

Character Recognition Using Hierarchical Vector Quantization and Temporal Pooling

John Thornton, Jolon Faichney, Michael Blumenstein, and Trevor Hine

Institute for Integrated and Intelligent Systems, Griffith University, QLD, Australia
{j.thornton,j.faichney,m.blumenstein,t.hine}@griffith.edu.au

Abstract. In recent years, there has been a cross-fertilization of ideas between computational neuroscience models of the operation of the neocortex and artificial intelligence models of machine learning. Much of this work has focussed on the mammalian visual cortex, treating it as a hierarchically-structured pattern recognition machine that exploits statistical regularities in retinal input. It has further been proposed that the neocortex represents sensory information probabilistically, using some form of Bayesian inference to disambiguate noisy data.

In the current paper, we focus on a particular model of the neocortex developed by Hawkins, known as hierarchical temporal memory (HTM). Our aim is to evaluate an important and recently implemented aspect of this model, namely its ability to represent temporal sequences of input within a hierarchically structured vector quantization algorithm. We test this *temporal pooling* feature of HTM on a benchmark of cursive handwriting recognition problems and compare it to a current state-of-the-art support vector machine implementation. We also examine whether two pre-processing techniques can enhance the temporal pooling algorithm's performance. Our results show that a relatively simple temporal pooling approach can produce recognition rates that approach the current state-of-the-art without the need for extensive tuning of parameters. We also show that temporal pooling performance is surprisingly unaffected by the use of preprocessing techniques.

1 Introduction

It has become commonplace to compare the remarkable robustness, reliability and adaptability of natural systems with the relatively narrow and fixed capabilities of computer software. Natural processes transform environmental information into reliable survival behaviours in ways that remain unexplained by modern science. Neuroscience has shown that the most sophisticated aspects of this transformational activity occur in the neocortex of the mammalian brain. This naturally suggests that an understanding of the computational principles underlying the neocortex will also provide a key to building similarly powerful software systems.

Viewed at the right level of abstraction, the architecture of the neocortex appears to be relatively simple: it can be divided into six histologically distinct

horizontal layers as well as into vertical *minicolumns* consisting of between 80-100 neurons each (with some exceptions) [1]. In turn, these minicolumns can be grouped into *cortical columns* that share common input and are linked via short-range horizontal connections. Current theories suggest that cortical columns act as feature detectors with a probabilistic output determined by the match to the feature to which the column is sensitive [2].

In a global context, it is widely accepted that different areas of the neocortex are connected together to form hierarchical structures [3]. This has been verified by detailed studies of the visual cortex, where, in the ventral pathway, information passing from the retina via the lateral geniculate nucleus (LGN) enters the lowest level (V1) of the visual cortex and then passes up in sequence to the V2, V4 and inferotemporal cortex (IT) areas. As information moves up the hierarchy, each area appears to be detecting increasingly more general and invariant features of the visual scene, confirming that some form of hierarchical composition is occurring [4].

The discovery that the neocortex has a fairly uniform structure of cortical columns and that these columns are connected in hierarchical structures has led to the development of several *feedforward* computational models aimed both at explaining how the neocortex functions and at developing software that exploits this functionality. One class of model, growing out of Hubel and Weisel's pioneering work on the simple-to-complex cell hierarchy in the visual cortex [18], uses a feedforward hierarchy to effect object recognition [4]. This approach decomposes an image into multiple receptive fields, each connected to an S (simple cell) unit that is tuned to respond to particular features (e.g. oriented bars and edges). The outputs of related S units (sharing the same preferred orientation but differing in size and position) are then connected to C (complex cell) units that detect orientation while remaining insensitive to scale and orientation. These C units are in turn connected to the next layer of the hierarchy, that repeats the S and C unit structure, but integrates an increasingly larger area of the original image. This structure continues until a supervised top layer is used to classify the entire image.

