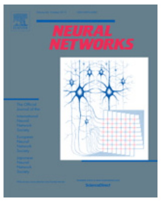
Neural Networks 54 (2014) 70–84

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Learning invariant object recognition from temporal correlation in a hierarchical network

Markus Lessmann, Rolf P. Würtz [*]

Institute for Neural Computation, Ruhr-University Bochum, Germany

ARTICLE INFO

Article history:
Received 19 July 2013
Received in revised form 21 February 2014
Accepted 23 February 2014

Keywords:
Invariant object recognition
Learning of temporal sequences
Hierarchical network

ABSTRACT

Invariant object recognition, which means the recognition of object categories independent of conditions like viewing angle, scale and illumination, is a task of great interest that humans can fulfill much better than artificial systems. During the last years several basic principles were derived from neurophysiological observations and careful consideration: (1) Developing invariance to possible transformations of the object by learning temporal sequences of visual features that occur during the respective alterations. (2) Learning in a hierarchical structure, so basic level (visual) knowledge can be reused for different kinds of objects. (3) Using feedback to compare predicted input with the current one for choosing an interpretation in the case of ambiguous signals. In this paper we propose a network which implements all of these concepts in a computationally efficient manner which gives very good results on standard object datasets. By dynamically switching off weakly active neurons and pruning weights computation is sped up and thus handling of large databases with several thousands of images and a number of categories in a similar order becomes possible. The involved parameters allow flexible adaptation to the information content of training data and allow tuning to different databases relatively easily. Precondition for successful learning is that training images are presented in an order assuring that images of the same object under similar viewing conditions follow each other. Through an implementation with sparse data structures the system has moderate memory demands and still yields very good recognition rates.

1. Introduction

Visual processing is probably the best-examined brain function in all neuroscience. Myriads of studies and experiments in neurophysiology, neuroimaging, cognitive neuropsychology, computational neuroscience and related fields like psychophysics have gathered a huge amount of data about the functionality of visual processing areas in the brain. Turning this data into functional models is difficult due to its diversity and lack of coherence. Thus, inductive reasoning is not enough for understanding object recognition in humans and has to resort to general concepts that allow deductive conclusions. This lack of theoretical concepts was criticized by Hawkins in Hawkins and Blakeslee (2004), where he proposed filling this gap by means of his *Memory Prediction Framework*. It is based on the assumption that the neocortex executes the same algorithm on input from each sensory modality and even for producing motor output. This idea goes back to Mountcastle (1997), who postulated this after realizing that on the neurophysiological level the neocortex looks remarkably alike in all of its regions despite the differences in processing. The Memory Prediction Framework collects many ideas about neural information processing in a coherent framework and has led to a software system named *Hierarchical Temporal Memory (HTM)*, which implements the main concepts (George, 2008). Three main ideas can be found in the HTM:

1. Learning of temporal sequences for creating invariance to transformations contained in the training data.
2. Learning in a hierarchical structure, in which lower level knowledge can be reused in higher level context, making memory usage efficient.
3. Prediction of future signals for disambiguation of noisy input by usage of feedback.

These principles can also be found in other systems (but not always all of them at the same time) as for example VisNet2 (Rolls, 2008) or HMAX (Serre, 2006).

In this article we present a computer vision system, which is able to do invariant object recognition with very good results by implementing these basic ideas. The system is efficient in memory

[*] Corresponding author. Tel.: +49 234 3227994.
E-mail addresses: markus.lessmann@ini.rub.de (M. Lessmann), rolf.wuertz@ini.rub.de (R.P. Würtz).

http://dx.doi.org/10.1016/j.neunet.2014.02.011
0893-6080/© 2014 Elsevier Ltd. All rights reserved.

usage since spatial and temporal patterns extracted on different levels are stored globally thus avoiding the need to store patterns repeatedly if they are encountered at different network positions (which can happen in the HTM). Additionally, the system excludes weakly active neurons from further computations. Together with weight pruning, simulations can be sped up enough to handle large databases containing thousands of images and categories. This can hardly be done with belief propagation equations like those in the HTM.

The layout of the article is the following: in the next chapter we present general techniques used for object recognition and some examples. In Section 3 we introduce our system in detail. Section 4 describes inference in the network and Section 5 how learning is done. Technical details of the implementation are explained in Section 6. Conducted experiments and results are presented in Section 7. Section 8 closes the article with a conclusion and an outlook on future work.

2. Related work

As a full review of object recognition would be far beyond the scope of this article we will restrict the discussion to a few examples. The oldest is VIEWNET by Bradski and Grossberg (1995), which combines different 2D-views of the same object (which could be related by temporal proximity) for deriving a 3D-model. This is done by feeding downsampled log-polar-descriptions of preprocessed and thereby noise-reduced contours of test objects into a Fuzzy ARTMAP. This system learns 2D-view categories, which can be combined to 3D-categories and yield a recognition rate of 90% on a database of images of airplane models. By accumulating votes for 3D-objects over several views the recognition rate can be increased to up to 98.5% when using 3 views of the same object consecutively.

Another example is the work of Luciw, Weng, and Zeng (2008). They perform supervised training of a neural network with two 2-dimensional topographic neuron layers and a top layer holding one neuron for each trained class using Hebbian learning rules. Whereas the lowest (sensory) layer 0 provides visual features from the current input image the middle layer 1 gets input from layer 0 as well as feedback input from top layer 2 with a delay of one time step (and hence one image). Feedback from layer 2, which is weighted against feedforward input, is provided using the same synaptic connection weights as for feedforward input into it. Since training images are ordered in sequences showing different objects rotating in 3D in most cases the next training image will show the same object as the current one, and the feedback signal can be seen as a prediction of future top layer activity. This is similar to the proposed system, which uses predictive feedback on all layers except the highest. Similar is also the handling of lateral inhibition by sorting neurons according to activity and inhibiting all except the k most active ones, but the proposed system does this locally at each network position.

In comparison to this (Mobahi, Collobert, & Weston, 2009) only use the continuity assumption during training of a deep convolutional network with backpropagation. The training set is enhanced with a video showing different objects undergoing the same 3D rotation as the training objects. The usual training criterion is expanded with a second term enforcing similar activations in the next-to-last-layer for two consecutive images of the video and dissimilar activations for non-consecutive frames and minimized alternatingly for labeled training images, randomly chosen consecutive video frames and randomly chosen non-consecutive frames. The authors then demonstrate that recognition on COIL100 dataset is improved even if the additional video shows totally different objects.

In Bergstra and Bengio (2009) temporal coherence is used for unsupervised learning of decorrelated filters whose responses vary slowly on videos of natural images. This is reached by minimizing a criterion involving the correlation coefficient of distinct features and the change of responses of each single feature over small batches of consecutive movie frames. The resulting filters are used as initialization for supervised training of a network with one layer of hidden neurons using a special complex-cell activation function. It is shown that prelearned features decrease test set error on MNIST data base compared to random initialization of weights and thus improve generalization.

In a similar approach (Zou, Ng, Zhu, & Yu, 2012) learn slowly varying features from videos which are created by tracking keypoints in natural videos. The desired features can be extracted by feedforward computation from input images and minimize a criterion including terms for slow change in time, minimal information loss and sparsity. This is done on two layers, where features of the second layer are learned from first level features after dimensionality reduction using PCA. Features of both levels are applied in training an SVM for object recognition and improve test results on several databases by ca. 4%–5% compared to features learned without enforcing slowness. The features are also demonstrated to possess some invariance against translation, rotation and zooming.

Another branch is the group of *bag of words* (BoW) methods as used, e.g., by Csurka, Dance, Fan, Willamowski, and Bray (2004) and Lazebnik, Schmid, and Ponce (2006). These methods collect typical visual features from training images in a bag of words (codebook) and additional information about their distribution with respect to object category. This information is used for inference after finding in the BoW features most similar to ones from a test image. Features are stored without any information about relative position within the object, and arbitrarily scrambled objects are still assigned to the same category.

The system by George and Hawkins (2009) is the most similar model to the network presented in this paper. They build a converging hierarchical network of nodes, with the lowest layer covering the input image. Each node fulfills two tasks, namely first to learn a small codebook of input patterns and then to learn temporal groups of those patterns. The first task can be accomplished in several ways, for example by choosing randomly input patterns for memorization or by using online k-means for the determination of suitable prototypes. The input patterns to be learned are either input data that comes from sensors (e.g. image features) or outputs of lower level nodes to which the current node is connected. These outputs reveal which temporal group is currently active in the lower level node and to which certainty. Since input patterns combine this information for several adjacent nodes they are also called coincidence patterns. The degrees of certainty about groups can only be delivered by a node when it has finished learning these temporal groupings, and an HTM network has to be trained layer by layer. For learning temporal groups a transition probability matrix is recorded during scanning of the training data. The matrix holds the information which spatial input patterns tend to follow each other in time. If a matrix element is big the corresponding input patterns often appeared consecutively in the input data and hence should be grouped together. The grouping is derived by clustering the transition probability matrix, which can be done using graph partitioning algorithms or incremental clustering. Once the groupings are known a node can do inference about which temporal group it has observed based on its certainty about currently active input patterns. The certainty of each temporal group being active is the output that a node sends to one on the next level. This information together with the same from adjacent nodes constitute higher level input patterns. The temporal groups of the highest level node represent different causes the system has learned to distinguish, in the case of an object recognition system each group represents one category. The inference in each node is carried out using a set of belief propagation equations, which compute degrees of belief for coincidence patterns and temporal groups. Degrees of belief for temporal groups are sent to the next higher level and those about coincidence patterns are fed back to the lower level.

A further approach with some conceptual similarity to the one proposed here is *Slow Feature Analysis* (SFA) by Wiskott and Sejnowski (2002). SFA extracts decorrelated, slowly varying features from input variables that vary quickly in time. This is done by first expanding the input variables using non-linear functions which have zero temporal mean and identity covariance matrix. Then output functions are constructed as linear combinations of these expansions using weights chosen to minimize temporal variance of the output functions. This can be achieved by finding the minimum of a quadratic form of the weights and a matrix containing the time averaged derivatives of the expansions. The final output functions reveal aspects of the input signal that vary slowly in time. Such a learning process is carried out by an SFA module. For object recognition this scheme is applied in a hierarchical network where SFA modules on the lowest level get the intensity values of several neighboring pixels of an image as input (Franzius, Wilbert, & Wiskott, 2011). Its output functions are the input to an SFA module on the next level. This is repeated up to a single module at the top level. When this system learns from image sequences showing objects undergoing transformations one of the slowest signals at the top level is correlated with one of the identities of presented objects and hence can be used for classification. Additionally, further aspects like object location can be determined. Although different in its learning scheme to conventional neural networks SFA also approximates a multimodal mapping between input and output variables using linear combinations of (potentially) nonlinear functions. In this aspect it is similar to ordinary neural networks and can for example be mapped to a radial basis function network.

