Optimization of Spatial Architectures for Acceleration of Machine Learning

Mateja Putic

Spring 2018

1 Introduction

Machine Learning Automates Extraction of Meaning From Big Data While the age of big data has enabled efficient large-scale data collection, automated analysis remains essential for extracting high semantic meaning quickly and on a large scale. Machine learning has offered a compelling set of solutions to this application domain because of its ability to autonomously identify structure and correlations in large datasets [1]. DNNs have successfully been deployed in platforms ranging from cloud services to embedded devices in image recognition and classification, cancer diagnosis, spam filtering, voice recognition, and applications controlling self-driving vehicles [35, 38, 39, 45, 52, 69, 40, 29].

Machine Learning Creates Massive Computation Demands Due to the implementation details of state-of-the-art DNNs, these algorithms are very computationally expensive. Tera- or peta-byte sized training sets are required for high accuracy, many arithmetic operations are required for a single inference, models can have up to 100 layers, and many parallel inference requests require running the model many times or continuously [17]. While demand for DNN performance has largely benefited from the relatively recent introduction of large GPUs that offer parallel computational capability [64], concerns about power and reliability [71] have grown proportionally, across deployment scales.

Machine Learning Algorithms Map Well to Spatial Architectures Machine learning algorithms are often realized as a directed graph of compute operations with varying degrees of data sharing between nodes. Spatial architectures are a class of hardware architecture that are characterized by a large array of identical compute units, a shared memory architecture, and an interconnect network that allows them to communicate [63, 49, 48]. Spatial architectures are a good fit for machine learning algorithms because they can exploit varying degrees of data reuse across multiple dimensions of their inputs to capture abundantly available thread parallelism in DNN dataflows. Examples of classical spatial architectures with such characteristics are TRIPS [68], Raw [75], Imagine [43], and more recently, Google's Tensor Processing Unit [42], Graphcore [76], and the Micron Automata Processor [20], as well as embedded platforms like Minerva [67], Eyeriss [10], ShiDianNao [23], and EIE [33]. Further optimization of hardware metrics of spatial architectures warrants understanding major properties of these algorithms.

Machine Learning Algorithms Have Diverse Workload Characteristics There exist a large number of DNN networks used for various applications, each of which are comprised of a sequence of operational layers. Within networks, there exists intra-network diversity among layers of differing types, *e.g.*, convolution, activation, and pooling, *etc.*. Although a majority of the operations in DNNs are implemented by GEMM, the data structures between these layers have varying dimensions, *i.e.*, shape and rank. As a consequence of this diversity, DNN program phases present a wide range of operational intensities to the underlying hardware, resulting in varied utilization in a fixed hardware architecture.

Machine Learning Algorithms Have Flexible Semantics DNNs have an abundance of resilience properties that can be leveraged to design more efficient spatial architecture DNN accelerators. The functionality and accuracy of DNNs has been shown to be resilient to distortion and noise in inputs by *generalization* of exact functions achieved through training [13, 21, 74, 86]. To avoid overfitting toward improving function generalization and achieving high accuracy, modern DNN training includes *pruning* the least important parameters, as determined by backpropagation of error [58, 72, 34]. Additionally, DNNs are resilient to

changes in network architecture, for example, by replacement of layers by operations with reduced data requirements and lower operational intensity [53, 73]. Several parameter approximation and quantization methods for DNNs have been applied to save memory and compute cycles to improve their performance, and make them suitable for deployment on memory-constrained platforms [77, 18, 31]. Furthermore, several previous works have demonstrated that resilience properties of neural networks can be leveraged to improve runtime hardware characteristics [24, 78, 83]

2 Problem Definitions and Research Hypothesis

Broadly, the challenge to be addressed by this work is to develop methodologies for the design of spatial architectures to accelerate a broad class of DNN applications and to give hardware designers tools to explore tradeoff spaces between application and hardware metrics. We do this by recognizing that if algorithmic properties of DNNs are heterogeneous in nature, then they can be exploited to make changes at the hardware level to improve and make tradeoffs and improve resource usage. To approach this challenge systematically, we identify three specific research problems to be addressed:

Problem 1: Statically Allocated DNN Accelerator Hardware Present Unclaimed Design Margins Inter- and intra-network diversity in DNN workloads result in diversity in the utilization of hardware throughout the execution phases of a DNN network. As a consequence, statically allocated accelerator hardware runs sub-optimally and leaves energy-efficiency margins unclaimed, creating opportunities for optimization of runtime metrics based on prior knowledge of these characteristics. To give hardware designers a tool to better align application requirements with resources available in the deployment scenario, what is needed is a methodology to design hardware that takes advantage of the workload diversity to make DNN execution more efficient, and to enable tradeoffs between better performance and lower power.

Problem 2: Biological Neuron Algorithms Lack Performance and Scalability on von Neumann Architectures Capturing a large degree of parallelism available in machine learning algorithms in hardware is challenging for two reasons: (i) existing hardware may have a small number of cores, bounding the maximum degree of parallelism, and (ii) memory access patterns within dataflow graphs exhaust traditional von Neumann memory hierarchy. Both of these issues limit scalability on such architectures, consequently limiting their throughput and performance. To demonstrate how to capture a large degree of parallelism and to make use of the scalability advantages of spatial architectures, what is needed is a demonstration of the process of developing a mapping of biological neurons to a spatial architecture, and the evaluation of the performance and resource scalability.

Problem 3: Soft Error Protection Techniques Designed Without Application Knowledge Present Unclaimed Design Margins Existing methods for soft error protection do not consider the application-level characteristics, and apply protection techniques uniformly. Consequently, a significant design margin exists for operations that are inherently resilient, and do not need the additional hardware-level protection. To give hardware designers a tool for improving DNN accelerator reliability, what is needed is a methodology to for identifying opportunities to leverage DNN resilience to reclaim design margins in the implementation of soft error protection techniques, and to make tradeoffs between reliability, power and performance.

To address the identified problems, we propose three tasks that will study performance, scalability, power, and resilience optimizations of machine learning workloads on spatial architectures.

Task 1: Leverage Dynamic Workload Characteristics to Optimize Hardware Efficiency of Spatial Architectures To make the case for dynamically configurable DNN hardware, in this task, we will study the workload characteristics of several DNN workloads. To address the challenge of low power DNN acceleration, and to give hardware designers tools to make tradeoffs between power and performance in DNN accelerator design, we will introduce a reconfigurable hardware paradigm, and will develop a methodology for systematic exploration of the design space. We propose a methodology for the design of reconfigurable DNN hardware that determines the best configuration for each DNN layer, while considering reconfiguration cost, across costs that consider both ASIC and FPGA platforms to improve latency and energy efficiency.

Task 2: Exploit Resilience to Accelerate Biological Neurons on a Spatial Architecture This task will study the correspondences between Hierarchical Temporal Memory and the cell-like spatial computing

fabric of the Automata Processor. To address the memory bottleneck problem in traditional memory hierarchies, we will propose a methodology for realization of Hierarchical Temporal Memory as automata. We will identify correspondences between these two computing paradigms, develop an efficient mapping, and evaluate performance and resource scalability. The result of this work will be a tool that will allow HTM application designers to synthesize, simulate, and validate performance of HTM applications with the Automata Processor.

Task 3: Exploit Resilience to Optimize Hardware Efficiency of Resilient Spatial Architectures To address the challenge of optimizing the performance and power metrics in resilient DNN accelerators, in this task we will develop methods for optimization of soft error protection techniques based on the criticality of neurons in a DNN. We will develop methods for sensitivity characterization, applying selective soft error protection based on a classification of neuron sensitivity, and simultaneous optimization of application accuracy and hardware resources. The result of this work will be a tool for making tradeoffs between resilience, power, and performance in the presence of faults to give hardware designers a tool for the design of efficient, resilient DNN accelerators.

