Object Detection and Feature Base Learning with Sparse Convolutional Neural Networks

Alexander R.T. Gepperth

Institute for Neural Dynamics Universitätsstraße 150,
44780 Bochum, Germany
alexander.gepperth@neuroinformatik.rub.de
http://www.neuroinformatik.rub.de/thbio

Abstract. A new convolutional neural network model termed *sparse convolutional neural network* (SCNN) is presented and its usefulness for real-time object detection in gray-valued, monocular video sequences is demonstrated. SCNNs are trained on "raw" gray values and are intended to perform feature selection as a part of regular neural network training. For this purpose, the learning rule is extended by an unsupervised component which performs a local nonlinear principal components analysis: in this way, meaningful and diverse properties can be computed from local image patches. The SCNN model can be used to train classifiers for different object classes which share a common first layer, i.e., a common preprocessing. This is of advantage since the information needs only to be calculated once for all classifiers. It is further demonstrated how SCNNs can be implemented by successive convolutions of the input image: scanning an image for objects at all possible locations is shown to be possible in real-time using this technique.

1 Introduction

In many real-world classification tasks there is a need for classifiers that can learn from examples, such as neural networks (NNs) or support vector machines. Typically, the performance of such classifiers depends strongly on a suitable preprocessing of the input, but it is far from clear what characterizes an optimal preprocessing or if there even exists an optimal solution. Sometimes it is required that the dimensionality of the input should be reduced as far as possible, whereas another objective is to make preprocessing invariant to certain transformations of the input (typically translation, rotation and scaling are investigated in this context). The process of choosing an appropriate preprocessing is referred to as *feature selection*. In addition to constraints on error rates, processing time is usually bounded from above, too, especially in computer vision. Therefore, not only the accuracy of classifiers is important but also their execution speed.

Convolutional neural networks (CNNs) [7] were proposed to address all of these issues. They are specialized instances of multilayer perceptrons (MLPs) and thus essentially feed-forward NNs. Due to their connectivity, CNNs can be implemented by successive convolutions of an input image, permitting very high

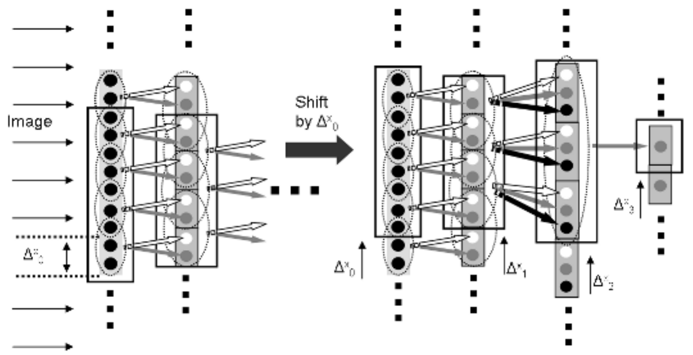

Fig. 1. Sketch of the SCNN network model (cross-section, y-dimension not shown). Receptive fields are drawn in by dotted ellipses, cells in the hidden layers are separated by black lines. Input filters connecting neurons to their receptive fields are shown as arrows in different shades of gray which match the shade of the destination neuron they project to. Arrows of the same shade of gray represent *equivalent* input filters, see text for details. In addition, the *step sizes* Δ^x_i are shown: they give the number of neurons by which the SCNN (indicated by large black boxes) samples its layers, i.e., the input image for $i = 0$, when performing whole-image searches. The effect of spatial sampling, i.e., shifting the classifier in layer 0, is shown on the right-hand side.

execution speed (see [1,11] for recent applications of CNNs). CNNs operating on unprocessed image data essentially *learn* a preprocessing transform, thus integrating feature selection into the training process.

In this article, a new convolutional neural network architecture termed *sparse convolutional neural network* (SCNN) is presented and its possibilities for object detection are explored. Since SCNNs can be implemented using consecutive convolutions, whole-image search at multiple scales is possible in real-time on standard present-day computer hardware. Furthermore, the SCNN model is intended to perform feature selection from unprocessed image data: a hybrid supervised-unsupervised learning algorithm is described which computes meaningful and diverse features by the interplay of local nonlinear PCA and error minimization. Lastly, an algorithm for learning a common image representation that is shared by several SCNN object classifiers is described. The advantage of different classifiers using the same preprocessing is that preprocessing needs only to be performed once per image when performing whole-image searches.

