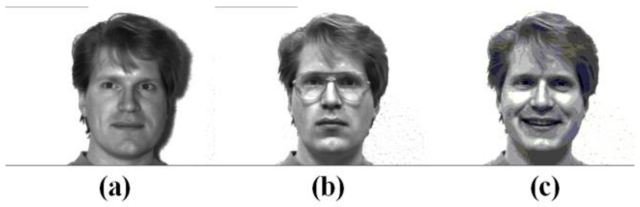


Fig. 2.11: Images of one of the 15 subjects from the Yale dataset. (a) Normal image, (b) Image with Occlusion, and (c) Image with variant expression.

The PCC is computed for the variant expression and occlusion-present image with the normal image as two separate tasks. In the occlusion task, attempts were made to compute the PCC between each of the 15 normal images and each of the 15 occlusion present images. To recognize a face correctly, the PCC between a particular occlusion-present image and its corresponding normal image should be greater than the PCC between the same occlusion-

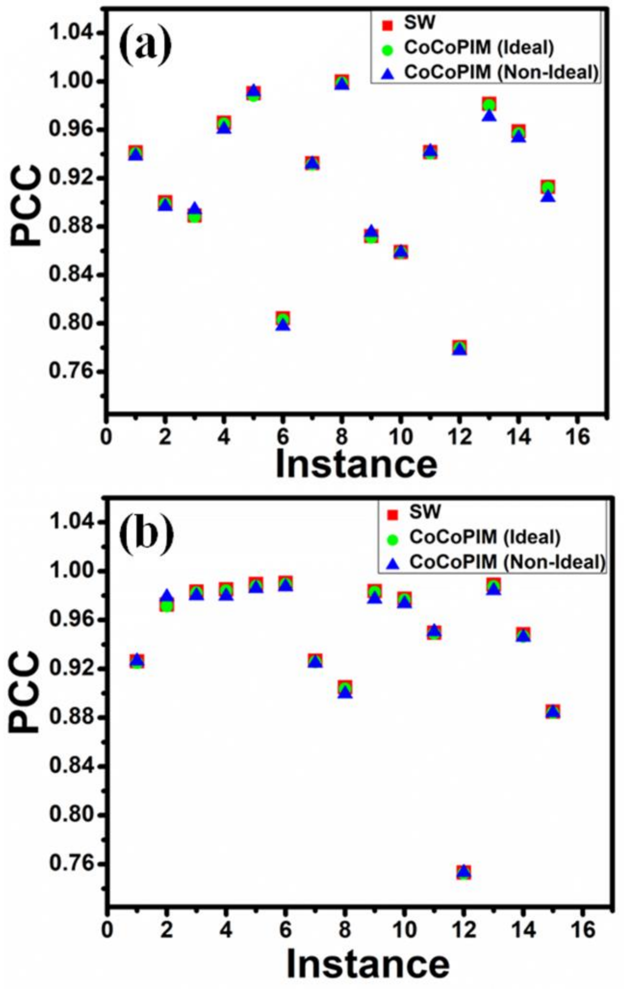


Fig. 2.12: Recognition accuracy of the data obtained from Software, CoCoPIM and CoCoPIM with variations in the (a) Occlusion task, and (b) Varying expression task.

present image and the other normal images. The variant expression task has also been implemented using the same condition.

The normal images are the *x* variable, which were initialized in CoCoPIM's crossbar, while the images with occlusion are the *y* variable. Since *y* can represent only one image, the PCC between the 15 occlusion-present images and the 15 normal images were computed in 15 cycles. In each cycle, *y* was updated to represent each of the 15 occlusion-present images. Similarly, in the variant expression task, *y* represents the images with varying expression. Each normal image contains 1600 pixels, and each instance of *x* is stored using four rows of the crossbar connected to a dedicated NDA through the switch array. The results obtained from both these tasks are shown in Fig. 2.12. The PCC computed using standard software (MATLAB) is denoted by (standard software) SW along with the ideal CoCoPIM and non-ideal (including device variations) CoCoPIM PCC results. In Fig. 2.12(a), the occlusion task's PCC is depicted and Fig. 2.12(b) consists of the PCC results from the variant expression task. From both these results, one can observe that CoCoPIM follows the SW output very closely despite the device variations.

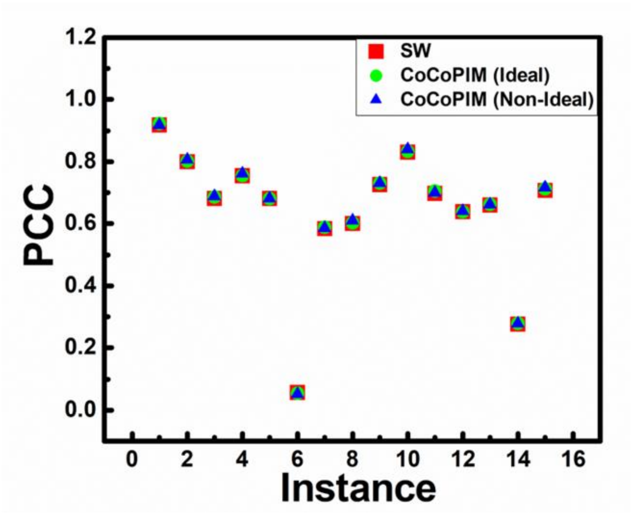


Fig. 2.13: Comparisons of PCC computed using CoCoPIM (ideal), CoCoPIM (non-ideal) and the standard software (SW). PCC between a non-occluded non-varying expression image with 15 occluded images.

To verify whether the PCC originated from CoCoPIM's ideal and nonideal implementation matches with the SW implementation, we have computed 15 PCCs which range from 0 to 1 (in Fig. 2.13). The first PCC is between a facial image (without occlusion and varying expression) and an image of the same face with occlusion. The 14 other PCCs are between the first image and 14 other faces with occlusion. The first PCC is the highest one as its operands have many similar features, while the 14 other PCCs are lower than the first one as they have significantly different features between their opernads. From Fig. 2.13, one can note that the computed PCC ranges from 0 upto 1 and that CoCoPIM's ideal implementation and nonideal implementation are very close to SW implementation.

2.5.2 ECG Signal Comparison

The PCC has innumerable uses in the field of biomedical research. One such example is the comparison of the original ECG signal and the denoised ECG signal [105]. The original ECG signal is noisy and has to be filtered for better diagnosis. Cook et al. [112] reported the PCC between 48 original signals and DWT-filtered clean signals, and MIT-BIH database was used for original signals. In this work, the original ECG signal is filtered using the same DWT method and on the same database to generate clean signal [101]. Thereafter, the PCC between the original and clean signals was computed using MATLAB and CoCoPIM. The MIT-BIH database consists of 48 ECG signals.

In CoCoPIM's crossbar, the original signals are initialized as x variable and the denoised ECG signals as y variable. Since y can represent only one signal, the whole task was computed in 48 cycles. In each cycle, the variable y was loaded with one of the 48 denoised signals. Each ECG signal has 3600 values recorded against time. Therefore, each of the 48 instances of x was initialized using eight rows of the crossbar. The last eight rows of the crossbar were initialized with y.

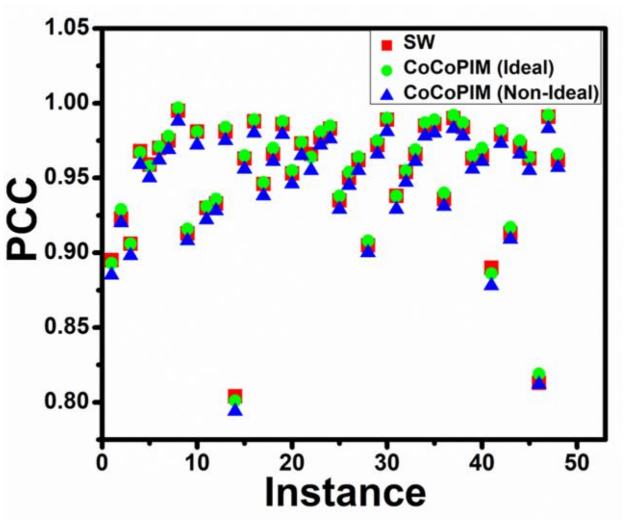


Fig. 2.14: Comparisons of PCC computed using CoCoPIM (ideal), CoCoPIM (non-ideal) and the standard software (SW) for the ECG dataset.

The PCC results obtained using the conventional SW (ideal and nonideal CoCoPIM) are depicted in Fig. 2.14. From Fig. 2.14, one can note that the ideal and nonideal CoCoPIM values are not significantly deviated as compared with the values obtained from SW.

2.5.3 H1N1 Model Comparison

Google Flu Trends (GFT) uses internet search activity to model outpatient Influenza-like Illness (ILI) to provide near-real time estimates of influenza activity. GFT estimates have exhibited a strong correlation with official influenza (H1N1) surveillance data obtained from the U.S. Centre for Disease Control (CDC) and Prevention [112]. In one of the studies [112], PCC is used to check how strongly the GFT estimates correlate with the official data. In this work, PCC is computed between the GFT model and the CDC ILI data using standard SW and CoCoPIM.

GFT and ILI data contain weekly observations from March to December of 2009 as shown in Fig. 2.15. GFT data are initialized in the crossbar as the x variable, while ILI data are initialized as the y variable. PCC computed by the SW, ideal CoCoPIM, and nonideal CoCoPIM was found to be 0.988, 0.982, and 0.979, respectively.

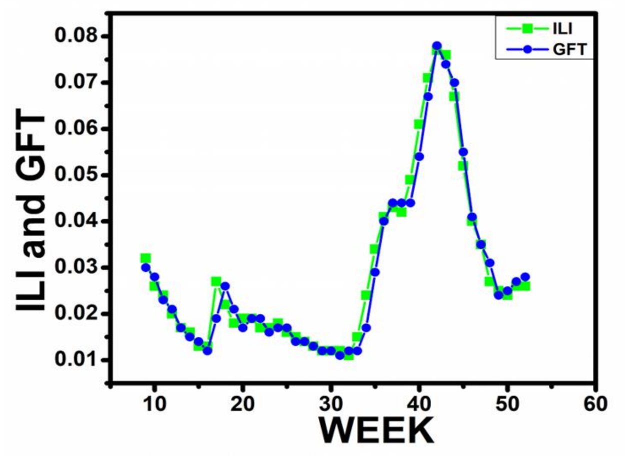


Fig. 2.15: Estimations of GFT and ILI for 55 weeks.

In all these three applications, it is noteworthy to mention that the ideal CoCoPIM's PCC values do not deviate significantly from that of nonideal CoCoPIM. This proves that the process variations do not significantly affect the accuracy of computation. We believe that this is due to the very high ratio of HRS to LRS of our Cu:ZnO device[50]. As the obtained data were mapped into a very large range of resistance, the process variations did not lead to coalescing of states. The MOSFETs were acquired from the 180-nm Taiwan Semiconductor Manufacturing Company (TSMC) library. The memristor has a width (D) of 10 nm which is significantly less than the minimum gate length of the CMOS technology node. Hence, we believe process variations to be more significant in memristor than in MOSFETs. Thus, only memristor's variations were included as part of nonideal CoCoPIM simulation.

2.5.4 System-Level Variation Analysis

The variations in the analog circuit and memristor were included to simulate 100 non-ideal instances of CoCoPIM. In this regard, the mean square error (MSE) between the non-ideal and software implementations was calculated for the 100 non-ideal instances and their mean values were subsequently used in all three applications. The average of the 100 MSEs was found to be 0.062, 0.093, and 0.012 for the face recognition, ECG signal comparison, and H1N1 model comparison, respectively. Interestingly, it was observed that when the MSE increases, the size of a variable (operand) also follows the same trend. The number of states in our framework has been modified to 25 and 50. When 25 states per memristor are concerned, the MSE w.r.t. 100 states was found to be 0.0095; however, for the 50 states per memristor, it was 0.00042. As a matter of fact, one can note that the MSE gets decreased with the increase in the number of states per memristor. It was also found that even though the number of states decreases, the CoCoPIM's PCC values are not deviated significantly from the ideal simulations.

2.6 Results and Discussion

2.6.1 System Level Analysis

2.6.1.1 Simulation Environment

Neurosim is a device-to-algorithm framework that was designed to benchmark memristor crossbar array-based architectures[14], [81]. An effort was made to modify the Neurosim to evaluate CoCoPIM's delay and energy consumption for the different applications.

Table 2.2: Von Neumann machine specifications

Frequency	3.40 GHz
Processor	4 out-of-order cores
L1 I/D Caches	64 KB, 8-way, private
L2 I/D Caches	256 KB, 8-way, private
Coherence Protocol	MESI
Memory	DDR4 - 1600, 4GB

In this work, the CMOS circuits were designed in SPICE using the 180-nm TSMC process. To benchmark CoCoPIM against a von Neumann machine, system simulator (MARSS) and multicore power, area, and timing (McPAT) were used [82], [83]. MARSS is a cycle-accurate simulator of multicore x86 processors, whereas McPAT is a power, area, and timing modeling framework of multicore and manycore systems. The specifications of the von Neumann Machine are tabulated in Table 2.2.

2.6.1.2 Energy and Delay

The initial state of the memristor crossbar (within CoCoPIM) contains all memristors initialized to HRS. Thereafter, the operands were written into the crossbar. The write latency and energy have been included in energy and delay calculations of CoCoPIM followed by computations of PCC in the peripheral circuitry.

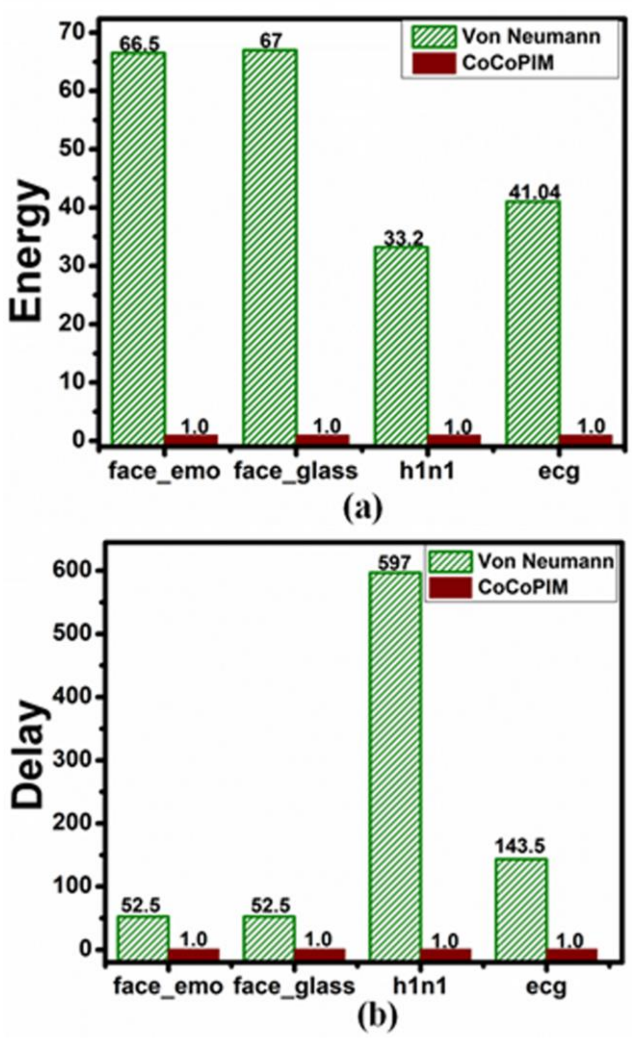


Fig. 2.16: (a) Energy of Von Neumann machine normalized to CoCoPIM, and (b) Delay of Von Neumann machine normalized to CoCoPIM.

The time and energy measurements continue unless the NDA blocks complete in performing their respective arithmetic operations. Fig. 2.16 shows comparison of the energy and delay of the baseline von Neumann machine against CoCoPIM. The energy and delay were normalized w.r.t. CoCoPIM. In Fig. 2.16(a) and 2.16(b), the first two tasks face_emo and face_glass are the face recognition tasks in the presence of varying emotions and occlusion, respectively. In the face_emo task, CoCoPIM was found to be about 66.5× and 52.5× times better than von Neumann machine's energy and delay, respectively. Similarly, in the face_glass task, CoCoPIM was found to be about 67× and 52.5× times better than von Neumann machine's energy and delay, respectively. On the other hand, the third task H1N1 computes PCC between its two models. In this task, CoCoPIM was found to be about 33.2× and 597× times better than von Neumann machine's energy and delay, respectively.

Similarly, in the fourth task, CoCoPIM computes the PCC between 48 ECG signals, and it was found to be better than von Neumann machine by 41.04× and 143.5× times improvement in energy and delay, respectively. Out of these four tasks, although H1N1 has the highest advancement in delay, it ends up in providing lowest improvement in energy. This is because of the small size of the H1N1 dataset and the higher parallelization of CoCoPIM when compared with the von Neumann machine. However, opamps and capacitors also contribute significantly to its energy consumption, and therefore it turned out to be the least energy-efficient than the other tasks.

2.6.1.3 Area Estimation

The area estimates of each component of CoCoPIM are presented in Table 2.3. The numerator includes all the circuits connected between (and including) SW1 and SW3 (Fig. 2.4), while the denominator includes all the circuits between (and including) SW2 and the division circuits. The peripheral circuit consists of the numerator, denominator, and ADCs, which occupy 615,055 μm2 or about 98% of the total area. In the crossbar's area, a 5% overhead has been considered, arising from the LERs, as mentioned in Section 2.3.

Table 2.3: CoCoPIM area estimation

Component	Area(μm2)
Switch Matrix	2433
Xbar	4246
Switch Array	16
Peripheral Circuit	615055
Numerator	34965
Denominator	41440
ADC	538650

2.7 Summary

In this work, we present a memristor crossbar-based architecture CoCoPIM, which is found to be 41.04×, 66.5×, 67×, and 33× times energy efficient against a von Neumann machine in computing PCC of ECG signals, Face Recognition, and H1N1 data sets, respectively. It also achieved a speedup of 143.5×, 52.5×, 52.5×, and 597× against the same von Neumann machine in the respective tasks. It has also been established that CoCoPIM does not deviate significantly even in the presence of process variations, which offers to realize a highly reliable computer architecture application. The proposed idea can also be extended to a much larger switch array.

Remarks: Below mentioned paper was published based on this chapter

- Souvik Kundu, **Priyanka B. Ganganaik,** Jeffry Louis, Hemanth Chalamalasetty, and BVVSN Prabhakar Rao. "Memristors Enabled Computing Correlation Parameter In-Memory System: A Potential Alternative to Von Neumann Architecture." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 30, no. 6 (2022): 755-768.

Chapter-3

Memristor-Based In-Memory Processor for High Precision Semantic Text Classification

3.1 Introduction

The limitations of Von Neumann architecture of currently used computers are discussed in the chapters 1 and 2 . This has resulted in the necessity for in-memory computing, in which data flows are controlled, so circumventing the drawbacks of Von Neumann architecture. In this regard, Memristor is proven to be effective emerging device for in-memory processing. It has found applications in analog computing, neuromorphic circuits, pseudorandom number generation, and non-volatile memory. The most profound application of a memristive crossbar is a vector product accelerator of linear time complexity [113], [114]. The same operation in a conventional computer is of quadratic time complexity and is a memory-intensive task. The efficiency of the memristive crossbar as an in-memory processor is due to its reduced data movement and parallel architecture. Despite the advantages of in-memory memristive crossbar arrays, there are research works highlighting its computational inaccuracy [115], [116]. The most prominent source of error in the crossbar is the analog – to – digital (AD) / digital – to – analog (DA) domain conversion.

In this chapter, the main aim is to develop a memristor-based in-memory processor for Bayesian inference, which is employed for high precision semantic text classification. Memristive crossbar-based computations are usually performed in the digital domain, and they fail to take advantage of the analog computation capabilities of the memristor device. The errors from domain conversion were avoided through performing the computation in the analog domain. This not only eliminates the need for any AD/DA conversion, but also reduces the errors in processing. A comprehensive study has been developed to investigate the feasibility of the proposed approach by designing the crossbar for Bayesian text classification. The Bayesian classifier is a simple and robust algorithm used to classify unstructured data [117]. In contrast to other classification algorithms, such as K-means, Decision tree, and Random forest, the Bayesian classifier gives relatively better results on smaller training datasets [118]. Finally, an attempt has been initiated to test the proposed approach's performance on two standard datasets [119], [120].

The main contributions of this chapter are listed below:

1) To the best of our knowledge from the available literature, this is the first time to develop a memristor-based Bayesian text classifier for applications in digital domain.
2) To determine the efficacy of the proposed circuit in classifying the texts with high accuracy.

The rest of the chapter is organized as follows. The proposed model and the implementation methodology of the memristor-based Bayesian text classifier system are described in Section 3.2 and 3.3. In Section 3.4, circuit design and modelling is been detailed. Experimental results are provided and discussed in section 3.5. Finally, the chapter is concluded in Section 3.6.

3.2 Mathematical Analysis of Classifier

From the literature, it was understood that the Bayes' theorem is the basic building block for the mathematical foundation for Bayesian text classifier [121]. For two events A and B (say), the theorem is mathematically stated as follows:

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)} \tag{3.1}$$

where $p(A)$ and $p(B)$ are the probabilities of occurrences of events A and B, respectively, and $p(A|B)$ and $p(B|A)$ represents conditional probabilities. The $p(A|B)$ measures the probability of occurrence of event A subject to the condition that event B has already occurred and vice versa.

Let T= $\{t_1, t_2, t_{3,} \ldots\ldots, t_i\}$ be the set of texts to be categorized and C = $\{c_1, c_2, c_3, \ldots\ldots, c_m\}$ be the set of tentative categories for the texts. Then according to Bayes' theorem, the probability of a text t_i belonging to a category c_m is given by the following equation:

$$p(c_m \mid t_i) = \frac{p(t_i \mid c_m) \times p(c_m)}{p(t_i)} \tag{3.2}$$

The above is the fundamental equation for our text classifier. $p(c_m)$, $p(t_i \mid c_m)$, and $p(c_m \mid t_i)$ are known as the prior, likelihood, and posterior probabilities, respectively. The category of text t_i represented by C_{t_i}, is determined as per the following equation:

$$C_{t_i} = argmax_m \left[\frac{p(t_i \mid c_m) \times p(c_m)}{p(t_i)}\right] \tag{3.3}$$

$p(t_i)$ is constant for an input text and is independent of the text category. Thus, Eqn. (3.3) can be simplified as follows:

$$C_{t_i} = argmax_m [p(t_i \mid c_m) \times p(c_m)] \tag{3.4}$$

Assuming the constituent words of the text occur independently of each other [122], if w_n is the n^{th} word in the text t_i, and g_n is the frequency of its occurrence then,

$$p(t_i \mid c_m) = \prod_n p(w_n \mid c_m)^{g_n} \tag{3.5}$$

From Eqn. (3.4) and Eqn. (3.5),

$$C_{t_i} = argmax_m [\prod_n [p(w_n \mid c_m)^{g_n}] \times p(c_m)] \tag{3.6}$$

For text with large word counts, calculation of $P(t_i \mid c_m)$ is a tedious task. This calculation is simplified by breaking down multiplication into simpler addition by utilizing the logarithmic function. As a result, Eqn. (3.6) is modified to Eqn. (3.7) as follows:

$$C_{t_i} = argmax_m\left[\sum_n\left[g_n \times log[p(w_n \mid c_m)]\right] + log[p(c_m)]\right] \qquad (3.7)$$

3.3 Crossbar Circuit for Sample Binary Dataset

A comprehensive effort was initiated to study the ability of the proposed circuit to semantically classify text data by designing it for dataset 1 and tabulated in Table 3.1. Dataset 1 is a binary dataset consisting of 4 texts that have been semantically categorized as negative and positive statements. The states of the memristor switches in the designed crossbar were derived from this dataset.

Table 3.1: Sample dataset 1

Text	Category
These 3 movies are really good !!!!	Positive
The food is too bland.	Negative
The Teaching Assistant for This Course Is Really Good.	Positive
The job is too tedious	Negative

The texts in the dataset were first cleaned of characters and words that were irrelevant to the classification process [123]. These words, known as stop-words, include commonly used terms like *this, that, is, etc., again, there, about, with, during, having, out,* and *very* [124]. Numerals and punctuations were also removed from the text as they do not help in determining the text category. Moreover, there is no semantic difference between a character in its upper and lower case, i.e., the character case of the text is immaterial to the process of text classification [125]. As a result, we chose to convert all texts in the dataset to small case characters. Dataset 2 given in Table 3.2 is the cleaned and normalized version of dataset 1. The prior and posterior probabilities required for text classification were calculated from dataset 2.

Table 3.2: Sample dataset 2

Text	**Category**
movies really good	Positive
food bland	Negative
teaching assistant course really good	Positive
job tedious	Negative

3.3.1 Prior Probability

The prior probability was calculated as follows:

$$p(c_m) = \frac{|c_m|}{|T|} \tag{3.8}$$

where $|c_m|$ is the total count of texts belonging to c_m and $|T|$ is the total count of texts in the dataset. In dataset 2, the four texts were equally distributed among the two possible categories. As a result,

$$p(c_{positive}) = p(c_{negative}) = \frac{1}{2} \tag{3.9}$$

3.3.2 Likelihood Probability and Accounting for Words Absent from the Training Dataset

The likelihood probability was calculated as follows:

$$p(\, w_n |\, c_m) = \frac{|w_n | c_m|}{|w| c_m|} \tag{3.10}$$

where $|w_n|c_m|$ represents the total count of w_n in the texts belonging c_m and $|w|\, c_m|$ is the total count of words in the texts of c_m. For input texts consisting of words that were absent from the training dataset, represented by w_{ua}, the likelihood probability $p(\, w_{ua}|\, c_m)$ was zero. As a result, the posterior probability in the governing equation of the classifier Eqn. (3.2), also diminished to zero and it became difficult to categorize such texts. This issue was addressed by adding a bias B, to the word counts. Thus, the likelihood probability was modified as follows:

$$p(w_n| \, c_m) = \frac{|w_n| \, c_m| + B}{|w| \, c_m| + 1 + (n \times B)} \tag{3.11}$$

3.4 Implementation Section

3.4.1 Circuit Design

As illustrated in Fig. 3.1, the proposed circuit is a crossbar with memristor switches. The crossbar consists of $n + 2$ rows and m columns, where n is the maximum number of distinct words present in the training dataset, and m is the number of tentative categories for the text. For dataset 2, n is 10 and m is 2.

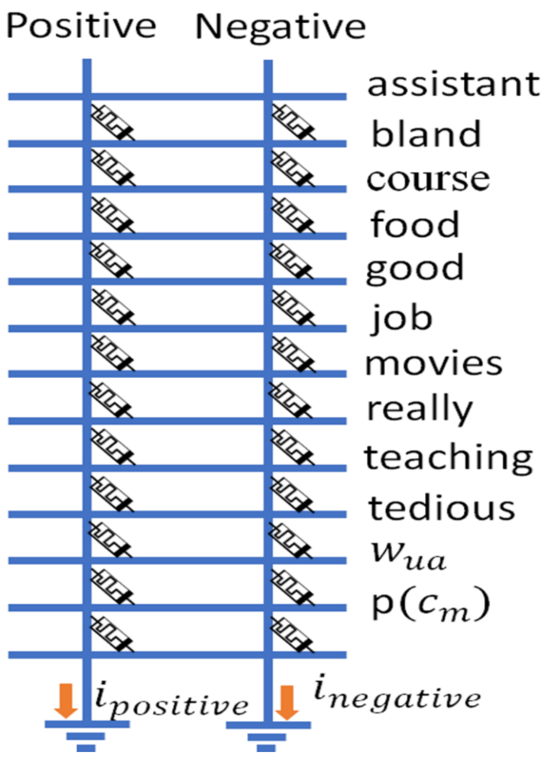


Fig. 3.1: Memristor crossbar network developed using dataset 2.

The top 10 rows map to a word and each of the 2 columns map to a possible category. Memristors at the intersections of the crossbar are employed to store probabilities required for text classification. On one side, the states of the memristors at any of the top 10 rows are determined by the likelihood probability $p(w_n|\, c_m)$. On the other side, the memristances at the bottom-most row are determined by the prior probability $p(c_m)$. Words absent from the training dataset are represented by the second last row of the crossbar. The memristors of this row are used to store the likelihood probabilities of encountering such words.

3.4.1.1 Modelling the Circuit

The crossbar designed for dataset 2 was modelled in MATLAB R2019. The crossbar switches are based on our previously fabricated and characterized Pt/Cu:ZnO/Nb:STO memristor [50]. The memristor is modelled using the Voltage ThrEshold Adaptive Memristor (VTEAM) paradigm [53]. According to the VTEAM model, state transitions are allowed only for voltages outside the range of V_{on} and V_{off}, which was -1.2 V and 1.35V, respectively in the present case. The constants K_{off} and K_{on} determining the rate of state transitions were set to a value of +40 and -80. These values were chosen such that the current-voltage characteristics of the modelled and fabricated memristor commensurate with one another [50]. Exponents α_{on} and α_{off} having a numerical value of 3 and 2 were used to introduce device nonlinearities into the model. Finally, the modelled memristor was programmed and tuned using pulse width modulated signals, as demonstrated in one of the studies [126].

3.4.1.2 States of Memristor Switches

The memristance at the intersection of any of the top $n+1$ rows and the m^{th} column is as follows:

$$M_{r,m} = \frac{-1}{\log(p(w_r|c_m))} \qquad 1 < r < n+1 \tag{3.12}$$

Where 'r' is the row between 1 and n+1th row. Similarly, the memristance at the intersection of the bottom-most row and the m^{th} column is given by:

$$M_{n+2,m} = \frac{-1}{\log(p(c_m))} \tag{3.13}$$

Table 3.3: Likelihood probability for negative category

Word	Count	Bias	Count + Bias	Likelihood Probability	Memristance	Memristance × scaling factor.
Assistant	0	1	1	0.0667	0.8504	850.4
Bland	1	1	2	0.1333	1.1426	1142.6
Course	0	1	1	0.0667	0.8504	850.4
Food	1	1	2	0.1333	1.1426	1142.6
Good	0	1	1	0.0667	0.8504	850.4
Job	1	1	2	0.1333	1.1426	1142.6
Movies	0	1	1	0.0667	0.8504	850.4
Really	0	1	1	0.0667	0.8504	850.4
Teaching	0	1	1	0.0667	0.8504	850.4
Tedious	1	1	2	0.1333	1.1426	1142.6
Any unencountered words	0	1	1	0.0667	0.8504	850.4

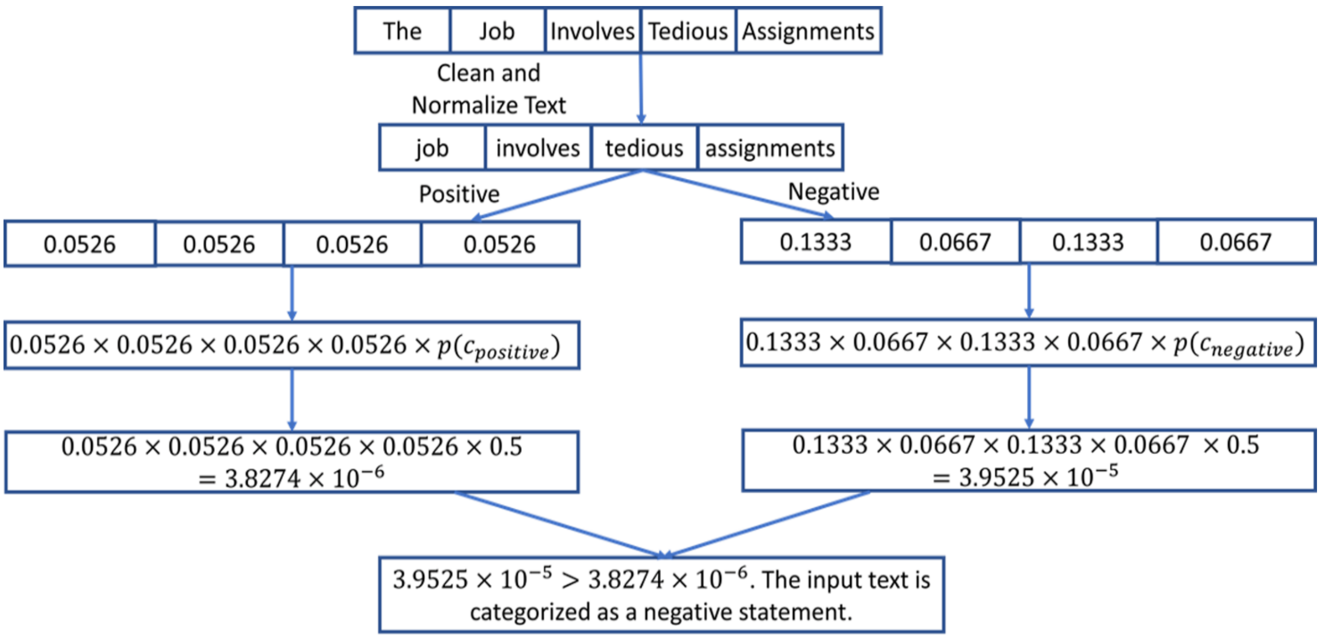
As expressed in Eqn. (3.12) and Eqn. (3.13), the prior and likelihood probabilities were written into the memristive switches in the negative inverse logarithmic form. This ensured that the resultant current flowing through the columns of the crossbar was in proportion with the posterior probability given by Eqn. (3.2). The range of numerical values that can be stored in the memristor is $\{R_{on,} R_{off}\}$. The memristors of our crossbar have a R_{on} and R_{off} value of 1.507×10^2 ohm and 1.524×10^5 ohm, respectively [51]. It is noteworthy to mention that the memristances calculated in Table 3.3 and Table 3.4 were significantly lower than R_{on}, therefore the calculated memristances were further scaled by a factor of 1000.

