ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Università degli Studi di Bologna
DEIS
Biometric System Laboratory

Pattern Recognition by Hierarchical Temporal Memory

Davide Maltoni

davide.maltoni@unibo.it

April 13, 2011

DEIS Technical Report

Pattern Recognition by Hierarchical Temporal Memory

Davide Maltoni, DEIS - University of Bologna (Italy)

davide.maltoni@unibo.it

***Abstract*— Hierarchical Temporal Memory (HTM) is still largely unknown by the pattern recognition community and only a few studies have been published in the scientific literature. This paper reviews HTM architecture and related learning algorithms by using formal notation and pseudocode description. Novel approaches are then proposed to encode coincidence-group membership (fuzzy grouping) and to derive temporal groups (maxstab temporal clustering). Systematic experiments on three line-drawing datasets have been carried out to better understand HTM peculiarities and to extensively compare it against other well-know pattern recognition approaches. Our results prove the effectiveness of the new algorithms introduced and that HTM, even if still in its infancy, compares favorably with other existing technologies.**

HIERARCHICAL temporal memory (HTM) is a biologically-inspired computational framework recently proposed by Hawkins and George [1-3] as a first practical implementation of the memory-prediction theory of brain function presented by Hawkins in [4]. A private company, called Numenta[1] [5], was setup to develop HTM technology and to make available to researches and practitioners a complete development platform. A number of technical reports and presentations are available in Numenta website [5] to describe HTM technology, application and results, but at today few independent studies [6-12] have been published to validate this computational framework and to frame it into the state-of-the-art.

HTM substantially differs from traditional neural network implementations (e.g., a multilayer perceptron) and can be conveniently framed into Deep Architectures [13][14]. In particular, Ranzato et al. [15] introduced the term *Multi-stage Hubel-Wiesel Architectures* (MHWA) to denote a specific subfamily of Deep Architectures. An MHWA is organized in alternating layers of feature detectors (reminiscent of Hubel and Wiesel's simple cells) and local pooling/subsampling of features (reminiscent of Hubel and Wiesel's complex cells); a final layer trained in supervised mode performs the classification. Neocognitron [16], Convolutional Networks [17][18], HMAX and its evolutions [19][20] are the best known implementations of MHWA. In analogy with MHWA, HTM alternates feature detection and feature pooling; however, in HTM feature pooling heavily relies on the temporal analysis of pattern sequences while in Neocognitron is hardwired and in Convolutional Network and HMAX is performed through simple spatial operators such as max or average. The temporal analysis and the modeling as a Bayesian Network make HTM similar in some aspects to Hierarchical [21] or Layered [22] versions of Hidden Markov Models (HMM); however, while HMM attempts to model the intrinsic temporal structure of input patterns[2], HTM exploits time continuity (mainly during learning) for unsupervised derivation of invariant representations, independently of the static or dynamic nature of the input patterns.

[1] the author of this paper has no business relationships with Numenta or with its founders, and has no commercial interest in promoting HTM technology.

[2] in fact, the most successful HMM applications are in domains where patterns have an intrinsic temporal structure (e.g., speech recognition) or a spatial structure that can be naturally decomposed in subsequent parts (e.g., handwriting recognition).

As pointed out by Hawkins and George "*... many of these ideas existed before HTMs and have been part of other models. The power of HTM comes from a unique synthesis of these ideas"*. In our opinion, HTM is the result of brilliant intuitions and clever engineering, and although HTM is still in its infancy, in the future it could help dealing with invariance which is the holy grail problem of pattern recognition and computer vision. Why HTM should overcome existing techniques in tackling invariance? There are some important properties that can be exploited to this purpose:

- *The use of time as supervisor*. A key problem in visual pattern recognition is that minor intra-class variations of a pattern can result in a substantially different spatial representation (e.g., in term of pixel intensities). Huge efforts have been done to develop variation-tolerant metrics (e.g., tangent distance [23]) or invariant feature extraction techniques (e.g., SIFT [24]), but to date, successful results have been achieved only for specific problems. HTM exploits time continuity to claim that two representations, even if spatially dissimilar, originate from the same object if they come close in time. This concept, which constitutes the basis of Slow Feature Analysis [25], is simple but extremely powerful because it is applicable to whatever form of invariance (i.e., geometry, pose, lighting). It also enables unsupervised learning: labels are provided by the time.
- *Hierarchical organization*. This is a largely used computation paradigm to put in practice the maxim "divide et impera". Recently a number of studies provided theoretical support to the advantages of hierarchical systems in learning invariant representations [13][26]. As the human brain HTM uses a hierarchy of levels to decompose object recognition complexity: at low levels the network learns basic features which are used as building blocks at higher levels to form representations of increasing complexity. Building blocks are also crucial for efficient coding and generalization since through their combination HTM can encode new objects never seen before.
- *Top down and bottom-up information flow*. In MHWA information typically flows one-way from lowers levels to upper levels. In the human cortex, both feed-forward and feed-back messages are continuously exchanged between different regions; although the precise role of feed-back messages is still very debated, neuroscientists agrees on their fundamental support in the perception of non-trivial patterns [4][27]. Memory-prediction theory postulates that feed-back messages from higher levels carry contextual information that can bias the behavior of lower levels. This is crucial to deal with uncertainty: if a node of a given level has to process an ambiguous pattern (e.g., a noisy version of an already encountered pattern) its decision could be better taken in presence of hints from upper levels, whose nodes are probably aware of the context the network is operating in (e.g., if one step back in time we were recognizing a car, probably we are still processing a traffic scene).
- *Bayesian probabilistic formulation*. Probabilistic decisions are often better than binary choices when dealing with uncertainty. The state of HTM nodes is encoded in probabilistic terms and Bayesian theory is largely used to process messages and fusing information. HTM can be viewed as a Bayesian Network where Bayesian Belief propagation equations are used to pass up and down the information across the hierarchy [28]. This formulation is not only elegant in mathematical terms, but also allows to solve practical burdens such as value normalization and threshold selection.

Although HTM can be used in a variety of contexts, in this paper we focus only on visual recognition applications (i.e., inputs are 2D images). We also ignore biological aspects of HTM theory: an excellent description of HTM biological underpinning is reported in [3] where its implementation in terms of biological circuits is presented.

When we started working with HTM we initially used the Numenta development platform, called Nupic [5] (most of the components are freely available to research organizations), but soon we decided to implement a new version from scratch: this is to have more flexibility and full control over the entire training/inference stages. Examples and experimental results reported throughout this paper have been obtained with our own HTM implementation.

The main contributions of this work are:

- an extensive description of HTM architecture (Sections 1, 2 and 3) and learning algorithms (Section 4) with consistent notation and pseudocode description;
- the introduction of novel approaches (Section 5) to encode coincidence-group membership more robustly (Fuzzy grouping) and to derive more stable temporal groups (MaxStab temporal clustering);
- the implementation of fast learning procedures, based on temporary data-buffering, to speed-up the training stage (Section 5.1.3);
- an extensive experimentation on three line-drawing datasets (Section 6) aimed at: (i) finding out optimal HTM architecture and parameters; (ii) assessing the effectiveness of Fuzzy grouping and MaxStab temporal clustering; (iii) comparing HTM with other existing approaches.
- further experiments to understand (and quantify) the efficacy of HTM mechanisms such as overlapped architectures (Section 6.2.3) and saccading (Section 6.2.5).

In Section 7 we draw some conclusions and summarize the huge amount of work we believe it is worth undertaking to overcome current HTM limitations and, hopefully, move some steps forward in solving challenging pattern recognition problems.

1. Overall HTM Structure

An HTM is a tree-like *network* composed of n_{levs} (≥ 2) *levels* $\mathcal{L}_i$ numbered from 0 to $n_{levs}-1$ (see Fig. 1). $\mathcal{L}_0$ is the *input* level; $\mathcal{L}_{n_{levs}-1}$ is the *output* level; $\mathcal{L}_i, i = 1 \dots n_{levs}-2$ are called *intermediate* levels (if n_{levs}= 2 the network has no intermediate levels). Each level $\mathcal{L}_i$ is composed of *nodes* $\mathcal{N}_j^{(i)}$, $j = 1 \dots n_{nodes}^{(i)}$. Nodes in input, intermediate and output levels are called input, intermediate and output nodes, respectively. To make notation lighter, a generic node can be denoted as $\mathcal{N}$ and a generic node at level i can be denoted as $\mathcal{N}^{(i)}$. When an HTM is used for visual pattern classification, typically:

- input nodes are in 1:1 relationship with image pixels;
- nodes in each level $\mathcal{L}_i$ are arranged in a rectangular grid (i.e., retinotopic mapping of the input);
- the network has only one output node, i.e. $n_{nodes}^{(n_{levs}-1)} = 1$, working as a pattern classifier;

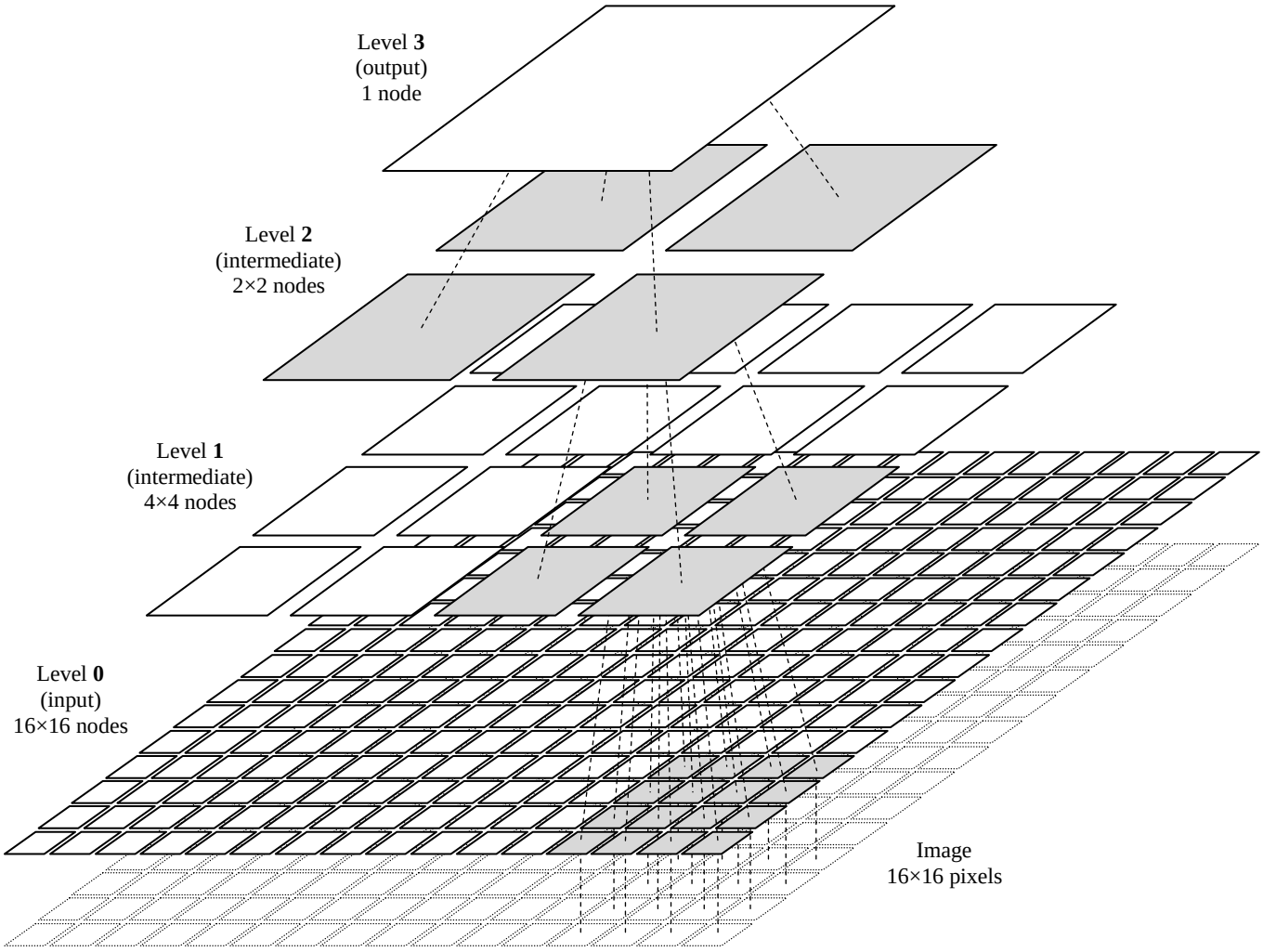

Fig. 1. A four-level HTM designed to work with 16x16 pixel images. Level 0 has 16x16 input nodes, each associated to a single pixel. Each level 1 node has 16 child nodes (arranged in a 4×4 region) and a receptive field of 16 pixels. Each level 2 node has 4 child nodes (2×2 region) and a receptive field of 64 pixels. Finally, the single output node at level 3 has 4 child nodes (2×2 region) and a receptive field of 256 pixels. In the figure only the downward connections of one node per level are shown.

- levels are sequentially interconnected through node connections: only connections between nodes in consecutive levels are allowed;
- each intermediate or output node $\mathcal{N}_j^{(i)}$ is connected to a set (called *region*) of spatially close *child* nodes in $\mathcal{L}_{i-1}$. Given a node $\mathcal{N}$, we denote with $childs(\mathcal{N})$ the set of its child nodes, with $n_{childs(\mathcal{N})}$ the number of its child nodes, and with $child_k(\mathcal{N}), k = 1 \dots n_{childs(\mathcal{N})}$ its k^{th} 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 $\mathcal{L}_{i-1}$ nodes to $\mathcal{L}_i$ nodes. For example, in the network of Fig. 1, $\mathcal{L}_0$ has 256 nodes arranged in a 16×16 grid whereas $\mathcal{L}_1$ has 16 nodes arranged in a 4×4 grid; each intermediate node $\mathcal{N}^{(1)}$ has 256/16=16 child nodes arranged in a (16/4)×(16/4) region;

- each input or intermediate node $\mathcal{N}^{(i)}$ is connected to a single *parent* node in $\mathcal{L}_{i+1}$. In the following, we denote with $parent(\mathcal{N})$ the parent node of $\mathcal{N}$. Actually, in some special configurations (see Section 6.2.3) the one-parent constraint is relaxed to allow the visual field of nodes in a given level to be partially *overlapped*;
- the *receptive field* (or visual field) of node can be conceived as the portion of input image that the node can see (i.e., the union of image pixels that can be reached by moving downward from the node). For input nodes, the receptive field is just one pixel. At higher levels a node receptive field is the union of its child receptive fields. As we move up in the hierarchy the receptive field gets larger: the receptive field of the output node is the entire image.

2. INFORMATION FLOW IN HTM

Information flow in HTM is bidirectional. Messages travelling *bottom-up* (*feed-forward* flow) are denoted with $\boldsymbol{\lambda}$ while messages travelling *top-down* (*feed-back* flow) are denoted with $\boldsymbol{\pi}$. Using the notation introduced by Pearl for Belief Propagation [28] and adjusted to HTM by Hawkins and George [1]:

- an input from below, denoted with e^-, is called *evidence*; in Bayesian terms, if e is a pattern, $p(e^-)$ corresponds to the pattern *density*;
- an input from above, denoted with e^+, is called *contextual information*; in Bayesian terms, if e is a pattern, $P(e^+)$ corresponds to the pattern *prior*;
- according to Bayes theorem, by fusing density with prior into a posterior probability we obtain the best probabilistic explanation of unknown patterns [28]. Analogously, by fusing bottom up and top down messages each HTM node reaches an internal state (called node *belief* and corresponding to Bayes posterior) which is an optimal probabilistic explanation of the external stimuli.

Although in the HTM framework feed-back flow is expected to be crucial for robust pattern classification, most of the practical achievements obtained until now rely on feed-forward flow only. This paper focuses on feed-forward flow. Details about feed-back equations can be found in [1] and the application of feed-back flow to segment out objects in cluttered scenes with multiple objects is presented in [3].

In the feed-forward flow each input or intermediate node $\mathcal{N}$ takes in input a message $\boldsymbol{\lambda}_i^- = p\left(e^-|child_i(\mathcal{N})\right)$ from each of its child nodes. The above equation means that $\boldsymbol{\lambda}_i^-$ corresponds to the conditional density[3] of the evidence given the status of $child_i(\mathcal{N})$. After internal processing of this information, the node produces an output $\boldsymbol{\lambda}^+ = p(e^-|\mathcal{N})$ for its parent node (see Fig. 2). Since node connections do not alter messages, output messages $\boldsymbol{\lambda}^+$ at level $\mathcal{L}_i$ coincide with input messages $\boldsymbol{\lambda}^-$ at level $\mathcal{L}_{i+1}$. Input messages to the output node (i.e., the single node in the

[3] Throughout this paper we often use the terms density (e.g., $p(e)$) and conditional density (e.g., $p(e|x)$). In the probability theory, $p(e)$ and $p(e|x)$ are density functions only if their summation (i.e. integral) over all possible values of e is 1. Since this constraint is not enforced in our formulation, we should define new functions $p'(e)$ and $p'(e|x)$ and claim that they are proportional to $p(e)$ and $p(e|x)$, where proportional means equal except for a normalizing factor. However, since the normalization factors have no influence on HTM information processing, we prefer to keep notation as simple as possible and to avoid such an intermediate definition.

output level) are equivalent to those of intermediate and input nodes, whereas the output message is a vector whose elements denote the (posterior) probability that the input pattern belongs to any of the problem classes $w_i, i = 1 \dots n_w$.

Feed-forward propagation of messages is performed level by level, starting from level 0. All $\mathcal{L}_i$ nodes must process their input $\boldsymbol{\lambda}^-$ (in any order) and produce their output $\boldsymbol{\lambda}^+$, before level $\mathcal{L}_{i+1}$ nodes can start their computation.

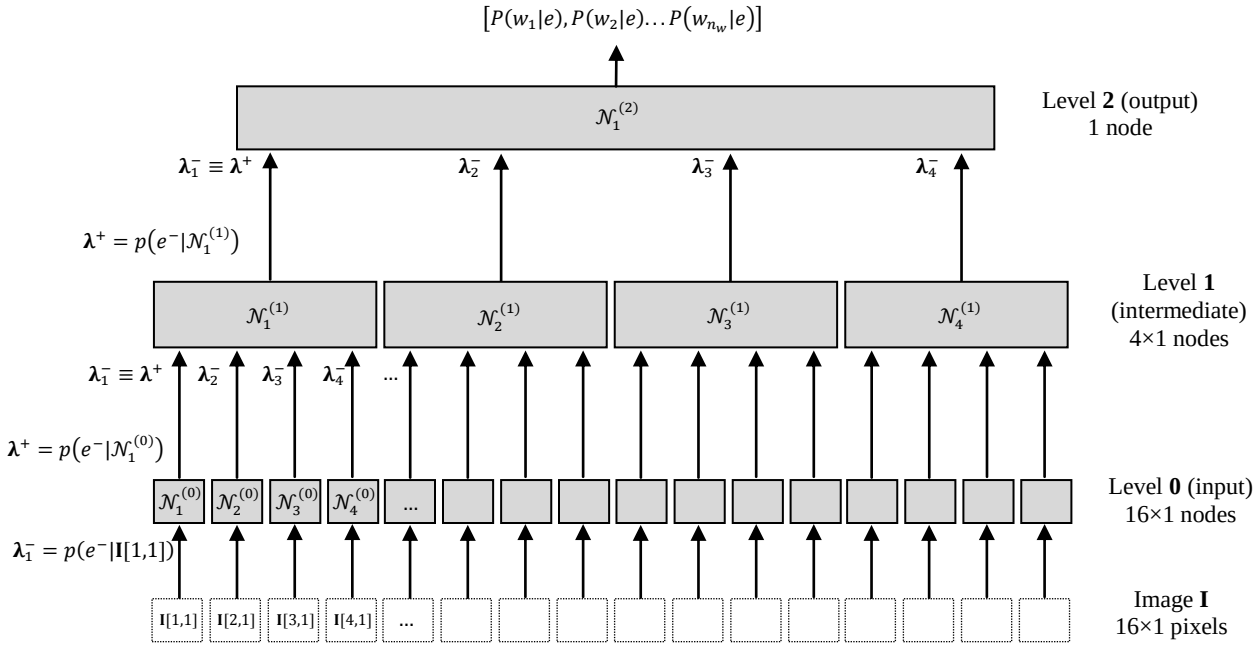

Fig. 2. A three-level HTM designed to work with 16×1 pixel images (such a special configuration allows to deal with one dimensional patterns). Feed-forward messages (on the left part of the network) are shown. For each node the input message $\boldsymbol{\lambda}_i^-$ coincide with the output messages $\boldsymbol{\lambda}^+$ of its i^{th} child node. The output message of the output node is a vector whose elements denote the probability that the input pattern belongs to any of the classes w_i.