The remainder of this proposal is organized as follows. Sections 3, 4, and 5 introduce the three areas of research as part of this program. Each of these sections contains four subsections. Subsection 1 provides a problem definition. Subsection 2 a research hypothesis. Subsection 3 highlights major contributions. Subsection 4 provides further motivation of the problem. Subsection 5 introduces the proposed approach, and highlights preliminary results that justify the insights and provide risk minimization for the overall project. Subsection 6 states evaluation criteria to determine success in the project. In Subsection 7, we present an analysis of related work, and differentiate our approach from previous approaches the defined challenges. Finally, in the Appendix, we document a summary and plan of work that details completed work and work left to be done.

3 Task 1: Hardware Optimization Based On Workload Characteristics

3.1 Problem Definition

Existing custom DNN hardware accelerators encompassing low-power ASICs [67, 11, 23], FPGAs [16], and large-scale accelerator systems [79, 12, 42] are specialized to leverage the fine-grained data parallelism, dataflow structure, and communication patterns in DNNs to achieve substantially better energy–efficiency over CPUs and GPUs. DNNs exhibit significant heterogeneity in their computational characteristics, *i.e.*, feature and kernel dimensions, and dramatic variances in computational intensity, even between adjacent layers in one DNN. However, a key limitation of these architectures is that their microarchitectural parameters are *statically* determined at design time, and consequently, they run sub-optimally and leave energy–efficiency margins unclaimed.

The key idea of the proposed approach is to reconfigure the microarchitectural parameters of the accelerator dynamically, based on the characteristics of the layer under execution. Consequently, performance and energy benefits can be achieved beyond statically configured accelerators with different reconfiguration overheads taken into account. The costs of augmenting the accelerator with the ability to reconfigure depends on the implementation scenario. For example, in the case of ASIC designs, under an iso-power constraint, reconfiguration involves clock/power-gating computation slices or interconnect links based on DNN layer dimensions. In contrast, reconfiguration in FPGA designs involves loading a new full or partial bitstream.

The insight enabling this idea is that layer dimensions are known before execution begins, so the best configuration for each layer can be determined offline. To this end, we will develop a methodology that in the *offline phase*, takes in a description of the DNN layers, the microarchitectural parameters of an accelerator augmented with *run time control logic*, a performance model that quantifies its run time, and the costs of reconfiguration. It explores the space of possible configurations to construct a *configuration map* with the best per-layer configurations, while minimizing the metric of interest, despite overhead cost. Then in the *execution phase*, configuration logic will enforce the configuration map to dynamically improve latency and energy–efficiency.

Table 1: VGG11 Parameters and Characteristics

	Parameters				Compute Characteristic		
Layer	N_{in}	N_{out}	N_{ij}	K_{ij}	Ops (M)	Bytes (M)	Ops/Byte
CONV_1	3	64	224×224	3×3	173.41	6.73	25.78
CONV_2	64	128	112×112	3×3	1849.69	4.96	372.59
$\text{CONV}_{3,1}$	128	256	56×56	3×3	1849.69	3.00	616.92
$\text{CONV}_{3,2}$	256	256	56×56	3×3	3699.38	4.39	842.51
$\text{CONV}_{4,1}$	256	512	28×28	3×3	1849.69	3.56	519.06
$\text{CONV}_{4,2}$	512	512	28×28	3×3	3699.38	6.32	584.95
$\text{CONV}_{5,1}$	512	512	14×14	3×3	924.84	5.12	180.63
$\text{CONV}_{5,2}$	512	512	14×14	3×3	924.84	5.12	180.63
FC_6	25088	4096	1×1	1×1	205.52	205.58	1.00
FC_7	4096	4096	1×1	1×1	33.55	33.57	1.00
FC_8	4096	1000	1×1	1×1	8.19	8.20	1.00

3.2 Research Hypothesis

In proposing this work, we hypothesize that *if* DNN accelerator hardware is dynamically reconfigured according to the workload characteristics of each layer, *then* a configuration map will be found that allows the accelerator to execute the workload with better runtime characteristics than under a static conditions.

3.3 Contributions

This research will make the following contributions:

- To improve the energy-efficiency of DNN accelerators, we will develop a new approach for dynamically configuring accelerator microarchitectural parameters that exploits the diversity in the computational characteristics of DNN layers.
- We will describe a DNN accelerator optimization methodology, consisting of: (i) an *offline* phase, wherein the configuration map to execute each layer is identified, considering costs of reconfiguration, and (ii) a *run time* phase, wherein the configuration map is enforced as the DNN layers execute on the accelerator
- To identify optimization opportunities for accelerator designers, we will study the latency–energy trade-off space by dynamic reconfiguration of microarchitectural parameters, across a range of overhead costs modeling ASIC and FPGA platforms.

The proposed methodology will be evaluated in the context of a SIMD 2D systolic array architecture with an on-chip scratchpad memory, synthesized to a commercial 14 nm low power technology [79].

3.4 Motivating Dynamic Spatial Array Hardware

In this section, we motivate the need for dynamic reconfiguration by analyzing a popular DNN benchmark and identify opportunities for improving performance and efficiency.

3.4.1 Diversity in DNN Workloads

To further motivate the proposed approach, Table 1 shows the dimensions and compute characteristics of each layer in the VGG11 image classification DNN [70]. VGG11 contains eight CONV and three FC layers. Although each layer performs the same type of operation, either 2D-convolution or matrix-vector multiplication, *data-structure shape and Ops/Byte requirements vary dramatically*.

We observe from Table 1 that the initial CONV layers are shallow and stout, *i.e.*, they have very few input and output features (N_{in} and N_{out}), but the feature dimensions (N_{ij}) are quite large. As we progress to the final layers, N_{in} and N_{out} increase by over two orders of magnitude, while N_{ij} decreases proportionately. In the case of more recent DNNs, *e.g.*, GoogLeNet [73] and ResNet [38], even K_{ij} varies more

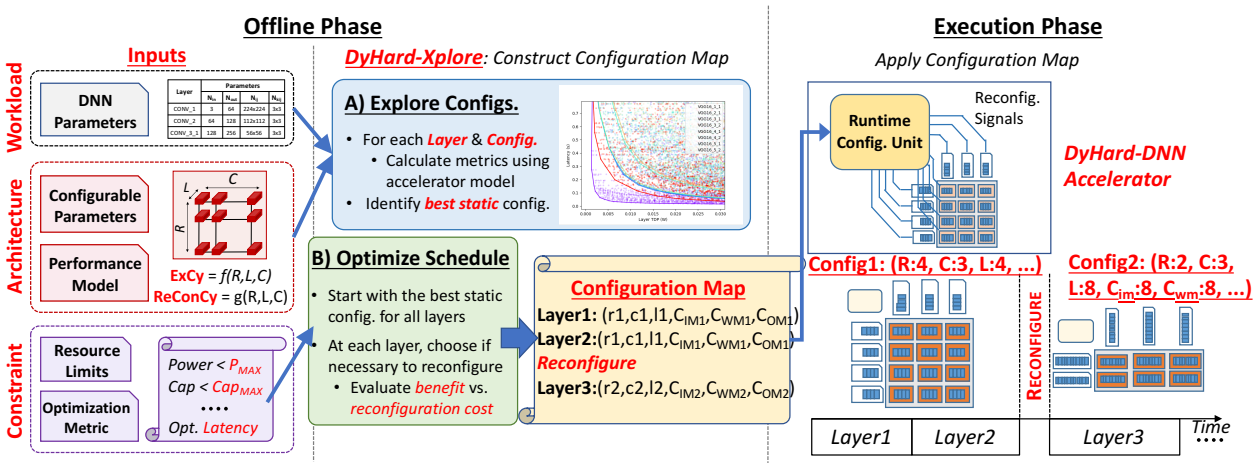

Figure 1: Overview of DyHard-DNN methodology

frequently, between 7×7 to 1×1. The dimensions directly impact the OPs/Byte of each layer. The CONV layers are dominated by sizes of the input and output volumes, while FC layers are dominated by weights. Also, CONV layers offer abundant opportunities for data-reuse, while FC layers are limited by memory bandwidth. We find the OPs/Byte requirement to vary by over an order of magnitude even within the CONV layers, and by over two orders of magnitude overall.

