ABSTRACT

KARLI, PRATYUSH. Investigate and Report on ASIC Options for Implementing Hierarchical Temporal Memory. (Under the direction of Dr. Paul Franzon).

In recent years there has been growing interest towards the field of computational neuroscience in the pattern recognition community. Many biologically inspired cognitive models are devised based on theories about operation of the mammalian neocortex to effectively solve the problem of pattern classification. One such model is the Hierarchical Temporal Memory (HTM). The goal of this work is to review HTM architecture and compare it against other learning algorithms and to evaluate the efficiency and performance of a hardware implementation over single-core single-threaded software simulations.

In this work we present detailed hardware implementation of the HTM. The hardware model significantly speeds up training and classification processes over current software implementations by exploiting the inherently parallel and modular nature of HTM architecture which is useful in many real world problems. Our results also demonstrate effectiveness of a HTM even though it is still in its inception and compares it favorably to other well-known approaches.

Investigate and Report on ASIC Options for Implementing Hierarchical Temporal Memory

by
Pratyush Karli

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2014

APPROVED BY:

_____	_____
Dr. Paul Franzon Chair of Advisory Committee	Dr. Eric Rotenberg
_____	
Dr. James Tuck	

DEDICATION

To my parents, family and Dr. Paul Franzon for their support.

BIOGRAPHY

Pratyush Karli was born in Hyderabad, India. He received a Bachelor of Science in Electronics and Communications Engineering from Jawaharlal Nehru Technological University in 2011. In 2011, he joined the graduate program at North Carolina State University with a major in Computer Engineering. During his graduate studies he worked as an intern at Cymer Inc., San Diego from May, 2012 to August, 2012. He has been working on his Master's thesis with Dr. Paul Franzon since Spring 2013.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Paul Franzon for his guidance and support throughout my thesis. I want to thank him for being patient with me, for all the time he took out to discuss my work and for his valuable feedback. I would like to thank my committee members Dr. Eric Rotenberg and Dr. James Tuck for supporting my research and approving to be my advisory committee. I would also like to thank mummy, nanna, chayapinni, prasadmama, uttamama, bhavanipinni, monu and friends for being a constant support throughout.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

This chapter will discuss the motivation behind HTM, its architectural suitability to hardware, its difference from other learning algorithms, aim of this work and an overview of techniques used to implement the design. The later part of the chapter presents a brief outline of the thesis structure.

1.1 Motivation

Pattern learning and recognition, especially in domains such as speech and vision, is a problem that has been researched by the Artificial Intelligence community for decades now. Despite years of efforts and tremendous computing power available today we are still struggling to solve this basic problem. For example, a major issue in visual pattern recognition is that minor variations to an image pattern can result in a substantially different spatial representation considering in terms of pixel intensities. The idea was to develop variation tolerant metrics like tangent distance [23] or invariant feature extraction techniques like SIFT [17], but these techniques have been successful only for specific problems. Considering that human beings performs this task almost effortlessly, neuroscientists in association with computer scientists have conducted extensive studies to understand the human neocortex's operation.

The major reason, the neocortex is able to solve this problem is that it forms invariant representations of objects. In the presence of noise, variations etc., it is able to use these

representations to get inferences about objects. It forms invariant representations by using its hierarchical structure. It takes inputs (nerve impulses) from sensory organs (eyes) which are primary representations. These representations pertain to simple details such as lines, angles etc. At every stage of its hierarchy it forms representations combining lower level representations using time as a yardstick. This process is called as subsampling. These higher level representations pertain to more complex details such as figures, objects etc. At higher stages these representations become invariant to changes in simpler details such position, angle of perception and noise. This makes pattern recognition accurate and robust.

This led to conception of many biologically inspired computational models, one of them being the Hierarchical Temporal Memory.

Hierarchical Temporal Memory is a biologically inspired computational framework proposed by George [4] and Hawkins [2]. It is devised based on the theory called Memory Prediction Framework [1] which states that the human neocortex is constructed by replicating a single basic building block called the Canonical Cortical Circuit [1] connected in a hierarchical structure. All these building blocks use fundamentally the same algorithm to learn patterns of different modalities. Hierarchical Temporal Memory architecture consists of layers called as Levels that are arranged in a hierarchical fashion. Each Level has a number of basic building blocks called Nodes or Columns. Each Node performs the same algorithm in all layers similar to the neocortex as mentioned above. Each Level has decreasing number of Nodes as we ascend the hierarchy giving it an inverted tree shaped structure.

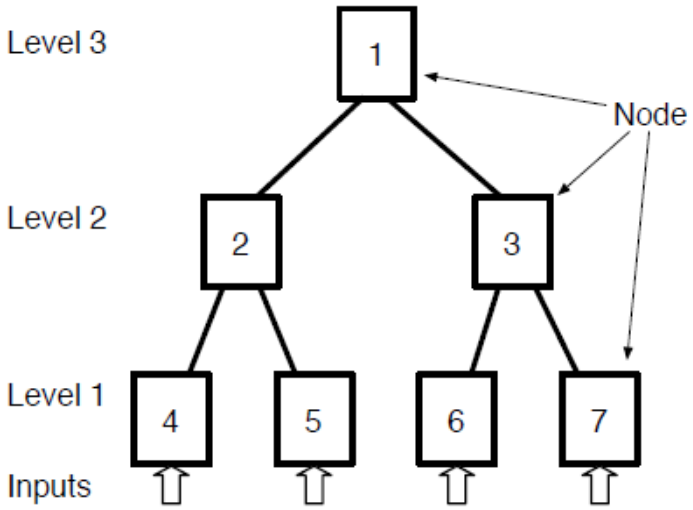

Figure 1.1: A simple HTM network that has 7 nodes arranged in a 3 level hierarchy. Original image presented by Dileep George [4].

Nodes in a Level are only connected to other Nodes in the next higher Level. No connections are present between Nodes of the same Level. These two qualities make the system extensively parallel and pipelined making them suitable for hardware implementation.

As opposed to traditional computers which are primarily sequential in operation biological systems are inherently parallel which explains their rapid functioning. Lack of extensive parallelism in existing Software implementations of the HTM leads to underutilization of the speed up offered by the parallel nature of the HTM architecture. In the path of building truly intelligent systems, along with accuracy, fast feature detection is a primary concern. This is a major bottle neck in current single-core software models. There is presence of multi-core, multi-threaded processors, dedicated processing units and various forms of parallelization techniques in software that can be used to improve HTM performance.

This work helps in characterizing the parallelism offered by HTM architecture which could be used in future hardware and software implementations of the HTM.

HTM substantially different from traditional neural networks where decisions are made by propagation of information based only on neuron's activation levels without any internal subsampling, eliminating the scope for neuron variable correlations. Therefore, invariance in pattern representations is highly dynamic in neural networks compared to HTM. HTM uses time as an implicit supervisor for unsupervised learning and forming correlations among spatial patterns called temporal sequences. This property makes it necessary for these neural networks to be trained on a large and diverse input database to make pattern detection more accurate. These features make neural networks effective only in specific problem domains whereas HTMs are more general in nature. A well-known implementation of a neural network is a Multi-layer Perceptron.

HTMs are closer to Multi-stage Hubel-Wiesel Architectures (MHWA) which is a subfamily of Deep Architectures. A MHWA consists of alternating layers of feature detectors which in turn locally subsamples features introducing some degree of invariance. Similar to a HTM, MHWAs also have a top layer trained in supervised mode that performs classification. MHWA use simple spatial operators like maximum or average value to subsample features instead of temporal pooling which reduces the invariance quality of their representations. Some examples of MHWA are HMAX [19] and Convolution Networks [18].

HTMs use temporal feature pooling and Bayesian belief propagation to propagate representations which make them similar to a layered version of Hidden Markov Model (HMM) [20]. While HMM models the intrinsic temporal structure of input patterns such as

audio, HTM exploits temporal continuity of inputs during training for unsupervised formation of invariant representations by temporal correlation of patterns. Even with static inputs such as images, HTM correlates patterns based on their appearance as time progresses i.e. two spatially dissimilar images are considered to originate from the same object if they frequently appear close in time. The reason for this property of HTM is the simple observation that the neocortex, in real world, scans an object thoroughly to perceive it before moving on.

1.2 Objective

This section highlights some of objectives and design techniques used. The main goal of this thesis is to build an RTL model of the HTM with training and inference operations. Focus is on building a single Node, instantiating it multiple times in a hierarchical organization to build the entire system.

Every node has a Spatial Pooler and Temporal pooler blocks. The Spatial Pooler's primary purpose is to quantize the input patterns. Vector Quantization algorithm with Euclidean distance is used to achieve this. Temporal information is stored in a Time Adjacency Matrix (TAM) during training process which is later partitioned into small temporal groups using Greedy Graphical Grouping algorithm. Probabilistic decisions are better than binary choices when dealing with uncertainty of novel input patterns. Therefore, probability information of the elements in a temporal group are stored which are passed to higher Levels. A Fixed point precession unit is designed to calculate probability values. Training of HTM is done on a Level by Level basis which increases training time. A data-buffer is implemented to speed-up the training stage.

Although HTM can be used for a variety of domains, in this thesis we focus only on image classification domain i.e. inputs are 2D images. Experimentation on two handwritten digit datasets is performed. Various metrics are compared and contrasted to find optimum performance configuration. Comparison of hardware implementation is done with software simulations of HTM running on a single-core single-threaded general purpose processor. Comparisons with other learning algorithms is conducted to determine whether a hardware implementation meets our expectations in performance and speed up.

1.3 Thesis Structure

Chapter 2 presents an extensive description of HTM architecture and learning algorithms. Chapter 3 describes the results of simulations. Chapter 4 covers the conclusions and scope for future work.

CHAPTER 2

HTM Architecture and Learning Algorithms

2.1 HTM Architecture

An HTM is a tree-like architecture composed of nLevels (≥ 2) Levels L(i) numbered from 0 to nLevels -1. L(0) is the input Level and L(nLevels) - 1 is the output Level. Reset are called intermediate Levels. If nLevels is equal to two then the network is said to have no intermediate Levels. Each Level is composed of Nodes N(Level)(Node). As we focus on designing a generic Node it is denoted as N or N(Level) to include information about its Level. HTM used for image pattern recognition, the input Nodes are in mapped directly with image pixels. This input Level could also be considered as the preprocessor. Nodes in each Level are arranged in visually a similar way to the image i.e. in rectangular grid. The Top most Node has only one output which is working as a pattern classifier.

Levels act as pipeline stages where they are sequentially interconnected through Node connections. Here, only connections between Nodes in consecutive Levels are allowed. Each intermediate or output Node is connected to a set of spatially close child Nodes in the lower Level called as the field of view. Given a Node N, we denote with child(N) the set of its child Nodes, with nchilds(N) the number of its child Nodes, and with child(N)(K) its Kth child Node. Regions are rectangular shaped and the number of Nodes along each of the two dimensions in a region is defined in such a way that allows an even partition of L(i-1) Nodes to L(i) Nodes. For example, in the Figure 2.1, L(0) has 256 Nodes arranged in a 16 X 16 grid whereas L(1)

has 16 Nodes arranged in a 4 X 4 grid; each intermediate Node has 256/16=16 child Nodes arranged in a (16/4) X (16/4) region.

Each input or intermediate Node is connected to a single parent Node in the next higher Level. The parent Node of N is denoted by parent(N). The field of view of a Node can be conceived as the portion of input image that the Node can see in terms of image pixels. For input Nodes, the receptive field is just one pixel. Higher in the hierarchy, a Node's receptive field becomes the union of its child receptive fields. As we move up in the hierarchy the receptive field also grows eventually spanning the entire image.

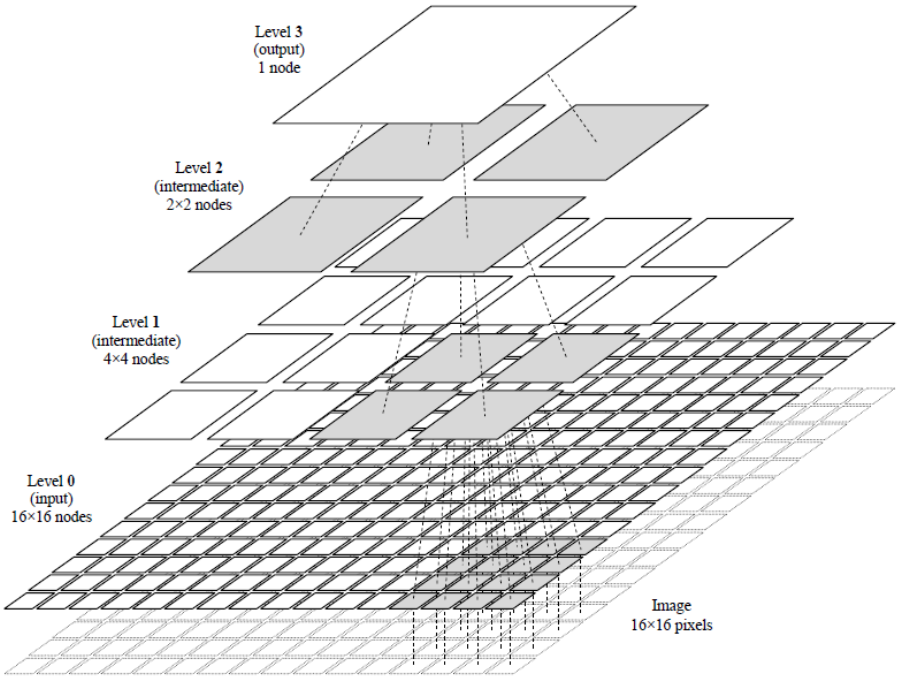

Figure 2.1: A 4 Level HTM network used to process 16 X 16 pixel images. Original image presented by Davide Maltoni [5].

