CORTICAL COLUMNS: A NON VON NEUMANN COMPUTATIONAL ABSTRACTION

by

Atif G. Hashmi

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2011

CORTICAL COLUMNS: A NON VON NEUMANN COMPUTATIONAL ABSTRACTION

submitted to the Graduate School of the
University of Wisconsin-Madison
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy

By

Atif G. Hashmi

Date of final oral examination: November 23, 2011
Month and year degree to be awarded: December 2011

The dissertation is approved by the following members of the Final Oral Committee:

Mikko Lipasti, Professor, Department of Electrical and Computer Engineering

Timothy Rogers, Assistant Professor, Department of Psychology

Matthew Banks, Associate Professor, Department of Anesthesiology

Yu Hen Hu, Professor, Department of Electrical and Computer Engineering

Nam Kim, Assistant Professor, Department of Electrical and Computer Engineering

I dedicate this thesis to my parents Gul Mohammad Hashmi and Zahida Hashmi and to my wife Irrum Hashmi for their continuous support and encouragements

ACKNOWLEDGMENTS

Throughout the course of my graduate studies, I received support and encouragements from several people. I take this opportunity to show my gratitude to all of them.

First of all, I thank my parents Gul Mohammad Hashmi and Zahida Hashmi for their continuous and unconditional support and love. They taught me never to give up and always strive to succeed. They stood by me through the thick and thin, believed in me, and prayed for me during all of my research endeavors. I know I could not have achieved all the success in my life without the sacrifices they have made for me.

I thank my loving wife, Irrum, for being extremely understanding of my schedule during these last few months when I was quite busy wrapping up my research and writing my dissertation. She made sure that I do not have to worry about anything else apart from working on my research and writing my dissertation. As a result, these last few months when I was wrapping things up went by quite smoothly.

I extend my deepest gratitude to my research adviser, Professor Mikko Lipasti, for providing me with an opportunity to work with him and conduct research on an extremely novel and interesting topic. Even though the research endeavor that I undertook was extremely risky, I realize that Mikko's unconditional guidance and eternal optimism made it a risk worth taking. My interactions with Mikko over the last few years have allowed me to significantly improve upon myself both personally and professionally. It was a pleasure and an honor to work under his supervision.

I also thank all of my collaborators Olivier Temam, Hugues Berry, and Andrew Nere.

Together, all of us were able to convince the computer architecture community that now is the right time to explore computational paradigms other than the traditional von Neumann model. We received a lot of criticisms and setbacks but we did not give up and eventually succeeded. I thank you all for your commitment towards the end goal, for all your efforts, and for your comments and suggestions. I am especially grateful to Andrew Nere for all the fruitful discussions we had during the course of this project and for providing me with his insightful comments and feedback. I look forward to collaborating with all of them in the future.

I extend my acknowledgments to my present and former lab-mates including Arslan Zulfiqar, Andrew Nere, Sean Franey, Mitch Hayenga, David Palframan, Vignyan Naresh, Natalie Jerger, Erica Gunadi, and Dana Vantrease for their company and friendship.

Finally, I thank my thesis committee for their insightful feedback and comments, overall interest in my research, and general support throughout the time I was conducting my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Recent advances in the understanding of the structure and function of the mammalian brain have provided researchers with an opportunity to investigate computational paradigms other than the traditional von Neumann model. These brain-like architectures, which are premised on our understanding of how the human neocortex computes, have the potential to be fault-tolerant, power-efficient, easily programmed, and manage to solve several difficult problems more reliably than traditional computational approaches.

This dissertation seeks to challenge the contemporary techniques for implementing computational models both in software and in hardware. Further, it argues that the traditional von Neumann model of computation is under growing pressures in terms of power dissipation, reliability, and programmability, therefore it is essential to investigate alternate computational paradigms that can cater to the needs of the future generation workloads and processing hardware. In this effort, this dissertation proposes a computational model inspired by the structural and functional properties of the neocortex and highlights various biologically inspired aspects which make such a model superior to conventional computational approaches. Further, these biologically inspired aspects endow the model's software implementation the ability to intrinsically preserve its functionality even in the presence of faulty hardware, without requiring any reprogramming or recompilation. Finally, this dissertation also establishes a symbiotic relationship between computer architecture and complex biological networks by proposing mechanisms that allow both of these fields co-evolve benefiting from each other.

This dissertation is geared towards developing a comprehensive and biologically inspired

understanding of the microarchitecture of computing systems that mimic the human neocortex, and applying such systems for robust realization of complex tasks.

1 INTRODUCTION

The original *von Neumann model*[1] of a computing unit has been a relatively nice fit for the technology evolutions of the past four decades. However, it is hard not to notice that this model is under growing pressure. The power dissipation bottleneck has made architects shift their focus to many-core architectures but the programming bottleneck of many-cores raises doubts on the ability to truly take advantage of many-core systems. More recently, the reliability bottleneck brings a whole new set of challenges to the table. Architects have attempted to meet all these challenges, but the proposed solutions progressively erode performance scalability.

1.1 Recent Challenges to the Von Neumann Model

In the recent years, the widely accepted von Neumann model of computation has been challenged by following three major issues.

1.1.1 Power Dissipation

Traditionally, the decrease in the feature size of individual transistors (technology scaling) has been governed by Moore's law [93] i.e. the transistor size is halved approximately every two years. This means that the number of transistors that can be packaged within a unit area of a chip doubles every two years. In terms of power dissipation, this increase in the

[1] A design model for a stored-program digital computer that uses a central processing unit (CPU) and a single separate storage structure (memory) to hold both instructions and data.

number of transistor per unit area along with their increased operating frequency increases the overall power dissipation of the chip [14]. This is mainly from an increase in leakage current and from the repeated capacitance charge and discharge on the output of billions of transistors in today's chips [72]. A decade ago, this increase in the power dissipation was not an issue but recently, with the advent of mobile computing and portable devices, it has become a major concern for computer processor designers [72, 96], and many software and hardware schemes including [32, 50, 58, 83] have been proposed to resolve this issue.

1.1.2 Reliability

One of the most common techniques to reduce the power dissipation due to leakage current is voltage scaling. In order to reduce the power dissipated by transistors, the voltage applied across individual transistors is decreased. This reduction in the applied voltage is concomitant with a degradation in the reliability of a transistor. Single-event upsets, also known as soft errors [7], are a major source of this degradation. Soft errors are caused by alpha particles that may exist in the chip material and in cosmic rays from space. With a reduced transistor feature size and decreased applied voltage, a small amount of charge is required to flip the state of a transistor and a small influx of alpha particles is enough for such an event to take place. This degradation in the reliability of the transistors will continue to increase over time [14, 97] and has already forced computer architects to rethink modern processor design. Some of these designs are discussed in [13, 26, 51, 91].

1.1.3 Programmability

Both the power dissipation and reliability issues have made architects shift their focus to many-core architectures, but the programming bottleneck of such architectures raises doubts on the ability to truly take advantage of many-core systems. Since most of the current generation programming languages, compilers, and software have been developed for a uniprocessor computational paradigm, there seems to exist a *programmability gap* between many-core based systems and current generation applications. Programming of many-core systems has emerged as one of the main issues that the computer industry has yet to solve.

1.2 Biologically Inspired Computing

With such limitations (as described in Section 1.1), it now makes sense to investigate alternative computational models better suited to cope with the technology evolution. Even the upcoming computational technologies like ultra-CMOS [122], nanotubes [25], and memristors [135] will suffer from the same limitations. Technology scaling will continue to provide an increasing number of transistors/elements on a single chip. These elements will not necessarily be much faster (slower in some cases) and will come with a growing number of defects and faults. Considering that these basic elements will be slower and faulty, it is hard not to observe that in order to realize complex computational tasks, nature has already found a way to package a large number of elements with similar properties in the form of the mammalian brain.

Considering the abilities of the brain, it is clear that computer architects should leverage the tremendous progress made in understanding the working of the brain to develop biologically inspired computational models. Computer architects are uniquely positioned for this task because they can apply traditional system design approaches to architect biologically inspired computing systems. These approaches include understanding how to combine and control biologically inspired elementary components hierarchically into increasingly complex building blocks, defining a programming approach for these computing systems, understanding their potential applications scope, and understanding the appropriate modeling level to integrate billions of components without being overwhelmed by complexity nor missing key properties.

1.3 Dissertation Contributions

Various contributions of this dissertation can be grouped into two major categories. First, this dissertation investigates a biologically inspired computational model inspired by the structural and functional properties of the mammalian brain. The emphasis here is to develop a biologically inspired hierarchical learning model, to justify its biological basis, and to evaluate its performance compared to traditional artificial intelligence approaches. Second, this dissertation discusses various architectural aspects that help generalize and optimize large biologically inspired learning networks and describes various biological properties that can help develop robust and fault-tolerant computational models.

1.3.1 Biologically Inspired Learning Model

In addition to presenting the implementation details of the biologically inspired model, this dissertation also puts an emphasis on the biological justifications for various components of the proposed model. In this regard, this dissertation makes the following contributions.

1.3.1.1 Cortical Column Model

The first major of contribution of this dissertation is that it presents a biologically inspired learning model motivated by the properties of the cortical columns [94, 95] that exist throughout the mammalian neocortex. Rather than modeling individual neurons as the basic implementation abstraction, this dissertation proposes using cortical columns as the basic functional unit within an intelligent learning network. This results in models that are both biologically inspired and computationally efficient as a single column can abstract the functionality of thousands of neurons.

1.3.1.2 Learning via Spontaneous Activations

This dissertation proposes using spatially localized and temporally correlated spontaneous activations rather than randomized weights [99] for triggering initial learning behavior within the network. This is a completely novel approach and has a strong biological basis. Unlike traditional artificial neural networks, spontaneous activations help the proposed cortical model avoid the trap where certain unique features in the training dataset are not learned because of the network's inability to distinguish them due to the initial random weights.

1.3.1.3 Feedback for Learning Invariant Representations

Due to the limited understanding of the role of feedback processing during learning, contemporary learning models do not effectively utilize feedback connections during the training phase. This dissertation presents a very powerful role that feedback connections can play in terms of learning variations of the same object to generate an invariant representations for each unique object.

1.3.1.4 Biologically Inspired Mechanisms for Developing Self Organizing Maps

Finally, combining the ideas of spatially localized and temporally correlated spontaneous activations and object permanence, this dissertation describes a biologically inspired technique for generating hierarchical self organizing feature maps. This technique overcomes the two main issues the traditional self organizing maps face, i.e. learning hierarchical feature maps and their computational requirements.

1.3.2 Architectural Solutions and Opportunities

In terms of utilizing computer architecture concepts to generalize and optimize biologically inspired networks and utilizing biological properties to develop robust computational models, this dissertation makes the following contributions.

1.3.2.1 Neuromorphic Instruction Set Architecture

With the advent of multiple neuromorphic hardware and software, there is a need to develop an abstraction layer that separates neural algorithm from the execution substrate. Such abstractions allow the programmer to implement complex neural networks without worrying about the details of the underlying hardware. This dissertation advocates the use of a neuromorphic instruction set architecture to realize such a goal.

1.3.2.2 Cortical Network Optimizations

Utilizing a neuromorphic instruction set architecture [46], this dissertation also discusses various optimizations that can enhance the functionality of a complex neural network. These optimizations result in reducing the overall complexity as well as the execution time of large neural networks.

1.3.2.3 Tolerance to Permanent Faults

Utilizing the ideas of automatic abstraction and learning via spontaneous activations [43], the proposed biologically inspired computational model demonstrates inherent tolerance to permanent hardware failures and can execute without any reprogramming or recompilation on a hardware substrate experiencing permanent faults. The performance of the proposed learning model gracefully degrades with an increase in the permanent hardware faults and even when 50% of the hardware is damaged, with retraining the software model can recover to achieve 100% functionality.

1.4 Related Published Work

This dissertation encompasses work that has appeared in four conference proceedings and will appear in one accepted journal paper.

- **Discovering Cortical Algorithms (ICNC - 2010).** This paper [44] provides the feed-forward cortical column model details and justifies its biological basis. This paper was coauthored by Mikko Lipasti.

- **Learning via Spontaneous Activations (ICCNS - 2011).** This article [45] describes the use of spatially localized and temporally correlated spontaneous activations and their role in developing hierarchical self organizing maps. This paper was coauthored by Andrew Nere and Mikko Lipasti.

- **Neuromorphic ISAs (ASPLOS - 2011).** This paper [46] discusses the need for an abstraction that separates the neural algorithm from the execution substrate using an xml-based neuromorphic instructions set architecture. This work was done in collaboration with Andrew Nere, James Thomas, and Mikko Lipasti.

- **Automatic Abstraction and Fault Tolerance (ISCA - 2011).** This article [43] details the use of hierarchal networks for generation of automatic abstractions and to develop biological learning models that are inherently fault tolerant. This paper was coauthored by Hugues Berry, Olivier Temam, and Mikko Lipasti.

- **Feedback Processing and Invariant Representations (SCI - 2011)** This journal article (to be published) provides the details of incorporating feedback processing in the

cortical column model, interactions between the feed-forward and feedback processing networks, and the role of feedback in learning variations of the same pattern. This article was coauthored by Mikko Lipasti.

1.5 Dissertation Structure

The rest of this dissertation is organized as: Chapter 2 provides background material pertaining to the structural and functional aspects of the mammalian brain and describes various neocortical properties that inspire the proposed learning model. Chapter 3 briefly describes the research in artificial intelligence and computer architecture that relates to the model proposed in this dissertations. Chapter 4 provides a detailed description of the biologically inspired computational model proposed in this dissertation. Chapter 5 establishes a bidirectional relationship between conventional computer architecture approaches and biologically inspired networks and describes how both the fields can benefit from each other. Chapter 6 concludes the dissertation and discusses several interesting ideas that initiated during the course of this project but are not part of this dissertation.

2 BACKGROUND

This chapter describes various biological phenomena that influence different components of the proposed biologically inspired computational model. It also familiarizes the general reader with these phenomena by explaining various terms and concepts and providing factual data.

2.1 Cortical Structures

2.1.1 The Mammalian Brain

The mammalian brain can be divided into two main parts; the old brain and the new brain. The old brain constitutes the parts that developed early in evolution, including pathways from sensory modalities to the new brain, spinal cord, and other parts that deal with instinctive behavior and basic urges. The new brain, also referred to as the neocortex, is the part of the brain that developed later in evolution. Figure 2.1 provides a very high level diagram of the human brain. In this figure, the regions titled middle brain and reptilian brain comprise the old brain while the top convoluted part is the neocortex.

2.1.2 The Neocortex

The neocortex is unique to mammals and is highly developed for humans; it accounts for about 77% of the human brain (in volume) [129] and is responsible for perception, language, mathematics, planning, and all the other aspects necessary for an intelligent system. For a typical adult, it is estimated the neocortex has around 11.5 billion neurons and 360 trillion

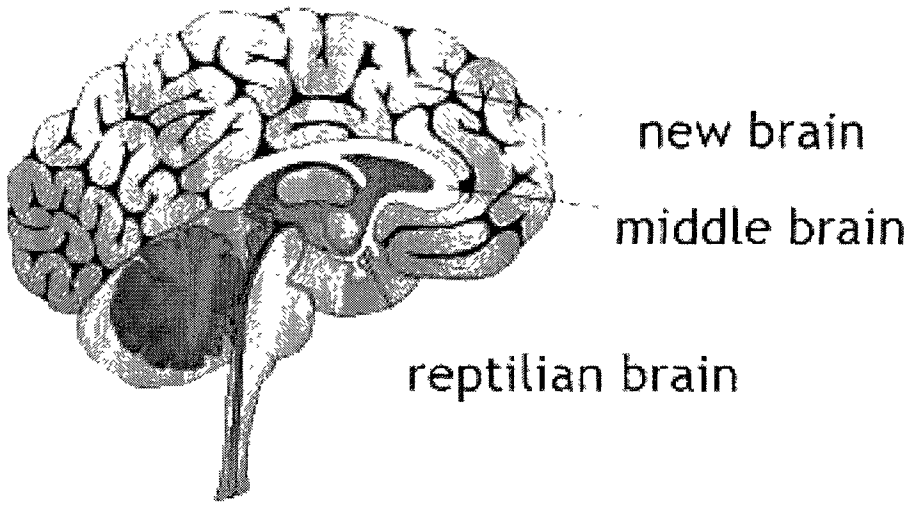

Figure 2.1: Overview of the human brain. Old brain constitute the middle brain and the reptilian complex. The upper convoluted part is the neocortex. [38].

synapses, or connections between neurons [111].

A very intriguing property of the neocortex is its apparent structural and functional uniformity [95]. Because of this property, the regions of the neocortex that process auditory inputs, for instance, appear very similar to the regions that handle visual processing. This uniformity suggests that even though different regions specialize in different tasks, they employ the same underlying processing algorithm.

2.1.3 Neurons and Synapses

Neurons and synapses are the most well-known elementary building blocks of the brain. A neuron performs two types of operations: it sums its inputs (dendrites) and it triggers a spike at its output (axon) if the sum is beyond a certain threshold. Typical firing interval for a neuron is 20ms to 200ms [76], orders of magnitude slower than CMOS transistors switching

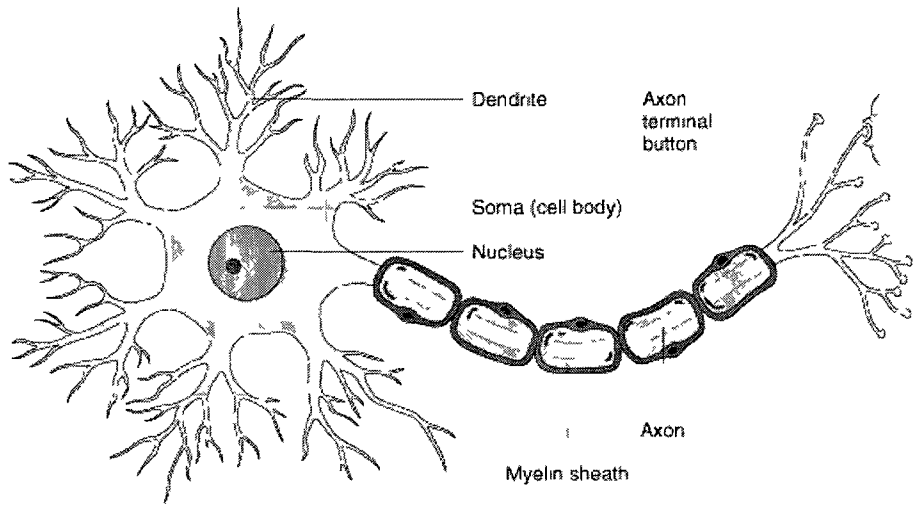

Figure 2.2: Diagram of a typical neuron [9].

times. A synapse is the connection between two neurons, each neuron has hundreds to thousands of synaptic connections. Neurons communicate with each other through these synapses by generating electrical spikes and releasing neurotransmitters. Figure 2.2 shows the structure of a typical neuron.

2.1.4 Receptive Field

The receptive field is a tool used by biologists to describe what a neuron *sees*. Typically, it defines the set of input connections to a neuron. A neuron responds only to the activations that occur within its receptive field. Experiments by Hubel and Wiesel [59] suggest that receptive fields of neurons in a certain cortical region consist of multiple neurons at lower cortical regions. Thus, neurons at higher cortical regions progressively combine small and simple receptive fields of their input neurons to develop large and complex receptive fields.

This ability of the neurons to progressively broaden their receptive fields is implemented in the biological inspired computational model proposed in this dissertation and is described in Section 4.3.

2.1.5 Cortical Columns

Within the neocortex, neurons are vertically grouped into structures called cortical columns. This columnar organization of the neurons was first observed by a neuroscientist Vernon Mountcastle as described in his seminal paper in 1957 [94]. Anatomically, a cortical column consists of six layers [17, 53, 111]. Feed-forward information from lower cortical regions is received by Layer IV. Layer IV sends this information to Layers II and III which communicate it to higher cortical regions. Similarly, feedback information from the higher cortical regions is received by Layer I and is transfered to Layers II and III. Layers II and III then communicate that information to the Layers V and VI. Layers V and VI then transfer this information to lower cortical regions. Apart from these vertical paths, which convey information up and down the hierarchy, there are also horizontal paths between the cortical columns at the same level. Layers II and III connect the columns at the same hierarchical level with each other.

The cortical columns are further classified into minicolumns and hypercolumns [95]. A hypercolumn comprises around 50-100 minicolumns bound together with lateral or horizontal connections. Each minicolumn is composed of around 200 neurons and minicolumns within the same hypercolumnn share the same receptive field. The term cortical column is sometimes used for both types of columns, though, in biology, it usually refers to a hypercolumn. Figure 2.3

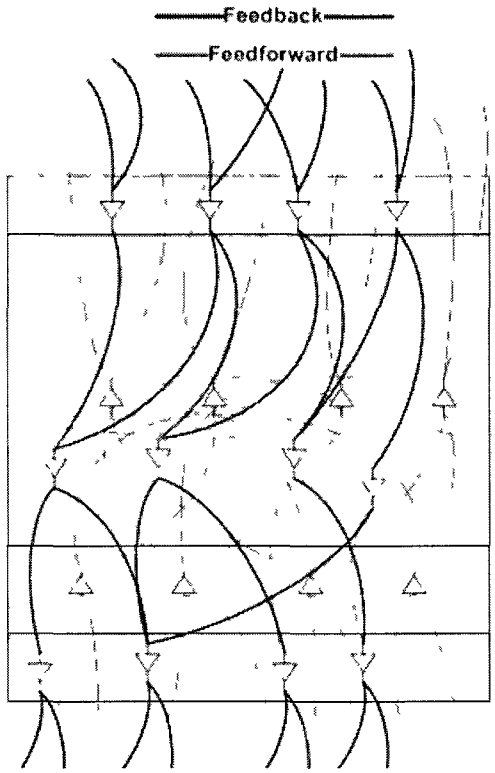

Figure 2.3: Forward, feedback and lateral connections between neurons and cortical columns. Neurons are represented by triangles in the figure.

shows a high level diagram of a column. Section 4.2 describes how the proposed computational model implements various functional aspects of minicolumns and hypercolumns.

2.2 Cortical Organization

Section 2.1 describes various structural and functional abstraction that exist within the mammalian brain. Along with these abstractions, their actual organization within the brain results in making the brain such a robust and powerful processing system.

2.2.1 Hierarchy

The cortical columns within the neocortex are organized in the form of a hierarchy. Columns at the lower levels extract simple features exciting their receptive fields and communicate their outputs to columns in the upper levels which extract more complex features. Columns at the higher levels also make projections to the columns in the lower levels and modulate the behavior of the lower levels.

Among others, [60, 95] present a detailed description of the arrangement and functionality of hypercolumns and minicolumns within the neocortex. Their findings suggest that minicolumns at the lower levels of the hierarchy learn to identify very basic features (like edges of different orientation) and communicate their responses to minicolumns at the higher levels. The minicolumns at the higher cortical levels combine the outputs to multiple lower level minicolumns to identify more complex shapes. Such a hierarchical organization of hypercolumns is discussed in Section 4.3 in the context of the proposed computational model.

2.2.2 Neocortical Connections

Within and across cortical columns, there are numerous forward, backward (called feedback) and lateral connections. These connections can be either excitatory (they can increase the output of the target neuron) or inhibitory (they decrease its output). This multi-directional flow of information can be observed at different levels of granularity: within a cortical column, across cortical columns, and across regions which derive from the hierarchical organization of cortical columns. Columns at the lower levels communicate information to the higher

levels via the feedforward connections while the columns at the higher levels project into the lower level columns via the feedback paths and modulate the lower levels' behavior in the presence of a global context [31]. These feedback paths that bring predictive information from the higher regions to the lower ones are one of the most important and powerful features of the neocortex. Research shows that the number of feedback paths taking predictions down the hierarchy is significantly greater than the number of feedforward paths going up the hierarchy [16, 31]. This clearly suggests the importance of the feedback predictive paths.

The minicolumns within the same hypercolumn are connected with each other through inhibitory lateral connections. These inhibitory connections may implement a competitive learning behavior [112] among minicolumns within a hypercolumn. Studies including [60] hypothesize that the minicolumns use these connections to learn unique and independent features from set of inputs exciting their receptive fields. Figure 2.3 exhibits the three types of connections that exist among various neuron populations within a cortical column.

Sections 4.2.3, 4.3.1, and 4.3.2 describe how the computational model proposed in this dissertation implements forward, backward, and lateral connections and the roles each of these connections plays in order to realize complex processing tasks.

2.3 Cortical Operations

The hierarchical organization of the neocortex along with the various types of connections provides a number of powerful operations. Some of these operations are discussed in this section.

2.3.1 Independent Feature Identification

The lateral or horizontal links among minicolumns within the same hypercolumn are responsible for dimensionality reduction, which plays a critical role during the learning process. Minicolumns monitor the activity of nearby minicolumns within the same hypercolumn using these horizontal paths, and can modify the synaptic connections of their neurons to identify features from the data that are not being detected by other columns [40]. At the same time, the minicolumns might also use these horizontal connection to determine if they are generating redundant information i.e. information that is being generated by some other minicolumn in the same network. By doing so, each of the minicolumns in the same network learns to detect independent features from the input data. One of the classical examples of this behavior is the primary visual cortex. Minicolumns in the primary visual cortex train themselves to identify edges of different orientation from the image formed at the retina [59, 60, 82]. Each of the edges can be treated as an independent feature. A direct consequence of independent feature detection is that it reduces the dimensionality of the data. Once the columns train themselves to identify independent features from the input, it gets very easy to identify columns that are providing redundant information. The outputs of columns that generate redundant information can be ignored, and the columns themselves can be pruned and reassigned to other tasks. Such an ability is provided by the computational model proposed in this dissertation and is described in details in Section 4.2.3.

2.3.2 Automatic Abstraction

It is believed that cortical regions operate by progressively abstracting and manipulating increasingly complex notions throughout the neural hierarchy [103]. For instance, from the set of pixels of an image, the visual cortex will first identify segments, then elementary shapes such as angles and intersections, and increasingly complex combinations, such as objects found in our environment, see Figure 2.4. This automatic abstraction capability for various inputs partly explains why the neocortex still outperforms traditional computers for a number of tasks. Emulating such capability is thus a major step in building computing systems that can compete with at least some processing characteristics of the brain. While the cortical structure of certain regions of the brain, such as the visual cortex, has been investigated for a long time, quantitative models, consistent with physiological data, and capable of accounting for complex visual tasks, were proposed only recently [119, 120]. Section 4.3.1.3 highlights the ability of the proposed computational model to emulate the notion of automatic abstractions.

2.4 The Visual Cortex

The neocortex is divided into a number of sub-regions e.g. the visual cortex, the auditory cortex, the somatosensory cortex, etc. Each of these subregions of the neocortex processes information provided by a different sensory modality. For example, the visual cortex deals with visual data, the auditory cortex processes sounds experienced by the ear, and so on. This dissertation focuses on the visual cortex for discussions and results mainly because

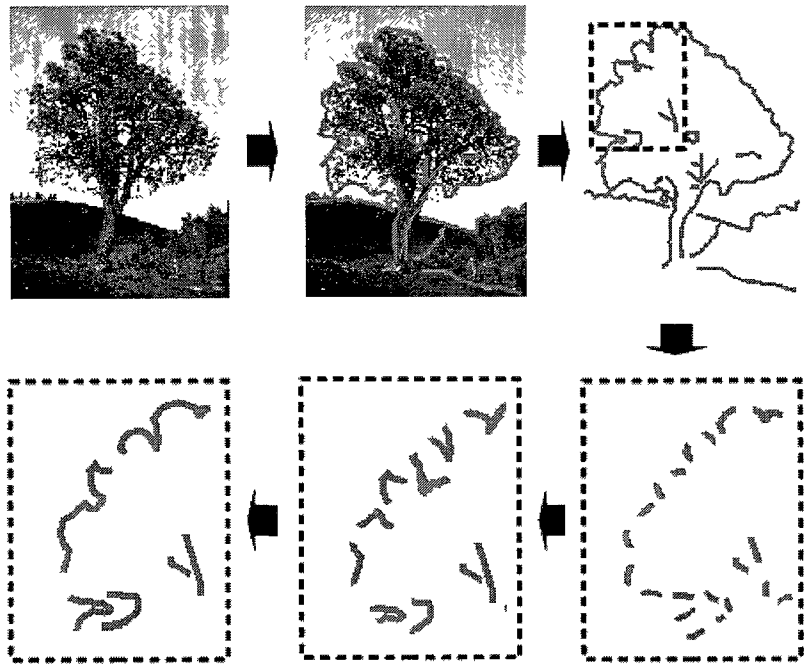

Figure 2.4: Increasingly complex visual abstractions (segments, angles and long segments, complex shapes, etc.) [43].

various aspects of the visual cortex have been studied in far more detail as compared to other cortical regions.

2.4.1 Input Pathways

In mammals, visual scenes are projected onto the retina. The activations of the retinal cells in response to the visual scene are transfered via the optical nerve to the Lateral Geniculate Nucleus (LGN) cells [69]. LGN is the first cortical stage for visual processing. The LGN cells are contrast sensitive. They react strongly to an illuminated point surrounded by darkness (known as on-off cells) or conversely to a dark point surrounded by light (off-on) cells. These

cells are spatially distributed in such a way that on-off and off-on cells are intertwined [109] and receive inputs from neighboring retinal cells. Finally, the activations of the LGN cells become the input to the primary visual cortex.

While the visual inputs are being transferred from the retina to the LGN cells, they undergo a transform commonly known as the log-polar transform [117, 118]. This transform converts the visual image from a Cartesian coordinate system to a log-polar one. Studies show that log-polar transform plays an integral role towards achieving rotation, scale, and translation invariance [136] since variations in scale and rotation in a Cartesian system are transformed into translations in a log-polar system. Section 4.1 describes the techniques that the computational model proposed in this dissertation uses to process visual patterns using mathematical operations to approximate the biological behaviors as implemented within the optical pathways.