3.4.2 Limits of Statically Configured Hardware

Workload diversity translates to a loss of utilization at design-time in statically configured accelerators due to two main factors.

Ops-to-byte ratio DNN accelerator designers allocate resources for compute, memory and interconnect such that performance is optimal for common layers. When the layer has abundant data-reuse, (*e.g.*, CONV layers with $\text{Ops/Byte}_{\text{layer}} > \text{Ops/Byte}_{\text{hw}}$), the accelerator is under-provisioned in compute, *i.e.*, resources could be shifted from memory bandwidth to compute, by powering up/down execution engines and interconnect channels. In contrast, layers with poor data-reuse (*e.g.*, FC layers $\text{Ops/Byte}_{\text{layer}} < \text{Ops/Byte}_{\text{hw}}$) underutilize compute, as they are unable feed data to the processing elements at a sufficient rate. Both scenarios result in unclaimed performance and energy margins.

Residual effects The main computational work in DNN accelerators is done by a multi-dimensional array of compute elements. If a hardware dimension is not an even multiple of the workload dimension, it results in a utilization drop. Imbalanced work divisions along each dimension have a multiplicative effect, and the performance drop builds up very quickly.

3.4.3 Reconfigurable SIMD Systolic Arrays

We illustrate the benefits of dynamic reconfiguration using a generic SIMD 2D systolic array architecture common in DNN hardware literature [79, 11, 12], such as shown in Figure 2. Specifically, we consider the embodiment proposed in ScaleDeep [79], wherein the architecture is comprised of a 2D array of processing elements (PE), with each PE connected to its neighbors. The PEs are a SIMD array of multiply-accumulate units. The number of rows (R), columns (C) and SIMD lanes (L) define the compute capability of the architecture. The accelerator also contains three scratchpad memories, one each to store the input, weight and output data structures. The capacities and bandwidth with which the PE array is interfaced to the input, weight, and output memories are given by Cap_{IM}, Cap_{WM} and Cap_{OM}, and BW_{IM}, BW_{WM} and BW_{OM}, respectively. Within the scope of this work, we enable run time reconfiguration of the compute array dimensions, memory capacity and interconnect bandwidth.

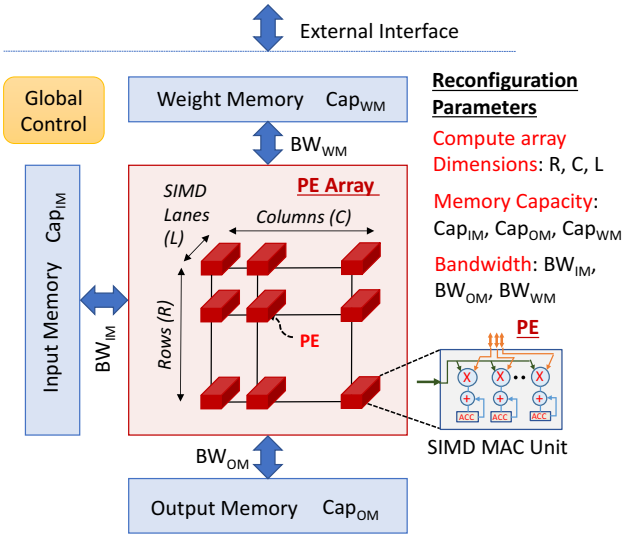

Figure 2: A SIMD 2D systolic array accelerator

3.5 Proposed Approach

Figure 1 shows an overview of the proposed approach encompassing *offline analysis* and *run-time execution* phases. An analysis tool will determine optimal microarchitectural parameters on a layer-by-layer basis and will construct an optimized *configuration map* that specifies per-layer run-time configurations, incorporating inter-layer reconfiguration cost. During execution, free microarchitectural parameters will be bound to values specified by the configuration map.

Within this study, we will develop the following components:

- **Reconfigurable SIMD Systolic Array Architecture** The reconfigurable systolic array platform will define and identify the bounds of reconfigurable parameters, *e.g.*, rows, columns, SIMD lanes, *etc.*, that will be available to the optimization methodology.
- **Optimization Methodology** The optimization methodology will consist of an offline phase, in which a configuration map is constructed for each hardware design candidate. In the runtime phase, the methodology will dictate how the systolic array will be reconfigured to meet the target metrics of interest.
- **Optimization Tool** The optimization tool will construct a configuration map from the dimensional parameters of a DNN layer. These will be used to generate a sequence of configurations while considering total physical design constraints and the cost of reconfiguration on a given circuit substrate. This tool will also facilitate evaluation of the performance and power of a given schedule of execution on the proposed reconfigurable systolic array architecture.

3.6 Evaluation Criteria

We will demonstrate dynamic reconfiguration in the context of the 2D systolic array architecture using simulation. We will consider two concrete scenarios where DyHard-DNNs could be deployed: (i) an embedded ASIC implementation and (ii) programmable SoCs with embedded FPGAs, *e.g.*, Intel Harp or Xilinx Zync. Success in this project will be achieved if simulation shows that dynamic configurations achieve better performance or power options than static when taking into account the costs of reconfiguration.

3.7 Related Work

Related work involves both accelerator optimization as well as quantization and compression.

DNN Accelerators and Dataflow Optimization To improve the compute efficiency of DNN, a wide variety of accelerator architectures span the spectrum from low-power IP cores [23, 11, 9, 5, 33, 67, 25] to large-scale systems [79, 42, 12]. Almost all these efforts exploit the reuse pattern in convolutions and matrix-multiplications using a 2D-array of processing elements and an associated data-flow. Some key architectural ideas proposed in these efforts include the use of heterogeneous processing cores for multiply-and-accumulate *vs.* special-function operations, software-managed distributed memory hierarchy, on-chip and off-chip interconnect topologies that match the DNN's communication patterns, exploiting the sparsity in computations, among others.

Very few efforts have proposed optimizations considering the heterogeneity in compute characteristics across layers in DNNs. For example, Eyeriss [11] modulates the dataflow in software between weight stationary *vs.* row stationary depending on layer characteristics. ScaleDeep [79] builds two types of chips to realize CONV and FC layers respectively. In contrast, for the first time, we propose to dynamically reconfigure hardware parameters, redistributing resources between compute, memory and bandwidth, determined by the layer's requirements.

Quantization and Compression Techniques such as reduced precision implementations [67] and model compression [33] for improving an accelerator's efficiency by leveraging its error resilient nature have been extensively explored. These efforts are complementary to DyHard-DNNs, *i.e.*, they can be applied on top of dynamically reconfigured hardware for further benefits.