2 Sparse Convolutional Neural Network Classifiers

Like the original proposal [7] they are derived from, SCNNs are feed-forward neural networks with local receptive fields (see fig. 1). However, the connection structure in SCNNs has been considerably modified as compared to [7]. The proposed model is simpler and can —once trained— be tested using existing software for simulating multilayer perceptrons. Furthermore, the issue of obtaining meaningful and diverse features is addressed using a direct approach. The original CNN model attempts to achieve this by connecting hidden layers only to certain (not all) succeeding layers, which has been experimentally shown to lead to dissimilar feature maps. It is unknown, though, what effect the global network structure has on this mechanism and how many experimental trials are necessary for this mechanism to work. In the SCNN model (see fig. 1), feature complexity and diversity are enforced by additional unsupervised terms in the learning algorithm. They cause outputs of different feature maps at the same image location to be (nonlinearly) decorrelated and to have extremal variance in a way very similar to nonlinear principal components analysis [4]. Employed principles are gradient-based variance maximization of neuron outputs, decorrelation and weight vector normalization. The SCNN model has an input layer of fixed dimension, one or more hidden layers, and an output layer containing a single element. Each layer receives input from one other layer (the preceding one) and projects to a single layer (the succeeding one, see also fig. 1).

2.1 Network Model

Since SCNNs are specialized instances of MLPs, the network structure is discussed without reference to the implementation as successive convolutions. Constraints arising from this implementation are discussed at the end of this section.

Connectivity. A layer $l = 0, \dots, l^{\max}$, having dimensions $L_l^x \times L_l^y$ is composed of identical *cells* of neurons of dimension $C_l^x \times C_l^y$. Thus, a neuron can be assigned coordinates $\boldsymbol{n} = (l, \boldsymbol{c}, \boldsymbol{i})$, where $\boldsymbol{c}$ denotes the two-dimensional index of the cell within layer l, and $\boldsymbol{i}$ the neuron's coordinate within its cell. Within one cell, each neuron is connected to the same rectangular patch of neurons in layer $l-1$ which is termed a neuron's *receptive field* (RF). Receptive fields in layer $l-1$ can overlap in x- and y-direction by $O_{l-1}^x \times O_{l-1}^y$. The set of all weights connecting a neuron to its RF is denoted *input filter*. Since it is in one-to-one correspondence to a RF, it can naturally be arranged in a rectangular scheme with dimensionality $I_{l-1}^x \times I_{l-1}^y$ which is identical to that of the RF. Connection strengths are denoted by $w_{\boldsymbol{n'n}}$ where $\boldsymbol{n}$ specifies the coordinates of the destination neuron and $\boldsymbol{n'}$ those of the source neuron. Please refer to fig. 1 for a visualization. Each neuron (except for those in the input layer) is connected by a trainable weight to a bias neuron whose activation is constant (here: 1.0).

Constraints. The first set of constraints comes from the geometrical consistency of the SCNN. Trivially, given a layer l, L_l^x, L_l^y must be integer multiples of C_l^x, C_l^y. Furthermore, the number of input filters in layer $l-1$ must be identical to the number of cells in layer l. Thus, we get two conditions

$$L_l^{x,y} = kC_l^{x,y}, \; k \in \mathbb{N}^+ \tag{1}$$

$$\frac{L_l^{x,y}}{C_l^{x,y}} = \frac{L_{l-1}^{x,y} - I_{l-1}^{x,y}}{I_{l-1}^{x,y} - O_{l-1}^{x,y}} \tag{2}$$

As in the CNN model, a weight-sharing constraint enters via the requirement that neurons within a layer l, having the same within-cell coordinates $\boldsymbol{i}$ but being connected to different RFs, must have identical input filters. It is this constraint which allows to implement a network run by a series of convolutions. In contrast, each neuron in one cell is allowed to be connected to the common RF by different filters than the other neurons in that cell. Effectively, the size of one cell, $C_l^x \times C_l^y$, specifies the number of convolution filters necessary for the simulation of each layer, whereas the size of receptive fields (equal to input filter size $I_{l-1}^x \times I_{l-1}^y$) determines the dimensions of the convolution filters. For each layer l, sets of weights that are required to be identical by the weight-sharing property are called *equivalent.* Obviously it is desirable to obtain a trained SCNN which requires as few convolution filters as possible while maintaining high classification accuracy.

A further constraint comes from the implementation that is used for whole-image search (see section 3) although it is not necessary for the simulation of the SCNN model per se: it requires that step sizes Δ_l^x, Δ_l^y in layer l (i.e., the differences between the size of input filters projecting to layer $l+1$ and their overlap) must be integer multiples of that layer's cell sizes. Thus it is ensured that the classifier starts and ends at cell boundaries in all layers if it is shifted in the input image by $\Delta_0^{x/y}$, see also fig. 1. In precise terms:

$$\Delta_l^{x,y} \equiv I_l^{x,y} - O_l^{x,y} = kC_l^{x,y},\ k \in \mathbb{N}. \tag{3}$$

Activation Functions. The activity $A_{\boldsymbol{n}}$ of a neuron is calculated from the activities of its RF and the weight values in its input filter as $A_{\boldsymbol{n}} = \sigma(\sum_{\boldsymbol{n'} \in \mathrm{RF}} A_{\boldsymbol{n'}} w_{\boldsymbol{nn'}})$ using the sigmoidal activation function $\sigma(x) = \frac{x}{1+|x|}$.