Table 3.4: Likelihood probability for positive category

Word	Count	Bias	Count + Bias	Likelihood Probability	Memristance	Memristance × scaling factor.
Assistant	1	1	2	0.1053	1.0229	1022.9
Bland	0	1	1	0.0526	0.7819	781.9
Course	1	1	2	0.1053	1.0229	1022.9
Food	0	1	1	0.0526	0.7819	781.9
Good	2	1	3	0.1579	1.2475	1247.5
Job	0	1	1	0.0526	0.7819	781.9
Movies	1	1	2	0.1053	1.0229	1022.9
Really	2	1	3	0.1579	1.2475	1247.5
Teaching	1	1	2	0.1053	1.0229	1022.9
Tedious	0	1	1	0.0526	0.7819	781.9
Any unencountered words	0	1	1	0.0526	0.7819	781.9

3.4.1.3 Determining Text Category

An input text was classified by applying a DC pulse of width 0.01ms to the rows (corresponding to the constituent words of the text) and measured the resultant current flowing through the columns of the crossbar. The amplitudes of the pulse were multiples of base voltage V. If g_n is the frequency of the word (w_n) in the input text, then the voltage applied to the corresponding row in the crossbar was $g_n \times V$. For words that were absent from the text, g_n is considered as zero; hence 0 V was applied to the rows representing these words. Furthermore, the voltage applied to the row corresponding to w_{ua} was $a \times V$, where a is the number of words in the input text that were not accounted for in the crossbar. The voltage applied at the bottom-most row was V. The base voltage V was chosen such that the applied voltages do not alter the already programmed states of the memristor. As described in section 3.3, the applied voltage must lie within the range {-1.2V, 1.35V}. In this work, the *V* was set to ± 0.01 V to ensure that for large word counts (say 100), the voltage applied to the rows lie within the allowed range.



The
Job
Involves
Tedious
Assignments
Clean and Normalize Text
job
involves
tedious
assignments
Positive
Negative
0.0526
0.0526
0.0526
0.0526
0.1333
0.0667
0.1333
0.0667
$0.0526 \times 0.0526 \times 0.0526 \times 0.0526 \times p(c_{positive})$
$0.1333 \times 0.0667 \times 0.1333 \times 0.0667 \times p(c_{negative})$
$0.0526 \times 0.0526 \times 0.0526 \times 0.0526 \times 0.5 = 3.8274 \times 10^{-6}$
$0.1333 \times 0.0667 \times 0.1333 \times 0.0667 \times 0.5 = 3.9525 \times 10^{-5}$
$3.9525 \times 10^{-5} > 3.8274 \times 10^{-6}$. The input text is categorized as a negative statement.

Fig. 3.2: Flowchart summarizing various steps involved in Bayesian text classification.

The magnitude of the resultant current (i_m) flowing through the columns of the crossbar was in negative proportion to the posterior probability in Eqn. (3.7) is expressed as:

$$i_m = \sum_{r=1}^{n} \left[\frac{g_r \times V}{M_{r,m}}\right] + \frac{a \times V}{M_{n+1,m}} + \frac{V}{M_{n+2,m}} \tag{3.14}$$

The negative proportionality between the resultant current and posterior probability was a result of the negation operation in Eqn. (3.12) and Eqn. (3.13) and, as a result, the input text was classified as per the following equation:

$$C_{t_i} = argmin_m \left|\sum_{r=1}^{n} \left[\frac{g_r \times V}{M_{r,m}}\right] + \frac{V}{M_{n+1,m}} + \frac{a \times V}{M_{n+2,m}}\right| \tag{3.15}$$

Attempts were put to validate the developed crossbar by testing it on the text *The job involves tedious assignments*. Fig. 3.2 summarises the steps involved in the Bayesian classification of the text.

3.5 Results and Discussions

The crossbar circuit was further validated on two standard binary datasets (Fig. 3.3). On one side, the first dataset comprised of short SMS messages that have been classified as either ham or spam [119]. On the other side, the second one consists of positive and negative reviews extracted from the IMDb movie database [120] (Fig. 3.4). Voltage signals were applied to rows corresponding to the words of the input text. The resultant current flowing through the columns of the crossbar were compared for text categorization. The SMS dataset comprised of 5,575 texts, 86,519 words, and 15,586 unique words, while the IMDb dataset consisted of 50,000 texts, 11,557,403 words, and 439,779 unique words.

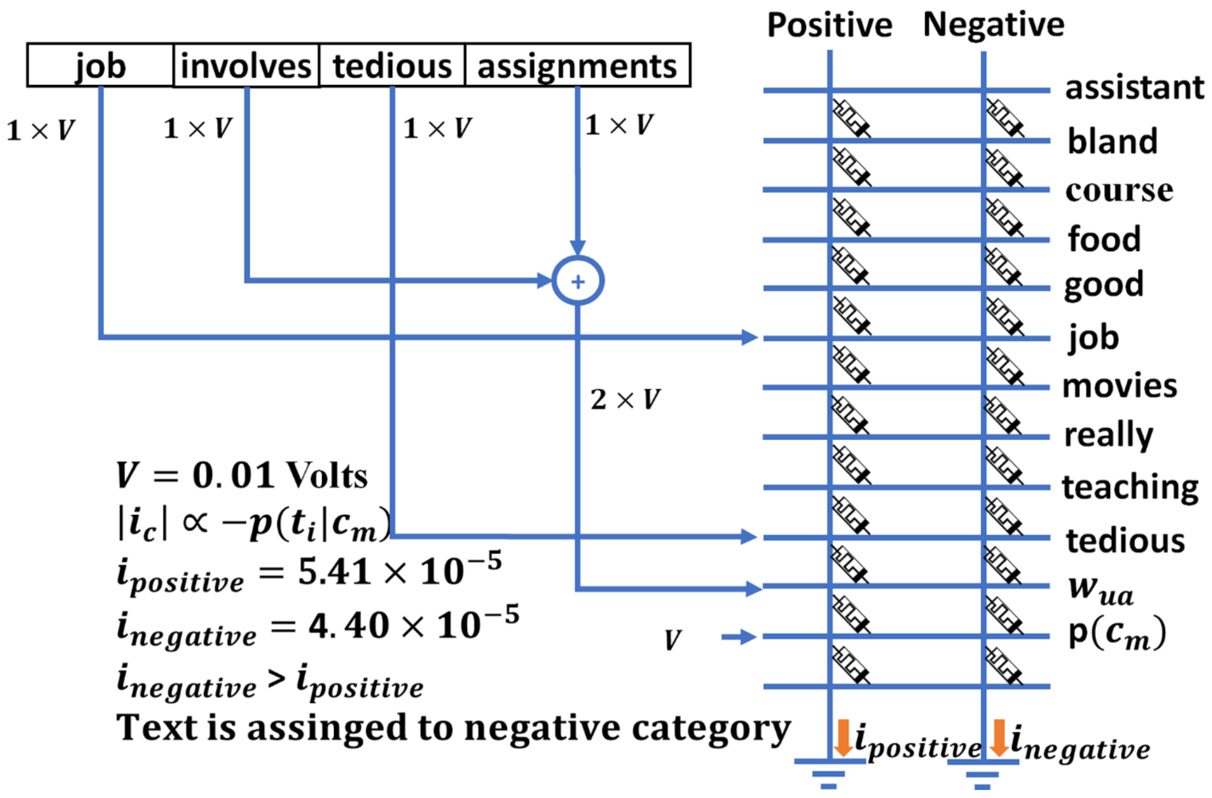


Fig. 3.3: Memristive crossbar network for text classification.

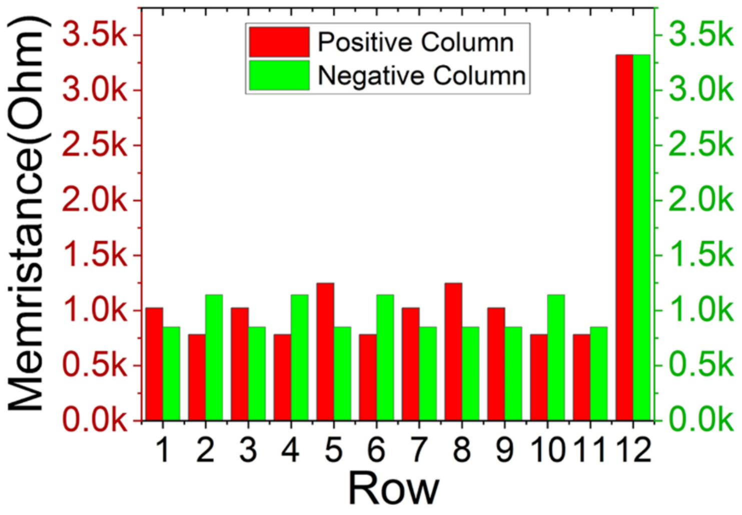


Fig. 3.4: Memristance *vs*. Row at the crossbar intersections.

Each dataset was divided into two parts: the first part was used for training the circuit, while the second part was considered for testing it. In this context, one can define the training ratio, e as follows

$$e = \frac{number\ of\ texts\ used\ for\ training\ the\ circuit}{total\ number\ of\ texts\ in\ the\ dataset} \tag{3.16}$$

The proportion of texts used to train the circuit was e, and that used to validate was $1 - e$. The accuracy of classification for selected values of e was calculated and tabulated in Table 3.5. As illustrated in Fig. 3.5(a) and Fig. 3.5(c), the crossbar circuit was trained using a small part of the dataset ($e < 0.3$), which was capable to categorize a relatively larger number of texts with high accuracy ($> 80\%$). This is because the Bayesian classifier assumes that the words of the text occur independently to each other [122]. As a result, the circuit can better learn the relationship between the texts and its corresponding categories. Fig. 3.5(b) and Fig. 3.5(d) depict the dependence of the number of rows in the crossbar on the training ratio. As expected, it was observed that the number of rows along with the accuracy of classification gets increased with the proportion of texts used in training the circuit.

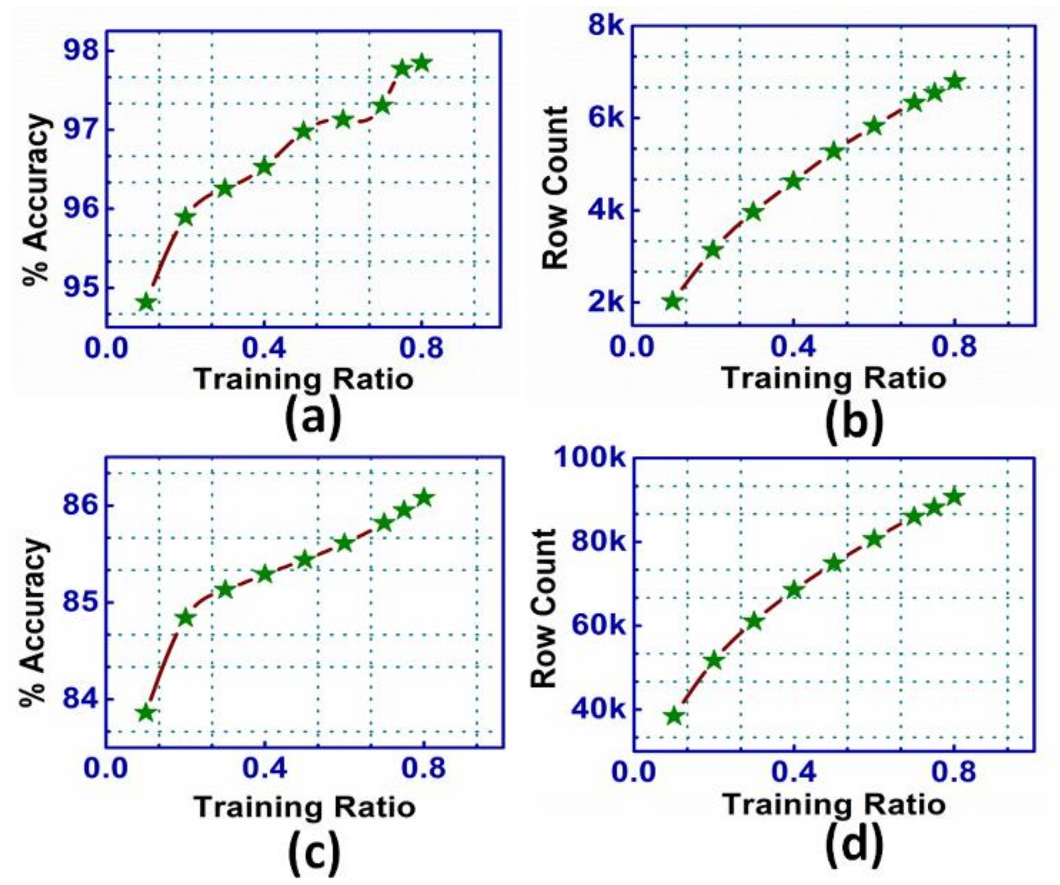


Fig. 3.5: (a) Accuracy *vs.* Training ratio of classification for SMS dataset. (b) Row count *vs.* training ratio for SMS dataset. (c) Accuracy *vs.* training ratio of classification for IMDb dataset. (d) Row count *vs.* training ratio for IMDb dataset.

Furthermore, as depicted in Fig. 3.6, it has been observed that performing computations in the analog domain results in higher classification accuracy. When computations are performed, the classification accuracy in the digital domain plateaus was found to be ~ 95% for the SMS dataset and 84% for the IMDb dataset. The plateauing of accuracy can be attributed to errors arising from AD/DA domain conversion. A suitable value of the training ratio is 0.75 [127]. For this value of e, the crossbar circuit for the SMS dataset had a dimension of 6545×2 and a classification accuracy of 97.77 %, while that for the IMDb dataset had a dimension of 88508×2 with an accuracy of 85.95%.

Table 3.5: Result of testing the proposed circuit on standard datasets

e	Crossbar Dimension $rows \times columns$		Accuracy (%)	
	SMS Dataset	IMDb Dataset	SMS Dataset	IMDb Dataset
0.1	2025×2	38402×2	94.82	83.86
0.2	3134×2	51691×2	95.90	84.84
0.3	3961×2	61027×2	96.26	85.13
0.4	4627×2	68512×2	96.53	85.29
0.5	5276×2	74901×2	96.98	85.45
0.6	5826×2	80657×2	97.13	85.61
0.7	6331×2	86006×2	97.31	85.83
0.75	6545× 2	88508×2	97.77	85.94
0.8	6797×2	90758×2	97.85	86.08

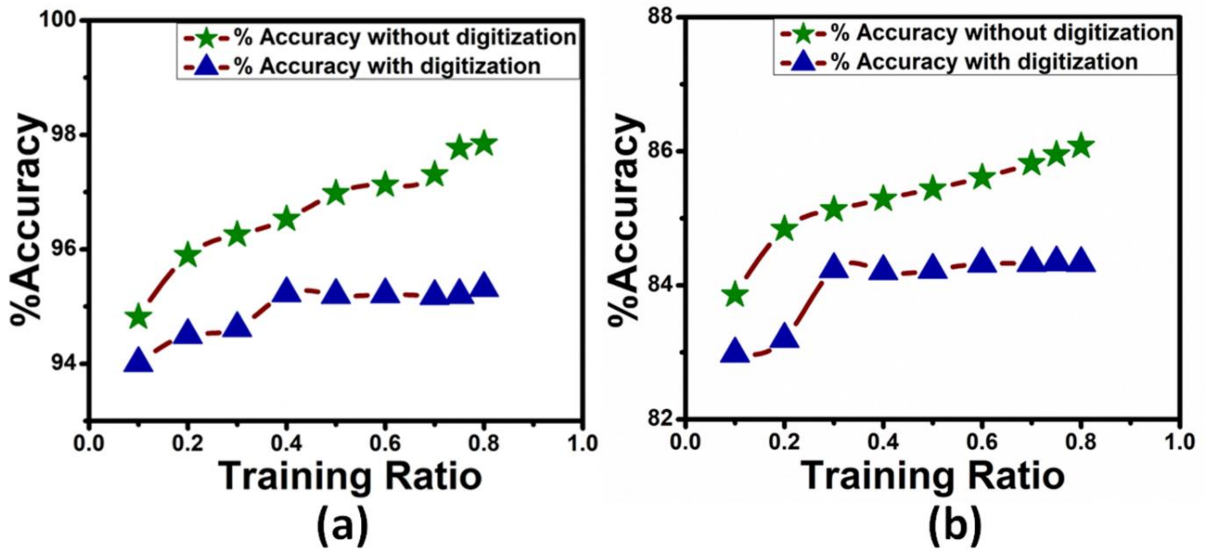


Fig. 3.6: (a) Accuracy *vs.* Training ratio, classification accuracy without digitization and after digitization on SMS dataset. (b) Accuracy *vs.* Training ratio, classification accuracy without digitization and after digitization on IMDb dataset.

Efforts were also dedicated to analyze the power, area, and latency of the proposed circuit for its practical consideration. It was observed that the average power consumed by SMS data is 1.4516×10^{-5} watts, while that for classifying IMDb data is 2.3×10^{-3} watts. Both these powers were found to be significantly smaller than that of a x86 CPU, which is anywhere between 65-100 watts [128]. The on-chip area of the memristor is given by $4F^2$, where F is the feature size [129]. For a feature size of 22 nm, i.e., for 45nm technology node, the average area of the crossbar for SMS dataset is 9.577×10^{-3} nm, and that for the IMDb dataset is 0.138 nm. The latency of the circuit is given by the product of the width of a single input pulse and the number of texts to be classified, this is because the voltages are applied to the crossbar in a parallel fashion.The latency for the SMS dataset was found to be 55.75 ms, and that for the IMDb dataset was 500ms.

In practice, the crossbars given in Table 3.5 might be hard to fabricate and scale. One can overcome this issue by breaking down the crossbars in smaller $n \times 2$ arrays and organizing them in a tiled structure, as explained in [130]. But unlike in [130], one does not have to face difficulties with data dependency between the tiles as the Bayesian classifier functions under the assumption that the words of the text occur independently one to another.

3.6 Summary

In this work, an attempt has been made to develop a high precision memristive crossbar circuit for Bayesian inference, followed by its specific implementation for semantic text classification. The errors arising from AD/DA domain conversion was circumvented by realizing that such domain conversion is unnecessary, and the associated errors can be avoided by taking advantage of memristors analog computational capabilities. The efficacy of the processor was tested on two different datasets consisting of a total of 55,575 texts. The circuit was able to classify the texts with an average accuracy of 91% while consuming 1000 times less power and area of a conventional processor, which open up a new path for developing future generation memristor based in-memory computing. In order to provide a contrast of future work, it is noteworthy to mention that the crossbar circuit lacks the flexibility of conventional processors, which can perform a plethora of tasks. However, with some careful modification to the proposed circuit, one can overcome these limitations and come up with a single crossbar that can perform multiple functions such as Boolean computation, image segmentation, and speech recognition. Furthermore, there is a need to devise a method for online training of the crossbar, which can eliminate the need for a software interface to load information into to memristor array. For the widespread adoption of in-memory processors, there is a requirement for smart systems that can interface the device with existing computing systems such as DRAM, graphical processing units, input-output devices like keyboard, speakers, and microphones.

Remarks: Below mentioned paper was published based on this chapter

- **Priyanka B. Ganganaik,** Aditya Viswakumar, P. Michael Preetam Raj, BVVSN Prabhakar Rao, and Souvik Kundu. "Memristor-based in-memory processor for high precision semantic text classification." Computers & Electrical Engineering 92 (2021): 107160.

Chapter-4

Realization of Memristive State Machine for Smart Edge Detector Applications

4.1 Introduction

As described in the earlier chapters that the memristor is a resistive switching non-volatile memory device that can remember its memristive states, it is worth noting that memristive state transitions can be attained by utilising external voltage sources [131]. Importantly, the transitions traverse through intermediate resistive states that exist between the extreme resistive limits (ON and OFF states). Interestingly, these intermediate states can be exploited for memory storage and computational applications [132], [133] . In order to develop different states, state machines were implemented based on CMOS digital logic circuits over the past few decades. However, these CMOS based circuits are disadvantageous in terms of their switching speed, computational complexity, and power consumption [134]. Under these circumstances, memristor based circuits render low power and high-speed computations with reduced complexity [135].

Recently there has been renewed interest in utilizing multilevel memristive circuits to implement emerging memory architectures [133], [136]–[138] . Since most of the real-time signals are analog in nature, a tremendous effort has been devoted to develop analog computing [139]. Interestingly, one could further extend the idea of multilevel logic to achieve a large number of states through exploiting the analog behaviour of a memristor. It is essential to mention that the number of these levels depends on the memristive noise margin [126]. Within the memristor, the resistive state transition takes place gradually and leads to several intermediate states if the R_{on}/R_{off} ratio is high (~$2\text{x}10^3$ in the current study). Based on the controllable transitions among these states, memristive state machines (MSMs) need to be developed. Realizing MSMs is an innovative approach because these systems can further perform necessary computations for artificial human reasoning, edge detection, etc. [140]–[142]. Unfortunately, this approach has not been exploited yet and the progress in this area has been slower as compared to the memristor based other applications such as different neuromorphic circuits, synapses, etc. Edge detection, which offers object segmentation in an image, has been an essential segment in the image processing area over the past few decades [143]. Besides, it has several other applications in the field of computer vision, medicine, artificial intelligence, and biometric based identification systems [13], [144]–[146]. Convolution and correlation techniques are the most conventional approaches so far to perform the edge detection of an image [147], [148]. However, these techniques were disadvantageous in terms of execution speed, computational complexity and the number of electronic components [149].

On the other side, fuzzy logic based implementation is advantageous as the computing is carried out in analog domain and the need for converters and conventional digital circuitry is eradicated [150]. Fuzzy systems were employed for edge detection of an image [15], [151], [152]. On one side, these systems produced improved edge detection when compared to any non-fuzzy methods [146], which brought about to incomplete edges [78]. On the other hand, fuzzy pre-processing lead to enhanced region distinction that further assisted in achieving better edge detection [78]. In one of the studues multilayer perceptron based edge detection through fuzzy clustering technique is developed [15]. In another study, edge detection was investigated, where it was performed and successfully tested with benchmark images [151]. However, these methods are computationally complex and disadvantageous in terms of area, power consumption and delay since conventional transistor based computational architectures were employed in their architecture[13], [145]. In order to address these issues, researchers have attempted edge detection by utilizing memristors and obtained better performances[146], [147]. However, their techniques are valid only for binary images and not suitable for grayscale images. Later, memristor based networks were developed for this application on grayscale images[148], [149]. In all these techniques, multiple memristors were employed, which consume more power and add on delays. Therefore, it will be interesting to develop MSM based edge detection in which a single memristor can store all the grayscale intensity values.

In this work, for the first time to the best of our knowledge, efforts are put to develop Cu:ZnO based MSM through simulation route for edge detection in an image. Initially, the Pt/Cu:ZnO/Nb:STO memristor data were incorporated into the simulation (*MATLAB* and *Cadence*) environment by utilizing the obtained electrical parameters. Once both the simulation and experimental current-voltage characteristics and other electrical performances got matched [50], [51], MSM circuits were simulated and efforts were devoted to obtain the multiple states within a memristor. The idea of MSM was further extended to perform tunable edge detection for image processing applications. The obtained edge detections were compared with other popular conventional software based edge detections such as *Canny, Sobel, and Prewitt* [147], [148], [153]. It was observed that the proposed MSM based system performed much better when compared to all the other above-mentioned systems. Its efficacy was discussed for future generation accurate and faster real time image processing applications.

4.2 Implementation Section

4.2.1 Device Structure and Simulation Setup

The VTEAM Simulink model for the switching memristor was based on the state variable update, as shown in Fig. 4.1. In this context, different equations were considered for computations based on the comparisons between the applied voltage and memristive threshold voltages. As per Fig. 4.1, the device equation was employed for memristive state variable that was seen to increase for value above the positive threshold voltage and decrease for a value below negative threshold as presented in Eqn. (4.1), [53].

$$\frac{dw(t)}{dx} = \begin{cases} K_{off}.\left(\frac{v(t)}{v_{off}} - 1\right)^{\alpha_{off}}.f_{off}(w), & 0 < v_{off} < v \\ 0, & v_{on} < v < v_{off} \\ K_{on}.\left(\frac{v(t)}{v_{on}} - 1\right)^{\alpha_{on}}.f_{on}(w), & v < v_{on} < 0 \end{cases} \tag{4.1}$$

where f_{off} and f_{on} are the window functions [53] with values chosen equal to 1 as per the device model. The other parameters and their values corresponding to the fabricated memristor are presented in [50], [51]. These parameter values were extracted by fitting the fabricated device I-V readings within the memristive equations in order to map the device characteristics.

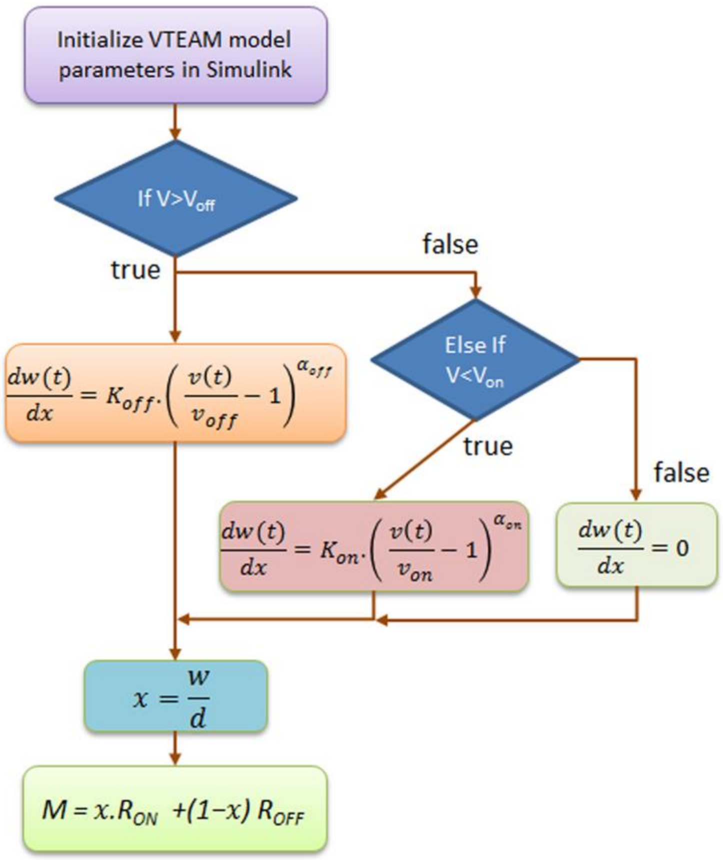

Initialize VTEAM model parameters in Simulink
If V>V$_{off}$
true
false
$\frac{dw(t)}{dx} = K_{off}.\left(\frac{v(t)}{v_{off}} - 1\right)^{\alpha_{off}}$
Else If V<V$_{on}$
true
false
$\frac{dw(t)}{dx} = K_{on}.\left(\frac{v(t)}{v_{on}} - 1\right)^{\alpha_{on}}$
$\frac{dw(t)}{dx} = 0$
$x = \frac{w}{d}$
$M = x.R_{ON} + (1-x) R_{OFF}$

Fig. 4.1: Flow chart for VTEAM model based memristor.

In the Eqn. (4.1), *d* is the thickness of the active material (*Cu:ZnO*), whereas the values of the constants K_{ON}, K_{OFF}, α_{OFF} and α_{ON} affect the variation of state variable x with respect to time [51], [53]. Further, R_{off} and R_{on} represent the high resistive and low resistive states of the memristor, respectively. Whereas, w represents the weight change of the memristor. The above mentioned parameter values were utilized to invoke the fabricated memristor into the *MATLAB Simulink* environment. This simulation based model was utilized to develop the MSM circuits in *MATLAB Simulink*. Further, a system was developed (in the same environment) through mapping the grayscale intensities of an image with the corresponding memristive states. This conversion was utilized to propose the memristive edge detection system in *Simulink* environment.

4.3 Results and Discussions

In order to generate multilevel states, an MSM was developed and is shown in Fig. 4.2. It was understood that the memristive state does not change when read with a lower voltage (below the threshold value) [54]. Therefore, a 2 V AC voltage was utilized to write, whereas a low DC voltage was considered for read operation. In the MSM circuit (Fig. 4.2(a)), these voltage sources were connected through a double pole double throw (DPDT) switch such that one input was kept at a DC supply of 1.10 V (read mode), while the other input was fixed to an AC supply (2 V, 10 kHz). It is essential to mention that one can tune the value of the AC supply in order to suit practical applications. The increase in the magnitude of the AC voltage or the reduction in the frequency may lead to faster memristive drift rates due to the fact that the large AC voltage will cause the ions (within the active layer of memristor) to drift at a faster rate. Further, the low frequency will facilitate the memristor to respond to the input signal. The DPDT switch was controlled by a signal voltage, which is basically a pulsating voltage source of amplitude 2 V, pulse width 1 ns and time period 1.2 ns. This allows the memristor connections to switch in between read and write operations. During analysis, it was identified that memristance drifts with the time duration for which the AC signal was applied. One can write a resistance value through disconnecting AC supply at the corresponding time. For example, among the multiple memristive states, 20 MΩ, 60 MΩ, 80 MΩ and 120 MΩ were written onto memristor through adjusting the memristance decreasing time durations to 5.624 µs, 5.55 µs, 5.39 µs, and 5.32 µs, respectively, as shown in Fig. 4.2(b).

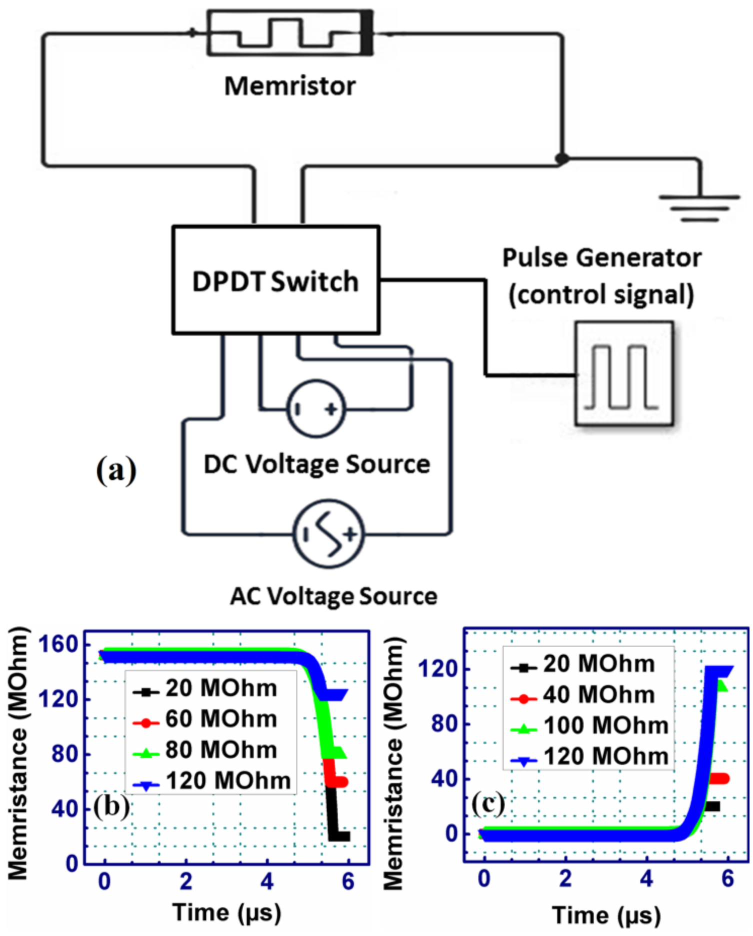


Fig. 4.2: (a) Threshold block for programming one memristor and Schematic of the proposed MSM circuit. The controlled memristive state transitions were possible through the continuous switching between the read and write cycles. (b) Decrementing memristance characteristics of *Cu:ZnO* based memristor. Different resistance values were written onto memristor through controlling the time duration for which the AC signal was supplied. (c) Incrementing memristance characteristics. 20 MΩ, 40 MΩ, 100 MΩ and 120 MΩ were written onto the memristor through utilizing the proposed switching technique. One can adjust the switching time durations in order to write any required value within the possible memristive states.

Similarly, the memristance can be increased from R_{on} to 20 MΩ, 40 MΩ, 100 MΩ and 120 MΩ by selecting the switching time durations as 5.24 µs, 5.36 µs, 5.57 µs and 5.59 µs, respectively.

Through utilizing these techniques, one could attain controlled switching in between the memristive states for MSM applications. In this process, memristance can be programmed to any value in between R_{on} and R_{off} (in this case only four values are shown). Each memristive state (Fig. 4.2) was characterized by the state variable *x*, where *x*= *w/D*, *w* is width of the region contain oxygen vacancies, and *D* is the length of the active layer in memristance *(M)*. *w* is seen to increase linearly as a function of the memristive state variable as represented in Eqn. (4.2) given below [53]:

$$M = x.R_{on} + (1 - x).R_{off} \tag{4.2}$$

From the Fig. 4.2, it can further be depicted that the memristance gets decreased with time and one can possess control over the value written in the memristor, through utilizing the DPDT switch. The characteristics of the state transitions towards the memristive incremental (Fig. 4.2(c)) direction were obtained by exchanging the memristor terminal connections in Fig. 4.2.

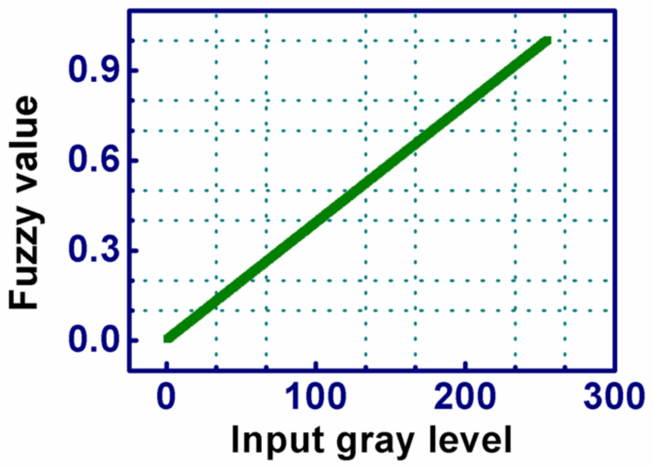


Fig. 4.3: Mapping Luma values onto memristive multilevel logic states. The 0 to 255 grayscale levels were mapped to the corresponding memristive states from 0 to 1.

The memristive state was characterized by the state variable x, where R_{on} (x = 1) corresponds to logic 1 and R_{off} (x = 0) represents logic 0. Further, the grayscale intensities were converted into memristive state values (shown in Fig. 4.3). For an 8-bit grayscale image, each pixel possesses a luma measure of L ranging from 0 to 255 [154]. With this in consideration, a pixel with luma value L is converted to a fuzzy logic value x= L/255 (if L varies from 0 to 255 then x varies from 0 to 1) that is stored in the memristor in terms of its state variable. For this purpose, Eqn. (4.3) was employed in order to write this respective fuzzy logic or rather change the memristive state according to the state variable x. This memristance value was read back using a low DC voltage and the operation was repeated for each iteration.