Finally, our system bears some resemblance to VisNet2 by Rolls (2008). This is also a hierarchical multilayer neural network learning sequences of visual features. The most important difference is, that VisNet2 uses a biologically plausible local learning rule, the trace rule, for determining temporal groups.

There is also a huge variety of possible feature extraction methods to precede the network learning. These have recently been identified as crucial for successful invariance (Lindeberg, 2013). One of the most successful and widely used features are SIFT (Lowe, 2004). In the present work we have not varied the basic features but used templates of Gabor jets throughout.

3. The network

The system we present here is partly a neural model, in that it contains entities with varying activities connected by synapses whose strengths are determined using a learning algorithm. They excite each other via these connections in both feedforward and feedback direction. It is not a fully neural model in that it does not apply a local learning rule to each possible connection of two neurons but learns these using a less biologically realistic scheme. Fig. 1 shows the general structure of the network. In the following, we specify the two types of neurons, the definition of their connection weights and a method to learn these weights. The network is not very complicated, the main burden in the description is to define and motivate the different types of weights, how they are used with and without feedback, and how they are learned.

Neurons are representatives of prototypes (like in a self-organizing map) and also connected by synaptic weights like in a

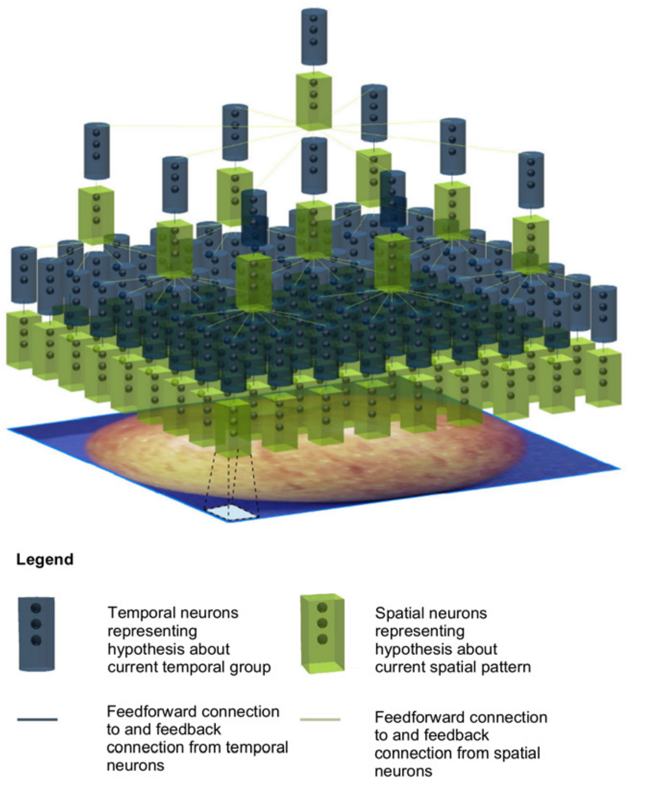

Legend
Temporal neurons representing hypothesis about current temporal group
Spatial neurons representing hypothesis about current spatial pattern
Feedforward connection to and feedback connection from temporal neurons
Feedforward connection to and feedback connection from spatial neurons

Fig. 1. Visualization of the network architecture. Connections of nodes represent possible synaptic connections between all neurons in one node and all in the other.

perceptron. Activation is by *similarity* of the input pattern to the prototype (during learning) and by the usual hyperbolic tangent of the sum of weighted inputs (for inference) and additionally by lateral inhibition. The neurons must represent important image features for all possible objects and their number will consequently be large (each input on the lowest level may activate one of over 10^5 neurons). This leads to large networks. In order to keep the simulation tractable we rely on sparse representations, meaning that only significantly active neurons are simulated and the rest is ignored. Therefore, the following description of the network will also include the data representations and computational shortcuts used.

The whole network consists of several (usually 3 or 4) layers of *node positions*, each made up of two sublayers of *nodes*. These nodes are collections of currently active neurons, which are represented by an integer index for identification and a real activity value. Nodes store this information in a hash map for fast data access and can sort neurons according to their activity to suppress all but the most active ones by lateral inhibition. Nodes $\mathbf{N}^{l,s}_{x_1,x_2}$ at position (x_1, x_2) in the lower sublayer of level l store neurons $\mathbf{n}^{l,s}_i$ (called spatial neurons) representing spatial patterns $\mathbf{s}^l_i$ on that level, whereas in higher sublayers nodes $\mathbf{N}^{l,t}_{x_1,x_2}$ hold neurons $\mathbf{n}^{l,t}_j$ (named *temporal neurons*) representing possible temporal feature groups $\mathbf{t}^l_j$. Inactive neurons are not stored.

The connectivity between levels is *convergent*, which means that (the temporal neurons of) C nodes on the lower level provide the input to one single (spatial) node on the next level. Between sublayers the mapping is one to one, each spatial node is connected to exactly one temporal node (and vice versa). The convergence is chosen to result in one single node in the top most level, each of the temporal neurons in this node is supposed to represent one of the object categories the system has learned. Thus the most active temporal neuron at the top level signals the recognized object category. Most experiments in this article use a convergence factor of $C = 9$, meaning that 3 nodes per direction converge on 1 node on the next level. This assignment yields a typical architecture of 9×9 nodes per sublayer on level 0, 3×3 on each sublayer of level 1 and 1 node per sublayer on level 2. Temporal neurons on level l have connections $W_{i,j}^{l,s;l,t}$ to spatial neurons of the same level, from which they get their input and connections $W_{j,i}^{l,t;l+1,s}(v)$ to spatial neurons on the next level to which they send their output and which depend on the position v of neuron $\mathbf{n}_j^{l,t}$ in the set of node positions converging on a higher level node. Weights do not depend on the absolute node position, making the network convolutional (LeCun et al., 1989). In Fig. 1 these are depicted by blue and green lines, where a line between two nodes represents all possible connections between a neuron of one node and one from the other. The weights are the same in feedforward and feedback direction

$$W_{i,j}^{l,s;l,t} = W_{j,i}^{l,t;l,s}, \tag{1}$$

$$W_{j,i}^{l,t;l+1,s}(v) = W_{i,j}^{l+1,s;l,t}(v). \tag{2}$$

In the sparse implementation, weights do not exist from the beginning but are established when training on one sublayer has finished. Thus only after full training of the network all synaptic connections are defined.

In order to recognize object categories irrespective of transformations like changes in scale or viewing angle the system has to learn from image sequences showing objects undergoing those transformations. By selecting these transformations observable on the training images it can be chosen which invariances the system is supposed to develop. On each level it first builds a database (codebook $\mathbf{C}^l$) of features typical for these objects using vector quantization. Then it learns sequences, in which these features occur at the same position when the object view changes. These sequences (or better temporal groups, because the actual order does not matter) form the input to the next layer of the network. Spatially adjacent groups constitute new spatial patterns that again build up a codebook learned by vector quantization. Again temporal groups are formed from these patterns and the process is repeated until the top level of the hierarchy. That means the system has to be trained layer by layer.

To understand the purpose of the temporal groups imagine the following: the system looks at images of a rotating object (a horse for example). The features at a certain image position change over time as the horse rotates. A feature of the left profile of the horse's head may be followed by a frontal view feature and a right profile one. If this happens often enough they are put into the same temporal group. This group now represents a horse's head (partly) independent of viewing angle. If enough views are presented during learning, any head feature may appear in a test image. As long as it is similar enough to some training view the same temporal group will be activated and the horse's head will be recognized. The same goes for other parts of the horse, which result in different temporal groups. These are used on the next level of the network hierarchy by concatenating their indices in an order depending on their relative spatial position. Thus relative positions are encoded so that the system only accepts horses which have (for example) the head above the legs and not the other way round. A further important reason for the hierarchical structure is the reuse of lower level features. The legs of horses and cows may look very similar. Hence the same temporal (leg) group can be used for representing horses and cows (and even dogs or cats and so on). On higher levels it is combined with other groups in which horses and cows differ more, thus enabling differentiation. This is efficient because the system needs only one representation for body parts of different species. The same holds for other cases like wheels of different cars and so forth.

It may happen that the system encounters (on some level) a feature similar to two different known patterns from different objects. Then it may confuse them. However, in previous images the feature at the current position should sometime have been unanimous and should have activated the correct temporal group. A feedback signal from temporal groups of previous images can select the correct interpretation by enhancing the activation of the feature in question that best fits to the last activated temporal groups. Also a pattern may appear in two different contexts (e.g. during scaling and during rotation of an object) and therefore is associated with two different temporal groups. Feedback now allows to choose the one that better fits in the temporal context.

4. Inference

Inference in our system is done by computing activities of all neurons for a given input image and then reading out the index of the most active temporal neuron at the top level. This neuron identifies the recognized object category. Activities on the lowest level are calculated as similarities to the neurons' prototypes. Then, activities are calculated from bottom to top for one node position after another according to

$$a_i = \tanh\left(\sum_j W_{ji} a_j\right), \tag{3}$$

with j running over all neurons sending connections to the given one. Once learning is finished this can also be done in parallel for node positions on the same level. Activities are calculated and transferred to an *activity stack* storing the activities of the last T time steps. This stack is necessary for learning and for calculation of delayed feedback.

The use of synaptic weights instead of codebooks during inference is necessary to allow the efficient use of multiple hypotheses. It is also expected to make the system robust against partial occlusion. Spatial neurons activate temporal neurons on the same level and these can activate spatial neurons on the next level. Since one temporal neuron at one node-position is enough to excite a higher level spatial neuron such a neuron can also be activated even if only one of its parts is observed.

4.1. Computing spatial feedforward input

First the spatial pattern at the current node position is extracted. On the lowest level, this is an image feature (parquet graph, see Section 6.1) on the corresponding image position, on the higher levels a concatenation of the indices of the most active temporal groups at node positions on the previous level that converge on the current node.

4.2. Inhibiting spatial neurons

After calculation of the feedforward input the amount of active neurons at the node position is reduced by setting the activity of all but the K most active neurons to zero and deleting them from the hash map. During learning $K = 1$, during recall it can also be higher. This step is an efficient implementation of lateral inhibition between neurons at the same node position. Without that inhibition the network runs into overexcitation and the first neuron which became most active at the top level will remain the winner for all following images. Additionally, this saves working memory and processing time since fewer operations need to be done for computation of activities on the next level. This step can also be seen as a reduction of the number of *hypotheses* about which spatial or temporal pattern was observed at the current position in space and time.