4 Task 2: Acceleration of Biological Neurons as Automata

4.1 Problem Definition

Neuroplasticity in the brain allows it to learn and to recover cognitive ability following changes at levels ranging from individual neurons to entire cortices [2]. Hierarchical temporal memory (HTM) is a machine learning model that mimics biological neurons with high accuracy, designed to capture brain-like cognitive capabilities of the neocortex in software [37, 36]. HTM can autonomously learn to predict future states from a stream of temporal data, and has demonstrated aptitude in several prediction and anomaly detection applications [22, 50, 4, 8]. Additionally, its biomimetic properties make it resilient to changes such as pruning neurons and synapses [57, 3].

CPU-based simulators for spiking networks such as Compass [65] and SpiNNaker [62] use von Neumann-based architecture, which is a fundamental bottleneck for large-scale, massively interconnected neural models. Specialized neural network accelerator hardware, such as the IBM TrueNorth [54] and the Qualcomm Zeroth [46] platforms, employ concepts that are direct analogues from biology, and use spatial architecture to mitigate memory bottleneck issues. Existing accelerators customized for HTM so far implement a quantity of cells not capable of supporting real-world sized applications [28].

The Automata Processor (AP) is a silicon implementation of non-deterministic finite automata that can be programmed to recognize patterns in a stream of temporal data [20]. Several natural correspondences between the execution model of the AP and of HTM suggest that an efficient mapping could capture significant speedups and desirable scalability due to its highly parallel execution model [66]. Despite these correspondences, a key challenge of implementing HTM in the AP is that the lack of arithmetic capability of the AP means that it cannot natively support core HTM algorithms. Furthermore, the costs of direct line on-chip communication modeling inter-cellular activations place an upper bound on the total number of cells that can be modeled, consequently limiting its real-world applicability.

The key idea of the proposed approach is to map the cell model of HTM to the AP to take advantage of the massive automata parallelism provided by its many parallel match elements to capture significant speedups over existing accelerators. Instead of spending time coalescing data from sparsely distributed synapses before computing activations for each neuron as in the von Neumann model, the AP model receives a single, global data stream, and each cell computes its activation independently. This approach essentially eliminates marginal cell bandwidth and scalably facilitates full-size models that can support useful, real-world HTM-based applications.

This approach is made possible by three key insights: (i) the resilience of the core dendritic segment activation algorithm can be exploited to *approximate* it as a decision problem, allowing cells to be realized

as independent, counter-enhanced automata (CEA) on the AP, eliminating the von Neumann bottleneck to capture speedups, (ii) inter-cellular communication can be separated into an off-chip phase, eliminating thread divergence, and allowing linear space complexity, and (iii) HTM's sparse connection properties can be leveraged to achieve average-case linear temporal scalability when synaptic connections are converted into temporally-spaced communications.

4.2 Research Hypothesis

In proposing this work, we hypothesize that *if* changes are made to the core HTM algorithms to make it compatible with implementing it in the AP, *then* despite the necessary approximations, applications using the resulting model will have significantly better throughput and more desirable resource scalability, while maintaining near constant application accuracy with the baseline.

4.3 Contributions

This research will make the following contributions:

- To demonstrate functionality and to show how the AP can be used as a specialized FPGA, we will introduce a model of HTM neurons as CEA that exploits the correspondences between the HTM neural model, CEAs, and the AP spatial architecture.
- To facilitate the synthesis, simulation, and validation of general HTM models as CEA, we will develop a general methodology and associated software tools to map trained HTM models to the AP architecture, and dataflow tools for pre- and post-processing inputs and outputs.
- To validate the potential speedups and scalability benefits and to facilitate comparisons with related work, we will simulate several appropriate HTM benchmarks and develop resource scalability models.

To evaluate these ideas on the AP architecture, we will use Micron's ANML structural design language for the AP, and will use the AP toolchain that provides a software simulator. Additionally, results will be validated using the VAsim [81] toolchain.

4.4 Motivating HTM Models As Automata

In this section, we motivate the realization of HTM models as automata by highlighting key correspondences between HTM and AP execution models.

4.4.1 Key Correspondences

Cell activation HTM cells and the AP's STE elements have similar state transition models. When STEs are activated by other elements, they "listen" to the input stream, and report when a matching string is seen. Similarly, HTM cells are enabled by external inputs through column dendrites and activate through lateral connections on local dendrites if they were predicted to do so by cells in a previous epoch.

Streaming dataflow model Both STEs and HTM cells operate on continuous streams of data. The AP natively implements regular expressions, which take strings of characters as input. Matches can also internally activate other elements. Similarly, in HTM, a stream of SDRs causes column activations, which activate cells whose lateral activations predict upcoming inputs. In both memory paradigms, input sequences are matched against stored patterns, which both constitute outputs and laterally enable ensuing elements to listen to the input and continue processing.

SDR bit overlap similarity Similarity between SDRs is determined by counting the number of overlapping bits. An analogous operation in the AP can be implemented by using STEs with symbol sets programmed for active bit indices, attached to a counter that accumulates the number of overlapping bits as the SDR active bit indices are provided on the input.

Model parallelism The AP can be programmed with a single automaton, or many independent, parallel automata, that all operate on the same input stream, and internally communicate state through activations.

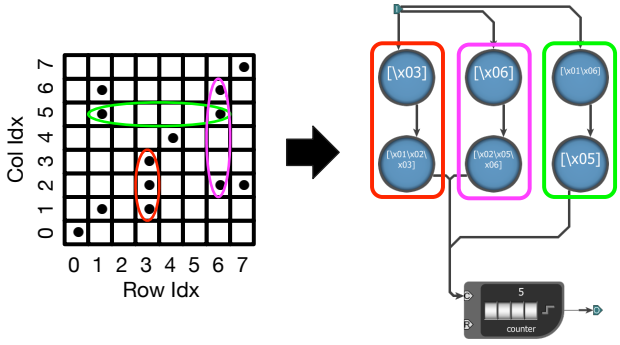

Figure 3: Example of an AP segment automaton construction. Connections between cells can be expressed as a sparse matrix (left). This sparse matrix lookup table is implemented in the AP with parallel automata that are OR-ed together and counted (right). When active connections are scanned in on the input, the counter's threshold is crossed if a sufficient number of connected synapses are present. Using two 8-bit symbols on the input, this type of automaton can model connections from any number of up to 65,536 cells.

Similarly, HTM columns and cells operate independently and communicate cell state laterally through synaptic connections.

Hierarchy Connection among multiple APs is flexible, which can be configured to all receive the same input stream, or multiple independent streams. Using intra-rank bussing, multiple APs could be configured to act as multiple HTM regions in a hierarchy.

4.4.2 Counter-enhanced Automata HTMs Key Insights

To motivate HTM models as counter-enhanced automata, we highlight design methods that leverage correspondences between the AP and HTM to convert HTMs into automata that are compatible with the AP's execution model.

Dendritic segment activation as decision problem HTM column and cell state variables (i.e., active, predicted, bursting, etc.) are binary, and next-state values of these variables are computed using counting and simple Boolean functions. STEs and counters can be used to store and count these variables, and the inherent Boolean properties of STEs can be used to implement next-state transition functions. For example, automata states in the AP natively `OR` their inputs, and automata naturally implement the `AND` function by matching when a state is connected to the input and it receives a matching input.

Virtual synaptic connections Implementing synapses with physical connections between STEs in the AP would not be scalable due to practical routing and fan-out limits. Instead, to take advantage of the MISD organization of the AP, cells can be connected "virtually" between epochs via the input stream, taking advantage of the high-bandwidth on-chip broadcast network on the AP.

Dendrite segments as automata HTM cells activate if any of its dendrite segments received a sufficient number of active synapses, of which the threshold is determined as part of HTM learning. The AP's counter element counts STE activations and reports if a threshold is crossed, and is a natural way of counting activations to determine if a cell was predicted. The automata connected to the counter implement a sparse connection matrix, as shown in Figure 3. With unique (row, col) IDs assigned to each cell, connections between the output of that cell and the segment are encoded in the symbol set of the STE elements. The segment recognizes an activation from any other cell in the HTM. The first STE matches on the row ID, and the second matches on the col ID.