It is noteworthy to mention that the white pixel is considered as logic input 1, whereas the black one is logic 0. The state value was chosen to be directly proportional to the grayscale intensities for scaling into the memristive state domain as depicted in Fig. 4.3. From the Fig. 4.3, it can be seen that the memristive state value increases linearly from 0 to 1 as the Luma value varies from 0 to 255. Further, the graph follows the relation x = $\frac{L}{255}$, where x was chosen to linearly increase with L. The main idea of programming memristors is to correlate the memristive state variable to the fuzzy logic. The circuit shown in Fig. 4.2(a) was utilized to program the memristor. As shown in Fig. 4.2(a) the threshold block employed to program the memristors contains each memristor connected across read and write voltage sources. The DPDT switch was utilized to shuttle between write and read cycles. Also, memristive write voltages (Vx) were determined from the following relation, which corresponds to writing the state values x of the memristor as shown in Eqn. (4.3) that was obtained by integrating Eqn. (4.1) for a voltage above threshold and solving for V, [53],

$$V_x = \left[\left(\frac{x \times D}{\mathrm{K_{off}} \times (pulse\ width)}\right)^{1/\alpha_{\mathrm{off}}} + 1\right] . V_{off} \tag{4.3}$$

The proposed circuit mainly works based on threshold gates implemented with the help of control signals and switches. The threshold block circuit consists of an array of four memristors arranged parallelly with each memristor connected across a DPDT switch (as shown in Fig. 4.2) and each accepting a pixel input. The switch is mainly used to alternate back and forth between the write and read cycles of the device with help of pulse generator that acts as the control signal to trigger the system for every cycle. During the write cycle the memristor is programmed with the pixel values and afterwards the computation is carried out during the read cycle. In order to detect edges, the pixel values input to the threshold block are the values of 2X2 window (shown in Fig. 4.4(a)) sampled from the set of pixels in the image. In order to further carry out the computation for the entire image with the array of four memristors in parallel, this 2X2 window was traversed throughout the image. The pixels were sampled by utilizing a memory buffer that starts with the position of 0 row count and 0 column count. This buffer was moved rightwards to sample all the columns in a given row and subsequently downwards to sample all the rows. With this sampling and processing of all pixels, the edge detected output of the complete image was obtained.

In order to perform edge detection for an image, one has to compare the grayscale intensity value of a pixel with its neighbouring pixels. In such case, for each iteration, four nearby pixels (*P1, P2, P3* and *P4*) of an image were considered (Fig. 4.4(a)). If-then rules were utilized on pixel value to perform the necessary comparison [155]. The if-then cases were: (i) if all the four pixels *P1, P2, P3* and *P4* are black (in the range of {1, 50}) or white (in the range of {200, 255}) or gray (in the range of {51, 199}), then the output is not an edge. (ii) Else if at least one of the pixels is contrast from the other pixels then the output is yielded as an edge. Through utilizing these rules and memristor based threshold logic, the proposed schematic of the computational flow is shown in Fig. 4.4(b). The circuit was composed of four memristors, which accept four inputs (each corresponds to four pixels). To process the input for edge detection in an image, our system utilizes memristive threshold logic. In order to have better insight on the memristors based threshold logic, one could assume if V and Mem are the voltage and memristance, respectively then the output (Y) can be defined as in [54]:

$$Y = \begin{cases} 0 \; if \; \frac{V}{Mem} < I_{ref} \\ 1 \; if \; \frac{V}{Mem} > I_{ref} \end{cases} \tag{4.4}$$

where the voltages are inputs, conductance/memristance is the weight, and the current is the output that needs to be validated with the threshold value I_{ref}. In this system, the output of a threshold gate depends on the weighted sum of its inputs. It is considered as high, only when this weighted sum is greater than a specified threshold as shown in Eqn. (4.4). From Fig. 4.4, one could observe that four memristors constitute a threshold block with each memristor circuit. For read and write operations, each of the four memristors have two voltage sources connected across it through a DPDT switch that is controlled by a pulse signal, similar to the case presented in Fig. 4.2(b). Apart from this, each memristive current was compared with the threshold value during read mode. If the current exceeds the threshold value, the output is logic *1*, else it is *0*. The four outputs from each of the threshold gates constitute 4-bit output from a threshold block for a given set of input pixel values. Individual shades of pixels were obtained at the outputs of threshold blocks from which *P* and *Q* (logic blocks) were assigned. The final *OR* operation on *P* and *Q* yielded the edge detected output. It was understood that memristance changes only when the read voltage exceeds the minimum memristive transition voltage i.e., *1.20 V* in this case, which is also *SET/RESET* voltage for the developed memristor [54].

In addition, threshold logic computations were accomplished through utilizing a common pulse width of 1 ms for write and read operations. For edge detection, once each of the memristor was written with a value, the voltage 1.10 V (less than the minimum transition voltage of the memristor, i.e., 1.20 V) was chosen to perform read operation (mentioned in Fig. 4.4(b)) which does not alter the memristance attained during the write operation. In order to capture black shade pixels, we compare the pixel values around the state x = 0.30. The value of 0.30 was chosen such that any value of x in the range 0 to 0.30 can be considered as black shaded.

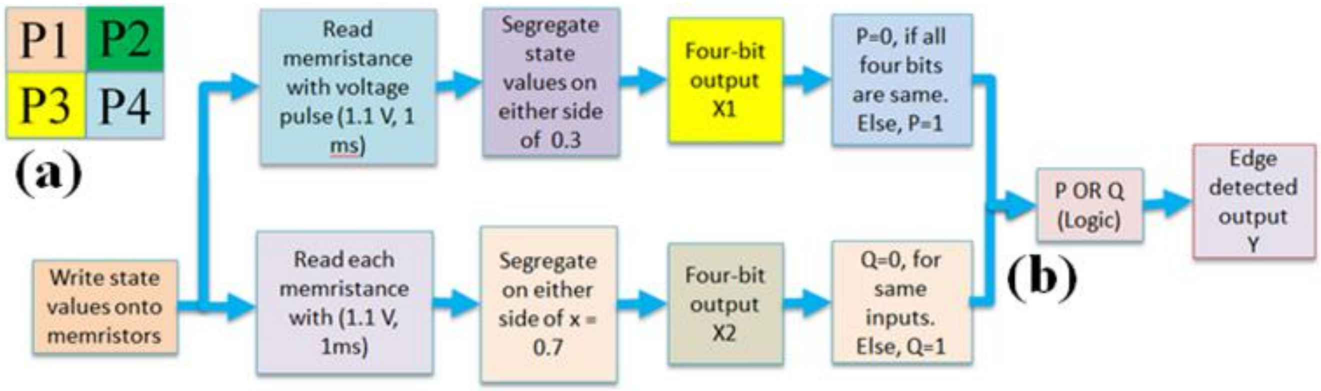

P1
P2
P3
P4
(a)
Write state values onto memristors
Read memristance with voltage pulse (1.1 V, 1 ms)
Segregate state values on either side of 0.3
Four-bit output X1
P=0, if all four bits are same. Else, P=1
Read each memristance with (1.1 V, 1ms)
Segregate on either side of x = 0.7
Four-bit output X2
Q=0, for same inputs. Else, Q=1
(b)
P OR Q (Logic)
Edge detected output Y

Fig 4.4: (a) The essential 2×2 pixel array for computing edge detection. The edges were detected based on the differences in intensities between the four neighbouring pixels. (b) Schematic of the computational flow through the threshold blocks. The edge detection was made possible through utilizing these logic computations.

This approach modulates the contrast of the image, which ultimately enhances edge detection. The threshold value for the above case was computed from the current flowing through the memristor during a read cycle, when the device is written with value 0.30. To obtain the voltage pulse magnitude ($V_{0.30}$) to write value 0.30, x = w/D was substituted in Eqn. (4.1) and integrated as given below:

$$\frac{\left[K_{off}\cdot\left(\frac{V_{0.30}}{V_{off}}-1\right)^{\alpha_{off}}\times(pulse\ width)\right]}{D} = x = 0.30 \tag{4.5}$$

$V_{0.30}$ was obtained by substituting the variables with the parametric values from [51]. This $V_{0.30}$ is essential to write a value 0.30 onto the memristor as shown in Eqn. (4.6). It is then substituted with parameters pulse width, K_{off} , α_{off} and V_{off} to find the value of $V_{0.30}$ as follows:

$$V_{0.30} = \left[\left(\frac{0.30\times D}{K_{off}\times(pulse\ width)}\right)^{1/\alpha_{off}} + 1\right].V_{off} = 1.3446\ V \tag{4.6}$$

The threshold current required to compare pixel values around the value 0.30 is computed as follows:

$$I_{0.30} = \frac{V_{read}}{M_{0.30}} = \frac{1.1\ V}{106\ M\Omega} = 0.010\ \mu A \tag{4.7}$$

where, $M_x = x.R_{ON} + (1-x).R_{OFF}$, as given in Eqn. (4.2). It is important to mention that the current flowing through the memristor would be greater than the threshold current $I_{0.30}$ if the write voltage (Vwrite) is greater than $V_{0.30}$. When the memristor was read in this manner, three different outputs possibilities exist, viz., (i) when all the pixel values are of black shade, the write voltages for all memristors will be less than 1.3446 V. This induces a current in each of the threshold gates that is lower than the threshold current during the read cycle. Hence, all the four bits of the output from the threshold block are 0 as threshold current is greater than induced current. (ii) if at least one pixel is black and at least one pixel has different shade (a gray or white, i.e. L/255 > 0.3), then during the read cycle, all the memristors that input dark shade pixels with write voltage less than 1.3446 V will have current less than the threshold value.

Further, the memristor that inputs a brighter pixel with write voltage greater than 1.3446 V will have current more than the threshold current value. For this case, among the 4-bits of the output from the threshold block, there would be both zeroes and ones, which corresponds to black and lighter shade pixels; and (iii) If none of the pixels are black then all the four outputs of the threshold block would be 1. If the four-bit output (each threshold block constitutes each bit) from this block is considered to be X1, then the final output from the block P is computed such that, if all the 4-bits in X1 are same then P is 0, else P is 1. Similarly, in order to capture white shade pixels, we compare the pixel values around the state x = 0.70. The value of 0.70 was chosen such that any value of x in the range 0.70 to 1 can be considered as white shaded. Here, the current induced during the read cycle was considered as the threshold value, when the memristor was written with value x = 0.70, where R_{off} is the initial resistive state. Similar to the previous case, the computations corresponding to the value x = 0.70 yielded $V_{0.70} = 1.3633\ V$ and $I_{0.70} = 0.024\ \mu A$, where $V_{0.70}$ is the voltage required to write value 0.70 onto the memristor and $I_{0.70}$ is the corresponding threshold current. In this case, during the read operation, three conditions were considered similar to the previous case, viz., (i) if all the pixels are darkly shaded (Vwrite < 1.3633 V and x < 0.7) then all four bits of the output from the threshold block is 0 as threshold current is greater than the induced current. (ii) If at least onepixel is white (Vwrite > 1.3633 V) and at least one pixel has different shade (Vwrite < 1.3633 V and x < 0.7) then among the 4-bits of the output from threshold block, there would be both zeroes and ones corresponding to darker and white shade pixels. Finally, (iii) if all the pixels are white (Vwrite > 1.3633 V and x > 0.7) then all the four outputs of the threshold block would be 1. If the four-bit output (each threshold block constitutes each bit) from this block is considered to be X2, then the final output from the block Q is computed as follows. If all the 4-bits in X2 are same, then Q output is 0, else Q is 1, where Q is a single bit output from the first threshold block.

The pixels that lie neither in the black region, nor in the white region, and has output of 0 from both the threshold blocks lie in this region. Finally, from the outputs of the threshold block, individual pixel shades were obtained. For non-edge region, either all the pixels must be white, gray or black. For all the pixels to lie in the same region (black, gray or white) all the four bits of the output must be the same for each of the threshold blocks. The final edge detected output (Y) is the OR operation between P and *Q*. If $Y = 1$ the result is as an edge and if $Y = 0$ then the region is not considered as an edge. The simulations for this memristive system for sample sets of pixel values were carried out, and the following edge detection results were obtained. As shown in Table 4.1, the 4-bit output *X1* contains *1's* for corresponding pixel values that are non-black shaded (*Luma value* > 77), and *0's* for black shaded pixels (*Luma value* < 77).

Similarly, the four-bit output *X2* contains *1's* for corresponding pixel values that are white shaded (*Luma value* > 179) and *0's* for dark (black or gray) shaded pixels (*Luma value* < 179). From *X1* and *X2*, *P* and *Q* were obtained and they take value *0* if all the corresponding four-bit outputs are same, else value *1* is chosen. The above computation was performed on a 5×5 sample image with the variations in *P1*, *P2*, *P3* and *P4* (Table 4.1 and Fig. 4.5) as the 2×2 window function (Fig. 4.4(a)) traverses throughout the image.

Table 4.1: Edge detected results for sample pixel inputs

S.No	**P1 [0,255]**	**P1 (V)**	**P2 [0,255]**	**P2 (V)**	**P3 [0,255]**	**P3 (V)**	**P4 [0,255]**	**P4 (V)**	**X1**	**P**	**X2**	**Q**	**Y**
1	154	1.360	136	1.357	129	1.356	136	1.357	1111	0	0000	0	0
2	136	1.357	142	1.358	136	1.357	135	1.357	1111	0	0000	0	0
3	142	1.358	138	1.358	135	1.357	146	1.359	1111	0	0000	0	0
4	138	1.358	138	1.358	146	1.359	228	1.369	1111	0	0001	1	1
5	129	1.356	136	1.357	140	1.358	144	1.359	1111	0	0000	0	0
6	136	1.357	135	1.357	144	1.359	144	1.359	1111	0	0000	0	0
7	135	1.357	146	1.359	144	1.359	252	1.372	1111	0	0001	1	1
8	146	1.359	228	1.369	252	1.372	217	1.368	1111	0	0111	1	1
9	140	1.358	144	1.359	132	1.357	128	1.356	1111	0	0000	0	0
10	144	1.359	144	1.359	128	1.356	243	1.371	1111	0	0001	1	1
11	144	1.359	252	1.372	243	1.371	239	1.371	1111	0	0111	1	1
12	252	1.372	217	1.368	239	1.371	202	1.366	1111	0	1111	0	0
13	132	1.357	128	1.356	163	1.362	2	1.286	1110	1	0000	0	1
14	128	1.356	243	1.371	2	1.286	231	1.370	1101	1	0101	1	1
15	243	1.371	239	1.371	231	1.370	231	1.370	1111	0	1111	0	0
16	239	1.371	202	1.366	231	1.370	246	1.371	1111	0	1111	0	0

Each iteration mentioned in the graph (Fig. 4.5) corresponds to each position of the window in the image (and a row in Table 4.1). For these corresponding *P* and *Q* values, final edge detected output was computed. It is noteworthy to mention that the P2 location of the window was substituted by an edge, whenever an edge was detected [156]. Further, this algorithm was tested for grayscale images with the help of *MATLAB* code that computes current for given sets of pixels and threshold values through utilizing Eqn. (4.4) to Eqn. (4.7).

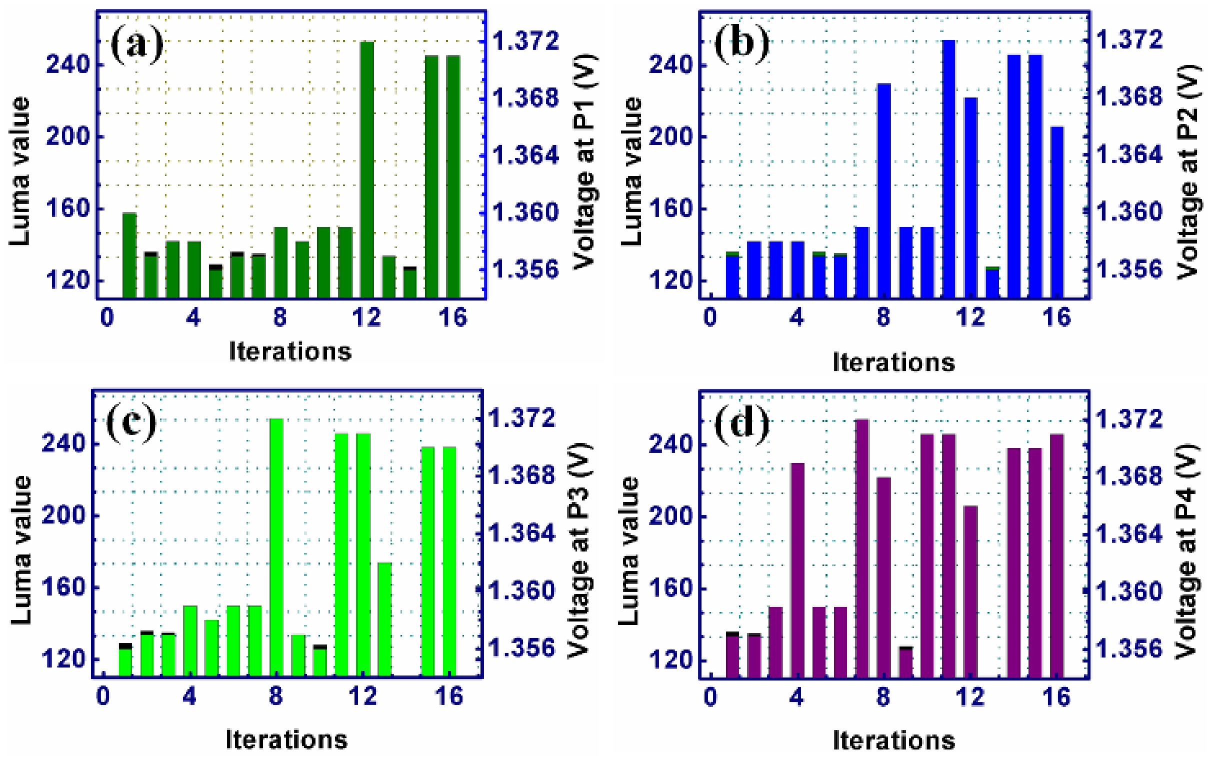


Fig. 4.5: Variation of pixel grayscale intensity with the number of computational iterations for pixels P1-P4 shown in (a)-(d). Each iteration corresponds to a 2 × 2 submatrix in the sample 5 × 5 image, which traverses throughout the image.

The obtained results for different grayscale images are shown in Fig. 4.6. One could notice that all edges were clearly detected and this accurate detection is owing to the fact that our proposed techniques enhance the image contrast, which further increases the pixels' intensity difference [78]. Since it deals with the analog signal, no further analog-to-digital conversion is required [157], and the proposed system could be used directly for edge detection based on the signals it received from the optical sensor. The obtained accuracy can further be enhanced through the increase in pixel classification ranges by utilizing a large number of threshold blocks and tuning the grayscale values. Real time analog signals are always prone to voltage noise; however, circuit systems were designed to make them immune to external noises. To analyze the noise tolerance for our circuit, we obtain the DC noise voltage to cause a logic error of 1% (Δx=0.01). While writing or reading logic x = 0.30 or 0.70, the analysis is as shown below.

For the case x = 0.30 and Δx = 0.01: from Eqn. (4.5), the V_N was calculated to be 0.91 mV. Similarly for the case x = 0.70 and Δx = 0.01, the V_N was 0.80 mV. Therefore the proposed circuit design had tolerance to DC noise approximately upto 1 mV, if the tolerance was considered to be Δx = 0.01. Moreover, the SET/RESET voltages were found to be very low ∼1.40 V, which offers adoptability for low-power device applications establishing the memristor as a reliable part of integrated circuits in terms of aging [53].

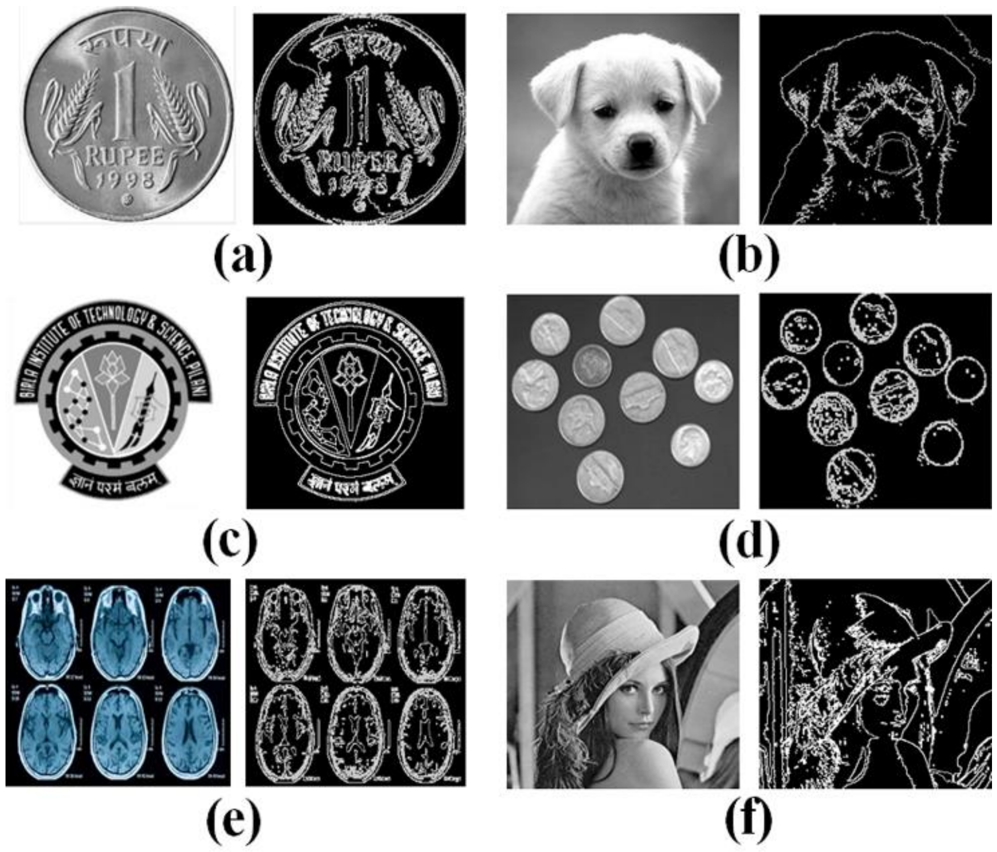


Fig. 4.6: (a)-(f) Images indicating edge detection. The proposed algorithm was able to successfully detect the edges in given images.

In this regard, it is important to mention that a large variation of memristance in the order of 100 Ω resulted in a low programming error of less than 1%. Hence, it was understood that the wire resistance and variability of the conductance had no significant effect on the performance of the proposed system.

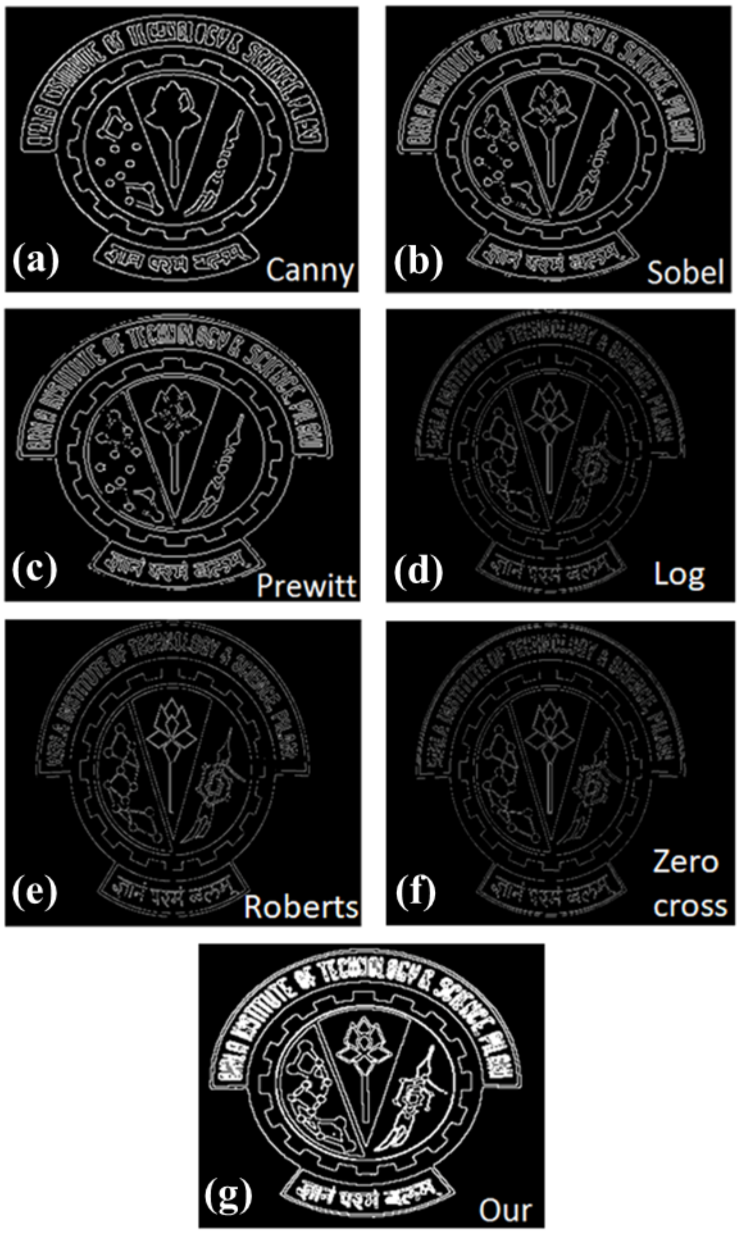


Fig. 4.7: Edge detection through utilizing (a) Canny's method, (b) Sobel's method, (c) Prewitt's method (d) Log method, (e) Roberts method, (f) Zerocross method and (g) our proposed method. It can be seen that highest number of edges were detected by the proposed method, when compared to the other cases.

The proposed prototype presents edge detection based on identifying the class of pixels (white, black or gray) and application of rule based computation. This system could also be extended to increase the class of pixels (more than 3 classes white, black and gray) through utilizing more than 2 threshold blocks in order to detect edges of variable sharpness. For example, as two threshold blocks were able to classify pixels into three categories, three threshold blocks could be used to classify pixels into four categories and so on depending upon the sharpness of the edges one needs to detect.

Further, in order to quantify the quality of edge detection, the proposed method was compared with the existing conventional and popular *Canny, Sobel, Prewitt, Log, Roberts and zerocross* methods (shown in Fig. 4.7) [143], [151], [152]. These algorithms rely on conventional hardware implementations for processes like convolution and filtering [148], [158]. From the Fig. 4.7, one could observe that a large number of edges were detected in our case when compared with the other techniques.

Table 4.2: Percentage of edges detected in different algorithms

Algorithm	**Edges detected**
Canny	8.68%
Sobel	8.60%
Prewitt	8.60%
Log	8.47%
Roberts	9.82%
Zerocross	9.73%
Our proposed work	16.71%

Further, the percentages of edges detected by each algorithm are tabulated in Table 4.2. It can be observed that the edge detection in our case was 16 %, whereas in the remaining cases, it was ~8.60 %. This improvement is due to the dual threshold (0.30 and 0.70, as mentioned earlier) approach adapted in our case. The proposed techniques are advantageous in terms of computational complexity as there is no necessity for the conventional convolution operation utilized in filter based approaches. This is owing to the fact that the convolution operation deals with a large amount of data transfer and requires numerous adders, multipliers and other digital circuitry [52], [53]. With ever growing demand for edge detection in futuristic complex systems and applications, there is a need for an efficient system. In the recent trend there have been works pertaining to accelerate existing methods [159], [160] or seek for an efficient algorithm different from the conventional implementations. One such optimal computation algorithm that researches have opted is the fuzzy logic [151]. Additionally with the advent of recent non-volatile Cu:ZnO memristor, the device has attributed to build a simpler system and the current work has proven to be a promising alternative for edge detection given its low complexity and efficiency.

Importantly, when compared to the recent work on memristor based state machines [161], the proposed MSM is advantageous (*87.5%* improvement) in terms of number of memristive components. It is important to mention that the methods proposed in this work have been implemented through utilizing the models of different memristors fabricated using type-2 fuzzy logic in [150], [154], [155]. The main aspect that makes this system stand apart from past implementations is that the computations are all carried out in the analog domain and the requirement of fuzzifier and de-fuzzifier logics are ruled out with the use of non-volatile memory device unlike for the cases of conventional CMOS devices.

4.4 Summary

In this work, a large number of stable resistive states were identified in a memristor. Efforts were devoted to utilize the transitions between these states for logic computations. Based on this understanding, MSM was implemented and it was applied for edge detection of images. The detection was performed based on the classification of pixels (white, black or gray) through utilizing if-then rules. The approach yielded improvements in terms of storage density and computational capabilities. Tune-ability in the features of edge detection was made possible through adjusting the threshold memristance within the available memristive states. Since memristor is the key element in this network and deals with the analog signal, the proposed system operates much faster as compared to other existing approaches with reduced circuitry to implement the network since analog-to-digital conversion would not be required. The MSM based edge detection system produced high accuracy in edge detection through increasing the pixels' intensity difference. In order to quantify the role of the proposed system, it was compared with other conventional approaches such as Canny, Sobel, and Prewitt's systems. The system proposed in work showed much better performances when compared to the other abovementioned systems. Further, the proposed techniques could be extended for multi-classification of pixels, by utilizing multiple threshold blocks in order to detect edges of variable sharpness for futuristic smart image processing applications. Memristors with improved parameters such as R_{on}/R_{off} ratio or bandwidth could be researched in order to increase the immunity of the device to noise and parasitic effects.

Remarks: Below mentioned paper was published based on this chapter.

- **Priyanka B. Ganganaik,** G. Abhijith, P. Michael Preetam Raj, H. Renuka, BVVSN Prabhakar Rao, and Souvik Kundu. "Realization of Memristive State Machine for Smart Edge Detector Applications." IETE Journal of Research (2020): 1-11.

Chapter-5

Implementation of Binary Particle Swarm Optimization for Image Thresholding using Memristor Crossbar Array

5.1 Introduction

Swarm Intelligence based algorithms are increasingly replacing traditional search algorithms with high time complexity. These algorithms are modeled after various natural phenomena such as ant colonies, bee hives, bird flocks, fish shoals, animal herds, etc. [162]. Particle Swarm Optimization (PSO) [163] is considered as one of the most widely used swarm intelligence algorithms, which makes very few approximations about the function that needs to be optimized, and they can search a large search space for an optimal solution. No information is required about the gradient of the objective function; therefore, even non-differentiable functions can be optimized using PSO. PSO has applications in various domains such as multi-objective function optimization [164], geotechnical engineering [165], solar energy systems [166], improving and optimizing neural network performance [167], [168], and image segmentation [17], [169], [170].

On the other hand, image thresholding is one of the fundamental methods for image segmentation based on similarity in features within the regions. Among various ways of image thresholding for segmentation, the two most successful methods are Otsu's method and Kapur's Entropy method. Grayscale image pixel intensities are generally saved as 8 (or integral multiples of 8) bit integers. Thus, discrete optimal thresholds are desired. The thresholds are 8-bit integers as pixel intensities range from 0 to 255. As image thresholding involves finding optimal thresholds from a discrete search space of thresholds, Binary Particle Swarm Optimization (BPSO) [171], a variant of PSO, is better suited for this application. Hence, finding them by employing BPSO would be more prudent when compared to any other continuous variant of PSO [172].

Although swarm intelligence-based algorithms are much faster than conventional search algorithms, their speed can further be improved significantly. The performace of these computationally intensive algorithms are limited by the drawbacks of currently used Von Neumann architecture-based computing systems as discussed in the earlier chapters. Thus, in terms of the demand for in-memory computing systems, memristors have been demonstrated to be a possible key component in designing such systems due to their unique physical qualities such as low energy consumption, nanoscale size, sub-nanosecond switching speed, and extended memory [173]–[176]. Several memristor architectures, primarily memristive crossbar arrays, have been employed in neuromorphic circuits such as synapses in spiking neural networks [43], applications in chaotic theory [177], and also for implementing machine learning algorithms like the Principle Component Analysis [178]. Recently, very few studies have tried implementing swarm intelligence-based algorithms using memristors. A memristor crossbar-based implementation of Ant colony optimization [179], another widely used swarm intelligence-based algorithm, was used in [176], [180] for edge detection. A modified variant of Particle Swarm Optimization, CPSO (Chaotic Particle Swarm Optimization), has been used to initialize parameters of a neural network [181].

BPSO has proven to be a very effective algorithm, especially for optimizing functions having their domain defined on discrete search spaces. In this work, BPSO for the image multi-thresholding problem using a novel Pt/Cu:ZnO/Nb:STO memristor crossbar array is implemented. To the best of our knowledge, this is the first study pertaining to the implementation of BPSO on the memristor crossbar. Otsu’s Function and Kapur’s Entropy Function are considered as the objective functions or the functions left for BPSO to optimize. It is hypothesized that results obtained from the implementation of BPSO on the memristor crossbar are at par and very close to the optimal thresholds. Further, this (Otsu and Kapur's entropy with BPSO) is applied for thresholding 4 T2- weighted transaxial brain Magnetic Resonance Imaging (MRI) scans of the widely used online open-access medical image repository by Harvard Medical School [182].

This chapter is organized as follows: In section 5.2, Otsu and Kapur's Entropy Functions are elaborated and brief description of the BPSO algorithm is given. In section 5.3 Implementation of BPSO using memristor crossbar is described.. In section 5.4, results for multi-thresholding on standard as well as the Brain MR images, followed by the results with device variations, are explained. Conclusions are given in section 5.5.

5.2 Objective Functions

In Swarm intelligence, the functions optimized by the swarm intelligence algorithm (BPSO in this case) are known as Objective Functions for the image thresholding problem. Our aim is to divide the pixels in an image I into N groups of pixels which has similar gray levels. It is done by finding N-1 optimal intensity thresholds. It is noted here that t_0 corresponds to the minimum possible intensity (0 intensity), t_1 to t_{N-1} are the N-1 optimal thresholds and t_N is the maximum possible intensity (255 in case of 8-bit gray-scale images). The swarm intelligence algorithm assigns all pixels having pixel intensity in the range between threshold t_i and threshold t_{i+1} to the gray-level value of G_i such that i can take integer values between 0 and N. It is mathematically represented as follows:

$$G_i = \{f(x,y) \in I | t_i \leq f(x,y) < t_{i+1}\} \tag{5.1}$$

Here, $f(x,y)$ is the integer pixel intensity of image I at position (x,y). So, as seen in Eqn. (5.1), all pixels with intensities in the interval [t_i, t_{i+1}-1] are associated with Group G_i where i is an integer between 0 and N-1. To find the optimal thresholds t_1 to t_{N-1}, Kapur's Entropy function [20] and Otsu's thresholding function [183] are used. These objective functions take the image grayscale histogram and a set of N-1 thresholds as input. The image grayscale histogram is a discrete graph of the number of pixels vs. pixel intensity. The swarm intelligence algorithm uses the objective functions to adjust the thresholds for optimal results.