4.3. Computing spatial feedback input

The next step is to add feedback input to the remaining active neurons. This feedback comes from temporal groups which have been active on the previous T images. If unrestricted feedback is given the system again runs into overexcitation. The connection weights between spatial patterns and temporal groups will be defined in Section 5.

4.4. Application of the activation function

The neuron activities are now processed by the activation function. The hyperbolic tangent was used for all experiments. The activation function prevents activity values from growing to infinity, which can happen because of feedback connections. It also provides one of the two nonlinearities in the system enabling robust classification (the other one being the inhibition of neurons).

4.5. Transfer to activity stack

Then activities are written into the activity stack of the last T images in a compressed form. If currently the same neurons are active as in the memory for time step 0 only their activities are updated. If a neuron has become inactive or a new one was activated all stored activities of the last time steps are shifted by one position (and the activity at time step $T - 1$ is deleted) and current activity is copied to position 0. This has the effect that not only one neuron occupies all stored positions and is the only one giving feedback. Also the system is prevented from only recording transitions of one neuron to itself while learning temporal groups. During learning the activity stack is emptied when a new object category is presented. This prevents the system from learning temporal transitions between different categories. The same effect would be reached by placing several empty images between images of different object categories.

4.6. Computation for temporal neurons

Now the same kind of calculations are done for neurons representing temporal groups. At first temporal neurons collect their feedforward input by summing up activities of connected spatial neurons multiplied with the corresponding connection weights. Then inhibition is applied and feedback is given to the remaining temporal neurons using synaptic weights to spatial patterns that have been active on the next level. This time only neurons that were active on the last image are used. Then the activation function is used again and activities are moved to the activity stack.

4.7. No inference possible

If learning data was too sparse or too few hypotheses are kept it may happen that no active temporal group can activate a spatial pattern on the next level because there are no connections. In that case no decision can be made about the category of the object on the current image. This problem diminishes with more learning data and more active hypotheses (higher K) during testing.

The computation of neural activities is subsumed with pseudocode in Algorithms 1, 2 and 3.

Algorithm 1: Activity calculation for inference on all and for training on finished levels

```
for $l \leftarrow 0$ to highest trained level do
    for $x \leftarrow 0$ to xSize[l]-1 do
        for $y \leftarrow 0$ to ySize[l]-1 do
            compute activity for $\mathbf{N}^{l,s}_{x,y}$;
            compute activity for $\mathbf{N}^{l,t}_{x,y}$;
        end
    end
end
```

Algorithm 2: Computation for $\mathbf{N}^{l,s}_{x,y}$

```
if $l{=}0 \vee$ training-mode then
    extract $\mathbf{s}^l$ at current position;
    find nearest neighbor $m$ in $\mathbf{C}^l$;
    set activity of neuron $m$ to similarity with $\mathbf{s}^l$;
end
else
    foreach input node $\mathbf{N}^{l-1,t}_{x',y'}$ of $\mathbf{N}^{l,s}_{x,y}$ do
        get weights to potential neurons in $\mathbf{N}^{l,s}_{x,y}$;
        foreach active neuron $n \in \mathbf{N}^{l-1,t}_{x',y'}$ do
            foreach potential neuron $m \in \mathbf{N}^{l,s}_{x,y}$ do
                multiply activity of $n$ with weight to $m$;
                add to activity of $m$;
            end
        end
    end
end
sort neurons in $\mathbf{N}^{l,s}_{x,y}$ according to activity;
delete all but $K$ most active neurons;
foreach active neuron $m \in \mathbf{N}^{l,s}_{x,y}$ do
    for $t \leftarrow 0$ to $T - 1$ do
        get weights to neuron $m$;
        foreach active neuron $n \in \mathbf{N}^{l,t}_{x,y}$ at time $t$ in stack do
            multiply activity of $n$ with weight to $m$;
            add to activity of $m$;
        end
    end
end
foreach active neuron $m \in \mathbf{N}^{l,s}_{x,y}$ do
    apply hyperbolic tangent to activity;
    store in activity stack;
end
```

5. Learning

Learning is done level by level and in two subsequent steps. First a codebook $\mathbf{C}^l$ of input patterns is learned using vector quantization for spatial neurons, then temporal groups of these are established. This is done on all positions of the network, but globally with only one codebook per level and one container for the groups. This is a shortcut, assuming that with enough learning examples and object presentations at all positions of the network the nodes should locally encounter the same patterns and groups. Doing this globally from the beginning can be seen as a way of speeding up learning (because objects do not need to be presented at each positions of the network) and saving memory (because not each position has to store its own codebook). Meanwhile, the database can be bigger and thus more distinctive than the small codebooks used in the HTM of George (2008).

Algorithm 3: Computation for $\mathbf{N}_{x,y}^{l,t}$

```
foreach active neuron $n \in \mathbf{N}_{x,y}^{l,s}$ do
    get weights to potential neurons in $\mathbf{N}_{x,y}^{l,t}$;
    foreach potential neuron $m \in \mathbf{N}_{x,y}^{l,t}$ do
        multiply activity of $n$ with weight to $m$;
        add to activity of $m$;
    end
end
sort neurons in $\mathbf{N}_{x,y}^{l,t}$ according to activity;
delete all but $K$ most active neurons;
foreach active neuron $m \in \mathbf{N}_{x,y}^{l,t}$ do
    foreach active neuron $n \in \mathbf{N}_{x'',y''}^{l+1,s}$ (connected higher level spatial node) at time 0 in stack do
        get weights to neuron $m$;
        multiply activity of $n$ with weight to $m$;
        add to activity of $m$;
    end
end
foreach active neuron $m \in \mathbf{N}_{x,y}^{l,t}$ do
    apply hyperbolic tangent to activity;
    store in activity stack;
end
```

5.1. Learning prototypes for spatial neurons

The set of prototypes of spatial neurons is defined as a subset of all training features, such that the maximal similarity between all features in $\mathbf{E}^l$ and codebook features is above ϑ_Q

$$\forall_{\mathbf{s}^l \in \mathbf{E}^l} \exists_{\mathbf{s}_i^l \in \mathbf{C}^l} : S(\mathbf{s}^l, \mathbf{s}_i^l) \geq \vartheta_Q . \tag{4}$$

When the codebook is finished, each prototype gives rise to one spatial neuron.

For learning the codebook on level l all training images are traversed in a fixed order. On each image visual features (see Section 6.1) are extracted and activities on all levels $l' < l$ of the network are computed. On each node position the spatial input pattern $\mathbf{s}^l$ is determined and its nearest neighbor $\mathbf{s}_{i_{\text{best}}}^l$ in the codebook $\mathbf{C}^l$ is looked up:

$$\mathbf{s}_{i_{\text{best}}}^l = \arg\max_{\mathbf{s}_i^l \in \mathbf{C}^l} S(\mathbf{s}^l, \mathbf{s}_i^l). \tag{5}$$

If $S(\mathbf{s}^l, \mathbf{s}_{i_{\text{best}}}^l) > \vartheta_Q$ (4) is fulfilled for all training patterns seen so far, and the codebook remains unchanged. Otherwise, the pattern $\mathbf{s}^l$ is added to the codebook, and the updated codebook again fulfills (4) for all training patterns seen so far. By induction, finally (4) holds for all features encountered during training. On level 0 the spatial patterns correspond to image features and the image feature similarity function S_{Image} is used (see Eq. (12)). On higher levels an additional similarity function S_{sp} needs to be defined that compares spatial patterns by taking the average similarity of temporal groups at corresponding positions (see Eq. (8)).

5.2. Learning temporal groups

On the lth layer and for two different neurons $i \neq j$ we define P_{ij}^l as the probability that at *some* node on the layer both neuron i and j were active during the time interval of length T. Cooccurrences in different locations do not count. Thus not only patterns following directly in time can be grouped together but also patterns which occur with a delay. These probabilities are turned into the adjacency matrix of a *Temporal Correlation Graph* as follows:

$$M_{ij}^l = \begin{cases} \dfrac{P_{ij}^l}{\max_i P_{ij}^l}, & \text{if } i \neq j \\ 1, & \text{if } i = j. \end{cases} \tag{6}$$

$\underline{M}^l$ is a symmetric matrix, which can easily be calculated by counting cooccurrences of active neurons (which have been stored in the activation stack) during one presentation of the whole training set. It is a sparse matrix because most codebook entries will never cooccur. The normalization by the row maximum reflects the idea that cooccurrences should be weighted independent of the prior probabilities of the neurons becoming active.

This step can only be done with the completed codebook and not during its creation, for nearest neighbor search in the codebook gives different results if it does not yet contain all patterns. Like the codebook $\mathbf{C}^l$ the Temporal Correlation Graph $\underline{M}^l$ is global for each level.

The entries of $\underline{M}^l$ can be interpreted as temporal similarities. This matrix is clustered hierarchically using spectral clustering, which is explained in more detail in Section 6. Two different stopping criteria are used for clustering. On lower levels of the network hierarchy the criterion is the size of each cluster. As long as it is above a parameter G the group is split up further. For temporal groups on the top level of the network hierarchy the matrix is partitioned into as many groups as categories need to be learned. This is one of three supervision steps that make learning not completely unsupervised. The second is emptying the activity stack between training images of different object categories. This is only done during learning, not in the recognition experiments. Third, each temporal neuron at the top level is assigned the name or the number of the category that it indicates.

Temporal groups of node positions converging on the same higher level node constitute new spatial patterns. Their global indices are written into one vector using some arbitrary but fixed order. These vectors need to be compared for learning higher level codebooks using vector quantization. Similarities of the vectors can be defined as average similarities of their components (which are temporal groups). Now a similarity measure must be devised for comparison of the groups. This can be reached using the temporal similarities stored in matrix $\underline{M}^l$. If one considers each similarity as an edge between two patterns and looks at two distinct clusters $\mathbf{t}_a^l$ and $\mathbf{t}_b^l$ there are 3 different sets of edges: edges within set $\mathbf{t}_a^l$, edges within set $\mathbf{t}_b^l$ and edges that connect patterns of $\mathbf{t}_a^l$ with elements of $\mathbf{t}_b^l$. (See Fig. 2.) The sum of edge weights in the last set is the *cut* between $\mathbf{t}_a^l$ and $\mathbf{t}_b^l$ (cut$(\mathbf{t}_a^l, \mathbf{t}_b^l)$, the sum of the weights of all medium blue edges in Fig. 2). The sum of edge weights in $\mathbf{t}_a^l$ (respectively $\mathbf{t}_b^l$) is called $V_r(\mathbf{t}_a^l)$ (resp. $V_r(\mathbf{t}_b^l)$) because it is the volume of $\mathbf{t}_a^l$ restricted to the set (the "normal" volume would be sum of edge weights of all vertices in $\mathbf{t}_a^l$ to any other vertex). In Fig. 2 these are the sums of the weights of all light blue respective all dark blue edges.