Another stream of research is based on the idea that the neocortex stores sensory information probabilistically, and is therefore best modelled using Bayesian probability theory. This "Bayesian coding hypothesis" has received some support from psychophysical studies although the neuroscientific evidence is still inconclusive [5]. The powerful aspect of the Bayesian model is that it allows for *feedback* within the neocortical hierarchy while providing some means to contain the potential combinatorial explosion of possible interpretations. This feedback provides contextual information that in turn can resolve the ambiguities that typically appear in feedforward models when given noisy, realistic input. Lee and Mumford [6] proposed a model of hierarchical Bayesian inference in the visual cortex that was further elaborated by Dean [7] to produce a *pyramidal Bayesian network*. This work showed that the task of performing Bayesian inference for pattern recognition in large brain-like structures is now becoming tractable (given the availability of modern parallel computing resources).

George and Hawkins [8] have also used a hierarchical Bayesian network to perform pattern recognition on line drawings. This work was a partial realization of a more ambitious project based on Hawkins' model of the neocortex proposed in [9] (now known as hierarchical temporal memory or HTM). HTM extends Lee and Mumford's work by explicitly handling temporal sequences of input within a hierarchical Bayesian framework [10]. In fact, Hawkins sees the fundamental task of neocortical processing as *prediction* and so places temporal change at the centre of his model. This emphasis on time was only partly realized in George and Hawkins original work [8] where movies of the line drawings were displayed and the system was "primed" to expect the repetition of the previous image. More recently, a new implementation of HTM has been developed that explicitly includes a *temporal pooling* component within a vector quantization framework [11]. Temporal pooling refers to the grouping of spatial patterns that appear most frequently in sequential temporal order. Such temporal pooling has a neurophysiological basis. The classic paper of Miyashita [12] demonstrated that neurons in inferior temporal cortex became responsive to stimuli that were presented in close temporal order even though their shapes were quite different.

The aim of the current research is to evaluate the efficacy of temporal pooling as a method for pattern recognition and more generally to evaluate the current HTM architecture. This will be done by testing our own HTM implementation on a large database of cursive handwriting images. In addition, we will be looking at the effect of preprocessing the images using (i) skeletonization and (ii) directional filtering. In the next section we describe how temporal pooling functions within the current HTM framework, and provide some justification for its use. We then provide a background to the cursive handwriting recognition problem and describe the various preprocessing techniques used in the experimental study. In Section 4 we present our results and finally we discuss their significance for future research in the area.

2 Temporal Pooling

The distinguishing feature of Hawkins' HTM model is its emphasis on *time* and the role of the neocortex in encoding "sequences of sequences" of patterns to fulfil the overarching task of *prediction* [9]. Until the recent release of the NuPIC software platform,¹ it was not clear how such temporal information would be incorporated into a hierarchical pattern recognition architecture. The new software achieves this by situating a temporal pooling algorithm within each node of the hierarchy. This algorithm groups together spatial patterns into temporal groups according to how frequently one pattern is succeeded by another during training.

¹ see <http://www.numenta.com/>

2.1 Forming Temporal Groups

To understand temporal pooling in more detail, consider traversing a 3×3 binary pixel sensor across two 5×5 binary pixel images, one containing a vertical line through the middle column and the other containing a horizontal line through the middle row. For each image, we start the sensor at the bottom left corner, sweeping one pixel at a time from left to right, moving up one pixel at the end of the row, sweeping again, and so on, until the image is entirely covered. Then we start again in the bottom left corner, moving *upwards* through each column and then across one pixel at a time. As the sweep progresses, we assign an index k to each *uniquely* appearing 3×3 pixel pattern and count how frequently one pattern follows another. This information is stored in a *time adjacency matrix*, such that for each pattern j that follows pattern i , we increment the count at matrix position i, j by one.²