2.4.2 Hierarchical Organization

As described in Section 2.2, the entire neocortex is organized in the form of a hierarchy. This hierarchical organization of the neocortex cortex has been studied in detail in the visual cortex which is subdivided into different hierarchical regions. These regions are broadly classified into the primary visual cortex (V1), the secondary visual cortex (V2/V4), the inferior temporal cortex (IT), and middle temporal cortex (MT).

The LGN cells send their feedforward projections to V1. V1 then project into V2/V4 which further project into IT and MT [11, 53, 82, 103]. Neurons in V1 specialize in identifying

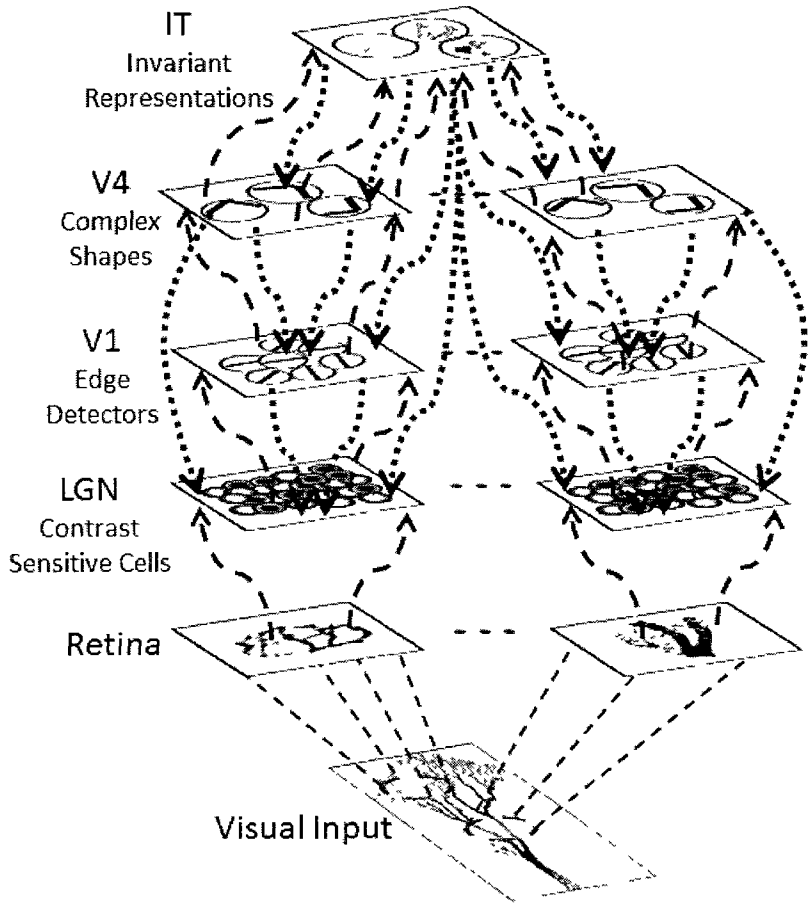

Figure 2.5: A simplified representation of hierarchical organization of the visual cortex.

edges of different orientation while neurons in V2/V4 identify complex combinations of these edges. Neurons in IT detect further complex visual patterns and also develops invariant representations for different visual inputs. Finally, neurons in MT play a major role in the perception of motion. Figure 2.5 presents a simplified representation of the hierarchical organization of the visual cortex. Section 4.3 describes how a hierarchical network that realizes complex visual processing tasks can be constructed using various functional blocks

provided by the proposed computational model. This section also describes the role of feedforward, feedback, and lateral connections within such a hierarchical network.

2.4.3 Invariant Representations

A very powerful attribute of the visual cortex is its ability to generate invariant representations. The mammalian visual cortex is able to recognize an object that it has learned before no matter what size or orientation it appears in. Even in the presence of significant distortions and noise, the visual cortex is able to recognize a previously learned object. Invariance of the visual cortex to variations in scale and rotation have been the focus of research for a while now [118, 106] but the exact mechanisms that the visual cortex employs to achieve this invariance are still not completely understood. Sections 4.1, 4.3.1.2, and 4.3.2.2 describe various biologically inspired techniques that the proposed computational model relies on to develop invariant representations for various complex patterns exposed to the network.

2.4.4 Feature Maps

A very interesting property of various subregions of the visual cortex is the generation of feature maps. Neurons within the visual cortex are organized into multiple feature maps according to parameters including receptive field, ocular dominance, orientation selectivity, and spatial frequency. Each of these complex maps is spatially interrelated, but it is unclear what guides the development of these relationships [30]. These feature maps have been studied in detail in the visual area V1 where their existence was first discovered by Hubel

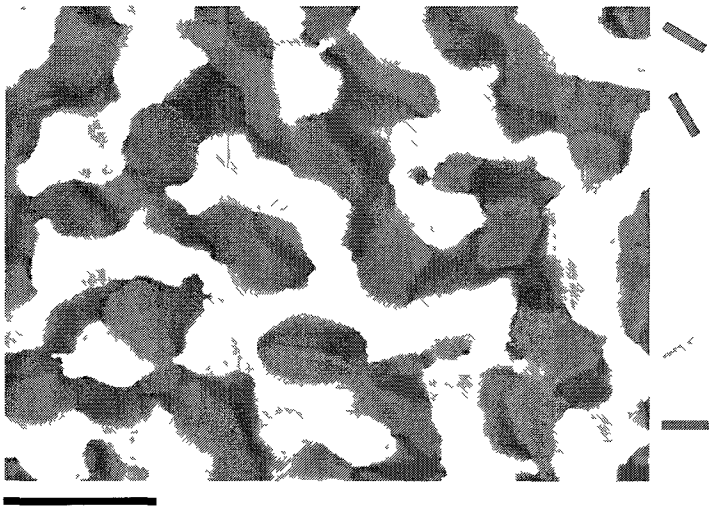

Figure 2.6: Layout of orientation preference observed in area V1 of macaque monkey [12].

and Wiesel in 1962 [59] in the form of edge specific orientation maps. In primates, including humans, feature maps are present in many, and possibly all, visual areas of the cerebral cortex beyond V1 [133]. Figure 2.6 shows the orientation map preference observed in the primary visual cortex of a macaque monkey. Section 4.3.1.1 describes how the proposed computational model develops topographical feature maps using spatially localized spontaneous activations.

2.4.5 Spontaneous Activations

Although various aspects of the neocortex have been studied in detail, there exist a number of *not so well* understood properties of the neocortex which may also contribute to its unique abilities. One such behavior is the spontaneous activation of the cortical neurons [33]; that is, neurons in the neocortex may exhibit activity even in the absence of a driving input stimulus. These spontaneous activations have been studied in detail in the retinal ganglion cells [18]

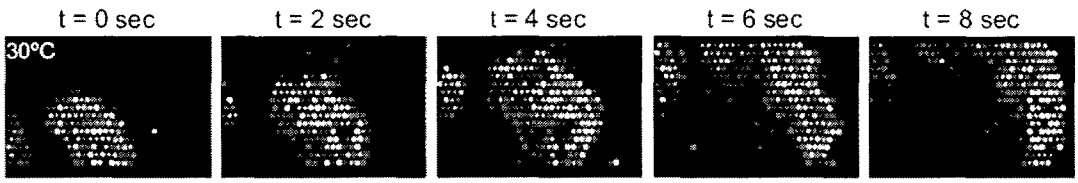

Figure 2.7: Spontaneous retinal activity in early post-natal retinas [12].

where they are shown to be both spatially localized and temporally correlated. Recently, various studies have shown that such activations are essential for the proper development of orientation maps within the primary visual cortex [4, 18]. Even though the exact role of these spontaneous activations is not understood, studies suggest that they are essential during the early development stages of the mammalian neocortex [18]. Figure 2.7 shows the spontaneous activations recorded in the early post natal mice retinas. In the figure, the spontaneous activations are triggered at the bottom-left and they propagate towards the top-right in a spatially localized and temporally correlated manner. Within the proposed computational model, spontaneous activations play two important roles. First, they allow the model to develop hierarchical feature maps (see Section 4.3.1.1). Second, they provide the model with an ability to robustly recover from permanent hardware failures (see Section 5.2).

2.4.6 Role of Feedback Connections

Along with the feedforward processing paths, the feedback paths play an important role in making the neocortex a very robust and powerful processing system. Lower hierarchical levels send feedforward projections to the higher cortical levels and the higher level in turn project into the lower levels. Even though the exact role of the feedback paths in the overall cortical

Figure 2.8: One of the possible roles of feedback paths is pattern completion. Richard Gregory's Dalmatian image [39].

processing and learning is not well understood, their importance has been highlighted in a number of studies including [16, 31, 62, 61, 124].

Feedback projections are an integral part of the mammalian visual system. Within the visual cortex, IT sends feedback projections to V2/V4, to V1 and also to the LGN cells. Similarly, V2/V4 send projections to V1 and LGN cells and V1 cells send feedback only to LGN. Recent studies show that these feedback paths from the higher regions to the lower ones modulate the lower level responses based on context and predictions [16, 61], improve discriminating objects of interest from background [62], and help in robust recognition of noisy visual inputs [131]. These are only a few examples of the powerful role of the feedback paths within the overall functionality of the visual cortex. Many other roles, especially the

role of feedback paths in learning, are yet to be completely understood.

Figure 2.8 contains Richard Gregory's Dalmatian image. This image demonstrates the power of feedback connections. It essentially contains black spots on a white background but after sometime, the observer is able to see a Dalmatian in the middle of the image. Using the feedback connections, the visual cortex is able to construct the missing features and a Dalmatian is suddenly recognized. Section 4.3.2 provides a detailed discussion on the role of feedback paths during the learning process in the context of the proposed computational model.

2.5 Summary

This chapter describes and explains various neocortical properties and operations that have inspired the construction and working of the biologically inspired computational model proposed in the dissertation. The main objective of this chapter is to introduce various biological concepts to the reader and briefly describe the current level of understanding of various neocortical regions especially the visual cortex.

3 RELATED WORK

This chapter summarizes research work related to this dissertations. Section 3.1 summarizes research in artificial intelligence and neural networks that pertains to the proposed learning model. Section 3.2 briefly describes various hardware based learning models and certain tools proposed by the computer architecture community that can benefit the design of a powerful and robust learning model.

3.1 Artificial Intelligence and Neural Networks

This sections describes various learning models proposed for intelligent processing of input data. These models have mostly been applied to classification tasks. The main goal of this section is to discuss these models in terms of their applications, limitations, and biological fidelity.

3.1.1 Abstract Models with Limited Biological Fidelity

This class of models includes artificial neural networks [99], Bayesian networks [49], deep belief networks [52], competitive learning [113, 112], self-organizing maps [74], convolutional neural networks [80], etc. These models have been utilized for different type of classification tasks including object recognition [79, 80], disease diagnostics [78, 102, 125], forecasting market business trends and share prices [3, 134], industrial process control [36, 73, 86], risk management [66, 70], and target marketing [71]. Even though these models were inspired

by biology, over time they have moved significantly away from their biological inspiration in terms of implementation and functionality.

3.1.1.1 Artificial Neural Networks

An artificial neural network (ANN) is an information processing paradigm that is inspired by the way mammalian brain processes information. A neural network consists of an interconnected group of artificial neurons also known as *perceptrons* which contain a set of weights that determine the behavior of the perceptron to its inputs. In most cases an ANN is an adaptive system that changes its behavior based on external or internal information that flows through the network during the learning phase. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data [79, 80].

A artificial neural network consists of multiple levels and each of these levels is connected to the previous one via feedforward processing paths. During every training iteration, a perceptron evaluates the correlation between its inputs and its set of weights. It this correlations is greater than a threshold, the perceptron sets its output to be high using a sigmoid activation function. Furthermore, perceptrons within each of the neural network's levels modify their set of weights according to a predefined learning rule. The biologically inspired learning model discussed in this dissertation borrows the notion of sigmoid activation function from the perceptron model and further extends it to evaluate the output of a node (refer to Section 4.2.2).

Even though ANNs have been employed to perform a number of pattern recognition and classification tasks, they have always been challenged in terms of their generalizations and correct identification of possible causal relationships [67]. Various parameters determining the behavior of an ANN need to be fine-tuned for each specific processing task [67] and to achieve best network performance. These parameters include the learning rate of the network, the number of hidden layers, the number of neurons within each hidden layer, random seeds to initialize the weights, etc. Another shortcoming of ANNs is their dependence on only the feedforward information during both the training and testing phases. As discussed in Chapter 2, within the neocortex, along with the feedforward processing paths, the feedback paths play a very important role in realizing complex tasks. Absence of these feedback processing paths leaves the ANNs at a major disadvantage. *Back propagation neural networks* [79], a commonly known type of ANNs, include feedback paths from the upper levels to lower levels. These feedback paths from higher levels to the lower ones essentially propagate the error between the output of the network and the desired response. Even though these networks contain a notion of feedback, this type of feedback has no biological basis. Finally, the convergence of an ANN towards an optimal solution is not guaranteed. Based on the random initial weight values, the network may converge to any local minimum on the error surface if the stochastic gradient descent traverses a surface which is not convex [37]. Section 4.2.1 describes that the computational model proposed in this dissertation does not rely on random initialization of weights. Rather the proposed model uses spontaneous activations for triggering the initial learning process and does not suffer from any convergence

issues. Section 4.3.2 discusses various biological inspired mechanisms to implement feedback processing paths within the computational model proposed in this dissertation and the powerful role these feedback paths play during the learning process.

3.1.1.2 Deep Belief Networks

Deep belief networks (DBN) are probabilistic generative models that are composed of multiple layers of latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors. The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector. Deep belief nets are learned one layer at a time. During each training epoch, the output values of the latent variables in one layer are treated as the training inputs for the next layer. This efficient, greedy learning can be followed by, or combined with, other learning procedures that fine-tune all of the weights to improve the generative or discriminative performance of the whole network. Among many other applications, DBNs have also been used for generating and recognizing images [52, 107], video sequences [127] and motion-capture data [130].

Even though DBN are considered to be quite powerful compared to other neural networks, they demonstrate moderate performance in terms of their recognition accuracy on complex data sets [77]. Even the most sophisticated and highly fine tuned DBN trained on 1.6 million images [77] only achieves the best recognition rate of 78% on simple natural images.

This clearly demonstrates that these DBNs lack the ability to develop generalized internal representations for effective pattern recognition. Apart from their hierarchical organization, DBNs do not model other interesting structural or functional aspects of the neocortex like the use of spontaneous activations for initial learning and fault tolerance, independent feature identification, generation of feature maps, and the role of feedback processing paths in learning and development of invariant representations. Sections 4.2.1, 4.2.3, 4.3.1.1, and 4.3.2 describe how such powerful neocortical aspects are implemented within the computational model proposed in this dissertation and how such aspects make the proposed model more attractive than other contemporary approaches.

3.1.1.3 Competitive Learning

Competitive learning is a powerful learning paradigms in which nodes within a spatial locality compete for the right to respond to a subset of input data [112]. Competitive learning networks usually perform feedforward processing on the input data and implement a winner-take-all behavior. During each epoch, all the nodes within a spatial locality evaluate their output in response to an input and the node with the highest response is allowed to fire while the others a inhibited. Furthermore the winner node updates its weights using a predefined weight update rule, similar to gradient descent. The learning model described in this dissertation further extends the idea of competitive learning to implement the functionality of minicolumns and hypercolumns described in Section 2.1.5. The competitive learning rule and further extensions to it are discussed in Sections 4.2.1 and 4.2.3.

Even though competitive learning provides powerful unsupervised learning capabilities, it suffers from a number of shortcomings as well. First, since most competitive learning networks contain only feedforward connections, they suffer from the same problems described in Section 3.1.1.1 in the context of multi-layer perceptron models. Furthermore, the inherent competitive nature of the nodes within a network makes them more prone to getting stuck in a local minimum. As a result, convergence of competitive learning networks to optimal solution is not guaranteed. Finally, learning generalized and invariant representations using solely a competitive learning rule is difficult as two variations of the same pattern may not be learned by the same node in a competitive learning paradigm. Sections 4.2.1, 4.2.2, and 4.3.2.2 describe various biologically inspired properties that help resolve various issues concerning the competitive learning paradigm.

3.1.1.4 Self Organizing Maps

A self-organizing map (SOM) [74] is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional, discretized representation of the input space of the training samples. Self-organizing maps are different from other artificial neural networks in the sense that they use a neighborhood function to preserve the topological properties of the input space. A self-organizing map consists of components called nodes or neurons. Associated with each node is a weight vector of the same dimension as the input data vectors and a position in the map space. The usual arrangement of nodes is a regular spacing in a hexagonal or rectangular grid. The self-organizing map describes a mapping

from a higher dimensional input space to a lower dimensional map space. The procedure for placing a vector from data space onto the map is to first find the node with the closest weight vector to the vector taken from data space. Once the closest node is located it is assigned the values from the vector taken from the data space.

The goal of learning in the self-organizing map is to cause different parts of the network to respond similarly to certain input patterns. This is partly motivated by how visual, auditory, or other sensory information is handled in separate parts of the cerebral cortex in the human brain [60]. The weights of the neurons are initialized either to small random values or sampled evenly from the subspace spanned by the two largest principal component eigenvectors. With the latter alternative, learning is much faster because the initial weights already give good approximation of SOM weights [75]. The network must be fed a large number of example vectors that represent, as close as possible, the kinds of vectors expected during mapping. The examples are usually administered several times as iterations. The training utilizes competitive learning [112]. When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron with weight vector most similar to the input is called the *best matching unit.* The weights of the best matching unit and neurons close to it in the SOM lattice are adjusted towards the input vector.

Despite the wide use of SOMs for data clustering and classification, they suffer from three major problems. First, the SOM uses a fixed network architecture in terms of number and arrangement of neurons which has to be defined prior to training. Obviously, in case of largely unknown input data characteristics it remains far from trivial to determine the network

architecture that allows for satisfying results. Second, hierarchical relations between the input data are not mirrored in a straight-forward manner [27]. Such relations are rather shown in the same representation space and are thus hard to identify. Hierarchical relations, however, may be observed in a wide spectrum of application domains. Thus, their proper identification remains a highly important data mining task that cannot be addressed conveniently within the framework of the SOM.

In terms of their biological fidelity, apart from creating feature maps, SOMs do not model any other properties inspired by the mammalian brain. Furthermore, the Euclidean distance based methodology that the SOMs use for creating the feature maps has no biological basis. Traditional SOMs organize themselves simply based on the similarity of input features. A neuron which has the highest response will not only improve its correlation with an input, but also the correlation of its neighbors to encourage a topological organization of similar features. However, such SOMs do not account for the temporal aspect of input patterns and how they can help shape the feature maps.

Sections 4.2.1, 4.3.1.1, 4.3.1.2, and 4.3.1.3 describe how the computational model proposed in this dissertation utilizes various biologically inspired mechanisms including spatially localized and temporally correlated spontaneous activations, automatic abstraction, and Gaussian like connectivity to develop hierarchical feature maps. Since these biological mechanisms do not rely on evaluation of any Euclidean distance like metric, they are inherently computationally efficient as compared to the traditional SOM approaches for developing feature maps.

3.1.1.5 Convolutional Neural Network

Convolutional Neural Networks (CNN) [80] are a kind of multi-layer neural networks. Like other ANNs, they are trained using a back-propagation algorithm. CNNs differ from other ANN approaches in terms of their architecture. First, CNNs exploit spatially local correlation by enforcing a local connectivity between neurons in adjacent layers. The input units in the m^{th} layer are connected to a local subset of units in the $(m-1)^{th}$ layer. Second, at each level, there exist a number of filter masks with shared receptive fields that are convolved with the input. The outputs of these filter masks undergo a max operation where the strongest output of all the filters with the same receptive field is transferred to the next level. Finally, in CNNs, each of the filters is replicated across the entire visual field. These replicated filters form a feature map, which share the same parameterization, i.e. the same weight vector and the same bias. Conceptually, a feature map is obtained by convolving the input image with a linear filter, adding a bias term and then applying a non-linear function.

CNNs are especially tricky to train, as they add even more hyper-parameters than a standard ANN. While the usual rules for learning rates and regularization constants still apply, the following should be kept in mind when optimizing CNNs.

- Number of filters per layer needs to be specified before the training period.
- Since different filter shapes have different effect on the overall performance of the network, filter shapes need to be fine-tuned to improve the network's performance.
- Number of filters per max operation has to be specified before the training phase.

With all these additional parameters, fine-tuning a convolution network to achieve optimal performance becomes quite cumbersome.

Similar to other artificial neural networks, apart from hierarchical arrangement, CNNs also do not incorporate other biological operations and abstractions including the use of spontaneous activations for initial learning and fault tolerance, independent feature identification, generation of feature maps, and the role of feedback processing paths in learning and development of invariant representations. Sections 4.2.1, 4.2.3, 4.3.1.1, and 4.3.2 describe how such powerful neocortical aspects are implemented within the computational model proposed in this dissertation and how such aspects make the proposed model more attractive than other contemporary approaches. .

3.1.2 Abstract Models with Moderate Biological Plausibility

Unlike the neural models discussed in Section 3.1.1, this class of models tends to implement several properties that are inspired by certain neocortical properties including hierarchy, spatial and temporal pooling, vigilance, competitive learning, etc. Among others, Hierarchical Temporal Memories (HTM) [34], Adaptive Resonance Theory (ART) [41], and HMAX [108, 119] are the most famous models within this class. Despite the fact that these models tend to be moderately close to the biology, they ignore a number of very powerful biological properties like plasticity, spontaneous activity, feedback processing paths, etc.

3.1.2.1 Hierarchical Temporal Memories

Hierarchical temporal memory (HTM) is a machine learning algorithm that models some of the structural and algorithmic properties of the neocortex. The functionality of HTM model is based on the memory-prediction theory of brain function described by Jeff Hawkins in his book *On Intelligence* [47]. A typical HTM network is a tree-shaped hierarchy of levels that are composed of smaller elements called nodes. Higher levels in the hierarchy have fewer nodes, so the resolution in space is lost. At the same time, higher levels reuse patterns learned at the lower levels by combining them to memorize more complex patterns. Input is exposed to the levels at the bottom of the hierarchy and from there activation propagate up the hierarchy until the top level where complex objects are recognized. The top level node is usually a classifier trained using back-propagation and it learns general categories (concepts) which are determined by smaller concepts in the lower levels. HTM networks have mainly been used for pattern recognition and classification tasks and have demonstrated comparable results as other neural network models [34].

A HTM network can simply be classified as a complex hierarchical Bayesian network. The main difference between HTM and a traditional Bayesian network is the spatial and temporal pooling aspect of HTMs. Each node within the HTM network has the ability to pool feature in terms of their spatial similarities and temporal co-occurrences. Using the spatial pooling aspect, an HTM node can generate invariant representations for variations of the same feature while the temporal pooling property allows the HTM network to predict the outcome using the global context. [34]

Even though HTM implements a number of neocortical properties like hierarchy, spatial pooling to generate partial invariant representations, and context based prediction, they do not model some other very powerful aspects of the neocortex like utilizing feedback paths to generate feature maps and hierarchically organized invariant representations, spontaneous activity, inherent fault tolerance, and connectivity rules among various hierarchical levels. These are some of the major shortcomings of HTM and the reason why HTM networks are not very widely applied for pattern recognition tasks. Sections 4.2.1, 4.2.3, 4.3.1.1, and 4.3.2 describe how such powerful neocortical aspects are implemented within the computational model proposed in this dissertation and how such aspects make the proposed model more attractive than other contemporary approaches.

3.1.2.2 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) [41] is a theory inspired by various aspects utilized by the brain to process information. It describes a number of neural network models which use supervised and unsupervised learning methods, and address problems such as pattern recognition and prediction. The primary intuition behind the ART model is that object identification and recognition generally occur as a result of the interactions between the feedforward sensory information and context dependent feedback predictions. The model postulates that top-level predictions take the form of a memory template that is then compared with the various features of an object as detected and communicated by the sensory modalities. This comparison is then used to evaluate the feedforward information and to

classify the input patterns. As long as this difference between feedforward sensation and feedback prediction does not exceed a set threshold called the *vigilance* of the network, the sensed object is considered a member of the expected class.

The basic ART system is an unsupervised learning model. It typically consists of a comparison field and a recognition field composed of neurons, a vigilance parameter, and a reset module. The vigilance parameter has considerable influence on the system: higher vigilance produces highly detailed memories (many, fine-grained categories), while lower vigilance results in more general memories (fewer, more-general categories). The comparison field takes an input vector (a one-dimensional array of values) and transfers it to its best match in the recognition field. Its best match is the single neuron whose set of weights most closely matches the input vector. Each recognition field neuron outputs a negative signal (proportional to that neuron's quality of match to the input vector) to each of the other recognition field neurons and inhibits their output accordingly. In this way the recognition field exhibits lateral inhibition, allowing each neuron in it to represent a category to which input vectors are classified. After the input vector is classified, the reset module compares the strength of the recognition match to the vigilance parameter. If the vigilance threshold is met, training commences. Otherwise, if the match level does not meet the vigilance parameter, the firing recognition neuron is inhibited. This allows some other neuron with the same recognition field to learn the feedforward pattern. In the search procedure, recognition neurons are disabled one by one by the reset function until the vigilance parameter is satisfied by a recognition match.

To date, a number of variations if ART have been proposed. These include,

- ART-1: Simplest variety of ART networks, accepting only binary inputs [21].
- ART-2: Extends network capabilities to support continuous inputs [19].
- Fuzzy ART: Implements fuzzy logic into ART pattern recognition algorithm, thus improving network's ability to generate invariant representation [20].
- ARTMAP: Combines two slightly modified ART-1 or ART-2 units into a supervised learning structure with a vigilance parameter [22].

ART models have been utilized for a number of pattern recognition and classification tasks. One of the major weaknesses of ART is that the performance of ART depends on the order in which the training data are processed. This effect can be reduced to some extent by using a slower learning rate, but is present regardless of the size of the input data set. Furthermore, although ART models a number of interesting neocortical properties, like HTM, it also disregards a number of powerful neocortical properties as well. These include the role of feedback in learning and generation of invariant representations, spontaneous activations, inherent fault-tolerance, and cortical rules for evolution of connections among various layers. Sections 4.2.1, 4.2.3, 4.3.1.1, and 4.3.2 describe how such powerful neocortical aspects are implemented within the computational model proposed in this dissertation and how such aspects make the proposed model more attractive than other contemporary approaches.

3.1.2.3 HMAX

The *Hierarchical Model and X* or HMAX model [108] implements the basic known facts about the feedforward information processing pathways in the ventral visual stream [84], a hierarchy of brain areas thought to mediate object recognition in the cortex. This model reflects the general organization of visual cortex in a series of layers from V1 to IT. A traditional HMAX network consists of 5 levels [108] and in terms of generating invariant representations, the model implements two types of cells: simple (S) cells and complex (C) cells. Input images are filtered using an array of 2D Gabor-like filters which model the first layer of simple cells (S1). Centered at each pixel location there are around 50 filters sensitive to bars of different orientations and sizes, thus roughly resembling properties of simple cells in striate cortex [84]. The output of the S1 cells is transferred to the next level which implements the first layer of complex cells (C1). The C1 cells pool the response of multiple S1 cells and apply a winner-take-all like *max* operation on the activations of the S1 cells within their receptive field. The output of the C1 cells is then fed into another simple cell layer (S2) which connects to a layer of complex cells (C2). Finally, the C2 cells connect to the view-tuned cells [84] (VTUs) which classify different objects. Figure 3.1 shows a standard 5 level HMAX network.

One of the major shortcomings of HMAX is that it adds a number of hyper-parameters that need to be fine tuned to optimize the performance of the network. This fine-tuning mechanism is highly dependent on the type of patterns in both the training and testing set. Furthermore, even though different layers of HMAX are inspired by the functionality of the ventral visual stream, they miss a number of cortical features that play a very important role

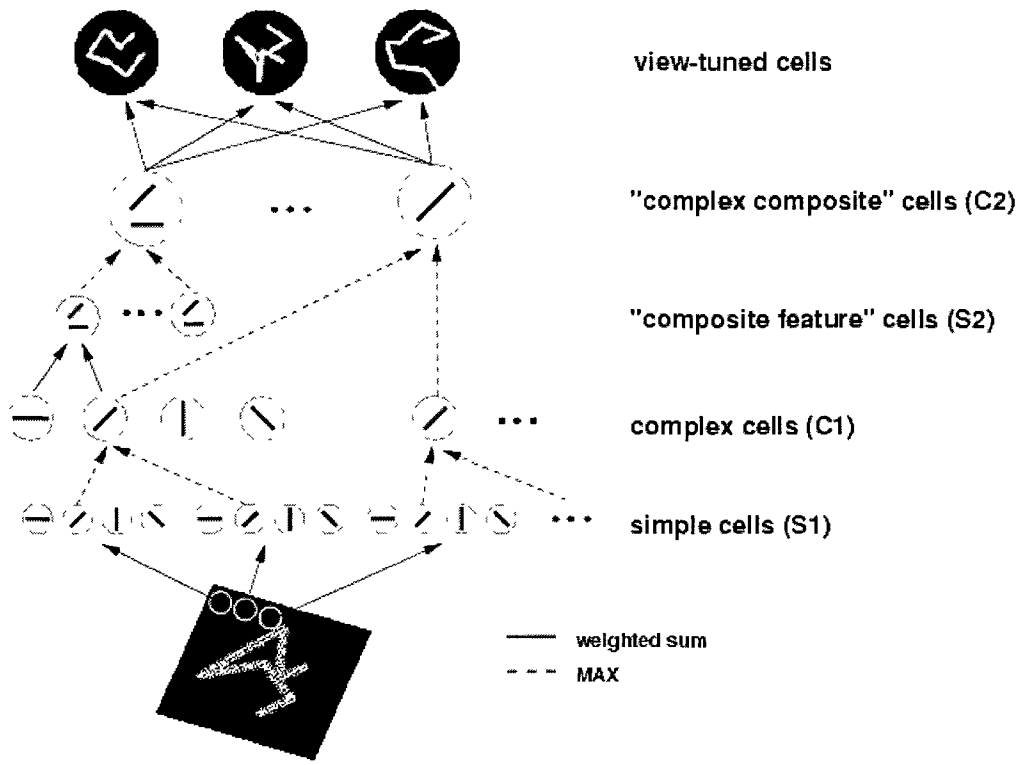

Figure 3.1: Schematic of the Standard HMAX network [108].

within the cortical processing streams. First, HMAX relies only on the feedforward processing paths of learning and recognition. Feedback paths are omitted in this model. Second, the properties of different simple and complex cells are hard-coded in the before the learning process takes place. Thus, they are not learned over time. Finally, the role of spontaneous activations in learning and inherent fault-tolerance is missing. Sections 4.2.1, 4.3.2, and 5.2 describe how such powerful neocortical aspects are implemented within the computational model proposed in this dissertation and how such aspects make the proposed model more attractive than other contemporary approaches.