3. NODE STRUCTURE

In the previous section we treated the network nodes as black boxes capable of transforming input messages into output ones. 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. Throughout this section we assume that the network nodes already undergone a *training* stage (node training is discussed in Sections 4 and 5) and therefore all the node internal data have been already initialized.

3.1 INPUT NODES

The structure of an input node $\mathcal{N}$ is very simple. $\mathcal{N}$ receives only one message from below. Let **I** be the input image, where $\mathbf{I}[x, y]$ denotes the image pixel at position x, y. Then, the input message, $\boldsymbol{\lambda}_1^- = p(e^-|\mathbf{I}[x, y])$ is a d-dimensional feature vector extracted from a local neighborhood of the image centered at $\mathbf{I}[x, y]$.

In the simplest case, if **I** is a grayscale image, a 1-dimensonal feature vector can be obtained as:

$\boldsymbol{\lambda}_1^- = graylevel(\mathbf{I}[x, y])$

However, better performance can be often achieved by using more powerful feature vectors such as the responses of a bank of Gabor filters:

$\boldsymbol{\lambda}_1^- = [Gabor_1(\mathbf{I}[x, y]), Gabor_2(\mathbf{I}[x, y]) \ldots Gabor_d(\mathbf{I}[x, y])]$

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:

$\boldsymbol{\lambda}^+ = \boldsymbol{\lambda}_1^-$.

3.2 INTERMEDIATE NODES

The internal structure of an intermediate node is shown in Fig. 3. The node maintains:

- a set **C** of *coincidences* $\mathbf{c}_i, i = 1 \ldots n_c$;
- a set **G** of *temporal groups* (or simply groups) $\mathbf{g}_j, j = 1 \ldots n_g$;
- a $n_c \times n_g$ matrix **PCG**.

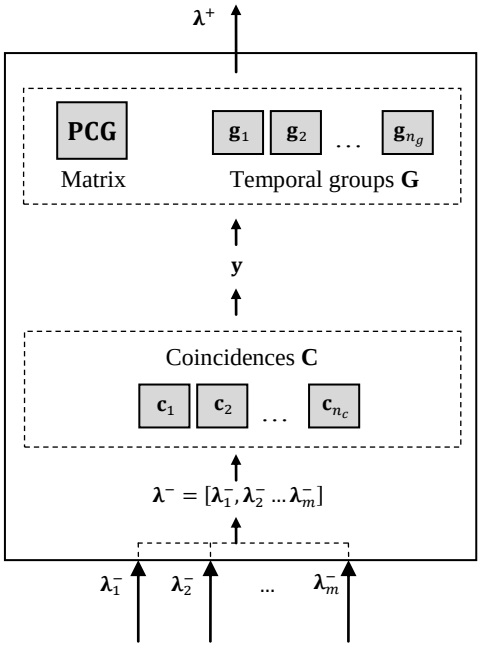

Fig. 3. An intermediate node working in inference mode.

3.2.1 Coincidences

Each coincidence is a sort of prototype pattern that spans a portion of the image corresponding to the node receptive field (i.e., small at low levels and large at high levels). Coincidences are used to perform a spatial analysis of input patterns and to find out spatial similarities. However, the coincidence structure depends on the node level:

- if $\mathcal{N}$ is an intermediate node at level 1 (hence its child nodes are input nodes), a coincidence corresponds to a small image patch. An example of coincidence graphical representation in a level 1 node is shown in Fig. 4 (left).

Note that the coincidence dimensionality is the same as the input message $\boldsymbol{\lambda}^-$ (i.e., the sum of the dimensionality of all the input messages coming from child nodes);

- if $\mathcal{N}$ is an intermediate node at level ≥ 2 (hence its child nodes are intermediate nodes), a coincidence $\mathbf{c}_i$ can be conceived as a feature selector: each element $\mathbf{c}_i[j]$ is the index of a single temporal group among the groups of $child_j(\mathcal{N})$. The dimensionality of coincidences is $m = n_{childs}(\mathcal{N})$. Although a graphical representation is here meaningless, a simple numerical example can help understanding: if $\mathcal{N}$ has 4 child nodes and $\mathbf{c}_1 = [5,3,1,1]$, then $\mathbf{c}_1$ selects: group 5 from child 1, group 3 from child 2, group 1 from child 3 and group 1 from child 4.

3.2.2 Temporal groups

A serious drawback of spatial-similarity-based 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 a short vertical bar (one pixel thick): the right (or left) movement of just one pixel is enough to dramatically reduce the spatial similarity with the original pattern. A *temporal group* (or simply group) is a subset of $\mathbf{C}$ 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. An example of level 1 temporal groups is shown in Fig. 4 (right). The name "temporal", as it will become clearer in Section 4, depends on the fact that HTM exploits temporal smoothness to create temporal groups; in other words, patterns that are presented to the network very close in time, are likely to be variants of the same pattern that is smoothly moving throughout the network receptive field.

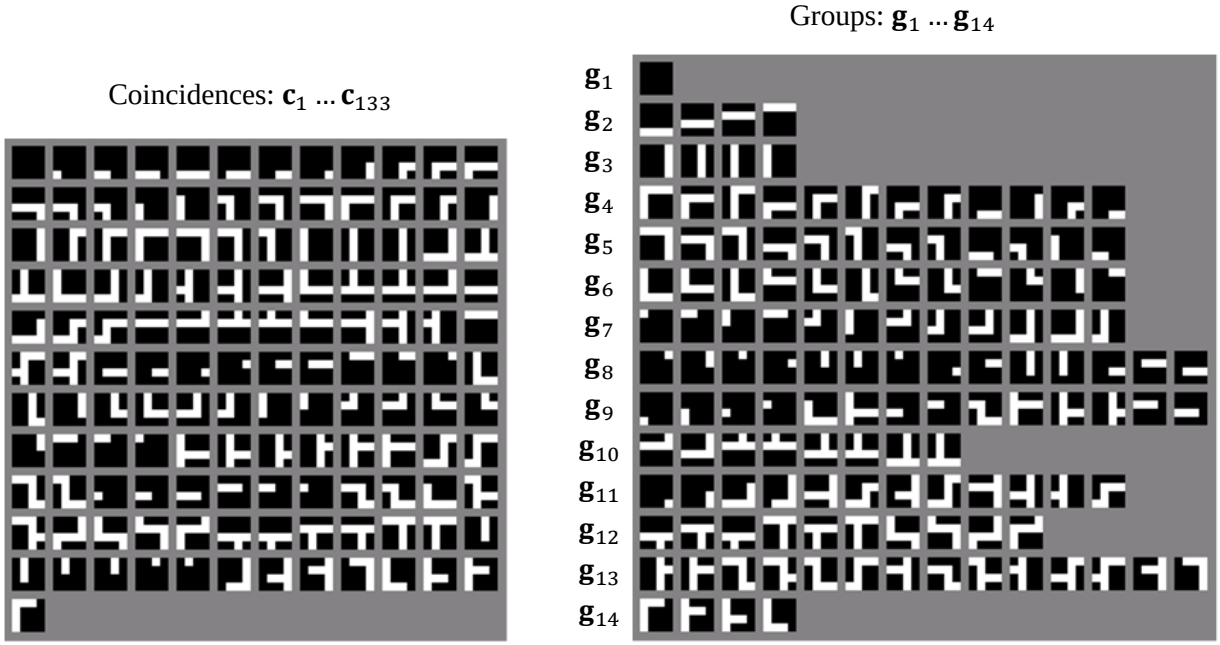

Fig. 4. Coincidences and temporal groups in a level 1 intermediate node trained on line-drawing patterns. Each of the 133 coincidences (on the left) is a 16=4×4 dimensional vector. On the right, graphical representation of 14 temporal groups (one per row). Group $\mathbf{g}_2$ denotes a horizontal (and $\mathbf{g}_3$ a vertical) bar at different positions within the node receptive field. Groups $\mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6$ and $\mathbf{g}_7$ correspond to four different types of corner. As explained in Section 4, HTM groups are not hardwired, but are the result of an unsupervised learning process.

3.2.3 PCG

PCG is a $n_c \times n_g$ matrix: element $\mathbf{PCG}[i,j] = P(\mathbf{c}_i|\mathbf{g}_j)$ denotes the conditional probability of coincidence $\mathbf{c}_i$ given the group $\mathbf{g}_j$, or, in other words, the relative probability of occurrence of coincidence $\mathbf{c}_i$ in the context of group $\mathbf{g}_j$. Hence, for each group $\mathbf{g}_j$, $\sum_{i=1}^{n_c} P(\mathbf{c}_i|\mathbf{g}_j) = 1$.

3.2.4 Inference steps

Inference in an intermediate node $\mathcal{N}$ can be decomposed in the following steps (see Fig. 3):

1. **Composition of input message**: a single input message $\boldsymbol{\lambda}^- = [\boldsymbol{\lambda}_1^-, \boldsymbol{\lambda}_2^- \ldots \boldsymbol{\lambda}_m^-]$ is obtained as juxtaposition of the $m = n_{childs}(\mathcal{N})$ input messages from the child nodes. The dimensionality d of $\boldsymbol{\lambda}$ is the sum of $\boldsymbol{\lambda}_i^-$ dimensionalities. In general $\boldsymbol{\lambda}_i^-$ dimensionality can vary across the child nodes.
2. **Computation of densities over coincidences**: vector $\mathbf{y}$ is composed by the conditional densities of the evidence given the coincidences: $\boldsymbol{y}[i] = p(e^-|\mathbf{c}_i), i = 1 \ldots n_c$. Intuitively each $\mathbf{y}[i]$ can be conceived as the *activation level* of coincidence $\mathbf{c}_i$ when the node input is $\boldsymbol{\lambda}^-$. $\mathbf{y}$ computation depends on the node level:
   - if $\mathcal{N}$ is an intermediate node at level 1 (hence its child nodes are input nodes), the input message is essentially an image patch and coincidences are prototype image patches. In this case $\mathbf{y}[i]$ encodes the spatial similarity between two image patches and can be conveniently computed as a Gaussian distance:
     $$\mathbf{y}[i] = e^{-\|\mathbf{c}_i - \boldsymbol{\lambda}^-\|^2/\sigma^2} \quad (1)$$
     where σ is a parameter controlling how quickly the activation level decays when $\boldsymbol{\lambda}^-$ deviates from $\mathbf{c}_i$. Fig. 5 shows an example of coincidence activations;
   - if $\mathcal{N}$ is an intermediate node at level ≥ 2 (hence its child nodes are intermediate nodes), the input message is a probability vector (see point 4 below). In this case $\mathbf{y}[i]$ is proportional to the probability of co-occurence of sub-evidences (each sub-evidence coming from a child), in the context of $\mathbf{c}_i$. Assuming the sub-evidences to be independent the probability is obtained by product rule:
     $$\mathbf{y}[i] = p(e^-|\mathbf{c}_i) = \prod_{j=1}^{m} p(e^-|child_j(\mathcal{N}), \mathbf{c}_i) = \prod_{j=1}^{m} \boldsymbol{\lambda}_j^-[\mathbf{c}_i[j]] \quad (2)$$
     where $\boldsymbol{\lambda}_j^-[\mathbf{c}_i[j]]$ is the element at position $\mathbf{c}_i[j]$ in input message $\boldsymbol{\lambda}_j^-$ from $child_j(\mathcal{N})$.
     For example, if $\mathcal{N}$ has 4 child nodes, $\mathbf{c}_1 = [5,3,1,1]$, $\boldsymbol{\lambda}_1^- = [0.05, 0.30, 0.70, 0.02, 0.23]$, $\boldsymbol{\lambda}_2^- = [0.15, 0.02, 0.18, 0.4]$, $\boldsymbol{\lambda}_3^- = [0.90, 0.02]$ and $\boldsymbol{\lambda}_4^- = [0.10, 0.42, 0.15]$, then $y_1 = 0.23 \times 0.18 \times 0.90 \times 0.10 = 3.726 \times 10^{-3}$;

   For numerical stability (i.e., to avoid that probabilities become too small as we move up in the hierarchy) it is preferable to normalize $\mathbf{y}$ such that $\sum_{i=1}^{n_c} \mathbf{y}[i] = 1$. This normalization does not alter the HTM behavior.
3. **Computation of densities over groups**: the conditional density over a group $\mathbf{g}_j$ (which intuitively can be conceived as the activation level of group $\mathbf{g}_j$) can be obtained by probability marginalization over the group coincidences:
   $$p(e^-|\mathbf{g}_j) = \sum_{i=1}^{n_c} p(e^-|\mathbf{c}_i, \mathbf{g}_j) \cdot P(\mathbf{c}_i|\mathbf{g}_j) = \sum_{i=1}^{n_c} p(e^-|\mathbf{c}_i) \cdot P(\mathbf{c}_i|\mathbf{g}_j) = \sum_{i=1}^{n_c} \mathbf{y}[i] \cdot \mathbf{PCG}[i,j] \quad (3)$$

where the assumption $p(e^-|\mathbf{c}_i, \mathbf{g}_j) = p(e^-|\mathbf{c}_i)$ holds because the knowledge of $\mathbf{g}_j$ is irrelevant for the estimation of e^- density in the context of $\mathbf{c}_i$. Fig. 5 shows an example of group activations.

4. **Composition of output message**: the output message $\boldsymbol{\lambda}^+ = p(e^-|\mathcal{N})$, whose dimensionality is n_g, is simply composed by the conditional densities over the groups: $\boldsymbol{\lambda}^+ = [p(e^-|\mathbf{g}_1), p(e^-|\mathbf{g}_2) \ldots p(e^-|\mathbf{g}_{n_g})]$.

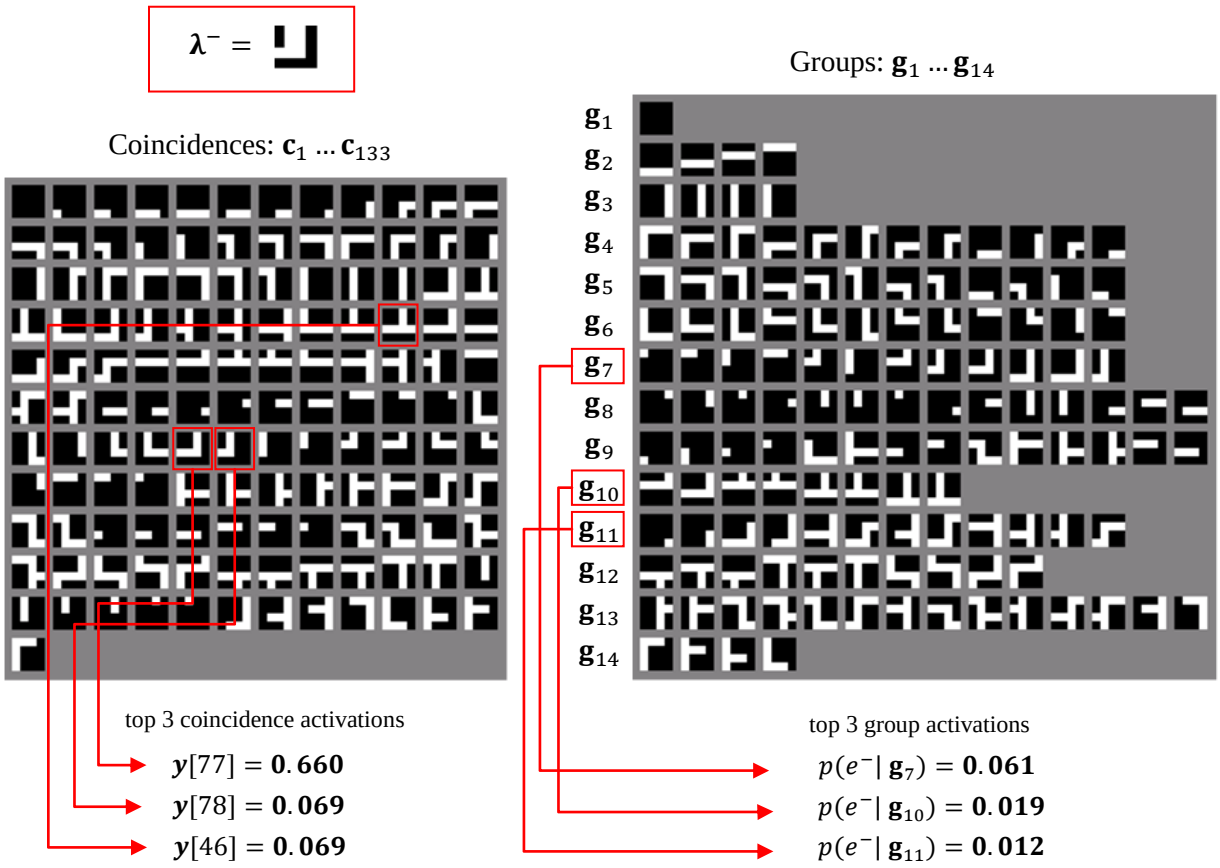

Fig. 5. This example shows, in the context of Fig. 4 intermediate node at level 1, the top 3 coincidence activations and the top 3 group activations produced by the input message $\boldsymbol{\lambda}^-$ graphically represented in the box located in the top-left part of this figure. In particular, even if the input patch is not identical to any of the node coincidences, it activates the three spatially closest coincidences. Group activations provide some generalization by associating the input patch to a corner-type pattern, independently of its precise location in the node receptive field.

3.3 Output nodes

The output node works as a pattern classifier. Its internal structure is shown in Fig. 6: the input part of the node is identical to an intermediate node, whereas in the output part group data are replaced by class data. The node maintains:

- a set $\mathbf{C}$ of *coincidences* $\mathbf{c}_i, i = 1 \ldots n_c$;
- a prior probability vector $[P(w_1), P(w_2) \ldots P(w_{n_w})]$ where $w_i, i = 1 \ldots n_w$ are the problem classes;
- a $n_c \times n_w$ matrix **PCW**.

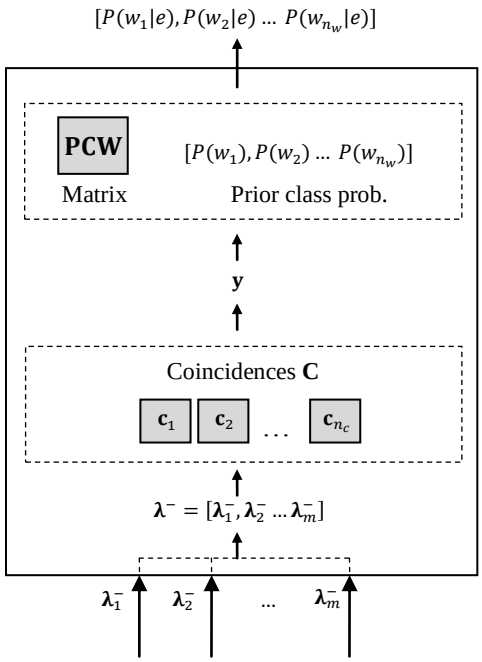

Fig. 6. The output node working in inference mode.

3.3.1 Coincidences

Output node coincidences are identical to intermediate node ones (see Section 3.2.1). However, except for degenerate cases where the network has no intermediate levels, the level of output node is ≥ 2 and therefore coincidences at this level work as feature selectors.

3.3.2 Prior class probabilities

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 $P(w_j), i = 1 \dots n_w$ are computed at training time.

3.3.3 PCW

PCW is a $n_c \times n_w$ matrix: element $\mathbf{PCW}[i,j] = P(\mathbf{c}_i|w_j)$ denotes the conditional probability of coincidence $\mathbf{c}_i$ given the class w_j, or, in other words, the relative probability of occurrence of coincidence $\mathbf{c}_i$ in the context of class w_j. Hence, for each class w_j, $\sum_{i=1}^{n_c} P(\mathbf{c}_i|w_j) = 1$.

3.3.4 Inference steps

Inference in the output node can be decomposed in the following steps (see Fig. 6):

1. **Composition of input message**: identical to intermediate nodes (see Section 3.2.4).
2. **Computation of densities over coincidences**: identical to intermediate nodes (see Section 3.2.4).