The cut is now normalized by the restricted volume of the union of both groups $\mathbf{t}_a^l$ and $\mathbf{t}_b^l$:

$$S_{\text{tg}}(\mathbf{t}_a^l, \mathbf{t}_b^l) = \frac{\text{cut}(\mathbf{t}_a^l, \mathbf{t}_b^l)}{V_r(\mathbf{t}_a^l \cup \mathbf{t}_b^l)}. \tag{7}$$

This leads to $S_{\text{tg}}(\mathbf{t}, \mathbf{t}) = 1$, which is required for a useful similarity measure.

Using similarity S_{tg} spatial patterns on higher levels can be compared exactly like image features by taking the average similarity of along all converging connections:

$$S_{\text{sp}}(\mathbf{s}_i^l, \mathbf{s}_j^l) = \frac{1}{C} \sum_{v=0}^{C-1} S_{\text{tg}}(\mathbf{s}_i^l(v), \mathbf{s}_j^l(v)). \tag{8}$$

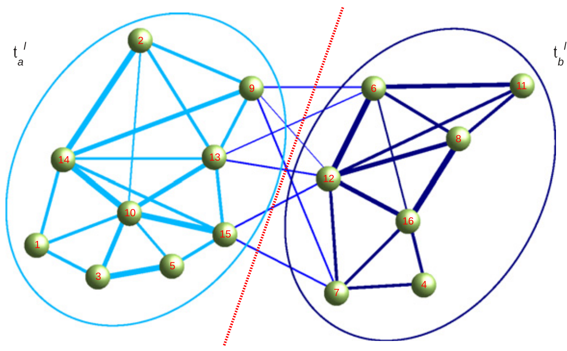


Fig. 2. Illustration of two different clusters. Line width of edges indicates similarity of elements. Light blue edges connect patterns in $\mathbf{t}_a^l$, dark blue edges elements in $\mathbf{t}_b^l$ and medium blue edges connect elements in $\mathbf{t}_a^l$ with elements in $\mathbf{t}_b^l$. The red line illustrates the cut which separates both clusters. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

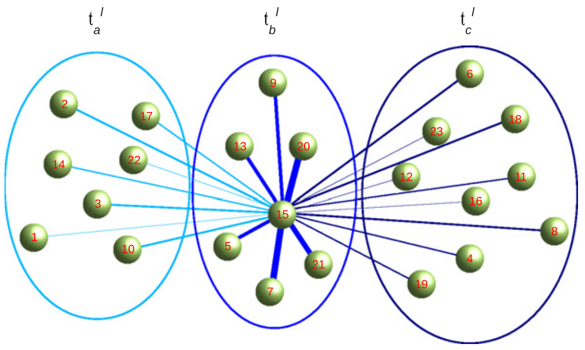


Fig. 3. Illustration of three different clusters. A membership value is computed for the central vertex using the drawn edges. Weights of all edges with the same color are added and divided by the sum of all edges irrespective of color. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The similarities of temporal groups are precomputed when the clustering is done, such that spatial patterns constructed from them can be compared very fast. Using this measure codebooks on higher levels are learned using the same quantization threshold ϑ_Q as on the lowest level.

S_{tg} also allows to define synaptic connection weights $W_{j,i}^{l,t;l+1,s}(v)$ between temporal neurons on one and spatial neurons on the next level. For each temporal neuron $\mathbf{t}_j^l$ its similarity to the temporal group $\mathbf{s}_i^{l+1}(v)$ at its corresponding position v within the spatial pattern $\mathbf{s}_i^{l+1}$ can be applied and be divided by the total number of groups C in that pattern:

$$W_{j,i}^{l,t;l+1,s}(v) = \frac{1}{C} S_{\text{tg}}(\mathbf{t}_j^l, \mathbf{s}_i^{l+1}(v)). \tag{9}$$

In this way any spatial pattern of the higher level can be activated by lower level neurons, even if it is only partly visible because of occlusion.

The only thing left to define are connection weights $W_{i,j}^{l,s;l,t}$ between spatial and temporal neurons on the same level. These determine how strongly a spatial pattern activates the neuron representing the temporal feature group it was put into and possibly also others. These weights are also defined using the temporal similarities stored in $\underline{M}^l$.

For definition of them temporal similarities of a spatial pattern to all patterns within a temporal group are added. In Fig. 3 these are the weights of all edges with the same color:

$$S_{\text{st}}(\mathbf{s}_i^l, \mathbf{t}_j^l) = \sum_{\{h|\mathbf{s}_h^l \in \mathbf{t}_j^l\}} \underline{M}(i,h)^l. \tag{10}$$

This sum should be highest for the group the pattern was put into since the aim of clustering is to find clusters with high similarities between their members. The N_G clusters which yield the biggest similarity sums are taken (three in Fig. 3) and the connection weight of the current spatial pattern (and hence neuron) to this group is defined as the similarity sum divided by the total similarity sum of all selected temporal groups (the sum of weights of all edges in Fig. 3 irrespective of color):

$$W_{i,j}^{l,s;l,t} = \frac{S_{\text{st}}(\mathbf{s}_i^l, \mathbf{t}_j^l)}{\sum_{h=0}^{N_G-1} S_{\text{st}}(\mathbf{s}_i^l, \mathbf{t}_h^l)}. \tag{11}$$

Hence any spatial pattern activates its own temporal group but also those to whose patterns it has a high similarity. Training is subsumed with pseudo-code in Algorithms 4, 5 and 6.

Algorithm 4: Training of the network

```
for $l \leftarrow 0$ to NoLevels-1 do
    foreach training image do
        calculate activities on trained levels;
        for $x \leftarrow 0$ to xSize[$l$]-1 do
            for $y \leftarrow 0$ to ySize[$l$]-1 do
                learn spatial patterns at $\mathbf{N}_{x,y}^{l,s}$;
            end
        end
    end
    foreach training image do
        calculate activities on trained levels;
        for $x \leftarrow 0$ to xSize[$l$]-1 do
            for $y \leftarrow 0$ to ySize[$l$]-1 do
                learn temporal transitions at $\mathbf{N}_{x,y}^{l,t}$;
            end
        end
    end
    normalize and cluster transition matrix on level $l$
end
```

Algorithm 5: Learning of spatial patterns at $\mathbf{N}_{x,y}^{l,s}$

```
if $l = 0$ then
    extract graph feast $\vec{J}$ at current position;
    $\mathbf{s}^l = \vec{J}$;
end
else
    foreach input node $\mathbf{N}_{x',y'}^{l-1,t}$ of $\mathbf{N}_{x,y}^{l,s}$ do
        get index $i$ of most active neuron;
        concatenate indices $i$ to a single vector $\mathbf{s}^l$;
    end
end
find nearest neighbor of $\mathbf{s}^l$ in $\mathbf{C}^l$;
if similarity to nearest neighbor $< \vartheta_Q$ then
    add $\mathbf{s}^l$ to $\mathbf{C}^l$;
end
```

6. Technical details

6.1. Low level features

As input features on the lowest level so called *parquet graphs* (Westphal & Würtz, 2009) are used. These are features of medium

Algorithm 6: Learning of temporal patterns at $\mathbf{N}_{x,y}^{l,t}$

```
get index $i$ of most active neuron at $\mathbf{N}_{x,y}^{l,s}$ in stack at time 0;
for $t \leftarrow 1$ to $T-1$ do
    get index $j$ of most active neuron at $\mathbf{N}_{x,y}^{l,s}$ in stack at time t;
    increment element $M_{ij}^{l}$ of transition matrix;
end
```

complexity built up of Gabor jets placed on a 3×3 grid. A Gabor jet (Lades et al., 1993) is a vector containing responses of Gabor filters to an image at the current image position. Gabor wavelets are plane waves multiplied with a Gaussian for dampening of their amplitude. They are used with 8 different orientations and 5 different scales yielding 40 complex values per image position, of which here only the absolute values are used. Gabor wavelets resemble the response properties of simple and (partly) complex cells in the primary visual cortex of mammals. They are not the only possible functions (derivatives of Gaussian also show similar behavior) but have been demonstrated to be well suitable as texture descriptors. Since each component is positive the normalized scalar product of two jets gives a value between 0 and 1 that is 1 for identical jets and can be seen as a similarity measure. By combining several of them at positions close-by in an image a bigger patch can be described. These parquet graphs can be compared by computing the average similarity of jets at the same position in the underlying grid. If segmentation masks are available for training images this can be incorporated into the parquet graphs by labeling each grid position inactive that was placed on the background during feature extraction. The corresponding Gabor jet then contains only (or better mostly) background information that is not used during comparisons. For exclusion of this information grid positions are only used for computing the average similarity of two parquet graphs if in both of them the jets at that positions are labeled active. Parquet graphs are compared by a similarity function, which averages the similarities of the jets contained in the parquet graph:

$$S_{\text{Image}}(F_1, F_2) = \frac{1}{|P_{\text{active}}|} \sum_{i \in P_{\text{active}}} \frac{\mathbf{J}_1(i) \cdot \mathbf{J}_2(i)}{\|\mathbf{J}_1(i)\| \, \|\mathbf{J}_2(i)\|}. \quad (12)$$

In Donatti, Lomp, and Würtz (2010) so called square graphs were used. These are similar to parquet graphs, the only difference is that they are constructed from 5 jets placed at the corners and in the center of a square region. These will also be used in one test in Section 7.

6.2. Clustering

On each level the matrix $\underline{M}$ is clustered using spectral clustering (von Luxburg, 2006). This is a clustering method that partitions a graph, in which each edge weight is the similarity of the connected vertices, into groups by "cutting" edges with small weights. While doing this a certain normalization criterion is maintained that prevents the method from creating too small clusters. Therefore the graph Laplacian $\underline{L_{\text{sym}}}$ is computed from matrix $\underline{M}$. Since matrix $\underline{M}$ is sparse $\underline{L_{\text{sym}}}$ is so, too. The solution to the clustering problem can now be formulated as minimum of a quadratic form of $\underline{L_{\text{sym}}}$ and an indicator vector for each cluster which has one discrete value in every component belonging to the cluster and another discrete value in all other components. Since finding an exact solution to this problem is an NP-complete task an approximate solution is obtained by computing eigenvectors of $\underline{L_{\text{sym}}}$. These provide a new representation of the input data in the eigenspace, where the clustering is much easier. For computing the eigenvectors the sparse eigenvalue problem library JADAMILU (Bollhöfer & Notay, 2007) is used. The actual clustering step is done using the K-Lines algorithm (Fischer & Poland, 2004). Clustering is applied in a hierarchical fashion here. First the complete matrix is split up into two partitions. Each new partition is further split up into two new subsets and the process is repeated until the stopping criterion is met.