3.1.3 Detailed Models with High Biologically Plausibility

This section discusses a few low level cortical models which implement various aspects of the structure of neurons and their functional behavior. This section also discusses the application of such models and their limitations. Two very famous models that fall under this category are the Hodgkin-Huxley model [54, 55, 56] and the Izhikevich spiking neuron model [65].

3.1.3.1 Hodgkin-Huxley Model

The Hodgkin-Huxley model is a mathematical model that describes how action potentials in neurons are initiated and propagated. It is a set of nonlinear ordinary differential equations that approximates the electrical characteristics of excitable cells such as neurons. Alan Lloyd Hodgkin and Andrew Huxley described the model in 1952 to explain the ionic mechanisms underlying the initiation and propagation of action potentials in the squid giant axon [54, 55, 56]. They received the 1963 Nobel Prize in Physiology or Medicine for this work.

The Hodgkin-Huxley model is quite useful to study and understand interactions between neurons as long as the number of neurons in the network is small. In case of larger networks, this model suffers two main problems. First, implementing large hierarchical networks that may realize complex tasks becomes quite cumbersome when using this model. Second, since various non-linear differential equations need to be evaluate every cycle, the overall computation cost of large Hodgkin-Huxley networks because very high. Finally, the Hodgkin-Huxley model does not define any rules for learning and training. These shortcomings make it quite hard and cumbersome to realize complex tasks using this model.

3.1.3.2 Izhikevich Spiking Neuron Model

The Izhikevich model reproduces spiking and bursting behavior of known types of cortical neurons. This model combines the biological plausibility of Hodgkin-Huxley dynamics and the computational efficiency of traditional artificial neurons. Using this model, one can simulate tens of thousands of spiking cortical neurons in real time (1 ms resolution). The Izhikevich model simplifies various non-linear complex differential equations into a system of ordinary differential equations. This significantly improves the computational cost required to evaluate the behavior of neurons during every execution cycle.

The Izhikevich model provides biologically plausible and computationally efficient tools to study detailed interactions among various neurons. However, like the Hogdkin-Huxley model, the Izhikevich model suffers from two main shortcomings. First, constructing large networks using the Izhikevich model to realize complex tasks gets quite cumbersome. Second, the Izhikevich model also does not provide any rules for learning or training. All the connections between different neurons need to be hardwired beforehand. As a result, there is no straightforward way to realize complex tasks that require online learning and training.

3.1.4 Biological vs. Artificial Neural Networks

In terms of developing neural models, the machine learning and the neurobiology communities pursue two different goals. Researchers in the machine learning community have an extreme interest in developing the most efficient classification algorithm. On the other hand, researchers in the neurobiology seek to build models inspired by various structural and functional aspects

of biological neural networks with an eventual goal of ultimately emulating the whole range of biological neural networks' capabilities, with classification being only one such capability. The goals of this dissertation are more in sync with the neurobiology community: emulate biological neural networks with the goal of reproducing a large range of their capabilities, along with the additional goal of achieving computational efficiency.

Additionally, the biological and the artificial neural networks differ from each other on a number of other dimensions as well. First, both of these networks typically rely on different learning strategies. ANNs often rely on back-propagation for learning classification tasks: the correct answer is known and is fed back through the network by adjusting the synapse weights based on the network error; this form of learning is called supervised learning [48]. Biological neural networks rely on what the machine-learning community calls unsupervised learning: the correct answer is not known, but the network learns through repeated exposure to the input via local learning rules, i.e., Hebbian learning [15]. There may also exist indirect supervisory feedback, e.g. reward/punishment (reinforcement learning) [128]. However, the way biological networks implement feedback is radically different from the back-propagation algorithm used by traditional ANNs. Second, both the biological and artificial neural networks differ in terms of the attention given to the network structure. ANNs often rely on full connectivity, or random structures [48]. Recent progress in neurobiology [42, 10] show that the network structure and the nature and arrangement of connections are complex and play a major role in the abstraction capabilities of the network.

Third, both single and multi-layer ANN models are intolerant of permanent hardware

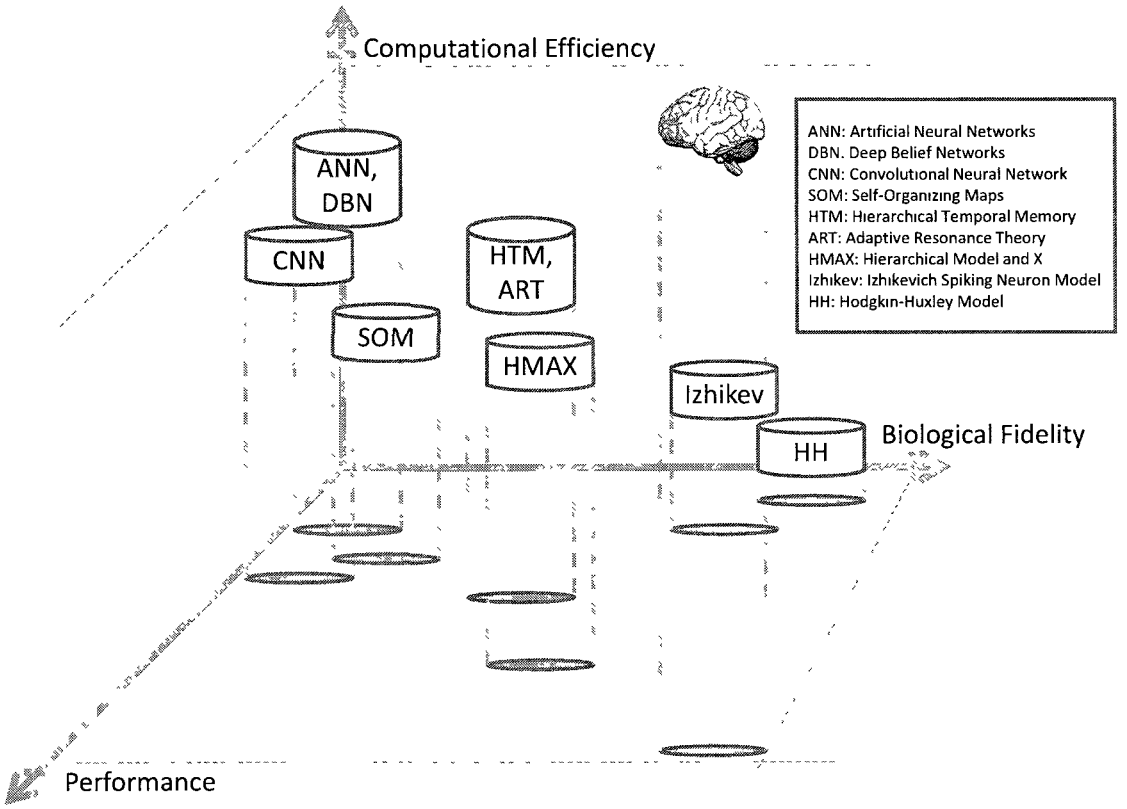

Figure 3.2: A comparison between various neural models in terms of their biological fidelity, computational efficiency, and performance.

defects. Emmerson and Damper [28] show that even a slight amount of faulty behavior (either in the perceptron or the connections) can significantly deteriorate the recognition rate of these models.

At the other end of the spectrum, there are highly detailed biologically plausible models which realize a number of low level cortical properties but these models suffer from their computational costs and difficulty in constructing large networks to realize complex tasks. Figure 3.2 compares all the models discussed in this section in terms of their biological fidelity,

computational efficiency, and performance. Performance in the figure is defined as the ability of the network to efficiently realize complex tasks. Ideally, we need a model that has high biological fidelity, is computationally efficient, and performs well in realizing complex tasks.

3.2 Computer Architecture

The purpose of this section is twofold. First, it provides a brief description of various biologically inspired hardware based learning models and discusses various limitations of such models. Second, it describes some of the tools developed by the computer architecture community that can help in constructing powerful learning models.

3.2.1 Learning Models

This section presents three hardware based learning models. These models include the Blue Brain project [68, 87], the FACETS project [29], and the SyNAPSE project [92]. The main focus of this section is on the implementation details of these hardware based systems and their limitations.

3.2.1.1 Blue Brain

The Blue Brain Project is an attempt to create a synthetic brain by reverse-engineering the mammalian brain down to the molecular level. The main goal of this project is to reconstruct the brain piece by piece and build a virtual brain in a supercomputer [87]. This virtual tool will give neuroscientists a better understanding of the brain and various neurological diseases.

As a first step, the project succeeded in simulating a rat cortical column [8]. A rat's brain has about 100,000 columns. Each of these columns consists of around 10,000 neurons. A longer term goal of this project is to build a detailed, functional simulation of the physiological processes in the human brain.

Even though the Blue Brain project strongly adheres to the structural properties of the brain, it suffers from two main problems. First, simulating even a small network of neurons requires massive computations and a huge amount of time. Thus, this model cannot be used for any applications with real time execution requirements. Second, as is the case with other biologically realistic models, constructing large networks to realize complex tasks gets quite cumbersome and extremely time consuming.

3.2.1.2 FACETS

The Fast Analog Computing with Emergent Transient States (FACETS) project aims to address the unsolved question of how the brain computes using analog circuits and devices. Another objective of this project is to create a microchip hardware equaling approximately 200,000 neurons with 50 million synapses on a single silicon wafer. One of the major issues that this project is facing is fabricating analog circuits on a wafer. This process is quite tricky and is prone to a number of unexpected behaviors and reliability issues [35].

3.2.1.3 SyNAPSE

Researchers at IBM have been working on a cognitive computing project called Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE). By reproducing the

structure and architecture of the brain, the SyNAPSE project models computing systems using digital circuits that emulate the brain's computing efficiency, size and power usage without being programmed. So far, the research team at IBM has been able to fabricate a chip with 256 neuron-like elements and 65536 synapses. This chip is the very first step towards making a powerful hardware based neuromorphic system. The SyNAPSE project is in a very early stages of development and its full potential is yet to be known.

3.2.1.4 Silicon-based Implementation

This dissertation does not present a specific silicon-based implementation of the proposed model, because the recent progress in neurobiology, which motivates this work, relates more to the structure and connections between neurons, than to the behavior of neurons and synapses themselves. As a result, the neural arrangements presented here can be readily implemented in silicon by leveraging the large body of work on hardware implementation of artificial neural networks, for example [57, 116].

3.2.2 Abstraction Layers

The externally-observable operation of many systems can be reproduced with a behavioral model that operates at an appropriate level. As architects, we are familiar with many examples of such behavioral models. Increasingly high-level models often aggregate the behavior of lower-level elements in time and/or in space. For example, a gate-level model of a digital circuit removes details related to individual transistors and does not model transient

switching behavior, reporting only steady-state logic values. Typically, the precision of these models varies inversely with their computational demands, and an appropriate model must be chosen to satisfy both reasonable time to completion and sufficient precision.

Sandberg and Bostrom provide a thorough introduction to neural modeling, and identify eleven different levels of model, ranging from ANNs all the way to molecular dynamics and even quantum-level simulation [115]. They argue that brains can be emulated without higher-level algorithmic understanding: as long as biological details are measured carefully and replicated faithfully, a feline brain, or even a human brain, will *boot up* and work as expected. A recent experiment showed that a large-scale supercomputer (IBM Blue Gene) possesses the computational throughput for modeling a rat cortex using this approach within an order of magnitude of real time (9x slowdown) [5]. Also, the DARPA Synapse [24] program has set a goal of scaling emulation up to a feline cortex.

In contrast, as architects, we want to build cortically-inspired computing systems which emulate selected functional subsets of the human cortex. This requires detailed high-level algorithmic understanding of the cortical properties, rather than simply precisely reconstructing the biological baseline. It also requires high computational efficiency, which suggests developing high-level behavior models emulating cortical functionality. The pitfall is development of high-level but unfaithful models which fail to emulate the target functions because they do not capture the key aspects of their biological implementation. For that reason, such high-level models must be validated against lower-level, less computationally efficient, but biologically plausible models.

Sections 5.1.2 and 5.1.3 describe mechanisms that use the notion of abstraction layers to achieve computational efficiency within the learning model proposed in this dissertation.

3.2.3 Instruction Set Architecture

An instruction set architecture (ISA) is the part of the processor architecture that specifies the native data types, instructions, registers, addressing modes, and memory architecture. An ISA includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor. Instruction set architecture is distinguished from the microarchitecture, which is the set of processor design techniques used to implement the instruction set. Computers with different microarchitectures can share a common instruction set. For example, the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs.

Given that a number of neural algorithms and hardware have been developed, there is a need to establish a common neuromorphic instruction set architecture. As architects, we can utilize the ideas behind the implementation of a traditional ISA and use them to implement a general neuromorphic ISA. This will separate the neural algorithm from the execution substrate and will allow the neural network developers to implement neural algorithms without the need to understand the details of the execution hardware. One such mechanism to separate neural algorithm from underlying substrate is discussed in Section 5.1.1 and various optimization tools that can be built on top of such a mechanism are discussed in Sections 5.1.2 and 5.1.3.

3.3 Summary

This chapter briefly describes various software and hardware based learning models and highlights various limitations of these models. It also compares various learning models it terms of their biological fidelity, computational efficiency, and performance. Finally, this section describes some computer architecture tools that can help in developing robust and powerful learning model.

4 BIOLOGICALLY INSPIRED LEARNING MODEL

This chapter provides a detailed description of the biologically inspired learning model proposed in this dissertation in the context of the visual processing stream in the mammalian neocortex. This chapter serves two main objectives. First, it presents and discusses various aspects of the proposed model and justifies their biological inspirations. Second, it highlights various aspects that make the proposed model computationally more attractive than the models discussed in Chapter 3 using various experimental studies.

4.1 Visual Input Pattern Preprocessing

In the mammalian visual cortex, a nerve path transfers visual data from the retina to the LGN cells. During this transmission, the visual data undergoes the log-polar transform (see Section 2.4.1). Modeling biology, before exposing the visual inputs (2D images) to the learning model, they are preprocessed using both the log-polar [104] and the LGN transform [109]. Essentially, if $F(x, y)$ represents a visual input pattern with origin at (x_0, y_0), then the log-polar transform $\mathcal{L}\{F(x, y)\}$ is defined by mapping the image pixels at (x, y) to (σ, ϕ) such that,

$$\sigma = \log(\sqrt{(x - x_0)^2 + (y - y_0)^2} \tag{4.1}$$

$$\phi = \tan^{-1}\left(\frac{y - y_0}{x - x_0}\right) \tag{4.2}$$

The log-polar transform converts the visual patterns from the Cartesian coordinate system to polar coordinate system and provides partial invariance to changes in rotation and scale of the input patterns. Section 4.4.1 demonstrates the amount of invariance that the learning model proposed in this dissertation achieves due to the log-polar transform. This transformed input pattern is processed further with the LGN transform. The LGN transform is contrast sensitive. Thus, contrast among different regions of the input patterns is enhanced after transforming them through the LGN transform. This transformed pattern then becomes the input to the proposed learning model.

4.2 Cortical Columns: Building Blocks for Intelligent Systems

Unlike traditional neural network models (see Chapter 3), remaining true to the biological inspiration is one of the key considerations while implementing various features of the proposed learning model. As discussed in Section 2.2, the neocortex is composed of a hierarchy of uniformly structured entities called the cortical columns. These columns are further classified into hypercolumns and minicolumns. A typical hypercolumn consists of around 50-100 minicolummns. Each of these minicolumns comprises around 200 neurons. Thus, on the average, a typical hypercolumn contains around 10,000 neurons. Chapter 2 also describes that the minicolumns within the same hypercolumn share the same receptive field (see Section 2.1.5) and neurons within the same minicolumn tend to have similar firing preferences [59, 60].

Even though neurons are the basic structural building blocks of the neocortex, in terms of functionality, they are not a very powerful abstraction. On the other hand, cortical columns (both minicolumns and hypercolumns), appear to be a very powerful and biologically inspired functional abstraction. Thus, in order to achieve biological fidelity and to construct powerful learning models, this dissertation proposes a learning model inspired by the structural and functional properties of the cortical columns.

Furthermore, complex neural network models as discussed in Chapter 3, use neurons as their basic functional abstraction. This results in very high computational requirements for such models. In the proposed learning model, computational efficiency is another key consideration. Computational efficiency in this model is achieved by the fact that a single hypercolumn can abstract out the working and computational requirements of 10,000 neurons. Thus, using hypercolumns as the basic functional abstraction within a learning model can provide significant computational speedups.

Figure 4.1 shows a high level schematic of a hypercolumn within the proposed learning model and compares it with a typical biological cortical column. In the proposed hypercolumn model, a hypercolumn contains multiple perceptron like elements, i.e. the minicolumns. Each of these minicolumns contains feedforward, feedback, and lateral inhibitory connections, corresponding weight vectors, a variable threshold value, and an activation function for evaluating its output. Based on the biological findings discussed in Section 2.1, in the proposed model, minicolumns are grouped into hypercolumns with the following rules. First, minicolumns within a hypercolumn are fully connected to all the other minicolumns within

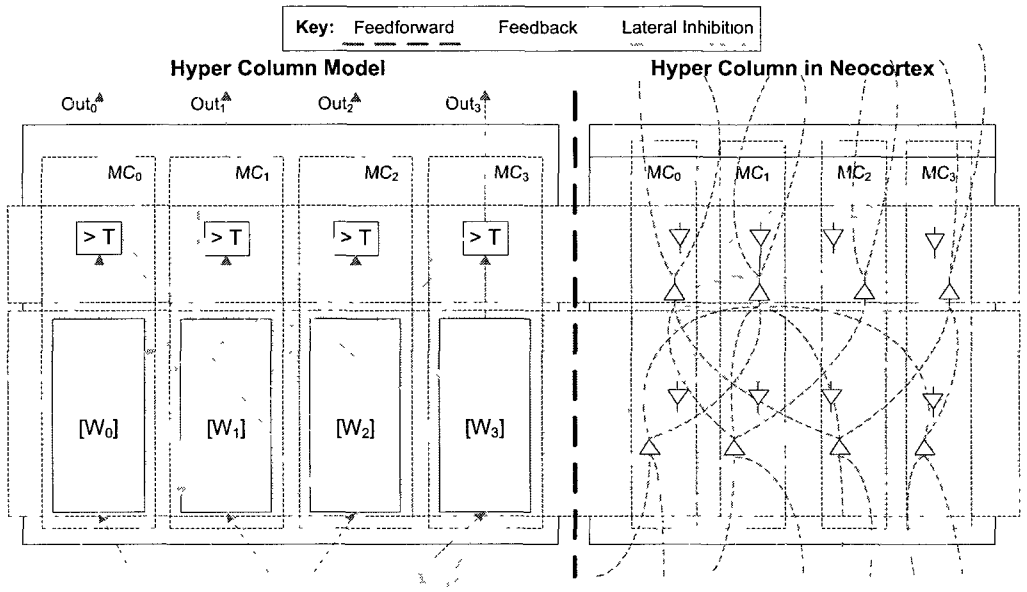

Figure 4.1: Left: The hypercolumn model with feedforward, feedback, and lateral inhibitory paths as defined by the cortical learning algorithm. Right: The structure of a typical biological hyppercolumn. MC=Minicolumn, T=Threshold of Activation Function.

the same hypercolumn via lateral inhibitory connections. This emulates a winner-take-all behavior among the minicolumns within the same hypercolumn. Second, all the minicolumns within the same hypercolumns share the same receptive field, i.e. minicolumns within the same hypercolumn share the same input connections (albeit with different synaptic weights). Third, minicolumns within a hypercolumn receive feedback connections from other minicolumns in upper hierarchical regions. These feedback connections modulate the learning behavior of the minicolumns in the lower hierarchical regions.

4.2.1 Spontaneous Activations and Unsupervised Learning

Unlike traditional neural network approaches, the synaptic weights of minicolumns within the modeled hypercolumns are initialized with near zero values. This suggests that minicolumns do not assume any initial connectivity and the whole network behaves like a blank slate. Thus, initially, none of the minicolumns show a preference to any input pattern.

The minicolumns within the modeled hypercolumn learn through their ability to generate spatially localized and temporally correlated spontaneous activations (see Section 2.4.5). These spontaneous activations, inspired by biology, are a very robust mechanism to initiate the learning process within individual minicolumns. During any training iteration, a minicolumn can become spontaneously active even in the absence of direct stimulus within its receptive field. More precisely, at each time step, each minicolumn has a small probability to become active even if its inputs do not justify that activity. Each minicolumn leverages on this spontaneous activation property for inducing random initial conditions, and for perturbing its weights as the learning process occurs. If a minicolumn spontaneously fires, its synaptic weights corresponding to its active inputs are reinforced. When the spontaneous activation of a specific minicolumn coincides frequently with a particular stimulus, the synaptic weights corresponding to the active inputs of the minicolumn are reinforced. In effect, the frequent co-occurrence of spontaneous activations and the presence of a particular input pattern (due to object permanence) initiates learning. The main advantage of learning through these spontaneous activations instead of initial random initialization of minicolumn synaptic weights is that they help the proposed learning model avoid the scenario where certain features are

Algorithm 1 Pseudo code for evaluation of spontaneous activity behavior of a minicolumn.

```
1: if hasFired == 1 then
2:    spontaneousActivity = 0
3:    hasFired = 0
4: else
5:    spontaneousActivity -= leakValue
6:    {//leakValue determines the time window of past neighbor activations}
7:    if spontaneousActivity < 0 then
8:       spontaneousActivity = 0
9:    end if
10:   for i = 1 to number of minicolumns do
11:      spontaneousActivity += (activity[i] * GaussConnectivity(myID - i))
12:      {//GaussConnectivity(i) = αe^(-i^2/V), V = variance}
13:   end for
14:   spontaneousActivity /= AbsoluteSum(synapticWeights)
15:   {//ensures less spontaneous activity as synaptic weights strengthen}
16:   if spontaneousActivity ⩾ activityThreshold then
17:      activity[myID] = Sigmoid(spontaneousActivity - activityThreshold)
18:      {//Sigmoid(x) = 1.0/(1.0 + e^(-x/β))}
19:      hasFired = 1
20:   end if
21: end if
```

not learned due to the networks disfavor to them because of the initial random weights (See Section 3.1.1.1).

The spontaneous activation property of a minicolumn depends on several factors, including the time since it was last activated, the absolute sum of its synaptic weights, and the recent activity of its neighboring minicolumns. As is observed within the retinal ganglion cells in the visual processing stream (see Section 2.4.5), a minicolumn within the modeled hypercolumn has a higher chance of becoming active if there already is some activity in its neighborhood. The influence of the activity of neighboring minicolumns on the activity of a particular minicolumn is modulated by a Gaussian distribution of relative connection strength. Algorithm 1 shows the pseudo-code for evaluating the spontaneous activation

behavior of a minicolumn. Various components of this pseudo-code are motivated by the analysis presented by Albert et al. [4] about the formation of spontaneous activations within the retinal ganglion cells. Section 4.4.2 evaluates the proposed learning model's ability to learn unique features using these spontaneous activations.

4.2.2 Evaluation of Minicolumn Activity using a Non-Linear Activation Function

Activity of a minicolumn depends on the activations of its inputs weighted by the corresponding synaptic weights. Formally, the output activity of a minicolumn in response to an input vector $\vec{x}$ at any time step is given by a sigmoid function $f(x)$,

$$f(x) = \frac{1}{1 + e^{-g(x)/\beta}} \tag{4.3}$$

$$g(x) = \Theta(x, W) - T \times \Omega(W) \tag{4.4}$$

$$\Omega(W) = \sum_{i=1}^{N} C_i W_i \tag{4.5}$$

$$C_i = \begin{cases} 1.0, & \text{if } W_i > 0.5 \\ 0.0, & \text{otherwise} \end{cases} \tag{4.6}$$

$$\Theta(x, W) = \sum_{i=1}^{N} \gamma(x_i, W_i) \tag{4.7}$$

$$\gamma(x_i, W_i) = \begin{cases} -2, & \text{if } x_i = 1.0 \text{ and } W_i < 0.5 \\ x_i W_i, & \text{otherwise} \end{cases} \tag{4.8}$$

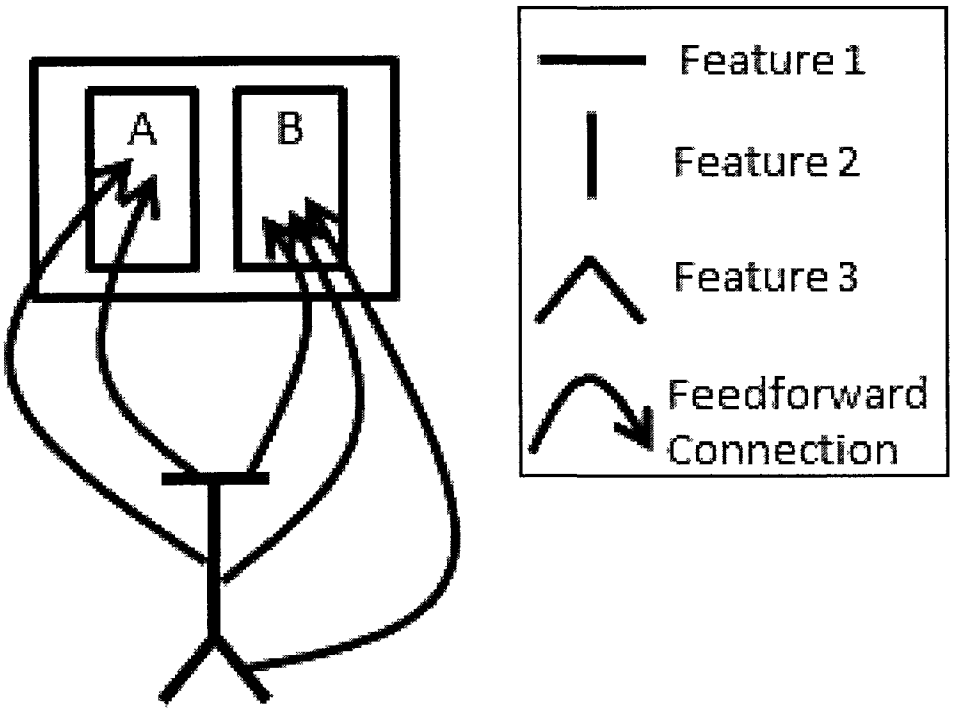

Figure 4.2: A simple example demonstrating the use of the proposed non-linear evaluation of correlation for propoer functional behavior of the hypercolumn model.

β in Equation 4.3 controls the step-like behavior of the sigmoid function. In Equation 4.4, $g(x)$ evaluates a non-linear correlation between the input vector and the synaptic weights of the minicolumn and compares it with $\Omega(W)$ which estimates the firing threshold based on the sum of high synaptic weight values of the minicolumn (see Equations 4.5 and 4.6). Note that $\Omega(W)$ is a (simple) emulation of the *synaptic scaling* phenomenon observed in several regions of the brain [63]. Within the brain, synaptic scaling serves to maintain strengths of synapses relative to each other. T in Equation 4.4 determines the tolerance of a minicolumn to noise and can take a value between 0.0 and 1.0. During each training or

testing epoch, every minicolumn evaluates the correlation $\Theta(x, W)$ between the input vector $\vec{x}$ and its synaptic weights W using Equation 4.7. Typical ANN models define the input of the activation function simply as $\sum x_i W_i$. However, in the proposed model, $\gamma(x_i, W_i)$ in Equation 4.8 can be seen as a reflection of a non-linear evaluation of correlation between the input vector $\vec{x}$ and the synaptic weights W.

If W_i corresponding to an active input x_i is low, W_i contributes negatively to the input of the activation function. Within the neocortex, these non-linear summation properties have been observed in some dendrites [85]. This non-linearity is found to to be necessary for proper functional behavior of the hypercolumn model as demonstrated by a simple example in Figure 4.2. In this figure, as demonstrated by the feedforward connectivity arrows, minicolumn A learns to fire in response to the co-occurrence of features 1 and 2, while minicolumn B learns to recognize the co-occurrence of features 1,2, and 3. In the absence of such a non-linear correlation evaluation, along with firing for the co-occurrence of features 1 and 2, minicolumn A will also fire whenever features 1,2, and 3 appear in its receptive field. Using the simple biologically inspired non-linear activation rule, such a situation can be prevented.

4.2.3 Lateral Inhibition and Independent Feature Identification

When an input $\vec{x}$ is presented to the hypercolumn, none of the untrained minicolumns fire for that input. However, if the random firing activity of a minicolumn coincides with the occurrence of an input pattern, that minicolumn adjusts its weights so that the correlation

between the input and the weights is improved. This is achieved by strengthening the weights corresponding to the inputs x_i that are currently active. Thus, over multiple iterations a minicolumn learns to identify a feature that initially coincided with the random activity of the minicolumn. At the same time, each minicolumn inhibits neighboring minicolumns from firing via lateral inhibitory connections for the pattern it has already learned to recognize. If multiple minicolumns fire at the same time, the one with the strongest response inhibits the ones with weaker responses. The inhibited minicolumns then weaken their weights corresponding to highly active x_i so that their dot-product with the input is minimized. As a result of this process, the hypercolumn network is able to recognize unique patterns without any supervision. Section 4.4.2 demonstrates the ability of the proposed learning model to learn independent features in an unsupervised manner.