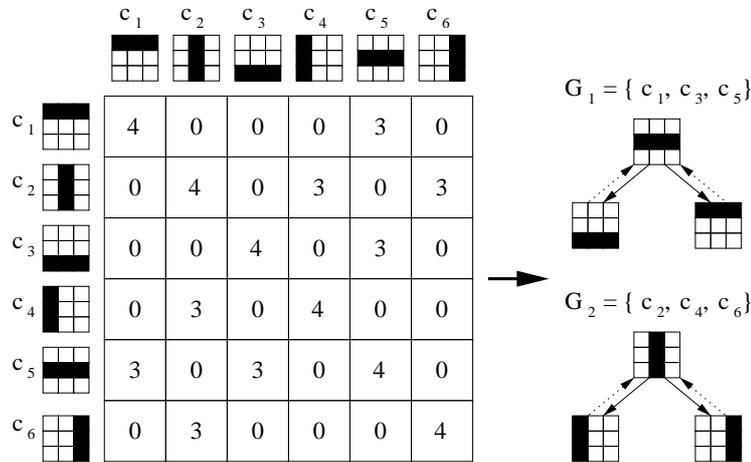


Fig. 1. An example time adjacency matrix.

Figure 1 shows the time adjacency matrix produced from the above sweeping process, with the addition of reflecting the counts through the diagonal (this fills in the counts that would have occurred had we also traversed from right to left and from top to bottom). To form temporal groups, we first select the pattern that was involved in the greatest number of transitions by summing the values in each row and taking the maximum. In this case both c_2 and c_5 have a maximum summed count of 10. Taking c_5 first, we delete it from the set of patterns that can be allocated to a temporal group, and use it to create a new temporal group G_1 . We continue by adding the two most probable neighbours of c_5 to G_1 , i.e. c_1 and c_3 as these are the patterns with the largest counts in the c_5 row. c_1 and

² Note, when the sensor backtracks from the end of one row or column to the beginning of the next, the corresponding patterns are *not* considered to follow each other.

c_3 are now deleted from the set of patterns that can be allocated to a temporal group and we continue recursively to add the two most probable neighbours of c_1 and c_3 to G_1 . In the present case there are no further candidates, as all c_1 and c_3 's neighbours are either already in G_1 or have a zero count. Consequently, G_1 is now *closed* and all members of G_1 and their associated counts are deleted from the time adjacency matrix. The process is then repeated until all remaining patterns have been allocated to a temporal group. In the current example, c_2 forms the “seed” for the next group G_2 and the grouping process completes once c_2 's remaining neighbours (c_4 and c_6) are added to G_2 .

In the general case, the kind of input used to define the patterns remains open and we can further adjust the number of neighbours that can be added at each recursion, and place a limit on the number of patterns that can be added to a group. Otherwise the above description accurately reflects the process of forming temporal groups used in the HTM software platform (see [11] for details).

The idea behind temporal pooling is to collect together patterns representing events that occur consecutively in time and to treat the entire group as representing an underlying *cause*. For instance, in the above example, we were able to reduce six spatial patterns into two temporal groups representing that either a horizontal or a vertical line was the underlying cause of any particular observation. Clearly, in the real-world of pattern recognition we are confronted with thousands of images, few of which reflect the clear cut division of our example. Nevertheless, the relative simplicity of this temporal pooling approach has proved surprisingly robust, as our experiments will show.

2.2 Temporal Pooling within Hierarchical Vector Quantization

If we remove temporal pooling from the current HTM platform, the underlying algorithm uses a form of hierarchical vector quantization (HVQ) [13]. This can be illustrated by considering a 32×32 pixel image and a hierarchy with 64 level one nodes, each connected to a non-overlapping 4×4 pixel area of the original image. Level two consists of 16 nodes, each corresponding to an 8×8 pixel area of the image (via their connection to four spatially adjacent level one nodes). Similarly, level three has 4 nodes, each connected to four level two nodes, and covering a 16×16 pixel area. Finally, a single level four node unifies the image via its connection to the four level three nodes.