6.3. Storage of neural activities and connections weights

Since the model contains a lot of neurons (possibly several thousand in each node) it is not feasible to store their activity all the time (especially after inhibition, when only few active neurons remain). Therefore, a special data structure was built which is basically a hash map with the index as key value and the activity as mapped value. In addition the map automatically deletes all entries whose activity becomes 0. When inhibition is applied the container for activities is sorted and resized to the number of kept activities and the mapping between hash values and activities is restored.

It is an intrinsic property of natural images that visual features are not placed arbitrarily within them. There is always an order. Therefore not each feature follows every other in a sequence of images showing a natural transformation of an object. Instead the pattern of temporal succession is very sparse. When the system is trained on one half of ETH80, e.g., a visual pattern on the lowest level is on average followed by ca. 42 others. This is approximately 0.08%. For higher levels this value increases to 0.17% and 2.36%, thus still stays small. This is why these successions can be stored in the sparse matrix $\underline{M}$ effectively. As a consequence also the connection weights between spatial patterns and their temporal groups are very sparse. Therefore, they are stored as vectors containing the index of the temporal group and the connection strength. This decreases memory demand and also computation time since no zero weights have to be traversed during computation of activities. Connection weights between temporal groups and spatial patterns on the next level are not stored at all but are computed on the fly when needed.

6.4. Parallelization

Computation can further be sped up by introducing parallelization using OpenMP. Activities in different nodes can then be computed in parallel for all nodes of the same level. This is possible because data is written only in distinct local data structures (the hash maps for the nodes). This is not possible for the learning routine, since here global containers as matrix $\underline{M}$ are accessed and threads would block each other.

7. Experiments

7.1. Determination of parameter values

Our system was tested using two standard databases for object recognition: the ETH80 (Leibe & Schiele, 2003) (in the cropped close *perimg* version), which contains images of 80 different objects belonging to 8 different categories (apple, car, cow, cup, dog, horse, pear, tomato). The images are taken on the upper viewing hemisphere and are parameterized using two viewing angles. 41 views are taken per object. The COIL100 (Nene, Nayar, & Murase, 1996) consists of images of 100 different objects, where each object is its own category. The images are taken at 72 viewing angles with constant latitude on the view sphere. In both databases a black background was used instead of additional segmentation information. All tests used the following scheme: the number of views per object was split into two groups, the first set of views of every object was used for training, the rest for testing. Most tests used a fifty–fifty partitioning, only the test of the generalization capabilities used different percentages for training and testing. Since the ETH80 contains an odd number of views, 21 views were used for training and 20 for testing, resulting in 1680, respectively 1600 images.

All images had a size of 128 × 128 pixels. A 3-layered network was used with 9 nodes in x- and y-direction on its lowest layer (so 81 in total), placed with a spacing of 14 pixels and an offset of 7 onto the input images. 9 nodes of a 3 × 3 block of the lowest level converged on the same upper level node, resulting in 3 nodes per direction on the intermediate level (9 in total). The same convergence led to a single node in the top level.

Training views were sorted according to their distance on the viewing hemisphere, computed from the viewing angles using the *Vincenty* formula for spheres (Vincenty, 1975) (formula 14–16). For COIL100 images the latitude was set to 45 degrees. Using this sorting according to viewing angle the system is expected to become invariant to the viewpoint of objects. If training images would show scale changes in a meaningful order the Temporal Correlation Graph would become invariant to object size and so forth.

For fifty–fifty partitioning every other view of the order was taken for training and the rest for testing, for other split-ups only every third or fourth and so forth.

The system has several parameters whose influence on recognition performance was tested in the following experiments. The parameter values may differ during training and recognition, as indicated by the indices "tr" (training) and "te" (test).

Recognition rates are given on basic level for ETH80 (apple, car, cow, cup, dog, horse, pear, tomato) and on name level for COIL100 (obj1, obj2 ..., obj100).

For an overview the relevant parameters are listed again:

1. Number of active neurons per node K: this parameter determines how many neurons are used for computing the input to the next layer/sublayer. It can be interpreted as lateral inhibition.
2. Number N_G of temporal groups that are activated by a spatial neuron: if N_G is bigger than 1 a spatial neuron not only activates the temporal group it was put into, but also others it has connections to.
3. Time Range T: this parameter determines how many past image activities are kept in the activation stack for learning and feedback calculation.
4. Similarity threshold ϑ_Q: determines the maximal distance of an input pattern to a codebook vector.
5. Maximal size of temporal groups G: determines how big a temporal group can be.

Several tests were conducted to find an appropriate set of parameter values for the ETH80. Then each of the parameters was changed to demonstrate its influence on the results and for showing the optimality of the chosen value. The optimal parameter set for the ETH80 is the following: $\vartheta_{Q\,\mathrm{tr}} = 0.92$ (not needed during recognition), $T_{\mathrm{tr}} = 20$, $T_{\mathrm{te}} = 2$, $K_{\mathrm{tr}} = 1$, $K_{\mathrm{te}} = 18$, $N_{G\mathrm{tr,te}} = \infty$ (all possible connections are used) and $G_{\mathrm{tr}} = 50$ (not needed during recognition). This gave a recognition rate of 99.06%. For COIL100 all parameters were kept except K_{te} which was set to 1, yielding a recognition rate of 100.00%. Since results were already optimal no more parameter tests were run on COIL100. Generalization capabilities and influence of G_{tr} were also tested on this dataset.

The first set of tests determined the influence of the time range parameter T during training and testing. During testing the system gets feedback from the last T_{te} images. When these are from the same category this is highly likely to have a positive effect on recognition rates. If the images show objects of different categories this additional information will have a negative impact on recognition rates. Thus each time the presented category changes the system gets useless feedback and the higher T_{te} the more images of the new category will be negatively affected. Hence T_{te} should not be too high during testing. Since during training activity values are deleted when a new category is presented it can be assumed that higher values for T_{tr} are desirable since they cannot lead to disturbing feedback but result in more relations between features that are recorded in the learning process. For testing these assumptions a set of experiments was performed on the fifty–fifty partitioning of ETH80. The system was trained with values of T_{tr} from 2 to 20, with an increase of 2 after each training. Each trained system was tested for recognition with values of T_{te} in the same range (and changed about the same amount) and with 3 different values of K_{te}: 2, 10 and 18. The best results were reached for the relatively high value of 20 for T_{tr}, whereas for T_{te} the lowest possible value 2 is optimal and K_{te} should be 18. These settings gave a recognition rate of 99.06%.

Table 1
Tests for K and N_G.

K	$N_G = 1$	$N_G = 2$	$N_G = 3$	$N_G = \infty$
1	84.69	85.25	84.88	84.88
2	88.31	89.50	89.12	89.44
3	90.06	91.50	91.25	91.31
4	91.12	93.19	93.25	92.50
5	92.44	93.19	93.81	94.38
6	93.31	93.94	94.75	95.06
7	93.56	94.75	94.88	95.12
8	94.38	95.19	95.31	95.62
9	95.19	95.38	95.69	95.62
10	95.31	95.94	95.81	95.94
11	95.38	96.81	96.38	96.88
12	95.88	96.38	96.12	97.25
13	95.94	96.88	96.44	97.38
14	96.00	96.75	97.25	98.25
15	96.06	97.06	97.44	98.62
16	96.50	97.31	97.56	98.81
17	96.56	97.44	97.69	98.38
18	96.81	97.94	97.69	**99.06**
19	97.06	97.94	97.94	98.44
20	97.12	98.06	98.38	98.56

The system behaves differently as a function of T_{tr} and T_{te} for different values of K_{te}. Whereas for $K_{\mathrm{te}} = 10$ and $K_{\mathrm{te}} = 18$ the recognition rates increase with increasing T_{tr} (which means it is good to record relations of temporally more distant features up to a certain distance) they decrease with increasing T_{te}. This can be interpreted as harmful effects of previous images of different categories which affect more subsequent images if T_{te} gets higher. For $K_{\mathrm{te}} = 2$ recognition rates also increase with T_{tr} but do not decrease with higher T_{te}. Obviously unfavorable feedback inputs have little effect when the number of hypotheses is reduced to 2 after collection of feedforward input to a node. Only the most active neurons (or hypotheses), which fit best to the input data get support via feedback calculations and no alternative hypotheses fitting the preceding object stay active.

The first tests used a value of N_G of ∞, which means that every spatial neuron actives all temporal groups (or neurons) it has connections to. To show the optimality of this setting a further set of experiments was executed with optimal T_{tr} and T_{te}. For this K_{te} was changed from 1 to 20 and N_G set to the values 1,2,3 and ∞. Table 1 summarizes all results. K_{te} has the biggest influence. A value of 18 is optimal in combination with the correct N_G value. N_G's impact on the recognition rate is weaker than K_{te}'s. For values of $K_{\mathrm{te}} < 5$ it is sometimes better to use only a few possible connections to temporal groups, for all higher values of K_{te} (except 9) best results are reached by using all connections.

The third test was for determination of the similarity threshold ϑ_Q. This has a very strong effect, see Table 2.

Table 2
Tests for ϑ_Q.

ϑ_Q	0.75	0.80	0.82	0.84	0.86	0.88	0.90	0.92	0.94	0.96	0.98	1.00
RR	14.00	21.69	23.81	24.44	47.06	61.62	77.06	**99.06**	98.94	98.50	97.31	94.06

Table 3
Tests for G.

Database	$G = 25$	$G = 50$	$G = 100$	$G = 250$	$G = 500$	$G = 1000$
ETH80	98.81	**99.06**	87.38	25.12	25.00	56.94
COIL100	100.00	**100.00**	99.42	98.56	98.22	99.47

On the one hand the system makes only very rough distinctions between spatial patterns if the threshold is 0.75 (the mean similarity of high-dimensional random Gabor jets). Almost all possible parquet graphs will be assigned to the same pattern in the codebook (which will contain very few patterns). Consequently, no distinction can be made on the lowest level and hence also on the higher levels. On the other hand setting the threshold very high (for example to 1.0) increases the demand for computing resources but does not necessarily bring the highest recognition rates. Since for testing the system has to generalize over patterns of the training set the recognition performance depends on the distribution of patterns in the test set compared to the distribution of the training set. For covering the one distribution with the other not all training patterns are required, and overfitting will occur for too high values of ϑ_Q. The test shows that 0.92 is the best value for ϑ_Q.

The last parameter is the maximal size of temporal groups G. The bigger G the bigger the temporal groups and the less distinctive they will be, resulting in groups that represent (parts of) different categories. Small temporal groups will have less ability to generalize over different instances of the same object.