A very interesting byproduct of having minicolumns learn independent features through spontaneous activations and lateral inhibition is inherent fault tolerance: if a minicolumn that is firing for a feature suddenly dies (permanent hardware or software error in a future synthetic application), over time, another available neighboring minicolumn will start firing for that feature. This makes the hypercolumn structure inherently tolerant to permanent faults (see section 5.2).

4.2.4 Minicolumn Weight Update Rule

Hebbian learning [15] is a dominant form of learning in large-scale biological neural networks. With Hebbian learning, if one input of a neuron has strong activation, and that neuron

itself has a strong output, then the synapse (synaptic weight) corresponding to that input is reinforced. Intuitively, if the input is strong at the same time as the output, it means that input plays a significant role in the output and should be reinforced. According to this definition, the synaptic weight W_i is increased if the input x_i to the minicolumn is active (emulating long-term potentiation), or decreased if the input x_i to the minicolumn is inactive (emulating long-term depression). This weight modification is only applied to active minicolumns x_i in accordance to Hebbian learning. Each time a minicolumn fires it modifies its weights so that its correlation with the input pattern that has caused it to fire increases. Weights are strengthened using the following update rule.

$$W_i = x_i \times \left(W_i + \left(\gamma \times \frac{1.0}{1.0 + e^{(-\frac{W_i - C}{\beta})}} \right) \right) \tag{4.9}$$

Here, x_i is the input corresponding to W_i, and C defines how the present W_i will affect the weight update. Finally, β controls the step-like behavior of the sigmoid weight update function and γ determines the learning rate.

In this weight strengthening rule, the update added to W_i is dependent upon the present value of W_i. This means that if W_i is strong it will get a higher update value. Essentially, if a minicolumn with strong synaptic weights fires in response to an input, its synapses are updated with higher value as compared to the scenario when a minicolumn with weak synaptic weights fire in response to an input. This is in accordance with biological data [110, 121].

In the case when a minicolumn is inhibited, it modifies the weights using the following

update rule.

$$W_i = x_i \times (W_i - \delta) \tag{4.10}$$

Here, δ defines the weight update rate in the presence of inhibition. As a result of these weight update rules, at each level, minicolumns will progressively react most strongly to inputs they receive repeatedly, in effect *learning* them. In the visual cortex, these inputs correspond to shapes, which become increasingly more complex in the upper levels.

4.2.5 Learning to Forget

Apart from updating the weights in the presence of excitation and inhibition, the weights also decay over time. This is quite similar to the forgetting behavior in animals. This update is done using a rule quite similar to the one used for excitatory updates and is given by.

$$W_i = W_i - \left(\epsilon \times \left(1 - \frac{1.0}{1.0 + e^{(-\frac{W_i - c}{\beta})}}\right)\right) \tag{4.11}$$

Here, ϵ determines the increase in forgetting rate proportional to the current weight value. It should be noted that $\epsilon << \gamma$. This insures that the forgetting rate is significantly slower than the learning rate.

4.2.6 Evaluation of Hypercolumn Activity

In our model, a hypercolumn is an abstract representation of a group of minicolumns that are tightly bound together via lateral inhibitory connections. A hypercolumn gets active if

any of the minicolumns within that hypercolumn gets activated. The minicolumns within a hypercolumn follow a winner-take-all approach i.e. if multiple minicolumns within a hypercolumn become active, the minicolumn with the strongest activation inhibits the rest of the activated minicolumns. Thus at any moment in time only one minicolumn within a hypercolumn is active. Furthermore, the inhibited minicolumns modify their synaptic weights to decrease their correlation with the present input so that they do not become activated by the same input pattern in the future. This results in minicolumns within a hypercolumn learning independent/unique inputs exciting the receptive field of the hypercolumn.

4.3 Hierarchy to Realize Complex Tasks

A single modeled hypercolumn can extract simple features from occurring within its small receptive field, but to realize complex tasks, these hypercolumns need to be hierarchically organized. This hierarchical organization of hypercolumns is also inspired by the biological example. Within the neocortex, especially in the visual cortex, there is a clear evidence of hierarchical organization of cortical columns (See Section 2.4.2). Figure 4.3 shows a two level hierarchical network of hypercolumns.

Hierarchical organization of modeled hypercolumns provides three main functions.

- Arranging the hypercolumns in the form of a hierarchy increases the overall receptive field of the hierarchical network as hypercolumns in upper levels can react to the activations of multiple lower level hypercolumns.

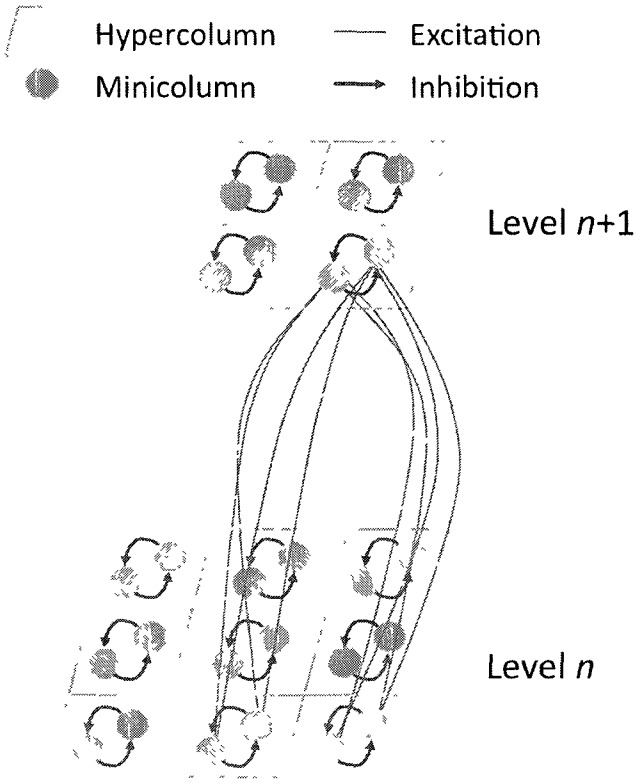

Figure 4.3: Organization of the connectivity. For clarity, this scheme only considers two minicolumns per hypercolumn and only shows some of the connections of a single hypercolumn in level $n + 1$. Further, excitatory connections can be in both feedforward and feedback direction. [43]

- In this hierarchical organization, using global context, minicolumns in the higher areas can modulate the response and learning behavior of hypercolumns in the lower hierarchical areas.

- Hierarchical organization of hypercolumns allows the network to develop the notion of *automatic abstraction*, where hypercolumns in lower regions learn to recognize simple features while hypercolumns in the upper regions learn to identify complex shapes.

Throughout the hypercolumn hierarchy, processing of a minicolumn activity depends on both the feedforward and feedback information.

4.3.1 Role of Feedforward Information Processing

The first type of information processing within the hypercolumn hierarchy is applied to the feedforward paths. Throughout the hierarchy, minicolumns within multiple hypercolumns at level n connect to minicolumns within a single hypercolumn at level $n+1$. As a result, the number of hypercolumns within a level decreases as we move up the hierarchy. Furthermore, receiving inputs from multiple lower level hypercolumns increases the effective receptive field of upper level hypercolumns.

As described in Section 4.2, the main role of feedforward information processing is to allow individual minicolumns within a hypercolumn to learn independent features from the input data occurring within their receptive field. Apart from this straight-forward application, within the proposed hypercolumn hierarchy, the feedforward processing also help hypercolumns generate feature maps and invariant representations using object permanence.

4.3.1.1 Generation of Topographic Feature Maps

It is well understood that topological feature maps are present in various cortical areas, as evidenced by a number of experiments [60, 126]. For example, tono-topic maps maps are known to arrange frequencies from low to high along the surface of the auditory cortex; color and orientation maps in the visual cortex are arranged similarly. In the proposed cortical

model, emergence of such feature maps is also investigated. The main hypothesis is that object permanence, the fact that natural stimuli maintain a temporal presence, plays a significant role in the formation of these maps. Spatially correlated spontaneous activations (as described in Section 4.2.1) contribute the second part to this equation. During learning, an object will maintain some level of permanence, even though the features perceived by sensory input may vary as the object moves or slightly changes its form. The spatially localized and temporally correlated spontaneous activations of the minicolumns within a hypercolumn encourage each of these features to be learned by a localized group of minicolumns, creating a robust topologically organized feature map.

To promote the formation of such feature maps, the hypercolumn network is trained on variations of the same object in a sequential manner (as opposed to random training inputs). Using object permanence and temporally correlated spontaneous activations, minicolumns responding to variations of a pattern are lumped together. Thus, over time, well-segregated topological feature maps for each digit emerge from a training regimen that respects object permanence. Essentially, if a minicolumn learns a specific feature and fires in response to it, it primes its neighboring minicolumns to fire. This increases the probability of the primed neighboring minicolumns to generate spontaneous activity in the future epochs. Due to object permanence, as the object slightly varies, the slightly different features are in turn learned by some of the neighboring minicolumns. As a result, variations of similar features are learned by minicolumns that are spatially localized. In this manner, using the concept of object permanence and spatially localized and temporally correlated spontaneous activations,

the proposed learning model generates hierarchical feature maps.

It should be noted that the proposed method of generating the feature maps is more attractive than the traditional SOMs discussed in Section 3.1.1.4 in two ways.

- The proposed method is computational quite efficient as compared to traditional SOMs as there is no need to evaluate the distance metric to determine spatial positioning of the node that learned a new feature within the network.
- A major shortcoming of traditional SOMs, i.e. difficulty to construct hierarchical relations, is overcome is a very straight-forward manner, simply by applying the correlated spontaneous activity rule throughout the hypercolumn hierarchy.

Section 4.4.3 evaluates the ability of the proposed learning model to develop hierarchical feature maps using spatially localized and temporally correlated spontaneous activations.

4.3.1.2 Gaussian-like Feedforward Connectivity for Invariant Representation

Within our cortical network, inputs to a minicolumn in a hypercolumn at Level n are actually the outputs of multiple minicolumns in the hypercolumns at Level $n-1$. Thus, an active minicolumn creates connections with the lower level active minicolumns within its receptive field. However, since the minicolumns responding to variations of the same pattern are topologically organized within a hypercolumn (as discussed in Section 4.3.1.1), a higher level minicolumn has the opportunity to learn feature invariance by forming connections to the minicolumns neighboring the lower level active minicolumn. To achieve this feature invariance, an active minicolumn in Level n also creates connections to the neighbors of the

Algorithm 2 Pseudo code for evaluation of a minicolumn's response to an input stimulus.

```
1: correlation = 0
2: for i = 1 to receptiveFieldSize do
3:   for j = 1 to receptiveFieldSize do
4:     correlation += (synapticWeight[i] * input[j] * GaussConnectivity(i − j)
5:     {//GaussConnectivity(i) = αe^(−i²/V), V = variance}
6:   end for
7: end for
8: for i = 1 to receptiveFieldSize do
9:   if synapticWeight[i] > 0.5 then
10:    foundActiveMinicolumn = False
11:    for j = −GAUSSIANWIDTH/2 to GAUSSIANWIDTH/2 do
12:      if input[i − j] > 0.8 then
13:        foundActiveMinicolumn = True
14:      end if
15:    end for
16:    if foundActiveMinicolumn == False then
17:      correlation −= 2
18:    end if
19:  end if
20: end for
21: {//The above piece of code models Equations 4.7 & 4.8}
22: threshold = 0
23: for i = 1 to receptiveFieldSize do
24:   if synapticWeight[i] > MINVALUE then
25:     threshold += synapticWeight[i]
26:   end if
27: end for{//The above for-loop models Equations 4.5 & 4.6}
28: activity[myID] = Sigmoid(correlation, threshold) {//Models Equation 4.3}
29: {//Sigmoid(x, y) = 1.0/(1.0 + e^(−g(x,y)/β))}
```

active minicolumn in Level $n-1$ using a Gaussian-like feedforward connectivity. Due to the formation of topologic feature maps, it is highly likely that these neighboring minicolumns have learned variations of the same pattern. Here, the width of the Gaussian determines the resolution of a minicolumn. If the Gaussian is narrow, the minicolumn is very specialized and will respond only to a single minicolumn (or feature) below it. However, if the Gaussian is

wide, the minicolumn is quite generalized and will respond to many minicolumns located in the same topological area of the feature map. Algorithm 2 shows the pseudo-code for evaluating a minicolumn's response to its inputs using the described Gaussian like connectivity.

The notion of this Gaussian-like connectivity is rooted in biology where neurons connect to other neurons (either within or between columns) with a Gaussian probabilistic law, i.e., the probability of connection decreases with distance. Empirically, the connectivity is actually fairly dense (about 70%), almost uniform, within a given neighborhood [68], and drops afterward [109]. Section 4.4.4 demonstrates and evaluates the role of this Gaussian-like feedforward connectivity within the proposed learning model.

4.3.1.3 Automatic Abstraction

Another powerful tool that the feedforward information processing system provides within the proposed hierarchical hypercolumn model is the notion of automatic abstraction. This notion of automatic abstraction is explained in a hypercolumn hierarchy, using our visual cortex example. Each minicolumn of the first level "samples" the input pixels within its receptive field, see level 1 in Figure 4.4. The subsequent minicolumns progressively respond to the activations of the minicolumns within their receptive fields and thus learn complex shapes. The first-level minicolumns connect to the output of the LGN cells. Based on the initial spontaneous activation of these minicolumns, connections between the first level minicolumns and the LGN cells within their receptive field are established.

Figure 4.4 demonstrates how the feedforward information processing utilizes the idea

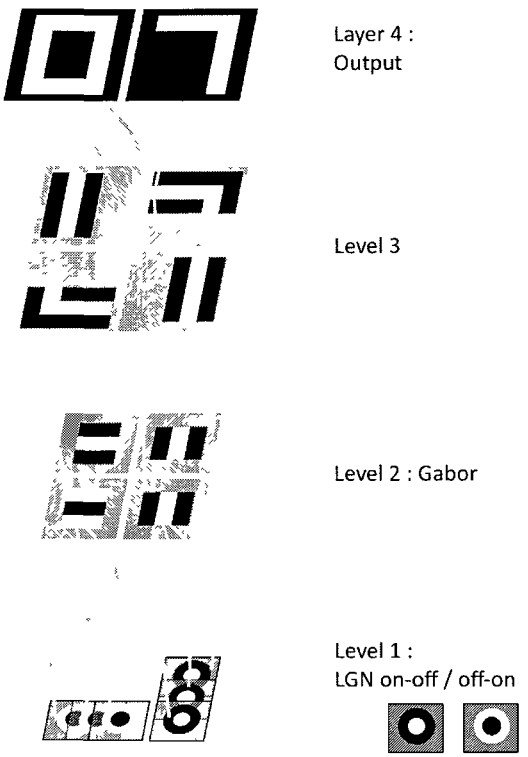

Figure 4.4: Illustration of automatic abstraction of complex objects applied to visual processing with a 4-layer hierarchy. For clarity, the minicolumn-hypercolumn structure is not illustrated and only a fraction of the connections is illustrated. [43]

of independent feature detection and automatic abstraction to learn and recognize various complex shapes. As explained in Section 4.1, the LGN cells detect contrasts and extract contours. These contours of most natural shapes decompose into tiny segments at a fine granularity. The first-level minicolumns, initially by virtue of spontaneous firing, respond to such segments, and afterwards are strengthened through the proposed Hebbian learning based weight update rules (See Section 4.2.4). These segments progressively emerge as some of the dominant shapes in the first-level columns, (see Figure 4.4). This behavior is supported by biological evidence of the existence of orientation specific neuron populations within

the primary visual cortex [60]. The subsequent levels continue the same process and learn increasingly complex shapes. For instance, combinations of segments can produce crosses, angles, and other complex shapes.

The minicolumns higher up in the hierarchy correspond to the aggregate information (sum) of an increasingly high number of lower level minicolums. As a result, their receptive field rapidly becomes a murky combination of more simple shapes, and does not carry a crisp semantic. Here, lateral inhibitory connections enable a necessary filtering role by silencing weak minicolumns, and allowing crisper shapes (with richer semantics) to emerge in upper-level columns. These lateral inhibitory connections implement a form of max operators among clusters of MCs. Section 4.4.2 demonstrates and evaluates the ability of the learning model proposed in this dissertation to develop automatic abstraction.

4.3.2 Role of Feedback Information Processing

As described in Section 2.4.6, feedback paths play a very important rule within the neocortical hierarchy. One of the major contributions of this thesis is to describe a learning model that effectively uses the feedback processing paths during the training and the testing phase. Throughout the hypercolumn hierarchy, feedback paths play three critical roles.

- Modulate the widths of the inter-level Gaussian connectivity described in Section 4.3.1.2 during the learning phase.

- Force lower level minicolumns to pool variations of the same feature.

- Help lower level minicolumns un-pool exceptions from generalized representations.

These three roles of feedback processing paths are described in detail in the following sections.

4.3.2.1 Width Modulations of Gaussian-like Feedforward Connectivity

Minicolumns within a hypercolumn are initialized with a very narrow Gaussian-like connectivity, so each minicolumn within a hypercolumn extracts very specific and unique features from their receptive field. In the first phase of training, these narrow Gaussian connections are maintained, and the network uses the Hebbian plasticity rules to achieve 100% recognition rate for the training data. In the second phase of training, these learned synaptic connections are maintained, but the Gaussian-like connectivity is modified to generalize the representations of unique features learned at various hierarchical levels. Since this widening occurs for all levels of a cortical network, the generalization is distributed in a hierarchical manner. Upper hierarchical levels modulate (widen or narrow) the connectivity of lower levels using the feedback processing paths. It should also be noted that the second phase of training uses the same training data as the first phase.

First, the minicolumns at the top-most level of the hypercolumn hierarchy use an external supervisory feedback signal to widen their Gaussian connectivity so long as the false-positive rate (the number of patterns recognized incorrectly) stays below a particular threshold. For our experiments, this threshold is set to be 0 (i.e. no false positives allowed). Once the false positive rate at a certain hierarchical level increases beyond the specified threshold, the Gaussian update process is shifted to the next lower level through a feedback path.

Algorithm 3 Pseudo code for updating the Feedforward Gaussian like connectivity of the minicolumns using feedback processing paths.

```
1: for i = 1 to N do
2:    {//Gaussian widths are updated for each unique pattern class}
3:    DeactivateAllMinicolumns()
4:    ExposeAllVariationsOfPatternToNetwork(i,'TrainingSet')
5:    for k = 1 to hypercolumnsAtLevel[j] do
6:       activeMinicolumnList = GetActiveMinicolumns(j, k)
7:       {//GetActiveMinicolumns(j,k) provides the list of active minicolumns in hypercolumn k at
         level j}
8:       while EvaluateFalsePositiveRate('TrainingSet') <= falsePositiveThreshold
         do
9:          {//falsePositiveThreshold is set to 0 in our experiments}
10:         IncreaseGaussianWidth(activeMinicolumnList,INCREMENT)
11:      end while
12:      DecreaseGaussianWidth(activeMinicolumnList,INCREMENT)
13:      {//Brings the false positive rate below the threshold again.}
14:      for j = 1 to NumChildren do
15:         TriggerChildGaussianConnectivityModulation(j)
16:      end for
17:   end for
18: end for
```

Eventually, the Gaussian update process reaches the lowest hierarchical level, and the widths of the Gaussian-like connectivity of all the hierarchical levels of the network are locked. While this simple heuristic proves to be effective for simple image recognition tasks, we believe that more complex update rules for feedforward Gaussian-like connectivity will be required for more complex datasets. Algorithm 3 shows the pseudo-code for updating the Feedforward Gaussian-like connectivity of the minicolumns. Section 4.4.4 evaluates the improvement in the performance of a hierarchical hypercolumn network due to the feedback based modulation in widths of the Gaussian-like connectivity.

4.3.2.2 Pooling to Develop Invariant Representations

Another important role that the feedback plays throughout the hypercolumn hierarchy is it allows minicolumns at different levels to develop invariant representations of the features they have learned to recognize. The feedforward learning process enables our cortical hierarchy to learn unique features from the input patterns. Each of the minicolumns can fire for minor variations of the same patterns but variations of a patterns with major differences are recognized as different patterns. This means that two variations of the same pattern may be recognized as two different patterns by different minicolumns. To resolve this issue and generate a robust invariant representation for variations of the same pattern, the hypercolumn hierarchy utilizes the supervised feedback based pooling algorithm.

Let us assume that during the training phase the hierarchical network has started to recognize a pattern. Now it is exposed to another variation of the same patterns that is quite different from the previous one e.g. two different variations of the same handwritten digit. At this point, only some of the minicolumns within the hierarchy may fire. Since there is not enough feedforward evidence, the top level minicolumn that is supposed to fire for that pattern does not fire. If this behavior persists, new minicolumns will train themselves to recognize features in the new variation that are quite different from the original pattern and over time, that new variation will be identified as a new pattern. This will be marked by firing of a new minicolumn in the top level of the hierarchy in response to that new variation. At this point, the top level hypercolumn receives an external feedback signal. This feedback signal forces the minicolumn firing for the original pattern to fire and at the same time

inhibits the top level minicolumn that is firing for the new variation. Now, the minicolumn receiving excitatory feedback adjusts its weights so that in the future it fires for the new variation. On the other hand, the inhibited minicolumn changes its weights so that it does not fire for that input pattern. Thus, over multiple exposures, the minicolumn firing for the original pattern will also start to fire for the new variation. Figures 4.5 and 4.6 explain this feedback based pooling using a simple example.

Once the top level minicolumn starts to give a stable activation for both the variations, it will start to send the feedback signal to the hypercolumns in the lower levels so that lower level minicolumns can also create pooled representations. The amount of feedback sent to each of the lower level minicolumns is proportional to its firing history: if a minicolumn has been firing a lot in the past, it will get stronger feedback. Thus, over time the most active minicolumn ends up pooling its child minicolumns to generate invariant representations. One of the by-products of the proposed feedback algorithm to develop invariant representations is the it results in significant resource optimizations. As minicolumns get freed, they can be used to recognize other features. The process of generating invariant representations within a minicolumn using feedback is explained in the pseudo-code provided in Algorithm 4. In Algorithm 4, UpdateSynapticWtsExcitatory models the functionality of Equation 4.9 while UpdateSynapticWtsInhibitory models Equation 4.10. Section 4.4.5 evaluates the improvements in resource utilization of a hypercolumn network due to the feedback based pooling algorithm and also present various invariant representations that the network generates for different input patters.

Algorithm 4 Pseudo code for generating invariant representations within a minicolumn using supervised feedback.

```
if feedback > 0 then
  if hasNotFired then
    if hasMaxFiringHistory then
      UpdateSynapticWtsExcitatory(feedback)
    end if
  else
    if hasMaxFiringHistory then
      UpdateSynapticWtsExcitatory(feedback)
      if isStable then
        for i = 1 to N do
          if IsActive(child[i]) then
            SendFBToChild(i, feedback)
          end if
        end for
      end if
    else
      UpdateSynapticWtsInhibitory(feedback)
    end if
  end if
end if
```

4.3.2.3 Un-pooling Exceptions from Generalized Representations

The feedback based pooling algorithm described in Section 4.3.2.2 helps the hypercolumn hierarchy to develop invariant representation of complex patterns by pooling different variations of the same pattern in a hierarchical manner. Relying only on this simple pooling scheme to develop invariant representations can also result in pooling exceptions with the generalized representation, e.g. the representation of letter 'Q' may pool with the generalized representation of letter 'O'. To unpool an exception from the generalized representation, an algorithm similar to the one proposed for feedback based pooling is implemented.

In the case when an exception needs to be unpooled from the generalized representation,

the minicolumn responding to the exception is inhibited via an external supervisory signal. This allows some other minicolumn within the same hypercolumn to learn the exception. Now, the firings of the minicolumn that has recently learned to recognize the exception coincide with the inhibition being received by the minicolumn with the generalized representation. At this point, the generalized minicolumn establishes a strong inhibitory connection with the minicolumn that has learned to recognize the exception. As a result, the generalized minicolumn stops firing for the exception as it is inhibited by the minicolumn responding to the exception. It should be noted that the inhibitory connection between the minicolumn responding to the exception and the generalized minicolumn is in addition to the lateral inhibitory connections that exist among all the minicolumns within a hypercolumn.

4.3.3 Hypercolumns as Universal Boolean Approximators

Both the feedforward and the feedback processing paths allow the hypercolumn network to approximate both linearly separable boolean operations like AND and OR and linearly inseparable boolean operations like XOR. A useful conceptual example of such a behavior is the exclusive-or (XOR) operation which incorporates both AND and OR operations in it. An XOR is symbolically represented as $\oplus$ and is defined as,

$$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B) \tag{4.12}$$

Figure 4.5 shows a 2-level hypercolumn network that initially learns to identify $(A \wedge \neg B)$, $(\neg A \wedge B)$, and $(A \wedge B)$ as three separate entities, i.e. three different minicolumns fire for

each of the two patterns. Minicolumn 1 in hypercolumn 0 in level 1 (L1H0M1) fires for $(A \wedge \neg B)$, Minicolumn 3 in hypercolumn 0 in level 1 (L1H0M3) fires for $(\neg A \wedge B)$, and Minicolumn 2 in hypercolumn 0 in level 1 (L1H0M2) fires for $(A \wedge B)$. At this point, the external supervisory signal triggers the pooling mechanism described in Section 4.3.2.2 in L1H0M1 so that it learns $A \oplus B$. As described in Section 4.3.2.2, the feedback based pooling mechanism inhibits L1H0M3 from firing for $(B \wedge \neg A)$ and forces L1H0M1 to fire for $(B \wedge \neg A)$. As a results, L1H0M1 starts pooling the inputs of L1H0M3. Over time, the input of L1H0M3 migrate to L1H0M1 and L1H0M1 starts to fire for both $(A \wedge \neg B)$ and $(\neg A \wedge B)$. Now, L1H0M1 has developed a generalized representation for $A \oplus B$ (See Figure 4.6). The problem is that because of the pooling process, L1H0M1 also starts to fire for $(A \wedge B)$. This means that L1H0M1 has developed an over-generalized representation and the exception $(A \wedge B)$ needs to be un-pooled from this generalized representation. At this point, the unpooling mechanism triggers. Whenever L1H0M1 fires for $(A \wedge B)$, it gets a negative feedback signal from the external supervisor. As a results L1H0M1 is inhibited from firing for $(A \wedge B)$. At the same time, L1H0M2 fires for $(A \wedge B)$. Since the firing of L1H0M2 coincides with the external inhibition received by L1H0M1, it develops a specific inhibitory connection with L1H0M2 (See Figure 4.7). This inhibitory connection is different from the lateral inhibitory connections discussed in Section 4.2.3. This specific inhibitory connection from L1H0M2 to L1H0M1 prevents L1H0M1 from firing whenver L1H0M2 fires for $(A \wedge B)$. Thus, using the unpooling mechanism, the hypercolumn in level 1 has unpooled the exception $(A \wedge B)$ from the generalized representation. By this process, L1H0M1 only fires for $A \oplus B$ while L1H0M2

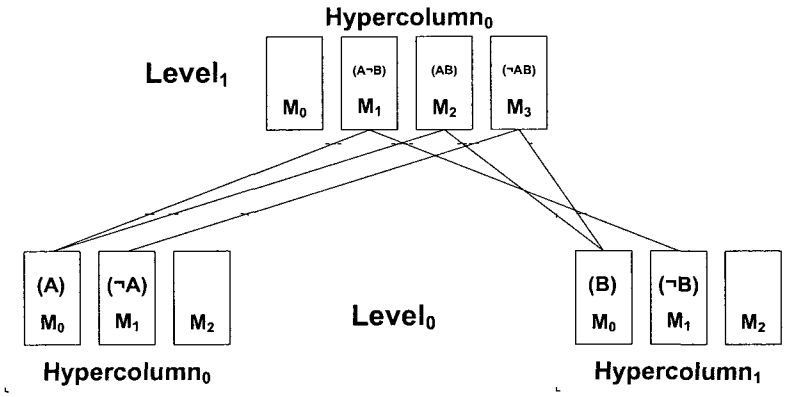

Figure 4 5 Initial hypercolumn hierarchy state L1H0M1 learns to recognize $(A \wedge \neg B)$, L1H0M3 learns to recognize $(\neg A \wedge B)$, and L1H0M2 learns to recognize $(A \wedge B)$

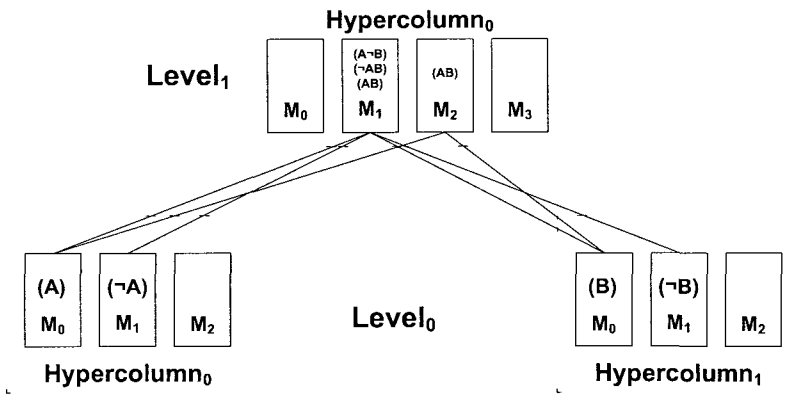

Figure 4 6 After the pooling process triggers, L1H0M1 pools the input of L1H0M3 and fires for $(A \wedge \neg B)$, $(\neg A \wedge B)$, and $(A \wedge B)$

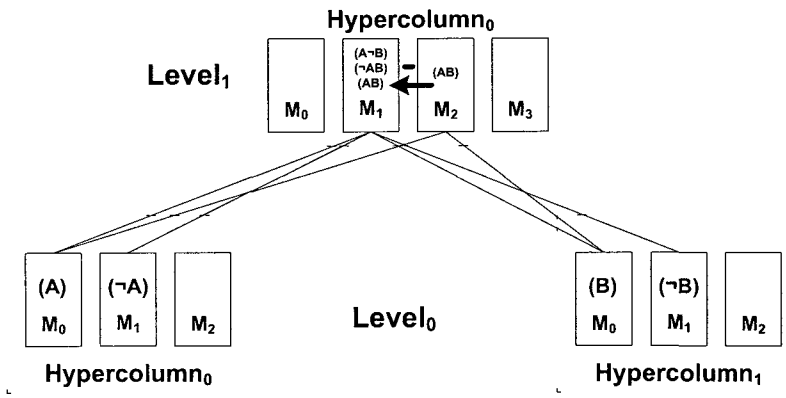

Figure 4 7 After the unpooling process triggers, L1H0M2 creates a specific inhibitory connection with L1H0M1 so that L1H0M1 does not fire for $(A \wedge B)$ Note that this specific connection is different from the regular lateral inhibitory connections discussed in Section 4 2 3

only fires for (A$\wedge$B), while L1H0M3 is declared redundant and can be used to recognize some different patterns. In order to correctly learn A $\oplus$ B, traditional artificial neural networks need to add an extra hidden layer in between the input layer and the output layer. In the proposed hypercolumn network, using the described pooling and unpooling mechanisms, A $\oplus$ B is correctly learned using just two levels. Using the feedforward processing paths, the hypercolumn can approximate AND operations. Furthermore, the feedback based pooling allows the hypercolumn to learn the OR operations. Finally, the feedback based unpooling allows the hypercolumn to learn the XOR operations.