3. **Computation of densities over classes**: the conditional density over a class w_j (which intuitively can be conceived as the activation level of class w_j) can be obtained by probability marginalization over the class coincidences:
$$p(e^-|w_j) = \sum_{i=1}^{n_c} p(e^-|\mathbf{c}_i, w_j) \cdot P(\mathbf{c}_i|w_j) = \sum_{i=1}^{n_c} p(e^-|\mathbf{c}_i) \cdot P(\mathbf{c}_i|w_j) = \sum_{i=1}^{n_c} \mathbf{y}[i] \cdot \mathbf{PCW}[i,j] \quad (4)$$
where the assumption $p(e^-|\mathbf{c}_i, w_j) = p(e^-|\mathbf{c}_i)$ holds because the knowledge of w_j is irrelevant for the estimation of e^- density in the context of $\mathbf{c}_i$.
4. **Computation of class posterior probabilities**: according to Bayes theorem, class posterior probabilities can be obtained as: $P(w_j|e) = \frac{p(e^-|w_j)P(w_j)}{\sum_{k=1}^{n_w} p(e^-|w_k)P(w_k)}, j = 1 \dots n_w.$ (5)
5. **Composition of output message**: the output message $\boldsymbol{\lambda}^+$, whose dimensionality is n_w, is simply composed by the class posterior probabilities: $\boldsymbol{\lambda}^+ = [P(w_1|e), P(w_2|e) \dots P(w_{n_w}|e)]$, where $\sum_{i=1}^{n_w} P(w_i|e) = 1$.

4. NETWORK TRAINING

With network training we denote a batch procedure aimed at computing: (i) coincidences $\mathbf{C}$, groups $\mathbf{G}$ and $\mathbf{PCG}$ matrix for all intermediate nodes; (ii) coincidences $\mathbf{C}$, priors $P(w_i)$ and $\mathbf{PCW}$ matrix 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 set $\mathcal{S}_{train} = \{\langle \mathbf{I}_i, w_i \rangle | i = 1 \dots n_{train}\}$, where $\mathbf{I}_i$ is a pattern (e.g., a grayscale image) and w_i the corresponding class. Intermediate levels are trained in unsupervised mode (i.e., pattern classes are not used), whereas the output node is trained in supervised mode. HTM training is performed level by level, from $\mathcal{L}_1$ to $\mathcal{L}_{n_{levs}-1}$; $\mathcal{L}_0$ (input level) does not require any training. When training nodes at level $\mathcal{L}_i$, all the network nodes at previous levels, whose training was already finalized, work in inference mode. In Section 4.2 we will present the HTM training procedure in details, but before it is necessary to understand how training sequences are generated (Section 4.1).

4.1 TRAINING SEQUENCES

Training an intermediate level $\mathcal{L}_i$ requires to expose the network 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 (e.g., translation, rotation, scale, etc.) and photometric (e.g., brightness, color, etc.) features. Although different strategies can be designed to extract a sequence of temporally close patterns from a training set $\mathcal{S}_{train}$, a baseline implementation is as follows: for each pattern $\mathbf{I}_i \in \mathcal{S}_{train}$ perform two scans (an horizontal zig-zag followed by a vertical zig-zag) by moving the foreground object contained in $\mathbf{I}_i$ in all the positions within $\mathbf{I}_i$. Fig. 7 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 an horizontal bar, an 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* (see Fig. 7). Temporal gaps occur when we abruptly move a pattern to a distant position to start a new scan or when the training pattern changes (e.g., we stop moving $\mathbf{I}_1$ and start with $\mathbf{I}_2$).

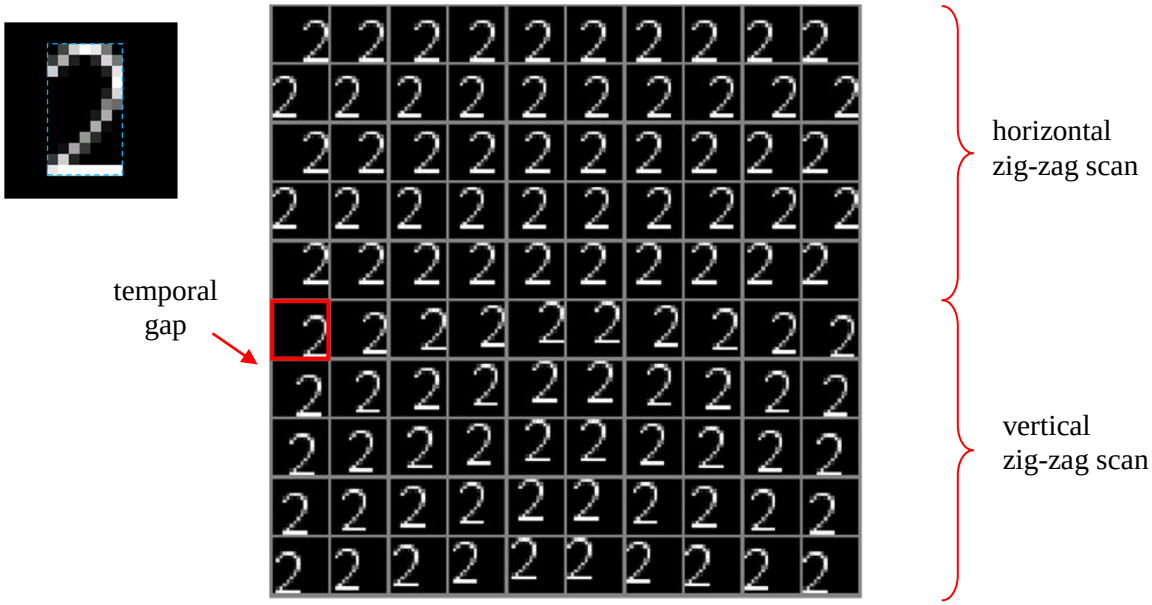

Fig. 7. An example of sequence obtained by the single 16×16 training image shown on the top-left (the foreground object it contains is highlighted with a dashed light blue rectangle). The first pattern, at the beginning of the second scan, is not obtained as a small movement of its predecessor in the sequence, hence it is marked as temporal gap.

Intermediate levels can operate in a special mode (denoted as *node sharing*): in this configuration all the level nodes share the same coincidences **C**, groups **G** and **PCG** matrix. When HTM are used for visual pattern recognition, node sharing is typically used for the bottom levels in the hierarchy (e.g., level 1 and/or level 2), whose nodes are expected to learn primitives such as bars, corners, etc. that can occur at any position in the image. Node sharing forces all the nodes of the level to respond in the same way to identical stimuli[4]. For levels working in shared mode, it is sufficient to train just one node (denoted as *master node*), and then cloning[5] **C**, **G** and **PCG** of the master node for all the other level nodes. When training a master node, the whole foreground object should be moved across the master node receptive field. In general this require to extend the movement of the foreground object outside the pattern boundaries[6]. A convenient strategy to generate such a sequence is shown in Fig. 8.

Finally, to train the output node it is sufficient to expose the node to single training patterns with associated class labels (in fact, no temporal information are processed by the output node). However, if the network is required to

[4] this is in analogy with early regions (e.g., V1 and V2) in the visual cortex.

[5] in practical HTM implementations, cloning is accomplished by using pointers to the master node data structures.

[6] note that Fig. 7 sequence does not allow the whole foreground object to be moved over the receptive field of a single (master) node. In fact, let us consider a level 1 master node with a receptive field of size 4×4: there is no 4×4 subwindow of the 16×16 image over which the entire foreground object is moved.

recognize patterns independently of their position (translation invariance), each training pattern must be presented at different positions. In practice, we can use training sequences like that reported in Fig. 7, but unlike for intermediate nodes, here only a single scan (either horizontal or vertical) is necessary.

Fig. 8. A convenient way to create the pattern sequence needed to train a master node, is to slide a window (whose size matches the node receptive field) across the foreground object. The example shows the sequence generated by an horizontal zig-zag scan.

4.2 Overall Training

A pseudo-code implementation of HTM training is here provided:

HTM Training

Reset coincidences // for all network nodes set $\mathbf{C} = \emptyset$
for each level $\mathcal{L}_i, i = 1 \dots n_{levs} - 1$
{ $\mathcal{Q}_i$ = Training Sequence for Level $\mathcal{L}_i$ from $\mathcal{S}_{train}$ // see Section 4.1
  $\langle \mathbf{I}, w^* \rangle$ = Get First Pattern from $\mathcal{Q}_i$ // w^* is the label of the pattern class
  while ($\mathbf{I}$ is not null)
  { Expose $\mathbf{I}$ to $\mathcal{L}_0$
    for each Level $\mathcal{L}_j, j = 0 \dots i - 1$
      Do Inference // see Sections 3.1 and 3.2.4
    Train $\mathcal{L}_i$ on $\langle \mathbf{I}, w^* \rangle$ // expanded below
    $\langle \mathbf{I}, w^* \rangle$ = Get Next Pattern from $\mathcal{Q}_i$
  }
  Finalize Training on $\mathcal{L}_i$ // expanded below
}

where:

Train $\mathcal{L}_i$ on $\langle \mathbf{I}, w^* \rangle$

if ($i = n_{levs} - 1$) // output level
  Train Output Node $\mathcal{N}_1^{(i)}$ on $\langle \mathbf{I}, w^* \rangle$ // see Section 4.4
else if ($\mathcal{L}_i$ is a *node sharing* Intermediate level)
  Train Intermediate Node $\mathcal{N}_1^{(i)}$ on $\mathbf{I}$ // see Section 4.3 (we assume that $\mathcal{N}_1^{(i)}$ is the master node)
else // intermediate level (no node sharing)
{ for each Node $\mathcal{N}_j^{(i)}, j = 1 \dots n_{nodes}^{(i)}$
    Train Intermediate Node $\mathcal{N}_j^{(i)}$ on $\mathbf{I}$ // see Section 4.3
}

and

Finalize Training on $\mathcal{L}_i$

if $(i = n_{levs} - 1)$ // output level
  Finalize Output Node Training $\mathcal{N}_1^{(i)}$ // see Section 4.4
else if ($\mathcal{L}_i$ is a *node sharing* Intermediate Level)
{ Finalize Intermediate Node Training $\mathcal{N}_1^{(i)}$ // see Section 4.3
  for each node $\mathcal{N}_j^{(i)}, j = 2 \dots n_{nodes}^{(i)}$
    Clone $\mathbf{C}, \mathbf{G}, \mathbf{PCG}$ from $\mathcal{N}_1^{(i)}$
}
else // intermediate level (no node sharing)
{ for each Node $\mathcal{N}_j^{(i)}, j = 1 \dots n_{nodes}^{(i)}$
    Finalize Intermediate Node Training $\mathcal{N}_j^{(i)}$ // see Section 4.3
}

4.3 INTERMEDIATE NODE TRAINING

The procedure Train Intermediate Node $\mathcal{N}^{(i)}$ on **I**, reported below, assumes that **I** has been presented to the network (level $\mathcal{L}_0$) and inference has been already performed until level $i - 1$. Hence messages $\boldsymbol{\lambda}_1^-, \boldsymbol{\lambda}_2^- \dots \boldsymbol{\lambda}_m^-$ are available from the m child nodes of $\mathcal{N}^{(i)}$.

Train Intermediate Node $\mathcal{N}^{(i)}$ on **I**

Compose Input Message $\boldsymbol{\lambda}^- = [\boldsymbol{\lambda}_1^-, \boldsymbol{\lambda}_2^- \dots \boldsymbol{\lambda}_m^-]$ // see 3.2.4, point 1
if $(i = 1)$ // node at level 1 → $\boldsymbol{\lambda}^-$ is an image patch
{ $k^* = \arg\min_{k=1\dots n_c}(\|\mathbf{c}_k - \boldsymbol{\lambda}^-\|)$ // select the coincidence closest to $\boldsymbol{\lambda}^-$, as candidate *active* coincidence
  if $(\|\mathbf{c}_{k^*} - \boldsymbol{\lambda}^-\| > thrDist)$ // none of the existing coincidences is spatially representative of $\boldsymbol{\lambda}^-$
    { $n_c = n_c + 1$ // increase number of coincidences
      $\mathbf{c}_{n_c} = \boldsymbol{\lambda}^-$ // add $\boldsymbol{\lambda}^-$ to **C** as a new coincidence
      $k^* = n_c$ // the active coincidence is the new one
    }
}
else // node at level ≥ 2 → $\boldsymbol{\lambda}^-$ is a probability vector
{ **widx** = Indices of Child Winning Groups in $\boldsymbol{\lambda}^-$ // see Equation 6
  $k^* = \arg\min_{k=1\dots n_c}(widxDistance(\mathbf{c}_k, \mathbf{widx}))$ // select the coincidence closest to **widx**, as candidate *active* coincidence
  if $(widxDistance(\mathbf{c}_{k^*}, \mathbf{widx}) > thrDist)$ // none of the existing coincidences is spatially representative of $\boldsymbol{\lambda}^-$
    { $n_c = n_c + 1$ // increase number of coincidences
      $\mathbf{c}_{n_c} = \mathbf{widx}$ // add **widx** to **C** as a new coincidence
      $k^* = n_c$ // the active coincidence is the new one
    }
}
$seen_{k^*} = seen_{k^*} + 1$ // number of times $\mathbf{c}_{k^*}$ was the active coincidence during training
if (**I** is not a *temporal gap* pattern)
  $\mathbf{T}[k_{prev}^*, k^*] = \mathbf{T}[k_{prev}^*, k^*] + 1$ // updates Temporal Activation Matrix
$k_{prev}^* = k^*$ // remember the active coincidence for next step

where:

- the *active* coincidence is the spatially closest coincidence to the node input. If all existing coincidences are too dissimilar from the input (with respect to a level specific threshold $thrDist$), then a new coincidence is created and selected as active coincidence. 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 (see Equation 3). It is worth noting that such an asymmetrical approach (winner take all for learning vs continuous for inference) is not atypical, and proved to be quite effective in training other deep architectures [15][30];
- **widx** is a vector of indices of child winning groups. Index **widx**$[j]$ is the group index, within the groups of child node $child_j(\mathcal{N}^{(i)})$, which obtained maximum activation. Let n_j be the number of groups in $child_j(\mathcal{N}^{(i)})$, and $\boldsymbol{\lambda}_j^-[k]$ be the k^{th} element of vector $\boldsymbol{\lambda}_j^-$, then:

  $$\mathbf{widx}[j] = \arg\max_{k=1\dots n_j} \left(\boldsymbol{\lambda}_j^-[k]\right) \quad (6)$$

  Here too a winner take all criterion is adopted. In fact, only the index of the most active group within each child node is considered. 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 (see Equation 2);
- $widxDistance(\mathbf{c}_k, \mathbf{widx})$ counts the number of differences between indices at corresponding positions in the two vectors. For example, if $\mathbf{c}_k = [3,5,1,2,5]$ and $\mathbf{widx} = [3,4,1,2,4]$, then $widxDistance(\mathbf{c}_k, \mathbf{widx}) = 2$;
- A scalar $seen_j$ is maintained for each coincidence $\mathbf{c}_j$ to count the number of times $\mathbf{c}_j$ was active during the node training. When finalizing node training these values will be used to quantify coincidence relevance;
- **T**, denoted as Temporal Activation Matrix (or TAM), is an $n_c \times n_c$ matrix, 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 **T** will be used to compute temporal groups **G** and **PCG**. For simplicity of presentation, in the above pseudocode **T** update is performed by looking only one step back in time (i.e., through k^*_{prev}). In general, better performance can be achieved by considering $transitionMemory$ steps back: let $k^*_{prev_{(t)}}$ be the index of the active coincidence t steps back in time, then:

for each $t = 1 \dots transitionMemory$

$$\mathbf{T}\left[k^*_{prev_{(t)}}, k^*\right] = \mathbf{T}\left[k^*_{prev_{(t)}}, k^*\right] + (1 + transitionMemory - t)$$

where linearly decreasing weights are here used to update **T** as the time gap increases.

Once all the patterns in the training sequence have been presented to the node, the intermediate node training can be finalized as:

Finalize Intermediate Node Training $\mathcal{N}^{(i)}$	
Forget Rare Coincidences	// coincidence $\mathbf{c}_j$ such that $seen_j < forgetThr$ are removed from $\mathbf{C}$
Compute Coincidence Priors	// see Equation 7
Make $\mathbf{T}$ Symmetric	// after this step: $\mathbf{T}[i,j] = \mathbf{T}[j,i] \;\; i,j = 1 \dots n_c$
Normalize $\mathbf{T}$ by Rows	// after this step: $\mathbf{T}[i,j] = P\left(\mathbf{c}_j^{(t)} \mid \mathbf{c}_i^{(t-1)}\right)$, see Equation 8
Temporal Grouping	// determine groups $\mathbf{G}$ from $\mathbf{T}$ clustering, see Section 4.3.1
Compute $\mathbf{PCG}$	// see Section 4.3.2

where:

- 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 $\mathbf{T}$ symmetric the upper diagonal part is summed to the lower diagonal part [7]:
  $\mathbf{T}^{(new)}[i,j] = \mathbf{T}^{(new)}[j,i] = (\mathbf{T}[i,j] + \mathbf{T}[j,i])$ for each pair $\langle i = 1 \dots n_c, j = 1 \dots n_c \rangle, \; j \geq i$
  Making $\mathbf{T}$ 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;
- coincidence priors can be simply obtained by normalizing the number of times coincidences have been activated during training:
  $P(\mathbf{c}_j) = seen_j / \left(\sum_{k=1 \dots n_c} seen_k\right), \; j = 1 \dots n_c$ (7)
- $\mathbf{T}$ values are proportional to the probability of (close in time) co-occurrence of coincidences. A simple normalization by rows makes $\mathbf{T}$ values true conditional probabilities:
  $\mathbf{T}^{(new)}[i,j] = \mathbf{T}[i,j] / \left(\sum_{k=1 \dots n_c} \mathbf{T}[i,k]\right)$ for each $i = 1 \dots n_c, j = 1 \dots n_c$
  After normalization:
  $\mathbf{T}[i,j] = P\left(\mathbf{c}_j^{(t)} \mid \mathbf{c}_i^{(t-1)}\right)$ (8)
  where $\mathbf{c}_j^{(t)}$ means that $\mathbf{c}_j$ was active at time t, and $\sum_{j=1 \dots n_c} \mathbf{T}[i,j] = 1$, for each $i = 1 \dots n_c$. Note that after normalization $\mathbf{T}$ is no longer symmetric.