Table 3 shows that $G = 50$ yields the best results for ETH80. For COIL100 $G = 25$ works just as well. Overall it is justified to chose 50 as maximal group size. COIL100 is not very sensitive to this parameter at all whereas recognition rates for ETH80 drop enormously for $G > 100$. A look at the confusion matrices of the ETH80 tests show that with $G = 250$ the majority of votes falls into the categories tomato, pear and cups. For $G = 500$ tomatoes and cars collect the most votes and with $G = 1000$ most votes fall into the car category. So with increasing G groups and patterns assigned to cars get more attractive, otherwise tomatoes have a bigger influence. Due to this shift fruits get distinguished relatively well and recognition rates increase again.

For comparison of the different visual features the standard test with optimal parameters and (roughly) half–half partitioning was also conducted using the square graph features from Donatti et al. (2010). The achieved recognition rates were 99.00% for ETH80 and 99.89% for COIL100. The difference to the parquet graph features is 0.06% in the first dataset and 0.11% in the second, so the shape of the parquet graphs has little influence.

7.2. Speed improvements

We then looked at the effect of parallelization via OpenMP on both datasets. Computations were done on an AMD Socket AM3 Phenom II X6 1090T CPU with 6 cores at 3200 MHz. For additional speedup we also applied weight pruning, i.e., discarding all weights below ϑ_P. All tests used the fifty–fifty partitioning of ETH80 and COIL100. As Table 4 shows, small weights have minor impact on recognition performance and can be ignored. Columns show the value of ϑ_P, the achieved recognition rate, the computation time in seconds (averaged over 4 runs) using a single core, and the time in seconds for parallel execution with 6 threads.

As can be seen in Table 4 computation time is reduced considerably whereas the drop in recognition rates is surprisingly small. Even for the highest value of ϑ_P the absolute number is only 24 of 1600 images less that have been correctly classified in the more sensitive database ETH80 compared to no pruning. For a pruning threshold of 0.001, which already reduces computation time a lot, the drop in correctly recognized images corresponds to only 4 images. In COIL100 the biggest decline corresponds to only 3 images of 3600 that had to be classified. Computation for ETH80 is approximately 22.42 times faster using pruning with a threshold of 0.1 and 29.96 times faster using parallelization additionally. For COIL100 the factor for pruning is 15.73 and including parallelization it is close to 22.02. Thus pruning is a very valuable tool in speeding up the system. A value of 0.1 for ϑ_P seems a suitable choice since it reduces computation time a lot but decreases recognition rates only slightly (even on ETH80 only by 1.5%).

Table 4
Tests for ϑ_P and parallelization.

ϑ_P	RR	Time serial (s)	Time parallel (s)
ETH80			
0	99.06	21 796	8 500
0.001	98.81	3 815	1 844
0.01	98.38	1 205	811
0.1	97.56	972	728
COIL100			
0	100.00	46 610	20 269
0.001	99.97	5 420	3 292
0.01	99.92	3 081	2 178
0.1	99.97	2 963	2 116

Parallelization also reduces the computation time, although the gain decreases with stronger pruning. The less computations need to be done per network node the less time can be saved by executing them in parallel. The test computer did not use 100% of CPU power during parallelized tests, therefore no gains should be expected with more cores. The minimal time needed on average per image was 0.455 s for ETH80 and 0.588 s for COIL100.

The nearest neighbor search in these experiments was done using Intel's Math Kernel Library (INTEL, 2007) for fast computation of the product between all query parquet graphs of one image in one matrix and the codebook stored in a second matrix. This can also be sped up a lot at the cost of slightly decreased accuracy using hashing methods (full details at (Lessmann & Würtz, 2012a)).

7.3. Generalization

Further tests show the generalization capabilities of the system for ETH80 and COIL100. The system was trained on 2.5%, 5%, 10%–90% of all images in the database and recognition was tested on the remaining images using the two optimal parameter sets. The first column of Table 5 shows the percentage of all images of each database that the system was requested to use for training, the second column the actual number of obtained images, the third the real percentage of used images and the last column the recognition rate achieved on the remaining images.

Table 5
Tests for generalization.

% requested	# obtained	% obtained	Recog. rate
ETH80			
2.50	160	4.88	27.44
5.00	240	7.32	25.95
10.00	400	12.20	31.70
20.00	720	21.95	87.89
30.00	1040	31.71	90.98
40.00	1360	41.46	98.12
50.00	1680	51.22	99.06
60.00	2000	60.98	99.30
70.00	2320	70.73	99.17
80.00	2640	80.49	95.47
90.00	2960	90.24	96.25
COIL100			
2.50	200	2.78	58.00
5.00	400	5.56	74.91
10.00	800	11.11	88.89
20.00	1500	20.83	97.46
30.00	2200	30.56	99.18
40.00	2900	40.28	99.63
50.00	3600	50.00	100.00
60.00	4300	59.72	99.93
70.00	5100	70.83	99.86
80.00	5800	80.56	99.93
90.00	6500	90.28	99.71

Table 6
Tests for codebook and feedback usage.

Database	Use codebook	No feedback
ETH80	83.25	83.75
COIL100	99.50	99.64

The results in Table 5 demonstrate very good generalization. Even with only 20% of the data recognition rates of almost 88% and 97% can be reached. 50% is definitely enough to capture the correct labels. The results show a peak of performance around 50% training data. The reason for this is that parameter tests were done using a fifty–fifty partitioning and parameter values are hence optimal for this amount of training data. With further tuning the results on higher amounts of training data can be improved. The system shows better results on high percentages of training data if a higher similarity threshold of 0.94 is used.

Next we conducted the standard test with both databases without feedback and with feedback but using nearest neighbor search in the higher level codebooks for activation of spatial patterns instead of using the neural connections (Table 6).

The outcome for the ETH80 reveals the advantage of using both feedback and neural connections, since both improve recognition performance considerably. The positive effects of feedback by resolving ambiguities have been described before, the neural mechanism causes more spatial patterns to become active and offers the system more valuable hypotheses to choose from. For COIL100 the effects of feedback and neural-like computation are nearly negligible. Since computation was done with $K = 1$ there are no multiple hypotheses between which feedback can make a decision. The reason that higher values than 1 for K decrease recognition rates might be that COIL100 only depends on one viewing angle and has more diverse objects. This probably has the effect that clearly separated temporal groups are learned and most additional hypotheses only add noise. Since the codebook activates only one possible hypothesis it probably has a similar effect as using the neural connections with $K = 1$.

A further test was run using unrestricted feedback, which enhances the activity of each neuron receiving feedback, irrespective of its previous activity. The result is that one object category stays active all the time, which means recognition rates of 12.5% and 1% for ETH80 and COIL100, respectively. This can be seen as overexcitation and also fits well to the finding that feedback connections to the neocortex are modulatory but not driving (only active neurons are modified). Otherwise strong feedback loops could result in chaotic oscillations of neural activity (Koch, 2004). The same happens when no inhibition is applied during inference and all neurons in each node stay active if ever activated. It can be assumed that neural activity in certain brain regions would degenerate if no inhibitory influences would operate.

Table 7
Architecture test on ETH80.

%	Arch-1	Arch-2	Arch-3	Arch-4	Arch-Def
2.5	**56.19**	53.65	49.65	41.67	48.43
5.0	70.03	**73.45**	61.09	49.05	62.40
10.0	80.17	**81.25**	71.32	47.43	71.70
20.0	90.12	90.86	92.30	90.59	**94.53**
30.0	91.96	94.55	96.21	94.78	**98.48**
40.0	91.88	94.48	94.84	87.34	**98.23**
50.0	90.38	94.88	94.94	87.50	**98.75**
60.0	92.27	94.53	97.81	88.59	**98.67**
70.0	95.42	98.33	97.81	92.29	**99.38**
80.0	94.38	96.09	97.81	92.97	**97.97**
90.0	96.56	96.88	97.19	96.56	**98.44**

7.4. *Evaluation of the basic principles*

We conducted additional tests to show the validity of the three basic principles introduced at the beginning: learning of temporal groups, hierarchical network structure and feedback. For checking the influence of network architecture and the number of levels the generalization test for ETH80 and COIL100 was performed with different networks, all with a pruning threshold $\vartheta_P = 0.01$. For demonstrating the general capabilities of the system the optimal K_{te} was determined for each test, whereas $T_{\text{tr}} = 20$ and $T_{\text{te}} = 2$ throughout. The tests were also repeated using the default-3-level-architecture used before. The architectures were all the same on the lowest level, containing 8×8 nodes placed with a distance of 16 pixel in x- and y-direction on the input image and using an offset of 8 pixels in both dimensions. Based on this 4 different settings were employed, one utilizing 2 levels, two different with 3-levels and a last one using 4 levels (in the following explanation 4×4 nodes on level 1 for example means 16 nodes on the middle spatial layer and 16 nodes in the temporal layer):

1. Arch-1: The 2-level-architecture, all 64 nodes of the lowest level converged on one top-level-node.
2. Arch-2: The first 3-level-network, 4×4 nodes of the lowest level converged on one node in the middle level of 2×2. All nodes of this level converged then on the same top-level-node.
3. Arch-3: The second 3-level-setting, 2×2 nodes of level 0 converged on the same node of the next level, resulting in a middle level of size 4×4. These 16 nodes then converged on the same top-level node.
4. Arch-4: The 4-level-architecture, on each level 2×2-nodes converged on one node of the next level. This resulted in 4×4 nodes on level 1, 2×2 on level 2 and 1 on the top level.
5. Arch-Def: The default architecture with 9×9 nodes on level 0, 3×3 on level 1 and 1 node on level 2.

Table 7 shows the results. Each column corresponds to one of the architectures, each row to the size of the training set.

The number of levels and the number of nodes per level determine the model complexity together with the neuron numbers. For higher complexity more parameters need to be learned and more training data is needed. Hence for low amounts of training data less complex systems perform better whereas with increasing data the more complex systems become better. This can

Table 8
Architecture test on COIL100.

%	Arch-1	Arch-2	Arch-3	Arch-4	Arch-Def
2.5	**60.69**	58.83	60.29	55.59	59.24
5.0	75.63	74.90	**77.22**	68.10	75.47
10.0	90.27	89.86	**91.92**	81.20	88.48
20.0	97.91	97.63	**98.46**	95.58	97.60
30.0	99.58	**99.64**	99.48	98.58	99.16
40.0	99.88	99.86	**99.91**	99.35	99.70
50.0	99.97	**100.00**	99.94	98.83	99.75
60.0	99.83	99.76	**99.97**	99.38	99.86
70.0	**100.00**	**100.00**	**100.00**	98.95	**100.00**
80.0	**100.00**	**100.00**	**100.00**	99.57	99.93
90.0	**100.00**	**100.00**	**100.00**	98.71	**100.00**

Table 9
Architecture test on ETH80 without feedback.