2.2 Belief Propagation in HTM

Information flow in HTM here is considered unidirectional. Messages travelling bottom-up known as feed-forward inputs are denoted with λ using the notation introduced by Pearl for Belief Propagation [16] and adjusted to HTM [4]. An input from below, denoted with e, is called the evidence. According to Bayesian theory if e is a pattern probability p(e) is the pattern density. A previous input state of input is denoted with e+, is called contextual information. In Bayesian terms, if e+ is a pattern then p(e+) corresponds to the pattern prior. According to Bayes theorem, by fusing density with prior we obtain the best probabilistic explanation of unknown patterns [16]. In the feed-forward flow each input or intermediate node takes in input a message from each of its child nodes.

$$\lambda = p(e \mid child(N)) \tag{2.1}$$

The above equation means that λ corresponds to the conditional density of the evidence given the status of the child(N). After internal processing of this information, the node produces an output for its parent node. The output message is a vector whose elements denote the probability that the input pattern belongs to any of the problem classes. When messages from all the child Nodes is combined will either concretize or alter the belief of the parent Node regarding the class of the input. Feed-forward propagation of messages is performed level by level, starting from level 0. All nodes must process their input and produce their output, before higher level nodes can start their computation. This is illustrated by the Figure 2.2

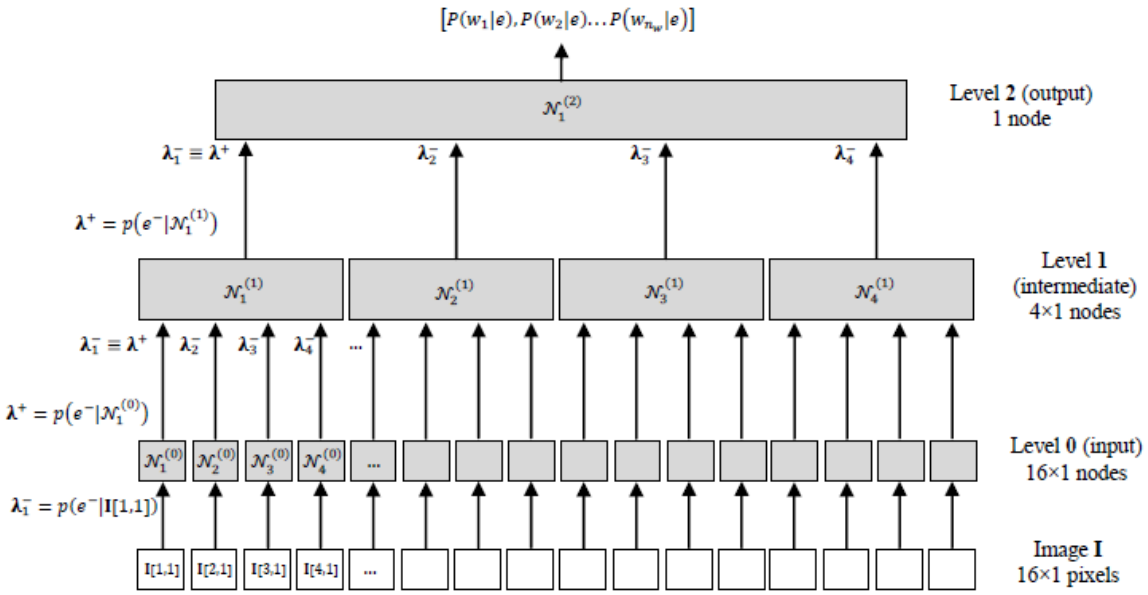

Figure 2.2: A 4 Level HTM network showing propagation of beliefs. Original Image presented by Davide Maltoni [5].

2.3 Node and Inference

Here we describe the internal structure of input, intermediate and output nodes and explain how nodes process information while performing inference. Inference is the phase where new patterns are presented to the HTM for classification. We are assuming that the network nodes are already trained and therefore all the node internal data have been already initialized. The training operation will be described in the later sections.

2.3.1 Input Nodes

Input nodes only receive patterns from below i.e. the input image, where I[x, y] denotes the image pixel at x, y position. The input pattern is an amalgamation of all the pixel intensities in the field of view of the node here only one pixel.

$$\lambda = \{ I[x1, y1], I[x2, y2], I[x3, y3]. . \} \quad (2.2)$$

Thus emulating the early processing performed by simple cells in the visual cortex [29]. Input nodes do not perform any internal processing, they simply propagate their input to the output and can be considered as a preprocessing step.

2.3.2 Intermediate Nodes

The internal structure of an intermediate node is shown in Figure 2.3. The node contains a set of columns a set of temporal groups and a PCG matrix.

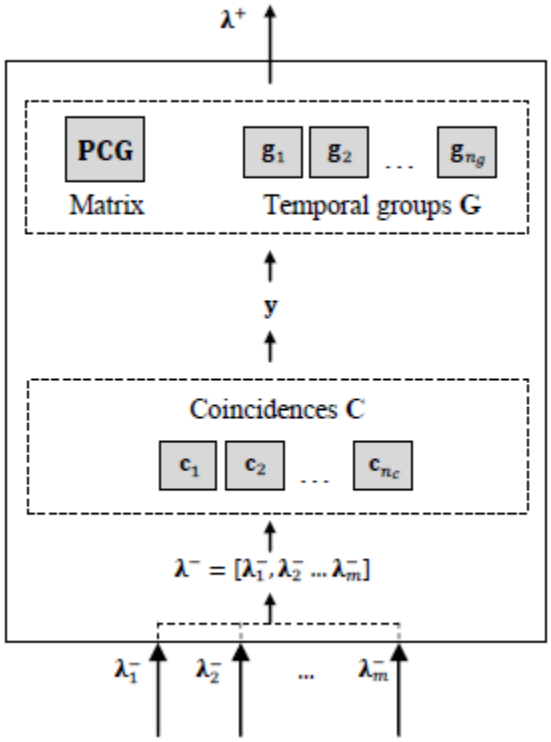

Figure 2.3: A node working in Inference Mode. Original Image presented by Davide Maltoni [5].

Each coincidence is a sort of quantization center and also can be considered as a prototype pattern that spans a portion of the image corresponding to the node field of view. Coincidences are used to perform a spatial analysis of input patterns and to find out spatial similarities. However, the coincidence meaning depends on the node level, if is an intermediate node at level 1 a coincidence corresponds to a small image patch. An example of coincidence graphical representation is shown in Figure 2.4. Note that the coincidence dimensionality is the same as the input message i.e. the sum of the dimensions of all the input messages coming from child nodes. If is an intermediate node at level 2 a coincidence can be conceived as a feature selector. Each element is the collection of indices of a single temporal group from each child node. An example is as follows. If a node in Level 2 has 4 child nodes whose output is the index of their winning temporal group, group 8 from child node 1, group 12 from child node 2, group 40 from child node 3, group 19 from child node 4 and group 17 from child node 5 then the column of the parent node stores [8, 12, 40, 19, 17].

Figure 2.4: Examples of Columns in Lower Level nodes.

A serious drawback of spatial similarity only pattern recognition is that slight variations of the input pattern can produce relevant changes in the feature representation. For example, let us consider the pixel level representation of the line drawings in Figure 2.5, small shift of just one pixel is enough to mismatch the spatial similarity with the original pattern.

Figure 2.5: Examples of Columns in Lower Level nodes and their variation effect on pattern.

A temporal group is a subset of coincidences, that could be spatially quite different each from the other, but that are likely to be originated from simple variations of the same pattern. Patterns originating from the same object generally are presented to the network very close in time. An example of temporal grouping is shown in Figure 2.6.

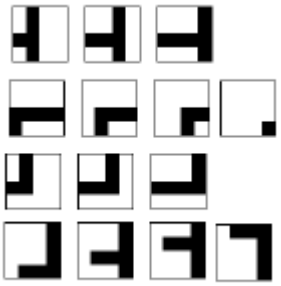

Figure 2.6: Example of temporal groups formed as subsets of columns.

Inference in an intermediate node executes the following algorithm. A parent node obtains a single input message by amalgamation of the input messages from the child nodes. The vector length depends on the number of temporal groups formed in child nodes and their maximum lengths put together. It is denoted as λparent = {λchild1, λchild2, λchild3, λchild4 . . .}. An input vector contains the temporal group index which means it represents the conditional densities of the evidence given the coincidences. Intuitively each can be conceived as the activation level of coincidence when the node input is λ. Activation computation is considered as a Gaussian distance.

$$y[i] = e^{-|\text{column}_{\text{vector}} - \lambda|^2/\sigma^2} \tag{2.4}$$

Here, y(i) is the Gaussian distance to find the conditional density and its normalization would yield the conditional probability. σ is a parameter controlling how quickly the activation level decays when λ deviates from column value. Figure 5 shows an example of coincidence activations. At every stage of calculating probability values normalization is done to keep probability values from dropping below. Normalization does not affect the HTM behavior.

The conditional density over a temporal group can be obtained by probability marginalization over the group coincidences as shown in following equation.

$$p(e|\text{temporal group}) = \sum_{i=1}^{\text{number of columns}} \sum_{j=1}^{\text{number of temporal groups}} y[i].\,PCG[i,j] \tag{2.5}$$

The output message from a intermediate node is the set of combined conditional probabilities of all temporal groups.

$$\lambda\ \text{output} = \{p(e|\text{temporal group1}), p(e|\text{temporal group2}),\ \ldots\} \tag{2.6}$$

A graphical example of column activation and group activation for an input λ is shown in Figure 2.6

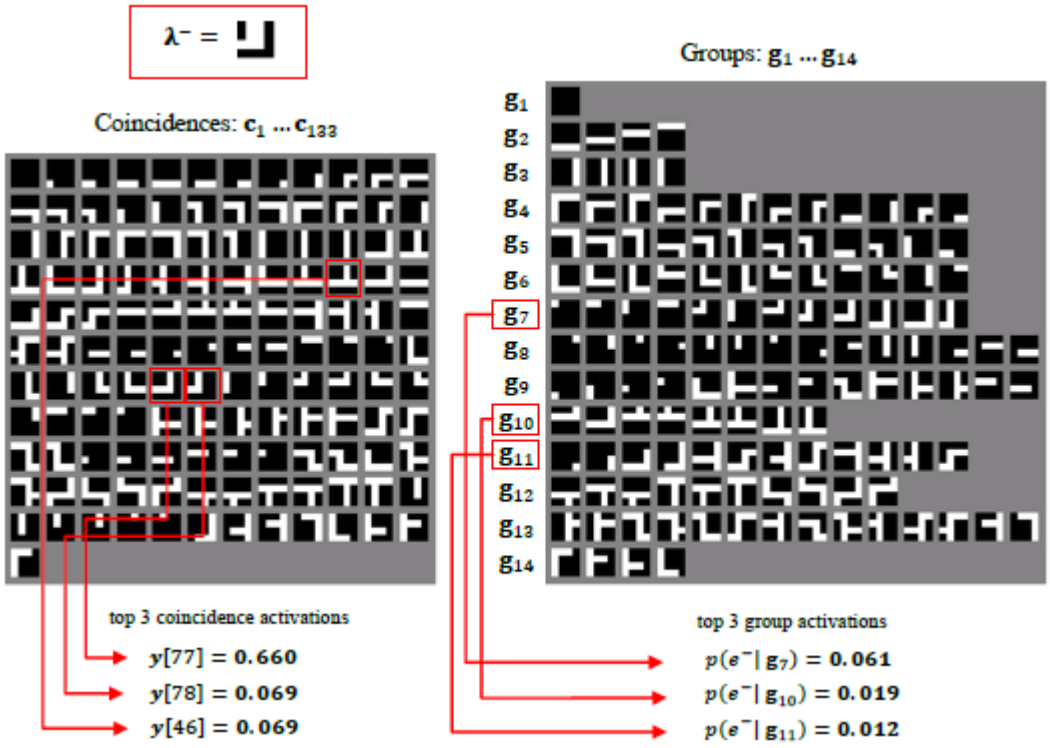

Figure 2.7: A graphical example of column activation and group activation for an input λ. Original Image presented by Davide Maltoni [5].

2.3.3 Top level node

The top level node or the output node works as a pattern classifier. Its internal structure is shown in Figure 2.7. The input part of the node is identical to an intermediate node, whereas in the temporal grouping section is replaced by class data aiding in supervised learning. The node maintains a set of coincidences. A PCW matrix and a prior probability vector to calculate probabilities of appearing columns from the PCW matrix.

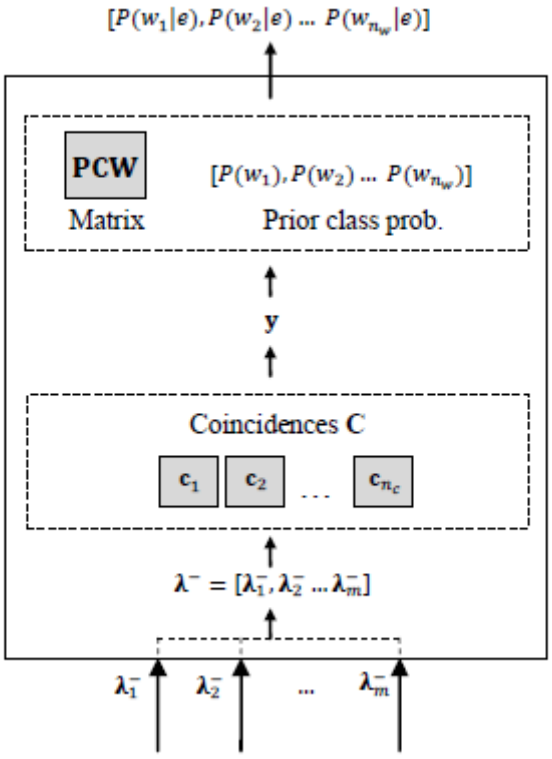

Figure 2.8: Internal structure of a Top Level node. Original Image presented by Davide Maltoni [5].

Output node coincidences are identical to intermediate node ones. However, except for degenerate cases where the network has no intermediate levels, the level of output node is greater than or equal to 2 and therefore coincidences at this level work as feature selectors. In all pattern classification problems, the knowledge of class prior probabilities allows to improve classification accuracy according to Bayes theory. In HTM prior class probabilities are computed at training time. PCW is a, number of columns X number of number of classes, matrix element and stores the conditional probability of coincidence given the class.

Inference in the output node can be decomposed in the following steps. Composition of input message and computation of densities over coincidences is identical to intermediate nodes. Computation of densities over classes can be obtained by probability marginalization over the class coincidences.

$$p(e|class) = \sum_{i=1}^{\text{number of columns}} \sum_{j=1}^{\text{number of classes}} y[i].\,PCG[i,j] \qquad (2.7)$$

Where the assumption holds that probabilities are independent implying unsupervised learning because the knowledge of class is irrelevant for the estimation of density in the context of columns. Composition of output message whose dimensionality is simply composed by the class total probabilities.

$$\lambda output = \{P(class1|e), P(class2|e), P(class3|e), P(class4|e) \ldots\} \qquad (2.8)$$

2.4 Training Mode of HTM

During network training the nodes aim at computing Columns C, groups TG and matrix PCG for all intermediate nodes. Coincidences C, priors P(classes) and matrix PCW for the output node. Once training is finalized all network nodes are switched in inference mode and the network can start classifying unknown patterns.