5.2.1 Kapur’s Entropy Function

Kapur’s Entropy function is a widely used function for thresholding. It is based on discrete entropy, which is the measure of unpredictability or the degree of randomness. In information theory, entropy (often referred to as Shannon’s Entropy) is mathematically defined as [184], [185]:

$$H(X) = -\sum_{i=0}^{n} P(x_i)log\big(P(x_i)\big) \tag{5.2}$$

Where X is a discrete random variable which can take n discrete values. x_i corresponds to the i^{th} value taken by X. P(x_i) is the probability of X assuming the value x_i. It is known that for a discrete random variable X if the Entropy of X is to be maximized, it has to follow a finite uniform Probability Mass Function (PMF), which has the maximum entropy. The swarm intelligence algorithm tries to maximize Kapur’s entropy function for the image thresholding application so that the modified histogram after thresholding is very close to a discrete uniform PMF. Hence, the contrast is maximized for the thresholded image. As it is known that Kapur’s Entropy function is a histogram-based thresholding function, let h(j) be the number of pixels in the image I with intensity “j”. For a particular pixel intensity j, the probability that a pixel in Image I has intensity j is:

$$p_j = \frac{h(j)}{\sum_{l=L(min)}^{L(max)} h(l)} \tag{5.3}$$

Where $L(min)$ and $L(max)$ are the minimum and maximum possible intensities in the image I. For gray-scale 8-bit images, $L(min)$ is 0, and $L(max)$ is 255 $(2^8 - 1)$. The expressions for Kapur’s Entropy for multi-thresholding from Eqn. (5.2) and Eqn. (5.3) is given in [18] is as follows:

$$H_i = -\sum_{j=t_i}^{t_{i+1}-1} \frac{p_j}{\omega_l} \ln\left(\frac{p_j}{\omega_l}\right);\ \omega_i = \sum_{j=t_i}^{t_{i+1}-1} p_j \tag{5.4}$$

It is noticed that Eqn. (5.4) is similar to Eqn. (5.2) where $\frac{p_j}{\omega_l}$ can be thought of as P(xj) where l is an integer between 0 to N-1 and j is a particular pixel intensity. The objective function is then just the summation of all the entropies, which is given below:

$$f_{Kapur}(I, \boldsymbol{t}) = \sum_{i=0}^{N-1} H_i \tag{5.5}$$

Where threshold $\boldsymbol{t} = [t_1, \dots t_i, \dots t_{N-1}]$ and I is the image concerned. Our goal is to find optimal threshold $\boldsymbol{t}^*$ using the swarm intelligence algorithm such that $\boldsymbol{t}^*$maximizes the Kapur's Entropy function or in other words, $f_{Kapur}(I, \boldsymbol{t}^*)$ is the maximum value of Kapur's Entropy Function.

5.2.2 Otsu's Function

Otsu's function is based on the concept of reducing intra-class variance and increasing inter-class variance. This simply means that in a group or gray level, say G_i, almost all pixels are homogenous and almost in the same range; however, for two adjacent groups or gray levels G_i and G_{i+1}, the pixels are drastically different. The within-class or intraclass variance is defined as:

$$\sigma_i = \omega_i(\mu_i - \mu_T)^2 \tag{5.6}$$

Where ω_i is the normalized probability of the pixel intensities, μ_i is the mean level of each class G_i and μ_T is the mean intensity of the Image I. μ_i is given by:

$$\mu_i = \frac{\sum_{j=t_i}^{t_{i+1}-1} j p_j}{\omega_i} \tag{5.7}$$

The Otsu function is defined as:

$$f_{Otsu}(I, \boldsymbol{t}) = \sum_{i=0}^{N-1} \sigma_i \quad (5.8)$$

Again, the goal of this work here is to find $\boldsymbol{t}^*$ such that it maximizes the Otsu's function. It is observed that if brute force method is used, that is, by manually trying out all possible unordered pairs of N thresholds, the time complexity would be $O(n^N)$. Although this is an effective technique for bi-thresholding, that is when N=1. As N increases, there is a dire need for faster techniques, such as swarm intelligence algorithms, to choose the N thresholds.

5.2.3 Binary Particle Swarm Optimization (BPSO)

Particle Swarm Optimization is a Swarm Intelligence Algorithm proposed by Kennedy and Eberhart in 1995 [18]. It is modeled after the social behavior of a flock of birds or a shoal of fish. It is used for optimizing objective functions. The domain on which the objective function is defined is referred to as the search space. A swarm of agents or particles are initialized as random points in the search space. Each of these particles can be represented by its current position, its velocity, and the personal best position the particle has attained till that point of time. The position and the velocity have dimensions equal to that of the dimension of the domain. Each particle makes its next move by updating velocity followed by position based on the best position attained by any of the particles till that point of time or the global best position and its personal best position. The velocity and position are updated in Eqn. (5.9) and Eqn. (5.10) respectively as given below:

$$\boldsymbol{v}_i^{t+1} = W\boldsymbol{v}_i^t + c1\boldsymbol{r1}.\left(\boldsymbol{x}_{gb}^t - \boldsymbol{x}_i^t\right) + c2\boldsymbol{r2}.\left(\boldsymbol{x}_{i,pb}^t - \boldsymbol{x}_i^t\right) \quad (5.9)$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \quad (5.10)$$

Where $\boldsymbol{v_i}$, $\boldsymbol{x_i}$ and $\boldsymbol{x_{i,pb}}$ are the velocity, position, and personal best position reached so far by the 'i'th particle respectively. $\boldsymbol{x_{gb}}$ is the global best position of the swarm, and W is a scalar often referred to as the momentum weight. The constant scalars c1 and c2 are referred to as the acceleration coefficients. **r1** and **r2** are two variable vectors of the same dimension as that of the search space. Each element of these vectors is between 0 and 1, which is randomly updated after every iteration. The personal best and the global best are decided depending on the output of a function which inputs the current position vector of a vector and outputs a scalar. This function is referred to as the fitness function. This fitness function is usually the objective function itself, although they can differ.

For discrete-valued search spaces, Kennedy and Eberhart came up with a discrete binary variant of PSO, the Binary Particle Swarm Optimization (BPSO) [18]. Here the positions of the particles are in discrete binary form. The particle velocities, however, are of continuous nature. The following is how the particle velocity and position updating in BPSO take place:

$$\boldsymbol{v}_{\boldsymbol{i}}^{\boldsymbol{t+1}} = W\boldsymbol{v}_{\boldsymbol{i}}^{\boldsymbol{t}} + c1\boldsymbol{r1}.\left(\boldsymbol{x}_{\boldsymbol{gb}}^{\boldsymbol{t}} - \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{t}}\right) + c2\boldsymbol{r2}.\left(\boldsymbol{x}_{\boldsymbol{i,pb}}^{\boldsymbol{t}} - \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{t}}\right) \tag{5.11}$$

For every dimension of the position and velocity vectors,

$$x_{i,d} = \begin{cases} 1, if\ sigmoid(v_{i,d}) > r \\ \quad 0, otherwise \end{cases} \tag{5.12}$$

Here, r is a random number between 0 to 1 which is updated after every iteration. $x_{i,d}$ is the position vector and $v_{i,d}$ is the velocity vector with respect to the dimension ‘d’. It is noted that in Eqn. (5.11) and Eqn. (5.12), the current and personal best positions of the particles, as well as the global best position are all discrete binary vectors. That is, all the elements in these vectors are either 0 or 1.

So, to essentially formulate the problem of N-level gray-scale image thresholding, the search space can be thought of as an 8xN dimensional binary discrete vector space. As stated earlier, each threshold is in the range of 0 to 255. To encode the decimal number in binary space, 8 bits for one threshold is needed. So, for N thresholds, an 8xN dimensional vector space is required. So, one particle corresponds to an 8xN dimensional vector or N thresholds. At the termination of the algorithm, the global best position, which corresponds to the N thresholds, is considered as the optimal solution and the N thresholds corresponding to this global best position are the optimal thresholds. In the next section, BPSO using a memristor crossbar is implemented with both Kapur's Entropy Function and Otsu's Function as the objective function and fitness function.

5.3 Implementation Section

5.3.1 Implementation of BPSO using Memristor Crossbar

As matrix multiplications are the key operations in most of the AI-based algorithms, implementing them using the memristor crossbar tackles the von Neumann Bottleneck faced by these algorithms. BPSO involves many matrix-vector dot products, and the use of memristor crossbars can improve the speed of the algorithm.

Discrete binary particles can be modeled using a memristor crossbar array, as shown in Fig. 5.1(a). A particle has a discrete binary position corresponding to 8N bits, where N corresponds to the number of thresholds. It also has its personal best position, which is also represented using 8N bits. A design of a memristor crossbar having (2Ns+1) rows and 8N columns is proposed, where Ns is the number of particles. Here, 2 rows correspond to one particle, which stores the current position and the personal best position of the particle, respectively. The last row is used to store the global best position. Each memristor represents one bit which can be logic state 0 if it is at a high resistance state or logic state 1 at a low resistance state. The particle positions are initialized randomly. This is done by applying v_{set} or v_{reset} voltages across each memristor individually. Reading voltages are considered as 0.5 V and -0.5 V as both lie between v_{on} (-1.2V) and v_{off} (+1.35V). From chapter 4, Eqn. (5.1), w does not change for this voltage. Hence, the memristor behaves as an ideal resistor for these reading voltages. For BPSO, 2 crossbars are simulated for finding 2 thresholds and 3 thresholds. The optimal parameters for BPSO were decided after multiple runs of the algorithm with varied parameters. The proximity of the value of the objective function is taken into consideration at the output thresholds to the maximum possible value of the objective function. The following Table 5.1 summarizes the parameters for BPSO and information about the crossbars:

Table 5.1: Parameters for BPSO and information about the crossbars used as suggested in the pseudo-code in section 5.2.3 and Eqn. (5.11)

Parameter	**N=2**	**N=3**
Number of particles (Ns)	50	100
Number of iterations	50	50
W	1.0	1.0
c1	2.0	2.0
c2	2.0	2.0
Total number of memristors	1,616	4,824
Dimensions of crossbar	101x16	201x24
Area of the crossbar	$1.309\text{x}10^{-11}$ m^2	$3.907\text{x}10^{-11}$ m^2

For the memristor model, the read time is 10 ns. As c1 and c2 are 2 each, all the elements of c1**r1** and c2**r2** will be in the range 0 to 2. Positions are modeled as conductances, whereas c1**r1** and c2**r2** are modeled as voltage pulses. A voltage pulse of magnitude equal to 0.5 V (read voltage) and a random time duration of 10*k ns is applied, where $k \in \{1,200\}$ to the last row or the global best position. This, in turn, ensures that all the elements of c1**r1** and c2**r2** have a precision of only 2 significant digits after the decimal point, and each element has a least count of 0.01. Ideally, the random numbers should have as much precision as possible, however, here, there is a tradeoff as the maximum pulse width increases 10 folds with an increase in the number of significant digits after the decimal point, and this slows down the implementation drastically. Simultaneously a voltage pulse with magnitude -0.5 V and the same pulse width as the one above to the row storing the current position of the particle is applied. Further, a charge amplifier is used, which will give a voltage output proportional to the charge or integral of the current at the input of the amplifier. This voltage vector, which is the output of the charge amplifiers, corresponds to $c1\boldsymbol{r1}.\left(\boldsymbol{x}_{gb}^{t}-\boldsymbol{x}_{i}^{t}\right)$ in Eqn. (5.11). This is depicted in Fig. 5.1(b). Here, all the random elements of **r1** have been considered equal, which are generated newly for different particles every iteration.

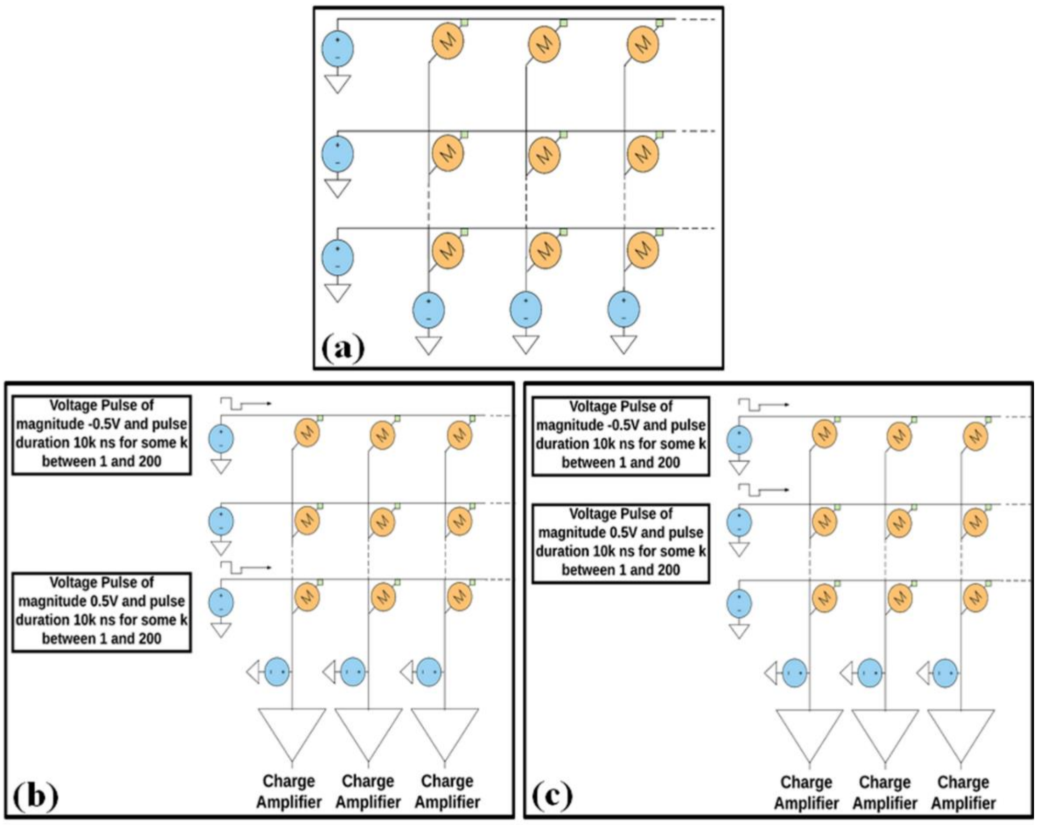
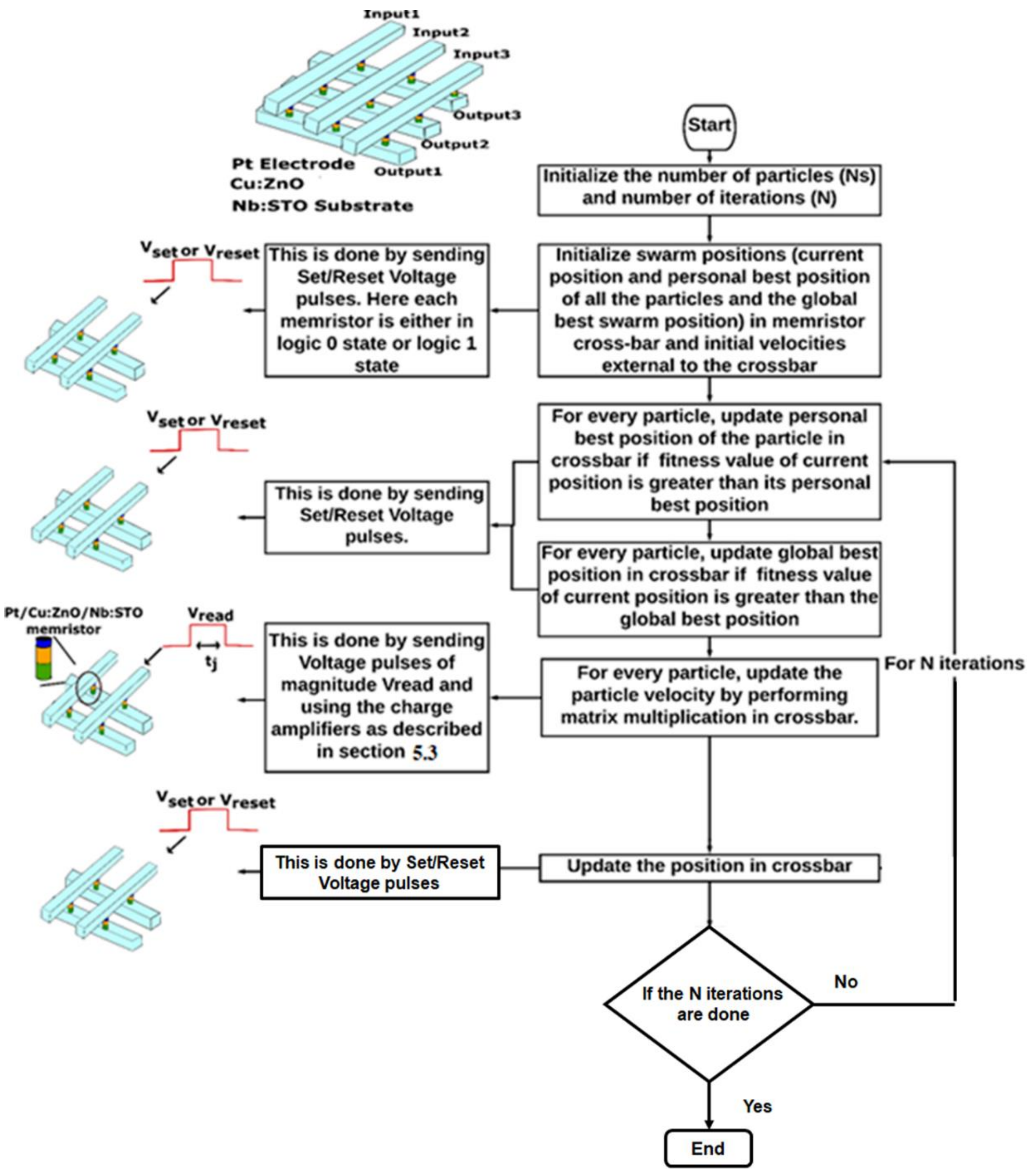

(a)
Voltage Pulse of magnitude -0.5V and pulse duration 10k ns for some k between 1 and 200
Voltage Pulse of magnitude 0.5V and pulse duration 10k ns for some k between 1 and 200
Charge Amplifier
Charge Amplifier
Charge Amplifier
(b)
Voltage Pulse of magnitude -0.5V and pulse duration 10k ns for some k between 1 and 200
Voltage Pulse of magnitude 0.5V and pulse duration 10k ns for some k between 1 and 200
Charge Amplifier
Charge Amplifier
Charge Amplifier
(c)

Fig. 5.1: (a) Memristor Crossbar Array (b) Calculating $c1\boldsymbol{r1}.\left(\boldsymbol{x}_{gb}^{t}-\boldsymbol{x}_{i}^{t}\right)$ in crossbar (c) Calculating $c2\boldsymbol{r2}.\,(\boldsymbol{x}_{i,pb}^{t}-\boldsymbol{x}_{i}^{t})$ in crossbar.

Now a voltage pulse of 0.5 V magnitude and pulse width is chosen randomly between 10ns and 2µs is sent to the second row (the row storing the personal best position), and simultaneously, another pulse of the same width but magnitude equal to -0.5V to the first row (the row storing the current position). Here, the output of the charge amplifiers corresponds to $c2\boldsymbol{r2}.\,(\boldsymbol{x}_{i,pb}^{t}-\boldsymbol{x}_{i}^{t})$ in Eqn. (5.11), and again, all elements of **r2** are assumed to be equal, which is shown in Fig. 5.1(c).



Input1
Input2
Input3
Output3
Output2
Output1
Pt Electrode
Cu:ZnO
Nb:STO Substrate
Start
Initialize the number of particles (Ns) and number of iterations (N)
Initialize swarm positions (current position and personal best position of all the particles and the global best swarm position) in memristor cross-bar and initial velocities external to the crossbar
Vset or Vreset
This is done by sending Set/Reset Voltage pulses. Here each memristor is either in logic 0 state or logic 1 state
For every particle, update personal best position of the particle in crossbar if fitness value of current position is greater than its personal best position
Vset or Vreset
This is done by sending Set/Reset Voltage pulses.
For every particle, update global best position in crossbar if fitness value of current position is greater than the global best position
Pt/Cu:ZnO/Nb:STO memristor
Vread
tj
This is done by sending Voltage pulses of magnitude Vread and using the charge amplifiers as described in section 5.3
For every particle, update the particle velocity by performing matrix multiplication in crossbar.
For N iterations
Vset or Vreset
This is done by Set/Reset Voltage pulses
Update the position in crossbar
If the N iterations are done
No
Yes
End

Fig. 5.2: Flowchart summarizing the entire procedure.

Thus, for the j^{th} dimension of the i^{th} particle, from Eqn. (5.18) of chapter 1:

$$Q_j = v_{read} * k * t_{read} * (g_{gb,j} - g_{i,j}) \tag{5.13}$$

$$Q_j = v_{read} * k * t_{read} * (g_{pb\_i,j} - g_{i,j}) \tag{5.14}$$

Here, Q_j is the output voltage at the j^{th} column/dimension. $g_{gb,j}$ is the j^{th} dimension of the global position and $g_{pb\_i,j}$ and $g_{i,j}$ are the j^{th} component of the personal best and current best positions of the i^{th} particle, respectively.

It is noted here that as R_{on} (of the order 1.2 kΩ) is much lesser than R_{off} (of the order of 1.2 MΩ), conductance corresponding to the logic 1 state (g_{on}=1/ R_{on}) is of the order of 1 $\mu\Omega^{-1}$ and the conductance corresponding to the logic 0 states (g_{off}=1/ R_{off}) is of the order of 1 $n\Omega^{-1}$ thus, the following approximation is used:

$$|g_{on} - g_{off}| \approx |g_{on}| \tag{5.15}$$

Further, the net velocity is calculated from Eqn. (5.11). This part is done using an 8N bit adder circuitry external to the crossbar. The position is updated as shown in Eqn. (5.12) by applying V_{set} / V_{reset} pulses for sufficient duration of time. This is followed by updating the personal best and global best positions by checking the fitness value. As the proposed architecture is a coprocessor, the crossbar is strapped onto the CPU using the PCI bus. It is noted here that the fitness values for different positions are calculated in the CPU (outside the crossbar). The flowchart given in Fig. 5.2 summarizes the entire process.

5.4 Results and Discussions

5.4.1 Results and Explanation for Standard Images

As mentioned in Section 5.3, two memristor crossbar arrays were simulated for 3 gray-level thresholding and 4 gray-level thresholding, respectively. The histogram of the normalized number of pixels (normalized frequency) vs. pixel intensity was obtained from gray-scale images. In this study, the results are explained using the ubiquitous “Lena” Image (225 × 225) shown in (Fig. 5.3(a)) and 4 T2- weighted transaxial brain MRI scans shown in (Fig. 5.6(a-d)), which were randomly selected from the online open-access medical image repository from Harvard Medical School [182].

Brute-force optimization is used on the Kapur and Otsu Functions, considering unordered pairs of 2 and 3 intensity thresholds ranging from 0 to 255. The results for the same are shown in

Fig. 5.3(b)-(e). For the 2 threshold-based thresholding, a 101×16 crossbar is initialized with V_{set} and V_{reset} pulses as suggested in Fig. 5.2. Fifty particles or 50 of the 65,280 possible domain points were arbitrarily considered where the number of iterations was 50.

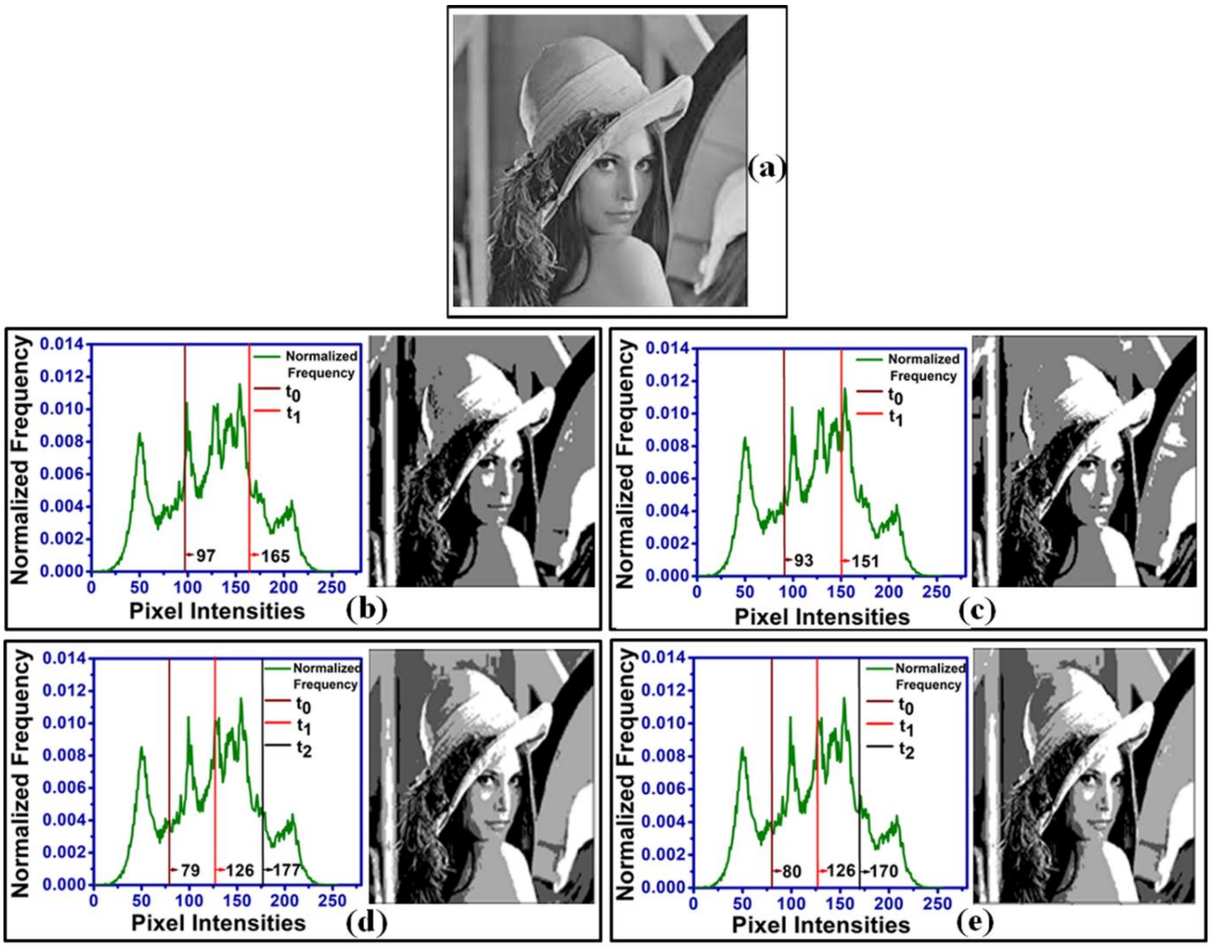


Fig. 5.3: Brute Force Optimization. (a) Gray-scale Lena Image (b) Histogram and Image for 2-thresholds while optimizing Kapur's Entropy Function, (c) Histogram and Image for 2-thresholds while optimizing Otsu's Function, (d) Histogram and Image for 3-thresholds while optimizing Kapur's Entropy Function, (e) Histogram and Image for 3-thresholds while optimizing Otsu's Function.

Kapur's entropy function was considered as the objective function as well as the fitness function. At the termination of the algorithm, the thresholds corresponding to the global best position were considered as the optimal thresholds. The same was repeated considering Otsu's Function as both the objective function and the fitness function. For 3 threshold-based thresholding, another crossbar was simulated with dimensions 201×24. In this case, the number of particles was taken as 100 or 100 of the 16,581,120 possible domain points that were arbitrarily considered. The number of iterations was again taken as 50. A similar process was repeated, taking Kapur's Entropy Function and Otsu's Function as both the objective and fitness functions. The results for the same have been depicted in Fig. 5.4(a)-(d).

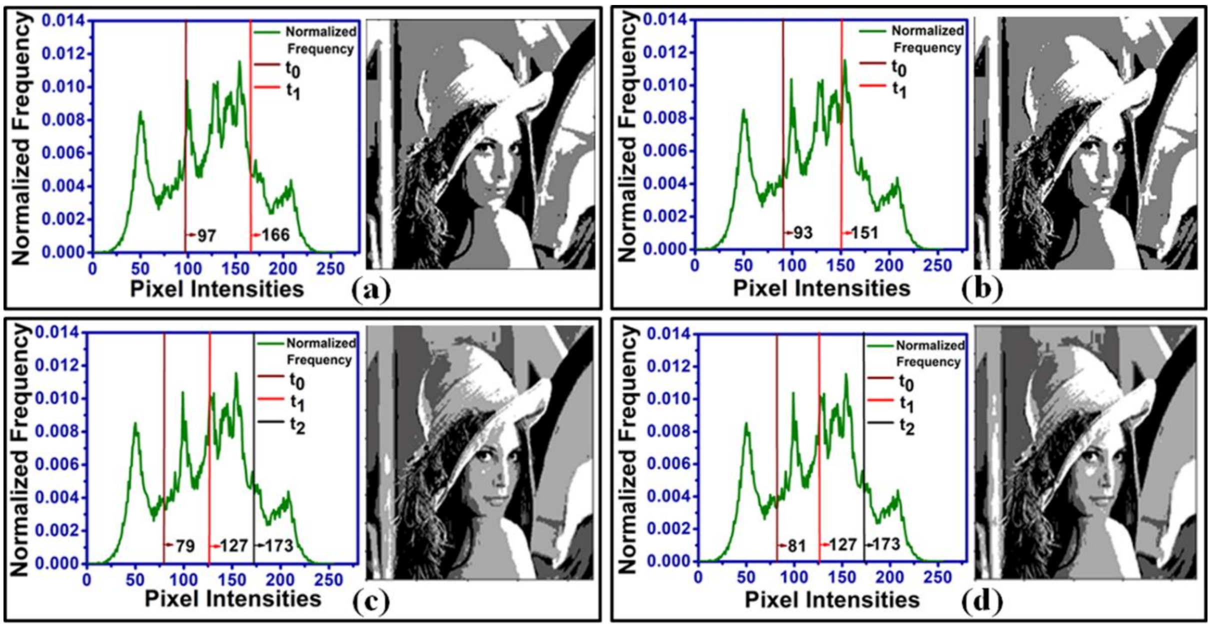


Fig. 5.4: BPSO Implementation using memristor crossbar. (a) Histogram and Image for 2-thresholds while optimizing Kapur's Entropy Function, (b) Histogram and Image for 2-thresholds while optimizing Otsu's Function, (c) Histogram and Image for 3-thresholds while optimizing Kapur's Entropy Function, (d) Histogram and Image for 3-thresholds while optimizing Otsu's Function.

To consider device variations, it is assumed that the High impedance (R_{off}) and Low impedance states (R_{on}) follow a Gaussian distribution with the mean as the designated values and standard deviation equal to 5% of the mean value. For each simulated memristor in both the crossbars, R_{on} and R_{off} were taken randomly from the corresponding distributions, and the entire process was repeated for both crossbars. The results for the same are shown in Fig. 5.5(a)-(d).

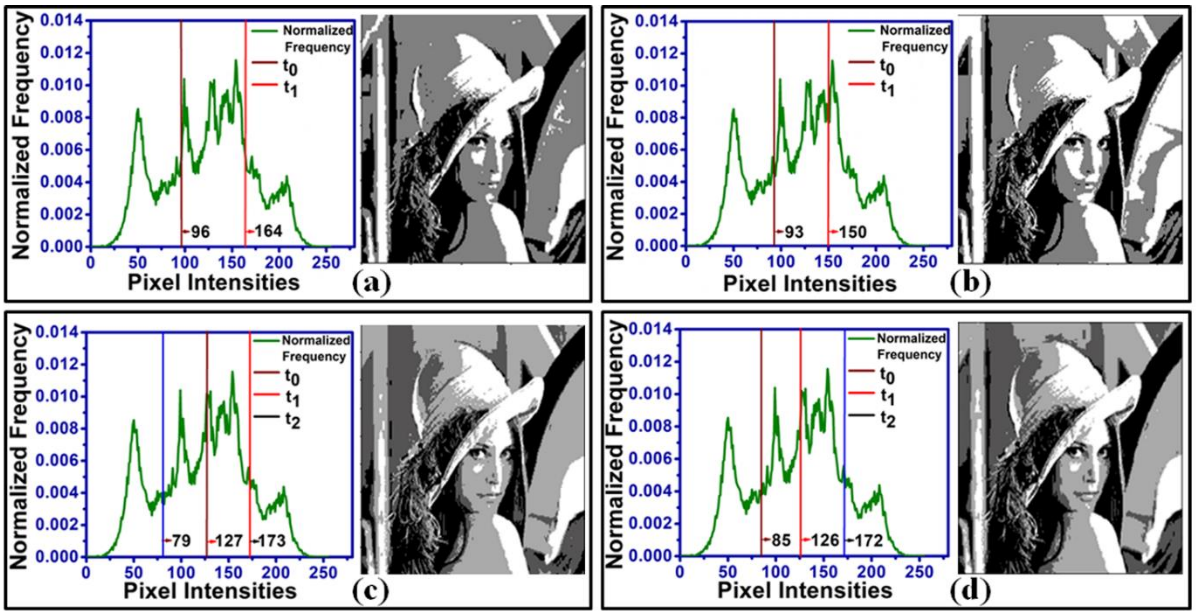


Fig. 5.5: BPSO crossbar implementation with device variations. (a) Histogram and Image for 2-thresholds while optimizing Kapur's Entropy Function, (b) Histogram and Image for 2-thresholds while optimizing Otsu's Function, (c) Histogram and Image for 3-thresholds while optimizing Kapur's Entropy Function, (d) Histogram and Image for 3-thresholds while optimizing Otsu's Function.

5.4.2 Biomedical Image Processing using the Proposed Methodology

Further, one of the conventional Brain MR images is thresholded, which are the essential source of detecting any abnormalities in the brain tissues. Multi-thresholding these images help enhance them by delineating the cerebrospinal fluid (CSF), white matter, and gray matter. This helps unveil some intricacies in the image, thus aiding the diagnosis and leading to better treatment. In this section, the results for 4 randomly selected T2-weighted transaxial brain slices, each of size 256 x 256 pixels, are discussed, which are shown in Fig. 5.6(a)-(d). Also, 4 gray-level thresholding (3 thresholds) for the above 4 images is applied. As described in section 5.1, manual thresholding to see the results for the optimal thresholds maximizing the Kapur's Entropy Function and Otsu's Function is applied, respectively; results are shown in Fig. 5.6(e)-(h) and Fig. 5.6(i)-(l).