To construct the automata in an efficient manner, we take advantage of the parallel lookup property of STEs, and use a method similar to Karnaugh maps. For example, to represent connections from cells $((3,1),(3,2),(3,3))$, the row STE is programmed to recognize the symbol set $(3,)$ and the col recognizes $(1,2,3)$. It does not matter if the same (row, col) ID is encoded in two different groups because counters `OR` their inputs and increment only once, regardless of the number of simultaneous activations.

The input data stream is then constructed of a string of (row, col) IDs of cells that activated in the previous epoch. At the end of the input tape, counters that crossed their thresholds mean a sufficient number of

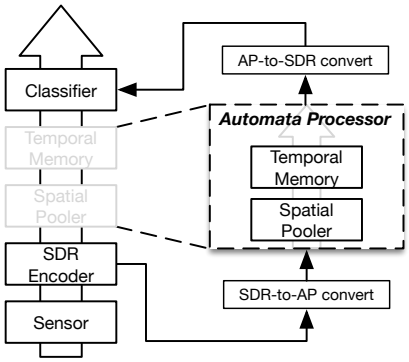

Figure 4: Overview of the HTM stack with the SP and TM layers offloaded to the AP.

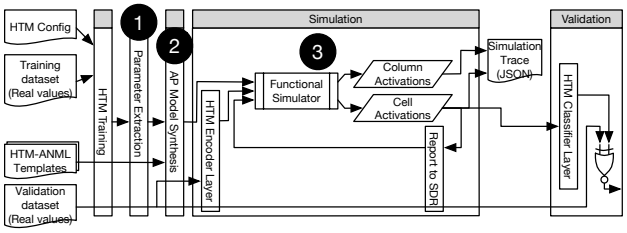

Figure 5: Overview of the Dendroplex components for synthesis, simulation, and validation of HTM models in the AP.

active connections were received, indicating the associated cell was predicted.

Although cells in HTM can potentially connect to any other cell in the region, due to the sparseness property, at most, this is about 2% of the total number of cells, and in practice, it is far less. This ensures that the size of each segment in the AP is kept within reasonable bounds.

Winner selection by thresholding Only those cells in columns with active segments from the input above a certain threshold are eligible to activate in the temporal memory phase. Although similar to the temporal memory dendrite activation method, the HTM spatial pooler algorithm selects column activations based on a k-winners-take-all algorithm, which necessarily requires sorting. Since sorting cannot be implemented efficiently on the AP, the global inhibition algorithm as part of the spatial pooler was modified to eliminate this dependency and to facilitate parallelization. Although sometimes resulting in incorrect column activations, the resilience property of HTM ensures that good overall accuracy is preserved.

4.5 Proposed Approach

Figure 5 shows an overview of the proposed approach. Parameters from a trained HTM will be synthesized into a model suitable for simulation and compilation on the AP, in the ANML structural definition language. The HTM stack will be modified to include the AP-based core HTM functions in the flow while surrounding pre- and post-processing layers will continue to reside on the host, and data translation layers will be added.

The proposed project will consist of three parts:

- **HTM Parameter Extractor Tool** This tool will take a trained HTM network as an input and output a set of cell labels and edges between cells that define the structure of the the HTM.
- **HTM-ANML Compiler Tool** This tool will take the extracted structural definition of the HTM and will construct an ANML model of the trained HTM that is suitable for simulation.
- **Simulation and Validation Methodology** This set of tools will contain three tools: (i) for converting sparse distributed representation inputs for an HTM benchmark to an AP input tape, (ii) for converting sparse distributed representation inputs into real-valued outputs, and (iii) for feeding inputs to the AP model under simulation and for capturing outputs.

4.6 Evaluation Criteria

A performance improvement will be shown if the resulting HTM-AP model completes the same core algorithm functions, including overhead, in less theoretical cycles than the baseline CPU model, using suitable benchmarks. Evaluation of the theoretical scalability of the model will determine the number of hardware resources that are necessary as a function of the number of synapses, dendrites, and cells in the HTM model. Good scalability will be achieved if the resulting design is bottlenecked by the more abundant STE components, rather than the more limited counter or logic elements of the AP.

4.7 Related Work

Previous works have introduced accelerator architectures for HTM. Billaudelle and Ahmad [6] port core HTM functions to a spiking neural network ASIC and verify that it meets basic HTM properties. Zyarah [87] demonstrates impressive speedups in an FPGA implementation using small (100 cols., 300 cells) HTMs against a MATLAB baseline implementation. An additional work by Deshpande [19] builds an FPGA-based implementation of principle component analysis inspired by HTM.

The major difference between related works and ours is that we will validate our model with HTMs of non-trivial size (up to 1804 columns, 44k cells), using real application benchmarks. Additionally, we will evaluate the communication overheads and scalability of our approach as a function of the number of synapses, dendrite segments, and cells of the baseline HTM model.

5 Task 3: Hardware Optimization Based On Workload Semantics

5.1 Problem Definition

DNNs are increasingly being used in mission-critical applications, such as autonomous vehicle control systems, banking, and medical diagnosis. Consequently, robust system design in the presence of soft errors in DNN accelerator architectures is a concern for applications ranging from resource-constrainted embedded to large-scale cloud platforms. Such faults originating from high energy particle strikes or voltage noise, can propagate to the application if their effects are latched, and can cause errors if they are not detected [7]. Of the components in a DNN accelerator, SRAM is especially susceptible, due to its greater sensitivity to process, voltage, and temperature variation than logic [56].

Designers use a variety of soft error detection and correction techniques to mitigate the effects of faults in static memory [56]. Since technology scaling has lowered the energy threshold needed to cause upsets, error correction codes (ECC) are becoming insufficient to mitigate the growth in such events in SRAM as process scaling has progressed [30]. Selective precision scaling is also be used to improve resilience by allocating bit depth based on neuron criticality [80, 44]. Circuit-level hardening and physical redundancy approaches come at the cost of significant area overhead, and the associated power costs and reduction in performance. Replication techniques such as N-modular redundancy (NMR) and redundant multithreading (RMT) are classical approaches for soft error protection through replication of vulnerable operations, that come at the cost of additional latency, power, and area. Comprehensive resilience methodologies, such as CLEAR can offer cross-layer resilience [14], optimizing based on the best technique for each design layer.

Any number of these techniques can be applied to significantly improve the resilience characteristics of DNN accelerators. However, a key limitation of doing so without knowledge of application-level characteristics is that soft error protection techniques are applied uniformly. Consequently, significant power, area, or latency costs are incurred, reducing the efficiency of the hardware.

Heterogeneous Neural Sensitivity In contrast, we observe two key insights: (i) that the contribution of the output among neurons to the accuracy of the application is non-uniform, and (ii) that non-critical neural operations provide a degree of fault masking for free, allowing savings by reducing or eliminating error protection where possible. Not all of the features in the input contribute the same amount to the accuracy of the output. Consequently, the same error magnitude on the output of some neurons may contribute widely varying amounts to a change in the accuracy of the application, and is therefore more or less critical.

Selective Application of Soft Error Protection The key idea in our approach is to make more efficient use of myriad soft error protection techniques in DNN accelerators by determining the minimum necessary protection factor needed for each neuron according to its criticality. Variation in sensitivity can be used to rank neural operations in order of criticality, and to selectively guide the application of soft error protection. After this optimization, when transient faults occur, their effects on high criticality neurons are masked by the protection technique, while their effects on low criticality neurons are inherently masked by the application. As a result, efficiency over uniform application is achieved by the significant reduction in circuit power, area, or latency while the overall application resilience benefits are preserved. And since neural sensitivity is already determined during back propagation, the overhead of this step is natively built in to the DNN training process.