HTM training requires a training data set, where Strain is a pattern here a binary image vector and the corresponding class. Strain = {binary vector, image class}. Intermediate levels are trained in unsupervised mode where pattern class information is not shared, whereas the output node is trained in supervised mode. HTM training is performed level by level, from L(1) to L(3). L(0) is considered as a preprocessor section as it does not do any function. When training nodes at level L(i), all the network nodes at previous levels, whose training was already finalized, work in inference mode.

Training an intermediate level requires exposure to a sequence of patterns. Such a sequence can be obtained by smoothly moving each training pattern across the network visual field. Since consecutive patterns in the sequence are close in time we can expect they are characterized by minor changes in terms of geometric introducing translation, rotation, scale change etc. Although different strategies can be designed to extract a sequence of temporally close patterns from a training set, a baseline implementation is to perform, for each pattern, two scans. One in a horizontal zig-zag flow followed by a vertical zig-zag by moving the foreground object contained in the field of view. Figure 2.8 shows an example of sequence for a single training pattern. Performing a double scan is important to learn pattern temporal similarities in fact, if we consider a pattern containing a horizontal bar, a horizontal scan alone

would not allow to temporally group variants of the same pattern where the bar occurs at slightly different vertical positions. A full training sequence can be obtained by concatenating sequences generated by single training patterns. In general, in a training sequence we can have discontinuities, denoted as temporal gap. Temporal gaps occur when we abruptly move a pattern to a distant position to start a new scan or when the training pattern changes. This induces temporal group change.

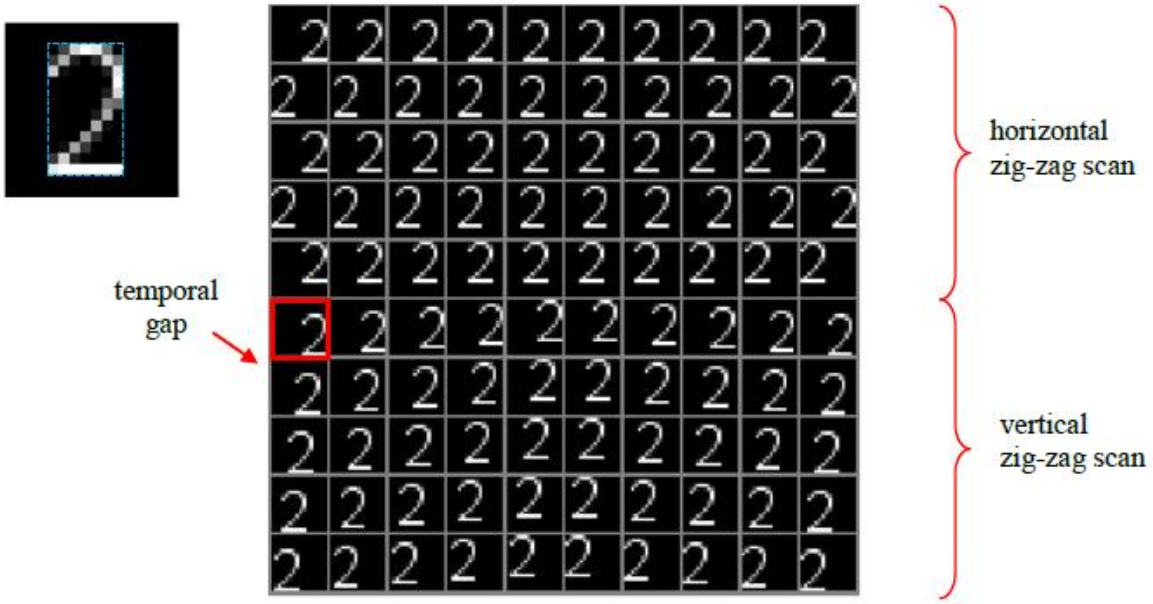

Figure 2.9: Scanning of images to produce temporal grouping. Original Image presented by Dileep George [4].

Finally, to train the output node it is sufficient to expose the node to single training patterns with associated class labels as no temporal information is processed by the output node. However, if the network is required to recognize patterns independently of their position,

each training pattern must be presented at different positions. In practice, we can use training sequences like the ones reported in Figure 2.8, but unlike for intermediate nodes, here only a single scan is necessary.

The procedure Train Intermediate Node in Level L(i) assumes that input pattern has been presented to the network at L(0) and inference has been already performed until level L(i-1). Hence output patterns {λ1, λ2, λ3 . . .} from low Level child nodes are available.

The active coincidence is the spatially closest coincidence to the node input calculated using Euclidean distance with a threshold value of 18. This threshold value is calculated using software simulations to keep the number of columns to an optimum level. If all existing coincidences are too dissimilar from the input because of threshold, then a new coincidence is created and selected as active coincidence. If the threshold value is high then many totally disconnected columns are put into a quantization center and when these coincidences are selected it induces error into the HTM. Too high threshold value will not group similar coincidences and will also result in large memory resources being utilized to store patterns of same object. Selecting and bringing forward only one coincidence, implements a winner take all criterion that is in contrast with the continuous criterion used during inference, when the activation of all coincidences are taken into account for the computation of group activations. It is worth noting that such an asymmetrical approach is not atypical, and proved to be quite effective in training other deep architectures [15] [30].

Y is a vector of indices of child winning temporal groups. Index Y[i] is the temporal group index, within the temporal groups of child nodes, which obtained maximum activation.

$$Y[i] = \{\max(\lambda[\text{child node1}]), \max(\lambda[\text{child node2}]), \max(\lambda[\text{child node3}]) \ldots\} \quad (2.9)$$

It means only 1 temporal group from 1 child node is allowed to move forward. Again, this is in contrast with the continuous criterion used during inference where the activations of all child groups are used to compute coincidence activations.

Ydistance(column value, Y) counts the number of differences between the column value present in the parent node and the value stored in Y at corresponding positions after equation 2.9 is executed on the input vector. This operation is illustrated with an example. Assume Y = {18, 9, 7, 4} and column in parent node C = {8, 9, 7, 7}, then the Ydistance will be 2.

A record keeper called seen is maintained for each coincidence to count the number of times it was active during the node training. When finalizing node training these values will be used to quantify coincidence relevance.

TAM is a Temporal Activation Matrix. It is number of columns X number of columns in size and is used to keep track of coincidences that have been activated in succession, and thus are good candidates to form a temporal group. When finalizing node training will be used to compute temporal groups and PCG. The TAM is update either only by looking only one step back in time or several steps to make stronger temporal groups. For better performance a look back buffer is used to see 3 steps back in time which gives optimum results.

T(previously appeared column, current column) =

T(previously appeared column, current column) + (1 + location in look back buffer -1) (2.10)

We linearly decrease weights to update the TAM matrix. Once all patterns in the data set are finished training can be finalized. Forgetting rare coincidences can be useful to reduce the number of coincidences. In fact, deletion of rarely activated coincidences usually has a minor impact on the network classification accuracy. To make TAM symmetric especially in image domain is very useful to give stronger temporal groups. Here the upper diagonal part is summed to the lower diagonal part for each pair. Making symmetric allows coincidences that occurred close in time to be grouped independently of the activation order. Therefore a pattern moving left-to-right across a node receptive field yields to the same groups as the same pattern moving right-to-left.

$$TAM(i, j) = TAM(j, i) = TAM(i, j) + TAM(j, i) \tag{2.13}$$

For i traversing through each column in TAM and j traversing through each row. Coincidence priors can be simply obtained by normalizing the number of times coincidences have been activated during training.

$$P(Cj) = seen(j) / \sum_{k=1 \text{ to number of columns}} seen(k)\, A = \pi r^2 \tag{2.12}$$

TAM values are proportional to the probability of co-occurrence of coincidences. A simple normalization by rows makes TAM values true conditional probabilities.

$$TAM(i,j) = TAM(i,j) / \sum_{k=1 \text{ to number of columns}} TAM(i,k) \quad (2.13)$$

For i traversing through each column in TAM and j traversing through each row. After the above normalization TAM(i, j) = P(Cj | Ci). Note that after normalization TAM is no longer symmetric.

2.4.1 Temporal Group Partitioning

A temporal group is a set of coincidences that are likely to occur close in time. Partitioning coincidences into a set of disjoint groups, can be formulated as a clustering problem aimed at maximizing the functional temporal activity. All the temporal groups are disjoint sets.

$$TGi \cap TGj = \emptyset \text{ for each } i, j, i \neq j \quad (2.14)$$

Group Maximum size is set to limit the number of columns assigned to a group. Group maximum size has to be carefully set not to group all the columns in a single group or to have single columns in subsequent groups. This value can be from anywhere from 10 to 15. Maximization of 12 columns to maximize the average group temporal connection, that is the within group temporal connections among coincidences.

Clustering is one of the most studied problem in pattern recognition and machine learning [31] and hundreds of algorithms have been proposed in the literature. The clustering problem at hand has some peculiarities. We can easily compute similarity between any pair of coincidences, but there is not an efficient way to compute the centroid of a set of coincidences.

The number of coincidences and groups can be quite large in practical applications, so we need computationally efficient approaches. We do not care too much about the optimality of the solution since HTM is robust enough with respect to suboptimal grouping. For this reasons an ad-hoc computationally efficient greedy grouping algorithm was introduced in [32].

In this algorithm most connected coincidence is chosen which is already not taken and initialized as the primary coincidence. Next the set of coincidences with highest temporal connection with the primary coincidence, excluding coincidences already assigned are selected and these coincidences are added to a temporal group, by inserting them at the end of the list and by excluding coincidences already present in the list. The same operation is repeated by making these highest temporally connected sequences and making them primary sequences. The operation is repeated until the temporal group reaches its group maximum size or all columns are exhausted. A typical value for highest temporal connected columns is 3.

The greedy grouping algorithm runs by creating one group at each time. The group seed is a single highly connected coincidence, to which its coincidence are associated. Making group growing recursive adding new columns in a tree like fashion will cause its coincidences to be associated as well.

2.4.2 PCG Computation

PCG denotes the conditional probability of coincidence given the in a group. The computation of this matrix is performed in two simple steps. PCG(temporal group, column) retains its value if the column belongs to the temporal group or else is set to zero. Later PCG is normalized for individual temporal groups.

$$\mathrm{PCG}(i,j) = \frac{\mathrm{PCG}(i,j)}{\sum_{k=1 \text{ to number of columns}} \mathrm{PCG}(k,j)} \tag{2.15}$$

The former step sets the conditional probability as the coincidence prior in case the coincidence belongs to the group. The latter is a within group normalization aimed at guaranteeing, for each group that

$$\sum_{i=1 \text{ to number of columns}} P(Ci \mid TGi) = 1 \tag{2.16}$$

2.4.3 Training Top Node

The coincidence formation and calculation of Y are similar to intermediate nodes. Instead of Temporal grouping there is PCW matrix. PCW stores how many times a coincidence has been active when a certain class was encountered. By normalizing the PCW matrix prior probabilities are found. These prior probabilities in association with Y during inference stages produce the final output.

$$\mathrm{PCW}(i,j) = \frac{\mathrm{PCW}(i,j)}{\sum_{k=1 \text{ to number of columns}} \mathrm{PCW}(k,j)} \tag{2.17}$$

For i is 1 to number of columns and j is 1 to number of classes.

2.5 Implemented Architecture

The architecture implemented in this thesis is a two level structure. Here each node makes use of a temporal pooler unit and a spatial pooler unit as shown in Figure 2.10. Each of these units make use of large internal memory modules for computation of beliefs.

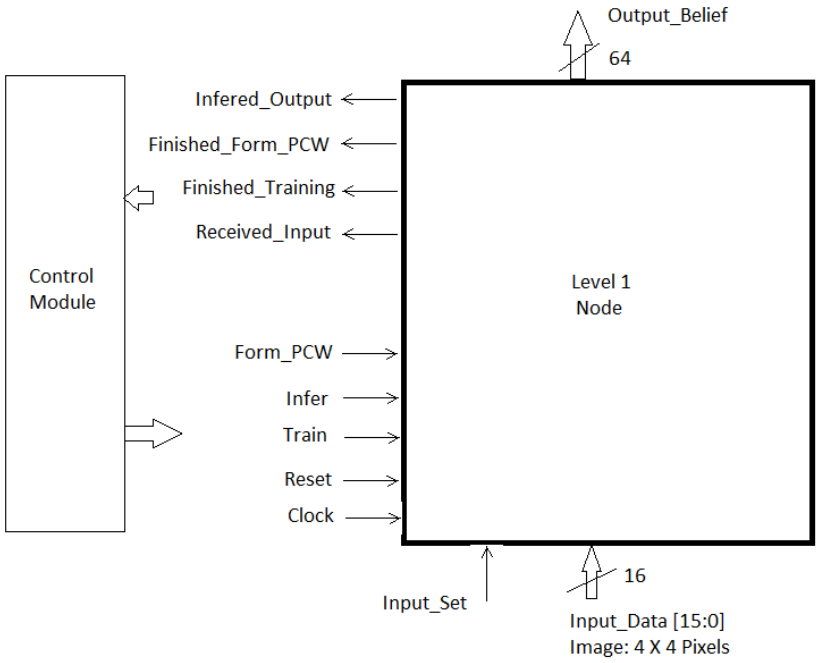

Figure 2.10: Block Diagram of a Level 1 Node and Interface with Control Module.

The Control Module performs feeding of image data using the control signals. The image data is used in training and inference modes. The internal structure of the Node is given in Figure 2.11.