4.4 Experimental Results

This section presents results demonstrating and validating several powerful aspects of the proposed the hierarchical hypercolumn model and various hypothesis presented in the previous sections. Section 4.4.1 validates the hypothesis that the log-polar transform plays a crucial role in providing rotation and scale invariance to a small extent in the early vision processing stages. Section 4.4.2 validates the ability of learning model proposed in this dissertation to learn unique features in an unsupervised manner by using spontaneous activations and lateral inhibitions. By combining different features in a hierarchical manner, the network learns to recognize complex shapes without any external supervisory signal. Section 4.4.3 validates the hypothesis that object permanence and temporally correlated spontaneous activations provide a simple mechanism to develop hierarchically organized feature maps. This section also demonstrates the ability of the hypercolumn network to develop hierarchical feature maps by

relying on simple biological mechanisms. Due to object permanence and temporally correlated spontaneous activations, variations of the same pattern a grouped together in a hierarchical manner. Section 4.4.4 evaluates the improvement in the performance of a hierarchical hypercolumn network by developing a Gaussian-like connectivity between various hierarchically organized feature maps. Significant improvements in terms of average recognition rate of the network are observed simply by modulating the widths of the feedforward Gaussian connectivity. Section 4.4.5 validates the hypothesis that feedback based pooling provides a robust mechanism to generate invariant representations for different variations of the same pattern. This feedback based pooling mechanism improves the performance of the network in terms of recognition rate and resource utilization. By pooling variations of the same patterns, minicolumns within a hypercolumn, previously responding to the pooled variations, can be used to learn other unique patterns. Section 4.4.5 compares the performance of the proposed hierarchical hypercolumn network with the contemporary state-of-the-art neural network implementations including convolution neural networks (CNN) and hierarchical temporal memory (HTM). This section clearly demonstrates that the proposed hypercolumn network outperforms both CNN and HTM in terms of generating robust invariant representations of the input patterns when using a very modest number of training samples.

For all the experiments discussed in this section, handwritten digits obtained from the MNIST database [81] are used. The MNIST database consists of 60,000 training images of handwritten digits from 0 to 9 and 10,000 test images. Each of the handwritten digits is a gray-scale image of size 28x28 pixels. A subset of these handwritten images is shown in Figure 4.8.

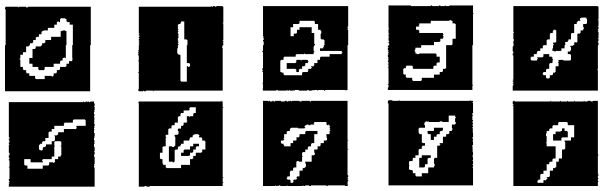

Figure 4.8: Sample of handwritten digits obtained from MNIST database.

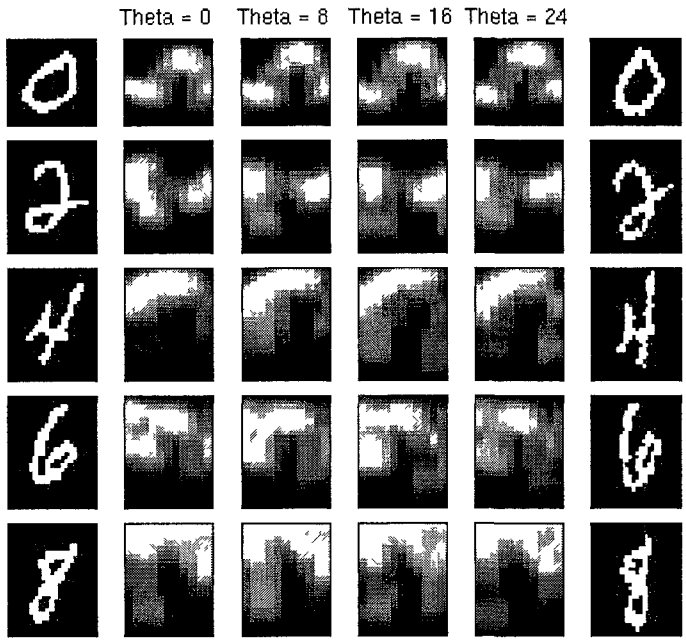

Figure 4.9: Digits and corresponding averaged log-polar transforms for different orientations between 0-24 degrees with an increment of 8 degrees.

All of these handwritten digit images are preprocessed using the log-polar/LGN transform described in Section 4.1 before they are exposed to a hypercolumn network. Figure 4.9 shows a subset of digits and their corresponding averaged log-polar transformed images at different orientations. Each of the log-polar transformed images corresponds to the digit rotated at a certain angle. The angle of orientation of each of the digits is varied from 0-24 degrees with an increment of 8 degrees. Figure 4.9 demonstrates that as the actual image is rotated, the corresponding averaged log-polar transformed images look quite similar. These log-polar

Level	Hypercolumns (HC)
4	1
3	3
2	6
1	12
0	24

Table 4.1: Description of the hierarchical hypercolumn network created for recognition of handwritten digit images.

transformed images incorporate the first level of scale and rotation invariance provided by the hypercolumn network. It should be noted that this first level of invariance relies on feedforward information processing only. Finally, for all the experiments described in this section, the hierarchical organization of hypercolums described in Table 4.1 is used.

4.4.1 Experiment 1: Invariance due to Log-Polar Transform

The first experiment validates the hypothesis that the log-polar transforms plays a key role in providing invariance to rotation and scale variations to a small extent in early vision processing stages. In this experiment, invariance achieved by preprocessing the digit images with the log-polar transform is studied. For this experiment, the hypercolumn network described in Table 4.1 is initialized with 10 minicolumns in each of the hypercolumns and the network is trained with one variation of each of the digits (0-9). The network is then tested with several rotated versions of all the digit variations in the training dataset and the overall recognition rate is measured. Figure 4.10 shows the results of this experiment. The maximum rotation invariance ($\pm$30 degrees) is achieved for digit 2 while the minimum rotation invariance is observed for digit 1 ($\pm$10 degrees). On average, the network demonstrates about $\pm$15 degrees

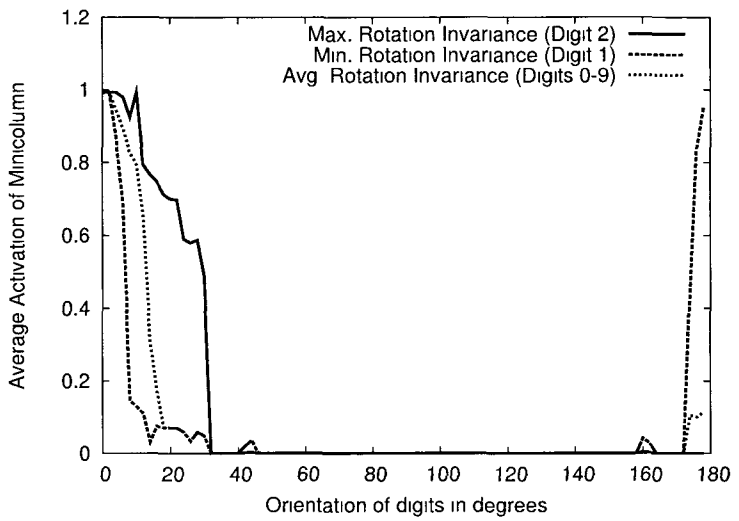

Figure 4.10: Average rotation invariance achieved by the network when tested with rotated digit variations.

of rotation invariance. These results conform with the rotation invariance observed in the mammalian visual cortex [59]. Similar network behavior was observed for scale invariance as well.

Modeling biology, invariant representations are learned by the hypercolumn network at different levels. The Log-polar transform applied on the input images is one of these levels. Even though the invariance developed due to the log-polar transform can not withstand large variations in rotation and scale, the log-polar transform plays a key role in terms of creating invariance to minor rotation and scale variations. This helps in improving the recognition rate of the hypercolumn network as well as the resources required by the network to robust recognition of input patterns. Resource requirement is measured in terms of the number of minicolumns throughout the hypercolumn hierarchy that have locked themselves to identify a certain feature within the training dataset. The recognition rate of the network is improved

because different variations of the same digit with minor rotation and scale variations are recognized by the same minicolumns. This also helps in lowering the resource requirements of the network as at each hierarchical level extra minicolumns are not allocated for each of the slightly different variations of the same input image.

4.4.2 Experiment 2: Independent Feature Identification and Automatic Abstraction

The second experiment verifies the hypothesis that the spontaneous activations ability of the minicolumns along with strong lateral inhibitory connections among the minicolumns within a hypercolumn provides an unsupervised mechanism for independent features identification and automatic abstraction. In this experiment, only the feedforward information processing and learning algorithm is tested and the independent feature identification capability of the proposed hierarchical hypercolumn model is validated. For this experiment, feedback processing paths are disabled and very narrow feedforward Gaussian-like connectivities (See Section 4.3.1.2) are enforced. Since there is no feedback and very narrow connectivity widths, it is anticipated that in Level 4 (top most level of the hierarchy), different minicolumns will recognize significantly different variations of same digits. For this experiment, each of the hypercolumns is initialized with 100 minicolumns and 100 handwritten digit images (10 variations of each digit) from the MNIST database are used. The network is trained with these 100 variations until it achieves 100% recognition rate on the training set.

Figure 4.11 shows the results of this experiment. This figures plots the number of

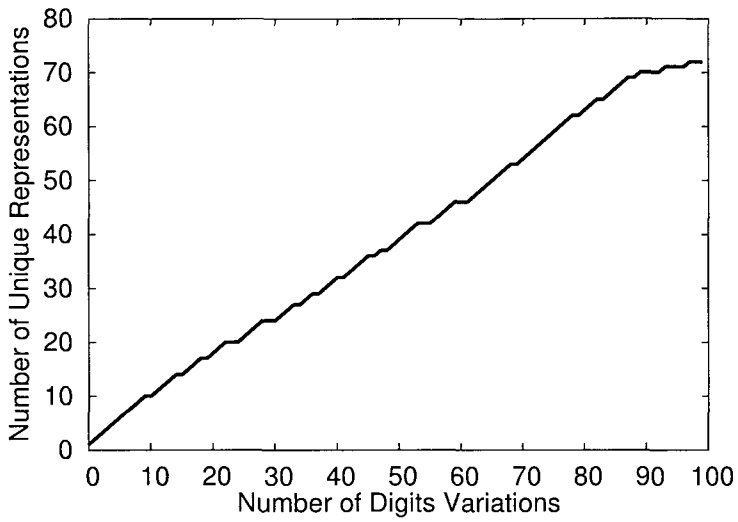

Figure 4.11: Unique digit variations learned by the hierarchical network in the absence of feedback.

unique digit representations (i.e. number of useful minicolumns) developed by the top-Level hypercolumn as it is exposed to more and more digit variations. In this figure, it is seen that the top level hypercolumn contains around 72 minicolumns that have learned to recognize various digit variations present in the training dataset. 28 digit variations are pooled with other variation of the same digit due to spatial similarities and due to the invariance provided by the log-polar transform. Clearly, this experiment demonstrates that the competitive nature of the feedforward processing paths helps the hierarchical hypercolumn network to extract independent features from the input dataset. Furthermore, hypercolumns in each of the hierarchical levels automatically generates an abstract representation of features being exposed to them. Thus, a hierarchical abstract representation of the input space is automatically created by the network.

This experiment also highlights the need for mechanisms that allow the network to pool

variations of the same pattern to develop a robust invariant representation for each unique pattern in the training set. The hypercolumn network has the ability to learn independent features in a unsupervised manner. Such a network that uses an unsupervised mechanism which relies solely on spontaneous activations of the minicolumns and the feedforward processing paths clearly lacks the ability to develop robust invariant representation. The top level hypercolumn should essentially contain 10 minicolumns each recognizing digits 0 through 9. Developing topological feature maps along with feedback based pooling operations allow the hierarchical hypercolumn network to overcome these shortcomings and develop robust invariant representation using a modest number of training samples. Experiments described in Sections 4.4.5 and 4.4.6 validate this hypothesis.

4.4.3 Experiment 3: Hierarchical Feature Maps

The aim of this experiment is to study the hierarchical hypercolumn network's ability to develop hierarchical feature maps to represent variations of different digits in a topological manner. This experiment also verifies the hypothesis object permanence along with that spatially localized and temporally correlated spontaneous activations of the minicolums lend the hierarchical hypercolumn network the ability to learn such hierarchical feature maps in a unsupervised manner. For this experiment, each hypercolumn within our cortical network is initialized with 200 minicolumns. To study the formation of hierarchical feature maps in our cortical network, the hierarchical network is trained with 100 variations of digits 0 and 2. Once the network is fully trained, i.e. 100% recognition rate is obtained for the training

set, the hypercolumn network is tested with 1,000 test images of digits 0 and 2. During the testing phase, the minicolumns which exhibit any response to any variation of 0 or 2 are tracked. Figure 4.12 presents the results of this experiment. In this figure, the topological organization of four sampled hypercolumns from Level-1 through Level-3, as well as the single hypercolumn in Level 4 is shown. The plots on the left show the normalized firing rate of each minicolumn within the observed hypercolumns when variations of the digit 0 are presented to the network; the plot on the right shows the normalized firing rates when the digit 2 is presented to the network. First, a clear topological segregation of minicolumns when comparing the figure on the left (responses to digit 0) and the figure on the right (responses to digit 2) can be seen. Second, the different variations of each of the digits (0 or 2) are learned by spatially localized minicolumns within each of the hypercolumns. Third, at the lower hierarchical levels, various features are shared between 0 and 2 but the segregation between these features increases in the upper hierarchical levels. Overlapping minicolumns in the lower levels are simply a result of features common to variations of both the digits.

In addition to the log-polar transform, such a hierarchical organization where neighboring minicolumns learn to recognize similar features provides another level of object invariance. As the minicolumns in the upper hierarchical levels learn to respond to the stable activations of lower level minicolums, they connect to the lower level minicolumns with a Gaussian-like connectivity. Both the topological organization of minicolumns with similar feature preferences and the Gaussian-like connectivity allow the hierarchical hypercolumn network to develop robust invariant representations. Experiment describes in Section 4.4.4 verifies

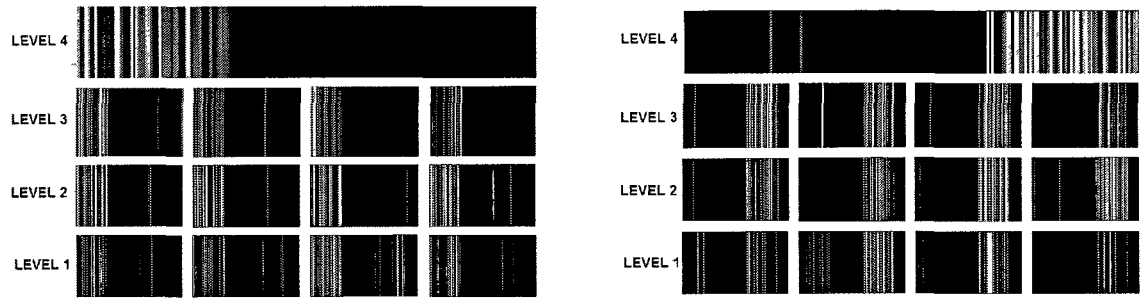

Figure 4.12: Topographical maps obtained for digits 0 (left) and 2 (right) in hypercolumns sampled from the hierarchical levels in the cortical network. We see the localized spontaneous activations and object permanence have lead to digit features becoming localized and segregated.

this hypothesis. Furthermore, the proposed mechanisms that use spatially localized and temporally correlated spontaneous activations for generating hierarchical feature maps also provide us with a theory on how orientation maps in the primary visual cortex might evolve.

4.4.4 Experiment 4: Feedforward Gaussian Connectivity

This experiment verifies the hypothesis that feedforward Gaussian connectivity among the hierarchically organized hypercolumns and the feedback based modulation of widths of such a connectivity allows the hypercolumn network to generate robust invariant representations. For this experiment, the five level hypercolumn network is initialize with 500 minicolumns within each of the hypercolumns. The hypercolumn network is trained with 50 variations of each digit (500 images total) and tested with the full 10,000 test images in the MNIST database. The first phase of the training alters synaptic weights until a recognition rate of 100% is achieved on the training set. In the second phase, Gaussian connectivity modulations (See Section 4.3.1.2) are enabled from Level-4 (the top level) to the feature maps of Level-3,

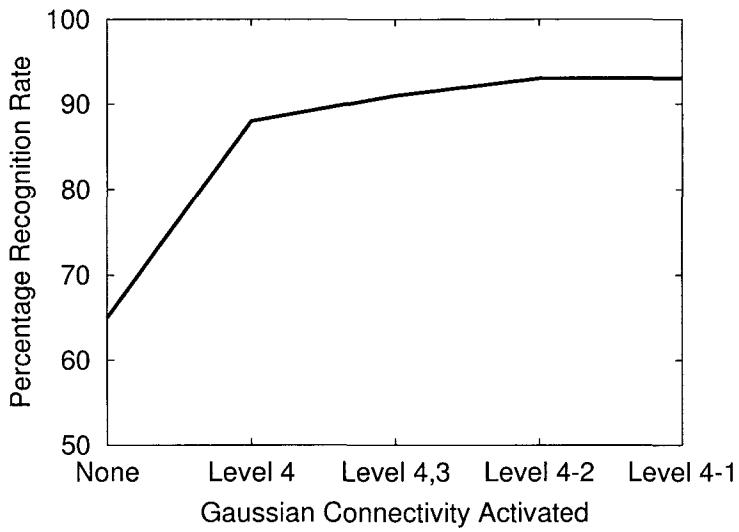

Figure 4.13: Improvement in the overall recognition rate as Gaussian connectivity is applied at different levels in cortical network with 500 training images.

and average recognition rate is measured. Afterwards, each subsequent Gaussian connectivity is enabled between the layers of the network, and the average recognition rate is evaluated. During the second phase, the Gaussian connectivity is updated via Algorithm 3 using the same 500 images in the training set.

Figure 4.13 shows the result of this experiment. Without the Gaussian connectivity update algorithm ('None' in Figure 4.13), the cortical network achieves a recognition rate of 65% on the test set. As the Gaussian connectivity is updated via Algorithm 3, the recognition rate improves. When the Gaussian connectivity is updated from Level-4 to the Level-3 feature maps, the recognition rate improves to 83%. After the second phase of training has updated the Gaussian connectivity of the minicolumns in Level-3, 2, and 1, the recognition rate improves to 88%, 91%, and 94% respectively. The improvement in performance directly shows the ability of our network to create generalized and invariant representations of the

Level	Average Gaussian Width (σ)	Average Connectivity (# of Minicolumns)
4	185	42
3	105	23
2	20	10
1	3	4
0	2.5	4

Table 4.2: Average Gaussian Widths obtained using Algorithm 3 for different levels in the cortical network. Connections between minicolumns in Level_i and minicolumns in Level_{i-1} are modulated by the strength of the Gaussian. Average Connectivity defines the minicolumns within one standard deviation of all the zero mean Gaussians.

learned images, as provided by the topological organization with Gaussian connectivity.

Table 4.2 shows the widths of Gaussian connectivity obtained by applying Algorithm 3 to the different levels in our cortical network. In this table, *Average Gaussian Width* defines the width of the Gaussian for feedforward connectivity determined during the second phase of training. From the table, it can be seen that minicolumns at the higher levels have wide Gaussian connectivity, but in the lower levels of the hierarchy the connectivity narrows progressively. The results in Figure 4.13 show that applying the Gaussian connectivity to the lower levels provides noticeably less performance gains, as many of these lower level features are shared among different digits; hence, the Gaussians are more narrow and less able to generalize to affect recognition. However, it should be noted that this is quite in accordance with the organization and behavior of the biological visual cortex. Higher processing levels of the visual cortex (such as the IT) exhibit a much higher degree of invariance with respect to image recognition. The cortical network shows exactly the same behavior, as exhibited by the drastic improvement in recognition rate when feature maps and Gaussian connectivity were learned for the highest level of the network.

The grouping of minicolumns with similar feature preferences using spatially localized and temporally correlated spontaneous activations and modulation on the feedforward connectivity widths allow the hypercolumn to develop invariant representations without any external intervention. This entire process simply relies on object permanence which has a strong biological basis. Table 4.2 also provides us with a theory that explains why lower cortical regions consists of a significantly larger number of neurons as compared to higher cortical regions. In order to develop robust generalized representations, neurons in higher cortical regions connect with large number of lower level neurons. However, a large number of neurons in the lower cortical regions are required since these neurons respond to very specific features (narrow Gaussian widths). At each cortical level, the width of the Gaussian connectivity are modulated and adjusted to create some notion of generalized representation.

The widths of the Gaussian connectivity can also be thought of as the vigilance of the hypercolumn. This idea can further be extended to model attention within a hypercolumn network. If a certain hypercolumn intends to pay more attention to a certain input, it can simple decrease its feedforward Gaussian width. As a results, this hypercolumn can extract more specific features from the input exciting its receptive field. Modeling such an ability within the model proposed in this dissertation is left as future work.

4.4.5 Experiment 5: Feedback based Pooling to Develop Invariant Object Representations

This experiment validates the hypothesis that a supervised feedback based pooling mechanism allows the hypercolumn network to pool significantly different variations of the same pattern together in order to generate robust invariant representations. To test how such a feedback processing algorithm generates invariant representations, the same hierarchical hypercolumn network described in Table 4.1 is used. For this experiment, each of the hypercolumns is initialized with 100 minicolumns each. For the input dataset, 100 digit images (10 variations for each digit) are used for training. The network is trained with these images till achieved 100% recognition rate is achieved. At this point, only 10 of the 100 minicolumns in the top level hypercolumn maintain a stable representation. This representation corresponds to each of the unique digits (0-9) in the training dataset. This means the feedback pooling algorithm has merged together all the different variations of the same digit and there is just one minicolumn that responds to all of the different variations of the same digit. To estimate the resource optimizations achieved by the feedback based pooling algorithm, the number of active minicolumns throughout the hierarchical network is calculated with and without the feedback. In steady state, without feedback based pooling, the network used 3876 minicolumns while with feedback it only used 1283 minicolumns. Thus, the proposed feedback processing algorithm results in about 3x resource optimization. Figure 4.14 shows the invariant representations for different digits developed by the hierarchical hypercolumn network using the feedback based pooling and unpooling mechanisms.

Figure 4.14: Invariant representations of different digits developed throughout the hierarchical hypercolumn network.

Generation of robust invariant representation of unique patterns is one of the most powerful abilities of the neocortex. The experiment described in this section demonstrates one such mechanism that the neocortex might utilize to generate these invariant representations. It should be noted that the feedback based pooling operation described here is localized within a hypercolumn. As a result, the invariant representation of a pattern is hierarchically distributed throughout the hypercolumn hierarchy. Minicolumns within the hypercolumns at lower hierarchical levels create invariant representations for simple features while the minicolumns in the hypercolumns at the higher hierarchical levels generate invariant representations for complex features that combine several simple features. Finally, at the top level, minicolumns generate invariant representations for various objects that may exist in the surroundings. This feedback based pooling mechanism must be accompanied with an unpooling mechanism (see Section 4.3.2.3) in order to separate exceptions from the generalized invariant representation.

The role of feedback paths within the neocortex in terms of learning and generation of invariant representations is not well-known. The results of this experiment allow us to hypothesize that such supervised feedback based pooling and unpooling mechanisms might

play an important role during the early years of a mammalian brain in order to develop robust invariant representations of various patterns that exist in the surrounding of the mammal. This supervised feedback can be thought of as a teaching signal that allows the neocortex to pool variations of the same animal e.g. different breeds of horses together and distinguish them from other similar looking animals like a donkey.

4.4.6 Experiment 6: Comparison with Conventional Neural Networks

In this final experiment, the ability of the proposed hierarchical hypercolumn network to generate invariant representations is compared against two contemporary state-of-the-art traditional neural network models. These include open-source implementations of a convolutional neural network (CNN) [101] and hierarchical temporal memory (HTM) [64]. For this experiment, the hypercolumn network described in Table 4.1 is initialize with 500 minicolumns in each hypercolumn. The hypercolumn network, the CNN, and the HTM are trained using only a small subset of the MNIST training images, and are tested using the entire MNIST test set that consists of 10,000 handwritten test images. Figure 4.15 clearly demonstrates that the proposed hierarchical hypercolumn network is able to outperform the CNN and HTM in terms of quickly generating invariant representations of each digit. While both the CNN and the HTM show a slightly higher recognition rate after training with only 100 images (65% and 66% respectively), the cortical network quickly overtakes the performance of both the CNN and HTM in terms of recognition rate and generalization of

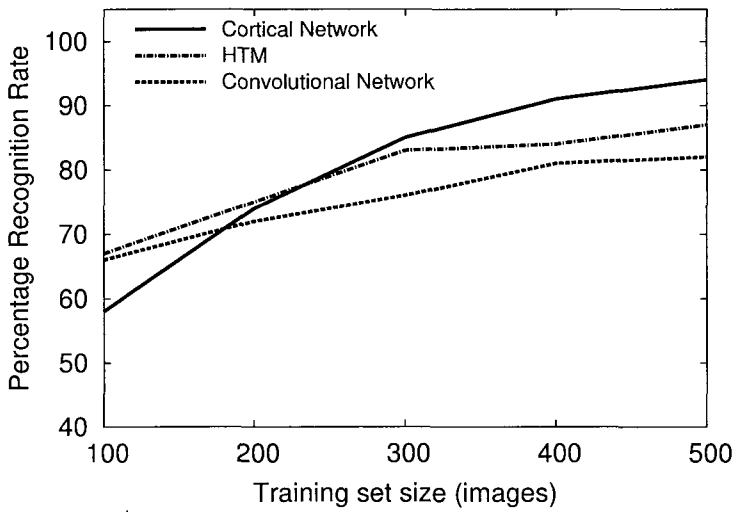

Figure 4.15: Recognition rate comparison between the proposed hierarchical hypercolumn network and fine-tuned implementations of CNN and HTM.

the input patterns. With 50 variations of each digit in the training set, our cortical network achieves a recognition rate of 94% while the CNN and HTM achieve a recognition rate of 84% and 87% respectively.

This experiment highlights the fact that utilizing all the biologically inspired aspects described in this chapter, the hypercolumn network is able to generate robust invariant representations for each of the digits in the training set. Further, the hypercolumn network is able to develop these invariant representations using a very modest number of training samples. The state-of-the-art neural network implementations which are fine-tuned specifically to perform optimally on the MNIST database generate invariant representations at a much slower rate as compared to the hypercolumn network. It should also be noted that hypercolumn network does not rely on fine tuning of any hyper-parameters. All the parameters like the size of the topological groupings, widths of the feedforward Gaussian-like connectivity, pooling

and unpooling parameters, etc. that determine the behavior of the hypercolumn network are learned by the network based on the type of type of data that it is exposed to. Unlike CNN and HTM implementation, which need to be fine tuned every time the type of the training data i.e. digits, faces, natural images, etc. changes, the hypercolumn network does not require fine tuning. The hypercolumn network learns to extract independent features from any type of input data exposed to it and to generate corresponding invariant representations using various feedforward and feedback processing operations discussed in this chapter.

4.4.7 Summary of Experimental Results

In this section, various experiments are used to validate the hypothesis proposed in this chapter. These experiments also evaluate the hypercolumn network in terms of learning independent features and generating robust invariant representations. Section 4.4.1 validates the hypothesis that the log-polar transforms plays a key role in providing invariance to rotation and scale variations to a small extent in early vision processing stages. Results discussed in this section show that log-polar transform helps the hypercolumn network develop invariance to minor variations in rotation and scale of the digit images. Section 4.4.2 verifies the hypothesis that the spontaneous activations ability of the minicolumns along with strong lateral inhibitory connections among the minicolumns within a hypercolumn provides an unsupervised mechanism for independent features identification and automatic abstraction. This section shows that the hypercolumn network is able to learn different digit variations in an unsupervised manner. Section 4.4.3 studies the hierarchical hypercolumn network's ability

to develop hierarchical feature maps to represent variations of different digits in a topological manner. Results provided in this section show that the hypercolumn network is able to effectively segregate features corresponding to digit 0 from the features corresponding to digit 2 by organizing features with temporal associations in close vicinity. Section 4.4.4 verifies the hypothesis that feedforward Gaussian connectivity among the hierarchically organized hypercolumns and the feedback based modulation of widths of such a connectivity allows the hypercolumn network to generate robust invariant representations. Results described in this section demonstrate that by allowing the network to learn the Gaussian connectivity widths, average recognition rate of the network is significantly improved. Section 4.4.5 validates the hypothesis that a supervised feedback based pooling mechanism allows the hypercolumn network to pool significantly different variations of the same pattern together in order to generate robust invariant representations. Finally, Section 4.4.6 compares the ability of the proposed hierarchical hypercolumn network to generate invariant representations against two contemporary state-of-the-art traditional neural network models. This section shows that the hypercolumn network outperforms tradition state-of-the-art neural network approaches in terms of generating robust invariant representations using a modest number of training samples.