Some further definitions are useful before discussing group computation:

- Equation 8 asserts that $\mathbf{T}[i,j]$ is the probability that the next active coincidence will be $\mathbf{c}_j$ if the current active coincidence is $\mathbf{c}_i$; in other words, $\mathbf{T}[i,j]$ denotes the *temporal connection* of the (ordered) pair $\mathbf{c}_i$, $\mathbf{c}_j$;
- the temporal connection TC of a single coincidence $\mathbf{c}_j$ is the probability that the next active coincide will be $\mathbf{c}_j$ independently of the currently active coincidence, and can be obtained from 8 by marginalization:
  $TC(\mathbf{c}_j) = \sum_{k=1 \dots n_c} P\left(\mathbf{c}_j^{(t)} \mid \mathbf{c}_k^{(t-1)}\right) \cdot P\left(\mathbf{c}_k^{(t-1)}\right)$ (9)
  where we assume that $P\left(\mathbf{c}_k^{(t-1)}\right)$ are the prior probabilities obtained from Equation 7, and therefore:

[7] throughout this paper, for Equations that require updating a whole matrix/vector by overwriting the same matrix/vector, we denote the target with the superscript *new*, in order to avoid any ambiguity due to possible interfering updates. This does not mean that updating require a temporary copy of the data structure.

$$TC(\mathbf{c}_j) = \sum_{k=1\dots n_c} \mathbf{T}[k,j] \cdot P(\mathbf{c}_k) \quad (10)$$

- the temporal connection TC of a group $\mathbf{g}_k$ is the average temporal connection between any two coincidences belonging to the group. Let $|\mathbf{g}_k|$ be the number of coincidences in $\mathbf{g}_k$, then:

$$TC(\mathbf{g}_k) = \frac{1}{|\mathbf{g}_k|^2} \sum_{\mathbf{c}_i \in \mathbf{g}_k} \sum_{\mathbf{c}_j \in \mathbf{g}_k} \mathbf{T}[i,j] \quad (11)$$

4.3.1 Temporal Grouping by T Clustering

A temporal group $\mathbf{g}_k$ is a set of coincidences that are likely to occur close in time. Partitioning $\mathbf{C}$ coincidences into a set of disjoint groups $\mathbf{G} = \{\mathbf{g}_1, \mathbf{g}_2, \dots \mathbf{g}_{n_g}\}$, can be formulated as a clustering problem aimed at maximizing the functional:

$$J_1(\mathbf{G}) = \frac{1}{n_g} \sum_{k=1}^{n_g} TC(\mathbf{g}_k) \quad (12)$$

subject to the constraints:

- $\mathbf{g}_i \cap \mathbf{g}_j = \emptyset$ for each $i, j,\ i \neq i$ (13)
- $\cup_{i=1\dots n_g} \mathbf{g}_i = \mathbf{C}$ (14)
- $|\mathbf{g}_i| \leq groupMaxSize$ for each i (15)

Equation 13 asserts that groups must be disjoint, equation 14 that all the coincidences must be assigned to groups and Equation 15 sets a maximum group size. Maximization of 12 leads 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: (i) 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 (this makes the application of *k-means* like approaches critical); (ii) the number of coincidences and groups can be quite large in practical applications, so we need computationally efficient approaches; (iii) 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 algorithm, here denoted as *Default temporal clustering*, was introduced in [32]:

Default Temporal Clustering

$\mathbf{G} = \emptyset,\ n_g = 0$
$assigned(\mathbf{c}_j) = false,\ j = 1 \dots n_c$ // a flag is maintained to denote coincidences already assigned
while (not all coincidences have been *assigned*)
{ $k^* = \arg\max_{\substack{k=1\dots n_c \\ assigned(\mathbf{c}_k)=false}} TC(\mathbf{c}_k)$ // select the non assigned coincidence with highest temporal connection
$\mathbf{\Omega} = \{\mathbf{c}_{k^*}\}$ // initialize a new list $\mathbf{\Omega}$ with a single coincidence
$pos = 1$ // this is a cursor used to scan $\mathbf{\Omega}$ elements by their position
while ($pos \leq len(\mathbf{\Omega})$ and $pos < groupMaxSize$) // until scan is completed or group is too large; $len(\mathbf{\Omega})$ is the list length
{ $\mathbf{c}_k = \mathbf{\Omega}[pos]$ // get coincidence at position pos in the list
$\mathbf{\Omega} = \mathbf{\Omega} + TopMostConnected(\mathbf{c}_k)$ // add to the tail of $\mathbf{\Omega}$ the $topNeighbors$ coincidences that are the most connected to $\mathbf{c}_k$, see below.

```
        pos = pos + 1
    }
    n_g = n_g + 1                          // create a new empty group and add it to G
    g_{n_g} = ∅
    G = G ∪ {g_{n_g}}
    for each j = 1 ... pos                 // add the first pos coincidences in Ω to the new group
    {   c_j = Ω[j]
        g_{n_g} = g_{n_g} ∪ {c_j}
        assigned(c_j) = true               // mark it as assigned
    }
}
```

where:

- $TopMostConnected(\mathbf{c}_k)$ selects the $topNeighbors$ coincidences $\mathbf{c}_i$ with highest temporal connection $\mathbf{T}[k,i]$ with $\mathbf{c}_k$, excluding coincidences already assigned. Selected coincidences are added to $\mathbf{\Omega}$, by inserting them at the end of the list and by excluding coincidences already present in the list. A typical value for $topNeighbors$ is 3.

The default temporal clustering algorithm runs by creating one group at each time. The group seed is a single highly connected coincidence, to which its $TopMostConnected$ coincidence are associated; group growing is recursive, i.e., each newly associated coincidence will cause its $TopMostConnected$ coincidences to be associated as well. Recursion terminates: (i) naturally, when the $TopMostConnected$ coincidences of all the coincidences in the list are already in the list; (ii) forcedly, if the list length exceeds $groupMaxSize$. Some nice graphical examples of group growing are shown in [32]. Temporal grouping shown in Fig. 4 has been created with the default temporal clustering algorithm. A different algorithm, based on Agglomerative Hierarchical Clustering, was proposed in [1].

4.3.2 PCG Computation

$\mathbf{PCG}[i,j] = P(\mathbf{c}_i|\mathbf{g}_j)$ denotes the conditional probability of coincidence $\mathbf{c}_i$ given the group $\mathbf{g}_j$. The computation of **PCG** matrix is performed in two simple steps:

1. $\mathbf{PCG}[i,j] = \begin{cases} P(\mathbf{c}_i) & \text{if } \mathbf{c}_i \in \mathbf{g}_j \\ 0 & \text{otherwise} \end{cases}$, for each $i = 1 \dots n_c, j = 1 \dots n_g$
2. $\mathbf{PCG}^{(new)}[i,j] = \mathbf{PCG}[i,j] / \left(\sum_{k=1\dots n_c} \mathbf{PCG}[k,j]\right)$, for each $i = 1 \dots n_c, j = 1 \dots n_g$

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 $\mathbf{g}_j$, that $\sum_{i=1\dots n_c} P(\mathbf{c}_i|\mathbf{g}_j) = 1$.

4.4 Output Node Training

Training the output node during pattern presentation is very simple:

Train Output Node $\mathcal{N}^{(i)}$ on $\langle \mathbf{I}, w^* \rangle$	
...	// the first part, aimed at selecting/creating the active coincidence $\mathbf{c}_{k^*}$, is identical to Intermediate Node Training, see Section 4.3.
$\mathbf{PCW}[k^*, w^*] = \mathbf{PCW}[k^*, w^*] + 1$	// increase the number of times $\mathbf{c}_{k^*}$ was active in the context of class w_{w^*}

A few steps are required to finalize the output node training:

Finalize Output Node Training $\mathcal{N}^{(i)}$	
Forget Rare Coincidences	// coincidence $\mathbf{c}_j$ such that $\sum_{k=1\dots n_w} \mathbf{PCW}[j,k] < forgetThr$ are removed from $\mathbf{C}$
Compute Class Priors $P(w_j), j = 1 \dots n_w$	// see Equation 16
Normalize **PCW**	// see Equation 17

where:

- $\sum_{k=1\dots n_w} \mathbf{PCW}[j,k]$ is the total number of times coincidence $\mathbf{c}_j$ has been active independently of the pattern class; these values are here used to forget rare coincidences;
- class Priors $P(w_j),\ j = 1 \dots n_w$ are computed by **PCW** marginalization and normalization:

$$P(w_j) = \frac{\sum_{i=1\dots n_c} \mathbf{PCW}[i,j]}{\sum_{k=1\dots n_w} \sum_{i=1\dots n_c} \mathbf{PCW}[i,k]} \tag{16}$$

- **PCW** normalization is aimed at guaranteeing, for each class w_j, that $\sum_{i=1\dots n_c} P(\mathbf{c}_i|w_j) = 1$:

$$\mathbf{PCW}^{(new)}[i,j] = \mathbf{PCW}[i,j] / \left(\sum_{k=1\dots n_c} \mathbf{PCW}[k,j]\right) \ , \quad \text{for each } i = 1 \dots n_c, j = 1 \dots n_w. \tag{17}$$

5. New Training Algorithms

In this Section we introduce new training techniques: in particular, Section 5.1.1 presents a new temporal clustering algorithm, Section 5.1.2 introduces a fuzzy grouping approach, and finally in Section 5.1.3 we discuss computational issues and show how activation buffering can markedly reduce training time.

A new constructive definition of $P(\mathbf{c}_i|\mathbf{g}_j)$ is fundamental for the discussion hereafter. Let $\mathbf{G} = \{\mathbf{g}_1, \mathbf{g}_2, \dots \mathbf{g}_{n_g}\}$ be a temporal grouping solution and **T** be a normalized temporal activation matrix (see Equation 8), then:

$$P(\mathbf{c}_i|\mathbf{g}_j) = \sum_{\mathbf{c}_k \in \mathbf{g}_j} P(\mathbf{c}_i{}^{(t)} \mid \mathbf{c}_k{}^{(t-1)}, \mathbf{g}_j) \cdot P(\mathbf{c}_k{}^{(t-1)}|\mathbf{g}_j) \tag{18}$$

where:

- $P(\mathbf{c}_i{}^{(t)} \mid \mathbf{c}_k{}^{(t-1)}, \mathbf{g}_j) = P(\mathbf{c}_i{}^{(t)} \mid \mathbf{c}_k{}^{(t-1)}) = \mathbf{T}[k,i]$, since given $\mathbf{c}_k{}^{(t-1)} \in \mathbf{g}_j$, the knowledge of $\mathbf{g}_j$ is irrelevant to determine $\mathbf{c}_i{}^{(t)}$;
- $P(\mathbf{c}_k{}^{(t-1)}|\mathbf{g}_j)$ is computed as the relative prior probability of $\mathbf{c}_k$ (see Equation 7) over the total prior probability of coincidences belonging to group $\mathbf{g}_j$:

$$P(\mathbf{c}_k{}^{(t-1)}|\mathbf{g}_j) = P(\mathbf{c}_k) / \left(\sum_{\mathbf{c}_l \in \mathbf{g}_j} P(\mathbf{c}_l)\right).$$

Hence Equation 18 becomes:

$$P(\mathbf{c}_i|\mathbf{g}_j) = \sum_{\mathbf{c}_k \in \mathbf{g}_j} \left(\mathbf{T}[k,i] \frac{P(\mathbf{c}_k)}{\sum_{\mathbf{c}_l \in \mathbf{g}_j} P(\mathbf{c}_l)} \right) \tag{19}$$

It can be simply proved that, for each group $\mathbf{g}_j$:

$$\sum_{i=1\dots n_c} P(\mathbf{c}_i|\mathbf{g}_j) = 1 \tag{20}$$

in fact, being $\sum_{i=1\dots n_c} \mathbf{T}[k,i] = 1$ (see Equation 8), then:

$$\sum_{i=1\dots n_c} P(\mathbf{c}_i|\mathbf{g}_j) = \sum_{i=1\dots n_c} \sum_{\mathbf{c}_k \in \mathbf{g}_j} \left(\mathbf{T}[k,i] \frac{P(\mathbf{c}_k)}{\sum_{\mathbf{c}_l \in \mathbf{g}_j} P(\mathbf{c}_l)} \right) = \frac{1}{\sum_{\mathbf{c}_l \in \mathbf{g}_j} P(\mathbf{c}_l)} \sum_{\mathbf{c}_k \in \mathbf{g}_j} P(\mathbf{c}_k) \sum_{i=1\dots n_c} \mathbf{T}[k,i] = \frac{\sum_{\mathbf{c}_k \in \mathbf{g}_j} P(\mathbf{c}_k)}{\sum_{\mathbf{c}_l \in \mathbf{g}_j} P(\mathbf{c}_l)} = 1.$$

According to Equations 18 (and 19) the conditional probability of a coincidence $\mathbf{c}_i$ given a group $\mathbf{g}_j$ can be conceived as the probability that the next active coincidence will be $\mathbf{c}_i$ if the currently active coincidence is one of the coincidences of the group $\mathbf{g}_j$. It is worth noting that Equation 19 allows to compute the degree of membership of a coincidence to a group either if the coincidence belongs to the group or not.

Equation 19 also allows to define the *stability* of a group $\mathbf{g}_j$:

$$stability(\mathbf{g}_j) = \sum_{\mathbf{c}_i \in \mathbf{g}_j} P(\mathbf{c}_i|\mathbf{g}_j) \quad (21)$$

$stability(\mathbf{g}_j)$ is the probability that if the currently active coincidence belongs to $\mathbf{g}_j$ then the next active coincidence will also belong to $\mathbf{g}_j$. It can be simple proved that $stability(\mathbf{g}_j)$ always lies in [0,1], where 1 means maximum stability; in fact, $stability(\mathbf{g}_j)$ is smaller than or equal to the summation in Equation 20.

It should be noted that the definition of group stability is quite similar to that of group temporal connection (see Equation 11): the only difference is that to compute group stability we make use of prior probabilities $P(\mathbf{c}_i)$ to weight $\mathbf{T}[i,j]$ values non uniformly.

5.1.1 MaxStab Temporal Clustering

The default temporal clustering approach introduced in Section 4.3.1 indirectly maximizes functional J_1 (Equation 12) by forming groups with high internal $\mathbf{T}[i,j]$ values. The algorithm here introduced performs a more direct maximization of the average group stability, expressed by the functional:

$$J_2(\mathbf{G}) = \frac{1}{n_g} \sum_{k=1}^{n_g} stability(\mathbf{g}_k) \quad (22)$$

subject to the constraints 13, 14 and 15.

MaxStab Temporal Clustering

$\mathbf{G} = \emptyset,\ n_g = 0$
$assigned(\mathbf{c}_j) = false,\ j = 1 \dots n_c$ // a flag is maintained to denote coincidences already assigned
while (not all coincidences have been *assigned*)
{ $k^* = \arg\max_{\substack{k=1\dots n_c \\ assigned(\mathbf{c}_k)=false}} TC(\mathbf{c}_k)$ // select the non assigned coincidence with highest temporal connection
$n_g = n_g + 1$ // create a new empty group and add it to $\mathbf{G}$
$\mathbf{g}_{n_g} = \emptyset$
$\mathbf{G} = \mathbf{G} \cup \{\mathbf{g}_{n_g}\}$
do
{ $\mathbf{g}_{n_g} = \mathbf{g}_{n_g} \cup \{\mathbf{c}_{k^*}\}$
$assigned(\mathbf{c}_{k^*}) = true$
$k^* = \arg\max_{\substack{k=1\dots n_c \\ assigned(\mathbf{c}_k)=false}} \Delta stab\left(\mathbf{g}_{n_g}, \mathbf{c}_k\right)$ // get coincidence that most increases group $\mathbf{g}_{n_g}$ stability
}
while ($\Delta stab\left(\mathbf{g}_{n_g}, \mathbf{c}_{k^*}\right) \geq \frac{min\Delta stab}{n_c}$ and $|\mathbf{g}_{n_g}| < groupMaxSize$)
}

where:

- $\Delta stab(\mathbf{g}_{n_g}, \mathbf{c}_k)$ computes the delta stability resulting from the inclusion of $\mathbf{c}_k$ in $\mathbf{g}_{n_g}$.

MaxStab creates one group at each time starting from a single highly connected coincidence. The group is then expanded by associating, step by step, the coincidence that most increases the group stability. The expansion continues while: (i) the increase in stability is larger than a given threshold computed as $min\Delta stab/n_c$, and (ii) the group size is smaller than $groupMaxSize$.

Because of the similarity between group stability and temporal connection both J_1 and J_2 maximization are expected to give similar results. However in our experiments, MaxStab usually leads to an higher average group stability (Equation 22) with respect to the default temporal clustering of Section 4.3.1, and this often results in better classification performance. On the other hand, since HTM networks are quite robust with respect to suboptimal grouping the accuracy improvement is often marginal. A graphical comparison of the solutions obtained with the two algorithms is shown in Fig. 9.

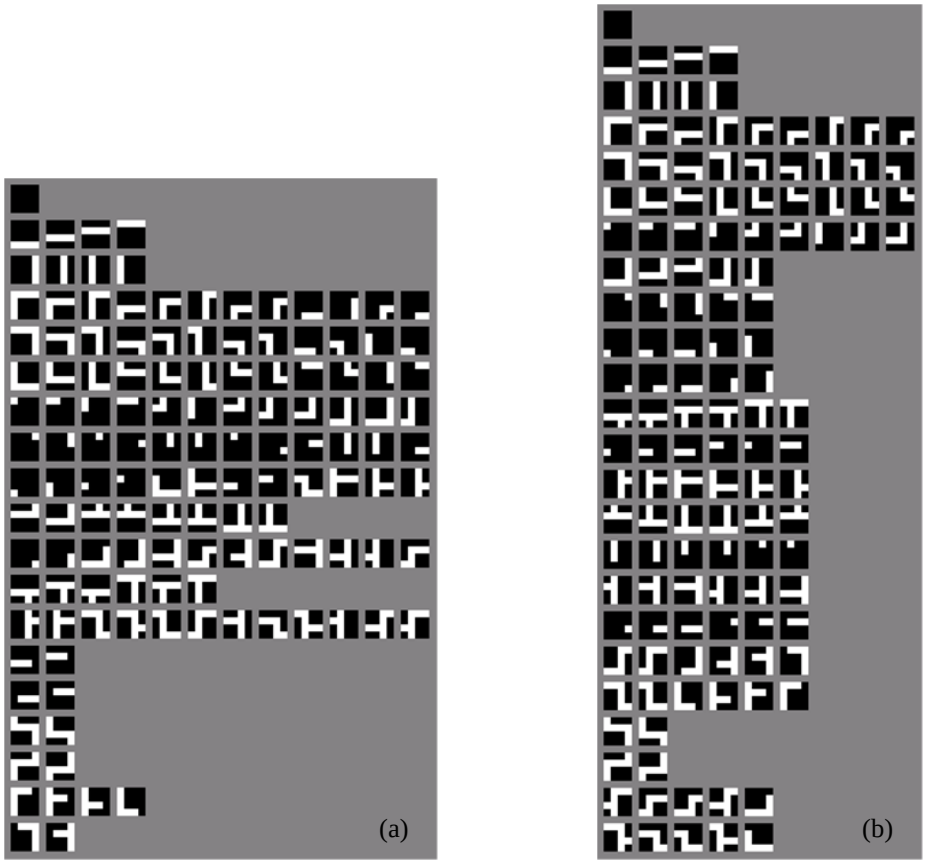

Fig. 9. Two temporal groupings over the coincidences of Fig. 4 (left) starting from the same temporal activation matrix: (a) clustering was performed with the default temporal grouping algorithm with parameter values: $groupMaxSize = 12$, $topNeighbors = 3$. The first group, i.e., that containing a single empty patch, was hardcoded; in fact, we noted that for line drawing classification, in case of perfectly clean background, this often leads to better accuracy. The average group stability obtained is $J_2 = 39\%$. (b) clustering was performed with MaxStab algorithm with parameter values: $groupMaxSize = 12$, $min\Delta stab = 0.33$. Unlike for the default algorithm, no group was hardcoded here, since MaxStab can create groups with a single element. The average group stability obtained is $J_2 = 42\%$. Finally it should be noted that groups in (b) are better balanced and that while in (a) group growing was terminated in 8 cases by the maximum group size constraint, in (b) group growing always terminated naturally because of the threshold imposed by $min\Delta stab$.

5.1.2 Fuzzy Grouping

In Section 4.3.1, Equation 13 requires the groups to be disjoint (i.e., no coincidence can be part of more than one group) and Equation 14 requires all coincidences to be assigned to one group. In real applications, rarely clusters can be clearly identified and even for optimal solutions some patterns can lie near the boundaries of two of more clusters. Forcing patterns to be member of only one cluster can lead to ambiguity. For this reason in many pattern recognition applications, probabilistic or fuzzy clustering, such as fuzzy-k-means [33] or Expectation-Maximization [34] is preferred to exclusive clustering. In the following we will relax Equation 13 and 14 constraints; this will lead to the formation of partially overlapped groups from which we will derive **PCG** in a novel way. Some steps in this direction (non exclusive grouping) were pioneered by Greg Kochaniak (unfortunately a formal description of his approach is not available), but his temporal grouping implementation was quite different from the *fuzzy grouping* approach here introduced.