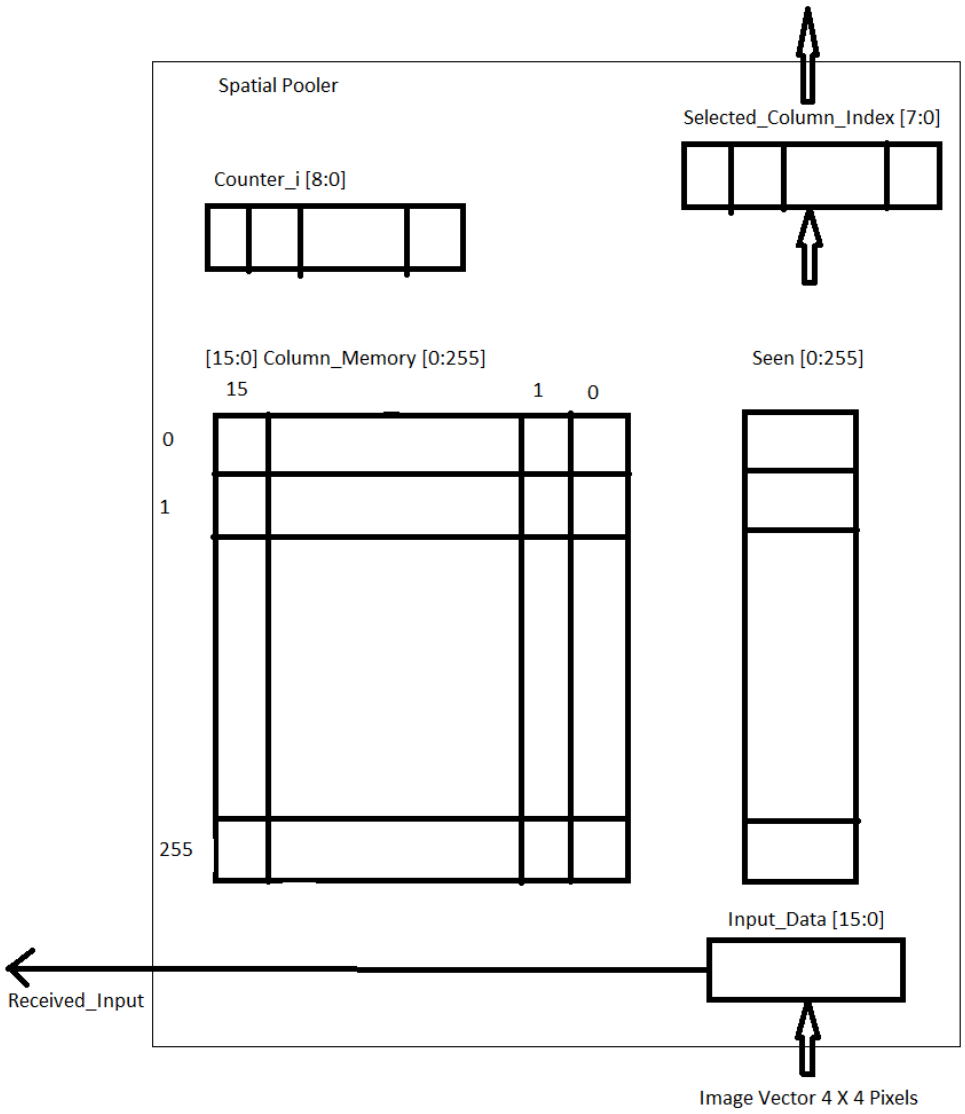

Figure 2.11: Internal Structure of Spatial Pooler of Level 1 and Level 2 nodes.

The Column memory is used to store quantization values. Seen memory is to identify all the quantization centers that are valid. Finally a column is selected from the column list and is propagated to the temporal pooler section. It has to be noted that the Level 1 and Level 2

spatial pooling operation is the same hence similar structure is used with only variation in input data size.

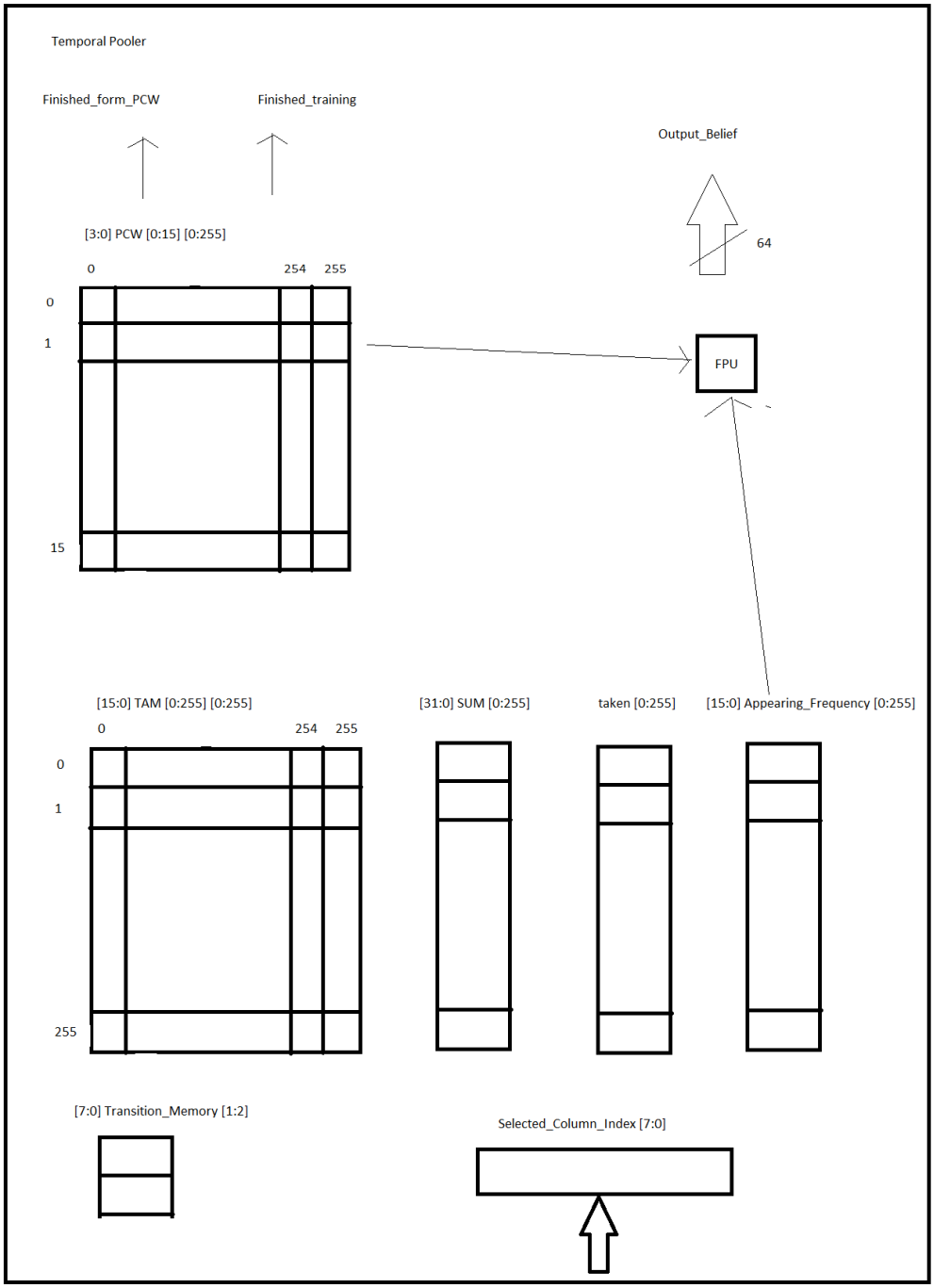

Figure 2.12: Internal Structure of Temporal Pooler of Level 1 nodes.

Here, the Time Adjacency Matrix (TAM) holds the transition relationship between the columns. Thus building a Markov graph. The Sum Memory is used to partition the graph into smaller sub graphs to form temporal groups. A PCW Matrix is formed by associating the columns with the respective temporal groups they are most like to be a part of. The appearing frequency unit stores the number of times a column appeared, giving a good estimate of probability of its re-occurrence.

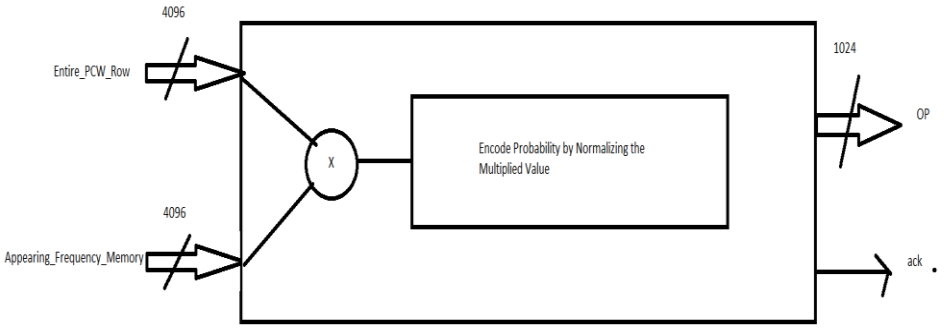

Figure 2.13: Operation of a Fixed Point Probability Normalizer.

A Fixed point unit is used to normalize the probability distribution over the columns and also reduce the amount of data propagating to next level. Generally, in situation of uncertainty about the category of image a probabilistic approach is more useful. Finally an Output_Belief is propagated to the next level. There are 16 nodes in level 1 and only 1 node in level 2. All the beliefs of level 1 nodes are combined to form a belief vector which is fed to the level 2 node.

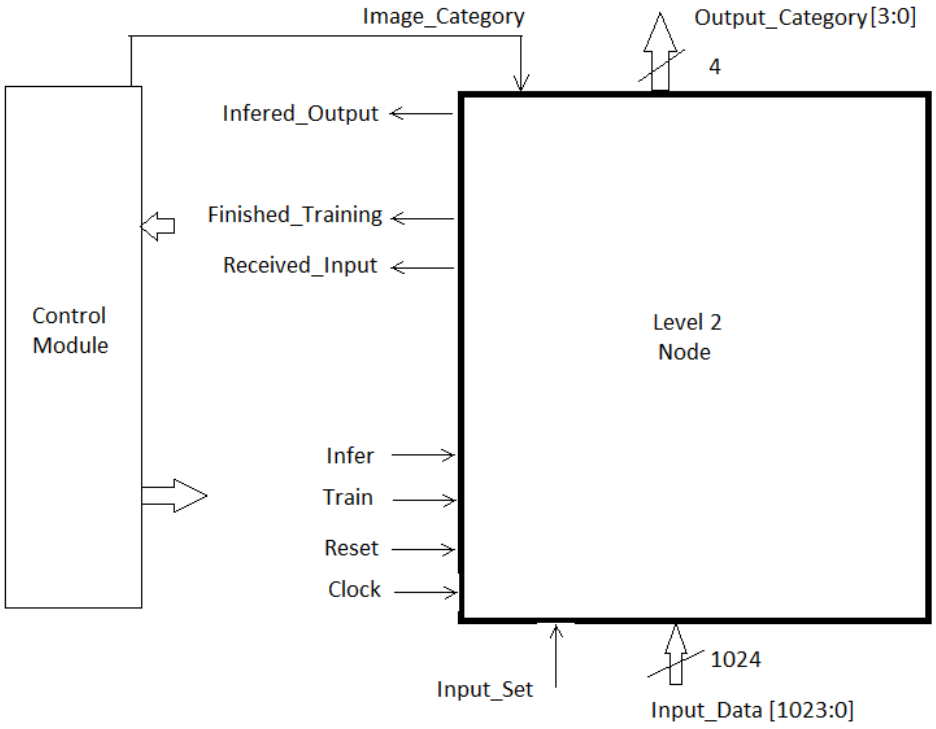

Figure 2.14: Block Diagram of a Level 2 node with supervised learning.

In Level 2 node the control module during training period provides the category of input image so that it can learn inputs by associating them to the correct category. In inference case the output is a belief that the input belongs to one of the categories that appeared during the learning phase. As previously suggested, the input is a concatenation of lower level inputs (64 X 16 = 1024). As there are only 10 categories in digit recognition only 4 bits are considered for output.

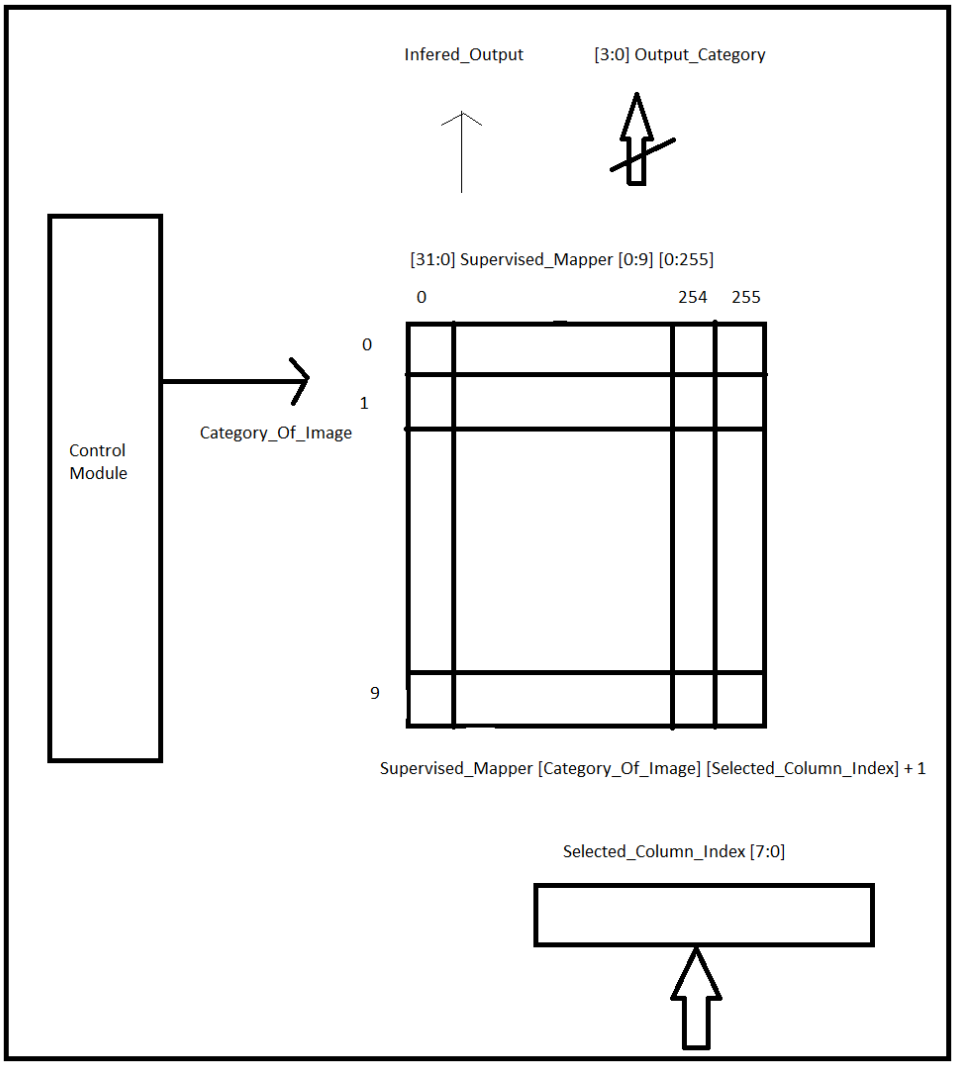

Figure 2.15: Internal Structure of a supervised mapper unit for Top Level Nodes.