4.5 Summary

This chapter provides a detailed description of the proposed learning model which inspired by various structural and functional properties of cortical columns. The main goal of this

chapter is to show that all the features incorporated in the proposed learning model are biologically inspired. This chapter also describes the role of feedforward, feedback, and lateral communications paths in terms of learning independent features, developing feature maps, and constructing robust invariant representations. A lot of emphasis is put on the powerful role of feedback communication paths within the proposed learning model. Finally, through a series of experimental studies this chapter highlights how the proposed learning model is more attractive than the contemporary traditional neural network approaches.

5 ARCHITECTURAL SOLUTIONS AND OPPORTUNITIES

This chapter serves to establish a bidirectional relationship between conventional computer architecture and the proposed biologically inspired model. On the one hand, this chapter describes various existing architectural solutions that can be applied to improve the performance of a complex biological models. On the other hand, it also highlights biological properties that can be helpful to contemporary architectural and processing models.

5.1 Leveraging Architectural Tools to Optimize Biological Networks

This section provides an overview of several architectural tools that can benefit the construction and working of complex biologically inspired networks. In this effort, first, this dissertation highlights the need for a unified Neuromorphic Instruction Set Architecture (NISA) to represent complex biological networks. Second, this section describes various optimizations that are build on top of the notion of NISA to improve a complex biological network's structure as well as to reduce its overall execution time.

5.1.1 A Unified Neuromorphic Instruction Set Architecture

In the recent years, a number of neuromorphic architectures, with the eventual goal of implementing brain-like devices, have been proposed. Some of these include the IBM's neurosynaptic chip [90], the FACETS hardware [1], and the SpiNNaker project [2]. At the

same time, a broad variety of neural algorithms has been proposed e.g. traditional neural networks, spiking neuron models (See Chapter 3), and the proposed hypercolumn model. Each of these algorithms uses very different primitives to implement complex networks. In order to run one of these neural algorithm on the aforementioned neural architectures, it has to be reimplemented using an application programming interface (API) provided by each of the specific hardware substrates. This approach strongly ties the neural algorithm to the underlying execution hardware. As a results, this neural algorithm cannot be ported to other neural hardwares in a straightforward manner. Also, this excessive dependence between the neural algorithm and hardware requires that any change in the underlying hardware be reflected in the neural algorithm as well.

This issue can be solved by borrowing a widely accepted idea from the computer architecture domain: Instruction Set Architecture (ISA). An ISA defines an interface between a hardware and software and by doing so it separates a conventional algorithm from the execution substrate. This means that the algorithm developer does not have to worry about the details of the execution hardware. Any changes in the hardware be incorporated into the ISA and no modifications are required on the algorithm development side. The same ISA concepts can be applied to separate the neural algorithms from the neuromorphic hardware (See Figure 5.1). This means that the neural algorithms are implemented using an invariant representation that this dissertation refers to as a neuromorphic ISA. This neuromorphic ISA defines an interface between the neural algorithm and the neuromorphic hardware. As a result, the neural algorithm developer does not need to worry about the details of the underlying

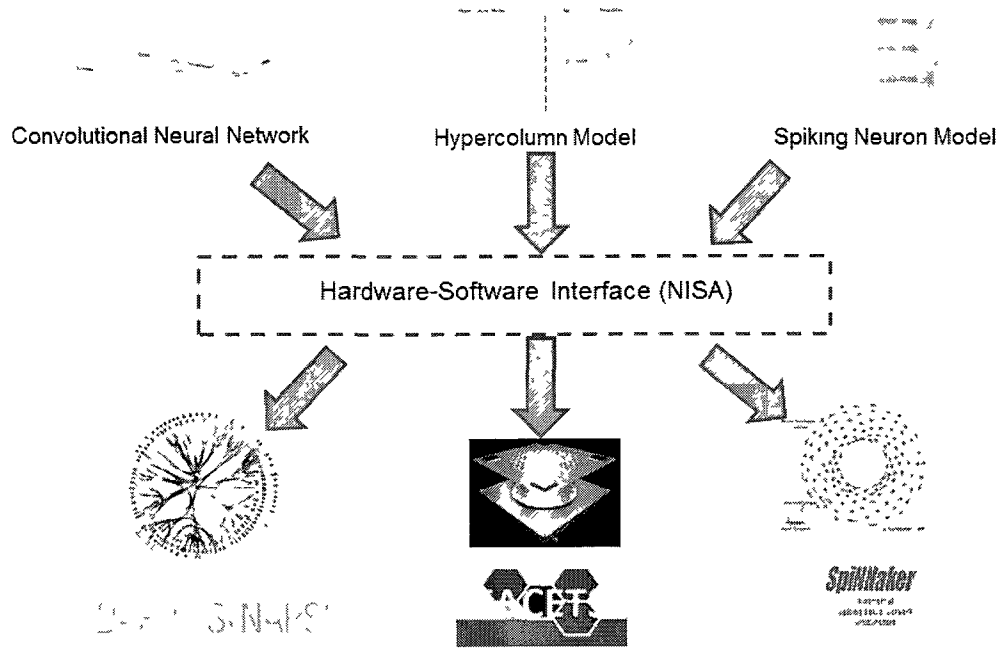

Figure 5.1: The Neuromorphic ISA defines an interface between the neural algorithm and the execution substrate.

hardware and any neural algorithm can run on any of the aforementioned neuromorphic architectures without any modifications as long as it conforms to a well-defined NISA.

A NISA abstraction model has been developed for the hypercolumn model proposed in this dissertation. Presently, the NISA based implementation of the hypercolumn model runs on multi-threaded CPUs, GPUs, and can also be ported to custom designed ASICS [46]. The details of this NISA are out of the scope of this dissertation, however, major contributions of Hashmi et al. [46] are summarized below. This dissertation describes various optimizations that are built on top of the developed NISA for the proposed hypercolumn model.

In [46] Hashmi et al. present the *Aivo*[1] framework. This framework provides a well-

[1] *Aivo* is the Finnish word for Brain

defined NISA to precisely specify state, structure, and semantics for a hypercolumn network described in Chapter 4. This NISA also separates the hypercolumn learning algorithm from the execution substrate. By separating the hypercolumn learning algorithm from the deployment substrate, each can be developed independently without one placing restrictions or limitations on the other. The Aivo NISA abstraction allows a complex hypercolumn hierarchy to be deployed on CPUs, GPGPUs, and custom designed ASICs. It also provides an integrated development environment (IDE) that simplifies the task of developing, initializing, and debugging complex hypercolumn networks. The Aivo framework provides the ability to profile complex hypercolumn network as well. The profiling information obtained from the Aivo framework can help optimize and restructure the hypercolumn network for improved robustness and reduced execution time.

5.1.2 Hypercolumn Network Optimizations

This section describes one of the optimizations implemented on top of the NISA abstraction described in Section 5.1.1. The hypercolumn network optimizer is a high level optimization tool developed to improve the structure of the cortical network, either by expanding the structure to improve learning robustness, or reducing the structure to reduce required processing time. When a user initially creates a hypercolumn network, the optimal number of resources (hypercolumns and minicolumns) required to achieve a particular task (e.g. robust recognition of the entire feature set) is usually not known. Thus, it is likely that resources have been either over allocated or under allocated for the network to learn a particular task.

When the resources are over allocated, the cortical network requires more computation than is necessary for each learning iteration. On the other hand, an under-allocated cortical network may not contain enough minicolumns to learn the full number of features in the dataset.

In the case when the cortical network resources are over allocated, all of the unique features of the dataset will be fully recognized after a sufficient amount of learning epochs. However, there may be a number of minicolumns that will not perform useful work, even though they must still evaluate at each learning iteration. While the evaluation of these minicolumns does not affect the activations propagated to the next level of the network, they still contribute to the total execution time. In such cases, the network optimizer can be invoked to perform network pruning and remove unnecessary minicolumns. First, the state and structure of the trained cortical network is exported using the NISA in the form of XML files. Then, the network optimizer parses these files and deletes the unnecessary minicolumns. A minicolumn is declared unnecessary if all its synaptic weights are close to zero, which suggests that it has not learned any interesting features from the training dataset; thus it is not required to robustly perform the learned task. Along with pruning unnecessary minicolumns, the network optimizer will regenerate the net-list that defines the connections between the minicolumns at various levels in the hierarchy to account for the deleted minicolumns and their connections.

Conversely, an under-allocated cortical network may not possess enough minicolumns to robustly recognize all of the features of the dataset. After a significant amount of learning epochs, the network optimizer may be invoked to perform a robustness expansion of the cortical

network. The trained network's state is again exported in the NISA format, which the network optimizer parses, and then allocates more resources for under-allocated hypercolumns, as determined by a defined threshold (i.e. if 90% of minicolumns within a hypercolumns are doing useful computations, network optimizer allocates more minicolumns to this hypercolumn). Minicolumns are useful if they contain strong weights corresponding to the input activations. This feature is quite useful because the network optimizer adds minicolumns only to the necessary hypercolumns. Thus, using multiple invocations, hypercolumn network optimizer generates a cortical network that is sufficient in terms of resource allocation and execution time for the given input dataset. Figure 5.2 provides a pictorial representation of the hypercolumn network optimizer.

An offline optimization approach is used for the network optimizations rather than performing such major structural changes to cortical network during runtime. Such offline structural changes can be considered biologically inspired as well, since there is evidence that memory-consolidations and translations that occur in the brain during sleep have a significant impact on how and where memories are stored [88]. Generally, changing the cortical network structure during runtime results in both code complications and a drastic increase in the execution time for each learning epoch. Furthermore, the structure does not significantly change on an iteration by iteration basis, so it makes sense to optimize after a large number of training epochs.

In the future, further optimizations can also be introduced in the hypercolumn network optimizer. These optimizations may include flattening of different hierarchical layers, adding new

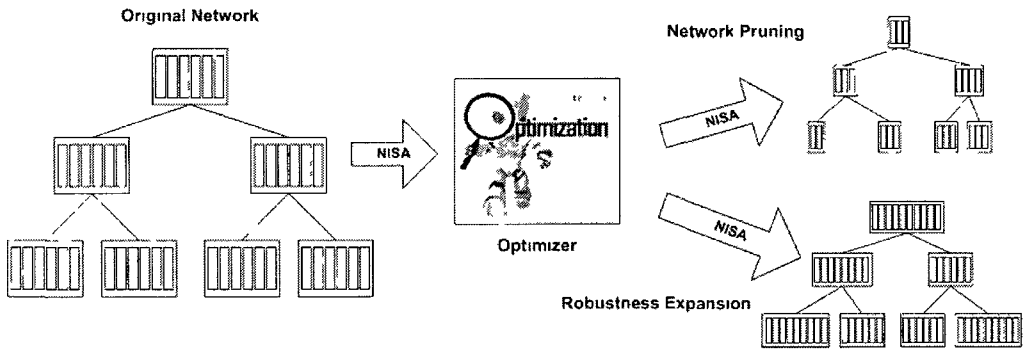

Figure 5.2: The Hypercolumn Network Optimizer optimizes a trained cortical network for resources utilization or for robustness.

hypercolumn layers within the hypercolumn hierarchy, generating long distance connections, etc. In terms of the human brain, the response time to perform a task improves with training. One of the hypotheses explaining such a behavior is that with training, intermediate cortical neurons are bypassed and information from sensory modalities is directly communicated to higher cortical regions. Such a mechanism can also be implemented using the offline hypercolumn network optimizer. After sufficient training, the network optimizer can restructure the network. Lower level hypercolumns can bypass intermediate level hypercolumns and connect directly to upper level hypercolumns. Over time, the intermediate hypercolumns can be completely pruned.

5.1.3 Hypercolumn Networks to Functional Boolean Logic Conversion

Another feature built on top of the NISA abstraction is the ability to convert a fully trained hypercolumn network into an equivalent functional logic circuit. This optimization utilizes

the observation that once the biological neurons are in a stable state i.e. they learn to recognize specific features, their synapses demonstrate an all or none response [105], i.e. they can be treated as binary synapses. The NISA abstraction supports deployment of a hypercolumn network in the form of logic functions that can be converted to netlists. AND- and OR-operations are hierarchically connected to represent the structure of the corresponding network. Once a network is fully trained (i.e. 100% recognition rate on the training dataset), it can be converted to a logic representation for efficient execution. To achieve this, the state of the hypercolumns and minicolumns of the fully trained network is exported into a NISA representation. The NISA representation is then processed off-line to generate the equivalent functional logic representation of the cortical network.

Once a minicolumn has concretely learned a particular feature, its weights can be considered as binary synapses, i.e. it has a strong synaptic connection or no synaptic connection to a particular input. In terms of boolean logic, the output 'Y' of such a minicolumn can be represented as:

$$Y_i = \forall_{k \in S} \mathrm{AND}(k)$$

Here, 'S' is the set of inputs to the minicolumn corresponding to high weights.

If a minicolumn has pooled different variations of an input as described in Section 4.3.2.2, then its output can be represented as:

$$Y_i = \forall_j \mathrm{AND}(O_j)$$

$$O_j = \forall_{k \in M} OR(k)$$

Here, 'M' is the set of inputs corresponding to high weights that pool different variations of the same pattern.

Figure 5.3 illustrates the logic generation process using a simple trained cortical network as an example. In this example, the two levels of hypercolumns are replaced with logic equations that perform the equivalent detection or classification function. However, since the LGN cells in this circuit perform a type of analog-to-digital conversion of the input image, they are not simplified to boolean logic.

Even though converting the cortical network to boolean representation results in significant reduction in execution time for a learned task, it comes with a trade-off: this boolean network cannot learn new tasks or features. Rather, it can only detect the features it has already learned, and will not respond to new features appearing in the input. To address this shortcoming, a runtime monitor is used to detect when a boolean logic equivalent network is not sufficient for the learning task at hand. This runtime monitor relies on a simple property of the competitive learning-based cortical column model: a fully-trained cortical network should evoke a single winning response for every input (i.e. one minicolumn in each hypercolumn should fire). The runtime system monitors the firing rate of each boolean circuit hypercolumn, and once it falls below a given threshold, runtime monitor reverts the boolean circuit back to a computational model that is able to learn the new features in the input. After the cortical network learns the new features, a new boolean logic circuit can be regenerated to obtain execution efficiency. This process is very similar to profile-driven

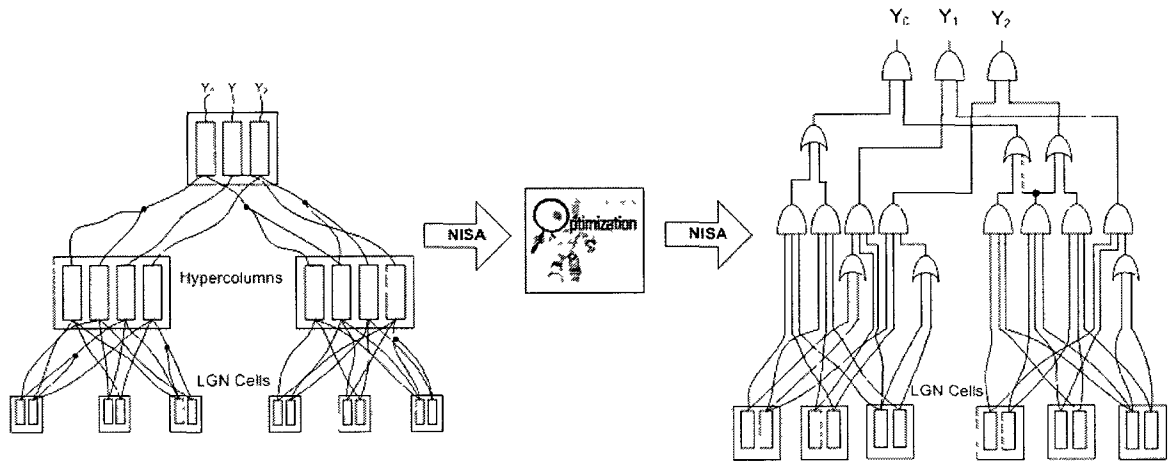

Figure 5.3: A simple fully trained cortical network and corresponding logic network.

re-optimization of machine code in managed runtime systems with just-in-time compilers (e.g. Java, C#).

5.1.4 Code/Logic Hybrid Generation

The NISA abstraction also supports generation of cortical networks using a code/logic hybrid approach. As described in Section 5.1.3, once a network is fully converted to logic, it is unable to learn new features. Therefore, the cortical network should achieve 100% recognition rate on the training dataset before it may be converted into an equivalent functional logic representation. This means that during the training period, the cortical network cannot benefit from the logic generation capability of the NISA abstraction, as recognition has not stabilized. To avoid this dilemma, the NISA abstraction is extended to allow cortical networks to be partially converted as they stabilize, which is referred to as Code/Logic hybrid networks. This addition lets the NISA abstraction to partially convert a cortical network into

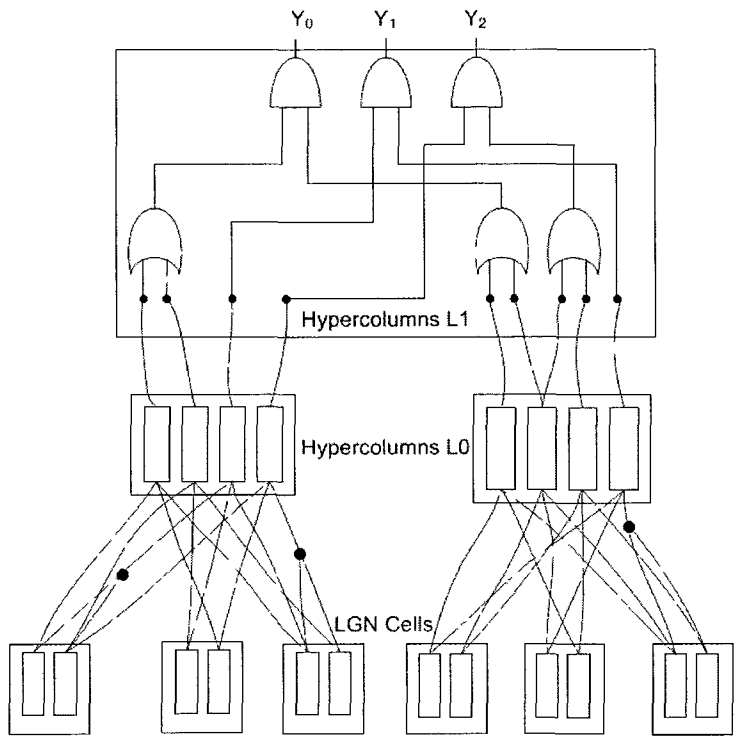

Figure 5.4: An example of a hybrid network created using the NISA abstraction.

logic (i.e. some of the hypercolumns are converted into their functional logic representation while others are not). To achieve this, the NISA abstraction can store the state activity of the hypercolumns in the network. If after a significant number of training epochs no new minicolumns within a hypercolumn learn to recognize any new features, the hypercolumn may be considered stabilized and can be safely converted to a functional logic representation.

The NISA abstraction also allows the programmer to explicitly control the conversion of a hypercolumns to a logic function (i.e. the programmer can mark certain hypercolumns so that they are not converted to logic). For example, the program may configure an explicit hybrid network where the upper levels of the hierarchy are converted to logic functions while the hypercolumns in the lower levels are not. Typically, this type of hybrid conversion is

useful for robust recognition of input patterns in the presence of noise. Since the lower levels execute the hypercolumn learning algorithm, they exhibit more resilience to noise or slight variations present in the inputs while the logic converted upper level hypercolumns identify complex objects with computational efficiency. Figure 5.4 demonstrates this hybrid approach.

In the future, we plan to extend the boolean logic circuit generating mechanism so that it can generate hypercolumn equivalent circuits for any of the execution substrates described in Section 5.1.1.

5.2 Leveraging Biological Behaviors to Improve Fault Tolerance

This section describes how the biologically inspired spontaneous activity property of the proposed hypercolumn model helps the model achieve tolerance to permanent hardware defects. Permanent defects and transient faults are not only a concern for future architectures, but are already a prevalent issue in some of the latest systems. For instance, the NVIDIA Fermi is the first Graphical Processing Unit (GPU) architecture to provide SECDED error correcting code for all DRAMs, caches and registers [100]. Permanent defects, at design time or during the chip lifetime, are also expected to further increase in the future.

Currently, applications programmed for GPU chips like Fermi or Tesla assume that all cores of the GPU function correctly. If any of the 512 shaders (cores) of a Fermi chip becomes dysfunctional (increasingly likely as the number of cores increases), it will be necessary to

rewrite applications so that no task is mapped to faulty shaders, or the compiler would have to perform that remapping automatically. Here, it is assumed that the programmer or compiler is given explicit control over the shader mapping, though no such ability is yet present in current generation GPGPUs. For instance, all existing vision recognition applications written for GPUs will have to go through that reprogramming or recompilation process to execute correctly on the defective GPU. Even vision recognition applications based on artificial neural networks will suffer from the same limitation. While ANNs are inspired by biological neural networks, neither their back-propagation learning process nor their software implementation as imperative array-based computations are defect tolerant.

The software implementation of the cortical column based learning model proposed in this dissertation for the GeForce 9800 GT GPU preserves the key concepts of the model: the connections (synaptic weights) are initialized with weak random values, operators (sum, max) are implemented in a robust manner through a set of synapses, there is no central control nor supervision for the learning process since it happens in a distributed manner, yet it can implement complex tasks such as vision recognition. Thanks to these properties, the software implementation of the learning model proposed in this dissertation is inherently tolerant of faulty GPU hardware. It can function properly and thus, unlike most other applications, it can take advantage of the GPU, *w*ithout requiring any reprogramming or recompilation, even if one or several of the cores is dysfunctional and has to be deactivated. All that is needed is to periodically retrain the application so that it adapts to the new configuration of the faulty hardware, but again without specifying that configuration; the learning process will

automatically adjust to the faulty hardware.

5.2.1 Model Implementation on GPUs

While the model described in this dissertation may eventually be realized with specialized hardware, fitting a software version of that model on a currently available architecture is also investigated. The most attractive architecture encountered so far is the general purpose graphics processing unit (GPGPU), specifically NVIDIA's CUDA. In this programming model, highly parallel workloads can be processed on hundreds to thousands of CUDA threads. In current top-end CUDA devices, groups of threads are scheduled to run on a streaming multiprocessor (SM) which is composed of eight in-order cores and 16KB of fast-access shared memory [23]. CUDA makes it easy for programmers to optimize their applications through a number of different methods, including memory access coalescing and using the shared memory space as a fast-access user-managed cache [114].

Nere et al. [98] have successfully demonstrated that the proposed hypercolumn model maps well onto the CUDA architecture. In their implementation, each of the hypercolumn maps onto a SM and each minicolumn maps onto a CUDA thread. By mapping a single minicolumn to a CUDA thread, thousands of minicolumns can be concurrently active on a GPGPU. Since the minicolumn's firing is based on the dot-product evaluation of the input and minicolumn weights, the model is an example of a high-throughput data-intensive application CUDA was invented for. Finally, the shared memory space per SM is ideal for fast lateral communication between neighboring minicolumns.

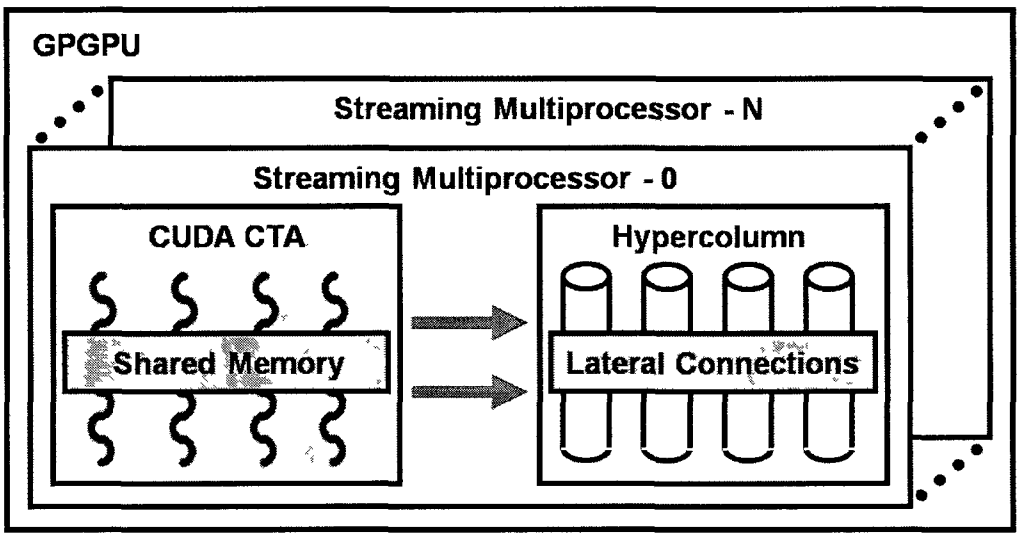

Figure 5.5: Mapping a hypercolumn to a CUDA CTA. [46]

Section 4.3 describes the proposed cortical architecture as having different hierarchically organized components, composed of minicolumns and hypercolumns. Similarly, NVIDIA's CUDA framework consists of a hierarchical organization, with threads, cooperative thread arrays (CTAs), and kernel launches. The GPU-accelerated code translates the components of the cortical architecture to the CUDA framework. With such an organization on CUDA, the minicolumns in a hypercolumn can easily synchronize as well as laterally communicate and share receptive field inputs in the fast access shared memory space, as seen in Figure 5.5.

5.2.2 Fault Identification and Detection Model

In the case of a biological network (the neocortex), the fault model is quite simple: if neurons or minicolumns become defective, they stop generating any activations and eventually die out. When this type of fault occurs, other neurons/ minicolumns modify their synaptic weights

to detect and interpret the feature that was previously recognized by the damaged neuron/ minicolumn.

In the software implementation of the proposed hypercolumn model, each minicolumn runs on a shader core, which can have multiple failure modes. Ultimately, due to the sigmoid nature of the output activation functions these faults will manifest themselves as a minicolumn either not firing when it should (a stuck-at-zero fault), or firing when it should not (a stuck-at-one fault).

A minicolumn stuck-at-zero behaves the same as a damaged neuron/ minicolumn in a biological network: since it is not generating any activity, its functionality is automatically taken over by the neighboring minicolumns. On the other hand, any minicolumn that is stuck-at-one can severely impact the performance of the hypercolumn network, since it will inhibit its peer minicolumns continuously, effectively masking all feedforward information that passes through that point in the network. Biological neurons in such a mode would exhaust their resources and eventually cease activity, allowing others to learn and assume their role in the network. Since the synthetic minicolumns proposed in this dissertation do not have built-in resource limits on their behavior, the resilient fall-back is mimicked by intermittently recomputing the response of the winning minicolumns on two neighboring shaders, and disabling shaders that exhibit stuck-at-one behavior. A shader is disabled if two of its neighbors disagree with its result. In subsequent iterations of the model, all neighboring minicolumns (in the same hypercolumn) as well as upstream minicolumns in the next level of the hierarchy ignore the output of the defective minicolumn.

At this point, the proposed learning model utilizes the idea of automatic abstraction and random firing to relearn the features being recognized by the minicolumns running on the defected shader core. It should be noted that since all the minicolumns within a hypercolumn share the same receptive field, minicolumns connected to defected minicolumns need not be reconnected as their output is already being exposed to multiple minicolumns at the upper level in the hierarchy.

5.3 Experimental Results

This section describes various experimental studies demonstrating the bidirectional benefits obtained by combining biological network ideas and conventional computer architecture concepts. Section 5.3.1 verifies the hypothesis that restructuring a complex hypercolumn network through hypercolumn network optimizer can result in significant gains in terms of execution time per epoch as well as the resources utilized by the trained network. The experiment discussed in this section shows that using the hypercolumn network optimizer, a complex hypercolumn network executes significantly faster than the unoptimized network and requires less resources when fully trained. Section 5.3.2 verifies the hypothesis that a fully trained hierarchical hypercolumn network can be converted into an equivalent boolean logic circuit. It also shows that such a conversion can allow the hypercolumn network to execute orders of magnitude faster than the one not converted to a boolean logic representation. Section 5.3.3 evaluates the ability of the hypercolumn runtime monitoring system to revert a highly optimized boolean logic network to a network of hypercolumns for improved robustness.

The experiment described in this section clearly demonstrates that the boolean logic network can be converted back to a hypercolumn network if the overall recognition rate of the network does not meet a specified threshold. Sections 5.3.4, 5.3.5, and 5.3.6 verify the hypothesis that spontaneous activations of the minicolumns along with the automatic abstraction property lends the hierarchical hypercolumn networks the ability to recover from permanent hardware defects. The results in these sections show that a complex hypercolumn network can fully recover in the presence of permanent hardware failures even in the case when 50% of the hardware is defected. These experiments also study the inherent fault tolerance ability of the hypercolumn network in various permanent defect scenarios that might exist in the future generation computational hardware.