Training: In order to train this hierarchy, each node receives input from the layer below, with level one nodes receiving the 4×4 raw pixel image as it is moved one pixel at a time across the node's receptive field. Starting at level one and moving in order to level three, each node executes the following vector quantization procedure (also known as *spatial pooling*): for each input pattern, if it is within a distance D of an existing quantization point then discard, otherwise store the pattern as new quantization point. With the addition of temporal pooling, these quantization points become the c_1, \dots, c_n inputs to the temporal adjacency matrix, where they are combined into temporal groups. Once the

temporal groups for the layer i nodes have been learnt, they can then act as combined input for the layer $i + 1$ nodes, where they are again quantized using the spatial pooler and grouped using the temporal pooler, and so on up to level four. Within level four, the temporal pooler is replaced by supervised learning, i.e. the input from level three is quantized and then directly classified according to an existing set of image labels.³

Recognition: During the recognition phase, the probability that a level *one* node's input matches a quantization point is estimated as: $e^{-d_i^2/\sigma}$ where d_i is the distance of the input from quantization point c_i and σ is a parameter adjusted according to our expectation of Gaussian noise, i.e. the noisier the input, the greater the setting of σ . As each quantization point is associated with exactly one temporal group, we estimate the probability that a particular temporal group is active to be the maximum value associated with the set of its associated quantization points. A node then sends up its probabilistic estimates that each temporal group is active to the next layer. After level one, a quantization point represents the combination of four temporal groups, each associated with a different lower level node. Up to level three, we estimate the probability that a particular quantization point is active by *adding* the probability estimates for each of its lower level temporal groups.⁴ Finally, at level four, the level three temporal group probability estimates are combined *multiplicatively* (again following the HTM default), and the object classification associated with the largest such product is output from the hierarchy.

Recognition also entails sweeping an image across the level one nodes from left to right and from top to bottom, one pixel at a time with an initial 4 pixel offset. In this way, the system will classify a 32×32 pixel image 16 times, producing the most probable classification as its final output.

3 The Cursive Handwriting Recognition Problem

We decided to evaluate our own Java implementation of the HTM platform on the domain of *offline* cursive handwriting recognition. This was partly to extend earlier work that showed HTMs performed well on handwritten digit recognition [14] and partly because handwritten character recognition is such a well-studied area. This means there is no shortage of benchmark problems, or well-engineered

³ Note that the method of traversing the entire image (as if playing a movie) means that during training each node eventually receives the same input as all other nodes on the same level. Consequently we only need train one node per level and then clone the quantization points and temporal groups to the remaining nodes when training is complete.

⁴ Lower level probability estimates were summed (rather than multiplied) to replicate the Numenta implementation. One justification for this is to make the system more tolerant of lower level misclassification, e.g. if a level one's temporal group has a near zero probability, this will have less effect if we are adding probabilistic estimates at level two.

specialized algorithms. After reviewing the area, we decided to focus on the work of Camastra, whose support vector machine (SVM)-based recognizer is one of the best performing of the current state-of-the-art techniques.

In his recent empirical study [15], Camastra created the *C-Cube* database of 57,293 cursive handwritten characters taken from the well-known CEDAR database and from the United States Postal Service database (*C-Cube* is available from ccc.idiap.ch). Here each character was extracted from digitized handwritten postal addresses using standard deslanting, desloping and segmentation techniques [16]. We decided to use this dataset (after scaling down each character to a 32×32 binary pixel image to fit with the HTM implementation described above) and to reproduce Camastra's strategy by testing on 19,133 instances (using the remainder for training).

It is standard practice in the cursive handwriting recognition community to perform preprocessing before recognition, so we experimented with two further techniques: (i) skeletonization, and (ii) directional filtering.

3.1 Skeletonization

The process of skeletonization reduces multi-pixel thick character lines to a single pixel thick line, removing differences in line thickness between characters and emphasizing the underlying shape. To produce character skeletons we used a process of iterative thinning followed by pruning.