The Top level node does not have a temporal pooler but instead a supervised mapper which learns using supervision in training phase and infers unsupervised during inference phase. The selected column from spatial pooler is then checked for strongest association with all the groups and whichever category exhibits this quality, the input is considered to belong to that category.

The Memory unit for lookup used in Spatial pooler of Level 1 and Level 2 of the HTM is Column_Memory which has 256 locations and is 16 bits wide. The Temporal pooler of Level 1 uses TAM memory storing a matrix which is 256 rows, 256 columns and each location holding 16 bits of data. It also has a PCW memory unit which is also a matrix memory with 256 columns 16 rows and 4 bits of each location. Finally, the Layer 2 has only a supervised mapper unit and no temporal unit which is a memory matrix of 256 columns 10rows with 32 bits each location. Therefore a Level 1 node combining spatial and temporal pooler has a total of 130.5KB (256 x 16 + 256 x 256 x 16 + 256 x 16 x 4 Bits). Total Level 1 has 16 nodes giving 2088KB (16 x 130.5KB) of memory. Level 2 has spatial pooler and Supervised mapper giving 10.5KB (16 x 256 + 256 x 10 x 32 Bits). Complete HTM with 2 Levels of 16 nodes in Level 1 and 1 node in Level 2 has 2098KB (10.5KB + 2088KB) or 2.049MB of memory for lookups.

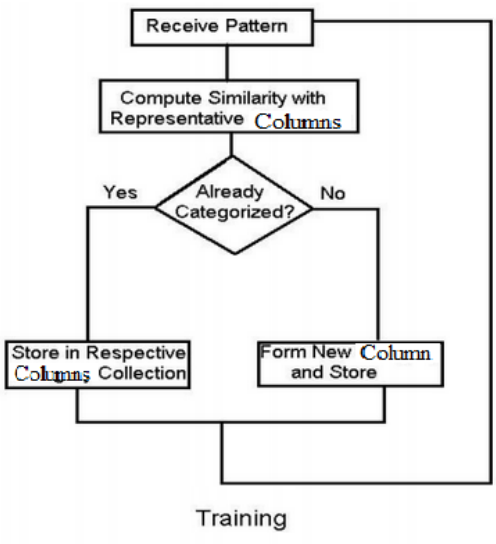

Figure 2.16: Flow chart of operation of a spatial pooler during training.

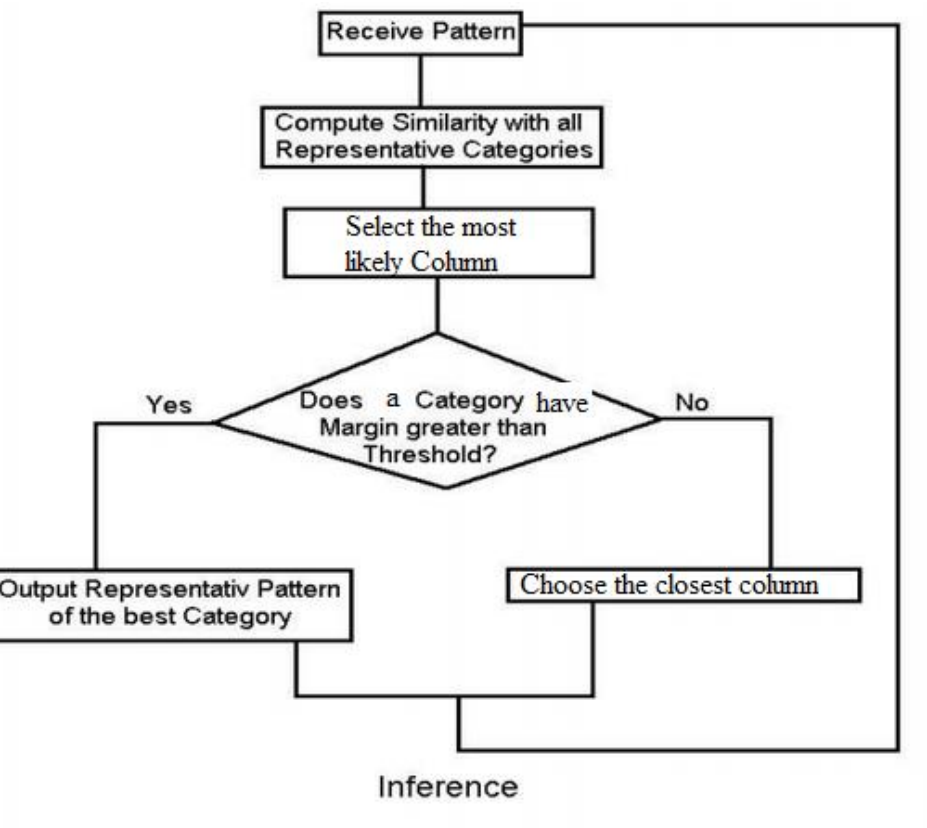

Figure 2.17: Flow chart of operation of a spatial pooler during inference.

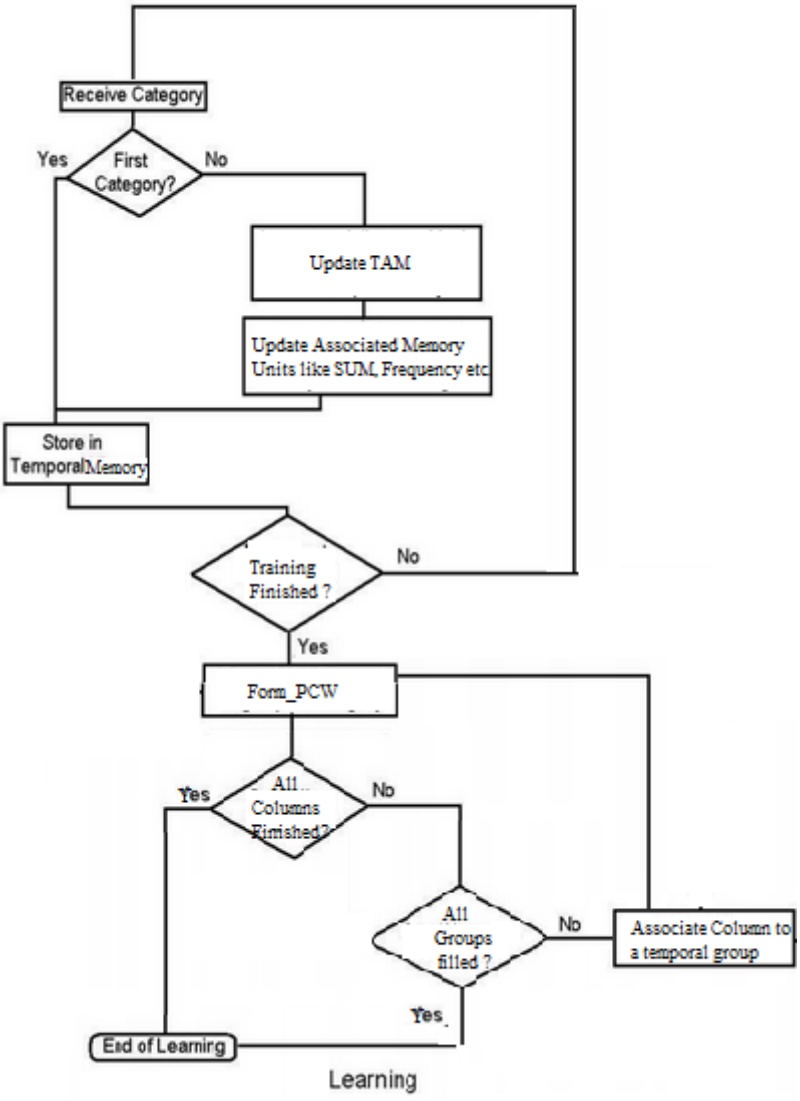

Figure 2.18: Flow chart of operation of a temporal pooler during training.

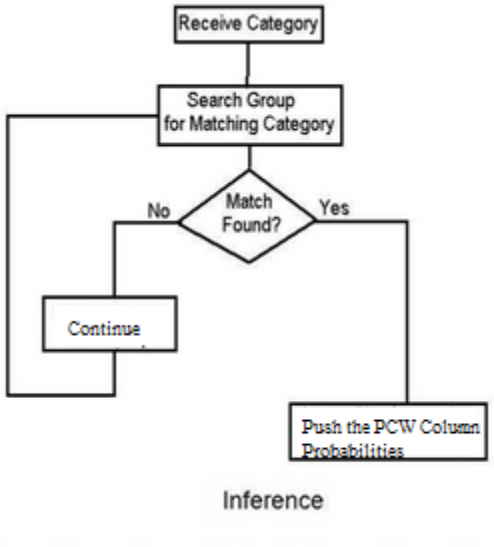

Figure 2.19: Flow chart of operation of a temporal pooler during inference.

In Comparison with HTM (hardware) a general purpose CPU's memory hierarchy is flexible. In HTM (hardware) the memory is hardwired. The size of a memory unit is fixed and depends on the type of dataset (i.e. 16 X 16 Pixel image), size of the dataset i.e. number of categories and number of training images in each category, parameters like quantization distance in grouping of similar images etc. The drawback of this custom-fit memory system is that it is not adaptable if any of these parameters are changed and the memories become small for it. In contrast, the dynamic and hierarchical memory configuration of a general purpose CPU is more adaptable to such changes.

CHAPTER 3

Experiments and Results

This chapter covers the results of pattern classification problem. Section 3.1 discusses the datasets and their features used in experiments. Section 3.2 analyzes the HTM training and parameterization for different datasets. Section 3.3 describes the setup and environment used to obtain the results. Section 3.4 compares the performance of HTM implemented in hardware against software simulations of HTM and other learning algorithms.

3.1 Datasets

For this research we have used two different image pattern classification problems, SDIGIT and USPS. In our opinion, these two datasets constitute a good benchmark to study 0invariance and robustness of a pattern classifier. However, in both the datasets the image patterns are small black-and-white i.e. binary images of 16 X 16 pixel.

3.1.1 SDIGIT

SDIGIT is a machine printed digit classification problem where just a single 16×16 pixels binary version image, called *primary* pattern, is provided for each of the 10 digit classes, and a number of variants are generated by geometric transformations of the primary patterns. By explicitly controlling the size and the amount of variation in both the training and the test set we can study specific characteristics of HTM related to training, invariance and robustness. We denote a set of patterns, including primary patterns of the 10 digits. The variation patterns generated by scaling and rotation of the primary pattern randomly. Primarily SDIGIT has 1000 Test patterns and 593 Training patterns. The creation of a test set starts by translating each of

the 10 primary pattern at all positions and further generating patterns from these subsets by repeated translations. Example of the SDIGIT test set from MATLAB simulation is shown below.

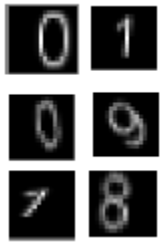

Figure 3.1: MATLAB simulation of images for translations of SDIGIT Data Set.

3.1.2 USPS

USPS is a widely known handwritten digit classification problem [36]. It is a major benchmark for most machine learning and pattern recognition approaches. USPS patterns are 16×16 pixels, 8-bit grayscale images which are converted to binary versions. It comes with two set of images, the training set and the test set. The training set contains 7291 training images and the test set contains 2007 novel test images. The advantage with USPS is that it has quite a large number of patterns for training. Similar to the translations in the SDIGIT we try to build subsets of these patterns to further enhance the idea of invariance by covering all kinds of input possibilities. Though the dataset is quiet extensive, the digits are centered in the 16 X 16 window and the test set variations are quite well covered by the large training set. Therefore, even a simple approach such as the Nearest-Neighbor classifier achieves good

classification results. Reporting HTM accuracy for software implementations is a requirement so considering this well-known benchmark becomes important. Some MATLAB simulations from training and test datasets of USPS are shown below.

Figure 3.2: MATLAB simulation of images from train and test datasets of USPS.

3.2 HTM Analysis

In a HTM implementation there are several subsystems where parameter setting is necessary. For example inside spatial pooler of a node in the Quantization algorithm, the distance between the columns and the input vector has to be set. Finding the optimal value for such parameters to generate the best performance needs lots of experimentation. There are numerous such parameters to be experimented with. Similar to many other pattern recognition approaches, the optimal architecture and parameter values are problem dependent and a proper parameter tuning can lead to a relevant performance improvement. Fortunately, HTM is quite robust with its parameterization.

In this experimentation, many software simulations were performed to come up with good parameter values. The parameter values are independently tested for specific areas to

give right performance. There may be other combinations of these parameters that gives even better performance.

A list of all the optimum parameter values are provided in Table 3.1 and Table 3.2. These parameter values are found after tweaking with software simulations and analyzing their localized gain. HTM implemented here is a four-level network. The L(0) is a preprocessor and only parses the image to the L(1) of the HTM. The next two Levels L(1) and L(2) perform both spatial and temporal analysis operations. The L(3) top level performs supervised learning and classifies the patterns during inference. Further adding levels makes the HTM rigid and increases the invariance across classes. A four-level architecture demonstrated to be an optimal choice for input patterns of size of 16×16 pixels. As reported in Table 3.1 and Table 3.2 node arrangements across the four levels is $16\times16 \rightarrow 4\times4 \rightarrow 2\times2 \rightarrow 1$.