To implement Fuzzy grouping, the last two steps of the Finalize Intermediate Node Training algorithm in Section 4.3 must be replaced with the following five sequential stages (previous steps remain unaltered):

1. **Compute initial groups G** with a clustering algorithm enforcing Equation 13 and 14. This initial clustering solution can be computed with the default algorithm described in Section 4.3.1 or with the MaxStab algorithm described in Section 5.1.1.
2. **Remove small groups and groups with low stability**. Groups with less than $groupMinSize$ coincidences are expected to bring limited generalization, so they are removed. Analogously, groups $\mathbf{g}_j$ such that $stability(\mathbf{g}_j) \leq groupMinStability$ are removed since their elements are not enough temporally close each other. At this stage coincidences of deleted groups remain orphans (Equation 14 is no longer enforced).
3. **Computation of PCG**. Each element $\mathbf{PCG}[i,j] = P(\mathbf{c}_i|\mathbf{g}_j)$ is calculated according to Equation 19.
4. **Group extension**. Given a group $\mathbf{g}_j$, coincidences already belonging to $\mathbf{g}_j$ at the end stage 2, are denoted as *primary* coincidences, whereas coincidences added subsequently (i.e., during this phase) are denoted as *secondary*. Since primary coincidences contributed to the group formation they are expected to be the most representative for the group. However, other coincidences not belonging to the group could be temporally close to coincidences in the group: we allow a coincidence $\mathbf{c}_i$ to be added to a group $\mathbf{g}_j$ as secondary coincidence if $\mathbf{PCG}[i,j]$ is high. However, instead of explicitly thresholding $\mathbf{PCG}[i,j]$, group extension is accomplished as:

Group Extension to Secondary Coincidences

for each $\mathbf{c}_i, i = 1 \ldots n_c$
{ $\mathbf{S} = \emptyset$
  $totPCG = 0$
  while $(totPCG < targetPCG)$ // we want to add $\mathbf{c}_i$ only to the top% groups (default value for $targetPCG = 75\%$)
  { $k^* = \arg\max_{k=1\ldots n_g,\ \mathbf{g}_k \notin \mathbf{S}} \mathbf{PCG}[i,k]$ // select the temporally closest group, excluding those already considered
    $\mathbf{S} = \mathbf{S} \cup \{\mathbf{g}_{k^*}\}$ // to avoid selecting $\mathbf{g}_{k^*}$ again
    $totPCG = totPCG + \mathbf{PCG}[i,k^*]$
    if $(\mathbf{c}_i \notin \mathbf{g}_{k^*})$ // $\mathbf{c}_i$ might already belong to $\mathbf{g}_{k^*}$ as primary coincidence

```
        g_{k*} = g_{k*} ∪ {c_i}          // c_i is added to g_{k*} as secondary coincidence
    }
}
```

At the end of this stage Equation 13 is no longer enforced.

5. **Cleaning and Normalization of PCG**

- $\mathbf{PCG}^{(new)}[i,j] = \begin{cases} \mathbf{PCG}[i,j] & \text{if } \mathbf{c}_i \in \mathbf{g}_j \\ 0 & \text{otherwise} \end{cases}$, for each $i = 1 \dots n_c, j = 1 \dots n_g$.

  This step clears a $\mathbf{PCG}[i,j]$ value, computed at stage 3, if $\mathbf{c}_i$ does not belong (neither as primary nor as secondary) to group $\mathbf{g}_j$.

- $\mathbf{PCG}^{(new)}[i,j] = \mathbf{PCG}[i,j] / \left(\sum_{k=1\dots n_c} \mathbf{PCG}[k,j]\right)$, for each $i = 1 \dots n_c, j = 1 \dots n_g$

  This step is necessary, after **PCG** cleaning, to ensure that for each group $\mathbf{g}_j$, $\sum_{i=1\dots n_c} \mathbf{PCG}[i,j] = 1$.

It is worth noting that fuzzy grouping could be implemented without group extension (stage 4) and cleaning/normalization (stage 5), since, at the end of stage 3, **PCG** matrix is already consistent. However, the proposed implementation leads to a sparse **PCG** (e.g., only a minor portion of its element are not 0) which is preferable for both robustness (as confirmed by experimental results) and computational efficiency. In the rest of this paper we will denote the temporal grouping introduced in Section 4.3.1 as *exclusive grouping* in order to distinguish it from the fuzzy grouping here proposed. Fig. 10 shows an example of fuzzy grouping and compares it with the exclusive grouping solution from which it was derived.

5.1.3 Activation Buffering

In Section 4.2 we explained that HTM training is performed level by level: while training level $\mathcal{L}_i$, all the nodes of previous levels work in inference mode. Therefore all the patterns in the training sequence $\mathcal{Q}_i$ used to train level $\mathcal{L}_i$ must be processed (i.e., inference) by all the levels $\mathcal{L}_0 \dots \mathcal{L}_{i-1}$. For huge training sequences this (lower level) processing can be computationally demanding thus leading to long training time. However, since the training sequences $\mathcal{Q}_i$ used to train the different levels are usually generated from the same training patterns, buffering the node responses (i.e., the group activations) allows re-processing of the same patterns to be avoided. This idea is derived by an HTM implementation developed by Greg Kochaniak.

Activation buffering implementation details depend on the training strategy and in particular on the composition of the training sequences. In the following we assume that training sequences are created as described in Section 4.1 where patterns in each training sequence $\mathcal{Q}_i$ are obtained by one or more exhaustive scans over the training set patterns $\mathcal{S}_{train}$ (see Fig. 7 and Fig. 8). Let $\mathbf{I}_{idx} \in \mathcal{S}_{train}$ be a training set pattern, then $\mathbf{q}_{idx,offs} \in \mathcal{Q}_i$ is the pattern extracted by $\mathbf{I}_{idx}$ when the scan offset is $offs$. Activation buffering can be enabled when training levels $\mathcal{L}_i$ with $1 \leq i \leq n_{levs} - 2$. The implementation is slightly different if the level is working in node sharing mode or not:

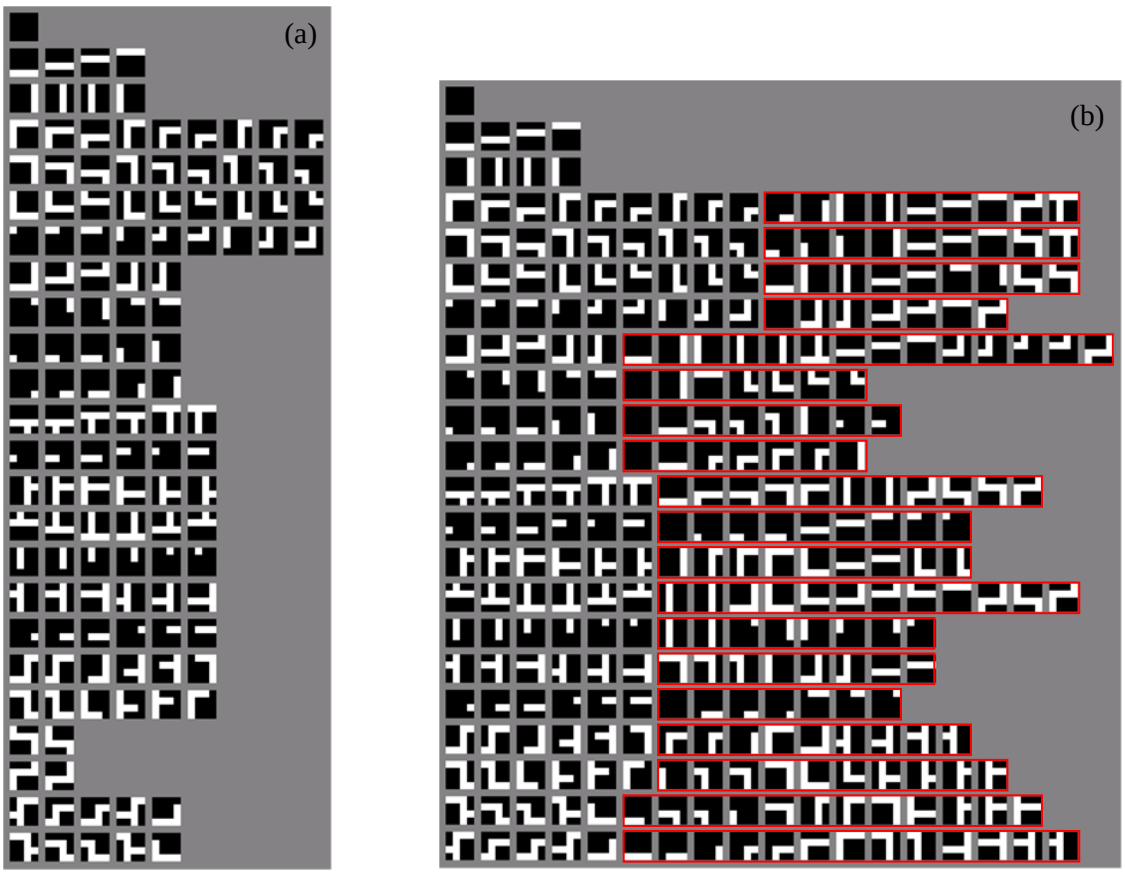

Fig. 10. (a) The exclusive grouping solution reported in Fig. 9.b is characterized by an average group stability $J_2 = 42\%$. (b) Fuzzy grouping solution obtained starting from groups in (a) and with parameter values: $groupMinSize = 4$, $groupMinStability = 20\%$, $targetPCG = 75\%$. Two groups have been deleted because of $groupMinSize < 4$. Coincidences enclosed inside red frames are secondary coincidences added during group extension. It can be noted that many of the secondary coincidences can be obtained by small translations of primary coincidences in the same group. Here the average group stability grows to $J_2 = 45\%$, and, even if the average group length increases from 5.5 to 14, the percentage of non-zero **PCG** elements remains quite small (10.5%).

- *no node sharing*. For each node $\mathcal{N}_j^{(i)}$ of level $\mathcal{L}_i$ and for each pattern $\mathbf{q}_{idx,offs}$ the index $widx$ of the winning group (i.e., the most active group) is stored in a buffer: $\mathbf{Buffer}[idx, offs, j] = widx$. When training nodes of level $\mathcal{L}_{i+1}$, indices of child winning groups (**widx**, see Equation 6) are composed by directly accessing the buffer without any lower level pattern re-processing.
- *node sharing*. Only the master node $\mathcal{N}_1^{(i)}$ is trained. In this case, the node reference j is always 1, so the buffer entries are $\mathbf{Buffer}[idx, offs] = widx$. When training nodes of level $\mathcal{L}_{i+1}$ the child winning group index of each non master node $\mathcal{N}_j^{(i)}$ can obtained by accessing the buffer at a position $offs$ that depends, not only on the current scan offset, but also on the relative position of $\mathcal{N}_j^{(i)}$ with respect to the master node $\mathcal{N}_1^{(i)}$ (in practice, the activation of a non master node is derived from the master node activation upon receptive field shifting).

Two types of activation buffering, denoted as *normalBuffering* and *fastBuffering*, can be implemented:

- *normalBuffering* results are identical to the non-buffering case. Buffering is performed at the end of node training when the entire training sequence is presented again to the network and inference is carried out through previous

levels $\mathcal{L}_0 \ldots \mathcal{L}_{i-1}$. However, NormalBuffering is effective only for levels operating in node sharing mode; making it advantageous also for "no node sharing" levels would be very complex and space-demanding.

- *fastBuffering* results are usually slightly different with respect to the non-buffering case. Buffering is carried out in two stages: (i) during the training of level $\mathcal{L}_i$, the winning coincidence indices (i.e., the most active coincidence indices) are buffered. At the end of $\mathcal{L}_i$ training, once **PCG** has been computed, winning coincidence indices are batch converted into winning group indices. This is an heuristic step (leading to a loss of information) because it ignores the contributions of the non-winning coincidences to the computation of group activation levels (see Equation 3). However, from experimental results (refer to Section 6) we noted that fastBuffering is not only more efficient, but sometimes is also more accurate than normalBuffering, and in general, even when it is less accurate, the accuracy drop is small.

6. PATTERN CLASSIFICATION EXPERIMENTS

In this Section we present several experimental results on pattern classification problems: Subsection 6.1 introduces the three datasets used in the experiments; in Subsection 6.2 we discuss HTM training, tuning and parameterization and we compare the new training algorithms of Section 5 with the default implementation reported in Section 4; finally, in Subsection 6.3 HTM is compared with other pattern recognition approaches.

6.1 DATASETS

For this study we selected three different pattern classification problems: SDIGIT, PICTURE and USPS. In our opinion, these three datasets constitute a good benchmark to study invariance, generalization and robustness of a pattern classifiers. However, in all the three cases the patterns are small black-and-white or grayscale images (32×32 or smaller). Even if HTM was already applied with success to object recognition problems with larger color images (see [3][11]) our current implementation need to be further enhanced to be able to efficiently works with large patterns. As discussed in Section 7, part of our future efforts will be dedicated to the demonstration of HTM capabilities on typical object recognition benchmarks such as CalTech and Pascal VOC datasets [35].

6.1.1 SDIGIT

SDIGIT is a machine-printed digit classification problem where just a single (16×16 pixels, 8-bit grayscale) 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, generalization/invariance, robustness. With $\mathcal{S}_{sdigit\ Train}\langle n, s_{xmin}, s_{xmax}, s_{ymin}, s_{ymax}, r_{max}\rangle$ we denote a set of n patterns, including, for each of the 10 digits, the primary pattern and further $(n/10) - 1$ patterns generated by simultaneous scaling and rotation of the primary pattern according to random triplets $\langle s_x, s_y, r\rangle$ where $s_x \in [s_{xmin}, s_{xmax}]$, $s_y \in [s_{ymin}, s_{ymax}]$ and $r \in [-r_{max}, r_{xmax}]$.

The creation of a test set $\mathcal{S}_{sdigit\ Test}\langle n, s_{xmin}, s_{xmax}, s_{ymin}, s_{ymax}, r_{max}\rangle$ starts by translating each of the 10 primary pattern at all positions that allow it to be fully contained (with a 2 pixel background offset) in the 16×16 window thus obtaining m patterns; then, for each of the m patterns, $(n/10) - 1$ further patterns are generated by transforming the pattern according to random triplets $\langle s_x, s_y, r\rangle$; the total number of patterns in the test set is then $m \times n/10$. Fig. 11 shows an example of test set generation.

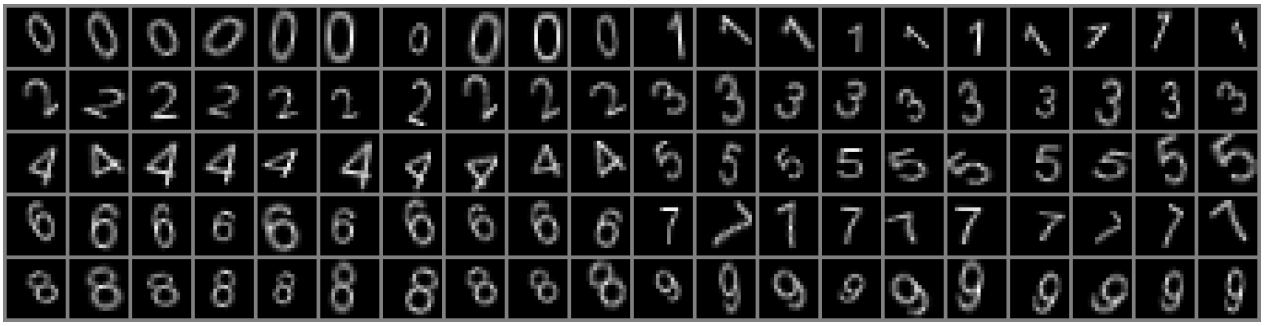

Fig. 11. SDIGIT: 10 patterns for each class extracted from a test set $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45°\rangle$. Note the large intra-class variation because of relevant rotation and scale changes; also note that some patterns of different classes appear to be very similar (e.g., rotated "1" and "7", small "5" and "6").

6.1.2 PICTURES

This is a difficult line-drawing classification problem introduced in [1]. The dataset can be obtained from [5]. Patterns are 32×32 pixels, 1-bit (i.e., black and white) images belonging to 48 classes, including: characters, stereotyped animals and simple objects. The training set $\mathcal{S}_{picture\ Train}$ is constituted by 453 images; pattern distribution over classes in unbalanced but all classes have more than one pattern. The test set $\mathcal{S}_{picture\ Test}$ is composed by 8,941 patterns which represent distorted versions of the training set ones. Distortion includes geometric change, line thickness change, noise (i.e., randomly flipped pixels), disconnection/cancellation of parts; some of the patterns are so severely distorted that also human classification is challenging. Fig. 12 shows some examples. A reduced version of PICTURE problem, denoted as PICTURE-α, can be obtained by considering only the first 8 classes: in particular, $\mathcal{S}_{picture-\alpha\ Train}$ contains 100 patterns and $\mathcal{S}_{picture-\alpha\ Test}$ contains 2,000 patterns.

6.1.3 USPS

USPS is a well known handwritten digit classification problem [36], largely used in the scientific literature as a benchmark for pattern recognition and machine learning approaches. USPS patterns are 16×16 pixels, 8-bit grayscale images; the training set $\mathcal{S}_{usps\ Train}$ and test set $\mathcal{S}_{usps\ Test}$ contains 7,291 and 2,007 patterns respectively. Fig. 13 shows some examples. With $\mathcal{S}_{usps\ Train}\langle n\rangle$ we denote a subset of the training set composed by the first $n/10$ patterns of each class. Although the shape variability in the USPS patterns is quite large, the digits are centered in their window and the test set variations are quite well covered by the large training set, and therefore even a simple approach such as the Nearest-Neighbor classifier achieves good classification results. While we believe this dataset is not ideal for studying invariance and generalization features of a pattern classifier, reporting and comparing HTM accuracy also on well-know benchmarks is essential.

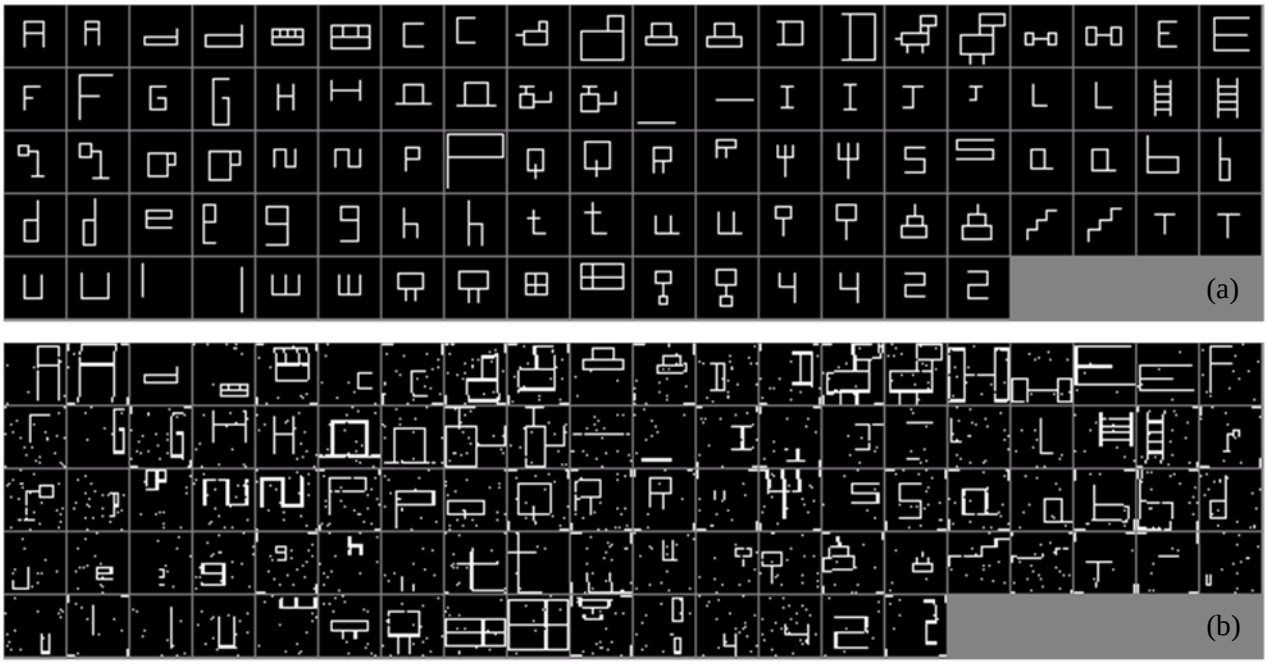

Fig. 12. Two examples per class extracted from (a) $\mathcal{S}_{picture\ Train}$ and (b) $\mathcal{S}_{picture\ Test}$.