Table 5.2: Results for the Lena Image

Optimizing Function	**Number of thresholds**	**The maximum value of the function**	**The optimum value reached by BPSO**	**BPSO crossbar implementation with 5% variations**	**Error for BPSO using the crossbar**	**Error for BPSO using crossbar considering the device variations**	**Optimal Thresholds**	**Thresholds from BPSO**	**Thresholds from crossbar with 5% device variations**
Kapur	2	12.426442	12.425018	12.424772	0.011%	0.013%	97, 165	97, 166	96, 164
Otsu	2	1944.730386	1944.140996	1944.533381	0.030%	0.010%	93, 151	94, 153	93, 150
Kapur	3	15.422701	15.416080	15.409498	0.043%	0.0860%	115, 149, 185	115, 149, 183	79, 127, 173
Otsu	3	2108.857182	2108.176481	2105.876604	0.032%	0.1413%	91, 134, 173	93, 134, 173	85,126, 172

From the above Table 5.2, it is observed that the memristor-crossbar-based BPSO is very efficacious as the maximum error obtained was only 0.1413% when compared to the maximum value of the Kapur and Otsu functions.

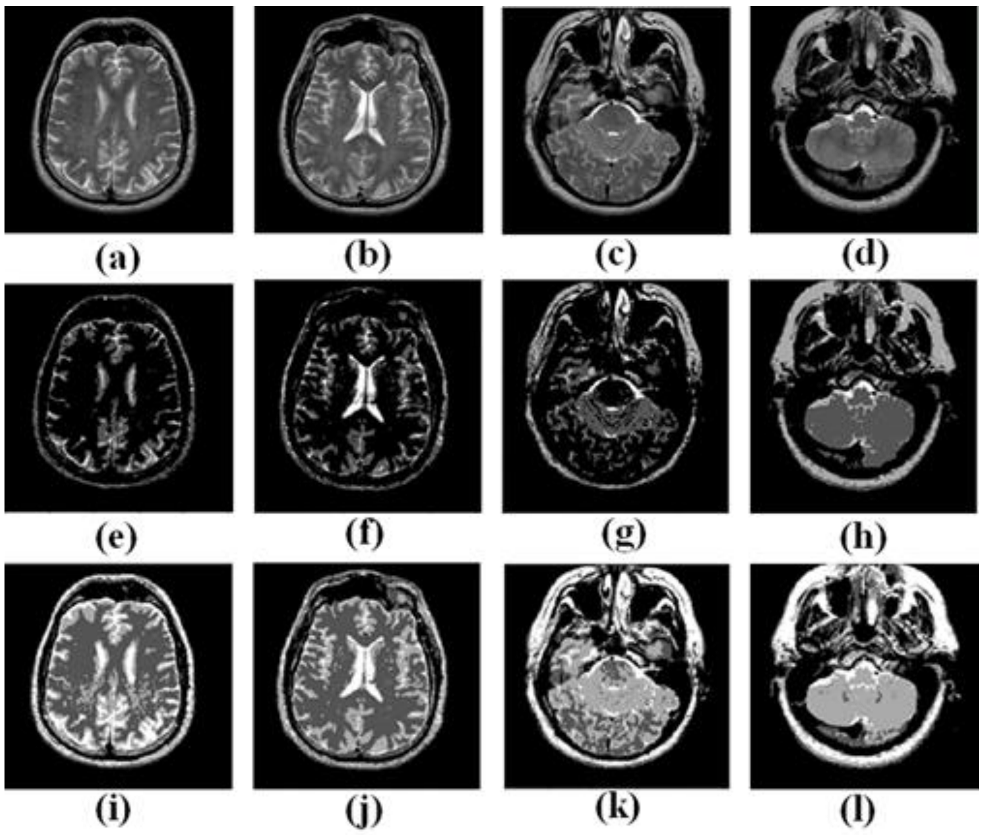


Fig. 5.6: (a)-(d) Original Gray-scale Brain MR Images. (e)-(h) show the corresponding 4 gray-level thresholding by optimal values of the Kapur's Entropy Function for images (a)-(d). (i)-(l) show the corresponding 4 gray-level thresholding by optimal values of the Otsu's Function for images (a)-(d).

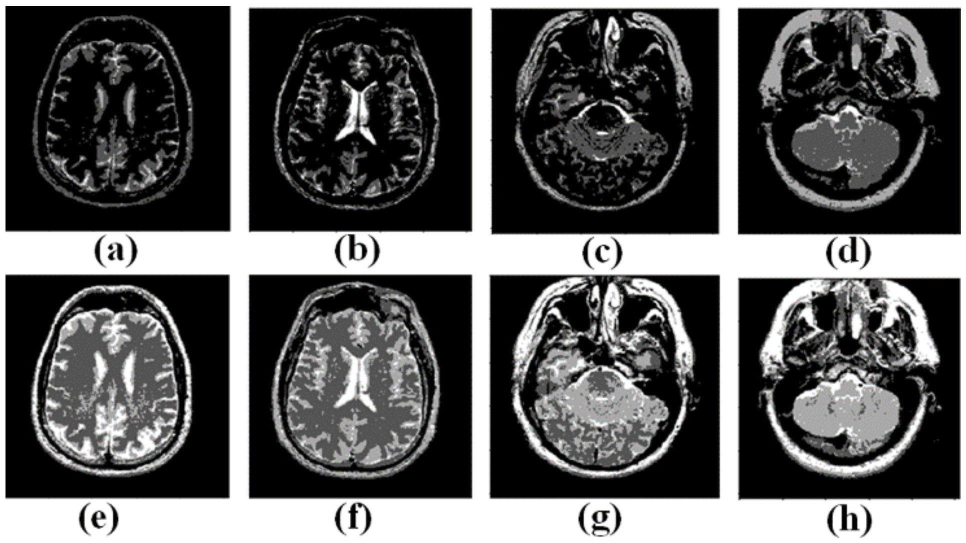


Fig. 5.7: (a)-(d). show 4 gray-level thresholding by BPSO with Kapur's Entropy Function as the objective function. (e)-(h) show 4 gray-level thresholding by BPSO with the Otsu's Function as the objective function.

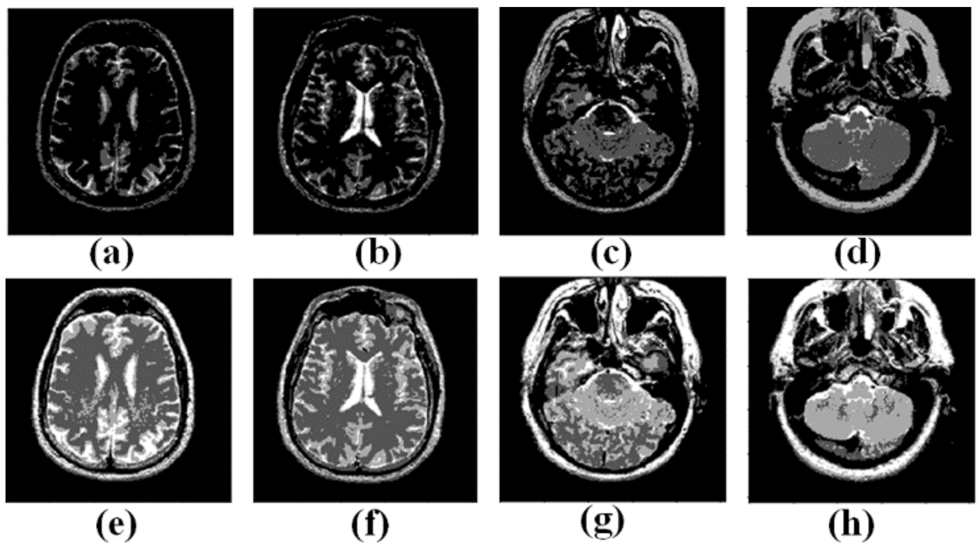


Fig. 5.8: (a)-(d). show 4 gray-level thresholding by BPSO with Kapur's Entropy Function as the objective function with 5% device variations. (e)-(h). show 4 gray-level thresholding by BPSO with Otsu's Function as the objective function with 5% device variations.

Later the crossbar simulated for 3 thresholds is considered; first, Kapur's Entropy function is considered as the objective function (Fig. 5.7(a)-(d)), followed by Otsu's Function (Fig. 5.7(e)-(h)). Also, the device variations are considered in the same crossbar, which is best depicted in Fig. 5.8(a)-(d) and Fig. 5.8(e)-(h).

Tables 5.3 and 5.4 recapitulate the quantitative results of the crossbar implementation for brain MR image thresholding. Further, these tables give a summary of results from Kapur's Entropy Function as the objective function and Otsu's Function as the Objective Function. From these tables , it is observed that there is a negligible difference between the optimal values and the values from BPSO implemented using the memristor crossbar. The error is significantly less when the crossbar implementation is compared to the brute force optimization method. It is further noticed that there is no significant difference when the device variations are considered; thus the system is robust and unsusceptible to the device variations.

Table 5.3: Results for Brain Images with Kapur’s Entropy Function as the Objective Function

Figure	**Maximum value reached by the Kapur's Entropy Function**	**Optimal Thresholds (Kapur)**	**Value reached by crossbar implemented BPSO (Kapur)**	**Thresholds from crossbar implemented BPSO**	**Value reached by crossbar implemented BPSO with 5% Variations**	**Thresholds from crossbar implemented BPSO with 5% variations**	**% Error for crossbar based BPSO**	**%Error for crossbar based BPSO with 5% device variations**
Brain Image 7 (a)	11.135578	107, 161, 209	11.13170273	112, 162, 209	11.118899	112, 162, 209	0.0348	0.1498
Brain Image 7 (b)	11.699239	101, 145, 193	11.40804753	106, 150, 189	11.420763	106, 150, 189	2.4890	2.3803
Brain Image 7 (c)	11.515785	97, 143, 187	11.51223486	91, 143, 187	11.501549	91, 143, 187	0.0308	0.1236
Brain Image 7 (d)	11.551296	57, 119, 187	11.53687572	61, 119, 188	11.548521	61, 119, 188	0.1248	0.0240

Table 5.4: Results for Brain Images with Otsu's Function as the Objective Function

Figure	**Maximum value reached by the Otsu's Function**	**Optimal Thresholds (Otsu)**	**Value reached by crossbar implemented BPSO (Otsu)**	**Thresholds from crossbar implemented BPSO**	**Value reached by crossbar implemented BPSO with 5% Variations**	**Thresholds from crossbar implemented BPSO with 5% variations**	**% Error for crossbar based BPSO**	**%Error for crossbar based BPSO with 5% device variations**
Brain Image 7 (a)	2796.147836	39, 93, 135	2795.908630	40, 94, 133	2794.171627	44, 94, 131	0.0086	0.0707
Brain Image 7 (b)	3041.200405	41, 103, 161	3041.000195	42, 104, 157	3040.677863	41, 103, 166	0.0066	0.0172
Brain Image 7 (c)	2954.556065	33, 83, 131	2954.276664	36, 86, 132	2954.276664	36, 86, 131	0.0095	0.0095
Brain Image 7 (d)	2824.621006	23, 71, 125	2823.774072	26, 76, 125	2824.424199	23, 72, 123	0.02998	0.0070

5.5 Summary

In this work, a novel memristor crossbar-based implementation of the BPSO algorithm was delineated using the characteristics of the Pt/Cu:ZnO/Nb:STO memristor. Results from memristor crossbar-based implementation agree well with the optimal values with a maximum error of only 2.5 %. The performance of the proposed implementation was validated using a variety of widely used practical gray-scale images of different sizes. Our results demonstrate that the memristor crossbar-based implementation of BPSO is a suitable alternative to the conventional CMOS system and lays the foundation for memristive devices to fasten the already fast swarm intelligence algorithms.

Although a lot of work has been proposed for the modeling of neural networks and other machine learning algorithms, very few works on other artificial intelligence-based algorithms have been proposed using memristor circuitry. There is much scope for research in the memristor-based implementation of other AI-based Algorithms like Genetic Algorithms, Simulated Annealing, and Swarm Intelligence Algorithms. Multi-thresholding for more than 4 gray-scale levels can be done at the cost of more memristive devices. The precision of c1**r1** and c2**r2** can be increased by fabricating memristors with a much lesser read time. Also, other applications of PSO and BPSO, like PSO-based neural network performance optimization, optimal filter design, or PSO-based controllers, can be designed using memristors. About Image Processing, various other subtypes of image segmentation, like morphological feature extraction, can be carried out by changing the objective function. This work can serve as an essential guide to all these future works.

Remarks: Below mentioned paper was published based on this chapter

- **Priyanka B. Ganganaik,** Omkar Mukul Gowaikar, V. Jeffry Louis, Rajesh K. Tripathy, Venkateswaran Rajagopalan, B. V. V. S. N. Prabhakar Rao, and Souvik Kundu. "Implementation of Binary Particle Swarm Optimization for Image Thresholding using Memristor Crossbar Array." In Advances in Electrical and Computer Technologies, pp. 915-936. Springer, Singapore, 2022.

Chapter-6

Memristor Based Image Fusion Architecture Using Iterative Kernel PCA

6.1 Introduction

In the digital era, the use of pictures for classification, object detection, and assistance in fields like remote sensing and surveillance is ubiquitous. Thus, the usage of pictures in different applications is increasing at a rapid rate day by day. Image fusion aims to maximize information from multiple images in one image, which can be used for different tasks like computer processing and human-visual perception. This information can be obtained through different platforms and contains the essential features from all images fused rather than viewing the individual source images [186]. Image fusion also aids in feature extraction, image segmentation, and object recognition. Its applications are very diverse in range, from remote sensing applications to medical images. Multi-sensor fusion can be used for military, surveillance, and security applications to detect smaller targets hiding in plain sight through the fusion of Infrared and Visible Images [187]. Medical Image fusion is gaining popularity due to the fusion of Computed Tomography (CT) scan images and Magnetic Resonance Imaging (MRI) images which are essential for diagnosis. They are also beneficial in satellite imagery.

To perform image fusion for the widespread range of applications, multiple methods can be used. Primary Image Fusion can be performed using a simple averaging filter as well [21]. Image fusion can happen at various levels ranging from pixel to region-based image fusion. Other methods include Principal Component Analysis, Discrete Wavelet Transform, Intensity Hue Saturation, and Pyramid Based transforms [188]. Fusion of images has also been achieved using different neural networks like Pulse Coupled Neural Networks and Generative Adversarial Networks [23], [187], [189].

Image fusion using Principal Component Analysis (PCA) is done by finding the normalized eigenvectors, which are multiplied and added to get the fused image [190], [191]. It explains the variance-covariance structure of the data through a few linear combinations of the original data. PCA is known for its high efficiency in the classification and compression of images [190], [192]. It aids in reducing the redundancy of input information and preserves data without much loss [193]. However, while PCA has the ability to perform linear transformations, the image input may not be correlated linearly. In such cases, the usage of Kernel PCA (KPCA) is promoted, as it takes higher-order statistics into account [190], [194]. The number of principal components extracted from KPCA helps keep more information when compared to the PCA and an improved result [190]. All the calculations are done in another space, F, and all the inputs are non-linearly mapped to this space [195]. One drawback of PCA is the size of the covariance matrix, leading to an increase in memory complexity [194]. These limitations are overcome by a Kernel Hebbian algorithm, an unsupervised learning technique derived from the Generalized Hebbian algorithm [194]. It is an iterative and approximative algorithm that can estimate the principal components in memory of linear order, extending its usage to larger problems.

The limitations of Von Neumann architectures as disussed in earlier chapters, with regard to data transfer rate, is one of the primary issues encountered by the computationally intensive machine learning algorithms, where the large sizes of images and other inputs used. Also, with the advancements of smaller CMOS transistors, there have been multiple issues faced, including scalability, power, and speed issues leading to the Von Neumann Bottleneck, which has led to the necessity of processing in-memory [196], [197]. Memristors are being explored for various logic implementations and memory operations, which eliminates the need to transfer data from different memory sources. As reported in [55], it has an innate ability to carry out matrix-vector multiplication through varying weights, which fosters its usage for various machine learning algorithms.

Memristor also possesses the ability to swiftly perform switching operations, consume less power, and retain values [50], [197]. So far, memristors have been used for medical image fusion and multi-focus image fusion using a Pulse Coupled Neural Network [198] and for image compression using the Discrete Cosine Transform [197]. The general Hebbian Algorithm has been implemented on the memristor for the task of classification [199]. Keeping in mind the crossbar's restraints, the Kernel Hebbian Algorithm was implemented using a 2x9 memristor crossbar. In this work, Pt/Cu:ZnO/Nb:STO memristor is used to implement Kernel Hebbian Algorithm , because of its advantages discussed in section 1.2 of chapter 1 [50]. A method to exploit the memristors to perform matrix-vector multiplications is proposed. The results of the fusion of multi-focus images and the fusion between Infrared Images and Visible Images, implemented through memristor and software, were compared.

Rest of the chapter is organized as follows. Section 6.2 explains the details of the Kernel Hebbian Algorithm (KHA) and its theory. It is followed by the implementation of KHA on the memristor in Section 6.3, and the results obtained are given in Section 6.4. The conclusions are drawn in Section 6.5.

6.1.1 Kernal Hebbian Algorithm

The Generalized Hebbian Algorithm (GHA) is used to find the mutually orthogonal eigenvectors using an iterative approach to create a new basis in the directions of maximum variance and are also called the principal components of the data set. GHA uses a single-layer feedforward neural network of the form y=wx, where w is the weight matrix, x is the input data, and y is the output data. The weight matrix is updated over multiple iterations to estimate the components. However, with an increase in the size of the data, the computational requirements also scale up. In addition, GHA only considers linear components and is not suitable for all data.

The Kernel Hebbian Algorithm (KHA) is an iterative algorithm to estimate the approximate components. It provides a memory-efficient approach to solve the problems faced by PCA and the Generalized Hebbian Algorithm. This algorithm considers higher-order information using the kernel that could not have been captured using the linear PCA model. For every pixel in each input image, a 3×3 kernel is chosen consisting of the neighboring pixels. To consider the nonlinearity of these images, they are mapped to a higher-order space using a kernel function. This kernel function is used to calculate the dot products of the mapped inputs, x and y as follows:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(x) \cdot \varphi(y) \tag{6.1}$$

It has been proved that if k is a definite positive kernel, there exists a map φ into a dot product space F such that Eqn. (6.1) is true. This space F follows the structure of a Reproducing Kernel Hilbert Space (RKHS) [195]. This is important as it is possible to deal with high-dimensional RKHS, and the specific value of φ or F doesn't need to be explicitly mentioned to perform kernel-based operations. Commonly used kernel functions include the polynomial kernel of degree, d where $k(x, y) = (x \cdot y)^d$, or the Gaussian kernel where $k(x, y) = \frac{\exp(-||x-y||^2).}{2(\sigma)^2}$

For each learning problem, there is an optimal kernel that can be used. As this is an unsupervised learning problem, the Gaussian Kernel can be used as a default kernel and is an infinite-dimensional RKHS [195]. It doesn't have a large memory requirement with the increasing size of the covariance matrix, proportional to the size of each image [194]. The weight update rule of KHA is derived from the General Hebbian Algorithm and is a linear, single-layer neural network of the form $y = Wx$ as shown in [200] where x is the input, y is the output and w is the weight matrix. For Image Fusion, x is the pixel value, and y are the principal components.

$$W(t+1) = W(t) + \eta(y(t)x(t)^T - LT\,[y(t)y(t)^T]W(t) \qquad (6.2)$$

Modifying Eqn. (6.2) to accommodate the Kernel, the network can be represented using y(t)=W(t) φ (x(t)), where φ (x(t)) are the mapped pixels. From [194], W(t) can be written as:

$$W(t) = A(t)\varphi \qquad (6.3)$$

Where A(t) are expansion coefficients of the size $(r \times l)$, where r is the number of eigenvalues, and l is the number of pixels. Replacing Eqn. (6.3) in Eqn. (6.2), we get as equation as shown below:

$$A(t+1)\varphi = A(t)\varphi + \eta(t)(y(t)\varphi(x(t))^T - \,LT\,[y(t)y(t)^T]A(t)\varphi) \qquad (6.4)$$

The mapped data points $\varphi\big(x(t)\big)$ can be depicted using canonical unit vector b(t)= (0,....1,…0) as $\varphi\big(x(t)\big) = \varphi^T b(t)$.

The reduced form from Eqn. (6.4) can be shown as:

$$A(t+1) = A(t) + \eta(t)(y(t)b(t) - LT\,[y(t)y(t)^T]A(t)) \qquad (6.5)$$

The expanded coefficients are updated over multiple iterations for each pixel in the input images until the difference between the iterations is 10^{-7}. Using the updated expansion coefficients, the eigenvalues are calculated using the formula given in [201]:

$$\lambda = \sqrt{\frac{diag(AK'(AK')')}{diag(AA')}} \qquad (6.6)$$

The eigenvectors are calculated using expansion coeffiecients A and Gaussian Kernel K as shown in [201] as shown below:

$$vec = \frac{(AK)(AK)'}{AA'} \qquad (6.7)$$

The respective eigenvector is multiplied with the input pixel value from each input image. The fused image is formed using:

$$FusedImage(i,j) = pca(1,1) * im1(i,j) + pca(2,1) * im2(i,j) \tag{6.8}$$

where (i,j) represent the location of the pixel, pca(1,1) and pca(2,1) represent the first and second principal components.

6.2 Implemention Section

6.2.1 Implementation of Memristor for KHA

To perform the matrix-vector multiplication for the Kernel Hebbian Algorithm, a (9×2) memristor crossbar was used and is shown in Fig 6.1. Each column of the crossbar corresponds to each input image, and the nine rows represent the neighbouring 3x3 kernel for a specific pixel. The memristor weights were initialized randomly to values between 0 and 1 and represented the expanded coefficients from Eqn. (6.5).

This non-volatile property of the memristor could be exploited to store this weight using the formula:

$$W_{ij} = 100\,(\frac{2X}{D} - 1) \tag{6.9}$$

A read voltage of 0.5 and -0.5 was considered as both lies between V_{on} (-1.2V) and V_{off} (+1.35V). Then the voltages for each row using voltage sources Vs1-Vs9, within range (V_{on} and V_{off}), are applied with a time pulse width proportional to the value to be multiplied with the matrix initialized to the crossbar. KHA updates the values on the memristor using individual columns using the Gaussian Kernel calculated in Eqn. (6.5).

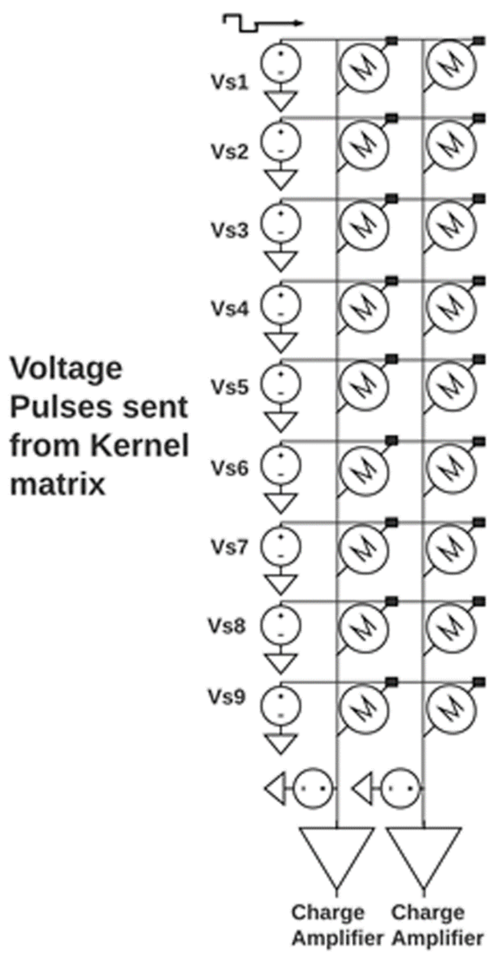


Fig. 6.1: The schematic shows the Cu:ZnO Memristive Crossbar Array. The inputs are given through a voltage pulse proportional to a single column from the Kernel Matrix, and the output charge is collected using the charge Amplifier.

The total charge accumulated in each column can be correlated to the product of the multiplication. This can be defined as:

$$Q = \sum_i \frac{v_i t_0 k_i}{R_{ij}} \tag{6.10}$$

Where the subscript i represents the ith row of the crossbar, k_i is the kernel value for rows i and v_i and, t_0 is the corresponding voltage and constant time period, respectively. The resulting product of the multiplication across each memristor can be represented as:

$$y_j = \sum_i 100 \left[2 \left(\frac{\frac{v_O t_i}{Q_{ij}} - R_{ON}}{R_{OFF} - R_{ON}} \right) - 1 \right] k_i \tag{6.11}$$

The pulse width was modelled dependent on the change in weight (Δg). If this difference was positive, an amplitude of +2V was used; otherwise, for a negative difference, a pulse of -2V was used. The pulse width can be modelled as:

$$\Delta t_{ij} = \frac{\left(\left(\frac{\frac{\Delta w_{ij}}{100}+1}{2}\right)D\right)u(\Delta g_{ij})}{K_{off}\left(\frac{v}{v_{off}}-1\right)^{\alpha_{off}}} + \frac{\left(\left(\frac{\frac{\Delta w_{ij}}{100}+1}{2}\right)D\right)u(-\Delta g_{ij})}{K_{on}\left(\frac{v}{v_{on}}-1\right)^{\alpha_{on}}} \tag{6.12}$$

Pulse Width Modulation can be used to store the pixel intensity, while Eqn. (6.12) is used to update the weight of the memristor [202]. Using this dot product of the jth column in the Gaussian Kernel Matrix and the expanded coefficients, the new expanded coefficient value is updated using the weight update rule in Eqn. (6.5). Furthermore, the multiplication of the Kernel matrix and expanded coefficients is carried out on the memristor using the matrix-vector multiplication to calculate the final components by diagonalising the resulting product.

6.3 Results and Discussions

To examine and compare the proposed algorithm's functionality, a software and hardware model was simulated on MATLAB. A memristor crossbar array of 9x2 was randomly initialized between 0 and 1 to perform the fusion of two multi-focus images. The results and observations presented in this section are based on the fusion of images from the 'Lytro Dataset' [203] and other common images used for multi-focus image fusion. The values on the memristor were updated using the weight update rule. For each kernel of pixels, it went through 500 iterations or stopped when the difference in error between the iterations was less than 10^{-7}. This updated weight is further used to calculate the two components to calculate the pixel weight with respect to the two input images. This process is repeated for each pixel in the image. The results of software and hardware implementation for the multi-focus image fusion are shown in Fig. 6.2 and are followed by a quantitative comparison from Table 6.1. The fusion of Infrared and Visible Images is highly beneficial in military applications.

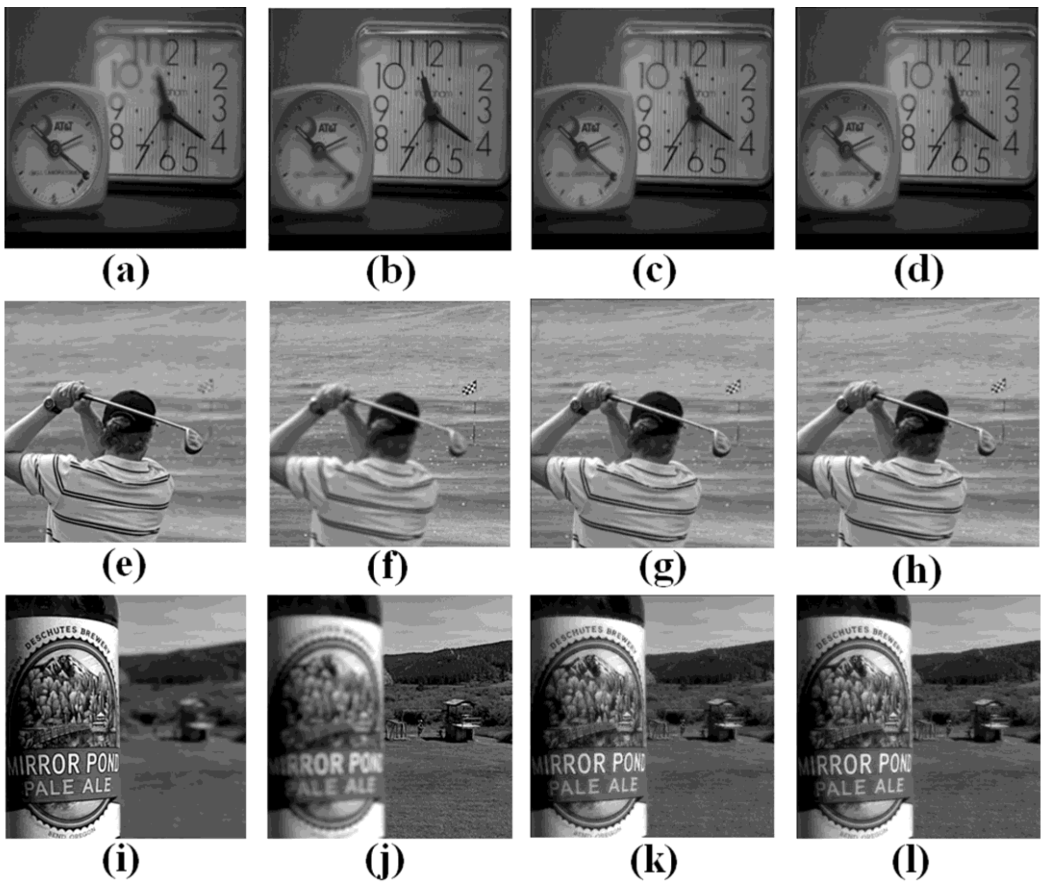


Fig. 6.2: (a-l) Fusion of multi-focus images. From (Left-Right), the first two images are the input images, the third image is the fusion of software, and the fourth image is the result of the hardware fusion.

The two images contain different information, and fusion can integrate the textural details from visible images and the thermal radiation information from infrared images to differentiate targets from the background [204]. The proposed algorithm has been tested on a set of 40 random images from the TNO dataset. This dataset is available for public use and includes multispectral images from different military relevant scenarios.

6.3.1 Performance Evaluation Metrics

The quality of the fused image was quantitatively assessed using four different performance metrics.

1. Mean Structural Similarity Index Measure (MSSIM)- This metric is used to compare the similarity of local patterns between the fused image (F) and the two source images (A, B). It is the mean of the SSIM between fused image (F), image (A), and image (B) and is calculated as shown in the equation below:

$$MSSIM(F, A, B) = \frac{SSIM(F,A)+SSIM(F,B)}{2} \tag{6.13}$$

The SSIM value is dependent on the mean and standard deviation of the two images and is calculated using the formula:

$$SSIM(A, F) = \frac{(2\mu(A)\mu(F)+C1)(2\sigma(A,F)+C2)}{(\mu(A)^2+\mu(F)^2+C1)(\sigma(A)^2+\sigma(F)^2+C2)} \tag{6.14}$$

Where $\mu(A)$ is the mean of all pixel intensities in image A, $\sigma(A)$ is the standard deviation of image A, and C1, C2 are constants [205].

2. Correlation Coefficient- The correlation coefficient is used to compare spectral features similarity in the input images and the fused image [206]. Images that are highly similar to each other have values close to 1. It is calculated using the formula:

$$CC = \frac{2C_{rf}}{C_r+C_f} \tag{6.15}$$

Where $C_r = \sum_{i=1}^{M}\sum_{j=1}^{N} I_r(i,j)^2$, $C_f = \sum_{i=1}^{M}\sum_{j=1}^{N} I_f(i,j)^2$, $C_{rf} = \sum_{i=1}^{M}\sum_{j=1}^{N} I_r(i,j)I_f(i,j)$.

3. Entropy- Entropy is used to calculate the total information content in the fused image [207]. A higher value shows that the image has more information content. It is given as follows:

$$Entropy(F) = \sum_{a,f} p(f) log\, p(f) \tag{6.16}$$

4. Q_0- This metric checks the local quality index to assess the fusion pixel by pixel [208]. It is calculated using the formula:

$$Q_0(A, F) = \frac{2\sigma_{af}}{\sigma_a^2+\sigma_f^2} \cdot \frac{2\mu_a\mu_f}{\mu_a^2+\mu_f^2} \tag{6.17}$$

Where σ_a is the standard deviation of a, σ_{af} is the covariance, μ_a is the mean. To calculate this metric with respect to both source images A and B, Q_0(A, F) and Q_0(B, F) are computed first and determined by the mean of these two values.

From Table 6.1, it is noted that the values of the memristor implementation of the KHA algorithm are almost 5% better in terms of MSSIM, 2% better in terms of Correlation Coefficient, 4.5% better in terms of Q0. In comparison, the hardware and software implementation entropy are almost similar (approximately 0.5%). It is observed from the results shown in Fig. 6.3 that the details of the two individual input images are preserved in the fused image. Quantitative analysis for the 40 images from TNO dataset was done using the metrics SSIM, Correlation Coefficient, Standard Deviation, and Entropy, as shown in Fig 6.4.