Table 3.1: Optimal Parameter Values for HTM for a SDIGIT Dataset

Parameters	Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
Node Arrangement	16 X 16	4 X 4	2 X 2	1 X 1
Transition Memory	N/A	2	5	N/A
Group Minimum Size	N/A	4	2	N/A
Group Maximum Size	N/A	10	12	N/A
Top Neighbors	N/A	3	2	N/A
Threshold Distance	N/A	18	0	0
Decay (σ)	N/A	15	N/A	N/A

Table 3.2: Optimal Parameter Values for HTM for a USPS Dataset

Parameters	Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
Node Arrangement	16 X 16	4 X 4	2 X 2	1 X 1
Transition Memory	N/A	2	5	N/A
Group Minimum Size	N/A	4	2	N/A
Group Maximum Size	N/A	10	12	N/A
Top Neighbors	N/A	3	2	N/A
Threshold Distance	N/A	55	0	0
Decay (σ)	N/A	25	N/A	N/A

The threshold value is very important to performance. In Level 2 and 3 have threshold = 0, this means that all the sub-patterns encountered during learning are stored as new coincidences if they are not identical to already seen coincidences. The meaning of threshold for Level 1 is different with respect to higher levels as the coincidence distances are computed as Gaussian distances in Level 1 but for higher levels they are computed in terms of winning

indices differences. In Level 1, the value of threshold directly controls the number of image pattern coincidences created and heavily influence the whole network complexity in terms of number of coincidences and groups at all levels. Setting threshold = 0 would result in a huge number of nearly identical coincidences at level 1 which is also an over kill on memory. HTM invariance is affected for the worse if too many nearly identical coincidences are retained at level 1. So it is adjusted by tweaking to a level where reasonable number of columns are formed.

The σ value is mainly used in the inference stage controlling the coincidence activation decay when the current sub-pattern deviates from the stored coincidences i.e. the probabilistic limitation that a column is meant to be active. A too high σ value states the activation of a large number coincidences thus leading to reduction in number of varied representations. Whereas a low σ value states the activation of just one or two coincidences resulting in direct mapping effect where a column starts representing a class and any deviation will result in erroneous results. To keep HTM adhered to the rules of neuroscience only 2% of column activation is encouraged. It means that only 2% of all columns should contribute to activation in Y and the rest should be at least possible probabilities.

The size of transition memory is increased as we ascend the hierarchy as more prior information is required to form more invariant representations and higher levels are expected to be more invariant compared to lower levels. Therefore the temporal pooling must be extended to longer priors.

3.3 Setup and Environment for measurements

The HTM is compared against different modifications of Data sets to analyze training intricacies for a variety of inputs. The above mentioned dataset simulations are done using synthesizable Verilog implementation of the HTM. Later ASIC HTM is compared against software versions of HTM and various other pattern recognition algorithms. All the software simulation results the ASIC HTM is compared against are considered from [5]. The environment used in [5] is as follows.

Algorithms are coded in C# (.net) running in Windows 7 on a Xeon CPU at 3.07 GHz. There are no multi-core CPU architectures used. The floating point encoding used is a Double precision i.e. 8 Bytes.

In the ASIC based implementation we use Verilog HDL to code all the algorithms. Simulations and functional verifications are done using Mentor Graphics ModelSim in a Linux machine running on an i7 CPU at 3.2GHz. The RTL is synthesized and Static Timing Analysis is performed using Synopsys Design Compiler to meet timing requirements. MATLAB is used in performing initial tweaking with binary image files during understanding the HTM. A 16 Bits (2 Byte) precision Fixed point unit is used for all floating point calculations.

3.4 Performance Comparison

3.4.1 Efficiency in use of Time as a supervisor

Initially, experiments are performed with increasing sizes of training data sets to see the effect of training on accuracy of HTM. This is interpreted as relationship between performance and complexity.

Here, 50 images from the SDIGIT dataset are taken which span through all the 10 categories. Now, each image is taken and modified 6 to 7 times. First by changing position, translating it up and down, left and right from its basic position. Later by changing its angle, rotating it by 45 degrees clockwise and anti-clockwise. This operation produces 6 X 50 = 300 + 10 extra patterns where some form 7 translations giving 310 training patterns. Similarly, 1000 images are taken and modified to form 1000 X 6 = 6000 + some 200 extra patterns giving 6200 patterns.

In the same way as described above the training data set is increased to 100 patterns and along with translations gives us 620 patterns and finally to 250 patterns giving us 1550 patterns. The same test data is used for the 3 variations. The results are shown in Table 3.3.

Table 3.3: Performance versus complexity in terms of size of train dataset for SDIGIT

Training Set Size	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Average number Columns in Levels per node
50 primary patterns translated to 310 derived patterns	99.9	68.83	L 1: 153 L2: 210 L3: 1121
100 primary patterns translated to 620 derived patterns	99.93	81.7	L 1: 294 L2: 412 L3: 2797
250 primary patterns translated to 1550 derived patterns	100	89.39	L 1: 459 L2: 622 L3: 6421

As the USPS Dataset contains many transformations in its primary patterns itself no translations are done to these patterns. Increasing number of patterns as 100, 1000, 7291 are applied to infer on 2007 test patterns. On applying these direct patterns we get the following results shown in Table 3.4.

Table 3.4: Performance versus complexity in terms of size of train dataset for USPS

Training Set Size	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Average number Columns in Levels per node
100 primary patterns	100	82.41	L 1: 63 L2: 81 L3: 357
1000 primary patterns	100	91.22	L 1: 152 L2: 537 L3: 4212
7291 primary patterns	100	92.87	L 1: 279 L2: 3112 L3: 31007

Performance versus complexity shows that the larger the training set, the higher is the accuracy on the test set. On the other hand, intensive training leads to the creation of a larger number of coincidences and groups and then to a more complex and larger network whose efficiency and performance can degrade. As usage of such huge memories to gain such small performance improvements may not be feasible.

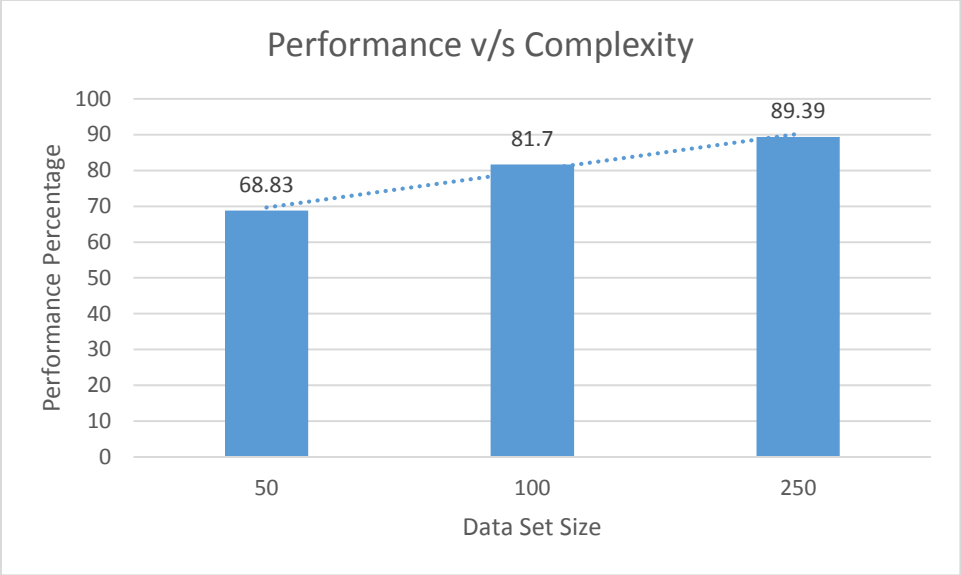

Figure 3.3: Comparison between performance and complexity for SDGIT where more variations were introduced in input patterns.

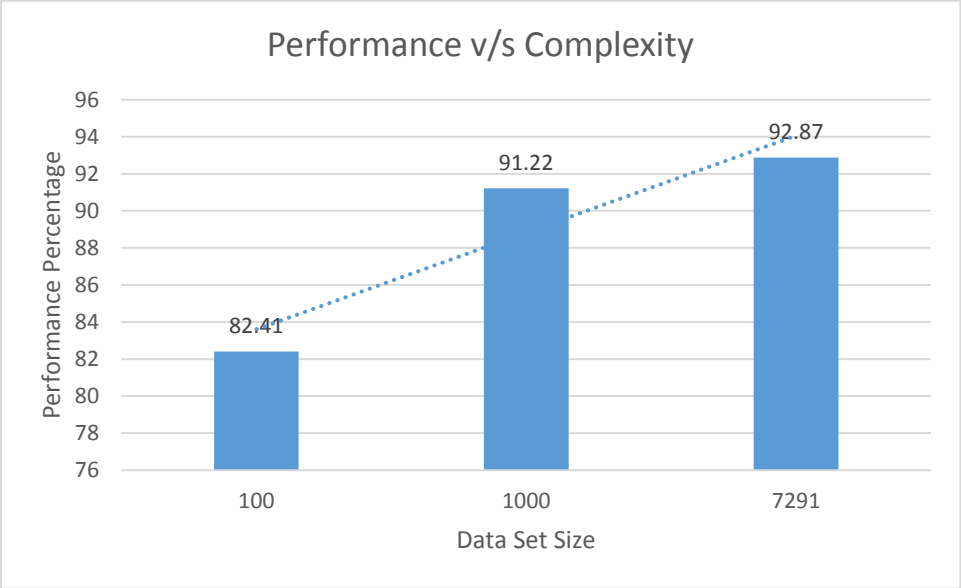

Figure 3.4: Comparison between performance and complexity for USPS more data patterns with less variations are input patterns.

Note that the geometric variations in the training sets are slightly smaller than corresponding test set variations in USPS compared to SDIGIT. This led to a minor performance improvements as we increase the datasets. The idea to be considered here is that it is not the size of the training data but the amount of variations included, yields performance. A HTM can train on low variation data and can form temporal sequences of them but when new variation appears it becomes clearly difficult for it to classify. So the more variations a HTM can incorporate in its temporal memory, more would the performance be.

This explains the steep curve in USPS dataset where with slight increase in size of data input new variations start to appear but when variations start to fade away even though data fed is growing enormously the performance grows slowly.

3.4.2 Comparison with Software HTM and other algorithms

Initially, experiments are performed with increasing sizes of training data sets to see the effect of training on accuracy. To compare HTM with other algorithms, the focus is on techniques that work at pixel level without any hardwired feature extraction. Comparison is done in terms of accuracy and efficiency. A simple Nearest Neighbor classifier is considered. This classifier works on direct pixel to pixel comparisons and gives a good baseline performance and is useful to estimate the problem difficulty. Another major class of algorithms are the artificial neural networks. MLP is a three layered input-hidden-output perceptron [11]. MLP is the best known neural network architecture and therefore it is interesting to understand how it performs in comparison with HTM. Finally, convolutional networks which are Multi stage Hubel Wiesel architectures of the Deep architectures family are considered. LeNet5 is a Convolutional Network (CN) designed to classify characters and small line-drawing [17]. CN

is one of the most interesting architectures for visual pattern recognition and therefore is a very good yard stick for HTM.

All these algorithms are implemented in [5] are in software using open source libraries provided by their developers for research purposes. Here, only time and accuracy are considered. The images included in increasing datasets of software may be differently chosen compared to hardware implementation. The comparison results are shown below.

3.4.2.1 Results of SDIGIT Dataset

Table 3.5: Comparison for SDIGIT 50 pattern translations

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
50 primary patterns translated to 1788 derived patterns	Nearest Neighbor	100	57.92	< 1sec	4 sec
	Artificial Neural Networks (MLP)	100	61.15	12 min 42 sec	3 sec
	Convolutional Networks (LeNet5)	100	67.28	7 min 13 sec	11 sec
	HTM (Software)	100	71.37	8 sec	13 sec
	HTM (Hardware)	100	72.32	0.04075 sec (or 40.75 us)	0.04316 sec (or 43.16 us)

Table 3.6 Comparison for SDIGIT 100 pattern translations

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
100 primary patterns translated to 3423 derived patterns	Nearest Neighbor	100	73.63	< 1sec	7 sec
	Artificial Neural Networks (MLP)	100	75.37	34 min 22 sec	3 sec
	Convolutional Networks (LeNet5)	100	79.31	10 min 5 sec	11 sec
	HTM (Software)	100	87.56	25 sec	23 sec
	HTM (Hardware)	100	86.29	0.1225 sec	0.11967 sec

Table 3.7: Comparison for SDIGIT 250 pattern translations

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
250 primary patterns translated to 8705 derived patterns	Nearest Neighbor	100	86.50	< 1sec	20 sec
	Artificial Neural Networks (MLP)	99.93	86.08	37 min 32 sec	3 sec
	Convolutional Networks (LeNet5)	100	89.17	14 min 37 sec	11 sec
	HTM (Software)	100	94.61	2 min 4 sec	55 sec
	HTM (Hardware)	100	93.48	0.1201 sec	0.0115 sec (or 1.5 us)

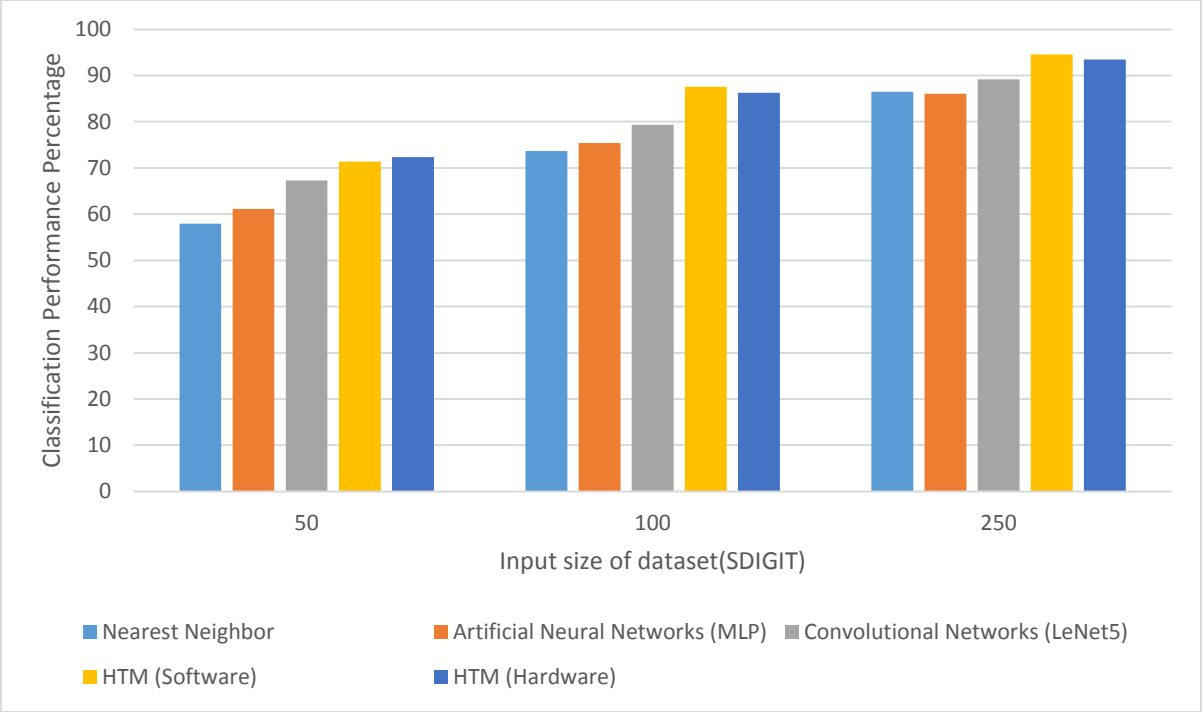

Figure 3.5: Comparison between performances of pattern classification algorithms for different input sizes of SDIGIT dataset. The four Software implementations are done by David Maltoni [5].