5.2 Research Hypothesis

In proposing this work, we hypothesize that *if* soft error protection is applied to neural operations in a DNN accelerator according to their average criticality in the application, *then* when the resulting DNN is executed using an appropriate validation dataset, resilience and accuracy characteristics of a comparable uniformly applied protection technique will be preserved, with improved hardware runtime metrics.

5.3 Contributions

- We characterize the resilience of several DNN applications in the presence of bit faults in a range of fault rates.
- We propose a novel approach for exploiting the diversity in neural operation criticality to significantly improve the efficiency of soft error protection techniques in DNN accelerators.
- We develop a resilience optimization methodology for determining the degree of soft error protection needed for each neuron to meet a set of design constraints.
- We study the energy-resilience tradeoff space made possible by soft error protection techniques across a range of microarchitectural parameters that models embedded and cloud platforms, giving application designers opportunities to make tradeoffs between circuit-level metrics and application accuracy.

We will evaluate this methodology for real-world DNN applications using full validation datasets, and characterize the distribution of neural sensitivity for each. To evaluate the costs and benefits of the methodology, we will use replication and majority voting as an implementation of soft error protection.

5.4 Motivating Non-uniform DNN Redundancy

In this section, we will motivate the need for applying non-uniform redundancy in the context of DNN applications for significantly improving the runtime metrics of spatial array DNN accelerators.

5.4.1 Neural Network Training Background

During the training process, the weights of neurons are adjusted so that the output of the network matches the expected output, and in so doing, it is iteratively trained to maximize accuracy. Training consists of two steps: (i) inference and (ii) backpropagation. During *inference*, or forward propagation, an input is provided to the network and the output is calculated by propagating linear combinations of the inputs and weights forward through a sequence of layers in the network until the final output is calculated. The error is then calculated between the predicted output and the expected output of the network. During *backpropagation*, the contribution to the output error of every neural input is calculated so that the weights can be adjusted to minimize the error. Equation 1 as part of backpropagation, provides a method to compute a neuron's weight error (∂w), and to relate the contribution of that error to the final output error (∂C), based on the output of the neuron (a), and the error at the output of the neuron (δ) [61].

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{1}$$

5.4.2 Non-uniformity in DNN neuron criticality

Non-uniformity of the criticality of neurons is granted by the expectation of non-uniform spatial locality present in a real dataset. For example, if a neuron is connected to a region of input that rarely contains any features, its output will have a lesser impact on the final output, than one that often sees important features. The weight of the neuron determines how much of its input it will pass to the output, and ultimately to the final output. The process of training adjusts weights on neurons such that those connected to parts of the input with features that correlate highly to the correct output contribute more strongly to the output. Therefore, we intuitively expect that neurons with more error in their weights are those that contribute non-critical information, and *vice versa*.

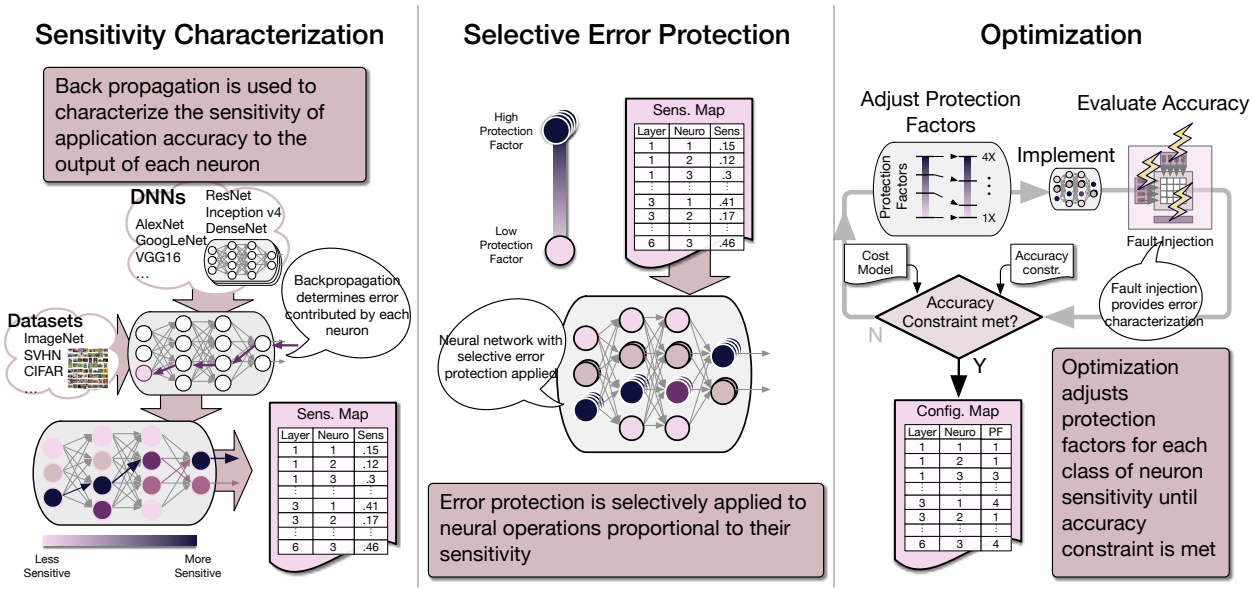

Figure 6: Overview of Doggett methodology

5.4.3 Disadvantages of uniformly applied error protection techniques

Existing soft error techniques applied uniformly leave unnecessary design margin for neurons that do not need the added protection. Non-critical neurons receiving the same amount of protection as critical neurons leaves unclaimed design margin. For example, although a fault in the weight value of a non-critical neuron may produce an error on its output, its output value will not be weighed as heavily on the path to the final output. Such a neuron does not need protection. A neuron that has low weight error means that the network considers its output important to the final output, and an error in the output of such a neuron would contribute a large error to the final output.

5.4.4 Selective soft error protection

Using neuron sensitivity as a proxy for resilience to direct the application of soft error protection, soft error protection is then applied to neurons proportional to their criticality. For example, using replication as a protection technique, neurons requiring the highest protection would be classified into categories requiring none, three, or four times replication. As a result, savings at the circuit level are claimed by using less power, area, or completing operations in less time, while maintaining accuracy at an equivalent fault rate.

5.5 Proposed Approach

Figure 6 shows an overview of the proposed approach for characterization and optimization of selective application of soft error protection for resilient DNN accelerators.

5.5.1 Sensitivity Characterization

To conduct sensitivity characterization, using a method similar to that outlined in [80], for each example in the training set, we can collect the error of the weights and biases of each neuron to compute the average error contributed over the entire dataset. The resulting average error contribution can then be used as a proxy for the sensitivity of the output accuracy to each respective neuron, and can be used as a key to sort and classify the neurons into a sensitivity map.

5.5.2 Selective Error Protection

The purpose of the selective error protection phase is to calculate resource usage based on the initial classification of neurons into protection factors. As part of the initialization step, each soft error protection technique defines its own cost of implementation. For example, a precision-scaling technique would require hardware overhead for larger registers. In the case of redundant multithreading, increased protection is achieved through multiple execution of a neural operation in time, followed by a majority voter, incurring a latency cost associated with repeated execution.