Thinning: Firstly, as thinning can overemphasize any holes in an original shape, we used a region filling algorithm to fill all holes of nine pixels or less (see Figure 3). For thinning, we then used the hit-or-miss transform and the sequence of structuring elements taken from [17] and shown in Figure 2a. This transform applies all eight structuring elements to each neighbourhood of 3×3 pixels. If an element matches the neighbourhood then the central pixel is set to the background colour, otherwise it is left unchanged. This process is repeated until no more pixels have changed.

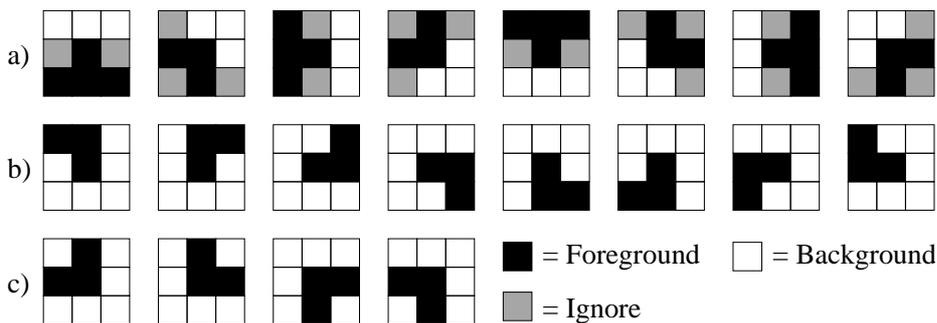


Fig. 2. Structuring elements for thinning.

Pruning: After this, the thinned characters can still contain spurs that we removed using a three-step process: Firstly, we performed morphological pruning using the hit-or-miss transform and the eight structuring elements shown in Figure 2b (only one pass is applied here since further iterations may prune non-spur artefacts). Secondly, spurs of length three or smaller were detected and removed. Here spurs are defined as pixels which have only one neighbouring pixel (i.e. the end of the spur). The line is followed as long as the next pixel has exactly two neighbours and the length of the line is three or smaller. When a pixel is encountered with more than two neighbours then the line segment is confirmed as a spur and the line is removed along with this last pixel. If the line ends with a pixel with only one neighbour then the line is not pruned (i.e. it is an isolated short line). Finally, corners were removed using the hit-or-miss transform and the structuring elements shown in Figure 2c. The entire skeletonization process is illustrated in the example of Figure 3, where an original image of a character “A” is successively scaled, filled, thinned and pruned.

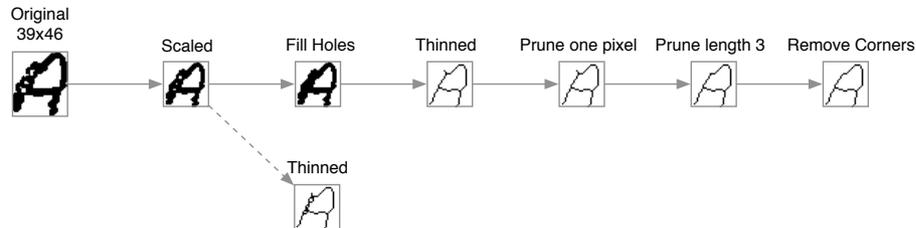


Fig. 3. An example of the skeletonization process. The dotted arrow indicates the result if only scaling and thinning are performed, whereas the horizontal path illustrates the full set of transforms used in the experimental study.

3.2 Directional Filters

We had three motivations for using directional filters. The first, mentioned above, was to reduce the effects of noise in the original images. The second was the similarity of directional filters to the oriented simple cells of the human vision system [18] and the third was the ability of directional filters to separate out structural components of the characters.

We designed 24 filters (see Figure 4) that exhibit the simplicity of masks whilst also detecting 8 different orientations [19]. Each filter was applied to a non-overlapping 4×4 pixel window from the 32×32 scaled image. A 24 element feature vector was then constructed for each window, containing the results of convolving each filter with the window. The feature vector was then converted to a binary string of 24 bits by finding the maximum convolution result and setting elements which have the same result to one and clearing all other elements. The

resulting 24-bit string was fed into level one of the HTM. The rest of the HTM remained unaffected except for the distance measure D which is now calculated over 24 dimensions instead of the previous 16.