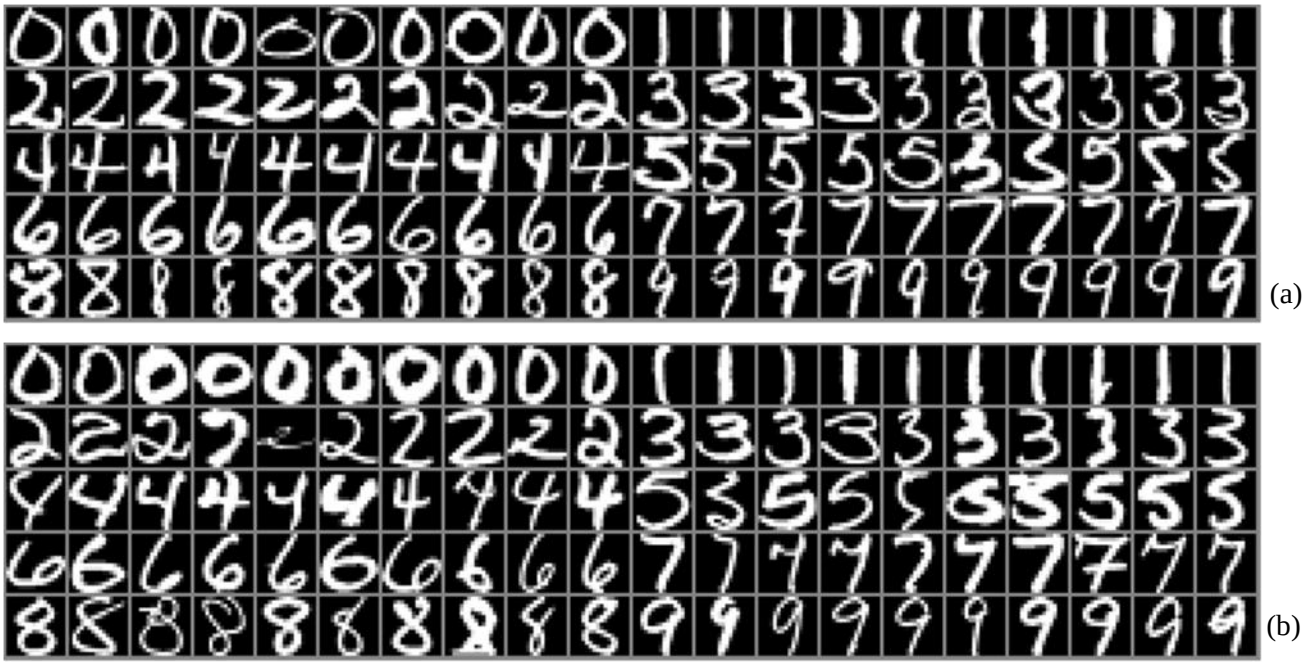

Fig. 13. Ten examples per class extracted from (a) $\mathcal{S}_{usps\ Train}$ and (b) $\mathcal{S}_{usps\ Test}$.

6.2 HTM ANALYSIS

Designing an HTM architecture and finding optimal values for the numerous parameters controlling the network learning and inference is not a trivial task. Furthermore, as for 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 respect to its parameterization and

performance just nicely degrades as parameters drift away from their optimal values. In our experimentation we tried to fix, as much as possible, the network architecture and the parameter values independently of the problem. This could lead to suboptimal accuracy, but in general allows to control data overfitting, especially when a validation set (disjoint from the test set) is not available to tune parameters.

6.2.1 Parameter selection

Table I list the parameter values that we found to be appropriate for all the classification problems addressed in this work, while Table II includes dataset specific parameter values.

All the HTM used are four-level networks: two intermediate levels perform both spatial and temporal analysis; the output level performs a further spatial analysis and then classifies the pattern. Some experiments have been carried out with three- and five-level networks too; although with proper parameterization these architectures can perform well (sometimes better than a four-level HTM), a four-level architecture demonstrated to be an optimal choice for input patterns of size 16×16 and 32×32. As reported in Table II, node arrangements across the four levels is 16×16 → 4×4 → 2×2 → 1 (as in Fig. 1 example) for SDIGIT and USPS, and 32×32 → 8×8 → 4×4 → 1 for PICTURE.

Level 1 always operates in *node sharing* mode since its nodes are expected to learn basic features that are somewhat independent of the position within the network receptive field. Level 2 also operates in *node sharing* mode for PICTURE while level 2 node sharing is not activated for SDIGIT and USPS since in these cases pattern translations across the input window is more limited and $\mathcal{L}_2$ nodes experience sub-patterns that are position dependent.

Common parameter values			
Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
	$node\ sharing = \text{true}$ $forgetThr = 0$ $transitionMemory = 2$ $groupMinSize = 4$ $groupMaxSize = 10$ $targetPCG = 75\%$ **Default Temporal Clustering** $topNeighbors = 3$ **MaxStab Temporal Clustering** $minΔstab = 0.33$	$thrDist = 0$ $forgetThr = 0$ $transitionMemory = 5$ $groupMinSize = 2$ $groupMaxSize = 12$ $targetPCG = 75\%$ **Default Temporal Clustering** $topNeighbors = 2$ **MaxStab Temporal Clustering** $minΔstab = 0.33$ $groupMinStability = 0.12$	$thrDist = 0$ $forgetThr = 0$

Table I. Common parameter values for all the HTM networks used in our experiments. Optimal parameters for both Default Temporal Clustering and MaxStab Temporal Clustering are reported. Parameter $targetPCG$ is used only if Fuzzy Grouping is activated.

SDIGIT			
Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
16×16	4×4 $thrDist = 18.0$ $\sigma = 15.0$ **MaxStab Temporal Clustering** $groupMinStability = 0.12$	2×2 $node\ sharing = \text{false}$	1×1

PICTURE			
Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
32×32	8×8 $thrDist = 0.0$ $\sigma = 25.0$ **MaxStab Temporal Clustering** $groupMinStability = 0.12$	4×4 $node\ sharing =$ true	1×1
USPS			
Level 0 (Input)	Level 1 (Intermediate)	Level 2 (Intermediate)	Level 3 (Output)
16×16	4×4 $thrDist = 55.0$ $\sigma = 25.0$ **MaxStab Temporal Clustering** $groupMinStability = 0.07$	2×2 $node\ sharing =$ false	1×1

Table II. Problem specific values for the HTM networks used in our experiments.

Parameters $thrDist$ and σ deserve particular attention:

- Level 2 and 3 always have $thrDist = 0$, this means that all the sub-patterns encountered during learning are stored as coincidences if they are not identical (i.e., $widxDistance > 0$) to already seen coincidences. Remember that the meaning of $thrDist$ for the intermediate level $\mathcal{L}_1$ is different with respect to subsequent levels; in fact, in $\mathcal{L}_1$ nodes the coincidence distances are computed as Gaussian distances while for higher level nodes they are computed in terms of winning indices differences (see Section 4.3). As to Level 1, the value of $thrDist$ directly controls the number of "basic" coincidences created and heavily influence the whole network complexity (in terms of number of coincidences and groups at all levels); for PICTURE we set $thrDist = 0$ because PICTURE patterns are black and white and the number of different sub-patterns presented to level 1 nodes is quite small (less than 200); on the other hand, SDIGIT and USPS patterns are gray-level and setting $thrDist = 0$ would result in a huge number of nearly-identical coincidences at level 1. Experimental results proved that the network invariance and generalization capabilities get worse if too many nearly-identical coincidences are retained at level 1, so $thrDist$ has to be adjusted (by trial and errors) according to the pattern variability in the problem at hand.
- Parameter σ (see Equation 1) is mainly involved in the inference stage where it controls how quickly the coincidence activations (in level 1 nodes) decay when the current sub-pattern deviates from the stored coincidences. In practice, a too high value of σ determines the activation of a large number coincidences thus leading to little spatial selectivity, while a too low value of σ determines the activation of just one coincidence and this penalizes generalization and robustness. We experimentally discovered that an effective way to estimate an optimal value for σ is to require that a given percentage (around 3%) of $\mathcal{L}_1$ coincidences are responsible for the 95% of the whole activation. In other words, if $\mathbf{y}[i]$ values are sorted in descending order, σ should be tuned in such a way that, on the average, the first 3% $\mathbf{y}[i]$ totals the 95% of the $\mathbf{y}[i]$ sum.

There is not much more to explain about parameters reported in Table I, except noting that the value of $transitionMemory$ has to be markedly increased moving from $\mathcal{L}_1$ to $\mathcal{L}_2$; this is consistent with the provision that HTM higher level nodes must be more invariant than lower level nodes with respect to spatial and temporal changes

of input patterns and therefore, during training, the temporal analysis must be extended to longer time periods. Finally, it is worth noting that the value of parameter $groupMinStability$, which is stable to 0.12 for almost all the cases, has been decreased to 0.07 only for USPS (level 1); this was necessary because of the high number of groups created at level 1 in USPS (due to the large pattern variability) and the consequent difficulty of achieving high group stability after the temporal clustering; using $groupMinStability = 0.12$ for USPS would lead to discard too many groups. An alternative approach, to make the choice of minimum group stability totally problem independent, could be to define this threshold as a percentage of the average group stability, thus avoiding to provide and absolute value.

6.2.2 Accuracy and efficiency

Table III, IV and V report results achieved on SDIGIT, PICTURE and USPS respectively. HTM parameters have been set as described in Section 6.2.1. Each table compares three configurations: (i) *Baseline* refers to an HTM trained with default algorithms described in Section 4; (ii) *Fuzzy grouping* refers to a network where coincidence-group memberships are computed according to the approach described in Section 5.1.2; (iii) *MaxStab* is the case where the Default temporal clustering is replaced with the MaxStab algorithm introduced in Section 5.1.1 (fuzzy grouping is also active in this configuration).

For all the experiments we report:

- details about the sequences Q_i used to train the corresponding HTM levels (see Section 4.1); in particular, for each sequence we provide the number of sub-patterns and the sub-pattern size;
- classification accuracy for both training and test set;
- time elapsed for training/test. The time measure refers to our C# (.net) implementation running on Windows 7 on a Xeon CPU W3550 at 3.07 GHz. Although our HTM implementation can take advantage of a multi-core CPU, only one core is here used for a fair comparison with other classifiers in Section 6.3.
- the size of the HTM in MB, that we define as the total amount of data that must be stored at the end of training to be able to run inference. For floating points data we used double precision encoding (8 bytes).
- for the intermediate levels $\mathcal{L}_1$ and $\mathcal{L}_2$ we show: the number of coincidences (n_c); the number of groups (n_g); the average length of each group (in term of coincidences) - this information is reported within brackets just after n_g; the average group stability J_2 (see Equation 22) resulting from temporal clustering. Note that if the level is operating in shared mode the above statistics refer to the master node; otherwise these values have to be intended as average values over all the level nodes (we used the notation $\overline{n_c}$, , $\overline{n_g}$ and $\overline{J_2}$ to distinguish such a case). For the output level $\mathcal{L}_3$ we report the number n_c of coincidences.

From this round of experiments we can conclude that:

- HTM performs well on the three datasets (how well will be more evident from the comparison with other approaches in Section 6.3).
- HTM training, mainly based on unsupervised learning, is computationally efficient.

- 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 can degrade (see test time).
- Fuzzy grouping improves (often markedly) HTM accuracy with respect to baseline configuration. A minor drawback is a certain increase in the average group length leading to larger size and lower efficiency.
- MaxStab temporal clustering generally improves accuracy, even if in this case the advantage with respect to Default temporal clustering is marginal. It is worth noting that MaxStab often leads to the formation of a lower number of (more stable) groups at $\mathcal{L}_2$ and, as a consequence, to a lower number of coincidences at $\mathcal{L}_3$; hence the total size is typically smaller and the network more efficient.
- Our HTM baseline implementation and Numenta's one achieved very similar performance in term of accuracy on PICTURE (see Table VII); however, our HTM training implementation seems to be computationally advantageous.

SDIGIT - test set: $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45° \rangle$ (6,200 patterns, 10 classes)							
Training set	HTM configuration	Accuracy (%)		Time (hh:mm:ss)		Size (MB)	Details
		train	test	train	Test		
$\mathcal{S}_{sdigit}$ <50,0.70,1.0,0.7,1.0,40°> 50 Patterns Training sequences: $\mathcal{Q}_1$: 32904 (4 × 4) $\mathcal{Q}_2$: 3576 (16 × 16) $\mathcal{Q}_3$: 1788 (16 × 16)	Baseline	99.94	66.35	00:00:05	00:00:10	0.49	$\mathcal{L}_1$: $n_c = 2680$, $n_g = 269$ (10.0), $J_2 = 36.4\%$ $\mathcal{L}_2$: $\overline{n_c} = 875.0$, $\overline{n_g} = 169.3$ (5.2), $\overline{J_2} = 33.8\%$ $\mathcal{L}_3$: $n_c = 1381$
	Fuzzy grouping	99.89	71.15	00:00:07	00:00:13	0.62	$\mathcal{L}_1$: $n_c = 2680$, $n_g = 266$ (28.0), $J_2 = 36.4\%$ $\mathcal{L}_2$: $\overline{n_c} = 895.3$, $\overline{n_g} = 171.5$ (14.6), $\overline{J_2} = 35.1\%$ $\mathcal{L}_3$: $n_c = 1295$
	MaxStab	100	71.37	00:00:08	00:00:13	0.58	$\mathcal{L}_1$: $n_c = 2680$, $n_g = 300$ (26.4), $J_2 = 36.6\%$ $\mathcal{L}_2$: $\overline{n_c} = 869.0$, $\overline{n_g} = 93.8$ (18.3), $\overline{J_2} = 51.1\%$ $\mathcal{L}_3$: $n_c = 1037$
$\mathcal{S}_{sdigit}$ <100,0.70,1.0,0.7,1.0,40°> 100 Patterns Training sequences: $\mathcal{Q}_1$: 64566 (4 × 4) $\mathcal{Q}_2$: 6846 (16 × 16) $\mathcal{Q}_3$: 3423 (16 × 16)	Baseline	99.97	82.34	00:00:17	00:00:16	0.83	$\mathcal{L}_1$: $n_c = 4358$, $n_g = 440$ (9.9), $J_2 = 32.7\%$ $\mathcal{L}_2$: $\overline{n_c} = 1600.3$, $\overline{n_g} = 280.5$ (5.7), $\overline{J_2} = 33.7\%$ $\mathcal{L}_3$: $n_c = 2811$
	Fuzzy grouping	100	87.45	00:00:24	00:00:24	1.08	$\mathcal{L}_1$: $n_c = 4358$, $n_g = 439$ (31.9), $J_2 = 31.9\%$ $\mathcal{L}_2$: $\overline{n_c} = 1644.3$, $\overline{n_g} = 294.8$ (16.1), $\overline{J_2} = 33.7\%$ $\mathcal{L}_3$: $n_c = 2709$
	MaxStab	100	87.56	00:00:25	00:00:23	1.00	$\mathcal{L}_1$: $n_c = 4358$, $n_g = 473$ (29.1), $J_2 = 34.9\%$ $\mathcal{L}_2$: $\overline{n_c} = 1629.5$, $\overline{n_g} = 178.0$ (19.6), $\overline{J_2} = 47.9\%$ $\mathcal{L}_3$: $n_c = 2299$
$\mathcal{S}_{sdigit}$ <250,0.70,1.0,0.7,1.0,40°> 250 Patterns Training sequences: $\mathcal{Q}_1$: 162082 (4 × 4) $\mathcal{Q}_2$: 17410 (16 × 16) $\mathcal{Q}_3$: 8705 (16 × 16)	Baseline	99.99	91.55	00:01:15	00:00:32	1.60	$\mathcal{L}_1$: $n_c = 7562$, $n_g = 755$ (10.0), $J_2 = 29.8\%$ $\mathcal{L}_2$: $\overline{n_c} = 3600.3$, $\overline{n_g} = 606.3$ (5.9), $\overline{J_2} = 31.4\%$ $\mathcal{L}_3$: $n_c = 7377$
	Fuzzy grouping	100	94.35	00:02:01	00:00:58	2.18	$\mathcal{L}_1$: $n_c = 7562$, $n_g = 758$ (37.0), $J_2 = 30.0\%$ $\mathcal{L}_2$: $\overline{n_c} = 3760.0$, $\overline{n_g} = 619.8$ (18.2), $\overline{J_2} = 32.1\%$ $\mathcal{L}_3$: $n_c = 7804$
	MaxStab	100	94.61	00:02:04	00:00:55	2.06	$\mathcal{L}_1$: $n_c = 7562$, $n_g = 791$ (35.2), $J_2 = 31.6\%$ $\mathcal{L}_2$: $\overline{n_c} = 3782.3$, $\overline{n_g} = 417.3$ (21.6), $\overline{J_2} = 42.9\%$ $\mathcal{L}_3$: $n_c = 6488$

Table III. HTM results on SDIGIT. Experiments are performed with three training sets of increasing size: n = 50, n = 100 and n = 250. A single test set with 6200 patterns (n = 1000, m = 62, see Section 6.1.1) was used in all the experiments. Note that the geometric variations in the training sets are slightly smaller than corresponding test set variations; this led to a minor performance improvements in all the tests we carried out (not only for HTM, but also for other classifiers introduced in Section 6.3).

PICTURE - test set: $\mathcal{S}_{picture-\alpha\ Test}$ (2,000 patterns, 8 classes)							
Training set	HTM configuration	Accuracy (%)		Time (hh:mm:ss)		Size (MB)	Details
		train	test	train	test		
$\mathcal{S}_{picture-\alpha\ Train}$ 100 Patterns Training sequences $\mathcal{Q}_1$: 351592 (4 × 4) $\mathcal{Q}_2$: 288000 (8 × 8) $\mathcal{Q}_3$: 20424 (32 × 32)	Baseline	100	78.30	00:00:13	00:00:03	1.21	$\mathcal{L}_1$: $n_c = 133$, $n_g = 14$ (9.5), $J_2 = 47.1\%$ $\mathcal{L}_2$: $n_c = 903$, $n_g = 216$ (4.2), $J_2 = 19.9\%$ $\mathcal{L}_3$: $n_c = 16140$
	Fuzzy grouping	99.85	83.50	00:00:16	00:00:05	1.21	$\mathcal{L}_1$: $n_c = 133$, $n_g = 14$ (19.9), $J_2 = 47.1\%$ $\mathcal{L}_2$: $n_c = 961$, $n_g = 228$ (22.3), $J_2 = 20.7\%$ $\mathcal{L}_3$: $n_c = 15469$
	MaxStab	99.89	84.20	00:00:16	00:00:04	1.10	$\mathcal{L}_1$: $n_c = 133$, $n_g = 22$ (14.0), $J_2 = 44.7\%$ $\mathcal{L}_2$: $n_c = 1139$, $n_g = 101$ (45.0), $J_2 = 28.5\%$ $\mathcal{L}_3$: $n_c = 13950$
PICTURE - test set: $\mathcal{S}_{picture\ Test}$ (8,941 patterns, 48 classes)							
Training set	HTM configuration	Accuracy (%)		Time (hh:mm:ss)		Size (MB)	Details
		train	Test	train	test		
$\mathcal{S}_{picture\ Train}$ 453 Patterns Training sequences: $\mathcal{Q}_1$: 1731068 (4 × 4) $\mathcal{Q}_2$: 1429996 (8 × 8) $\mathcal{Q}_3$: 116198 (32 × 32)	Baseline	99.87	66.78	00:01:51	00:01:17	6.05	$\mathcal{L}_1$: $n_c = 173$, $n_g = 20$ (8.7), $J_2 = 43.1\%$ $\mathcal{L}_2$: $n_c = 1777$, $n_g = 441$ (4.0), $J_2 = 18.6\%$ $\mathcal{L}_3$: $n_c = 82582$
	Fuzzy grouping	99.69	71.11	00:02:16	00:01:37	6.22	$\mathcal{L}_1$: $n_c = 173$, $n_g = 20$ (18.8), $J_2 = 43.1\%$ $\mathcal{L}_2$: $n_c = 2127$, $n_g = 448$ (26.8), $J_2 = 17.9\%$ $\mathcal{L}_3$: $n_c = 83128$
	MaxStab	98.92	71.08	00:01:54	00:01:14	4.81	$\mathcal{L}_1$: $n_c = 173$, $n_g = 26$ (15.5), $J_2 = 43.8\%$ $\mathcal{L}_2$: $n_c = 2093$, $n_g = 185$ (43.4), $J_2 = 27.7\%$ $\mathcal{L}_3$: $n_c = 64231$
	Nupic 1.7	100	66.3	00:09:54	00:00:45	-	$\mathcal{L}_1$: $n_c = 134$, $n_g = 30$ $\mathcal{L}_2$: $n_c = 951$, $n_g = 300$ $\mathcal{L}_3$: $n_c = 80817$

Table IV. HTM results on PICTURE. (Top) results achieved on the reduced version PICTURE-α; (bottom) results obtained on the full PICTURE dataset. In the last row, Nupic 1.7 refers to Numenta's implementation (Vision Toolkit in Nupic 1.7) with factory tuned parameters [5]. While learning algorithms, parameters and training sequences in Nupic 1.7 could partially differ from the Baseline configuration here provided, the accuracy of the two versions on PICTURE full problem is surprisingly similar.