5.5.3 Optimization

Then during the optimization phase, given an accuracy requirement, a fault rate upper bound, and a fault protection cost model, the tool performs fault injection to evaluate the effects, and systematically minimizes the amount of protection needed for each neuron. This process is repeated until the either the accuracy and resource constraints are met, or a solution cannot be found. The output of this tool provides a mapping from each neuron in a DNN to the degree of protection that must be applied to it to meet design constraints.

Within this study, we will develop the following components:

- **Evaluation of the Effect of Bit Faults on DNNs** Several popular DNN workloads will be evaluated with full validation sets to evaluate the effects of faults on their accuracy and performance
- **Resilience Optimization Methodology** The optimization methodology will consist of a mapping from neuron criticality to protection factors. A further optimization step will determine how much protection factor to be applied to each neuron to meet accuracy and resource constraints.
- **Resilience Optimization Tool** The optimization tool will construct a sensitivity map from the analysis of criticality of neuron operations in the DNN. These will be used to generate a set of configurations while considering total physical design constraints and the cost of soft error protection on an example circuit substrate. This tool will also facilitate evaluation of the performance, power, and accuracy of a given soft error protection configuration map.

This methodology can be used to study and mitigate the effects of faults originating from one or multiple sources, each with a unique fault distribution. Furthermore, an extension of this work could be used to prioritize the fault resilience and associated hardware metrics of output classes, depending on their importance to the application.

5.6 Evaluation Criteria

Success in this project will be determined if The application accuracy will be within 10% of the baseline, while the runtime characteristics (*i.e.*, power, latency) will be measurably better than if the same soft error protection technique had been applied uniformly across all neurons, as determined by simulation.

5.7 Related Work

Approximate Circuits Several previous works have proposed circuit reduction through precision scaling [78], logic reduction [32, 15, 27], approximating functions of less critical neurons [85, 80, 59], and caching or elimination of redundant operations [82, 34, 55]. The optimizations proposed in these works capture linear power savings by relying on either area reduction made possible by reducing the size of logic or number of gates needed to implement approximable functions, or runtime reduction by reducing the number of needed operations.

Exploiting DNN resilience to improve circuit performance and power Several papers [59, 47, 60] in this area introduce techniques for precision optimization as a means of saving power. As an example, Venkataramani et al. [80] introduce a methodology for making approximate the outputs of neurons with low impact on the output, and mapping these operations to corresponding approximate hardware to save power and improve performance. Furthermore, they observe that retraining can be used to recover some of the quality impact of neuron approximation. This work is significantly different from our proposed

approach because it exploits datapath width scaling as a means of improving circuit metrics, whereas our proposed approach uses deliberate timing violation and voltage scaling. As a result, our approach can achieve quadratic voltage scaling instead of linear.

Jiao et al. [41] develop a method for optimizing the output quality of DNNs in the presence of errors caused by circuit-level process, voltage, and temperature variations. They develop methods for simulating these effects by characterizing timing errors in an arithmetic circuit model and evaluating neural network performance in the presence of those errors. While the error characterization approach is similar from the proposed approach, this work is different because it does not consider ways to improve circuit performance and instead focuses on application metrics.

Zhang et al. [84] uses a Razor[26]-like timing speculation-based approach to detect and recover errors in MAC operations by inserting bubbles in the systolic array pipeline. This work differs from our approach in that it does not consider restricting the set of values used in the network to those that cause short propagation delays. Furthermore, it proposes delay penalties as a means of recovering errors instead of exploiting the resilience of DNN training.

Li et al. [51] analyze software techniques for improving the resilience of DNN accelerators. The main technique for improving the resilience of DNN accelerators introduced in this work is to harden latches following a bit-level sensitivity analysis. While it serves to identify the effects faults on errors in DNN applications, it is sufficiently different in that it does not recognize the correspondence between the vulnerability of neurons within DNNs and the vulnerability of hardware components.

A Summary of Tasks and Deliverables

A.1 Task 1: Hardware Optimization Based on Workload Characteristics

1. COMPLETE: Develop a methodology for the optimization of neural network accelerators based on its dynamic workload characteristics, and evaluation with suitable DNN workloads
2. COMPLETE: Design and implement a tool for optimization of spatial architecture hardware parameters based on the workload characteristics of DNN workloads
3. COMPLETE: Evaluate energy-delay and efficiency tradeoffs of DNN workloads across several spatial architecture variants

A.1.1 Deliverables

1. COMPLETE: Publish results in first-author conference or journal publication (DAC, 2018)
2. Open source release of codebase published to GitHub, with documentation

A.2 Task 2: Acceleration of Biological Neurons as Counter-Enhanced Automata

1. COMPLETE: Develop a functional model and methodology for synthesis of general HTM function in the identified spatial architecture
2. COMPLETE: Apply the methodology to implement HTM in the chosen architecture, and evaluate performance and scalability in comparison to von Neumann architectures using suitable benchmarks
3. COMPLETE: Design and implement a tool for the synthesis, simulation, and verification of general HTM models in AP architecture

A.2.1 Deliverables

- COMPLETE: Publish results in first-author conference or journal publication (IEEE MICRO, 2017)
- Open source release of codebase published to GitHub, with documentation

A.3 Task 3: Hardware Optimization Based On Workload Semantics

1. Develop an approach for characterization of neurons by criticality
2. Develop a methodology for determining the level of redundancy needed to achieve DNN application in the presence of faults
3. Design and implement a tool for optimization of DNN redundancy based on the criticality of DNN neurons

A.3.1 Deliverables

- Publish results in first-author conference or journal publication
- Open source release of codebase published to GitHub, with documentation

A.4 Dissertation Chapters

The expected main chapters of my dissertation are:

- Optimization of runtime characteristics with dynamic hardware reconfiguration
- Mapping Hierarchical Temporal Memory to the Automata processor
- Optimization of soft error protection techniques with DNN resilience

References

[1] Data, data everywhere. *The Economist*, February 2010.

[2] Neuroplasticity, February 2018. Page Version ID: 825984492.

[3] Subutai Ahmad and Jeff Hawkins. Properties of sparse distributed representations and their application to hierarchical temporal memory. *arXiv preprint arXiv:1503.07469*, 2015.

[4] Subutai Ahmad and Scott Purdy. Real-time anomaly detection for streaming analytics. *arXiv preprint arXiv:1607.02480*, 2016.

[5] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 1–13, June 2016.

[6] Sebastian Billaudelle and Subutai Ahmad. Porting htm models to the heidelberg neuromorphic computing platform. *eprint arXiv:1505.02142*, 5 2015.

[7] Jason A Blome, Shantanu Gupta, Shuguang Feng, and Scott Mahlke. Cost-efficient soft error protection for embedded microprocessors. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 421–431. ACM, 2006.

[8] Gerod M Bonhoff. Using hierarchical temporal memory for detecting anomalous network activity. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ENGINEERING AND MANAGEMENT, 2008.

[9] Srimat T. Chakradhar, Murugan Sankaradass, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *37th ISCA, June 19-23, 2010, Saint-Malo, France*, 2010.

[10] Y. H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, June 2016.

[11] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *J. Solid-State Circuits*, 52(1):127–138, 2017.

[12] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *47th Annual IEEE/ACM MICRO, Cambridge, United Kingdom, December 13-17, 2014*, 2014.

[13] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks, 2017.

[14] Eric Cheng, Shahrzad Mirkhani, Lukasz G Szafaryn, Chen-Yong Cher, Hyungmin Cho, Kevin Skadron, Mircea R Stan, Klas Lilja, Jacob A Abraham, Pradip Bose, et al. Tolerating soft errors in processor cores using clear (cross-layer exploration for architecting resilience). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[15] Vinay K Chippa, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Storm: a stochastic recognition and mining processor. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 39–44. ACM, 2014.