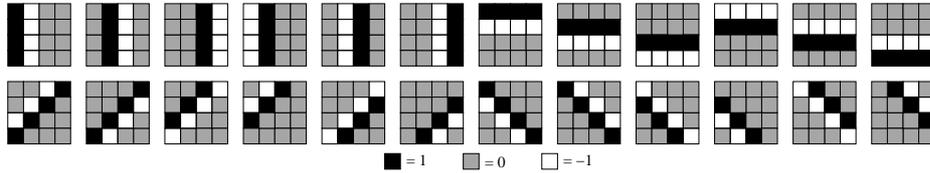


Fig. 4. Directional filters.

4 Experimental Results

For the experimental study we first set up our HTM software to use the defaults of the NuPIC platform. This involved setting the maximum temporal pooling group size to 32 at each level and setting the number of temporal pooling neighbours in the grouping algorithm to three at level one and two at all other levels (see Section 2). We used the Hamming distance to calculate D and during training we set $D = 0$, i.e. we recorded *every* spatial training pattern observed at level one as a quantization point. This meant that temporal pooling was the only method for grouping the input during training. The Hamming distance was again used to measure D during recognition. The other settings for the algorithm were the same as those described in the example of Section 2.2.

We also experimented with various settings for the maximum temporal group size at each level, as this proved important for our preprocessing techniques. The overall results are shown in Table 1, which also reproduces the results on the same data set for the algorithms in Camastra’s study [15].

Table 1. Recognition rates in percent on the *C-Cube* dataset. The support vector machine (SVM), learning vector quantization (LVQ) and multi-layer perceptron (MLP) results are taken directly from [15]. The preprocess row defines the preprocessing technique and the group size row defines the HTM maximum temporal group size for levels one, two and three respectively. Note, the HTM, SVM and MLP results classify upper and lower case letters together, i.e. into 26 classes. Higher recognition rates are possible when dissimilar upper and lower case letters are separated into different classes, as is the case for the LVQ results, which Camastra only reported for the 39 class case.

Method	HTM						SVM	LVQ	MLP
	none		skeleton		directional				
Preprocess							n/a	n/a	n/a
Group size	32,32,32	16,16,16	32,32,32	6,16,32	6,16,32	1,10,20	n/a	n/a	n/a
Rec. rate	84.72	85.58	83.53	85.22	73.31	84.32	89.61	84.52	71.42

5 Discussion and Conclusions

The results show that HTM has its best recognition rate (85.58%) on the raw image with temporal group sizes of 16 at each level (although the default setting of 32 is only slightly worse). Interestingly, the two preprocessing techniques make little difference to the HTM’s performance, although directional filters do not work well unless the level one group size is set to one (causing the directional filters to replace the level one temporal pooling). This reflects the fact that the directional filters are already collapsing the level one data into a maximum of 24 quantization points from which further temporal pooling will result in a significant loss of detail. Otherwise, varying the temporal group sizes only had a small effect on performance. Combining this together shows that HTM performance remained remarkably robust to significant changes both in the form of its input and to the settings of the temporal grouping parameter.

Comparing the HTM’s performance with Camastra’s results shows that the HTM was better than both the LVQ⁵ and MLP implementations reported in [15] but did not reach the 89.61% recognition rate of the SVM. However, when analysing these results we must bear in mind that the SVM algorithm could only reach such high recognition rates after an extensive period of tuning classifiers for each letter (estimated by Camastra at between three to four months of work). In contrast, the HTM can achieve 85% recognition using default settings. We should also consider that this is a *partial* and simplistic implementation of the HTM platform. There are more sophisticated vector quantization algorithms that could be applied before temporal pooling is invoked, and the temporal pooling algorithm itself is quite naïve. For instance, it is unable to represent sequences that share a common quantization point. The hierarchical approach of HTM also has the potential to reuse a letter recognition hierarchy within a larger word recognition hierarchy, without the need to retrain and re-engineer the existing system. Finally, HTM has the potential to predict future input via the information stored in the temporal adjacency matrix (this “time-based inference” has already been implemented in the latest release of the NuPIC software).