HTM consistently outperforms all the other techniques in pattern classification. The reason for slightly low performance of the ASIC HTM is that we are using a 23 bit floating point precision whereas the software is using 8 Byte precision. Considering the slight fall in performance over resource and power consumption is fairly good tradeoff.

Another reason why HTM is better is that it exploits its time parameter in learning as many variety of variations and grouping them together which is not possible in special systems as they depend on only the pattern to classify. This is why HTM exhibits more growth in performance even with low number of input vectors.

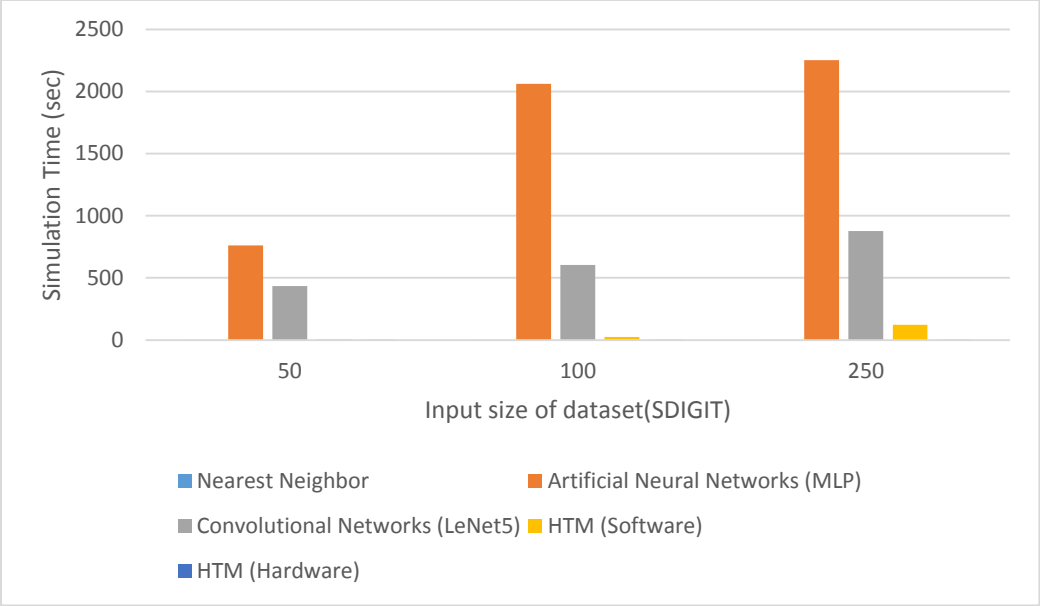

Figure 3.6: Comparison between Training time of pattern classification algorithms for different input sizes of SDIGIT dataset. The Software implementations are done by David Maltoni [5].

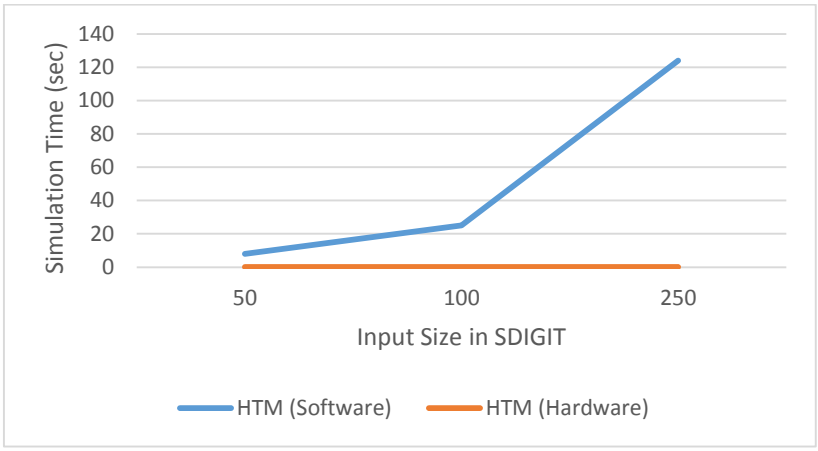

Figure 3.7: Comparison between Training time of HTM software and HTM hardware for different input sizes of SDIGIT dataset. The Software implementations are done by David Maltoni [5].

Here as we can see the training of systems with data takes a lot of time and as inputs grow the training time also increases even with the software implementation of HTM. It is noteworthy that ASIC HTM exploits parallelism offered by HTMs thereby keeping its training time close to zero. Using a three Level HTMs instead of four Levels may increase the on chip memory bandwidth but it further reduces the training times.

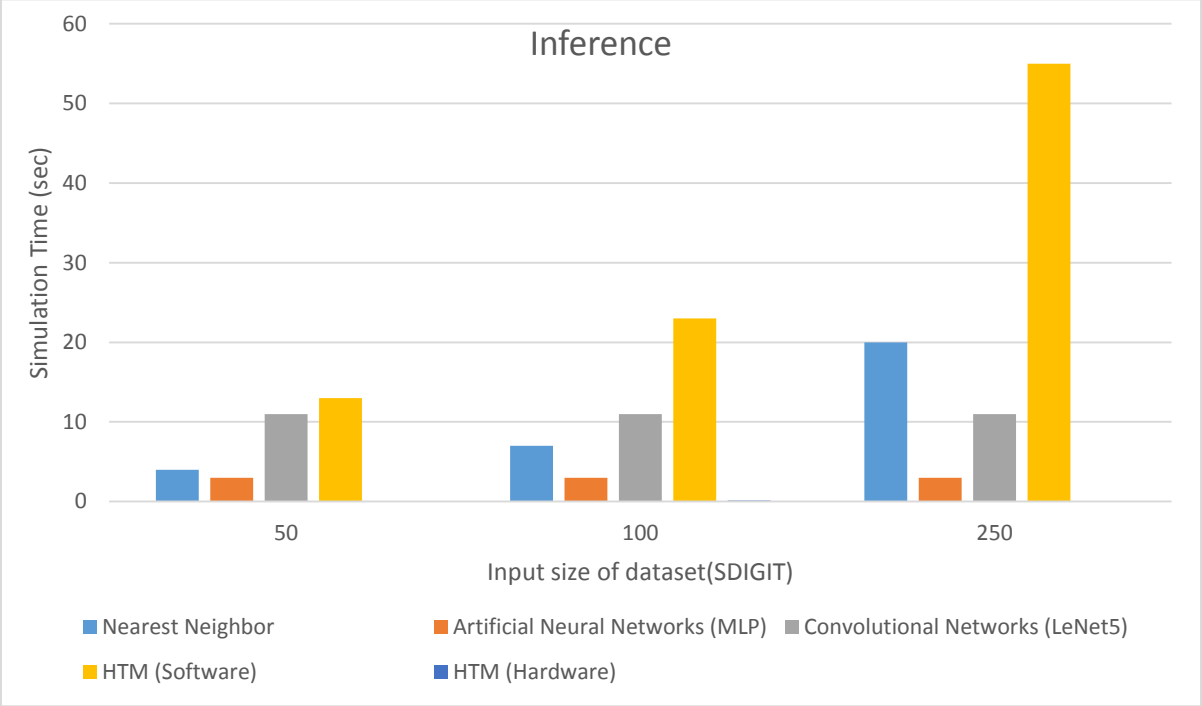

Figure 3.8: Comparison between Inference times of pattern classification algorithms for different input sizes of SDIGIT dataset. The Software implementations are done by David Maltoni [5].

Evident from above graph HTM hardware outperforms all the other algorithms including HTM software in infering a Dataset with comparably high performance in the

tremendously short time. HTM software uses maximum time among all the other algorithms. HTM has efficient trainng but such large simulation times is a major drawback.

These high performance results are because of considerable amount of variation introduced by translating extensively the basic patterns of SDIGIT. Now to strongly establish the role of variant patterns that are pattern wise disconnected, temporally connected. We see the USPS data set that has large number of patterns in its test and train sets. Most of the patterns are small variations but not as much as needed to distinguish HTMs performance against other algorithms. One of the important considerations are that all the software implementations may consider different image subsets so performance may vary.

3.4.2.2 Results of USPS Dataset

Table 3.8 Comparisons for USPS 100 patterns

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
100 primary patterns	Nearest Neighbor	100	76.53	< 1sec	< 1sec
	Artificial Neural Networks (MLP)	100	73.20	4 min 29 sec	< 1sec
	Convolutional Networks (LeNet5)	100	83.53	19 sec	4 sec
	HTM (Software)	100	84.11	3 sec	2 sec
	HTM (Hardware)	100	85.4	0.002279 sec (or 2.279 us)	0.000696 sec (or 0.696 us)

Table 3.9: Comparison for USPS 1000 patterns

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
1000 primary patterns	Nearest Neighbor	100	91.18	< 1sec	< 1sec
	Artificial Neural Networks (MLP)	100	89.54	3 min 53 sec	< 1sec
	Convolutional Networks (LeNet5)	100	94.42	6min 11 sec	4 sec
	HTM (Software)	100	93.42	1 min 39 sec	9 sec
	HTM (Hardware)	100	92.49	0.02579 sec (or 25.79 us)	0.00696 sec (or 6.96 us)

Table 3.10: Comparison for USPS 7291 patterns

Training Set Size	Pattern Recognition Algorithm	Accuracy (%) for Training patterns	Accuracy (%) for Test patterns	Training Time	Inference Time
7291 primary patterns	Nearest Neighbor	100	94.42	< 1sec	5 sec
	Artificial Neural Networks (MLP)	99.52	94.52	50 min 53 sec	< 1sec
	Convolutional Networks (LeNet5)	99.87	96.36	36min 59 sec	4 sec
	HTM (Software)	99.39	95.57	26 min 45 sec	30 sec
	HTM (Hardware)	99.22	93.12	0.166169 sec (or 166.2 us)	0.0139 sec (or 13.9 us)

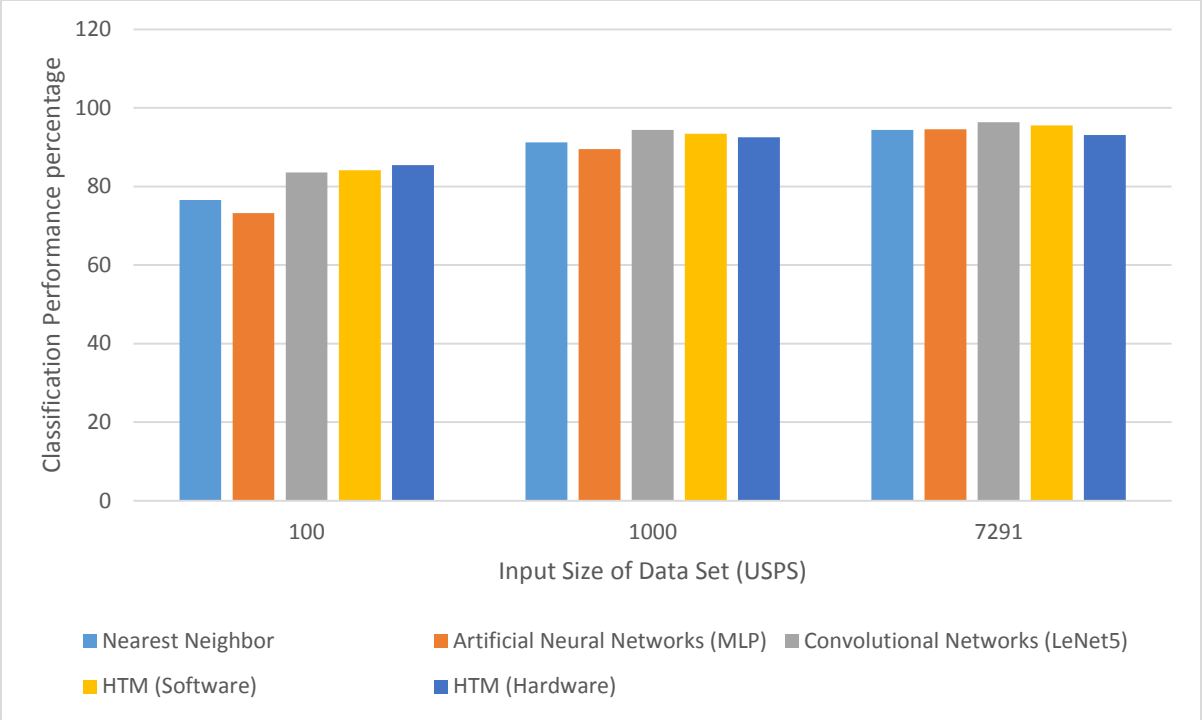

Figure 3.9: Comparison between performances of pattern classification algorithms for different input sizes of USPS dataset. The Software implementations are done by David Maltoni [5].

Here as we can clearly see that the USPS dataset is used directly to train and infer. As most patterns of the test set are covered in the train set it becomes easy for other algorithms to perform similar to HTM. If you take a look at the left of the graph HTM clearly moves ahead of other algorithms proving that spatial dissimilarity is compensated by temporal connectivity. As more similar patterns come in the temporal nature of HTM is drowned in too much of data and making it loose track of sequential information. Therefore it is suggested that though limited but highly variant patterns must be provided as inputs to understand times role in pattern classification.