[16] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengil, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Christian Boehn, Oren Firestein, Aless andro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Tamas Juhasz, Ratna Kumar Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Steve Reinhardt, Adam Sapek, Ñaja Seera, Balaji Sridharan, Lisa Woods, Phillip Yi-Xiao, Ritchie Zhao, and Doug Burger. Accelerating persistent neural networks at datacenter scale. In *Symposium on High Performance Chips (Hot Chips), 2017*, 2017.

[17] Jeff Dean. Machine learning for systems and systems for machine learning. Advances in Neural Information Processing Systems 30, 2017.

[18] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[19] Mandar Deshpande. Fpga implementation and acceleration of building blocks for biologically inspired computational models. Master's thesis, PDX, 2011.

[20] Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3088–3098, 2014.

[21] Samuel Dodge and Lina Karam. Quality resilient deep neural networks, 2017.

[22] Joost van Doremalen and Lou Boves. Spoken digit recognition using a hierarchical temporal memory. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[23] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ISCA*, June 2015.

[24] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna Palem, Olivier Temam, and Chengyong Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 201–206. IEEE, 2014.

[25] Schuyler Eldridge, Amos Waterland, Margo Seltzer, Jonathan Appavoo, and Ajay Joshi. Towards general-purpose neural network computing. In *PACT 2015, San Francisco, CA, USA, October 18-21, 2015*, 2015.

[26] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 7. IEEE Computer Society, 2003.

[27] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE Computer Society, 2012.

[28] Deliang Fan, Mrigank Sharad, Abhronil Sengupta, and Kaushik Roy. Hierarchical temporal memory based on spin-neurons and resistive memory for energy-efficient brain-inspired computing. *IEEE transactions on neural networks and learning systems*, 27(9):1907–1919, 2016.

[29] Lex Fridman, Daniel E Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Julia Kindelsberger, Li Ding, Sean Seaman, et al. Mit autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *arXiv preprint arXiv:1711.06976*, 2017.

[30] Shrikanth Ganapathy, John Kalamatianos, Keith Kasprak, and Steven Raasch. On characterizing near-threshold sram failures in finfet technology. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 53. ACM, 2017.

[31] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[32] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. Impact: imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pages 409–414. IEEE Press, 2011.

[33] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254, June 2016.

[34] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[35] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[36] Jeff Hawkins, Subutai Ahmad, and Donna Dubinsky. Hierarchical temporal memory including htm cortical learning algorithms. *Techical report, Numenta, Inc, Palto Alto http://www. numenta. com/htmoverview/education/HTM_CorticalLearningAlgorithms. pdf*, 2010.

[37] Jeff Hawkins and Dileep George. Hierarchical temporal memory: Concepts, theory and terminology. Technical report, Technical report, Numenta, 2006.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE CVPR, Las Vegas, NV, USA, June 27-30, 2016*, 2016.

[39] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[40] Mohammad H Jafari, Ebrahim Nasr-Esfahani, Nader Karimi, SM Soroushmehr, Shadrokh Samavi, and Kayvan Najarian. Extraction of skin lesions from non-dermoscopic images using deep learning. *arXiv preprint arXiv:1609.02374*, 2016.

[41] Xun Jiao, Mulong Luo, Jeng-Hau Lin, and Rajesh K Gupta. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, USA*, 2017.

[42] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *44th ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 1–12, 2017.

[43] Stephen W Keckler, H Peter Hofstee, and Kunle Olukotun. *Multicore processors and systems*. Springer, 2009.

[44] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1742–1752, 2017.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS 2012, Lake Tahoe, NV, USA, December 3-6, 2012*, 2012.

[46] Samir Kumar. Introducing qualcomm zeroth processors: Brain-inspired computing, 2013.

[47] Jaeha Kung, Duckhwan Kim, and Saibal Mukhopadhyay. A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 85–90. IEEE, 2015.

[48] Sun-Yuan Kung. On supercomputing with systolic/wavefront array processors. *Proceedings of the IEEE*, 72(7):867–884, 1984.

[49] Sun-Yuan Kung, Bhaskar Rao, et al. Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers*, 100(11):1054–1066, 1982.

[50] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 38–44. IEEE, 2015.

[51] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 8. ACM, 2017.

[52] Jiwei Li, Michel Galley, Chris Brockett, Georgios P. Spithourakis, Jianfeng Gao, and William B. Dolan. A persona-based neural conversation model. In *54th ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[53] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[54] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[55] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C Rinard. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *ACM SIGPLAN Notices*, volume 49, pages 309–328. ACM, 2014.

[56] Subhasish Mitra, Norbert Seifert, Ming Zhang, Quan Shi, and Kee Sup Kim. Robust system design with built-in soft-error resilience. *Computer*, 38(2):43–52, 2005.

[57] James Mnatzaganian, Ernest Fokoué, and Dhireesha Kudithipudi. A mathematical formalization of hierarchical temporal memory's spatial pooler. *Frontiers in Robotics and AI*, 3:81, 2017.

[58] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference, 2016.

[59] Bert Moons, Bert De Brabandere, Luc Van Gool, and Marian Verhelst. Energy-efficient convnets through approximate computing. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–8. IEEE, 2016.

[60] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 488–493. IEEE, 2017.

[61] Michael A. Nielsen. Neural Networks and Deep Learning. 2015.

[62] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[63] Angshuman Parashar, Michael Pellauer, Michael Adler, Bushra Ahsan, Neal Crago, Daniel Lustig, Vladimir Pavlov, Antonia Zhai, Mohit Gambhir, Aamer Jaleel, et al. Triggered instructions: a control paradigm for spatially-programmed architectures. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 142–153. ACM, 2013.

[64] Roger Parloff. Why deep learning is suddenly changing your life.

[65] Robert Preissl, Theodore M Wong, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K Esser, William P Risk, Horst D Simon, and Dharmendra S Modha. Compass: A scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 54. IEEE Computer Society Press, 2012.

[66] Mateja Putic, AJ Varshneya, and Mircea R Stan. Hierarchical temporal memory on the automata processor. *IEEE Micro*, 37(1):52–59, 2017.

[67] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Y. Wei, and D. Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278, June 2016.

[68] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W Keckler, and Charles R Moore. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. In *ACM SIGARCH Computer Architecture News*, volume 31, pages 422–433. ACM, 2003.

[69] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[70] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[71] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.

[72] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[73] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015.

[74] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[75] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro*, 22(2):25–35, 2002.

[76] UC Berkeley EECS Events. Simon Knowles: Designing Processors for Intelligence.

[77] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4, 2011.

[78] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. ACM, 2013.

[79] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, and Anand Raghunathan. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *44th ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, 2017.

[80] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 27–32. ACM, 2014.

[81] Jack Wadden and Kevin Skadron. Vasim: An open virtual automata simulator for automata processing application and architecture research. *Tech. Rep. CS2016–03, University of Virginia*, 2016.

[82] Ali Yasoubi, Reza Hojabr, and Mehdi Modarressi. Power-efficient accelerator design for neural networks using computation reuse. *IEEE Computer Architecture Letters*, 16(1):72–75, 2017.

[83] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560. ACM, 2017.

[84] Jeff Zhang, Zahra Ghodsi, Kartheek Rangineni, and Siddharth Garg. Enabling extreme energy efficiency via timing speculation for deep neural network accelerators. 2017.

[85] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: an approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706. EDA Consortium, 2015.

[86] Yiren Zhou, Sibo Song, and Ngai-Man Cheung. On classification of distorted images with deep convolutional neural networks, 2017.

[87] Abdullah M. Zyarah. Design and analysis of a reconfigurable hierarchical temporal memory architecture. Master's thesis, RIT, 2015.