5.3.1 Experiment 1: Hypercolumn Network Optimizations

This experiment validates the improvements achieved in terms of execution time and resource requirements when the structure of a hypercolumn network is optimized using the profiling information provided by the NISA abstraction. In this experiment, four variations of the hypercolumn network described in Table 4.1 are created and are deployed on the CPU. Each of these variations is trained with 64, 128, 256, and 512 images of handwritten digits from the MNIST database respectivley. The network trained with 64 images is initialized with 64 minicolumns per hypercolumn, the network learning 128 images began with 128 minicolumns, and so on. Figure 5.6 shows the execution time of each of the network configurations deployed on the CPU. As can be seen, the execution time grows substantially as the number of

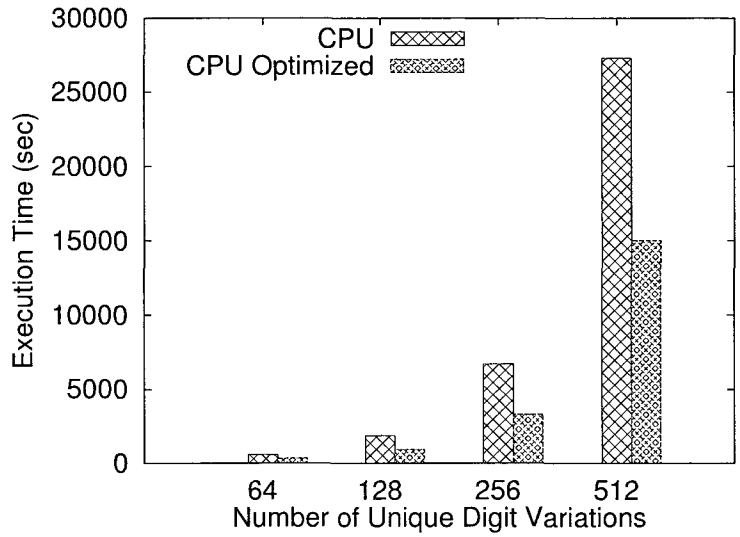

Figure 5.6: Performance of original and optimized networks on CPU.

minicolumns is increased in each hypercolumn. After the hypercolumn network is fully trained and it achieves 100% recognition rate on the training set, the hypercolumn network optimizer is invoked to restructure the network. After hypercolumn network is profiled and optimized, the performance benefit of pruning the unused minicolumns can clearly be seen in Figure 5.6. These optimizations result in nearly 2x speedup for each of the network configurations tested.

Further, the amount of resources recovered by the network optimized is examined in in Figure 5.7. Here, the resources utilization is estimated using the memory footprint of the entire hypercolumn network. As can been seen in Figure 5.7, pruning unnecessary minicolumns significantly reduces the resources required by the hypercolumn network for each of the different minicolumn configurations. On the average, the hypercolumn network optimizer is able to recover approximately 50% of the resources initially allocated to each of

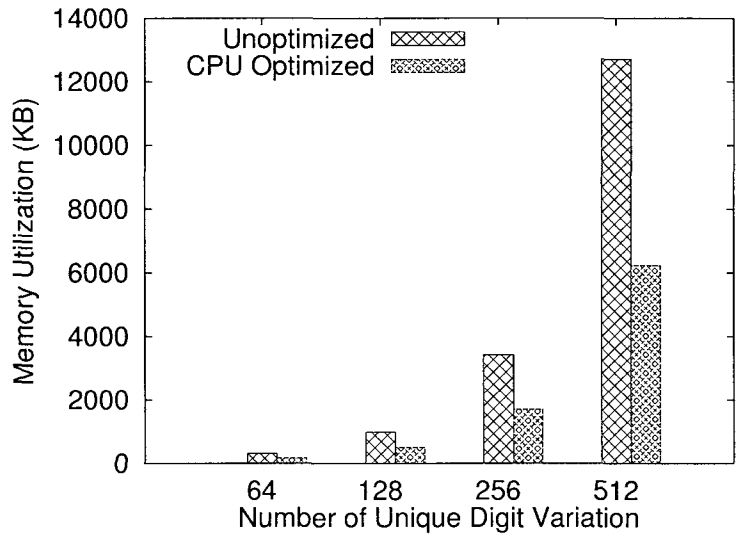

Figure 5.7: Memory footprint of minicolumns before and after optimization.

the hypercolumn network configurations.

Pruning of unnecessary minicolumns improves both the execution time of the network and its resource requirements. These improvements can be attributed to two main reasons. First, if unnecessary minicolumns are reduced, the computational cycles wasted in evaluating the response of those minicolumns as well as the memory allocated to those minicolumns are recovered. Second, when the number of minicolumns is reduced across various hypercolumns, the receptive field size of higher level hypercolumns is reduced. As this receptive field size is reduced, the number of synaptic weights needed for these upper level hypercolumns is in turn minimized as well. This reduces the overall computational and memory requirements of the useful minicolumns as well.

5.3.2 Experiment 2: Speedups Due to Boolean Logic Conversion

In this experiment, the ability of the hypercolumn model proposed in this dissertation to interpret a trained cortical network exported using the NISA abstraction and to translate it into a boolean logic equivalent is evaluated. For this experiment, the hypercolumn network described in Table 4.1 is initialized with 20 minicolumns within each hypercolumn, and is trained with 10 variations of a single digit until 100% recognition rate is achieved. At this point, the hierarchical hypercolumn network is converted into an equivalent boolean logic circuit through the operations described in Section 5.1.3. Once the boolean logic is generated, the equivalent logic circuit is deployed to perform the recognition tasks. For the purposes of this experiment, the boolean logic circuit is deployed using C++ boolean logic constructs on a conventional CPU. Table 5.1 compares the performance of the optimized cortical network deployed on the CPU with the corresponding boolean logic equivalent circuit. From the table it can be seen that using the hypercolumn network to prune unnecesary minicolumns provides a nearly 3x speedup, while deploying an equivalent boolean logic network results in 44x speedup for the hypercolumn network described in Table 4.1.

For this experiment, the working of boolean logic equivalent circuit is emulated using C++ boolean constructs. However, further improvements in the execution time can be achieved if such a boolean logic equivalent circuit of a hypercolumn is deployed on a FPGAs or customized ASICs. Once a hypercolumn network is fully trained and provides acceptable results on the test set, it can be converted into an equivalent boolean logic circuit and then can be deployed on a customized hardware. Such an approach can be quite fruitful for

	Original Network	Optimized Network	Logic Network
Execution Time / Iteration	3000 us	1080 us	68 us
Speedup	1x	2.78x	44.12x

Table 5.1: The NISA abstraction provides performance benefits for equivalent optimized networks and boolean logic conversion.

applications with real time requirements or stringent power or energy limits.

5.3.3 Experiment 3: Online Monitoring of Boolean Logic Network

In this experiment, the ability of the network monitoring system to revert the highly optimized boolean logic network back to a network of hypercolumns when new features are introduced into the learning dataset is evaluated. If the boolean logic network deployed on the CPU does not exhibit any activity for a large number of epochs, network monitoring system detects it as an anomaly and reverts the boolean logic network back to a network of hypercolumns in an effort to learn new features. For this experiment, a similar hierarchy as described in Table 4.1 is constructed and 10 variations of a single digit are exposed to it until 100% recognition rate is achieved. At this point the boolean logic equivalent circuit of the hypercolumn network is generated as C++ boolean logic constructs. Once the optimized boolean logic network for the trained cortical network is generated and deployed, five new variations of the same digit are introduced to the original network and optimized boolean logic network. For the sake of this experiment, both the hypercolumn network and boolean logic translated network are run alongside to observe their corresponding recognition rates.

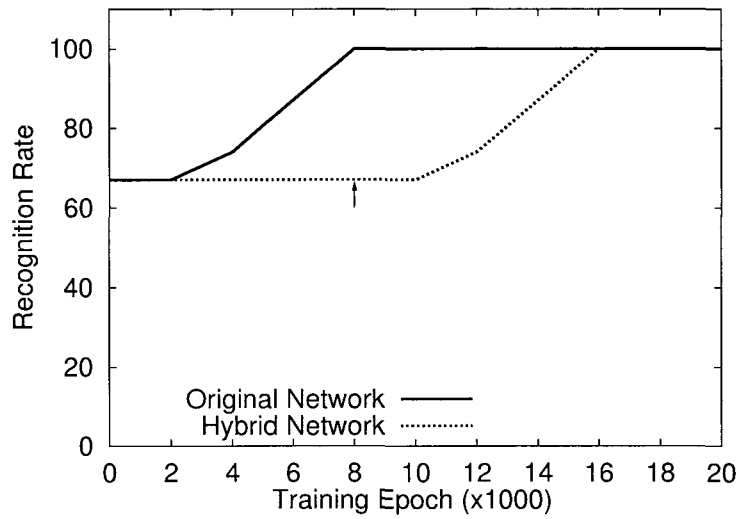

Figure 5.8: Recognition performance of the original cortical network and its logic based equivalent. At 8000 learning epochs (arrow), the boolean logic equivalent network is replaced with a hypercolumn network.

In Figure 5.8, it can be seen that initially the new variations are not recognized by either the hypercolumn network or the equivalent boolean logic circuit. Thus, both the original and the converted logic network exhibit a recognition rate of 67%, i.e. 10 out of 15 digit variations are recognized. Since the original network of hypercolumns has enough resources available to learn new features, after 3000 training epochs, it starts to learn the new variations added to the training dataset. On the other hand, the optimized boolean logic network does not show any improvement in the recognition rate since it lacks the ability to modify its structure in response to the new inputs in the training set. The runtime monitoring system eventually recognizes that the logic network does not show any response to the new variations introduced in the training set. As a result, it translates the boolean logic network back into a trainable hypercolumn network. This hypercolumn network is then redeployed instead of the boolean logic equivalent. Once the logic network is replaced with a hypercolumn network

(at around 8000 epochs in Figure 5.8), improvement in the recognition rate of the hybrid network are observed and at around 14000 epochs this network achieves 100% recognition rate on the training set. At this point, this network can again be converted into a boolean logic equivalent circuit that can be redeployed for improved execution time.

Since the logic conversion mechanism evaluated here is very infrequently invoked, it incurs minimal overheads on the overall performance and execution of a hypercolumn network. However, such mechanisms can play a very important role in developing intelligent systems that can operate in real-time and at the same time are robust enough to adapt to the changing environment. As long as the intelligent system demonstrates acceptable results in terms of recognition, classification, or other complex decision making scenarios, it can stay in the boolean logic state. This allows the system to quickly provide a response. However, if the performance of the system does not meet a predefined criteria, it can be reverted into a conventional hypercolumn network. This allows the system to restructure itself to adapt to the changes introduced in the environment. Once the network adapts and improves its performance, it can be redeployed as a boolean logic circuit.

5.3.4 Experiment 4: Fast Emulation of a Faulty GPU using a Fault-free GPU

In this experiment, it is demonstrated that the behavior of a faulty GPU as described in Section 5.2 can be emulated by simply deactivating one or more minicolumns within the hierarchical hypercolumn network and running the network on a fault-free GPU. Demon-

strating this equivalence allows performing long-running and large-scale experiments on the robustness of the model without incurring the overhead of detailed simulation of a faulty GPGPU.

In order to show this equivalence, a cycle-accurate GPU simulator, GPGPUSim [6], configured to emulate the real GPU (GeForce 9800 GT) is used. GPGPUSim is used to simulate the effects of deactivated shader cores on the hypercolumn model. For this experiments, GPGPUSim is modified to support deactivation of different shader cores. The output of deactivated shader cores remain unchanged at zero.

On GPGPUSim, the following experiment is performed. A single hypercolumn is initialized with 32 minicolumns with a receptive field of size 3×3. Fifteen unique patterns of size 3×3 are then exposed to this hypercolumn. After a few training epochs, the hypercolumn starts to recognize each of the unique patterns. This means that, out of 32 minicolumns, 15 adjust their weights so that they fire for one of the unique features in the training set.

Consequently, randomly selected shader cores within GPGPUSim are deactivated to emulate permanent fault injections. These defected shader cores are then detected and deactivated. At this point, the impact of deactivating the defected shader cores on the recognition rate of the hypercolumn is evaluated. The results of this experiment are reported in Figure 5.9, labeled as `GPGPUSim`. All the results are averaged over 20 trials unless otherwise stated. The initial recognition rate (IRR) is the recognition rate immediately after the shader cores are deactivated, before any retraining, while the final recognition rate (FRR) is the recognition rate after retraining. Afterwards, the following experiment is conducted on the

real GPU, labeled as `HC Model`. 4 randomly selected minicolumns are deactivated by setting their output to zero and IRR and FRR for the real GPU are reported.

Two observations can be drawn from these experiments. First, the model demonstrates graceful degradation in the presence of permanent defects. After fault injection, the recognition rate of the hypercolumn falls. After retraining, the recognition rate of the hypercolumn recovers to 100% until the number of faulty shader cores per multiprocessor equals 5. GeForce 9800 GT has 8 shader cores per SM. This means that when executing a hypercolumn with 32 minicolumns on a SM, each shader core executes 4 minicolumns. Therefore, deactivating 5 shader cores is effectively equivalent to disabling 20 minicolumns. As a result, 3 out of the 15 features are not learned even after retraining because there are only 12 working minicolumns within the hypercolumn. Second, the GPGPUSim simulator with faulty shaders behaves almost exactly the same as the real GPU with deactivated minicolumns. Hence, for subsequent large-scale experiments, faulty shaders are emulated on a real GPU with deactivated minicolumns.

It should be noted that the shader core model and failures studied in this experiment are a proxy for the types of hardware and defects expected in a future computational hardware. Future generation processing hardware is expected to have multiple simple processing units similar to the shader cores available on current generation GPUs. Further, these simple cores might consist of multiple defective units. Considering these factors, the results of this experiment can be extrapolated to estimate how the proposed learning model might behave on faulty future technologies.

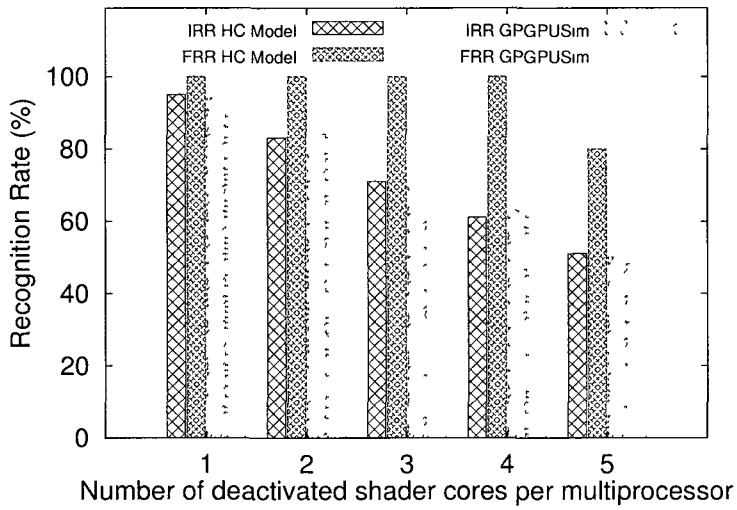

Figure 5.9: Effects of shader core deactivation on hypercolumn recognition rate while executing on GPGPUSim and real GPU (GeForce 9800 GT).

5.3.5 Experiment 5: Spatially Distributed Defects

In this experiment, the impact of faulty shaders, which are randomly spatially distributed, on the performance of a complex hypercolumn network is studied. For this experiment, the hierarchical hypercolumn network described in Table 4.1 is used and each of the hypercolumns is initialized with 20 minicolumns. This network is trained on a sample of handwritten digits (0-9) obtained from the MNIST database until it achieves 100% recognition rate. At this point, deactivation of randomly selected shader cores is simulated. Within the GeForce 9800, each multiprocessor has 8 shader cores and all the minicolumns within a hypercolumn are evenly distributed among these 8 shader cores. Since there are 20 minicolumns within a hypercolumn, deactivating one shader core affects 2.5 minicolumns per hypercolumn on average. Since on average 3.3 out of 47 hypercolumns map onto a single GeForce 9800 GT multiprocessor, deactivating 8.5 minicolumns emulates the behavior of one permanently

damaged shader core. After deactivating the minicolumns to emulate damaged shader cores, IRR is evaluated. At this point, the network is retrained until it achieves a stable recognition rate and FRR is measured. The results for this experiment are presented in Figure 5.10. On average, the same behavior as in the validation experiment described in Section 5.3.4 is observed. Immediately after fault injection, the recognition rate of the hypercolumn network drops. For example, when 6% of the shader cores are deactivated, IRR is measures to be 50%. Afterwards, with retraining using the same training set, the hypercolumn network recovers and FRR is measured to be 100%. Eventually, it should be noted that 100% final recognition accuracy is achieved even with only 50% functional shader cores. Damaging more that 50% shader cores affects results in deteriorating FRR. For example, with 56% shader cores deactivated FRR is measured to be around 40%. This is mainly because of the fact that at this point enough resources are not available for the hypercolumn network to recover the functionality lost due to the defected shader cores.

Figure 5.10 also highlights a shortcoming of using a single hierarchy of hypercolumns. Even with small number of defected shader cores, the recognition rate of the hypercolumn network is significantly affected. For example, even with only 6% defected shader cores, the measured IRR is around 50%. Such an issue can be resolved by exploiting the redundant information representation idea which also a strong biological basis. The robustness of the proposed hypercolumn model can directly benefit from redundant parallel hierarchies without any special algorithmic modification. All of these hierarchies share the same input but due to the randomness incorporated within the spontaneous activation behavior of the minicolumns,

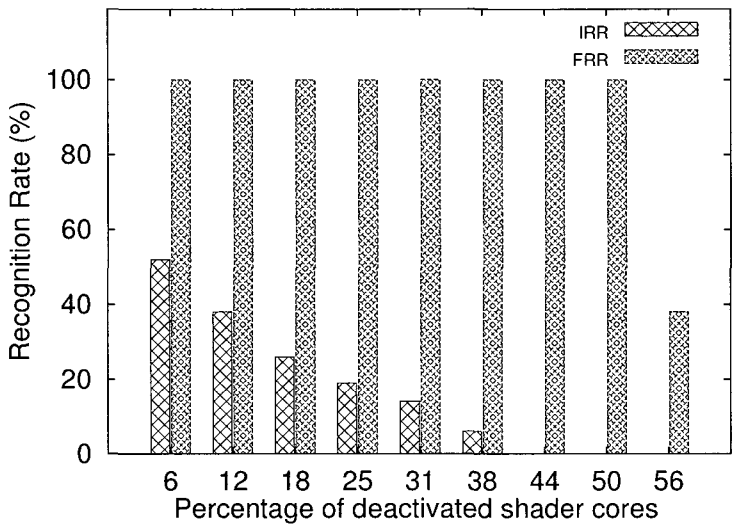

Figure 5.10: Effects of shader core deactivation on the hierarchical hypercolumn network recognition rate.

different minicolumns in each of the hierarchies will learn to recognize a specific feature. Thus, the overall chance of that specific feature getting lost due to any permanent defects is reduced.

To evaluate this hypothesis, 3 parallel hierarchical networks similar to the one described above are created. Each hypercolumn within these hierarchies contains 20 minicolumns. The output of each of these hierarchies is fed to an association network which pools minicolumns in the top level of each of the hierarchies firing for the same digit. Thus, if a minicolumn corresponding to a digit in any of the 3 hierarchies fires, the minicolumn in the association network associated to the digit will fire. Essentially, a minicolumn in the association network can be thought of as an 'OR' of of its inputs from the each of the hierarchical hypercolumn networks.

Each of these 3 hierarchies is trained on a sample of MNIST digits. After training,

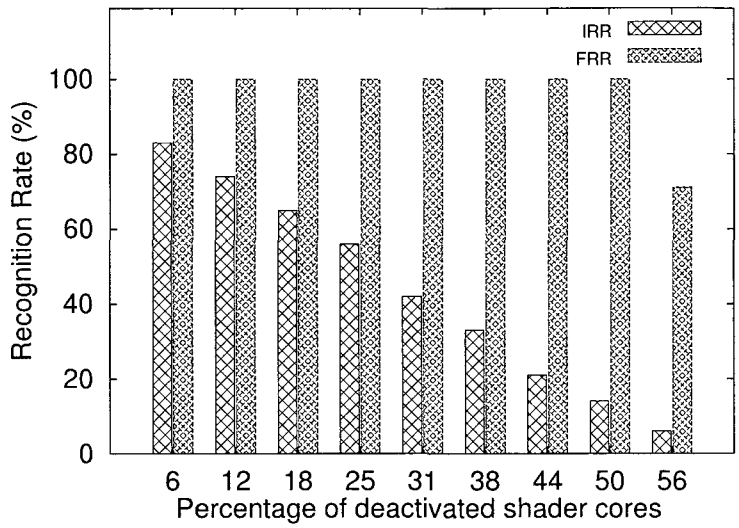

Figure 5.11: Effects of shader core deactivation on hypercolumn recognition rate with redundant hierarchies.

the same shader core deactivation process as described above is repeated. For this case, deactivating a single shader core once again affects 2.5 minicolumns per hypercolumn. But for this case 10 hypercolumns map onto a single multiprocessor. Thus deactivating a single shader core affects approximately 25 minicolumns throughout the 3 hierarchical networks. Figure 5.11 shows the IRR and FRR for this experiment. Comparing with Figure 5.10, it can be seen that the IRR of the larger hierarchical network degrades much more slowly and its FRR is improved as well. For this experiment, with 6% shader cores deactivated, IRR is measured to be 80% which is significantly better than the case with a single hierarchical hypercolumn network. Creating redundant hierarchies also improves FRR in the case when more that 50% shader cores are defective. For this case, with redundant hierarchies, FRR is measured to be around 70% as compared to 40% in the case of a single hypercolumn network.

The experiments described in this section mainly cover the case when defects are introduced

within the execution substrate due to wear-out or process variations. The results provided in Figures 5.10 and 5.11 clearly demonstrate that the hypercolumn model proposed in this dissertation has an inherent ability to gracefully tolerate such errors and recover its functionality through retraining. In addition to various modeled biological aspects, such a tolerance to permanent hardware failure is also attributed to the localized learning and operational rules of the hypercolumn model proposed in this dissertation. Since all the rules determining the behavior of a hypercolumn are local to it, any permanent hardware defect does not have global consequences. Thus, if a hypercolumn is affected due to a permanent hardware failure, it does not affect the correct behavior of any other hypercolumn. The only this that is affected is the structural connectivity among hypercolumns, that can be relearned through training. Conventional artificial neural networks lack such an ability to recover from permanent hardware defects. Any damage in the execution hardware can significantly affect the performance of these networks and in most cases renders them nonoperational.

5.3.6 Experiment 6: Spatially Clustered Defects

Apart from spatially distributed defects studied in Section 5.3.5, hardware defects can also occur in spatial clusters. For example some memory banks within a multi-processor unit in the GPU might get damaged. This will affect the working of multiple shader cores within the same multi-processor unit.

The sensitivity of a hypercolumn hierarchy to clustered defects is assessed in this experiment. In the model, neighboring minicolumns are more likely to carry similar information, or

to interact in an inhibitory fashion. As a result, clustered defects can potentially be more harmful to the task functionality. On the other hand, the fact that the information quickly spreads out across hierarchy levels can compensate for that vulnerability. For this experiment, the same hierarchical hypercolumn network as described in Section 4.1 is constructed and trained on a subset of MNIST digits. 5 neighbor shader cores at any hierarchical level are deactivated at a time and the recognition rates are measured. Figure 5.12 plots the results of this experiment. This figure plots the number of training iterations required by the network to achieve a stable FRR after the fault injection. It can be seen that if the deactivated shader cores are used for minicolumns higher in the hypercolumn hierarchy, it takes fewer retraining iterations for the network to achieve a stable FRR as compared to the case when the affected minicolumns are lower in the hierarchy. This is mainly due to the fact that if a minicolumn at level n is disabled, all the hypercolumns above level n getting input from the disabled minicolumn have to retrain themselves to relearn the lost features. This is an incremental process: first, the hypercolumn in level n will relearn the lost feature; then, given the new activation pattern in level n minicolumns, level $n+1$ hypercolumns must relearn this pattern; this learning is repeated all the way to the top of the hierarchy.

The results of this experiment also provide an insight into how a damaged region within the neocortex might manifest itself. If the damage is in some higher cortical region, the subject might recover at a faster rate as compared to the case when the damage is in some lower cortical regions. A damage in lower cortical regions indirectly affects higher cortical regions because higher regions need to restructure themselves to reflect or incorporate

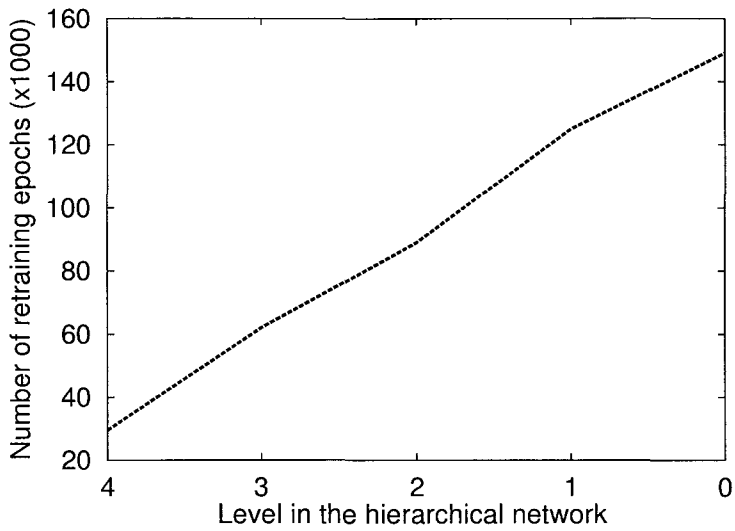

Figure 5.12: Number of retraining epochs required to achieve a stable FRR with spatially localized shader core deactivation.

the structural changes that took place in the lower cortical regions to recover the lost functionality. In such a case, depending on the location of the damaged region within the cortical hierarchy, more time is required to recover the lost functionality. Furthermore, the amount of functionality recovered directly depends on the amount of available resources. In the case of the hypercolumn network, the number of free minicolumns determine the amount of available resources. If a large number of minicolumns are available, most of the functionality lost because of the defects introduced will be recovered. However, if there are not enough minicolumns available, all of the functionality might not be recovered.

5.3.7 Summary of Experimental Results

This section describes several experiments that verify various hypothesis presented in the first half of this chapter. The main goal of these experiments is to verify that various conventional

computer architecture tools can be used to improve the performance as well as the resource utilization of complex biologically inspired networks. Furthermore, these experiments also demonstrate that biological properties like spontaneous activation and automatic abstraction allow computational models to develop inherent tolerance to permanent hardware failures. Section 5.3.1 demonstrates the ability of the hypercolumn network optimizer to restructure complex hierarchical networks to improve its performance as well as the robustness. Results described in this section clearly show that pruning the unnecessary minicolumns within a hypercolumn can result in 2x average speedup as well as 50% decrease in the memory requirement of a fully trained hypercolumn network. Section Section 5.3.2 demonstrates that the execution time of a hierarchical network can further be improved by generating an equivalent boolean logic circuit corresponding to the fully trained hypercolumn network in which synaptic weights are approximated with binary values. Such a conversion can allow the hypercolumn network execute orders of magnitude faster than the one not converted to a boolean logic representation. Section 5.3.3 evaluates the ability of the hypercolumn runtime monitoring system to revert a highly optimized boolean logic network to a network of hypercolumns for improved robustness. The experiment described in this section clearly demonstrates that the boolean logic network can be converted back to a hypercolumn network if the overall recognition rate of the network does not meet a specified threshold. Section 5.3.4 validates the hypothesis that permanent hardware faults can be emulated in software. As a result, instead of modeling permanent hardware faults in a detailed simulator which in extremely time consuming, permanent hardware faults are emulated in software by disabling

individual minicolumns within a hypercolumn. The experiment described in this section also demonstrate the inherent fault tolerance ability of the hypercolumn model proposed in this dissertation using a simple hypercolumn network. Section 5.3.5 validates the hypothesis that spontaneous activation property of the minicolumns within a hypercolumn provides the hypercolumn network with an inherent ability to tolerate randomly distributed permanent hardware failures. The results of the experiment described in this section clearly demonstrate the ability of the hypercolumn network to recover from permanent hardware failures simple through retraining. This section also shows that the tolerance of a hierarchical hypercolumn network to permanent hardware defects can further be improved using redundant parallel hierarchies. Finally, Section 5.3.6 demonstrates the ability of the hypercolumn network to tolerate spatially clustered hardware defects. The results of this experiment demonstrate that the time taken by the hypercolumn network to recover from spatially clustered defects is determined by the location of the minicolumns with the hypercolumn hierarchy that are affected by such defects. If the minicolumns affected by the defects are within the hypercolumns in the upper hierarchical levels, the network takes less time to recover as compared to the case when affected minicolumns are in the lower hierarchical levels.

5.4 Summary

This chapter establishes a bidirectional relationship between computer architecture and the proposed biologically inspired model. Through various experiments, this section demonstrates that various conventional architectural tools can help optimize the performance of complex

biological networks. At the same time, the automatic abstraction and spontaneous activation properties of the proposed hypercolumn model help develop computational models that are inherently tolerant to permanent hardware failures.

6 CONCLUSION AND REFLECTIONS

Advances in the understanding of the structural and functional properties of the brain paired with the recent technological pressures on the traditional von Neumann model has led researchers to explore alternative computational paradigms. This dissertation advocates using the cortical columns that exist throughout the mammalian neocortex as an inspiration towards developing a biologically inspired computational abstraction that addresses some of the recent architectural challenges.

In this chapter, first, various contributions and findings of this dissertation are reviewed. Second, reflections on various interesting ideas that did not become part of this dissertation are discussed as future work.

6.1 Summary and Conclusion

This dissertation is grounded in the philosophy that computational paradigms radically different from the traditional von Neumann model must be explored in order to cater to the needs of the future generation applications as well as hardware constraints. In this effort, this dissertation proposes a uniformly structured, hierarchically organized, and biologically inspired computational model inspired by the structural and functional properties of the mammalian neocortex. This model uses cortical columns as its basic functional abstraction and organizes these columns in the form of hierarchical networks to realize complex tasks. Biological inspirations of the proposed model is ascribed to its detailed

modeling of various aspects of biological cortical columns (hypercolumns and minicolumns) as well as the feedforward, feedback, and lateral inhibitory processing pathways. In addition to being biologically inspired, the proposed model is computationally efficient as well, as each modeled hypercolumn abstracts away the working of around 10,000 neurons.