In conclusion, we have shown that a fairly simple temporal pooling algorithm, embedded within a hierarchical vector quantization algorithm, can come within 4% of one of the best algorithms available, on a data set for which the competitor algorithm was specifically tuned. This shows there is significant potential for pattern recognition algorithms that can exploit temporal connections between moving patterns, even when the patterns are not obviously part of a dynamic environment (i.e. as in offline character recognition). We have also shown that temporal pooling is surprisingly robust, both to the form of its input and to the settings of its internal parameter.

⁵ While the best HTM result is only 1% better than for LVQ, it should be noted that the LVQ result reports the optimal case of 39 classes. We would expect the HTM result to improve further if difficult letters were separated into different classes.

In future work it would be worthwhile to experiment with different temporal and spatial pooling algorithms and to exploit the temporal adjacency matrix for prediction. A further important step is to implement temporal pooling within a full hierarchical Bayesian network, enabling disambiguating feedback from higher to lower levels.

References

1. Mountcastle, V.B.: Introduction to the special issue on computation in cortical columns. *Cerebral Cortex* **13**(1) (2003) 2–4
2. Dean, T.: A computational model of the cerebral cortex. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05). (2005) 938–943
3. Felleman, D., van Essen, D.: Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex* **1**(Jan/Feb) (1991) 1–47
4. Serre, T., Oliva, A., Poggio, T.: A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences of the USA* **104**(15) (2007) 6424–6429
5. Knill, D.C., Pouget, A.: The Bayesian brain: the role of uncertainty in neural coding and computation. *Trends in neurosciences* **27**(12) (2004) 712–719
6. Lee, T.S., Mumford, D.: Hierarchical Bayesian inference in visual cortex. *Journal of the Optical Society of America A* **20**(7) (2003) 1434–1448
7. Dean, T.: Scalable inference in hierarchical generative models. In: Proceedings of the 9th International Symposium on Artificial Intelligence and Maths. (2006)
8. George, D., Hawkins, J.: A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN-05). (2005)
9. Hawkins, J., Blakeslee, S.: *On intelligence*. Henry Holt, New York (2004)
10. Hawkins, J., George, D.: Hierarchical temporal memory: Concepts, theory and terminology. Technical report, Numenta, Inc, Palto Alto (www.numenta.com/Numenta_HTM_Concepts.pdf) (2006)
11. George, D., Jaros, B.: The HTM learning algorithms. Technical report, Numenta, Inc, Palto Alto (www.numenta.com/Numenta_HTM_Learning_Algos.pdf) (2006)
12. Miyashita, Y.: Neuronal correlate of visual associate long-term memory in the primate temporal cortex. *Nature* **335** (1988) 817–820
13. Gersho, A., Gray, R.M.: *Vector quantization and signal compression*. Springer, Boston (1992)
14. Thornton, J., Gustafsson, T., Blumenstein, M., Hine, T.: Robust character recognition using a hierarchical Bayesian network. In: Proceedings of the 19th Australian Joint Conference on Artificial Intelligence, Hobart, Tasmania (2006) 1259–1264
15. Camastra, F.: A SVM-based cursive character recognizer. *Pattern Recognition* **40** (2007) 3721–3727
16. Nicchiotti, G., Scagliola, C.: Generalised projections: a tool for cursive character normalization. In: Proceedings of the 5th International Conference on Document Analysis and Recognition, New York (1999) 729–732
17. Gonzalez, R., Woods, R.: *Digital image processing*. Addison-Wesley, New York (1992)
18. Hubel, D.H., Wiesel, T.N.: Early exploration of the visual cortex review. *Neuron* **20** (1998) 401–412
19. Pratt, W.K.: *Digital image processing*. John Wiley and Sons, New York (2001)