%	Arch-1	Arch-2	Arch-3	Arch-4	Arch-Def
2.5	**49.13**	47.31	46.60	36.76	43.40
5.0	**62.70**	61.12	59.21	45.49	58.82
10.0	**68.16**	67.19	65.73	44.62	63.85
20.0	**81.25**	78.63	80.00	77.38	79.14
30.0	**83.26**	81.56	81.88	80.49	81.25
40.0	**84.17**	81.61	82.34	79.27	82.86
50.0	**83.25**	80.62	82.19	80.06	82.81
60.0	**84.30**	81.80	83.44	82.03	83.36
70.0	**88.44**	86.46	85.83	82.81	86.25
80.0	88.12	87.19	**88.91**	84.53	87.03
90.0	**92.19**	90.31	90.94	90.62	90.00

be observed in the results above. For 2.5% of data used for learning the 2-level-system performs best, considerably better than the 4-level-system, also notably better than the default system and Arch-3 but only a little bit better than Arch-2. The reason is that Arch-2 with 4 nodes in the middle level is less complex than Arch-3 with its 16 middle-level-nodes. For training set sizes of 5% and 10% Arch-1 is too simple and Arch-2 becomes the best performing system. For higher amounts of training data Arch-3 should be more appropriate and indeed performs better than Arch-1 and Arch-2 (except for one outlier). Nevertheless, it is still not as good as Arch-Def. This can be explained by a relatively similar complexity but a bigger input layer that samples the input images more densely and hence increases the training data further. The 4-layer-architecture is too complex for all cases, leading to very poor results for few training data and good but not optimal performance for many training images.

It can be expected that systems with at least 3 levels have further advantages over 2-level-systems when training objects are not rigid and contain subparts that can move relative to each other more freely. Then the data shows more hierarchical organization and the greater flexibility of the system can make use of it.

The systems were also tested on COIL100, leading to results presented in Table 8. Here also the least complex system performs best for the least amount of training images and the 4-level-system is always the worst (but not much worse for high amounts of data). The distinction between the 3-level-systems is less clear, but since this database is easy to classify the differences in the results are negligible.

For demonstrating the positive influence of feedback from earlier images we repeated the test on ETH80 without feedback (the influence on COIL100 can be expected to be much less pronounced since for COIL100 K_{te} is always 1, and feedback cannot distinguish between different hypotheses). The results are shown in Table 9.

It can clearly be seen that feedback leads to an improvement, for Arch-Def up to nearly 20% points, although this difference becomes smaller for higher amounts of training data. Another observation is that missing feedback favors less complex systems. In almost all tests without feedback Arch-1 gives the best results.

Table 10
Training list shuffling test on ETH80.

Shuffle on level	RR	σ	K_{te}
No	98.06	0.0	52
Identity	81.25	6.32	46
Basic	86.56	6.65	44
Abstract	50.63	5.88	32
Complete	20.45	5.26	12

Finally, we have tested how important it is for the system to see images in meaningful temporal order during learning of temporal groups, or, how strongly is learning affected if images are sorted randomly and not according to viewing angle? For this the fifty–fifty partitioning test on ETH80 was conducted with training image lists shuffled randomly on different levels. The control experiment contained all images in the usual order, sorted by basic category, then identity, and finally viewing angle.

In further tests training images were shuffled randomly on different levels. First all images of the same identity were shuffled, destroying sorting by viewing angle but keeping sorting by identity and categories. Then images were shuffled on the level of basic category, such that, e.g., images of different cows followed each other randomly, but basic categories still stayed separated. In the next tests basic categories of the same abstract category were mixed. As a consequence images of apples, pears and tomatoes could follow each other (abstract category “fruits and vegetables”) as well as images of cows, dogs and horses (abstract category “animals”). For the two other abstract categories “human-made,small (graspable)” and “human-made,big (vehicle)” this has no influence, since each of them consists of only one basic category (cups respectively cars). The last tests were done using completely randomly shuffled lists.

Each test was done 10 times with different random seeds and average recognition rates were computed for the optimal value of K_{te} (except in case of the control experiment with the non-shuffled list).

In the earlier tests activities of the previous T_{tr} training images were deleted when a new object category was encountered during training. This is not possible here, since when training on a fully shuffled list each image can contain a new category and hence activities would be deleted after each image and nothing could be learned. Hence deletion of old activities had to be dropped and a new optimal value for T_{tr} was determined for optimal results on the non-shuffled list. This was found to be $T_{tr} = 35$ and gave a recognition rate of 98.06%. That this is only 1% less than the best result using deletion of activities shows that the system does not critically depend on this weakly supervised aspect of training. The pruning threshold was $\vartheta_P = 0.01$ during testing. Table 10 shows the results.

As expected recognition rates decrease with shuffling. This shows that order of training images is very important for the training to work. What surprises a little bit is that shuffling on basic category level is less harmful than on identity level (although both mean recognition rates are about one standard deviation of each other). A possible explanation is that viewing angle of consecutive images is more important than object identity. If images are shuffled on identity level the sorting according to viewing angles is definitively destroyed. In the case of shuffling on basic category level it may lead to (at least partly) advantageous succession of viewing angles even if the identity of objects differs.

7.5. Comparison with other methods

The first system we compare our with is the HTM by George and Hawkins. Their company Numenta offers the “vision toolkit”. This uses also Gabor functions for extracting features from input images which are scaled to size 200 × 200 pixels. Then it trains a network

Table 11
Comparison TCG and HTM on ETH80.

%	TCG	TCG w/o feedback	HTM
2.5	48.43	49.13	60.20
5.0	62.40	62.70	67.00
10.0	71.70	68.16	69.90
20.0	94.53	81.25	86.00
30.0	98.48	83.26	89.10
40.0	98.23	84.17	90.80
50.0	98.75	83.25	92.10
60.0	98.67	84.30	93.20
70.0	99.38	88.44	94.00
80.0	97.97	88.12	94.40
90.0	98.44	92.19	96.30

on a given training dataset and performs a recognition test on a given test set. The network comes pretrained on the lowest level on several thousand natural images, thus that only higher levels of the network need to be trained, which makes training really fast. The software offers an optimization-option, which tests several different network architectures and parameter values on training and testing data for optimizing the recognition rates. This option was used for the following results. Unfortunately we were not able to find out two important information about the system. First of all we do not know if and when how training images are sorted. The documentation does not provide information about this and also a request per email did not result in an answer. Maybe they are sorted according to their similarity, maybe they are just used in the order in which they are added to the training set or in alphabetical order. The second unclear point is if time based inference is used and thus activities of past images influence the classification of the current test image per feedback. Contradicting information was found about this aspect. The following Table 11 shows the result of the HTM in comparison to our default architecture using feedback and the 2-level-architecture using no feedback.

For the two lowest percentages of training data the HTM outperforms both results of the Temporal Correlation Graph. For higher amounts this changes. Whereas the HTM is always better than TCG without feedback (although the difference decreases with increasing training data) it is outperformed by the TCG using feedback. Because of the missing information about the HTM it is hard to judge the results. Would the HTM using the same order of training images perform better? Could it in principle use feedback to be better which was not included in the used version? This cannot be answered here, but it seems fair to say that both systems show comparable behavior on the test data. What the HTM may not be suited for is to handle larger numbers of categories. The vision toolkit is restricted to a maximum number of 50 categories. For this reason the systems could not be compared on the COIL100 dataset. The Temporal Correlation Graph scales well with the number of categories. For demonstrating this the Amsterdam Library of Object Images ALOI (Geusebroek, Burghouts, & Smeulders, 2005) was used. It contains 1000 different objects in COIL-style (72 viewing angles per object) and hence contains 72 000 images in total. A second part of ALOI contains the same 1000 objects with the same viewing angle but 24 different illumination directions, resulting in 24 000 images. On both data sets a fifty–fifty partitioning test was performed with one half of viewing angles respectively illumination conditions per object in the training set and the rest in the testing set. The default architecture was used together with the following parameter values: $T_{\mathrm{tr}} = 20$, $T_{\mathrm{te}} = 2$, $K_{\mathrm{tr}} = 1$, $K_{\mathrm{te}} = 1$, $\vartheta_Q = 0.92$, $G = 200$, $N_G = \infty$ and $\vartheta_P = 0.01$. On the standard dataset a recognition rate of 99.57% was reached and on the illumination set a rate of 99.37%. This shows that the Temporal Correlation Graph is able to apply its classification capabilities without any restrictions to data sets containing a great many categories (and to other transformations than rotation in depth).

Table 12
Test results on ALOI1000.

Method	RR (viewpoint)	RR (illumination direction)
HMAX	80.76	83.13
SalBayes	89.71	75.50
SIFT	70.95	71.47
TCG	**98.06**	**98.73**

It is hard to find test results of other object recognition systems that use all objects of the ALOI1000. Some results could be found in Elazary and Itti (2010) introducing the *SalBayes* system. It employs several features maps (42 in the implementation) which are based on center–surround responses of 7 different features (intensity, opponent color and orientations of edges) on different scales. Resulting maps are processed by a normalization operator that promotes maps in which the global maximum of feature values clearly stands out in comparison to the other local maxima. The feature value at the global maximum position in the *i*th normalized map constitutes the *i*th component of a 42-dimensional feature vector extracted for each training image. Using several training images a Gaussian is fitted to the distribution of feature vector components over each object category by determining mean value and variance. These parameters can then be used to assign feature vectors of the same kind extracted on test images (unnormalized) class probabilities using Bayes' theorem which are then used for classification. For the tests presented there only 25% of images were used for training. Because of this we have repeated our tests using only one quarter of images for training. Table 12 shows the results of the TCG ($K = 2$ for illumination direction) and the comparison values from Elazary and Itti (2010).

As can be seen the Temporal Correlation Graph outperforms the other approaches clearly. Whereas on the viewpoint set the distance to the second best (SalBayes) is already 8.35% on the illumination direction set the difference to the second best (HMAX) gets 15.60%. In this test the TCG's learning of the transformation present in the training data seems to be clearly advantageous compared to scale- and shift-invariance as built into HMAX and the relying on extraction of invariant features at salient points as in SIFT and SalBayes.

Table 13 subsumes results for the generalization test on the COIL100 of our system and two others. For comparison we took the system of Westphal (Westphal & Würtz, 2009), which uses the same features as ours, and the system from Linde and Lindeberg (2004). For this the best results that we could find are reported. It differs considerably in its method from the proposed system, for example it uses just one global histogram per image as features. These histograms are classified by using nearest neighbor search on the χ-measure or using an SVM. The results shown here have been obtained using an SVM and 14-dimensional histograms. It has to be noted that results of Westphal have been obtained using 5-fold cross validation and not a single partitioning into training and test set as for both other systems. The first column shows the distance in viewing angle of two consecutive images in the training set. The second through fourth columns show the obtained recognition rates.