In terms of modeling a biologically inspired hierarchical network, this dissertation demonstrates the use of spatially localized and temporally correlated spontaneous activations, feedforward and lateral communication paths, and Hebbian-like learning rules to develop hierarchical organized feature maps. These feature maps that evolve through the concept of automatic abstraction are used to generate invariant representations of various unique objects. Furthermore, this dissertation also highlights and models several important roles of feedback communication paths in the neocortical hierarchy. Within the proposed model, these feedback paths help in modulating the Gaussian-like feedforward connectivity using global context. These feedback paths are also used to implement various pooling mechanisms to develop generalized representations for the variations of the same pattern as well as unpooling mechanism to separate exceptions for the generalized representations.

Another key contribution of this dissertation is to establish a symbiotic relationship between cortical networks and computer architecture. On the one hand, complex networks can benefit from several computer architecture concepts like functional abstractions, Instruction Set Architecture, online profiling and optimizations, efficient resource management, etc. These architectural concepts make a complex biological model scalable, portable, and computationally efficient. On the other hand, conventional algorithms and hardware can

benefit from biological properties like automatic abstraction and spontaneous activations to develop tolerance to permanent hardware defects.

Finally, this dissertation is geared towards developing a comprehensive and biologically inspired understanding of the microarchitecture of computing systems that mimic the human neocortex, and applying such systems for robust realization of complex tasks.

6.2 Reflections and Future Work

This section describes various interesting ideas that initiated during the course of this project but are not part of this dissertation.

6.2.1 From a Blank Slate to a Complex Hierarchical Network

A very interesting property of the mammalian brain, especially the neocortex, is the formation of complex and densely interconnected structures of neurons to realize complex behavior. The process of forming these complex structures involves both developing new connections as well as pruning unnecessary connections. This clearly suggests that initially the neocortex has limited or even no connectivity (like a blank slate) but, over time, based on the learning experiences of an individual, the connectivity and the hierarchy evolves.

In a typical artificial or biological neural network model, the hierarchical organization does not evolve over time; rather some notion of hierarchical connectivity is assumed. This may limit the robustness of such a network in terms of developing an efficient hierarchical network to realize a complex task. We hypothesize that initially, the neocortex appears

as a blank slate and neurons have very limited connectivity but the neurons observe three operational rules that ensure their survival. First, neurons must fire in response to some stable stimulus in order to survive. Second, firing neurons want someone to listen to them. Any useful activity generated by a neuron results in release of certain low-level chemicals (a reward mechanism) that results in nourishing that neuron. Third, populations of neuron compete with each other to improve their chances of survival. In summary, neurons want to fire and they want their firing to be heard because these two mechanisms ensure the survival of neurons. With the help of these operational rules, complex hierarchical structures evolve over time. Initially, the neurons physically located near the sensory modalities start to respond to stable input stimuli and learn to extract independent features. Neurons with similar feature preferences form tightly connected networks and compete with other neurons firing for the same feature. This implements a winner-take-all sort of a behavior. Afterwards, neurons that are physically distant from the sensory modalities, connect with populations closer to the sensory modalities and start to extract higher order features. This process continues and eventually complex hierarchical networks of densely connected populations of neurons are formed to realize complex higher order tasks.

In summary, formal mechanisms that allow artificial or biologically inspired neural networks to evolve hierarchically and in a robust manner need to be investigated. Rather than defining the hierarchical organization of a neural network prior to exposing it to a dataset, the network should learn to develop its hierarchical organization based on the training data properties.

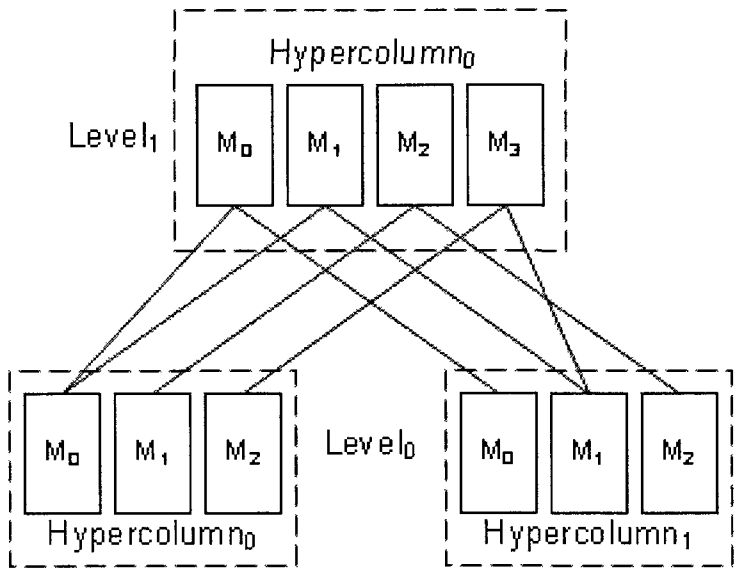

Figure 6.1: A graphical representation of spatial correlations created among various features recognized by the 2-Level hypercolumn network after it reaches a steady state.

6.2.2 Identifying Sufficient Features for Recognition

Although there exists a large body of literature explaining the various neocortical properties described in Chapter 3, certain alternative properties of the neocortex have not been sufficiently explored. One such property of the neocortex is its ability to determine the set of features that are sufficient to recognize variations of an object. Obviously, the neocortex does not require all of the object's features to recognize it. For example, Maw and Pomplun [89] show that while recognizing a human face, subjects tend to gaze more at the eyes, lips, and nose. Thus, out of all the features that constitute a human face the neocortex uses a small subset to process the facial recognition task. Sigala and Logothetis [123] show similar results in more details. Thus, within the neocortex, there exists of notion of a sets of features that are sufficient to differentiate visual patterns. To determine the set of features sufficient

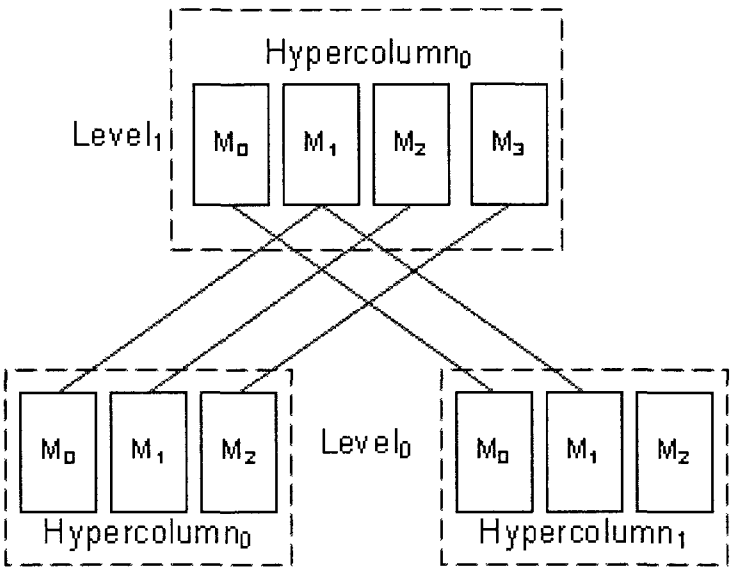

Figure 6.2: Reduced graph obtained after removing the connections with redundant information from the graph shown in Figure 6.1.

for recognition of unique images, we propose considering the spatial correlations that exist among the occurrence of different independent features constructing an object. This results in a connectivity graph of features with spatial correlations. One such spatial correlation graph is shown in Figure 6.1. This figure contains a 2-Level hierarchical hypercolumn network and shows various spatial correlations that exist among the minicolumns in $Level_0$ hypercolumns. For example, Minicolumn 0 in Hypercolumn 0 within Level 0 (M0H0L0) has a spatial correlation with M0H1L0 and M1H1L0 because M0H0L0 always co-occurs with M0H1L0 OR M1H1L0.

Now, the set of features sufficient for identifying the shapes exposed to the network are determined using the spatial correlation graph like the one shown in Figure 6.1.

At each level of the hierarchy, the spatial correlations between different minicolumns are

observed and the associations that provide redundant information are marked. For example, if minicolumn A is just spatially associated to minicolumn B and minicolumn B is spatially associated to some other minicolumn as well, then the feature recognized by minicolumn A is sufficient enough to identify the shape typically recognized by minicolumns A and B together. Figure 6.2 shows the same graph as displayed in Figure 6.1 but with the redundant connections removed. Now, the output of minicolumn A is associated with a relatively higher weight value. This ensures that if minicolumn A fires then it is highly likely that its the same shape that is typically recognized by minicolumns A and B together.

In summary, in order to recognize a complex object, the neocortex does not utilize all the features defining that object. Rather, a small subset of these features, that are sufficient for recognition, are considered. Similar approaches must be considered while implementing complex neural models. Instead of treating all the features that constitute an object equally, a subset of these features may be given more importance. These features may be determined based on various spatial and temporal correlations that may exist among different objects or different features of the same object

6.2.3 Integrating Multi-modal Information

A number of neuroscience studies show that the higher neocortical regions simultaneously integrate information from multiple sensory modalities. This integration of multi-model information plays a very important role in that case when useful information needs to be extracted from noise input. A typical example where this phenomenon is quite evident is the

the cocktail party problem. When listening to someone in a noisy environment, we are able to make sense out of a noisy input by integration the auditory input with the lip movement of the speaker. In such a scenario, multi-model information integration allows the human brain to combine the information obtained from the auditory cortex and the visual cortex to better approximate the auditory input.

We also plan to add such a capability within our hierarchical hypercolumn model. At the lower level, two hypercolumn networks process visual and auditory information separately. On top of these two hypercolumn networks, there is another hypercolumn network that integrates information communicated by each of the lower hypercolumn networks. This type of integration may improve the overall robustness and performance of the network in the presence of noisy inputs.

6.2.4 Temporal Information Processing

Another interesting aspects of the neocortex is its ability to learn and generate temporal sequences. Even though generation and learning of temporal sequences play critical roles within the cortical processing, exact biological mechanisms governing these cortical attributes are not well-understood [132].

Temporal information processing within the neocortex can be attributed to a number of powerful cortical abilities. First, this aspect plays a key role during the auditory cortex information processing. The ability of the auditory cortex to temporally integrate auditory data to comprehend musical notes, musical patterns, words, sentences, etc. is attributed

to this temporal sequence learning mechanisms. This auditory temporal processing also incorporates a notion of invariance to it. For example, once we learn to recognize a song being played in a certain tempo or beat, we can recognize it no matter what tempo or beat it is played in. Apart from the auditory cortex temporal processing also manifests itself within the visual cortex in the form of temporal visual pattern recognition. Second, temporal information processing allows the motor cortex to generate complex temporal response to a stimulus. These responses can be used to implement simple tasks like muscular movements or they can be used to realize extremely complex tasks like riding a bicycle, catching a ball, swimming, driving a car, etc. Essentially, temporal information processing is required for any complex tasks that involves the motor cortex. Third, temporal processing of the input data plays an important role in modulating the information being received through the sensory modalities according to a global context. It is well known that the neocortex is able to predict future actions based on global context. Based on our experience in the near or distant past, the neocortex can make several complex predictions that determine our actions. Temporal information integration may allow the neocortex to realize complex prediction-making mechanisms.

Temporal information processing mechanisms can be implemented within the proposed hypercolumn network by employing delay nodes between various hierarchical levels. A delay node replicates the activity of the minicolumn it is connected to with a delay of one cycle/epoch. Thus, the minicoumns within the hypercolumn in the next hierarchical level can combine the delayed information with the present input stimulus and can learn to associate

the two stimuli in a sequential manner. This approach can be implemented in a hierarchical manner to learn and generate sequences of arbitrary lengths.

In summary, temporal information processing plays a key role in throughout the cortical hierarchy. Such a mechanism needs to be implemented within any learning model in order to robustly generate and learn temporal sequences and to make prediction based on global context.

6.2.5 Summary

This dissertation proposes a learning model inspired by various structural and functional properties of the neocortex. Even though this proposed learning model incorporates several powerful biological properties, there are a number of other powerful biological aspects that still need to be added to this model. This section discusses various cortical properties like automatic emergence of hierarchy, identifying features sufficient for recognition, multi-model information integration, and temporal sequence learning and highlights various mechanisms to implement such properties within the proposed learning model.

BIBLIOGRAPHY

[1] The facets project. http://facets.kip. uni-heidelberg. de, 2010. retrieved: Oct, 2011.

[2] A universal spiking neural network architecture (spinnaker). http://apt.cs.man.ac.uk/ projects/ SpiNNaker/, 2010. retrieved: Oct, 2011.

[3] M. Aiken and M. Bsat. Forecasting market trends with neural networks. *Information Systems Management*, 16(4):1–7, 1999.

[4] M. Albert, A. Schnabel, and D. Field. Innate visual learning through spontaneous activity patterns. *PLoS Computational Biology*, 4(8):e1000137, 2008.

[5] R. Ananthanarayanan and D. Modha. Anatomy of a cortical simulator. In *Proceedings of Supercomputing*, 2007.

[6] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamondt. Analyzing cuda workloads using a detailed gpu simulator. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009.

[7] R. Baumann. Soft errors in advanced computer systems. *IEEE Design Test of Computers*, 22(3):258 – 266, 2005.

[8] J. Le Be and H. Markram. A new mechanism for memory: neuronal networks rewiring in the young rat neocortex. *Medical Science*, 22(12):1031–1033, 2006.

[9] S. Berg. Neurology. `http://willistonberg.pbworks.com`, 2010. retrieved: Sep, 2011.

[10] H. Berry and O. Temam. Modeling self-developping biological neural network. *Neurocomputing*, 70(16-18):2723–2734, 2007.

[11] T. Binzegger, R.J. Douglas, and K.A.C. Martin. A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci.*, 24(39):8441–8453, Sep 2004.

[12] G. Blasdel and G. Salama. Voltage sensitive dyes reveal a modular organization in monkey striate cortex. *Nature*, 321:579–585, 1986.

[13] J. Blome, S. Gupta, S. Feng, and S. Mahlke. Cost-efficient soft error protection for embedded microprocessors. In *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems*, pages 421–431, 2006.

[14] S. Borkar. Design challenges of technology scaling. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, volume 19, pages 23 –29, jul-aug 1999.

[15] R. Brown and P. Milner. The legacy of Donald O. Hebb: more than the Hebb synapse. *Nature Neuroscience*, 4(12):1013–1019, 2003.

[16] J. Bullier, J. Hupe, A. James, and P. Girard. The role of feedback connections in shaping the responses of visual cortical neurons. *Progress in Brain Research*, 134:193–204, 2001.

[17] W. Calvin. Cortical columns, modules, and hebbian cell assemblies. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 269–272. MIT Press, Cambridge, MA, 1998.

[18] J. Cang, R. Renteria, M. Kaneko, X. Liu, D. Copenhagen, and M. Stryker. Development of precise maps in visual cortex requires patterned spontaneous activity in retina. *Neuron*, 48(5):797–809, 2005.

[19] G. Carpenter and S. Grossberg. Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, 1987.

[20] G. Carpenter and S. Grossberg. Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.

[21] G. Carpenter and S. Grossberg. *Adaptive Resonance Theory*. MIT Press, 2003.

[22] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, and D. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 4:698–713, 1992.

[23] NVIDIA Corporation. *CUDA Programming Guide*. NVIDIA Corporation, 2701 San Toman Expressway, Santa Clara, CA 95050, USA, 2007.

[24] DARPA. Systems of neuromorphic adaptive plastic scalable electronics. http://www.darpa.mil/dso/solicitations/baa08-28.html, 2008. retrieved: Sep, 2011.

[25] A. Dehon. Nanowire-based programmable architectures. *Journal of Emerging Technology and Computing System*, 1(2):109–162, 2005.

[26] H. Deogun, D. Sylvester, and D. Blaauw. Gate-level mitigation techniques for neutron-induced soft error rate. In *Proceedings of the Sixth International Symposium on Quality of Electronic Design*, pages 175 – 180, 2005.

[27] M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchical self-organizing map. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 15–19, 2000.

[28] M. Emmerson and R. Damper. Determining and improving the fault tolerance of of multilayer perceptrons in a pattern-recognition application. In *IEEE Transactions on Neural Networks*, volume 4, pages 788–793, 1993.

[29] FACETS. `http://facets.kip.uni-heidelberg.de/index.html`, 2011. retrieved: Sep, 2011.

[30] B. Farley, H. Yu, D. Jin, and M. Sur. Alteration of visual input results in a coordinated reorganization of multiple visual cortex maps. *Journal of Neuroscience*, 27(38):10299 –10310, 2007.

[31] D. Felleman and D. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1:1–47, 1991.

[32] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 148 –157, 2002.

[33] W.J. Freeman. Random activity at the microscopic neural level in cortex ("noise") sustains and is regulated by low-dimensional dynamics of macroscopic activity ("chaos"). *International Journal of Neural Systems*, 7(4):473–480, 1996.

[34] D. George and J. Hawkins. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of International Joint Conference on Neural Networks*, volume 3, pages 1812–1817. IEEE International Joint Conference on Neural Network, 2005.

[35] G. Gielen, E. Maricau, and P. De Wit. Design automation towards reliable analog integrated circuits. In *IEEE International Conference on Computer-Aided Design*, pages 248 –251, 2010.

[36] J. Gonzaga, L. Meleiro, C. Kiang, and R. Filho. Ann-based soft-sensor for real-time process monitoring and control of an industrial polymerization process. *Computers and Chemical Engineering*, 33(1):43–49, 2009.

[37] M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Learning*, 14(1):76–86, 1992.

[38] GravitySpa. Three brain. `http://www.gravityspa.com`, 2010. retrieved: Sep, 2011.

[39] Richard L. Gregory. *Eye and Brain: the Psychology of Seeing.* Princeton University Press, fifth edition, 1997.

[40] S. Grillner. Bridging the gap from ion channels to networks and behavior. *Current Opinion in Neuroscience*, 9:663–669, 1999.

[41] S. Grossberg. *Competitive learning: From interactive activation to adaptive resonance*, pages 213–250. Massachusetts Institute of Technology, Cambridge, MA, USA, 1988.

[42] S. Haeusler and W. Maass. A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cereb. Cortex*, 17(1):149–162, Jan 2007.

[43] A. Hashmi, H. Berry, O. Temam, and M. Lipasti. Automatic abstraction and fault tolerance in cortical microarchitectures. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 1–12, 2011.

[44] A. Hashmi and M. Lipasti. Discovering cortical algorithms. In *Proceedings of the International Conference on Neural Computation (ICNC)*, October 2010.

[45] A. Hashmi, A. Nere, and M. Lipasti. Learning through spatially localized and temporally correlated spontaneous activations. In *International Conference on Cognitive and Neural Systems*, pages 21–28, 2011.

[46] A. Hashmi, A. Nere, J. Thomas, and M. Lipasti. A case for neuromorphic isas. In *Proceedings of the International conference on Architectural support for programming languages and operating systems*, pages 145–158, 2011.

[47] J. Hawkins and S. Blakeslee. *On Intelligence.* Henry Holt & Company, Inc., 2005.

[48] Simon Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition).* Prentice Hall, 1998.

[49] D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *MACHINE LEARNING*, 20(3):197–243, 1995.

[50] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 137 –147, 2002.

[51] E. Hill, M. Lipasti, and K. Saluja. An accurate flip-flop selection technique for reducing logic ser. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 128 –136, 2008.

[52] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[53] J. Hirsch and L. Martinez. Laminar processing in the visual cortical column. *Current Opinion in Neurobiology*, 16:377–384, 2006.

[54] A. Hodgkin and A. Huxley. The components of membrane conductance in the giant axon of loligo. *Journal of Physiology*, 116(4):473–496, 1952.

[55] A. Hodgkin and A. Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *Journal of Physiology*, 116(4):449–472, 1952.

[56] A. Hodgkin, A. Huxley, and B. Katz. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *Journal of Physiology*, 116(4):424–448, 1952.

[57] M. Holler, S. Tam, H. Castro, and R. Benson. An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses. In *Proceedings of the International Joint Conference on Neural Networks*, pages 191–196, 1989.

[58] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 32–37, 2004.

[59] D. Hubel and T. Wiesel. Receptive fields, binocular interactions and functional architecture in cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[60] D. Hubel and T. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

[61] J. Hupe, A. James, P. Girard, S. Lomber, B. Payne, and J. Bullier. Feedback connections act on the early part of the responses in monkey visual cortex. *Journal of Neurophysiology*, 85:134–145, 2001.

[62] J. Hupe, A. James, B. Payne, S. Lomber, P. Girard, and J. Bullier. Cortical feedback improves discrimination between figure and background by v1, v2 and v3 neurons. *Nature*, 394:784–787, 1998.

[63] K. Ibata, Q. Sun, and G.G. Turrigiano. Rapid Synaptic Scaling Induced by Changes in Postsynaptic Firing. *Neuron*, 57(6):819–826, 2008.

[64] Numenta Inc. Hierarchical temporal memory. http://www. numenta.com, 2007. retrieved: Apr, 2011.

[65] E. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14:1569–1572, 2003.

[66] X. Jin and G. Zhang. Modelling optimal risk allocation in ppp projects using artificial neural networks. *International Journal of Project Management*, 29(5):591–603, 2011.

[67] J.Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, 1996.

[68] N. Kalisman, G. Silberberg, and H. Markram. The neocortical microcircuit as a tabula rasa. *Proceedings of the National Academy of Science*, 102:880–885, 2005.

[69] E.R. Kandel, J.H. Schwartz, and T.M. Jessell. *Principles of Neural Science*. McGraw-Hill, 4 edition, 2000.

[70] A. Khashman. Neural networks for credit risk evaluation: Investigation of different neural models and learning schemes. *Expert Systems with Applications*, 37(9):6233–6239, 2010.

[71] J. Kim and H. Ahn. A new perspective for neural networks: Application to a marketing management problem. *Journal of Information Science and Engineering*, 25(1):1605–1616, 209.

[72] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68 – 75, 2003.

[73] P. Kittisupakorna, P. Thitiyasooka, M. Hussain, and W. Daosud. Neural network based model predictive control for a steel pickling process. *Journal of Process Control*, 19(4):579–590, 2009.

[74] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464 –1480, 1990.

[75] T. Kohonen and P. Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30, 1998.

[76] G. Kreiman, C. Koch, and I. Fried. Category-specific visual responses of single neurons in the human medial temporal lobe. *Nature Neuroscience*, 3:946–953, 2000.

[77] A. Krizhevsky. Convolutional deep belief networks on cifar-10. Technical report, UOFT, 2010.

[78] P. Lapuerta, S. Azen, and L. Labree. Use of neural networks in predicting the risk of coronary artery disease. *Computers and Biomedical Research*, 20(1):38–52, 1995.

[79] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

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

[81] Y. Lecun and C. Cortes. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. retrieved: Sep, 2007.

[82] T. Lee, D. Mumford, R. Romero, and V. Lamme. The role of primary visual cortex in higher level vision. *Vision Research*, 38:2429–2454, 1998.

[83] H. Li, S. Bhunia, Y. Chen, T. Vijaykumar, and K. Roy. Deterministic clock gating for microprocessor power reduction. In *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture*, pages 113 – 122, 2003.

[84] N. Logothetis, J. Pauls, and T. Poggio. Shape representation in the inferior temporal cortex of monkeys. *Current Biology*, 5(5):552–563, 1995.

[85] A. Losonczy and J.C. Magee. Integrative properties of radial oblique dendrites in hippocampal ca1 pyramidal neurons. *Neuron*, 50:291–307, 2006.

[86] C. Lu and C. Tsai. Generalized predictive control using recurrent fuzzy neural networks for industrial processes. *Journal of Process Control*, 17(1):83–92, 2007.

[87] H. Markram. The blue brain project. *Nature Reviews Neuroscience*, 1:153–160, 2006.

[88] M. Matthias and J. Born. Hippocampus whispering in deep sleep to prefrontal cortex for good memories? *Neuron*, 61:496–498, 2009.

[89] N. Maw and M. Pomplun. Studying human face recognition with the gaze-contingent window technique. In *Proceedings of the Twenty-Sixth Annual Meeting of Cognitive Science Society*, pages 927–932, 2004.

[90] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Proceeding of the IEEE Custom Integrated Circuits Conference*, 2011.

[91] S. Mitra, M. Zhang, S. Waqas, N. Seifert, B. Gill, and K. Kim. Combinational logic soft error correction. In *Proceedings of the IEEE International Test Conference*, pages 1 –9, 2006.

[92] D. Modha. Synapse. `http://www.ibm.com/`, 2011. retrieved: Sep, 2011.

[93] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):237–285, 1965.

[94] V. Mountcastle. Modality and topographic properties of single neurons of cat's somatic sensory cortex. *Journal of Neurophysiology*, 20(4):408–434, 1957.

[95] V. Mountcastle. The columnar organization of the neocortex. *Brain*, 120:701–722, 1997.

[96] T. Mudge. Power: a first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[97] S. Mukherjee, J. Emer, and S. Reinhardt. The soft error problem: an architectural perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 243 – 247, 2005.

[98] A. Nere and M. Lipasti. Cortical architectures on a gpgpu. In *GPGPU '10: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 12–18, New York, NY, USA, 2010. ACM.

[99] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proceedings of the International Joint Conference on Neural Networks*, pages 21–26, 1990.

[100] J. Nickolls and W. Dally. The gpu computing era. *IEEE, Micro*, 30(2):56–69, 2010.

[101] M. O'Neil. Neural network for recognition of handwritten digits. http://www. codeproject. com/ KB/ library/ NeuralNetRecognition.aspx, 2010. retrieved: Apr, 2011.

[102] J. Ott. Neural networks and disease association studies. *American Journal of Medical Genetics*, 105(1):60–61, 2001.

[103] J.J. Peissig and M.J. Tarr. Visual object recognition: do we know more now than we did 20 years ago? *Annu. Rev. Psychol.*, 58:75–96, 2007.

[104] R. A. Peters. On the computation of discrete log-polar transform. Technical report, Department of Electrical and Computer Engineering, Vanderbilt University, 2007.

[105] C. Peterson, R. Malenka, R. Nicoll, and J. Hopfield. All-or-none potentiation at ca3-ca1 synapses. *Proceedings of the National Academy of Sciecne*, 95:4732–4737, 1998.

[106] N. Pinto, D.D. Cox, and J.J DiCarlo. Why is real-world visual object recognition hard? *PLoS Comput. Biol.*, 4(1):e27, Jan 2008.

[107] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[108] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[109] D.L. Ringach. Haphazard wiring of simple receptive fields and orientation columns in visual cortex. *J. Neurophysiol.*, 92(1):468–476, Jul 2004.

[110] U. Rokni, A. Richardson, E. Bizzi, and H. Seung. Motor learning with unstable neural representations. *Neuron*, 64:653–666, 2007.

[111] G. Roth and U. Dicke. Evolution of brain and intelligence. *TRENDS in Cognitive Sciences*, 5:250–257, 2005.

[112] D. Rumelhart and D. Zipser. *Feature discovery by competitive learning*, volume 1, pages 151–193. MIT Press, Cambridge, MA, USA, 1986.

[113] D. Rumelhart and D. Zisper. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[114] S. Ryoo, C. Rodrigues, S. Babhsorkhi, S. Stone, D. Kirk, and W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceesings Symposium on principles and practices of parallel programming, SIGPLAN*, pages 73 –82, 2008.

[115] Anders Sandberg and Nick Bostrom. Whole brain emulation: A roadmap. Technical Report 2008-3, Future of Humanity Institute, Oxford University, 2008.

[116] J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN*, pages 431–438, June 2008.

[117] E. Schwartz. Spatial mapping in the primate sensory projection: Analytic structure and relevance to perception. *Biology Cybernetics*, 25:181–194, 1977.

[118] E. Schwartz. Computational anatomy and functional architecture of striate cortex: A spatial mapping approach to perceptual coding. *Vision Research*, 20:645–669, 1980.

[119] T. Serre, A. Oliva, and T. Poggio. A feedforward architecture accounts for rapid categorization. *Proc. Natl. Acad. Sci. USA*, 104(15):6424–6429, Apr 2007.

[120] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):411–426, Mar 2007.

[121] H. Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40:1063–1073, 2003.

[122] SIA. Semiconductor industry association 2007 roadmap. http://www.sia-online.org/, 2007.

[123] N. Sigala and N. Logothetis. Visual categorization shapes feature selectivity in the primate temporal cortex. *Nature*, 415:318–320, 2002.

[124] A.M. Sillito, J. Cudeiro, and H.E. Jones. Always returning: feedback and sensory processing in visual cortex and thalamus. *Trends in Neuroscience*, 29(6):307–316, 2006.

[125] P. Snow, S. Smith, and W. Catalona. Artificial neural networks in the diagnosis and prognosis of prostate cancer: a pilot study. *The journal of urology*, 152(2):1923–1926, 1994.

[126] M. Spratling. Presynaptic lateral inhibition provides a better architecture for self-organising neural networks. *Network: Computation in Neural Systems,*, 10(4):285–301, 1999.

[127] I. Sutskever and G. Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 544–551, 2007.

[128] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[129] L.W. Swanson. Mapping the human brain: past, present, and future. *Trends in Neurosciences*, 18(11):471 –474, 1995.

[130] G. Taylor, G. Hinton, and S. Roweis. Modeling human motion using binary latent variables. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1345–1352, 2007.

[131] B. Tjan, V. Lestou, and Z. Kourtzi. Uncertainty and invariance in the human visual cortex. *Journal of Neuropsychology*, 96(3):1556–1568, 2006.

[132] S. Verduzco-Flores, M. Bodner, and B. Ermentrout. Learning and generation of temporal sequences in the neocortex. *BMC Neuroscience*, 11:101, 2010.

[133] B. Wandell, S. Dumoulin, and A. Brewer. Visual field maps in human cortex. *Neuron*, 56:366–383, 2007.

[134] G. Weckman, S. Lakshminarayanan, J. Marvel, and A. Snow. An integrated stock market forecasting model using neural networks. *International Journal of Business Forecasting and Marketing Intelligence*, 1(1):30–49, 2008.

[135] R. Williams. How we found the missing memristor. *IEEE, Spectrum*, 45(12):28–35, 2008.

[136] D. Zheng, J. Zhao, and A. Saddik. Rst-invariant digital image watermarking based on log-polar mapping and phase correlation. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(8):753 – 765, 2003.