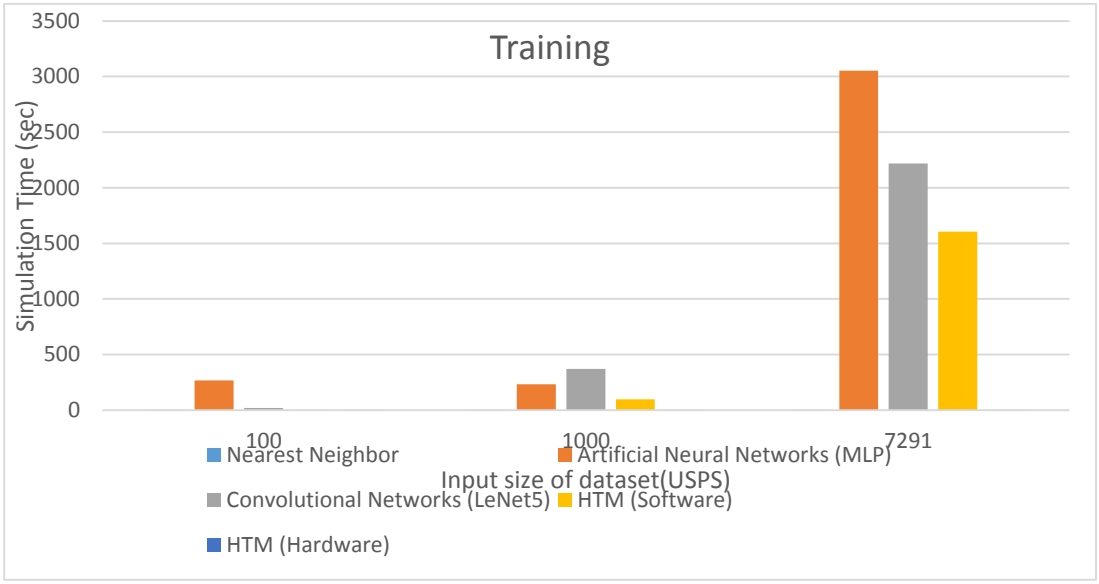

Figure 3.10: Comparison between Training time of pattern classification algorithms for different input sizes of USPS dataset. The Software implementations are done by David Maltoni [5].

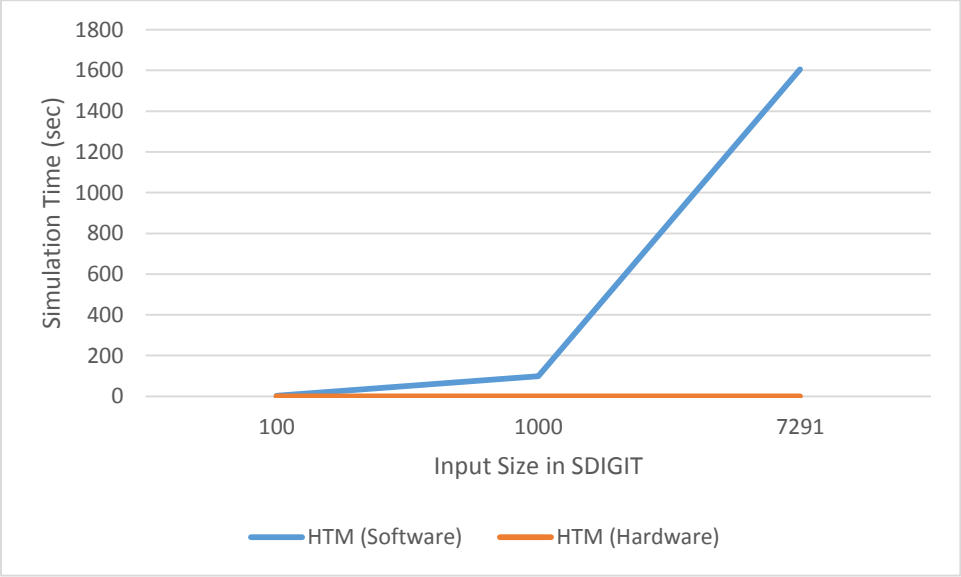

Figure 3.11: Comparison between Training time of HTM software and HTM hardware for different input sizes of USPS dataset. The Software implementations are done by David Maltoni [5].

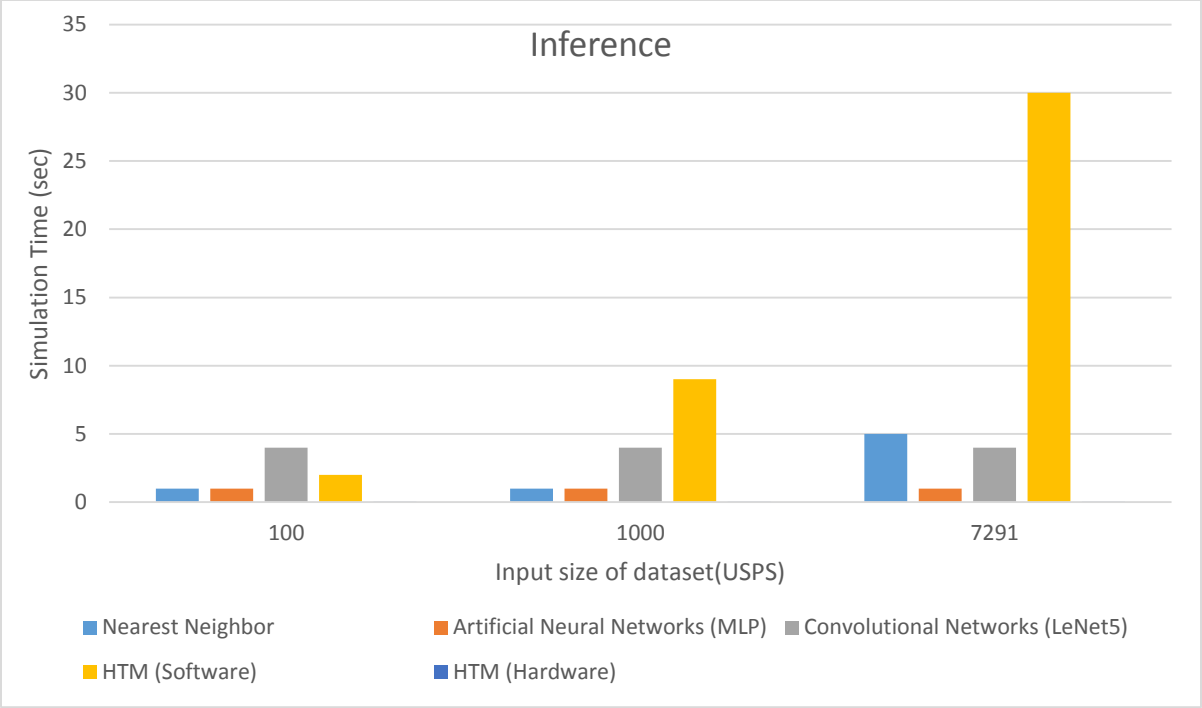

Figure 3.12: Comparison between Inference times of pattern classification algorithms for different input sizes of USPS dataset. The Software implementations are done by David Maltoni [5].

3.5 Power and Area Metrics

There are 19628658 cells in the design and has a total cell area of 23449682.57 um^2 (23.44mm^2). It has high switching activity consuming power of 851.49mW. The calculations are done by synthesizing the design and finding out the SAIF and SPEF files post-layout analysis. Synopsys Design Compiler has been used to perform synthesis, Encounter for layout extraction, QRC for paracitics extraction and modelsim for simulations.

A clock of period 20ns i.e. 50MHz (1/20ns) is used in synthesis with no timing violations reported. We arrived at 20ns by using a large clock initialy and progressively

reducing it to 20ns below which there are violations arising. The clock can be further reduced by more inter Node pipelining. The Intel Xenon CPU W3550 used in software simulations by Davide Maltoni [5] against which the HTM hardware is compared against runs at 3.07 GHz.

Here the chip area of 23.44 mm^2 is large for just doing this special function. The large area is because all the memories are synthesized as D Flip Flops and compiled memory macros may reduce the area and power of the chip.

CHAPTER 4

Conclusion and Future Work

This chapter will discuss the motivation behind HTM, its architectural suitability to hardware, its difference from other learning algorithms, aim of this work and an overview of techniques used to implement the design. The later part of the chapter presents a brief outline of the thesis structure.

4.1 Conclusion

In this paper we provided an in-depth analysis of Hierarchical Temporal Memory to pattern recognition. Hardware development of HTM is proposed and its pros and cons are discussed and demonstrated on two different datasets through a number of experiments. HTM performance was then systematically compared with other pattern classification systems including Linear Classification, Convolution Networks in the Multi-stage Huber-Wiesel Architectures domain and also traditional neural networks. In almost all our experiments HTM performance especially in Hardware was favorable than other systems tested and learning was also more efficient. On the other hand, inference time is often longer in HTM with respect to some of the other systems tested. To solve this problem ASIC is a good solution in exploiting HTMs parallel architecture. A limitation of this work is no comparisons are made to parallel software implementation. Finally, there are other algorithmic changes in HTMs with parallelization and optimization to induce faster performance in software which is left for future work.

Although results achieved so far are very interesting, we believe that Hierarchical Temporal Memory framework could be significantly improved in the future. The most evident weakness of current implementation is scalability. In fact the network complexity considerably increases with the number and dimensionality of training patterns. Also in hardware, inter Level communication uses high bandwidth routes which increases power dissipation. Number of coincidences and groups rapidly increases with the number of sub-patterns in the training sequences. Here frequent and high speed access to the memory is required which is addressed by on chip memories but memories can be expensive and therefore limited resource versus increasing column access has to be addressed. On the other hand, to deal with complex pattern recognition problems with large intra-class variance the presentation of a number of long training sequences that are very diverse, appears to be necessary for the formation of robust groups and to keep the column count low. The way the brain encode patterns is still debated by neuroscientists but a sparse distributed population encoding is one the most plausible hypotheses. This means that the simultaneous activation of a group of cells is responsible for the conscious perception of a stimulus.

4.2 Future Work

Future research efforts could be diverted towards developing novel sparse population coding mechanisms to improve HTM scalability. One possibility is removing the constraint that a coincidence of a node must be formed with contributions of all its nodes. A coincidence created from a *subset* of would cover only a portion of receptive field and probably would be more general and reusable to encode a larger number of patterns. Exploiting feedback

to develop new effective strategies tom make predictions. Feed back in these networks and intra Level communications are two areas that need to be extensively researched.

Applying HTM to difficult object recognition benchmarks such as CalTech and Pascal VOC and systematically compare HTM with state-of-the-art approaches. There will be unavoidable increase in the network complexity due to large color images which are not just difficult to preprocess but also need more on chip memory storage space. It will be necessary to embed low-level feature extraction at level 0 which would be another area to look into. To define different non exhaustive strategies to create training sequences based on random walks in huge pattern spaces. It would be interesting to see how new temporal sequencing strategies would affect the invariant representation formation.

Training HTM incrementally is a huge bottle neck in hardware implementation. A continuous form of training needs to be developed that facilitates smooth movement of patterns through HTM as in the case of inference. Online learning of patterns is one area that requires more study. With online learning and feedback it would be a significant step forward to create truly intelligent machines. Implementing and comparing a parallel software model on multi-threaded, multi-core processor would provide a better comparison for future work.

REFERENCES

[1] J. Hawkins and S. Blakeslee, On Intelligence, Times Books, Henry Holt and Company, New York, 2004.

[2] J. Hawkins, S. Ahmad and D. Dubinsky, "Hierarchical Temporal Memory including HTM Cortical Learning Algorithms", Numenta tech. report, December 10, 2010.

[3] D. George and J. Hawkins, "A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex", proc. International Joint Conference on Neural Networks (IJCNN), 2005.

[4] D. George, "How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition", Ph.D Thesis, Stanford University, June 2008.

[5] Davide Maltoni, "Pattern Recognition by Hierarchical Temporal Memory", University of Bologna, April 2011.

[6] D. George and J. Hawkins, "A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex", proc. International Joint Conference on Neural Networks (IJCNN), 2005.

[7] D. Chandra, "Speech Recognition Co-processor", Ph.D. Dissertation, North Carolina State University, 2007

[8] J. Thornton et al., "Robust Character Recognition using a Hierarchical Bayesian Network", proc. Australian Joint Conference on Artificial Intelligence, 2006.

[9] Y. Hall, R. Poplin, "Using Numenta's Hierarchical Temporal Memory to Recognize CAPTCHAs", Carnegie Mellon University, 2007.

[10] I. Arel, D.C. Rose, and T.P. Karnowski, "Deep Machine Learning - A New Frontier in Artificial Intelligence Research", IEEE Computational Intelligence Magazine, vol. 5, no. 4.pp. 13-18, 2010.

[11] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 2 edition, 1998.

[12] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. Machine Learning, 1998.

[13] N. Oliver, A. Garg and E. Horvitz, "Layered Representations for Learning and Inferring Office Activity from Multiple Sensory Channels", Computer Vision and Image Understanding, vol. 96, pp. 163-180, 2004.

[14] J.Bromley and E. Sackinger, "Neural-Network and K-Nearest Neighbor Classifiers", Tech. Rep. 11359-91081916TM, AT&T, 1991.

[15] L. Wang et al., "Object Recognition Using a Bayesian Network Imitating Human Neocortex", proc. Int. Congress on Image and Signal Processing, 2009.

[16] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan-Kaufmann, 1988.

[17] D.G. Lowe, "Object Recognition from Local Scale-Invariant Features", proc. Int. Conf. Computer Vision, pp. 1150-1157, 1999.

[18] Y. LeCun, F.J. Huang and L. Bottou, "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting," proc. Conf. Computer Vision and Pattern Recognition (CVPR), 2004.

[19] M. Riesenhuber and T. Poggio, "Hierarchical Models of Object Recognition in Cortex", Nature Neuroscience, vol. 2, no. 11, pp. 1019-1025, 1999.

[20] S. Fine, Y. Singer and N. Tishby, "The Hierarchical Hidden Markov Model: Analysis and Applications", Machine Learning, vol. 32, p. 41-62, 1998.