Fig. 14 shows the results of a further experiment aimed at making the relationship between accuracy and complexity (i.e., size) more explicit. To control the HTM complexity we progressively increase level 1 *thrDist*, directly controlling the number of level 1 coincidences and therefore indirectly influencing the whole network size. The graph shows that as we reduce complexity, the network accuracy degrades gracefully. Details are provided for two operating points: the point on the left is the same as in the last row of Table III; for the operating point on the right, in spite of a marginal decrease in performance (94.61% → 91.84%), the network complexity is nearly halved.

6.2.3 Overlapping

Overlapping consists in forcing the receptive field of network nodes at the same level to be partially overlapped. This can be implemented at level 1, but also at higher levels. The procedure of bottom-up message passing used in training and inference is not affected by overlapping: basically, a node belonging to an overlapped region has more than one parent node and simply sends $\boldsymbol{\lambda}^+$message to all its parent nodes. We carried out a number of experiments and we found the following two overlapped architectures to be quite effective for the input size 16×16 (SDIGIT and USPS) and 32×32 (PICTURE):

USPS - test set: $\mathcal{S}_{usps\ Test}$ (2,007 patterns, 10 classes)							
Training set	HTM configuration	Accuracy (%)		Time (hh:mm:ss)		Size (MB)	Details
		train	Test	Train	test		
$\mathcal{S}_{usps\ Train}\langle 100\rangle$ 100 Patterns Training sequences: Q_1: 45396 (4 × 4) Q_2: 1092 (16 × 16) Q_3: 546 (16 × 16)	Baseline	100	83.36	00:00:02	00:00:01	0.18	$\mathcal{L}_1$: $n_c = 957$, $n_g = 93$ (10.3), $J_2 = 22.9\%$ $\mathcal{L}_2$: $\overline{n_c} = 348.0$, $\overline{n_g} = 72.0$ (4.8), $\overline{J_2} = 72.2\%$ $\mathcal{L}_3$: $n_c = 381$
	Fuzzy grouping	100	83.91	00:00:03	00:00:02	0.23	$\mathcal{L}_1$: $n_c = 957$, $n_g = 92$ (57.3), $J_2 = 22.7\%$ $\mathcal{L}_2$: $\overline{n_c} = 353.3$, $\overline{n_g} = 72.3$ (6.4), $\overline{J_2} = 72.8\%$ $\mathcal{L}_3$: $n_c = 377$
	MaxStab	100	84.11	00:00:03	00:00:02	0.23	$\mathcal{L}_1$: $n_c = 957$, $n_g = 97$ (55.2), $J_2 = 22.5\%$ $\mathcal{L}_2$: $\overline{n_c} = 339.0$, $\overline{n_g} = 65.3$ (6.2), $\overline{J_2} = 77.7\%$ $\mathcal{L}_3$: $n_c = 366$
$\mathcal{S}_{usps\ Train}\langle 1000\rangle$ 1000 Patterns Training sequences: Q_1: 448292 (4 × 4) Q_2: 10484 (16 × 16) Q_3: 5242 (16 × 16)	Baseline	100	92.38	00:00:47	00:00:04	0.68	$\mathcal{L}_1$: $n_c = 2250$, $n_g = 216$ (10.4), $J_2 = 13.7\%$ $\mathcal{L}_2$: $\overline{n_c} = 2513.5$, $\overline{n_g} = 525.0$ (4.8), $\overline{J_2} = 64.2\%$ $\mathcal{L}_3$: $n_c = 4327$
	Fuzzy grouping	99.96	92.87	00:01:33	00:00:09	1.00	$\mathcal{L}_1$: $n_c = 2250$, $n_g = 214$ (105.3), $J_2 = 13.7\%$ $\mathcal{L}_2$: $\overline{n_c} = 2802.3$, $\overline{n_g} = 578.8$ (6.7), $\overline{J_2} = 68.3\%$ $\mathcal{L}_3$: $n_c = 4459$
	MaxStab	100	93.42	00:01:39	00:00:09	0.97	$\mathcal{L}_1$: $n_c = 2250$, $n_g = 205$ (107.0), $J_2 = 13.5\%$ $\mathcal{L}_2$: $\overline{n_c} = 2738.0$, $\overline{n_g} = 534.3$ (6.6), $\overline{J_2} = 71.7\%$ $\mathcal{L}_3$: $n_c = 4316$
$\mathcal{S}_{usps\ Train}\langle 7291\rangle$ 7291 Patterns Training sequences: Q_1: 3298360 (4 × 4) Q_2: 78736 (16 × 16) Q_3: 39368 (16 × 16)	Baseline	99.90	93.32	00:13:46	00:00:17	2.77	$\mathcal{L}_1$: $n_c = 4474$, $n_g = 430$ (10.4), $J_2 = 8.9\%$ $\mathcal{L}_2$: $\overline{n_c} = 12582$, $\overline{n_g} = 2657$ (4.7), $\overline{J_2} = 57.0\%$ $\mathcal{L}_3$: $n_c = 31128$
	Fuzzy grouping	99.34	94.67	00:35:23	00:00:43	4.09	$\mathcal{L}_1$: $n_c = 4474$, $n_g = 427$ (159.7), $J_2 = 8.9\%$ $\mathcal{L}_2$: $\overline{n_c} = 14887$, $\overline{n_g} = 3105$ (7.1), $\overline{J_2} = 62.2\%$ $\mathcal{L}_3$: $n_c = 31650$
	MaxStab	99.39	95.57	00:26:45	00:00:30	3.52	$\mathcal{L}_1$: $n_c = 4474$, $n_g = 207$ (174.5), $J_2 = 12.0\%$ $\mathcal{L}_2$: $\overline{n_c} = 13396$, $\overline{n_g} = 2519$ (8.0), $\overline{J_2} = 62.7\%$ $\mathcal{L}_3$: $n_c = 30936$

Table V. HTM results on USPS. Experiments are performed with three training sets of increasing size: *n* = 100, *n* = 1000 and *n* = 7291 (full dataset).

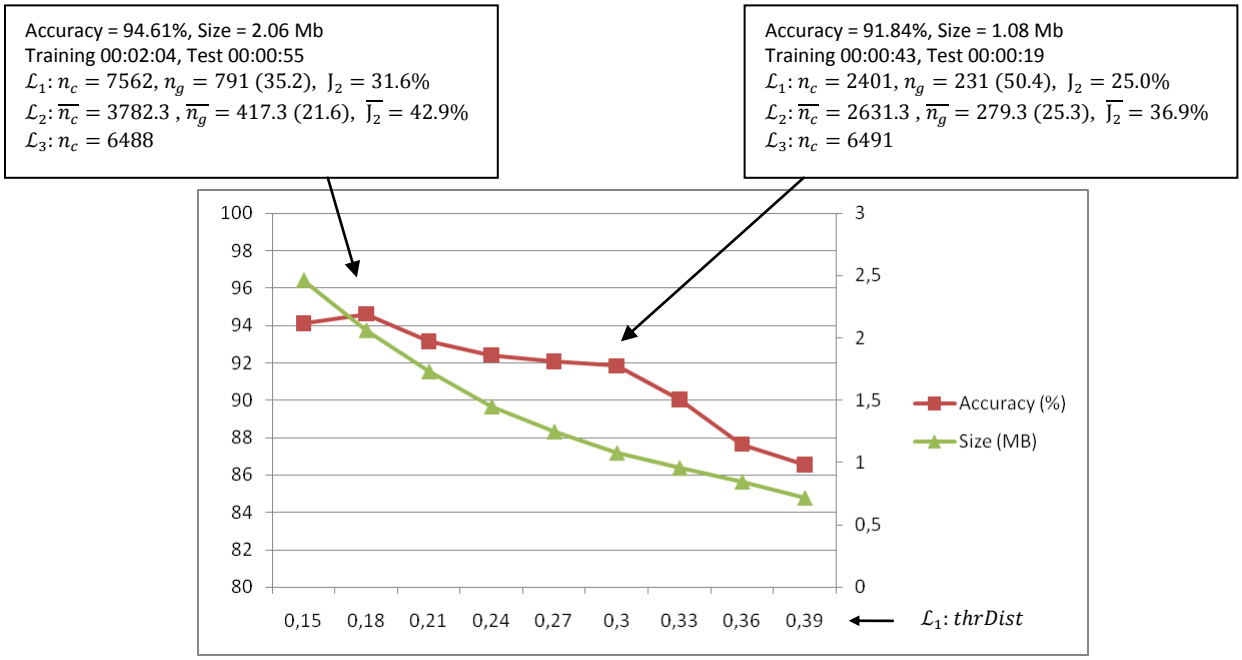

Fig. 14. HTM accuracy and size on SDIGIT (training set $\mathcal{S}_{sdigit} < 250{,}0.70{,}1.0{,}0.7{,}1.0{,}40° >$) as function of $\mathcal{L}_1$ $thrDist$. The HTM configuration used is MaxStab (with Fuzzy grouping).

- HTM $OV_{18\times18\ (2,0)}$: this network has 2 child-node overlapping at $\mathcal{L}_1$ and no child-node overlapping at $\mathcal{L}_2$. To allow an even node partitioning the 16×16 input has been expanded to 18×18 (by simply adding a 1 pixel neutral border to the patters): the resulting number of nodes per level is 18×18, 8×8, 4×4, 1. Overlapping at $\mathcal{L}_1$ implicitly causes $\mathcal{L}_2$ nodes to be partially overlapped (1/3 along each dimensions) in terms of input pixels. Receptive field of $\mathcal{L}_2$ nodes is 2×2 in terms of $\mathcal{L}_1$ nodes and 6×6 in terms of $\mathcal{L}_0$ nodes (or pixels). See Fig. 15 (left).
- HTM $OV_{32\times32\ (2,1)}$: this network has 2 child-node overlapping at $\mathcal{L}_1$ and 1 child-node overlapping[8] at $\mathcal{L}_2$: the resulting number of nodes per level is 32×32, 15×15, 7×7, 1. The contribution of both $\mathcal{L}_1$ and $\mathcal{L}_2$ overlapping determines $\mathcal{L}_2$ nodes to be 1/2 overlapped in terms of input size along each dimensions. Receptive field of $\mathcal{L}_2$ nodes is 3×3 in terms of $\mathcal{L}_1$ nodes and 8×8 in terms of $\mathcal{L}_0$ nodes (or pixels). See Fig. 15 (right).

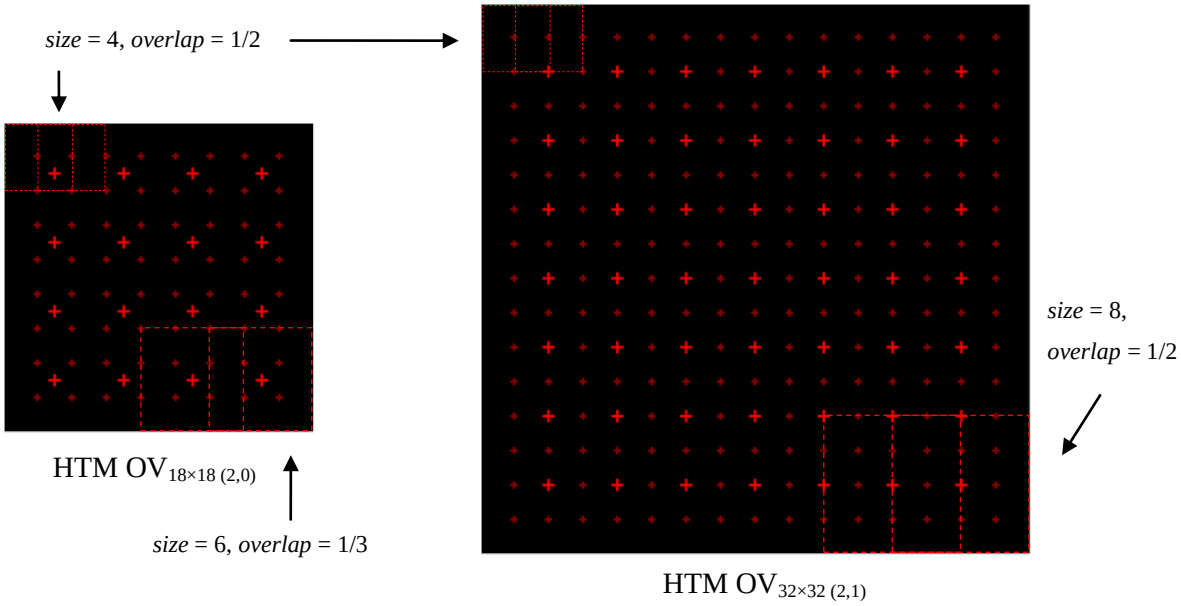

Fig. 15. A graphical visualization of the receptive field center and size of two overlapped architectures. Small and large crosses denote the receptive field centers of level 1 and level 2 nodes respectively. On the top-left part of both the networks the figure shows the receptive field size and overlapping of two level 1 nodes. On the bottom-right part of both the networks the figure shows the receptive field size and overlapping of two level 2 nodes.

Table VI highlights accuracy improvements over Section 6.2.2 results: the advantage in terms of accuracy is very relevant, even if it comes at expense of network complexity and efficiency. Implementing overlapping under the assumption of evenly partitioning child nodes at each level is tricky (especially for small input size), and with respect to the non-overlapping case the resulting architecture typically differs in terms of: number of nodes, receptive field size of the nodes, composition of the training sequences. Hence, the improvements obtained could be due to a mix of factors and further experiments would be necessary to isolate the real advantages of node overlapping.

[8] 1 child-node overlapping at level 2 is necessary for the 32×32 input size due to the constraint of even node partitioning.

SDIGIT - test set: $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45° \rangle$ (6,200 patterns, 10 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		Train	test	Train	test	
$\mathcal{S}_{sdigit} < 50,0.70,1.0,0.7,1.0,40° >$	HTM	100	71.37	00:00:08	00:00:13	0.58
	HTM $OV_{18\times18\ (2,0)}$	100	73.39	00:00:35	00:01:02	1.47
$\mathcal{S}_{sdigit} < 100,0.70,1.0,0.7,1.0,40° >$	HTM	100	87.56	00:00:25	00:00:23	1.00
	HTM $OV_{18\times18\ (2,0)}$	100	91.39	00:02:10	00:01:55	2.74
$\mathcal{S}_{sdigit} < 250,0.70,1.0,0.7,1.0,40° >$	HTM	100	94.61	00:02:04	00:00:55	2.06
	HTM $OV_{18\times18\ (2,0)}$	100	97.06	00:04:04	00:11:33	6.28
PICTURE - test set: $\mathcal{S}_{picture-\alpha\ Test}$ (2,000 patterns, 8 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		train	test	Train	test	
$\mathcal{S}_{picture\ Train}$- 100 Patterns	HTM	99.89	84.20	00:00:16	00:00:04	1.10
	HTM $OV_{32\times32\ (2,1)}$	100	85.50	00:00:34	00:00:42	4.32
PICTURE - test set: $\mathcal{S}_{picture\ Test}$ (8,941 patterns, 48 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		Train	test	Train	test	
$\mathcal{S}_{picture\ Train}$- 453 Patterns	HTM	98.92	71.08	00:01:54	00:01:14	4.81
	HTM $OV_{32\times32\ (2,1)}$	100	73.36	00:06:06	00:10:07	21.89
USPS - test set: $\mathcal{S}_{usps\ Test}$ (2,007 patterns, 10 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		Train	test	Train	test	
$\mathcal{S}_{usps\ Train}\langle 100 \rangle$	HTM	100	84.11	00:00:03	00:00:02	0.23
	HTM $OV_{18\times18\ (2,0)}$	100	89.29	00:00:23	00:00:12	0.91
$\mathcal{S}_{usps\ Train}\langle 1000 \rangle$	HTM	100	93.42	00:01:39	00:00:09	0.97
	HTM $OV_{18\times18\ (2,0)}$	100	95.67	00:18:26	00:01:03	6.79
$\mathcal{S}_{usps\ Train}\langle 7291 \rangle$	HTM	99.39	95.57	00:26:45	00:00:30	3.52
	HTM $OV_{18\times18\ (2,0)}$	99.97	97.06	05:41:40	00:03:23	32.89

Table VI. Accuracy improvements achieved by overlapped configurations; baseline HTM performance refers to MaxStab configuration reported in tables III, IV and V.

6.2.4 Activation buffering

Table VII shows the result of some experiments aimed at determining the speed-up given by activation buffering. As explained in Section 5.1.3, normalBuffering does not alter the training results (the network obtained is identical to the non buffering case), while fastBuffering leads to a somewhat "approximated" solution. However, if we look at the Accuracy column in the table, we note that the accuracy drop due to fastBuffering is marginal in two cases (SDIGIT and USPS), and for PICTURE dataset the fastBuffering solution even prevail over the default one. As to the resulting training speed-up we note that normalBuffering (in two over three cases) lead to a relevant computational save, and fastBuffering saving is always very relevant.

6.2.5 Saccading

Saccading consists in performing multiple inferences on the same pattern while the pattern is moved of a few pixels each time. This emulates fast eye movements used to focus attention on different parts of an object while recognizing it [37]. A simple but effective strategy for HTM is to activate saccading only at test time and presenting each pattern of the test set 5 times to the network: the first time in canonical position and then by moving it one pixel left, up, right and down. Instead of fusing HTM results at decision level (i.e., Majority Vote Rule) or at confidence level (i.e., Sum Rule) we found the following on-line approach to be both simple and effective:

1. when a pattern is presented to the HTM in the first (canonical) position, the Prior class probabilities $P(w_j)$ of the output node are (re)set to the values computed at training time (Equation 16);
2. after inference, the output node Prior class probabilities take the value of the Posterior class probabilities $P(w_j) = P(w_j|e)$, thus biasing successive inferences;
3. step 2 is repeated 4 times during successive saccades.

Table VIII highlights accuracy improvements over Section 6.2.2 results: the advantage in terms of accuracy is quite relevant.

SDIGIT - test set: $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45° \rangle$ (6,200 patterns, 10 classes)				
Training set	HTM configuration	Activation Buffering	Time (hh:mm:ss)	Accuracy (%)
		train	train	test
$\mathcal{S}_{sdigit}$ <250,0.70,1.0,0.7,1.0,40°>	MaxStab	None	00:03:23	94.61
	MaxStab	normalBuffering	00:02:04	94.61
	MaxStab	fastBuffering	00:00:26	92.87
PICTURE - test set: $\mathcal{S}_{picture\ Test}$ (8,941 patterns, 48 classes)				
Training set	HTM configuration	Activation Buffering	Time (hh:mm:ss)	Accuracy (%)
		train	train	test
$\mathcal{S}_{picture\ Train}$ 453 Patterns	MaxStab	None	00:05:37	71.08
	MaxStab	normalBuffering	00:01:54	71.08
	MaxStab	fastBuffering	00:00:52	71.50
USPS - test set: $\mathcal{S}_{usps\ Test}$ (2,007 patterns, 10 classes)				
Training set	HTM configuration	Activation Buffering	Time (hh:mm:ss)	Accuracy (%)
		train	train	test
$\mathcal{S}_{usps\ Train}\langle 7291 \rangle$	MaxStab	None	00:24:06	95.57
	MaxStab	normalBuffering	00:26:45	95.57
	MaxStab	fastBuffering	00:07:00	94.52

Table VII. Comparison of HTM training with: (i) no activation buffering, (ii) normal buffering and (iii) fast buffering.