In Linde and Lindeberg (2012) the results of Linde and Lindberg for 30° were improved to 100.00% and for 60° to 99.00%. The comparison reveals that our system outperforms the approach of Westphal. Nevertheless it cannot fully compete with the system of Linde and Lindeberg for very sparse training sets. Whereas they reach recognition rates of over 97% for 90° distance between training images our system drops to 70.35%. However, it needs to be considered that they include several different features and information channels as color information for example, which our system does not use. It uses only Gabor responses to grayscale images.

Table 13
Tests for generalization on COIL100.

Dis.	Proposed system	Westphal	Linde/Lindeberg
10°	**100.00**	99.68	**100.00**
20°	98.91	97.97	**99.96**
30°	95.87	92.93	**99.88**
40°	**93.05**	88.45	–
45°	88.61	–	**99.37**
50°	**88.89**	83.20	–
60°	79.02	76.61	**97.99**
70°	**81.44**	75.79	–
80°	**77.28**	72.39	–
90°	70.38	65.63	**97.13**

Table 14
Leave-one-out-cross-validation on ETH80.

cat.	TCG	Donatti	Westphal	Leibe	Suard
apple	97.32	70.98	91.22	91.46	**98.29**
car	**100.00**	98.54	80.98	99.51	99.76
cow	86.59	59.02	49.76	**86.83**	84.15
cup	**100.00**	91.22	98.05	**100.00**	**100.00**
dog	**92.20**	60.73	35.85	83.66	86.10
horse	**90.73**	50.73	57.80	84.88	86.34
pear	89.76	95.61	87.80	99.51	**100.00**
tomato	97.07	74.63	95.12	**98.29**	**98.29**
all	**94.21**	75.18	74.56	93.02	94.12

At last leave-one-object-out-cross-validation was performed on the ETH80, where the system was trained on all images except that of one object identity and then was tested on all left out images. This was done for all 80 different object identities and then the average recognition rate was determined. For better comparability the unsegmented color version of the ETH80 was used. Quantization threshold ϑ_Q was set to 0.94 for this test and K set to 38, the other parameter values stayed the same. The Temporal Correlation Graph was compared with the systems of Donatti (Donatti et al., 2010), Westphal (Westphal & Würtz, 2009), Leibe (Leibe & Schiele, 2003) and Suard (Suard, Rakotomamonjy, & Bensrhair, 2006). Results can be seen in Table 14.

Whereas the systems of Westphal uses the same features as our system and Donatti the very similar square graph features, the results from Leibe and Schiele are the best they obtained when introducing the ETH80 using an optimal multi-cue decision tree incorporating color, texture and contour features. Suard et al. report the best results we could find. They use a labeled graph, which describes the contour of an object and histograms of oriented gradients that describe the texture of image regions. These are combined by multiplying suitable kernel functions, which are then used for classification by an SVM. In comparison to our method this one utilizes two different cues and learns fully supervised since an SVM needs class labels. Additionally, segmentation masks are needed for building contour graphs. Table 14 shows that among these systems ours gets the bests results in 4 categories and has the best overall performance. For apples, cows and tomatoes it is slightly worse than the decision tree from Leibe and Suard's approach, for pears the difference to them is higher. Because of the advance in the difficult categories "dog" and "horse" our system leads with respect to the total recognition rate nevertheless. The overall best result for a leave-one-out-cross-validation on ETH80 we could find are those from Linde and Lindeberg (2012), where a total recognition rate of 97.7% was reached using a 25-dimensional histogram for each image computed from partial Gaussian derivatives on different scales. This exceeds our result by ca. 3.5% but was not performed on the full ETH80 dataset but with a subset of 1280 images including only viewpoints on the equator of the view sphere (comparable to viewpoint selection in COIL100) with blackened backgrounds. It is hard to judge the difficulty of this changed test setting. On the one hand the smaller number of images leads to smaller training and test sets for each identity, which can be assumed to increase difficulty. Additionally the authors mention as reason for blackening the background discarding of unintended background cues which simplify recognition. On the other hand by restricting views to constant latitude one dimension of transformation the used recognition system should be invariant to is completely removed. It is reasonable to assume that this is one of the reasons why COIL100 is easier to classify than the complete ETH80 and hence this viewpoint selection would also facilitate leave-one-out-cross-validation on ETH80. A fair comparison could only be done if our system was tested on the reduced ETH80 or the system from Linde and Lindeberg (2012) on the full database.

8. Conclusion and future work

We have presented an object recognition system capable of object recognition on standard benchmark data sets. It generalizes well over different views of the same object, but different identities of the same object category remain difficult. A major goal of future work is to make the system more neural network like by using a biologically plausible local learning rule for determining the synaptic connection weights.

Acknowledgments

We thank the editor and reviewers for their comments, which improved the paper considerably. We gratefully acknowledge funding from the DFG in the priority program "Organic Computing" (WU 314/5-3) and from the land of Northrhine–Westphalia in the project *MoGES*, which is co-financed by the EFRE program from the European Commission. We thank Inga Schumacher for help with the graphics. Parts of this paper have been published as extended abstract in Lessmann and Würtz (2012b).

References

Bergstra, J., & Bengio, Y. (2009). Slow, decorrelated features for pretraining complex cell-like networks. In *Proc. NIPS* (pp. 99–107).

Bollhöfer, M., & Notay, Y. (2007). Jadamilu: a software code for computing selected eigenvalues of large sparse symmetric matrices. *Computer Physics Communications*, *177*, 951–964.

Bradski, G., & Grossberg, S. (1995). Fast-learning VIEWNET architectures for recognizing three-dimensional objects from multiple two-dimensional views. *Neural Networks*, *8*(7–8), 1053–1080.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., & Bray, C. (2004). Visual categorization with bags of keypoints. In *Proc. ECCV workshop on statistical learning in computer vision*.

Donatti, G. S., Lomp, O., & Würtz, R. P. (2010). Evolutionary optimization of growing neural gas parameters for object categorization and recognition. In *Proc. IJCNN* (pp. 1862–1869). Los Alamitos, CA: IEEE Computer Society.

Elazary, L., & Itti, L. (2010). A Bayesian model for efficient visual search and recognition. *Vision Research*, *50*(14), 1338–1352.

Fischer, I., & Poland, J. (2004). *New methods for spectral clustering. Tech. rep. 12. IDSIA. Instituto Dalle Molle di Studi sull'Intelligenza Artificiale. Galleria 2, 6928 Manno-Lugano, Switzerland*.

Franzius, M., Wilbert, N., & Wiskott, L. (2011). Invariant object recognition and pose estimation with slow feature analysis. *Neural Computation*, *23*(9), 2289–2323.

George, D. (2008). How the brain might work: a hierarchical and temporal model for learning and recognition. *Ph.D. Thesis*. Stanford University Stanford. Stanford, CA, USA, aAI3313576.

George, D., & Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, *5*(10).

Geusebroek, J., Burghouts, G., & Smeulders, A. (2005). The Amsterdam Library of Object Images. *International Journal of Computer Vision*, *61*(1), 103–112.

Hawkins, J., & Blakeslee, S. (2004). *On intelligence*. Times Books.

INTEL (2007). MKL—Intel Math Kernel Library. URL: http://software.intel.com/en-us/intel-mkl/.

Koch, C. (2004). *The quest for consciousness: a neurobiological approach*. Roberts and Co.

Lades, M., Vorbrüggen, J. C., Buhmann, J., Lange, J., von der Malsburg, C., Würtz, R. P., et al. (1993). Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, *42*(3), 300–311.

Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In *Proc. CVPR* (pp. 2169–2178).

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*, 541–551.

Leibe, B., & Schiele, B. (2003). Analyzing appearance and contour based methods for object categorization. In *Proc. CVPR (2)* (pp. 409–415).

Lessmann, M., & Würtz, R. P. (2012a). Fast nearest neighbor search in pseudosemimetric spaces. In G. Csurka, & J. Braz (Eds.), *Proc. VISAPP Rome. Vol. 1* (pp. 667–674). SciTePress.

Lessmann, M., & Würtz, R. P. (2012b). Learning of invariant object recognition in a hierarchical network. In Hammer, B., Villmann, T. (Eds.), *Proceedings of new challenges in neural computation, Graz. No. 03/2012 in machine learning reports* (pp. 104–112).

Linde, O., & Lindeberg, T. (2004). Object recognition using composed receptive field histograms of higher dimensionality. In *Proc. ICPR*, ICPR'04. (pp. 1–6). IEEE Computer Society.

Linde, O., & Lindeberg, T. (2012). Composed complex-cue histograms: an investigation of the information content in receptive field based image descriptors for object recognition. *Computer Vision and Image Understanding*, *116*(4), 538–560.

Lindeberg, T. (2013). Invariance of visual operations at the level of receptive fields. *PLoS One*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Luciw, M. D., Weng, J., & Zeng, S. (2008). Motor initiated expectation through top-down connections as abstract context in a physical world. In *Proc. international conference on development and learning (ICDL)* (pp. 115–120). IEEE.

Mobahi, H., Collobert, R., & Weston, J. (2009). Deep learning from temporal coherence in video. In *International conference on machine learning* (pp. 93–744).

Mountcastle, V. (1997). The columnar organization of the neocortex. *Brain*, *120*(4), 701–722.

Nene, S. A., Nayar, S. K., & Murase, H. (1996). *Columbia Object Image Library (COIL-100). Tech. rep. Department of Computer Science. Columbia University*.

Rolls, E. (2008). *Memory, attention, and decision-making: a unifying computational neuroscience approach*. Oxford University Press.

Serre, T. (2006). Learning a dictionary of shape-components in visual cortex: comparison with neurons, humans, and machines. *Ph.D. Thesis*. Massachusetts Inst. of Technology, Cambridge. URL: http://cbcl.mit.edu/projects/cbcl/publications/ps/MIT-CSAIL-TR-2006-028.pdf.

Suard, F., Rakotomamonjy, A., & Bensrhair, A. (2006). Object categorization using kernels combining graphs and histograms of gradients. In *LNCS*, *Proc. ICIAR* (pp. 23–34). Springer.

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, *22*(176), 88–93.

von Luxburg, U. (2006). *A tutorial on spectral clustering. Tech. rep. 149. Max Planck Institute for Biological Cybernetics*.

Westphal, G., & Würtz, R. P. (2009). Combining feature- and correspondence-based methods for visual object recognition. *Neural Computation*, *21*(7), 1952–1989.

Wiskott, L., & Sejnowski, T. J. (2002). Slow feature analysis: unsupervised learning of invariances. *Neural Computation*, *14*(4), 715–770.

Zou, W. Y., Ng, A. Y., Zhu, S., & Yu, K. (2012). Deep learning of invariant features via simulated fixations in video. In *Proc. NIPS* (pp. 3212–3220).