Table 6.1: Comparison between the results of software and hardware fusion of images using quantitative metrics

Image	**MSSIM-Mem**	**MSSIM-SW**	**CC-Mem**	**CC-SW**	**Entropy-Mem**	**Entropy-SW**	**Q_0-Mem**	**Q_0-SW**
lytro1	0.883	0.8631	0.96	0.94	6.92	6.92	0.91	0.88
lytro9	0.741	0.703	0.96	0.95	7.56	7.57	0.79	0.74
lytro-19	0.8389	0.8173	0.97	0.96	7.04	7.06	0.88	0.86
lytro-5	0.8627	0.8391	0.98	0.97	7.80	7.80	0.88	0.87
source14	0.8877	0.8569	0.96	0.94	7.46	7.44	0.91	0.87
d13	0.8458	0.7856	0.98	0.98	7.48	7.49	0.87	0.80
source2	0.726	0.5866	0.95	0.89	7.64	7.21	0.64	0.64
cameraman23	0.8803	0.765	0.98	0.94	7.08	7.06	0.90	0.82
clock	0.9315	0.9314	0.99	0.99	7.00	6.99	0.95	0.95
h34	0.8999	0.9009	0.97	0.97	7.04	7.04	0.92	0.92
Average	0.84968	0.80489	0.97	0.95	7.31	7.26	0.87	0.83

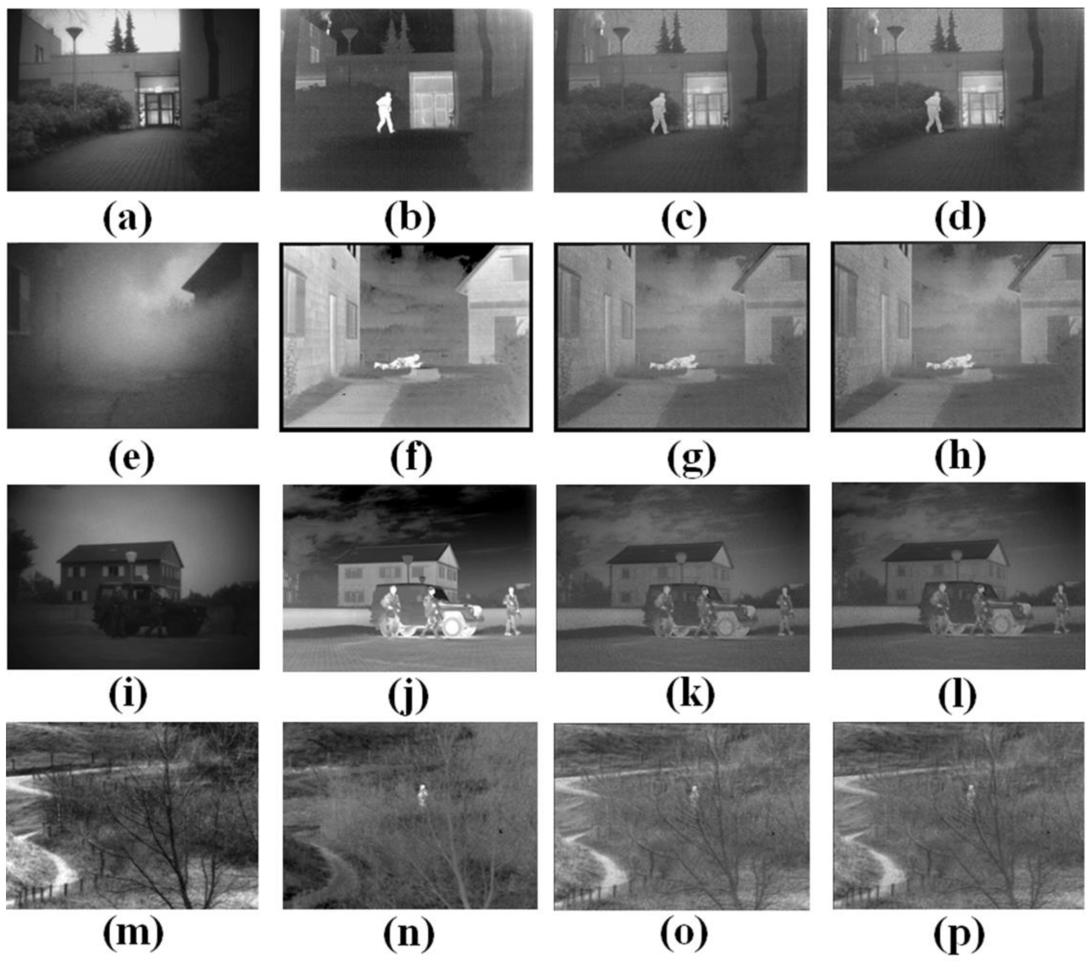


Fig. 6.3: (a-p) The fusion of Visible and Infrared Images. From (Left-Right), the first two images are the input images, the third is the result of the software implementation, and the fourth is the result of the hardware implementation.

Graphs in Fig. 6.4 show the comparison of software and hardware fusion for the 40 images fused from the TNO dataset for the quantitative metrics. Comparing the average of the memristors and software implementation, it is seen that the values are almost equal to each other, and the quality of the image through the Kernel Hebbian Algorithm was chosen as a suitable algorithm to be implemented on the memristor crossbar for its simplicity and adaptability. Unlike most multilayer neural network algorithms, KHA doesn't require any training dataset and can be used for any image. In one of the recent studies, an adversarial learning technique (GAN) was applied to 40 random images from the TNO dataset [187]. Comparing our results with the GAN, it can be found that in terms of entropy, the GAN is 2% better than KHA; however, in terms of SSIM, KHA was able to improve the results by 15%.

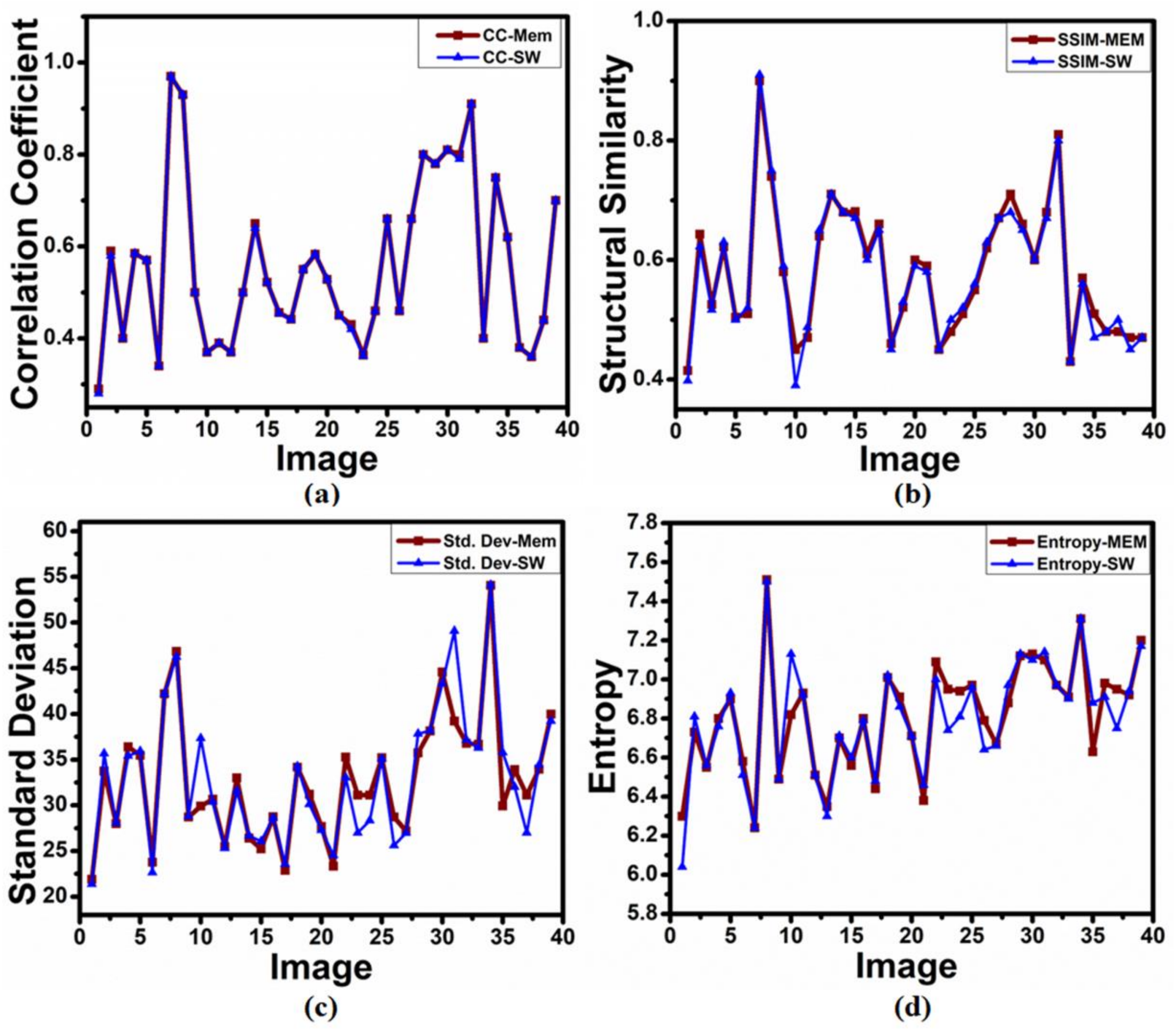


Fig. 6.4: (a) Correlation Coefficient, (b) Structural Similarity, (c) Standard deviation, and (d)Entropy.

6.4 Summary

In this chapter, a novel image fusion architecture using a memristor crossbar was proposed. This work would be highly beneficial compared to other hardware implementations of image fusion due to the minimal memory requirements, flexibility to fuse various images, and its ability to process in-memory, proving it to be a viable alternative to conventional CMOS-based image fusion architectures. This architecture has proven to give results of comparable quality to the software implementation, if not better. The application of this architecture in various areas, such as defence, medical, etc., would be highly beneficial to speed up the process of image fusion on hardware.

Remarks: Below mentioned paper is submitted and under review based on this chapter.

- **Priyanka B. Ganganaik,** Plava Kattamuri, and BVVSN Prabhakar Rao "Memristor Based Image Fusion Architecture Using Iterative Kernel PCA" submitted to Indian Journal of Engineering and material sciences.

Chapter-7

Conclusions and Future Scope

7.1 Conclusions

To overcome the limitations of Von Neumann architecture, the memristor crossbar array is delineated using the characteristics of the Pt/Cu:ZnO/Nb:STO memristor, which is used for in-memory processing as an efficient solution because of its characteristics such as long retention time, ability to store and process the data at the same time, less area, less power, and low complex systems.

- In this aspect, a memristor crossbar-based architecture CoCoPIM is presented to compute Pearson Correlation Coefficient (PCC) within the memory array. The proposed architecture is found to be 41×, 67×, and 33× times energy efficient against a Von Neumann machine in computing PCC of Electrocardiogram signals, face recognition, and Influenza illness (H1N1) data sets, respectively. It also achieved an improved speed of 143.5×, 52.5×, and 597× against the same Von Neumann machine in the respective tasks. It has also been established that CoCoPIM does not deviate significantly even in the presence of process variations, which offers to realize a highly reliable computer architecture application. The proposed idea can also be extended to a much larger switch array.
- An attempt has been made to develop a high-precision memristive crossbar circuit for Bayesian inference, followed by its specific implementation for semantic text classification by exploring the advantage of memristors' analog computational capabilities. The efficacy of the processor was tested on two different datasets consisting of 55,575 texts. The circuit was able to classify the texts with an average

accuracy of 91% while consuming 1000 times less power and area than a conventional processor.

- Efforts were devoted to investigating the resistive states of a memristor, and the transitions between these states for logic computations were utilized. In regard to this, MSM was implemented, and it was applied for tunable edge detection of images. In order to quantify the role of the proposed system, it was compared with other conventional approaches, such as Canny, Sobel, and Prewitt's systems. It is observed that the edge detection in our case was 16 %, whereas in the remaining cases, it was ~8.60 %. This improvement is due to the dual threshold approach adapted in our case. The proposed techniques are advantageous in terms of computational complexity as there is no necessity for the conventional convolution operation utilized in filter-based approaches, and it also yield improvements in terms of storage density.

- To demonstrate a novel memristor crossbar-based implementation of the BPSO algorithm for image thresholding. Results from memristor crossbar-based implementation agree well with the optimal values with a maximum error of only 2.5 %. The performance of the proposed implementation was validated using a variety of widely used practical gray-scale images of different sizes. Results obtained also demonstrate that the memristor crossbar-based implementation of BPSO is a suitable alternative to the conventional CMOS system and lays the foundation for memristive devices to fasten the presently fast swarm intelligence algorithms.
- Furthermore, a novel image fusion architecture using a memristor crossbar was proposed. To test the proposed algorithm, experiments were carried out using different multi-focus images as well as Infrared-Visible images quantitative metrics. A comparison between the conventional Von Neumann architecture and the proposed architecture is carried out. It is noted that the values of the memristor implementation

of the KHA algorithm are almost 5% better in terms of MSSIM, 2% better in terms of Correlation Coefficient, and 4.5% better in terms of Q_0. In comparison, the hardware and software implementation of entropy are almost similar (approximately 0.5%). An adversarial learning technique (GAN) was applied to 40 random images from the TNO dataset. Comparing our results with the GAN, it can be found that in terms of entropy, the GAN is 2% better than KHA; however, in terms of SSIM, KHA was able to improve the results by 15%.

- Results obtained demonstrate that the memristor crossbar implemented systems are a suitable alternative to the conventional CMOS system and lay the foundation for memristive devices for signal processing applications using machine learning algorithms.

7.2 Thesis Contributions

- For the first time, an innovative memristor crossbar-based architecture, CoCoPIM, is proposed to accelerate Correlation Coefficient computations. Three different applications are implemented based on this architecture, such as computing correlation between ECG signals, faces, and H1N1 models.
- An idea to develop an in-memory processor for Bayesian text classification using memristive crossbar architecture, in which memristive switches were employed to store information required for the classification of text.
- The strategy of implementing various image processing algorithms like tunable edge detection, image thresholding, and image fusion using a memristor crossbar array, which has the potential to overcome the Von Neumann bottleneck.

7.3 Future Scope

- Although a lot of studies has been carried out for the modeling of neural networks and other machine learning algorithms, very few studies on different artificial intelligence-based algorithms have been proposed using memristor circuitry. There is much scope for research in the memristor-based implementation of other AI-based Algorithms like Genetic Algorithms, Simulated Annealing, and Swarm Intelligence Algorithms. About Image Processing, various other subtypes of image segmentation, like morphological feature extraction, can be carried out by changing the objective function. It can also be used for the classification of EEG signals. Also, Hierarchal temporal memory can be implemented for content-based image retrieval using memristor crossbar arrays. Furthermore, there is a need to devise a method for online training of the crossbar, which can eliminate the need for a software interface to load information into to memristor array. For the widespread adoption of in-memory processors, there is a requirement for smart systems that can interface the device with existing computing systems such as DRAM, graphical processing units, and input-output devices like keyboards, speakers, and microphones. This dissertation work can serve as an essential guide to all these future works.

References

[1] Y. X. and shuo F. Junfei Qui, Qihui Wu, Guoru Ding, “A survey of machine learning for big data processing,” EURASIP J. Adv. Signal Process., pp. 1–16, 2016.

[2] G. Singh et al., “A review of near-memory computing architectures: Opportunities and challenges,” in Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018, Oct. 2018, pp. 608–617. doi: 10.1109/DSD.2018.00106.

[3] M. Motoyoshi, “Through-Silicon Via (TSV),” Proc. IEEE, vol. 97, no. 1, pp. 43–48, 2009, doi: 10.1109/JPROC.2008.2007462.

[4] S. Khoram, Y. Zha, J. Zhang, and J. Li, “Challenges and opportunities: From near-memory computing to in-memory computing,” in Proceedings of the International Symposium on Physical Design, Mar. 2017, vol. Part F1271, pp. 43–46. doi: 10.1145/3036669.3038242.

[5] M. M.-R. José Luis Rojo-Álvarez and and G. C.-V. Jordi Muñoz-Marí, Digital Signal Processing with Kernel Methods, First edit. JohnWiley & Sons Ltd., 2018.

[6] H. Dahrouj et al., “An Overview of Machine Learning-Based Techniques for Solving Optimization Problems in Communications and Signal Processing,” IEEE Access, vol. 9. Institute of Electrical and Electronics Engineers Inc., pp. 74908–74938, 2021. doi: 10.1109/ACCESS.2021.3079639.

[7] C. G. Li, Yuanqing, Kai Keng Ang, “Digital signal processing and machine learning,” Brain-Computer Interfaces, pp. 305–330, 2009.

[8] G. A. Papakostas, K. I. Diamantaras, and F. A. N. Palmieri, “Emerging Trends in Machine Learning for Signal Processing,” Computational Intelligence and Neuroscience, vol. 2017. Hindawi Limited, 2017. doi: 10.1155/2017/6521367.

[9] H. Jiang, “Design Issues in VLSI Implementation of Image Processing Hardware Accelerators Methodology and Implementation,” 2007.

[10] K. V. T. Tirupathirao, M. Tarun, S. Reddy, and K. V. Kumar, “Face Emotion Detection Using CNN,” vol. 10, no. 5, 2022, doi: 10.15680/IJIRCCE.2022.1005240.

[11] C. Li et al., “Analogue signal and image processing with large memristor crossbars,” 2017.

[12] K. Pandey, R. Lilani, P. Naik, and G. Pol, “Human Face Recognition Using Image Processing.” [Online]. Available: www.ijert.org

[13] J. Zhou, J. J. Wu, and Y. H. Tang, “Edge Detection of Binary Image Based on Memristors,” Adv. Mater. Res., vol. 791–793, pp. 2066–2070, Sep. 2013, doi: 10.4028/www.scientific.net/AMR.791-793.2066.

[14] S. Yu, “Neuro-Inspired Computing with Emerging Nonvolatile Memorys,” Proc. IEEE, vol. 106, no. 2, pp. 260–285, 2018, doi: 10.1109/JPROC.2018.2790840.

[15] V. Boskovitz and H. Guterman, “An adaptive neuro-fuzzy system for automatic image segmentation and edge detection,” IEEE Trans. Fuzzy Syst., vol. 10, no. 2, pp. 247–262, Apr. 2002, doi: 10.1109/91.995125.

[16] L. Djerou, “Automatic Multilevel Thresholding using Binary Particle Swarm Optimization for image segmentation,” 2009 Int. Conf. Soft Comput. Pattern Recognit., pp. 66–71, 2009, doi: 10.1109/SoCPaR.2009.25.

[17] S. Borjigin and P. K. Sahoo, “Color image segmentation based on multi-level Tsallis–Havrda–Charvát entropy and 2D histogram using PSO algorithms,” Pattern Recognit., vol. 92, pp. 107–118, 2019, doi: 10.1016/j.patcog.2019.03.011.

[18] M. Portes de Albuquerque, I. A. Esquef, A. R. Gesualdi Mello, and M. Portes de Albuquerque, “Image thresholding using Tsallis entropy,” Pattern Recognit. Lett., vol. 25, no. 9, pp. 1059–1065, 2004, doi: 10.1016/j.patrec.2004.03.003.

[19] N. Grover, “A study of various Fuzzy Clustering Algorithms,” Int. J. Eng. Res., vol. 3, no. 3, pp. 177–181, 2015, doi: 10.17950/ijer/v3s3/310.

[20] A. K. C. W. J. N. Kapur, P. K. Sahoo, “A New Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram,” Comput. Vision, Graph. Image Process., vol. 29, pp. 273–285, 1985.

[21] E. Lallier, M. Wing, and M. Farooq, “A real time pixel-level based image fusion via adaptive weight averaging,” Proc. 3rd Int. Conf. Inf. Fusion, FUSION 2000, vol. 2, pp. 3–13, 2000, doi: 10.1109/IFIC.2000.859841.

[22] Z. Wang and Y. Ma, “Medical image fusion using m-PCNN,” Inf. Fusion, vol. 9, no. 2, pp. 176–185, 2008, doi: 10.1016/j.inffus.2007.04.003.

[23] S. Li, J. T. Kwok, and Y. Wang, “Multifocus image fusion using artificial neural networks,” Pattern Recognit. Lett., vol. 23, no. 8, pp. 985–997, 2002, doi: 10.1016/S0167-8655(02)00029-6.

[24] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, “Text classification algorithms: A survey,” Information (Switzerland), vol. 10, no. 4. MDPI AG, 2019. doi: 10.3390/info10040150.

[25] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, “Text Classification Using Machine Learning Techniques.”

[26] M. A. Zidan, J. P. Strachan, and W. D. Lu, “The future of electronics based on memristive systems,” Nat. Electron., vol. 1, no. 1, pp. 22–29, Jan. 2018, doi: 10.1038/s41928-017-0006-8.

[27] Z. Liu et al., “A P P L I E D S C I E N C E S A N D E N G I N E E R I N G Multichannel parallel processing of neural signals in memristor arrays,” 2020. [Online]. Available: http://advances.sciencemag.org/

[28] Z. Liu et al., “Neural signal analysis with memristor arrays towards high-efficiency

brain–machine interfaces,” Nat. Commun., vol. 11, no. 1, Dec. 2020, doi: 10.1038/s41467-020-18105-4.

[29] J. V. der S. Liu, Xilin, Milin Zhang, Tao Xiong, Andrew G. Richardson, Timothy H. Lucas, Peter S. Chin, Ralph Etienne-Cummings, Trac D. Tran, “A fully integrated wireless compressed sensing neural signal acquisition system for chronic recording and brain machine interface,” IEEE Trans. Biomed. Circuits Syst., vol. 10, no. 4, pp. 874–883, 2016.

[30] Z. Y. Wu, Tong, Wenfeng Zhao, Hongsun Guo, Hubert H. Lim, “A streaming PCA VLSI chip for neural data compression,” IEEE Trans. Biomed. Circuits Syst., vol. 11, no. 6, pp. 1290–1302, 2017.

[31] K. V. S. Kao, Jonathan C., Sergey D. Stavisky, David Sussillo, Paul Nuyujukian, “Information systems opportunities in brain–machine interface decoders,” Proc. IEEE, vol. 102, no. 5, pp. 666–682, 2014.

[32] T. R. Moreno, Félix, Jaime Alarcón, Rubén Salvador, “Reconfigurable hardware architecture of a shape recognition system based on specialized tiny neural networks with online training,” IEEE Trans. Ind. Electron., vol. 56, no. 8, pp. 3253–3263, 2009.

[33] W.-Y. Y. Zhuang, Hualiang, Kay-Soon Low, “A pulsed neural network with on-chip learning and its practical applications,” IEEE Trans. Ind. Electron., vol. 54, no. 1, pp. 34–42, 2007.

[34] M. Chu et al., “Neuromorphic Hardware System for Visual Pattern Recognition with Memristor Array and CMOS Neuron,” IEEE Trans. Ind. Electron., vol. 62, no. 4, pp. 2410–2419, 2015, doi: 10.1109/TIE.2014.2356439.

[35] K. K. L. Türel, Özgür, Jung Hoon Lee, Xiaolong Ma, “Neuromorphic architectures for nanoelectronic circuits,” Int. J. Circuit Theory Appl., vol. 32, no. 5, pp. 277–302, 2004.

[36] C. G. Zhao, W. S., Guillaume Agnus, Vincent Derycke, A. Filoramo, J. P. Bourgoin, “Nanotube devices based crossbar architecture: toward neuromorphic computing,” Nanotechnology, vol. 21, no. 17, p. 175202, 2010.

[37] L. O. Chua, “Memristor—The Missing Circuit Element,” IEEE Trans. Circuit Theory, vol. 18, no. 5, pp. 507–519, 1971, doi: 10.1109/TCT.1971.1083337.

[38] I. E. Ebong and P. Mazumder, “CMOS and memristor-based neural network design for position detection,” in Proceedings of the IEEE, 2012, vol. 100, no. 6, pp. 2050–2060. doi: 10.1109/JPROC.2011.2173089.

[39] A. Chen., “A review of emerging non-volatile memory (NVM) technologies and applications,” Solid. State. Electron., vol. 125, pp. 25-38., 2016.

[40] R. Berdan, “Imperial College of Science , Technology and Medicine Department of Electrical and Electronic Engineering Applications of Memristors in Conventional Analogue Electronics Radu Berdan,” no. November, 2016.

[41] R. Courtland, “Gordon moore: The man whose name means progress,” IEEE Spectr., vol. 30, 2015.

[42] “Memristor Circuits and Systems Thesis by Mohammed Affan Zidan In Partial Fulfillment of the Requirements For the Degree of Doctor of Philosophy,” 2015.

[43] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale memristor device as synapse in neuromorphic systems,” Nano Lett., vol. 10, no. 4, pp. 1297–1301, 2010, doi: 10.1021/nl904092h.

[44] S. . Shin. S., Kim. K., Kang, “Memristor applications for programmable analog ICs,” IEEE Trans. Nanotechnol., vol. 10(2), pp. 266–274, 2011.

[45] H. H. Li and M. Hu, “Compact Model of Memristors and Its Application in Computing Systems,” 2010.

[46] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” Nature, vol. 453, no. 7191, pp. 80–83, May 2008, doi: 10.1038/nature06932.

[47] O. Krestinskaya, S. Member, A. James, S. Member, and L. O. Chua, “Neuro-memristive Circuits for Edge Computing : A Review,” pp. 1–20.

[48] D. Niu, Y. Chen, C. Xu, and Y. Xie, “Impact of process variations on emerging memristor,” Proc. - Des. Autom. Conf., pp. 877–882, 2010, doi: 10.1145/1837274.1837495.

[49] T. P. C. Hu, S. G., S. Y. Wu, W. W. Jia, Q. Yu, L. J. Deng, Yong Qing Fu, Y. Liu, “Review of nanostructured resistive switching memristor and its applications.,” Nanosci. Nanotechnol. Lett. 6, vol. 9, pp. 729–757, 2014.

[50] P. K. R. Boppidi et al., “Unveiling the dual role of chemically synthesized copper doped zinc oxide for resistive switching applications,” J. Appl. Phys., vol. 124, no. 21, p. 214901, Dec. 2018, doi: 10.1063/1.5052619.

[51] B. Suresh, P. K. R. Boppidi, B. V. V. S. N. Prabhakar Rao, S. Banerjee, and S. Kundu, “Realizing spike-timing dependent plasticity learning rule in Pt/Cu:ZnO/Nb:STO memristors for implementing single spike based denoising autoencoder,” J. Micromechanics Microengineering, vol. 29, no. 8, p. 085006, Aug. 2019, doi: 10.1088/1361-6439/ab235f.

[52] T. Wang and J. Roychowdhury, “Well-Posed Models of Memristive Devices,” May 2016.

[53] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, “VTEAM: A General Model for Voltage-Controlled Memristors,” IEEE Trans. Circuits Syst. II Express Briefs, vol. 62, no. 8, pp. 786–790, Aug. 2015, doi: 10.1109/TCSII.2015.2433536.

[54] T. D. Dongale et al., “Effect of write voltage and frequency on the reliability aspects of memristor-based RRAM,” Int. Nano Lett., vol. 7, no. 3, pp. 209–216, Sep. 2017, doi: 10.1007/s40089-017-0217-z.

[55] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, “Memristor crossbar-based neuromorphic computing system: A case study,” IEEE Trans. Neural Networks Learn. Syst., vol. 25, no. 10, pp. 1864–1878, Oct. 2014, doi: 10.1109/TNNLS.2013.2296777.

[56] H. U. Xiaofang, D. Shukai, W. Lidan, and L. Xiaofeng, “Memristive crossbar array with applications in image processing,” vol. 55, no. 2, pp. 461–472, 2012, doi: 10.1007/s11432-011-4410-9.

[57] B. P. Mazumder and F. Ieee, “Memristors : Devices , Models , and Applications,” vol. 100, no. 6, pp. 1911–1919, 2012.

[58] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, “‘Memristive’ switches enable ‘stateful’ logic operations via material implication,” Nature, vol. 464, no. 7290, pp. 873–876, Apr. 2010, doi: 10.1038/nature08940.

[59] M. L. Lehtonen, Eero, “Stateful implication logic with memristors.",” IEEE/ACM Int. Symp. Nanoscale Archit. IEEE, pp. 33-36., 2009.

[60] M. L. Lehtonen, Eero, J. H. Poikonen, “Two memristors suffice to compute all Boolean functions,” Electron. Lett., vol. 46, no. 3, p. 230, 2010.

[61] M. D. V. Pershin, Yuriy V., “Practical approach to programmable analog circuits with memristors,” IEEE Trans. Circuits Syst., vol. 57, no. 8, pp. 1857–1864, 2010.

[62] W. D. J. Wey, Todd A., “Variable gain amplifier circuit using titanium dioxide memristors,” IET circuits, devices Syst., vol. 5, no. 1, p. 59, 2011.

[63] T. C. Berdan R., Prodromakis T, Salaoru I, Khiat A, “Memristive devices as parameter setting elements in programmable gain amplifiers,” Appl. Phys. Lett., vol. 101, no. 24, p. 243502, 2012.

[64] L. L. Chew, Z. J., “Printed circuit board based memristor in adaptive lowpass filter,” Electron. Lett., vol. 48, no. 25, pp. 1610–1611, 2012.

[65] S. J. W. Joglekar, Yogesh N., “The elusive memristor: properties of basic electrical circuits,” Eur. J. Phys., vol. 30, no. 4, p. 661, 2009.

[66] E. P. Amador, Andrés, Emilio Freire, “Bifurcation set for a disregarded Bogdanov-Takens unfolding,” Appl. to 3D cubic memristor Oscil. Nonlinear Dyn., vol. 104, no. 2, pp. 1657–1675, 2021.

[67] J. P. X. Bao, B. C., Zhong Liu, “Steady periodic memristor oscillator with transient chaotic behaviours.",” Electron. Lett., vol. 46, no. 3, p. 228, 2010.

[68] Y. M. Yu, Qin, Zhiguang Qin, Juebang Yu, “Transmission characteristics study of memristors based op-amp circuits,” Int. Conf. Commun. circuits Syst., pp. 974–977, 2009.

[69] S. B. S. Merrikh-Bayat, Farnood, “Memristor-based circuits for performing basic arithmetic operations,” Procedia Comput. Sci., vol. 3, pp. 128–132, 2011.

[70] L. E. Guckert, “Memristor-based arithmetic units,” 2016.

[71] B. Mouttet, “Proposal for memristors in signal processing,” Int. Conf. nano-networks,Springer, Berlin, Heidelberg, pp. 11–13, 2008.

[72] S. L. E. Gonzalez, Rafael C., Richard E. Woods, Digital image processing using MATLAB. Pearson Education India, 2004.

[73] N. K. Upadhyay, H. Jiang, Z. Wang, S. Asapu, Q. Xia, and J. J. Yang, “Emerging Memory Devices for Neuromorphic Computing,” vol. 1800589, pp. 1–13, 2019, doi: 10.1002/admt.201800589.

[74] J. A. Starzyk and Basawaraj, “Memristor crossbar architecture for synchronous neural networks,” IEEE Trans. Circuits Syst. I Regul. Pap., vol. 61, no. 8, pp. 2390–2401, Aug. 2014, doi: 10.1109/TCSI.2014.2304653.

[75] X. F. Hu, S. K. Duan, L. D. Wang, and X. F. Liao, “Memristive crossbar array with applications in image processing,” Sci. China Inf. Sci., vol. 55, no. 2, pp. 461–472, Feb. 2012, doi: 10.1007/s11432-011-4410-9.

[76] C. Yang, Q. Hu, Y. Yu, R. Zhang, Y. Yao, and J. Cai, “Memristor-based Chaotic Circuit for Text / Image Encryption and Decryption,” 2015 8th Int. Symp. Comput. Intell. Des., vol. 1, no. 3, pp. 447–450, 2015, doi: 10.1109/ISCID.2015.156.

[77] I. J. Electron, C. Aeü, S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, “Memristor augmented approximate adders and subtractors for image processing applications : An approach,” Int. J. Electron. Commun., vol. 91, no. January, pp. 91–102, 2018, doi: 10.1016/j.aeue.2018.05.003.

[78] A. Sarmiento-Reyes and Y. R. Velasquez, “A charge-controlled memristor model for image edge detection with a memristive grid,” in 2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS), Jun. 2017, pp. 49–52. doi: 10.1109/ICCDCS.2017.7959703.

[79] S. Zhu, L. Wang, and S. Duan, “Neurocomputing Memristive pulse coupled neural network with applications in medical image processing,” Neurocomputing, vol. 227, no. June 2016, pp. 149–157, 2017, doi: 10.1016/j.neucom.2016.07.068.

[80] K. Roy, “Image Edge Detection Based on Swarm Intelligence Using Memristive Networks,” vol. 37, no. 9, pp. 1774–1787, 2018.

[81] L. D. Q. G. X et al., “NeuroSim + : An Integrated Device - to - Algorithm Framework for Benchmarking Synaptic Devices and Array Architectures,” pp. 135–138, 2017.

[82] A. Patel, F. Afram, S. Chen, and K. Ghose, “MARSS: A full system simulator for multicore x86 CPUs,” Proc. - Des. Autom. Conf., pp. 1050–1055, 2011.

[83] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing,” Trans. Archit. Code Optim., vol. 10, no. 1, pp. 469–480, 2013,

doi: 10.1145/2445572.2445577.

[84] P. Kogge, S. Lead, D. Campbell, J. Hiller, M. Richards, and A. Snavely, “ExaScale Computing Study : Technology Challenges in Achieving Exascale Systems,” Gov. Procure., 2008.

[85] J. Hennessy and D. Patterson, Computer Architecture : A Quantitative Approach. Burlington, MA, USA: Morgan-Kaufmann, 1996.

[86] S. Kvatinsky et al., “MAGIC – Memristor Aided LoGIC,” IEEE Trans. Circuits Syst., vol. 61, 2014.

[87] J. Louis, B. Hoffer, and S. Kvatinsky, “Performing Memristor-Aided Logic (MAGIC) using,” pp. 787–790, 2019.

[88] N. P. Jouppi, N. Patil, and D. Patterson, “Motivation for and Evaluation of the First Tensor Processing Unit,” IEEE Micro, vol. 38, no. June, pp. 10–19, 2018, doi: 10.1109/MM.2018.032271057.

[89] T. Eslami, M. G. Awan, and F. Saeed, “GPU-PCC: A GPU based technique to compute pairwise Pearson’s Correlation Coefficients for big fMRI data,” ACM-BCB 2017 - Proc. 8th ACM Int. Conf. Bioinformatics, Comput. Biol. Heal. Informatics, pp. 723–728, 2017, doi: 10.1145/3107411.3108173.

[90] Y. Liu, T. Pan, and S. Aluru, “Parallel Pairwise Correlation Computation on Intel Xeon Phi Clusters,” Proc. - Symp. Comput. Archit. High Perform. Comput., no. 2, pp. 141–149, 2016, doi: 10.1109/SBAC-PAD.2016.26.

[91] F. T. L. Cavallaro, Joseph R., “CORDIC arithmetic for an SVD processor,” J. Parallel Distrib. Comput., vol. 5, no. 3, pp. 271–290, 1988.

[92] A. A. Mopuri, Suresh, “Low-complexity methodology for complex square-root computation,” IEEE Trans. Very Large Scale Integr. Syst., vol. 25, no. 11, pp. 3255–3259, 2017.

[93] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 19, no. 7, pp. 711–720, 1997, doi: 10.1109/34.598228.

[94] R. B. Tanvir, T. Aqila, M. Maharjan, A. Al Mamun, and A. M. Mondal, “Graph theoretic and pearson correlation-based discovery of network biomarkers for cancer,” Data, vol. 4, no. 2, 2019, doi: 10.3390/data4020081.

[95] I. Laaridh, W. Ben Kheder, C. Fredouille, and C. Meunier, “Automatic prediction of speech evaluation metrics for dysarthric speech,” Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH, vol. 2017-Augus, no. August, pp. 1834–1838, 2017, doi: 10.21437/Interspeech.2017-1363.

[96] Z. Wang et al., “Assessment of Blood Tumor Mutational Burden as a Potential Biomarker for Immunotherapy in Patients with Non-Small Cell Lung Cancer with Use

of a Next-Generation Sequencing Cancer Gene Panel," JAMA Oncol., vol. 5, no. 5, pp. 696–702, 2019, doi: 10.1001/jamaoncol.2018.7098.