SDIGIT - test set: $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45° \rangle$		
Training set	Approach	Accuracy (%)
$\mathcal{S}_{sdigit} < 250,0.70,1.0,0.7,1.0,40° >$	HTM	94.61
	HTM (4 saccades)	97.24
	HTM $OV_{18\times18\ (2,0)}$ (4 saccades)	97.97
PICTURE - test set: $\mathcal{S}_{picture\ Test}$		
Training set	Approach	Accuracy (%)
$\mathcal{S}_{picture\ Train}$- 453 Patterns	HTM	71.08
	HTM (4 saccades)	76.09
	HTM $OV_{32\times32\ (2,1)}$ (4 saccades)	74.58
USPS - test set: $\mathcal{S}_{usps\ Test}$ (2,007 patterns, 10 classes)		
Training set	Approach	Accuracy (%)
$\mathcal{S}_{usps\ Train}\langle 7291 \rangle$	HTM	95.57
	HTM (4 saccades)	96.46
	HTM $OV_{18\times18\ (2,0)}$ (4 saccades)	97.21

Table VIII. Accuracy improvement obtained by enabling saccading at test time (i.e., inference over 5 positions for each test pattern) for both MaxStab and overlapped configurations.

It is worth noting that saccading determines a linear increase in the classification time (i.e., 5 times higher). Rising the number of saccades beyond 4 still produces a small accuracy improvement, but the advantage become marginal with respect to the efficiency drop.

6.3 COMPARISONS WITH OTHER SYSTEMS

Table IX, X and XI compares HTM with other approaches in terms of accuracy and efficiency. Of course, a large number of other pattern recognition approaches could be considered for comparison. Here we focused on techniques working at pixel level (as HTM), without any hardwired feature extraction. Our selection also privileged techniques made available as software libraries by their developers:

- NN is a simple Nearest Neighbor classifiers; this classifier gives a good baseline performance and is useful to estimate the problem difficulty.
- MLP is a three layers (input-hidden-output) perceptron [38]; MLP is the best known neural network architecture and therefore it is interesting to understand how it performs in comparison with HTM;
- LeNet5 is a Convolutional Network (CN) designed to classify characters and/or small line-drawing [17]. CN is one of the most interesting MHWA architectures for visual pattern recognition and therefore is a very good reference point for HTM.

Some notes on the experiment setup:

- For LeNet5 and MLP implementation we used the primitives made available by EBLearn [39], which is a very powerful (C++) library to experiment energy-based learning techniques. EBLearn provides an efficient second order backpropagation learning (i.e., exploiting Hessian to speed-up convergence).
- For LeNet5 and MLP instead of stopping the learning after a given number of epochs or by inspecting the error trend on the training set, we used the test set as validation set and stopped learning when the error reached the minimum over the test set. In general this is not a correct strategy, since it can lead to an optimistic estimation of accuracy; however since our aim is HTM comparison with other techniques, in this way we are slightly favoring HTM competitors.
- LeNet5 receptive field is a 32×32 image; in order to use existing (optimal) parameterization we adapted 16×16 SDIGIT and USPS patterns to 32×32 size by adding a neutral (i.e., colored as the background) border.
- The same input adaptation was done for MLP; the resulting number of neurons per level is: $1024{-}n_h{-}n_w$, where: n_w is the number of classes; $n_h = 240$ for SDIGIT and USPS, and $n_h = 480$ for PICTURE (near-optimal values for n_h have been determined by trial and error).
- An important issue is the use of translated patterns for the training: since SDIGIT and PICTURE test sets contains translated versions of the corresponding training set patterns we cannot expect that NN and MLP architectures can deal with such translations if we do not explicitly provide examples during training. Things are different for LeNet5, where thank to the presence of two subsampling layers, moderate object translations are natively tolerated. However to avoid any bias in favor of HTM, for experiment on SDIGIT and PICTURE, we trained NN,

MLP and LeNet5 exactly on the full Q_3 sequences used to train HTM $\mathcal{L}_3$ (the number of translated patterns is reported in the first column of tables IX and X). Since USPS patterns are always centered inside their window, learning with translated patterns was not necessary and in our experiments it resulted in a slight performance drop; for this reason to produce the results reported in table XI translated patterns were not used.

- Timing for MLP and LeNet5 have been measured on the same hardware used for HTM, even if the C++ (Win32 native) EBLearn implementation is expected to be from 2 to 3 times more efficient than C# (.Net managed) HTM implementation; hence the time comparison is biased in favor of MLP and LeNet5.
- In analogy with HTM (see Section 6.2.2), to determine the size in MB we assumed to store floating point data in double precision format; for example LeNet5 (for a 10 class problem) has 51046 weights and coding each weight with 8 bytes yields to a total size of 0.39 MB.
- the label HTM reported in the tables refers to the MaxStab configuration with Fuzzy grouping enabled and with activation buffering set as normalBuffering; results for overlapped HTM are also reported in table XI. In this Section we intentionally ignored HTM improvements given by saccading (see Section 6.2.5) because in principle, also the other approaches could benefit of information fusion.

SDIGIT - test set: $\mathcal{S}_{sdigit\ Test}\langle 1000,0.60,1.10,0.60,1.10,45° \rangle$ (6,200 patterns, 10 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		train	test	train	test	
$\mathcal{S}_{sdigit}$ <50,0.70,1.0,0.7,1.0,40°> 1788 translated patterns	NN	100	57.92	< 1 sec	00:00:04	3.50
	MLP	100	61.15	00:12:42	00:00:03	1.90
	LeNet5	100	67.28	00:07:13	00:00:11	0.39
	HTM	100	71.37	00:00:08	00:00:13	0.58
$\mathcal{S}_{sdigit}$ <100,0.70,1.0,0.7,1.0,40°> 3423 translated patterns	NN	100	73.63	< 1 sec	00:00:07	6.84
	MLP	100	75.37	00:34:22	00:00:03	1.90
	LeNet5	100	79.31	00:10:05	00:00:11	0.39
	HTM	100	87.56	00:00:25	00:00:23	1.00
$\mathcal{S}_{sdigit}$ <250,0.70,1.0,0.7,1.0,40°> 8705 translated patterns	NN	100	86.50	< 1 sec	00:00:20	17.0
	MLP	99.93	86.08	00:37:32	00:00:03	1.90
	LeNet5	100	89.17	00:14:37	00:00:11	0.39
	HTM	100	94.61	00:02:04	00:00:55	2.06

Table IX. HTM compared against other techniques on SDIGIT.

PICTURE - test set: $\mathcal{S}_{picture-\alpha\ Test}$ (2,000 patterns, 8 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		train	test	train	test	
$\mathcal{S}_{picture\ Train}$ 100 Patterns 20424 translated patterns	NN	100	33.70	< 1 sec	00:00:59	159.6
	MLP	71.44	44.40	04:23:43	00:00:02	3.78
	LeNet5	99.98	55.25	00:16:56	00:00:03	0.39
	HTM	99.89	84.20	00:00:16	00:00:04	1.10
PICTURE - test set: $\mathcal{S}_{picture\ Test}$ (8,941 patterns, 48 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		train	test	train	test	
$\mathcal{S}_{picture\ Train}$ 453 Patterns 116198 translated patterns	NN	100	23.31	00:00:04	00:24:08	907.8
	LeNet5	88.78	21.02	01:17:03	00:00:16	0.42
	HTM	98.92	71.08	00:01:54	00:01:14	4.81

Table X. HTM compared against other techniques on PICTURE. MLP result are not reported for the full problem, because of lack of convergence.

USPS - test set: $\mathcal{S}_{usps\ Test}$ (2,007 patterns, 10 classes)						
Training set	Approach	Accuracy (%)		Time (hh:mm:ss)		Size (MB)
		train	test	train	test	
$\mathcal{S}_{usps\ Train}\langle 100\rangle$	NN	100	76.53	< 1 sec	< 1 sec	0.20
	MLP	100	73.20	00:04:29	< 1 sec	1.90
	LeNet5	100	83.53	00:00:19	00:00:04	0.39
	HTM	100	84.11	00:00:03	00:00:02	0.23
$\mathcal{S}_{usps\ Train}\langle 1000\rangle$	NN	100	91.18	< 1 sec	< 1 sec	1.96
	MLP	99.7	89.54	00:03:53	< 1 sec	1.90
	LeNet5	100	94.42	00:06:11	00:00:04	0.39
	HTM	100	93.42	00:01:39	00:00:09	0.97
	HTM $OV_{18\times18\ (2,0)}$	100	95.67	00:18:26	00:01:03	6.79
$\mathcal{S}_{usps\ Train}\langle 7291\rangle$	NN	100	94.42	< 1 sec	00:00:05	14.24
	MLP	99.52	94.52	00:50:44	< 1 sec	1.90
	LeNet5	99.87	96.36	00:36:59	00:00:04	0.39
	HTM	99.39	95.57	00:26:45	00:00:30	3.52
	HTM $OV_{18\times18\ (2,0)}$	99.97	97.06	05:41:40	00:03:23	32.89
	HTM [12]	-	96.1	-	-	-
	HTM [8]	-	96.26	-	-	-
	PCA [40]	-	94.42	-	-	-
	5 Layer MLP [41]	-	95.8	-	-	-
	SVM [41]	-	96.0	-	-	-
	Virtual SVM [42]	-	96.8	-	-	-
	Human perf. [43]	-	97.5	-	-	-

Table XI. HTM compared against other techniques on USPS. The last 7 rows of the table (with grayed background) report results already published in the literature. The very last row is an estimation of human performance on USPS classification.

Experimental results on SDIGIT (table IX) show that:

- HTM consistently outperforms the other techniques in term of accuracy. LeNet5 is the second-best and MLP gains the third place.
- The training time is also very advantageous for HTM with respect to MLP and LeNet5. The gap would further increase if fastBuffering was activated (see Table VII). NN has actually no training (we must simply store all the patterns presented).
- Test time is larger in HTM with respect to LeNet5 and MLP, because of higher inference complexity in HTM.
- LeNet5 is the smallest size architecture; HTM is more compact than MLP for small training sets and about the same size for the largest training set. Of course NN size become unfeasible for very large datasets.

Moving on PICTURE (table X) we note that:

- This problem is much more difficult than SDIGIT and USPS as testified by the low accuracy of NN.
- PICTURE appears to be particularly well suited for HTM that totally overcomes other techniques both in terms of accuracy and training time. The black and white nature of PICTURE patterns leads to the creation of a smaller number of (robust) coincidences and groups at level 1 with respect to SDIGIT and USPS (see details in Section 6.2.2 tables); this appear to be one of the reason for the very good HTM performance.
- Due to the large number of (translated) training patterns, NN test time (and size) grows a lot and MLP does not converge on the full problem.
- Here too HTM if the most efficient on training, whereas LeNet5 is faster than HTM in classification.

Finally the analysis of USPS results (table XI) can be summarized as:

- As noted in Section 6.1.3 USPS is not ideal to study invariance and generalization; in fact test set variations are covered by training set patterns to a large extent; this is testified by the good performance of NN.
- While HTM (non overlapped) still outperform NN and MLP on all the experiments done, LeNet5 performs slightly better than HTM (non overlapped) when 1000 and 7291 training patterns are used. However, this is not the case for $HTM_{OV18\times18\,(2,0)}$, whose accuracy is better than LeNet5 both for 1000 and 7291 training patterns.
- HTM (non overlapped) still has the most efficient training while $HTM_{OV18\times18\ (2,0)}$ high complexity leads to a relevant drop in training/test efficiency.
- Other authors have reported HTM performance on USPS [12][8] which are in line with those here presented.
- If we consider only approaches working on pixel intensity (no intermediate feature extraction) and whose training does not rely on the use of further patterns (machine printed or generated), $HTM_{OV18\times18\,(2,0)}$ accuracy is one of the best performance reported so far.

7. Conclusion and Future Work

In this paper we provided an in-deep analysis of Hierarchical Temporal Memory application to pattern recognition. Novel learning approaches (fuzzy grouping and temporal clustering) have been proposed and their efficacy have been demonstrated on three different datasets through a number of experiments. HTM performance (both accuracy and efficiency) was then systematically compared with other pattern classification systems including Convolutional Network, which at today remains one of the most successful implementation of Multi-stage Huber-Wiesel Architectures to vision problems. In almost all our experiments HTM accuracy was better than other system tested and learning was also more efficient. On the other hand, classification time is often longer in HTM (even if not radically) with respect to some of the other systems tested. Finally, node overlapping, saccading and training buffering have been demonstrated to be effective in further improving HTM accuracy and efficiency.

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. This is evident from tables III, IV and V where the number of coincidences and groups (at level 2 and 3) rapidly increases with the number of sub-patterns in the training sequences. On the other hand, to deal with complex pattern recognition problems (with large intra-class variance) the presentation of a large number of potentially long training sequences appear to be necessary for the formation of robust groups. Most of the HTM complexity is due to the coding adopted at higher levels where each coincidence often encodes one (or very few) variation(s) of a given pattern. In other words, given n bits of information HTM higher levels encode $O(n)$ configurations and not $O(2^n)$ as an ideal information theoretic scenario would suggest. The way the brain encode patterns is still debated by neuroscientists (see the discussion on *Grandmother cells and population coding* in Section 2.2 of [27]), 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. If the group is composed of just one cell we

fall into the Grandmother cell case; if all the cells are included we are in a fully distributed code; the intermediate case is the sparse distributed encoding. Going back to the *n* bits, this means that we could split them in smaller groups (also overlapped) and with each subgroups of length *m* we could encode $O(2^m)$ patterns. Translating this in HTM terms could be very important to overcome current limitation. To this purpose new cortical learning algorithms are being developed by Hawkins et al. [44].

Our future research efforts will be devoted to:

- develop novel sparse population coding mechanisms to improve HTM scalability. One possibility is removing the constraint that a coincidence of a node $\mathcal{N}$ must be formed with contributions of all its $childs(\mathcal{N})$ nodes. A coincidence created from a *subset* of $childs(\mathcal{N})$ would cover only a portion of $\mathcal{N}$ receptive field and probably would be more general and (re)usable to encode a larger number of patterns.
- consider saliency of sub-patterns during training and inference; in current implementation a patch containing a salient corner and a totally empty patch are treated in the same way by HTM nodes. We believe that saliency could play an important role in the development of new effective sparse population coding.
- exploit top-down messaging (not addressed in this paper) to develop new effective fusion strategies based on saccading. In particular, instead of biasing successive classification only through the adaption of Prior probabilities in the output node, we think that biasing should be extended to the whole network by influencing the belief of all nodes through top-down (i.e., feed-back) messages.
- apply HTM to difficult object recognition benchmarks such as CalTech and Pascal VOC and systematically compare HTM with state-of-the-art approaches. To deal with the unavoidable increase in the network complexity due to the need of processing larger (color) images, it will be necessary to embed low-level feature extraction at level 0 (e.g., through a bank of Gabor filters as discussed in Section 3.1) and to define different (non exhaustive) strategies to create training sequences based on random walks in huge pattern spaces. Interesting results in this direction have been already achieved by Numenta.
- training HTM incrementally, that is exposing a trained network to new patterns and updating internal groups/coincidences accordingly is not trivial. In fact, changing coincidences/groups at a given network level would invalidate the results of previous training at higher levels. An interesting alternative is initially training an HTM with the algorithms proposed in this paper and then fix coincidences and groups throughout the whole network; then, during successive training (that could be on-line or batch) adapt the probabilities in **PCW** (for the output node) and **PCG** (for the intermediate nodes) as if they were the weights of a neural network trained with backpropagation. This kind of unsupervised pre-training and supervised refinement was recently demonstrated to be successful in deep architectures [13]. We carried out some preliminary experiments in this direction and we achieved interesting results.

ACKNOWLEDGEMENTS

The author would like to thank Greg Kochaniak for providing an HTM implementation which inspired some of the improvements here proposed, in particular the coincidence buffering techniques described in Section 5.1.3. Acknowledgments also go to Pierre Sermanet for providing the EBLearn library and some suggestions for the implementation of LeNet5 and MLP under EBLearn. Finally I would like to thank Dario Maio for the very useful discussions and suggestions about HTM development and experimentations.

REFERENCES

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

[2] 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.

[3] D. George and J. Hawkins, "Towards a Mathematical Theory of Cortical Micro-circuits", *PLoS Computational Biology*, 5(10), 2009.

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

[5] http://www.numenta.com (last accessed April 1st, 2011)

[6] S. Garalevicius, "Memory–Prediction Framework for Pattern Recognition: Performance and Suitability of the Bayesian Model of Visual Cortex", *proc. of Int. Florida Artificial Intelligence Research Society Conference*, 2007.

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

[8] B. Bobier, "Handwritten Digit Recognition using Hierarchical Temporal Memory", University of Guelph, 2007. Available at http://arts.uwaterloo.ca/~cnrglab/?q=system/files/SoftComputingFinalProject.pdf (last accessed April 1st, 2011).

[9] Y. Hall, R. Poplin, "Using Numenta's Hierarchical Temporal Memory to Recognize CAPTCHAs", Carnegie Mellon University, 2007. Available at http://www.pembrokeballet.com/10701-HTM_CAPTCHA.pdf (last accessed April 1st, 2011).

[10] B. Bobier and M. Wirth, "Content-Based Image Retrieval Using Hierarchical Temporal Memory", *proc. Int. Conf. on Multimedia*, 2008.

[11] A. Csapo, P. Baranyi and D. Tikk, "Object Categorization Using Vfa-generated Nodemaps and Hierarchical Temporal Memories", *proc. IEEE International Conference on Computational Cybernetics* (ICCC), 2007.

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

[13] Y. Bengio, "Learning Deep Architectures for AI", *Foundations and Trends in Machine Learning*, vol. 2, no. 1, 2009.

[14] 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.

[15] M. Ranzato et al., "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition", *proc. Computer Vision and Pattern Recognition* (CVPR), 2007.

[16] K. Fukushima, "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition", *Neural Networks*, vol. 1, no. 2, pp. 119-130, 1988.

[17] Y. LeCun et al., "Gradient Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[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] T. Serre et al., "Robust Object Recognition with Cortex-Like Mechanisms", *IEEE trans. on Pattern Analysis Machine Intelligence*, vol. 29, no. 3, 2007.

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

[22] 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.

[23] P. Simard, Y. LeCun, and J. Denker, "Efficient Pattern Recognition Using a New Transformation Distance", in Hanson, S. and Cowan, J. and Giles, L. (Eds), *Advances in Neural Information Processing Systems*, 5, Morgan Kaufmann, 1993.

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

[25] L. Wiskott and T. Sejnowski, "Slow Feature Analysis: Unsupervised Learning of Invariances", *Neural Computation*, vol. 14, no. 4, pp. 715-770, 2002.

[26] J. Bouvrie, L. Rosasco and T. Poggio, "On Invariance in Hierarchical Models", *Proc. Neural Information Processing Systems* (NIPS), 2009.

[27] C. Koch, *The Quest for Consciousness: A Neurobiological Approach*, Roberts & Company Publishers, 2004.

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

[29] J. G. Daugman, "Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-dimensional Visual Cortical Filters", *Journal of the Optical Society of America A*, vol. 2, no. 7, pp. 1160-1169, 1985.

[30] G. Hinton, S. Osindero and Y.W. Teh, "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation*, vol. 18, pp. 1527-1554, 2006.

[31] A.K. Jain, M.N. Murthy and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Reviews*, Nov 1999.

[32] D. George and B. Jaros, "The HTM Learning Algorithms", *Numenta tech. report*, March 1, 2007. http://www.numenta.com (last accessed April 1st, 2011).

[33] J.C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[34] G.J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, Wiley, 2008.

[35] J. Ponce et al., "Dataset Issues in Object Recognition", *LNCS 4170*, pp. 29-48, 2006.

[36] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer, 2001.

[37] J.M. Findlay and I.D. Gilchrist, *Active Vision*, Oxford University Press, 2003.

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

[39] P. Sermanet, K. Kavukcuglu and Y. LeCun, "EBLearn: Open-Source Energy-Based Leraning in C++", *proc. Int. Conf. on Tools with Artificial Intelligence* (ICTAI), 2009.

[40] Y. Hu et al., "Handwritten Digit Recognition Using Low Rank Approximation Based Competitive Neural Network", J. Wang et al. (Eds.), *LNCS 3972*, pp. 287 - 292, 2006.

[41] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 1995.

[42] B. Scholkopf, C.J.C. Burges and V. Vapnik, "Incorporating Invariances in Support Vector Learning Machines", *LNCS 1112*, pp. 47-52, 1996.

[43] J. Bromley and E. Sackinger, "Neural-Network and K-Nearest-Neighbor Classifiers", *Tech. Rep.* 11359-910819-16TM, AT&T, 1991.

[44] J. Hawkins, S. Ahmad and D. Dubinsky, "Hierarchical Temporal Memory including HTM Cortical Learning Algorithms", *Numenta tech. report*, December 10, 2010. Available at: http://www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf (last accessed April 1st, 2011).