[97] S. M. Long, Yun, Taesik Na, "ReRAM-based processing-in-memory architecture for recurrent neural network acceleration," IEEE Trans. Very Large Scale Integr. Syst., vol. 26, no. 12, pp. 2781–2794, 2018.

[98] P. Chi et al., "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," Proc. - 2016 43rd Int. Symp. Comput. Archit. ISCA 2016, pp. 27–39, 2016, doi: 10.1109/ISCA.2016.13.

[99] L. J. Zhao, Yilong, Zhezhi He, Naifeng Jing, Xiaoyao Liang, "Re2PIM: A reconfigurable ReRAM-based PIM design for variable-sized vector-matrix multiplication," Proc. 2021 Gt. Lakes Symp. VLSI, pp. 15-20., 2021.

[100] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016, 2016, pp. 14–26. doi: 10.1109/ISCA.2016.12.

[101] A. R. Jain, Shubham, Ashish Ranjan, Kaushik Roy, "Computing in memory with spin-transfer torque magnetic RAM," IEEE Trans. Very Large Scale Integr. Syst., vol. 26, no. 3, pp. 470-483., 2017.

[102] D. F. Angizi, Shaahin, Zhezhi He, Amro Awad, "MRIMA: An MRAM-based in-memory accelerator," IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 39, no. 5, pp. 1123–1136, 2019.

[103] X. S. H. Reis, Dayane, Michael Niemier, "Computing in memory with FeFETs," Proc. Int. Symp. low power Electron. Des., pp. 1-6., 2018.

[104] H. Akoglu, "User's guide to correlation coefficients," Turkish J. Emerg. Med., vol. 18, no. 3, pp. 91–93, 2018, doi: 10.1016/j.tjem.2018.08.001.

[105] S. Belkacem, N. Messaoudi, and Z. Dibi, "Artifact removal from electrocardiogram signal: A comparative study S.," 2018 Int. Conf. Signal, Image, Vis. their Appl. SIVA 2018, pp. 1–5, 2019, doi: 10.1109/SIVA.2018.8661013.

[106] Y. Liu, Y. Li, X. Ma, and R. Song, "Facial expression recognition with fusion features extracted from salient facial areas," Sensors (Switzerland), vol. 17, no. 4, pp. 1–18, 2017, doi: 10.3390/s17040712.

[107] Z. Zhou, J. Zhao, and K. Xu, "Can online emotions predict the stock market in China?," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10041 LNCS, no. April, pp. 328–342, 2016, doi: 10.1007/978-3-319-48740-3_24.

[108] P. Oldiges, Q. Lin, K. Petrillo, M. Sanchez, M. Ieong, and M. Hargrove, "Modeling line edge roughness effects in sub 100 nanometer gate length devices," Int. Conf. Simul. Semicond. Process. Devices, SISPAD, no. June 2014, pp. 131–134, 2000, doi: 10.1109/sispad.2000.871225.

[109] R. P. Paily, “Low Power Squaring and Square root Circuits Using Subthreshold MOS Transistors,” no. 2, pp. 96–99, 2009.

[110] K. A. B. Andreou, Andreas G., “Translinear circuits in subthreshold MOS,” Analog Integr. Circuits Signal Process., vol. 9, no. 2, pp. 141-166., 1996.

[111] A. L. Goldberger et al., “PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals.,” Circulation, vol. 101, no. 23, 2000, doi: 10.1161/01.cir.101.23.e215.

[112] S. Cook, C. Conrad, A. L. Fowlkes, and M. H. Mohebbi, “Assessing Google Flu trends performance in the United States during the 2009 influenza virus A (H1N1) pandemic,” PLoS One, vol. 6, no. 8, pp. 1–8, 2011, doi: 10.1371/journal.pone.0023610.

[113] M. Hu et al., “Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication,” in Proceedings - Design Automation Conference, Jun. 2016, vol. 05-09-June. doi: 10.1145/2897937.2898010.

[114] M. Hu, J. P. Strachan, Z. Li, R. Stanley, and Williams, “Dot-product engine as computing memory to accelerate machine learning algorithms,” in Proceedings - International Symposium on Quality Electronic Design, ISQED, May 2016, vol. 2016-May, pp. 374–379. doi: 10.1109/ISQED.2016.7479230.

[115] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, “Design implications of memristor-based RRAM cross-point structures,” in Proceedings -Design, Automation and Test in Europe, DATE, 2011, pp. 734–739. doi: 10.1109/date.2011.5763125.

[116] J. Joshua Yang et al., “Engineering nonlinearity into memristors for passive crossbar applications,” Appl. Phys. Lett., vol. 100, no. 11, p. 113501, Mar. 2012, doi: 10.1063/1.3693392.

[117] M. Farhadloo, R. A. Patterson, and E. Rolland, “Modeling customer satisfaction from unstructured data using a Bayesian approach,” Decis. Support Syst., vol. 90, pp. 1–11, Oct. 2016, doi: 10.1016/j.dss.2016.06.010.

[118] P. Tsangaratos and I. Ilia, “Comparison of a logistic regression and Naïve Bayes classifier in landslide susceptibility assessments: The influence of models complexity and training dataset size,” Catena, vol. 145, pp. 164–179, Oct. 2016, doi: 10.1016/j.catena.2016.06.004.

[119] T. A. Almeida, J. María Gómez Hidalgo, and T. P. Silva, “Towards SMS Spam Filtering: Results under a New Dataset,” Int. J. Inf. Secur. Sci., vol. 2, no. 1, pp. 1–18, 2016.

[120] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in ACL-HLT 2011 - Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, vol. 1, pp. 142–150.

[121] B. Efron, “Bayes’ theorem in the 21st century,” Science, vol. 340, no. 6137. pp. 1177–1178, 2013. doi: 10.1126/science.1236536.

[122] D. D. Lewis, "Naive(Bayes)at forty: The independence assumption in information retrieval," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1998, vol. 1398, pp. 4–15. doi: 10.1007/bfb0026666.

[123] C. Silva and B. Ribeiro, "The Importance of Stop Word Removal on Recall Values in Text Categorization," in Proceedings of the International Joint Conference on Neural Networks, 2003, vol. 3, pp. 1661–1666. doi: 10.1109/ijcnn.2003.1223656.

[124] L. Dolamic and J. Savoy, "Brief communication: When stopword lists make the difference," J. Am. Soc. Inf. Sci. Technol., vol. 61, no. 1, pp. 200–203, 2010, doi: 10.1002/asi.21186.

[125] N. Desai and M. Narvekar, "Normalization of noisy text data," in Procedia Computer Science, 2015, vol. 45, no. C, pp. 127–132. doi: 10.1016/j.procs.2015.03.104.

[126] P. M. P. Raj, A. Subramaniam, S. Priya, S. Banerjee, and S. Kundu, "Programming of Memristive Artificial Synaptic Crossbar Network Using PWM Techniques," J. Circuits, Syst. Comput., vol. 28, no. 12, 2019, doi: 10.1142/S0218126619502013.

[127] I. Guyon, "A scaling law for the validation-set training-set size ratio," AT&T Bell Lab., pp. 1–11, 1997.

[128] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müllier, "Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors," in 2010 International Conference on Green Computing, Green Comp 2010, 2010, pp. 123–133. doi: 10.1109/GREENCOMP.2010.5598316.

[129] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," IEEE Trans. Nanotechnol., vol. 15, no. 4, pp. 635–650, Jul. 2016, doi: 10.1109/TNANO.2016.2570248.

[130] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in ACM SIGPLAN Notices, Mar. 2018, vol. 53, no. 2, pp. 1–14. doi: 10.1145/3173162.3173171.

[131] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," Nat. Nanotechnol., vol. 3, no. 7, pp. 429–433, Jul. 2008, doi: 10.1038/nnano.2008.160.

[132] Y. Ho, G. M. Huang, and P. Li, "Dynamical Properties and Design Analysis for Nonvolatile Memristor Memories," IEEE Trans. Circuits Syst. I Regul. Pap., vol. 58, no. 4, pp. 724–736, Apr. 2011, doi: 10.1109/TCSI.2010.2078710.

[133] H. Manem, J. Rajendran, and G. S. Rose, "Design Considerations for Multilevel CMOS/Nano Memristive Memory," ACM J. Emerg. Technol. Comput. Syst., vol. 8, no. 1, pp. 1–22, Feb. 2012, doi: 10.1145/2093145.2093151.

[134] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in [1992] Proceedings 29th ACM/IEEE Design Automation Conference, pp. 253–259. doi:

10.1109/DAC.1992.227826.

[135] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," Nat. Nanotechnol., vol. 8, no. 1, pp. 13–24, Jan. 2013, doi: 10.1038/nnano.2012.240.

[136] A. K. Maan, D. A. Jayadevi, and A. P. James, "A Survey of Memristive Threshold Logic Circuits," IEEE Trans. Neural Networks Learn. Syst., vol. 28, no. 8, pp. 1734–1746, Aug. 2017, doi: 10.1109/TNNLS.2016.2547842.

[137] A. Irmanova and A. P. James, "Multi-level memristive memory with resistive networks," in 2017 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia), Oct. 2017, pp. 69–72. doi: 10.1109/PRIMEASIA.2017.8280366.

[138] M. Park, S. Park, and K.-H. Yoo, "Multilevel Nonvolatile Memristive and Memcapacitive Switching in Stacked Graphene Sheets," ACS Appl. Mater. Interfaces, vol. 8, no. 22, pp. 14046–14052, Jun. 2016, doi: 10.1021/acsami.6b01962.

[139] Y. Sharma, R. Gera, 450,359 GP Singh - US Patent 5, and undefined 1995, "Analog video interactive (AVI) PC Add-On Card for controlling consumer grade VHS-VCR," Google Patents.

[140] L. A. Zadeh, "Fuzzy logic," Computer (Long. Beach. Calif)., vol. 21, no. 4, pp. 83–93, Apr. 1988, doi: 10.1109/2.53.

[141] P. Maragos, "Lattice Image Processing: A Unification of Morphological and Fuzzy Algebraic Systems," J. Math. Imaging Vis., vol. 22, no. 2–3, pp. 333–353, May 2005, doi: 10.1007/s10851-005-4897-z.

[142] B. K. Bose, "Expert system, fuzzy logic, and neural network applications in power electronics and motion control," Proc. IEEE, vol. 82, no. 8, pp. 1303–1323, 1994, doi: 10.1109/5.301690.

[143] L. Hu, H. D. Cheng, and M. Zhang, "A high performance edge detector based on fuzzy inference rules," Inf. Sci. (Ny)., vol. 177, no. 21, pp. 4768–4784, Nov. 2007, doi: 10.1016/J.INS.2007.04.001.

[144] R. C. Hardie and C. G. Boncelet, "Gradient-based edge detection using nonlinear edge enhancing prefilters," IEEE Trans. Image Process., vol. 4, no. 11, pp. 1572–1577, 1995, doi: 10.1109/83.469938.

[145] A. Sarmiento-Reyes and Y. R. Velásquez, "Charge-Controlled Memristor Grid for Edge Detection," in Advances in Memristor Neural Networks - Modeling and Applications, InTech, 2018. doi: 10.5772/intechopen.78610.

[146] F. Merrikh-Bayat, S. Bagheri Shouraki, and F. Merrikh-Bayat, "Memristive fuzzy edge detector," J. Real-Time Image Process., vol. 9, no. 3, pp. 479–489, Sep. 2014, doi: 10.1007/s11554-012-0254-9.

[147] J. Canny, "A Computational Approach to Edge Detection," IEEE Trans. Pattern Anal.

Mach. Intell., vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[148] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, “Design of an image edge detection filter using the Sobel operator,” IEEE J. Solid-State Circuits, vol. 23, no. 2, pp. 358–367, Apr. 1988, doi: 10.1109/4.996.

[149] E. V. Paika and E. P. Bhambri, “Edge Detection Fuzzy Inference System,” Int. J. Manag. Inf. Technol., vol. 4, no. 1, pp. 148–155, Jun. 2013, doi: 10.24297/ijmit.v4i1.811.

[150] O. Castillo, P. Melin, J. Kacprzyk, and W. Pedrycz, “Type-2 Fuzzy Logic: Theory and Applications,” in 2007 IEEE International Conference on Granular Computing (GRC 2007), Nov. 2007, pp. 145–145. doi: 10.1109/GrC.2007.118.

[151] P. Melin, C. I. Gonzalez, J. R. Castro, O. Mendoza, and O. Castillo, “Edge-Detection Method for Image Processing Based on Generalized Type-2 Fuzzy Logic,” IEEE Trans. Fuzzy Syst., vol. 22, no. 6, pp. 1515–1525, Dec. 2014, doi: 10.1109/TFUZZ.2013.2297159.

[152] P. Melin and O. Castillo, “A review on type-2 fuzzy logic applications in clustering, classification and pattern recognition,” Applied Soft Computing Journal, vol. 21. Elsevier Ltd, pp. 568–577, Aug. 2014. doi: 10.1016/j.asoc.2014.04.017.

[153] G. N. Chaple, R. D. Daruwala, and M. S. Gofane, “Comparisions of Robert, Prewitt, Sobel operator based edge detection methods for real time uses on FPGA,” in 2015 International Conference on Technologies for Sustainable Development (ICTSD), Feb. 2015, pp. 1–4. doi: 10.1109/ICTSD.2015.7095920.

[154] S. Bezryadin, P. Bourov, and D. Ilinih, “Brightness Calculation in Digital Image Processing,” Int. Symp. Technol. Digit. Photo Fulfillment, vol. 2007, no. 1, pp. 10–15, Sep. 2014, doi: 10.2352/issn.2169-4672.2007.1.0.10.

[155] V. Novák and S. Lehmke, “Logical structure of fuzzy IF-THEN rules,” Fuzzy Sets Syst., vol. 157, no. 15, pp. 2003–2029, Aug. 2006, doi: 10.1016/J.FSS.2006.02.011.

[156] A. D. Borkar and M. Atulkar, “Fuzzy inference system for image processing,” Int. J. Adv. Res. Comput. Eng. Technol., vol. 2, no. 3, pp. 1007–1010, 2013.

[157] F. Merrikh-Bayat and S. B. Shouraki, “Memristive Neuro-Fuzzy System,” IEEE Trans. Cybern., vol. 43, no. 1, pp. 269–285, Feb. 2013, doi: 10.1109/TSMCB.2012.2205676.

[158] Y. Shim, A. Sengupta, and K. Roy, “Low-power approximate convolution computing unit with domain-wall motion based ‘spin-memristor’ for image processing applications,” in Proceedings of the 53rd Annual Design Automation Conference on - DAC ’16, 2016, pp. 1–6. doi: 10.1145/2897937.2898042.

[159] N. Anastasiadis, I. Sideris, and K. Pekmestzi, “A fast multiplier-less edge detection accelerator for FPGAs,” in Proceedings of the ACM Symposium on Applied Computing, 2010, pp. 510–515. doi: 10.1145/1774088.1774193.

[160] G. Kornaros, “A soft multi-core architecture for edge detection and data analysis of microarray images,” J. Syst. Archit., vol. 56, no. 1, pp. 48–62, Jan. 2010, doi: 10.1016/j.sysarc.2009.11.004.

[161] P. Michael Preetam Raj, A. Ranjan Kalita, M. K. Hudait, S. Priya, and S. Kundu, “Nonlinear DC equivalent circuits for ferroelectric memristor and Its FSM application,” Integr. Ferroelectr., vol. 192, no. 1, pp. 16–27, Sep. 2018, doi: 10.1080/10584587.2018.1521667.

[162] G. Beni and J. Wang, “Swarm Intelligence in Cellular Robotic Systems,” Robot. Biol. Syst. Towar. a New Bionics?, no. 2, pp. 703–712, 1993, doi: 10.1007/978-3-642-58069-7_38.

[163] R. E. Kennedy, James, “Particle swarm optimization,” Proc. ICNN’95-International Conf. Neural Networks, vol. 4, pp. 1942–1948, 1995.

[164] K. Mason, J. Duggan, and E. Howley, “Multi-objective dynamic economic emission dispatch using particle swarm optimisation variants,” Neurocomputing, vol. 270, pp. 188–197, 2017, doi: 10.1016/j.neucom.2017.03.086.

[165] M. Hajihassani, D. Jahed Armaghani, and R. Kalatehjari, “Applications of Particle Swarm Optimization in Geotechnical Engineering: A Comprehensive Review,” Geotech. Geol. Eng., vol. 36, no. 2, pp. 705–722, 2018, doi: 10.1007/s10706-017-0356-z.

[166] A. H. Elsheikh and M. Abd Elaziz, “Review on applications of particle swarm optimization in solar energy systems,” Int. J. Environ. Sci. Technol., vol. 16, no. 2, pp. 1159–1170, 2019, doi: 10.1007/s13762-018-1970-x.

[167] H. Han, S. Member, W. Lu, Y. Hou, and J. Qiao, “An Adaptive-PSO-Based Self-Organizing,” vol. 29, no. 1, pp. 104–117, 2018.

[168] H. Melo and J. Watada, “Gaussian-PSO with fuzzy reasoning based on structural learning for training a Neural Network,” Neurocomputing, vol. 172, pp. 405–412, 2016, doi: 10.1016/j.neucom.2015.03.104.

[169] R. Sivaranjani, S. M. M. Roomi, and M. Senthilarasi, “Speckle noise removal in SAR images using Multi-Objective PSO (MOPSO) algorithm,” Appl. Soft Comput. J., vol. 76, pp. 671–681, 2019, doi: 10.1016/j.asoc.2018.12.030.

[170] T. X. Pham, P. Siarry, and H. Oulhadj, “Integrating fuzzy entropy clustering with an improved PSO for MRI brain image segmentation,” Appl. Soft Comput. J., vol. 65, pp. 230–242, 2018, doi: 10.1016/j.asoc.2018.01.003.

[171] J. Kennedy and R. C. Eberhart, “Discrete binary version of the particle swarm algorithm,” Proc. IEEE Int. Conf. Syst. Man Cybern., vol. 5, pp. 4104–4108, 1997, doi: 10.1109/icsmc.1997.637339.

[172] L. Djerou, N. Khelil, H. E. Dehimi, and M. Batouche, “Automatic multilevel thresholding using binary particle swarm optimization for image segmentation,”

SoCPaR 2009 - Soft Comput. Pattern Recognit., pp. 66–71, 2009, doi: 10.1109/SoCPaR.2009.25.

[173] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, “Sub-nanosecond switching of a tantalum oxide memristor,” Nanotechnology, vol. 22, no. 48, p. 485203, Dec. 2011, doi: 10.1088/0957-4484/22/48/485203.

[174] B. J. Choi et al., “High-Speed and Low-Energy Nitride Memristors,” Adv. Funct. Mater., vol. 26, no. 29, pp. 5290–5296, Aug. 2016, doi: 10.1002/adfm.201600680.

[175] J. Zhou, F. Cai, Q. Wang, B. Chen, S. Gaba, and W. D. Lu, “Very low-programming-current RRAM with self-rectifying characteristics,” IEEE Electron Device Lett., vol. 37, no. 4, pp. 404–407, 2016, doi: 10.1109/LED.2016.2530942.

[176] Y. Yu, Q. Deng, L. Ren, and N. Tashi, “Memristor Crossbar Array Based ACO For Image Edge Detection,” Neural Process. Lett., vol. 51, no. 2, pp. 1891–1905, 2020, doi: 10.1007/s11063-019-10179-6.

[177] X. Zhang, Y. Zhang, Z. Zhang, S. Mahadevan, A. Adamatzky, and Y. Deng, “Rapid Physarum Algorithm for shortest path problem,” Appl. Soft Comput. J., vol. 23, pp. 19–26, 2014, doi: 10.1016/j.asoc.2014.05.032.

[178] S. Choi, P. Sheridan, and W. D. Lu, “Data Clustering using Memristor Networks,” Nat. Publ. Gr., pp. 1–10, 2015, doi: 10.1038/srep10492.

[179] M. Dorigo and G. Di Caro, “Ant colony optimization: A new metaheuristic, evolutionary computation,” CEC 99. Proc. 1999 Congr., vol. 2, pp. 1470–1477, 1999.

[180] Z. Pajouhi and K. Roy, “Image edge detection based on swarm intelligence using memristive networks,” IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 37, no. 9, pp. 1774–1787, 2018, doi: 10.1109/TCAD.2017.2775227.

[181] Y. Geng, S. Duan, Z. Dong, and L. Wang, “A novel PID neural network controller based on memristor,” Chinese Control Conf. CCC, vol. 2, no. 4, pp. 3988–3993, 2017, doi: 10.23919/ChiCC.2017.8027982.

[182] W. L. Johnson KA, Becker JA, The whole brain atlas. 1999.

[183] N. Otsu, “Threshold Selection Method From Gray-Level Histograms.,” IEEE Trans Syst Man Cybern, vol. SMC-9, no. 1, pp. 62–66, 1979, doi: 10.1109/TSMC.1979.4310076.

[184] G.F.S., “The Bell system technical journal,” J. Franklin Inst., vol. 196, no. 4, pp. 519–520, 1923, doi: 10.1016/s0016-0032(23)90506-5.

[185] C. E. Shannon, “A Mathematical Theory of Communication,” Bell Syst. Tech. J., vol. 27, no. 4, pp. 623–656, 1948, doi: 10.1002/j.1538-7305.1948.tb00917.x.

[186] H. Li and B. s. Manjunath, “Multisensor-Image-Fusion-Using-the-Wavelet-Transform_1995_Graphical-Models-and-Image-Processing.pdf,” Graphical models and image processing, vol. 57, no. 3. pp. 235–245, 1995.

[187] J. Ma, W. Yu, P. Liang, C. Li, and J. Jiang, “FusionGAN: A generative adversarial network for infrared and visible image fusion,” Inf. Fusion, vol. 48, pp. 11–26, 2019, doi: 10.1016/j.inffus.2018.09.004.

[188] R. Kaur and R. Kaur, “A Survey on Image Fusion Techniques for Image Enhancement in Digital Image Processing,” Int. J. Comput. Appl., vol. 179, no. 45, pp. 24–27, 2018, doi: 10.5120/ijca2018917133.

[189] Z. Dong, C. S. Lai, D. Qi, Z. Xu, C. Li, and S. Duan, “A general memristor-based pulse coupled neural network with variable linking coefficient for multi-focus image fusion,” Neurocomputing, vol. 308, pp. 172–183, 2018, doi: 10.1016/j.neucom.2018.04.066.

[190] L. J. Cao, K. S. Chua, W. K. Chong, H. P. Lee, and Q. M. Gu, “A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine,” Neurocomputing, vol. 55, no. 1–2, pp. 321–336, 2003, doi: 10.1016/S0925-2312(03)00433-8.

[191] R. P. Desale and S. V. Verma, “Study and analysis of PCA, DCT & DWT based image fusion techniques,” Int. Conf. Signal Process. Image Process. Pattern Recognit. 2013, ICSIPR 2013, vol. 1, pp. 66–69, 2013, doi: 10.1109/ICSIPR.2013.6497960.

[192] A. George, “Anomaly Detection based on Machine Learning Dimensionality Reduction using PCA and Classification using SVM,” Int. J. Comput. Appl., vol. 47, no. 21, pp. 5–8, 2012, doi: 10.5120/7470-0475.

[193] R. Kaur and S. Kaur, “An Approach for Image Fusion using PCA and Genetic Algorithm,” Int. J. Comput. Appl., vol. 145, no. 6, pp. 54–59, 2016, doi: 10.5120/ijca2016910816.

[194] K. I. Kim, M. O. Franz, and B. Scho, “Iterative Kernel Principal Component Analysis for Image Modeling,” Analysis, vol. 27, no. 9, pp. 1351–1366, 2005.

[195] B. S. Smola, Alex J., Learning with kernels. GMD-Forschungszentrum Informationstechnik, 1998.

[196] W. A. Wulf and S. A. McKee, “Hitting the memory wall,” ACM SIGARCH Comput. Archit. News, vol. 23, no. 1, pp. 20–24, 1995, doi: 10.1145/216585.216588.

[197] C. Li et al., “Analogue signal and image processing with large memristor crossbars,” Nat. Electron., vol. 1, no. 1, pp. 52–59, 2018, doi: 10.1038/s41928-017-0002-z.

[198] Z. Wang and Y. Ma, “Medical image fusion using m -PCNN,” vol. 9, pp. 176–185, 2008, doi: 10.1016/j.inffus.2007.04.003.

[199] P. M. P. Raj, V. J. Louis, S. K. Chatterjee, S. Kanungo, and S. Kundu, “Ferroelectric Memristive Networks for Dimensionality Reduction: A Process for Effectively Classifying Cancer Datasets,” Integr. Ferroelectr., vol. 201, no. 1, pp. 126–141, 2019, doi: 10.1080/10584587.2019.1668697.

[200] T. D. (Massachusetts I. of T. Sanger, “Optimal Unsupervised Learning in Feedforward

Neural Networks," IEEE Trans. Neural Networks, vol. 2, pp. 459–473, 1989.

[201] S. Günter, N. N. Schraudolph, and S. V. N. Vishwanathan, "Fast iterative kernel principal component analysis," J. Mach. Learn. Res., vol. 8, pp. 1893–1918, 2007.

[202] S. Raj, P. M. P., Subramaniam, A., Priya, S., Banerjee, S., & Kundu, "Programming of memristive artificial synaptic crossbar network using PWM techniques," J. Circuits, Syst. Comput., vol. 28, no. 12, p. 1950201, 2019.

[203] M. Nejati, S. Samavi, and S. Shirani, "Multi-focus image fusion using dictionary-based sparse representation," Inf. Fusion, vol. 25, no. October, pp. 72–84, 2015, doi: 10.1016/j.inffus.2014.10.004.

[204] J. Ma, Y. Ma, and C. Li, "Infrared and visible image fusion methods and applications: A survey," Inf. Fusion, vol. 45, no. Chang Li, pp. 153–178, 2019, doi: 10.1016/j.inffus.2018.02.004.

[205] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, 2004, doi: 10.1109/TIP.2003.819861.

[206] Z. Han, X. Tang, X. Gao, and F. Hu, "Image Fusion and Image Quality Assessment of Fused Images," ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci., vol. XL-7/W1, no. 4, pp. 33–36, 2013, doi: 10.5194/isprsarchives-xl-7-w1-33-2013.

[207] C. Pohl and J. L. Van Genderen, Review article Multisensor image fusion in remote sensing: Concepts, methods and applications, vol. 19, no. 5. 1998. doi: 10.1080/014311698215748.

[208] Z. Wang and A. C. Bovik, "A universal image quality index," IEEE Signal Process. Lett., vol. 9, no. 3, pp. 81–84, 2002, doi: 10.1109/97.995823.

Appendix

A1. Implementation of Memristor Based Correlation Coefficient Parameter in Memory Processing

MATLAB Code:

```
%%ECG correlation Coefficient%%
%folder acess
folder = 'C:\Users\Priyanka B G\Desktop\4_2\IMFRA\applications';
filePattern = fullfile(folder, '*.mat');
srcFiles = dir(filePattern);
numFiles = length(srcFiles);
if numFiles == 0
  message = sprintf('There are no mat files are in folder:\n%s', folder);
  uiwait(warndlg(message));
else
  fprintf('There are so many files in %s:\n', numFiles, folder);
  for l = 1 : numFiles
    fprintf('    %s\n', srcFiles(l).name);
  end
end
mydata = cell(1, numFiles);
store1 = zeros(48,3600);
store2 = zeros(48,3600);
for l = 1:numFiles
  mydata{l} = load(srcFiles(l).name);
  mydata1= mydata{l};
  a1=mydata1;
  a11=(a1.val(1,:));
  a12=a11./100;
  t=linspace(-1,1,3600);
an = a12+0.15*randn(size(t));
store1(l,:) = an;
```

```
% store1(1,:) = strcat(num2str(store1(1,:)),',');
n=0.15*randn(size(t));
[cA,cD] = dwt(an,'db8');
x = idwt(cA,cD,'db8');
yx = wden(x,'sqtwolog','s','mln',3,'db8');
store2(l,:) = yx;
ARx=corr2(an,yx);
jeff_corr(l) = ARx;
an = round(an,2);
yx = round(yx,2);
an1 = round(abs(an-mean(an)),2);
an_mean = round(abs(mean(an)),2);
yx_mean = round(abs(mean(yx)),2);
            Roff = 120000;
            Ron = 1200;
            Gon = 1/Ron;
            Goff = 1/Roff;
var = 1 + normrnd(0,0.05);
Nr1 = an-an_mean;
Nr1 = (Nr1./100)*(Gon-Goff)*var;
Nr2 = yx-yx_mean;
Nr2 = Nr2*8.9e-9*var;
Mem_Nr = (Nr1)*(Nr2)'*100/((Gon-Goff)*8.9e-9);
Dr1 = (((an-an_mean)./100*(Gon-Goff)*var).*100/(Gon-Goff)).^2;
Dr2 = (((yx-yx_mean)./100*(Gon-Goff)*var).*100/(Gon-Goff)).^2;
Mem_Dr = (sum(Dr1) * sum(Dr2))^0.5;
jeff_corr_mem(l) = Mem_Nr/Mem_Dr;
net_time = (sum(an1)/0.99)*100*10e-9 + 3600 *10e-9;
E_nr = 0;
    for i = 1:1:511
            E_nr = E_nr + yx(i)/an1(i);
    end
```

```
   W_nr = 48*3600*10e-9*1.5*1.5/(8.25*1e-4);
   E_nr = E_nr *48*8*10e-9*0.5*0.5/(8.25*1e-4);
   a2 = an1.^-1;
   a2 = 4*10e-9*1*1*a2./(8.25*1e-4);
   E_dr = sum(a2,2);
   E_net = E_nr + E_dr;
for i=1:3600
cr(i,:)=ARx;
i=i+1
end
end
%%Face Recognition%%
folder = 'C:\Users\Jeffry Louis\Desktop\DOP3\codes';
filePattern = fullfile(folder, '*.pgm');
srcFiles = dir(filePattern);
numFiles = length(srcFiles);
if numFiles == 0
message = sprintf('There are no jpg files are in folder:\n%s', folder);
uiwait(warndlg(message));
else
fprintf('There are %d files in %s:\n', numFiles, folder);
for k = 1 : numFiles
fprintf('    %s\n', srcFiles(k).name);
end
end
mydata = cell(1, numFiles);
for k = 1:numFiles
mydata{k} = imread(srcFiles(k).name);
end
for k = 1:20
Pic(k) = mydata(k);
l = k+20;
```

```
Ref(k) = mydata(l);

end

param_corr = zeros(1,100);
X = imread('straight_neutral_sunglassesboland_.pgm');
X_original   = imread('straight_neutral_openboland_.pgm');
param_corr = zeros(1,20);
U = (randi([0,10],[120,10]));
delU = zeros(120,10);
VT =  (randi([0,10],[10,128]));
V = VT.';
imshow(X_original);
param_corr = corr2(X_original,X);
Wstore = zeros(120,10,100);
for k = 1:1:10
num1  = X*VT.';
denom1= U*VT*VT.';
num2  = X.'*U;
denom2= V*U.'*U;
for i = 1:1:120
for j = 1:1:10
U(i,j)= U(i,j)*(num1(i,j)/denom1(i,j));
U(i,j) = round(U(i,j),0);
 [U(i,j)] = Uupdate(delU(i,j),U(i,j));
end
end
for i = 1:1:128
for j = 1:1:10
V(i,j)= V(i,j)*(num2(i,j)/denom2(i,j));
V(i,j) = round(V(i,j),0);
end
```

```
end
for i = 1:1:120
for j = 1:1:10
U2(i,j) = U(i,j)*U(i,j);
end
end
for i = 1:1:120
for j = 1:1:10
U(i,j) = U(i,j)/sqrt(sum(U2(:,j)));
end
end
for i = 1:1:128
for j = 1:1:10
V(i,j) = V(i,j)*sqrt(sum(U2(:,j)));
end
end
for a = 1:1:120
for b=1:1:10
Wstore(a,b,k) = U(a,b);
end
end
X2 = U*V.';
X2 = X2;
param_corr(1,k) = corr2(X_original,X2);
end
Wplot = zeros(1,100);
for l =1:1:100
Wplot(1,l) = Wstore(80,6,l);
end
function [U1] = Uupdate(delU1,U2)
U1 = delU1 + U2;
x  = (U2)*25;
```

```
dx = (delU1)*25;
x = x + dx;
 if (delU1>0)
[x] = mem(2,x,dx);
else
[x] = mem(-2,x,dx);
end
U1 = (x/25);
% end
 function [x_new] = mem(v,x_prev,dx)
Roff    =152000000;
Ron     =150000;
voff    =1.35; %Must be postive;
von     =-1.2; %Must be negative;
koff    =200;
kon     =-250;
aoff    =3;
aon     =2;
D       =50e-09;
xoff    =  D;
xon     =  0;
if v>=voff
dxdt=(koff*((v/voff-1))^aoff);
%calculating dxdt as per voltage;
elseif v<=von
dxdt=(kon*((v/von)-1)^aon);
 else
dxdt=0;
end
dxdt = koff*(v/voff -1)^aoff;
dt   = dx/dxdt;
x_new = x_prev + dxdt*dt;
```

```
end

%%H1N1 correlation Coefficient%%
% % % Storing image in variable arrays
a = zeros(1,44);
b = zeros(1,44);
b_abs = zeros(1,44);
param_corr = zeros(1,1);
t_nr = zeros(1,1);
t_dr = zeros(1,1);
t_total = zeros(1,1);
Nr_time1 = zeros(1,1);
Nr_time = zeros(1,1);
a_G = zeros(1,44);
E_nr = zeros(1,1);
E_dr = zeros(1,1);
E_total = zeros(1,1);
for N = 1:1:1
b(1,:) = gt(N,:);
b = b - round(mean(b));
for i = 1:1:1
   a(i,:) = ili(i,:);
end
% % % Storing image in crossbars
xbar1 = zeros(512,512);
% 20 because we initialize only the first 20 rows. The rest are not used
for i = 1:1:1
   for j = 1:1:512
         if (j >44)
            break;
         else
            xbar1(i,j) = a(i,j);
            end
    end
end
for i=1:1:1
    xbar1(i,512) = round(mean(a(i,:)));
end
% % % Numerator of correlation coefficient
Nr = zeros(1,1);
for i=1:1:1
    for j=1:1:511
        if(j > 44)
            break;
        else
            Nr(i,1) = Nr(i,1) + get_prod(xbar1(i,j),b(1,j),1) + get_prod(xbar1(i,512),b(1,j),-1);
%               Nr(i,1) = Nr(i,1) + xbar1(i,j)*b(1,j) - xbar1(i,512)*b(1,j);
        end
    end
end
% % % Denominator of correlation coefficient
Dr = zeros(1,1);
for i=1:1:1
    for j=1:1:511
        if(j > 44)
            break;
        else
            Dr(i,1) = Dr(i,1) + get_square(xbar1(i,j),xbar1(i,512));
%               Dr(i,1) = Dr(i,1) + (xbar1(i,j)-xbar1(i,512))^2;
        end
    end
end
Dr_summed = Dr;
% Dr = Dr./1600;
```

```
Dr = Dr.^0.5;
Dr_after_root = Dr;
for i=1:1:1
   Dr(i,1) = Dr(i,1)*std(b)*6.63;
end
for i=1:1:1
param_corr(i,N) = Nr(i,1)/Dr(i,1);
end
% % Calculating time
b_abs = abs(b);
t_dr(N,1) = 44;
% % Calculating Nr time
   for j=1:1:44
            t_nr(N,1) = t_nr(N,1) + b(1,j);
   end
t_total(N,1) = (t_nr(N,1) + t_dr(N,1))*8.9e-9;
% % Calculating Energy of Nr
Roff = 120000;
Ron = 1200;
Gon = 1/Ron;
Goff = 1/Roff;
a_G = (a./258).*(Gon-Goff) + Goff;
b_abs = b_abs.*8.9e-9;
for i=1:1:1
   a_Gtemp = a_G(i,:);
   E_nr(N,1) = E_nr(N,1) + 0.5*0.5*(a_Gtemp*b_abs')*10e-9;
%     E_nr(N,1) = E_nr(N,1) + 0.5*0.5*4*(mean(a_Gtemp)*t_nr(N,1));
end
for i=1:1:1
   a_Gtemp = a_G(i,:);
   E_dr(N,1) = E_dr(N,1) + 0.5*0.5*(a_Gtemp*b_abs');
   E_dr(N,1) = E_dr(N,1) + 0.5*0.5*(4*mean(a_Gtemp)*( 8.9e-9)*511);
```

```
end
 E_total = E_nr + E_dr;
end
        function Q = get_prod(a,b,v)
%                   Q = v*a*b;
            Roff = 120000;
            Ron = 1200;
            voff    =1.35; %Must be postive;
            von     =-1.2; %Must be negative;
            koff    =200;
            kon     =-250;
            aoff    =3;
            aon     =2;
            D       =50e-09;
            xoff    =  D;
            xon     =  0;
           var = 1 + normrnd(0,0.05);
            Roff = var*Roff;
            Ron = var*Ron;
            D = var*D;
            Gon = 1/Ron;
            Goff = 1/Roff;
            G1 = (a/100)*(Gon-Goff);
            v_width = b*8.9e-9;
            Q1 = v*G1*v_width;
   % % Peripheral Circuits for Numerator%%
            Q = Q1*100/((Gon-Goff)*8.9e-9);
         end
        function x = get_square(a,b)
%              x = (a-b)^2;
            Roff = 120000;
            Ron = 1200;
```

```
    voff    =1.35; %Must be postive;
    von     =-1.2; %Must be negative;
    koff    =200;
    kon     =-250;
    aoff    =3;
    aon     =2;
    D       =50e-09;
    xoff    =  D;
    xon     =  0;
    var = 1 + normrnd(0,0.05);
    Roff = var*Roff;
    Ron = var*Ron;
    D = var*D;
    Gon = 1/Ron;
    Goff = 1/Roff;
    G1 = (a/100)*(Gon-Goff) + Goff;
    G2 = (b/100)*(Gon-Goff) + Goff;
    I1 = 1*G1;
    I2 = -1*G2;
    y = (I1+I2);
   % % Peripheral Circuits for Denominator%%
    y = y*100/(Gon-Goff);
    x = (y)^2;
    end

%%Memristor crossbar modelling%%
% % % Storing image in variable arrays
a = zeros(20,1600);
b = zeros(1,1600);
b(1,:) = sno(2,:);
b = b - round(mean(b));
param_corr = zeros(1,20);
```

```
for i = 1:1:20
    a(i,:) = round(sho(i,:)/3)*3;
%     param_corr(1,i) = calc_corr(a,b);
end
% % % Storing image in crossbars
xbar1 = zeros(512,512);
xbar2 = zeros(512,512);
xbar3 = zeros(512,512);
xbar4 = zeros(512,512);
% 20 because we initialize only the first 20 rows. The rest are not used
for i = 1:1:20
    for j = 1:1:512
       if(j == 512)
           xbar1(i,j) = round(mean(a(i,:)));
           xbar2(i,j) = round(mean(a(i,:)));
           xbar3(i,j) = round(mean(a(i,:)));
           xbar4(i,j) = round(mean(a(i,:)));
       else
           xbar1(i,j) = a(i,j);
           j2 = 512 + j;
           xbar2(i,j) = a(i,j2);
           j3 = 1024 + j;
           xbar3(i,j) = a(i,j3);
       end
    end
end
for i = 1:1:20
    for j = 1:1:512
        j4 = 1536 + j;
            if (j4 >1600)
                break;
            else
            xbar4(i,j) = a(i,j4);
         end
   end
end
for i=1:1:20
   xbar4(i,512) = round(mean(a(i,:)));
end
% % % Numerator of correlation coefficient
Nr = zeros(20,1);
for i=1:1:20
   for j=1:1:511
      j2 = 512 + j;
      j3 = 1024 + j;
      Nr(i,1) = Nr(i,1) + xbar1(i,j)*b(1,j)  - xbar1(i,512)*b(1,j) + xbar2(i,j)*b(1,j2) -
xbar2(i,512)*b(1,j2) + xbar3(i,j)*b(1,j3) - xbar3(i,512)*b(1,j3) ;
   end
end
for i=1:1:20
   for j=1:1:511
   j4 = 1536 + j;
      if(j4 > 1600)
         break;
      else
         Nr(i,1) = Nr(i,1) +xbar4(i,j)*b(1,j4) - xbar4(i,512)*b(1,j4);
      end
   end
end
% % % Denominator of correlation coefficient
Dr = zeros(20,1);
for i=1:1:20
   for j=1:1:511
      j2 = 512 + j;
```

```
        j3 = 1024 + j;
        Dr(i,1) = Dr(i,1) + (xbar1(i,j) - xbar1(i,512))^2 + (xbar2(i,j) - xbar2(i,512))^2 +
(xbar3(i,j) - xbar3(i,512))^2 ;
    end
end
for i=1:1:20
    for j=1:1:511
    j4 = 1536 + j;
        if(j4 > 1600)
            break;
        else
            Dr(i,1) = Dr(i,1) +(xbar4(i,j) - xbar4(i,512))^2;
        end
    end
end

Dr = Dr./1600;
Dr = Dr.^0.5;
for i=1:1:20
    Dr(i,1) = Dr(i,1)*std(b)*1600;
end
param_corr = zeros(20,1);
for i=1:1:20
param_corr(i,1) = Nr(i,1)/Dr(i,1);
end
function corr = calc_corr(a,b)
val = 0;
for i = 1:1:1600
a1 = a(1,i)-mean2(a);
b1 = b(1,i)-mean2(b);
val = val + a1*b1;
end
```

```
corr = val/(std(a)*std(b)*1600);
end
```

A2. Implementation of Memristor Based in memory processing for high precisoion text classification

```
%%Cleaning the text%%
function text =clean_text(text)
        text=lower(text);
    %rempve all words starting with @
    text = regexprep(text,"(@)\w*","");
    %     remove all words starting with @
    text = regexprep(text,"\w*'\w*d","");
        %remove all stop words
    text = remove_stop_words(text);
end
%%Classify the text%%
function [accuracy,row]= classifier(label,text,training_ratio)
training_label = label(1:int32(end*training_ratio));
training_text = text(1:int32(end*training_ratio));
test_label = label(int32(end*training_ratio)+1:end);
test_text = text(int32(end*training_ratio)+1:end);
disp("Generating Model");
model = generate_model_2(training_label,training_text);
disp("Predicting Categories")
accuracy = evaluate(model,test_label,test_text);
row = length (model {2,2});
end
% d= sprintf("Accuracy Of Classification %f",accuracy);
% disp(d);
%%Modelling Crossbar
function current = mem(voltage,memristance)
Roff = 152426795;
Ron = 150793;
```

```
voff = 1.35;
von     =-1.2;
koff    =200;
kon     =-250;
aoff    =3;
aon     =2;
D       =50e-09;
xoff    =  D;
xon     =  0;
dt=0.01;
% memristance
if voltage>= voff
   dwdt=(koff*((voltage/voff-1))^aoff);
elseif voltage<=von
   dwdt=(kon*((voltage/von)-1)^aon);
else
   dwdt=0;
end
x=(memristance-Roff)/(Ron-Roff);
% x
w=x*D;
w =w+(dt*dwdt);
% % s = sprintf("w = %d",w);
% disp(s)
if (w<0)
   w=0;
   dwdt=0;
elseif (w > D)
   w=D;
   dwdt=0;
end
x = w/D;
```

```
memristance = Ron*x+Roff*(1-x);
current = voltage/memristance;
end
%%Generation of model 2 using crossbar%%
function cross_bar = generate_model_2(label,text)
      categories = unique(label);
    cross_bar = cell(1,length(categories));
    words = [];
    for i = 1:length(text)
         d = sprintf("Tokenising %f",i);
           disp(d);
        curr_text = text(i);
        tokens= split(curr_text)';
        words = [words tokens];
    end
    words = unique(words);
    words = [words ,"@ue"];
    for i = 1:length(categories)
        total_word_count =length(words);
        category = categories(i);
        bag = containers.Map(words,ones(1,length(words)));
        train_texts = text(find(label == category));
        for j = 1:length(train_texts)
           d = sprintf("Processing Category %d Training Text %d",i,j);
           disp(d);
            temp = train_texts(j);
            token = split(temp)';
            for word = token
                bag(word)=bag(word)+1;
                total_word_count = total_word_count+1;
            end
        end
        cross_bar{1,i}=category;
        cross_bar{2,i}=bag;
        cross_bar{3,i}=total_word_count;
    end
         for i=1:length(categories)
        key = string(keys(cross_bar{2,i}));
        value = string(values(cross_bar{2,i}));
        value = join(value);
        value = str2num(value);
        value = value/cross_bar{3,i};
        value = -1./log10(value);
        value = 1000000*value;
        cross_bar{2,i} = containers.Map(key,value);
    end
    for i = 1:length(categories)
        category = categories(i);
        count = sum(label == category);
        prob = count/length(label);
        prob = -1/log10(prob);
        prob=prob*1000000;
        cross_bar{3,i} = prob;
end
%%Evaluation%%
function [accuracy]= evaluate(model,label,text)
voltage = 0.01;
    [x,category_count] = size(model(1,:));
    categories = model(1,:);
    predicted_categories=[];
    count =1;
    for i = text
        d = sprintf("Evauating Text %d",count);
        disp(d);
        count=count+1;
        words = split(i)';
        calculated_current=[];
        for j = 1:category_count
            current=0;
            bag = model{2,j};
            for word = words
%                disp(word)
                if isKey(bag,word)
                    current_increment = mem(voltage,bag(word));
%                       current_increment = voltage/bag(word);
                    current = current+current_increment;
%                       d = sprintf("%s %f ",word,current_increment);
%                       disp(d);
                else
%                       d= sprintf("%s is not key",word)
%                       disp(d)
                    current_increment = voltage/bag("@ue");
                    current = current+current_increment;
%                       d = sprintf("%s %f ",word,current_increment);
%                       disp(d);
 end
 end
current = current + (voltage/model{3,j});
calculated_current=[calculated_current current];
end
[value,index] = min(calculated_current);
predicted_categories=[predicted_categories,categories{1,index}]        ;
end
accuracy = sum(predicted_categories == label);
accuracy = accuracy/length(label);
accuracy = accuracy*100;
end
```

A3. Realization of Memristive State Machine for Smart Edge Detector Applications

```
%%Memristor implementation of edge detection%%
Roff    =152426795;
Ron     =150792;
voff    = 1.2%Must be postive;
von     = -1.35%Must be negative;
koff    =40;
kon     =-80;
aoff    =7;
aon     =5;
D       =5e-8;
xoff    =  D;
xon     =  0;
M=zeros(255,1);
L=zeros(255,1);
x=zeros(255,1);
V=zeros(255,1);
I=zeros(255,1);
I1=imread('BITS.png'); %read image
I1=rgb2gray(I1); %convert rgb image to grayscale
I2=I1;
[m,n]=size(I1);
i=0;
for i=1:255
 L(i)=i;
 x(i)=L(i)/255; %compute fuzzy value for each pixel in the rnge
 V(i)=(((x(i)*D)/(koff*1e-3))^(1/aoff)+1)*voff;  %compute voltage values for each pixel
value that has a fuzzy value x(i)
 M(i)=x(i)*150800+(1-x(i))*152426800;  %compute memristance value for corresponding
state variable x(i)
```

```
 I(i)=V(i)/M(i); % compute current flowing through the circuit for corresponding voltage and
memristance

end

plot(V,x)

t3=1.2710e-08;  % threshold current value for reading memristor with voltage pulse that
induces state x=0.3

t7= 3.0348e-08;   % threshold current value for reading memristor with voltage pulse that
induces state x=0.7

count1=zeros(n,1);

i=1;

j=1;

c11=0;

c21=0;

c31=0;

c41=0;

P=0;

Q=0;

c12=0;

c22=0;

c32=0;

c42=0;

for i=1:m-1

   for j=1:n-1 %traverse through all pixels in the image to implemnt threshold block logic

      if(I(I1(i,j)+1)>t3) %threshold block for x=0.3 that assigns 1 if current is greater than t3
fore each pixel(four bit output of the threshold block is c11,c21,c31,c41)

          c11=1;

      else

          c11=0;

      end

      if(I(I1(i+1,j)+1)>t3)

          c21=1;

      else

          c21=0;
```

```
     end
     if(I(I1(i,j+1)+1)>t3)
        c31=1;
     else
        c31=0;
     end
     if(I(I1(i+1,j+1)+1)>t3)
        c41=1;
     else
        c41=0;
     end
          if(I(I1(i,j)+1)>t7) %threshold block for x=0.7 that assigns 1 if current is greater than
t7 fore each pixel(four bit output is c12,c22,c32,c42)
        c12=1;
     else
        c12=0;
     end
     if(I(I1(i+1,j)+1)>t7)
        c22=1;
     else
        c22=0;
     end
     if(I(I1(i,j+1)+1)>t7)
        c32=1;
     else
        c32=0;
     end
     if(I(I1(i+1,j+1)+1)>t7)
        c42=1;
     else
        c42=0;
     end
```

```
    if(c11==c21 && c11==c31 && c11==c41 && c21==c31 && c21==c41 && c41==c31)
% assign P=0 if all the four bits of the first threshold block are same
          P=0;
      else
          P=1;
      end
          if(c12==c22 && c12==c32 && c12==c42 && c22==c32 && c22==c42 &&
c42==c32)% assign Q=0 if all the four bits of the Second threshold block are same
          Q=0;
      else
          Q=1;
      end
        if(P==0 && Q==0) %assign output as an edge or not an edge depending upon P and Q
          I2(i,j)=uint8(1);
               else
          I2(i,j)=uint8(255);
      end
   end
end
count2=zeros(n,1);
for  j=1:n-1
    for i=1:m-1
       if(I2(i,j)==uint8(255))
           count1(j)=count1(j)+1;
       else
           count2(j)=count2(j)+1;
       end
    end
end
count3=zeros(n,1);
count4=zeros(n,1);
for  j=1:n-1
    for i=1:m-1
```

```
       if(I3(i,j)==1)
           count4(j)=count4(j)+1;
       else
           count3(j)=count3(j)+1;
       end
    end
end
imshow(I2)

%%Edge Detection%%
i=1;
j=1;
c1=0;
c2=0;
c3=0;
c4=0;
for i=1:249
   for j=1:319
      if(I1(i,j)>170)
          c1=3;
      elseif (I1(i,j)<84)
          c1=1;
      else
          c1=2;
      end
      if(I1(i+1,j)>170)
          c2=3;
      elseif (I1(i,j+1)<84)
          c2=1;
      else
          c2=2;
      end
      if(I1(i,j+1)>170)
          c3=3;
      elseif (I1(i,j+1)<84)
          c3=1;
      else
          c3=2;
      end
      if(I1(i+1,j+1)>170)
          c4=3;
      elseif (I1(i+1,j+1)<84)
          c4=1;
      else
          c4=2;
      end
      if(c1==c2 && c1==c3 && c1==c4 && c2==c3 && c2==c4 && c4==c3)
          I2(i,j)=uint8(1);
      else
          I2(i,j)=uint8(200);
      end
   end
end
acc=0;
count=0;
for i=1:249
    for j=1:319
        if(I3(i,j)==true && I2(i,j)==uint8(200)||I3(i,j)==false &&
I2(i,j)==uint8(1)||I3(i+1,j+1)==true && I2(i+1,j+1)==uint8(200)||I3(i+1,j+1)==false &&
I2(i+1,j+1)==uint8(1))
         count=count+1;
        end
    end
end
acc=count/79431;
```

A4. Implementation of Binary Particle Swarm Optimization for Image Thresholding using Memristive Crossbar Array

%%Python code for Two threshold Kapur’s Objective Function

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [],
   "source": [
    "#Importing Libraries\n",
    "import numpy as np\n",
    "import matplotlib.pyplot as plt\n",
    "import random\n",
    "import cv2\n",
    "from scipy import ndimage\n",
    "%matplotlib inline"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Memristor Data:\n",
    "Roff=152000000\n",
    "Ron=150000\n",
    "voff=1.35 (Must be postive);  \n",
    "von=-1.2 (Must be negative); \n",
    "koff=200\n",
    "kon=-250\n",
  "aoff=3\n",
  "aon=2\n",
  "D=50e-09\n",
  "HCL=1/Ron\n",
  "LCL=1/Roff"
 ]
},
{
 "cell_type": "code",
 "execution_count": 3,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Initializing the values\n",
  "Roff=152000000\n",
  "Ron=150000\n",
  "HCL=1/Ron\n",
  "LCL=1/Roff"
 ]
},
{
 "cell_type": "code",
 "execution_count": 4,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Class particle\n",
  "class particle():\n",
  "    def __init__(self):\n",
  "        self.global_best = np.asarray([random.randint(0,1) for i in range(16)])\n",
  "        self.position = np.asarray([random.randint(0,1) for i in range(16)])\n",
  "        self.velocity=np.zeros(16)\n",
 "        self.personal_best= np.asarray([random.randint(0,1) for i in range(16)])  "
 ]
},
{
 "cell_type": "code",
 "execution_count": 71,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Number of particles and iterations\n",
  "n_p=100\n",
  "n_iter=100"
 ]
},
{
 "cell_type": "code",
 "execution_count": 72,
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/plain": [
     "201"
    ]
   },
   "execution_count": 72,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "#possible values of c1r1 and c2r2 assuming both c1 and c2 are 2\n",
    "c1r1=[i/100 for i in range(201)]#21\n",
    "c2r2=[i/100 for i in range(201)]#21\n",
    "len(c1r1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 73,
   "metadata": {},
   "outputs": [],
   "source": [
    "def encoder(ll): #Takes a list of 0s and 1s and converts it to LCL and HCL
respectively\n",
    "    ll1=[0 for i in range(len(ll))]\n",
    "    Roff=152\n",
    "    Ron=0.15\n",
    "    HCL=1/Ron\n",
    "    LCL=1/Roff\n",
    "    for i in range(len(ll)):\n",
    "        if(ll[i]==1):\n",
    "            ll1[i]=HCL\n",
    "        else:\n",
    "            ll1[i]=LCL\n",
    "    return ll1\n",
    "#Call a write function of V+/- 2 10ns"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 74,
   "metadata": {},
   "outputs": [],
 "source": [
  "#Memristor crossbar based matrix multiplication\n",
  "def mem_multiply(P,l1=c1r1,l2=c2r2):\n",
  "    C1R1=[i*5*0.5 for i in range(len(l1))]\n",
  "    C2R2=[i*5*0.5 for i in range(len(l2))]\n",
  "    rand1=random.randint(0,len(C1R1)-1)\n",
  "    rand2=random.randint(0,len(C2R2)-1)\n",
  "    c1r1gb=[l1[rand1]*P.global_best[i] for i in range(len(P.global_best))]\n",
  "    c1r1p=[l1[rand1]*P.position[i] for i in range(len(P.global_best))]\n",
  "    c2r2pb=[l2[rand2]*P.personal_best[i] for i in range(len(P.global_best))]\n",
  "    c2r2p=[l2[rand2]*P.position[i] for i in range(len(P.global_best))]\n",
  "    term_g=[c1r1gb[i]-c1r1p[i] for i in range(len(c1r1p))]\n",
  "    term_p=[c2r2pb[i]-c2r2p[i] for i in range(len(c2r2p))]\n",
  "    return term_g,term_p\n",
  "#     rand1=C1R1[random.randint(0,len(C1R1))]\n",
  "#     c1r1_gb_p=[rand1*(P.global_best[i]-P.position[i]) for i in range(len(P.position))]  "
 ]
},
{
 "cell_type": "code",
 "execution_count": 75,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Decoder Function\n",
  "def decoder(term_g,term_p,lambda_v=None):\n",
  "    lis1=np.zeros(16)\n",
  "    lis2=np.zeros(16)\n",
  "    term_g=np.asarray(term_g)\n",
  "    term_p=np.asarray(term_p)\n",
  "    term_gm=term_g/2.5\n",
  "    term_pm=term_p/2.5\n",
  "    for i in range(16):\n",
  "        if(abs(term_gm[i])<1.3158):\n",
  "            lis1[i]=0\n",
  "        else:\n",
  "            lis1[i]=term_gm[i]/6.66\n",
  "        if(abs(term_p[i])<0.0066):\n",
  "            lis2[i]=0\n",
  "        else:\n",
  "            lis2[i]=term_pm[i]/6.66\n",
  "    return lis1,lis2"
 ]
},
{
 "cell_type": "code",
 "execution_count": 76,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Binary to decimal converter\n",
  "def bToD(binary): \n",
  "\tbinary1 = binary \n",
  "\tdecimal, iii, ni = 0, 0, 0\n",
  "\twhile(binary != 0): \n",
  "\t\tdec = binary % 10\n",
  "\t\tdecimal = decimal + dec * pow(2, iii) \n",
  "\t\tbinary = binary//10\n",
  "\t\tiii += 1\n",
  "\treturn decimal"
 ]
},
{
 "cell_type": "code",
   "execution_count": 77,
   "metadata": {},
   "outputs": [
    {
     "data": {
“Image.png”:”/n”
      "text/plain": [
       "<Figure size 432x288 with 1 Axes>"
      ]
     },
     "metadata": {
      "needs_background": "light"
     },
     "output_type": "display_data"
    }
   ],
   "source": [
    "#Reading and visualizing the image\n",
    "pict=cv2.imread('B9.png',0)\n",
    "plt.imshow(pict,cmap='gray',vmin=0,vmax=255)\n",
    "row,col=pict.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 78,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
    "[4.96185303e-01 0.00000000e+00 8.17871094e-03 0.00000000e+00\n",
   ]
  }
 ],
 "source": [
  "#Image Histogram\n",
  "p_i=np.zeros(256)\n",
  "for r in range (row):\n",
  "    for c in range (col):\n",
  "        p_i[pict[r][c]]+=1\n",
  "p_i=p_i/(row*col)\n",
  "print(p_i)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 79,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Swarm Initialization\n",
  "l=[]\n",
  "for i in range (n_p):\n",
  "    l.append(particle())"
 ]
},
{
 "cell_type": "code",
 "execution_count": 80,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Sigmoid Function\n",
  "def sigmoid(z):\n",
  "    return 1/(1+np.exp(-z))"
 ]
},
{
 "cell_type": "code",
 "execution_count": 81,
 "metadata": {},
 "outputs": [],
 "source": [
  "#Takes a list of 0s and 1s and returns the corresponding Decimal Number\n",
  "def c(x):\n",
  "    pp=[str(int(x[i1])) for i1 in range (len(x))]\n",
  "    pp=list(pp)\n",
  "    if(pp==[str(int(0)) for qr in range(len(pp))]):\n",
  "        return 0\n",
  "    while(pp[0]=='0'): \n",
  "        pp.pop(0)\n",
  "#     if(len(pp)==0):\n",
  "#         return 0\n",
  "    else:\n",
  "        lan=''\n",
  "        for i in range(len(pp)):\n",
  "            lan+=pp[i]\n",
  "        return bToD(int(lan))"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
"outputs": [],
"source": [
 "#Kapur's Entropy Function for 2 thresholds\n",
 "def f(a,b,P_list=p_i):\n",
 "    cp=0\n",
 "    p1=[]\n",
 "    p2=[]\n",
 "    p3=[]\n",
 "    if(a>b):\n",
 "        a,b=b,a\n",
 "        \n",
 "    for cp in range((len(p_i))):\n",
 "        if(cp<a):\n",
 "            p1.append(p_i[cp])\n",
 "        elif(cp>=a and cp<b):\n",
 "    #while(cp>=a and cp<len(p_i)):\n",
 "            p2.append(p_i[cp])\n",
 "        elif(cp>=b and cp<len(p_i)):\n",
 "            p3.append(p_i[cp])\n",
 "    w1=sum(p1)\n",
 "    w2=sum(p2)\n",
 "    w3=sum(p3)\n",
 "    \n",
 "#    if(p1==0 or p2==0):\n",
 "#        return 0\n",
 "    H1=-sum([p1[rr]/w1*np.log(p1[rr]/w1) for rr in range(len(p1)) if p1[rr]!=0])\n",
 "    H2=-sum([p2[rr]/w2*np.log(p2[rr]/w2) for rr in range(len(p2)) if p2[rr]!=0])\n",
 "    H3=-sum([p3[rr]/w3*np.log(p3[rr]/w3) for rr in range(len(p3)) if p3[rr]!=0])\n",
 "    return H1+H2+H3\n",
 "        "
]
},
  {
   "cell_type": "code",
   "execution_count": 82,
   "metadata": {},
   "outputs": [],
   "source": [
    "#Initializing W\n",
    "W=1.0"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 83,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[  9.06388134  60.          97.         185.        ]\n",
     ]
    }
   ],
   "source": [
    "tv=np.asarray([f(c(list(map(int,l[k].position[:8]))),c(list(map(int,l[k].position[8:])))) for k
in range(n_p)])\n",
    "# tv=[]\n",
    "# for k1 in range(n_p):c(list(map(int,l[k].position[:8])))\n",
    "#     cc=c(list(map(int,l[k1].position))\n",
    "#     tv[k1]=f(cc)\n",
    "# n_p=10\n",
    "g_bval=np.zeros(4,)    \n",
   "tv1=tv\n",
   "for i in range(n_iter):\n",
   "#     tv1=np.asarray([f(c(list(map(int,l[k].position)))) for k in range(n_p)])
#list(map(int,l[k].position))\n",
   "    tv1=np.asarray([f(c(list(map(int,l[k].position[:8]))),c(list(map(int,l[k].position[8:])))) 
for k in range(n_p)])\n",
   "    if(g_bval[0]<max(tv1)):\n",
   "        \n",
   "        g_bval[1]=np.argmax(tv1)\n",
   "        g_bval[0]=max(tv1)\n",
   "        g_bval[2], g_bval[3]
=c(l[int(g_bval[1])].position[:8]),c(l[int(g_bval[1])].position[8:])\n",
   "        global_best=l[int(g_bval[1])].position\n",
   "    for j in range(n_p):\n",
   "        if(tv1[j]>tv[j]):\n",
   "            l[j].best_value=tv1[j]\n",
   "            l[j].best_position=l[j].position\n",
   "    print(g_bval)\n",
   "    tv=tv1\n",
   "#     print(tv)\n",
   "# for i in range(100):\n",
   "# # while(f(c(global_best))<8):    \n",
   "#     tv1=np.asarray([f(c(list(map(int,l[k].position)))) for k in range(n_p)])
#list(map(int,l[k].position))\n",
   "# #     print(tv1)\n",
   "#     if(g_bval[0]<max(tv1)):\n",
   "#         g_bval[1]=np.argmax(tv1)\n",
   "#         g_bval[0]=max(tv1)\n",
   "#         global_best=l[int(g_bval[1])].position\n",
   "#     for j in range(n_p):\n",
   "#         if(tv1[j]>tv[j]):\n",
   "#             l[j].best_value=tv1[j]\n",
   "#             l[j].best_position=l[j].position\n",
   "# #     if(g_bval[0]!=max(tv1)):\n",
   "# #         g_bval[1]=np.argmax(tv1)\n",
   "# #         g_bval[0]=max(tv1)\n",
   "# #         global_best=l[int(g_bval[1])].position\n",
   "#     print(g_bval)\n",
   "#     tv=tv1\n",
   "    \n",
   "#     for j in range(n_p):\n",
   "#         if(tv1[j]>tv[j]):\n",
   "#             l[j].personal_best=l[j].position\n",
   "#     global_best=l[np.argmax(tv)].position\n",
   "#     lis_gb.append(c(global_best))\n",
   "#     for part in (l):\n",
   "#         part.global_best=global_best\n",
   "    \n",
   "#     tv=tv1\n",
   "#     lis_max.append(np.mean(tv))\n",
   "    for ii, j in enumerate(l):\n",
   "        \n",
   "        encoder(j.position)\n",
   "        encoder(j.global_best)\n",
   "        encoder(j.personal_best)\n",
   "        tg,tp=mem_multiply(j)\n",
   "        lll1,lll2=decoder(tg,tp)\n",
   "        lc1r1=np.asarray(lll1)\n",
   "        lc2r2=np.asarray(lll2)\n",
   "        j.velocity=W*np.asarray(j.velocity)+lc1r1+lc2r2\n",
   "        j.velocity=np.asarray([min(max(-4,j.velocity[tt]),4) for tt in
range(len(j.velocity))])\n",
   "        for d in range(8):\n",
   "            rb=random.random()\n",
   "            if(rb<sigmoid(j.velocity[d])):\n",
       "               j.position[d]=int(0)\n",
       "           else:\n",
       "               j.position[d]=int(1)\n",
       "               \n",
       "       \n",
       "       \n",
       "       \n",
       "    "
      ]
     },
     {
      "cell_type": "code",
      "execution_count": 88,
      "metadata": {},
      "outputs": [
       {
        "data": {
   “Image.png”:”/n”
         "text/plain": [
          "<Figure size 432x288 with 1 Axes>"
         ]
        },
        "metadata": {
         "needs_background": "light"
        },
        "output_type": "display_data"
       }
      ],
      "source": [
       "#Resulting Image viewing and Storing\n",
       "p=pict\n",
       "final=np.zeros((row,col))\n",
     "a,b=sorted([g_bval[2],g_bval[3]])\n",
     "for rows in range (row):\n",
     "    for cols in range (col):   #175.         111.\n",
     "        if(p[rows][cols]<=a): #10     #96 160\n",
     "            final[rows][cols]=0\n",
     "        elif(p[rows][cols]>a and p[rows][cols]<b): #125\n",
     "            final[rows][cols]=1\n",
     "        else:\n",
     "            final[rows][cols]=2\n",
     "plt.imshow(final,cmap='gray',vmin=0, vmax=2)\n",
     "# plt.savefig(\"PSO_MEM\\B9K_{}_{}.png\".format(a,b))"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 24,
    "metadata": {},
    "outputs": [
     {
      "data": {
       "text/plain": [
        "9.2103349274696"
       ]
      },
      "execution_count": 24,
      "metadata": {},
      "output_type": "execute_result"
     }
    ],
    "source": [
     "#Brute Force Optimization\n",
     "mmm=0\n",
     "tt1=0\n",
     "tt2=0\n",
     "for t1 in range(256):\n",
     "    for t2 in range (t1,256):\n",
     "        if(f(t1,t2)>mmm):\n",
     "            tt1=t1\n",
     "            tt2=t2\n",
     "        mmm=max(mmm,f(t1,t2))\n",
     "mmm"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 25,
    "metadata": {},
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "119 185\n"
      ]
     }
    ],
    "source": [
     "print(tt1,tt2)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {},
    "outputs": [],
    "source": []
   },
```

Publications:

1. **Priyanka B. Ganganaik,** G. Abhijith, P. Michael Preetam Raj, H. Renuka, BVVSN Prabhakar Rao, and Souvik Kundu. "Realization of Memristive State Machine for Smart Edge Detector Applications." IETE Journal of Research (2020): 1-11.

2. **Priyanka B. Ganganaik,** Aditya Viswakumar, P. Michael Preetam Raj, BVVSN Prabhakar Rao, and Souvik Kundu. "Memristor-based in-memory processor for high precision semantic text classification." Computers & Electrical Engineering 92 (2021): 107160.

3. **Priyanka B. Ganganaik,** Omkar Mukul Gowaikar, V. Jeffry Louis, Rajesh K. Tripathy, Venkateswaran Rajagopalan, B. V. V. S. N. Prabhakar Rao, and Souvik Kundu. "Implementation of Binary Particle Swarm Optimization for Image Thresholding using Memristor Crossbar Array." In Advances in Electrical and Computer Technologies, pp. 915-936. Springer, Singapore, 2022.

4. Souvik Kundu, **Priyanka B. Ganganaik,** Jeffry Louis, Hemanth Chalamalasetty, and BVVSN Prabhakar Rao. "Memristors Enabled Computing Correlation Parameter In-Memory System: A Potential Alternative to Von Neumann Architecture." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 30, no. 6 (2022): 755-768.

5. **Priyanka B. Ganganaik,** Plava Kattamuri, and BVVSN Prabhakar Rao "Memristor Based Image Fusion Architecture Using Iterative Kernel PCA" submitted to Indian Journal of Engineering and Material Sciences.

Biography of Priyanka B G

Priyanka B G completed his B.Tech (Telecommunication Engineering) from Visveswaraya Technological University, Belgaum, in the year 2012 and her Master of Technology (Digital electronics and Communication systems) in 2014 from Visveswaraya Technological University, Belgaum. Prior to joining the Ph.D. program at BITS-Pilani Hyderabad Campus, she had obtained one year and ten months of teaching experience from a reputed institute in Karnataka. She has joined for a Ph.D. (2018) in the Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani– Hyderabad Campus, India, under the supervision of Prof. BVVSN Prabhakar Rao. Her current research interests include modeling or simulating the memristor-based circuits, their applications in the area of signal processing, and machine learning for devices.

<u>Biography of Prof. BVVSN Prabhakar Rao</u>

Professor BVVSN Prabhakar Rao received his M.Tech and Ph.D. from IIT Delhi. He is Presently working at BITS-Pilani, Hyderabad Campus as Professor in the Electrical and Electronics engineering department. His research areas include studies on energy management, solar energy, biomedical signal processing, and image processing, in which he has more than 20 publications in journals and conferences.