2.2 Learning in SCNNs

Initially, all weights are initialized to small random values between -0.01 and 0.01 (see [8] for a motivation of this initialization). Then, a weight-sharing step is performed: for each layer l, the average of each set of equivalent weights is computed. Subsequently, all equivalent weights within layer l are set to their previously computed average value. In this way, all equivalent weights have identical values at the start of training. During each learning step or *epoch*, all weights of the SCNN are treated as if they were independent. An improved variant of the well-known Rprop learning algorithm (IRprop+, see [5]) is applied to the SCNN using dataset D_{train} for 80 epochs. After each epoch, the weight-sharing condition is enforced as described before. Note that weight-sharing is enforced separately for the bias weights of each layer.

The mean squared error (MSE) is calculated as $E_{\text{MSE}}(D) = \frac{1}{|D|}\sum_{p=0}^{|D|}(A_p^{out} - c_p)^2$ using a dataset D. It uses the class label c_p of pattern p and the activation A_p^{out} of the CNN's output neuron in response to pattern p. The learning rule for each weight is composed of the usual MSE-minimizing term plus an additional unsupervised term. The additional term is a nonlinear version of Oja's rule [4]:

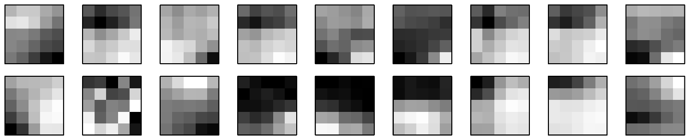

Fig. 2. Layer 0 input filters of an SCNN trained on cars (see section 5.1). Shown are filters obtained by different learning rules: MSE gradient (upper row) and hybrid learning rule described in the text (lower row). Many filters in the upper row are almost identical whereas, in the lower row, such redundancy does not occur.

$$\Delta w_{\boldsymbol{n'n}} = \gamma A_{\boldsymbol{n}}(A_{\boldsymbol{n'}}\sigma(\sigma^{-1}(A_{\boldsymbol{n}})) - A_{\boldsymbol{n}} w_{\boldsymbol{n'n}} - \sum_{\boldsymbol{j}\in\text{cell}(\boldsymbol{n}),\boldsymbol{j}<\boldsymbol{n}} A_{\boldsymbol{j}} w_{\boldsymbol{n'j}} \tag{4}$$

where γ is a small positive constant and the sum on the right-hand side of the equation runs over all neurons in the same cell as $\boldsymbol{n}$ whose within-cell coordinates are component-wise smaller than those of $\boldsymbol{n}$.

During training, model selection is performed using $E_{\text{MSE}}(D_{\text{val}})$ alone. When evaluating the performance of a trained network, the classification error $\text{CE}(D_{\text{test}})$ is used. It is defined as $\text{CE}(D) = 1 - \frac{1}{|D|}\sum_{p=0}^{|D|}\theta(A_p^{out} - \tau)$, where θ denotes the step function and τ a threshold assigned to each NN (always taken to be 0).

A few comments on the chosen learning algorithm are in order: local nonlinear principal components analysis is performed within each receptive field, but modified by the MSE gradient. The unsupervised part of the learning rule performs decorrelation of all neurons within a cell and tends to input filters with an euclidean norm of 1.0. It is an extension of the algorithm given in [2] where only orthonormalization was performed (not by gradient descent but operating directly on the weights). Due to unsupervised learning, neurons within a cell capture a part of their input whose variance is maximally large. Furthermore, the neurons' input filters tend to orthogonality, i.e., diversity (see fig. 2).
For weights connecting to the output neuron, the unsupervised term is not considered because it interferes too much with minimizing the MSE.

3 A Convolutional Architecture for Whole-Image Search

The neural network architecture described in the previous sections is particularly suited, due to the weight-sharing constraint, for fast implementation by means of convolution filters (see, e.g., [6] for an introduction). However, it is possible to achieve far greater speed gains when considering *whole-image search*, i.e., the application of a fixed-size classifier at every conceivable position within an image, possibly at several scales. In this context, CNN architectures like the SCNN model have the tremendous advantage that convolutions for overlapping classifiers need only be computed once. This can be understood by considering that input filters in the SCNN do not depend on their spatial position within a

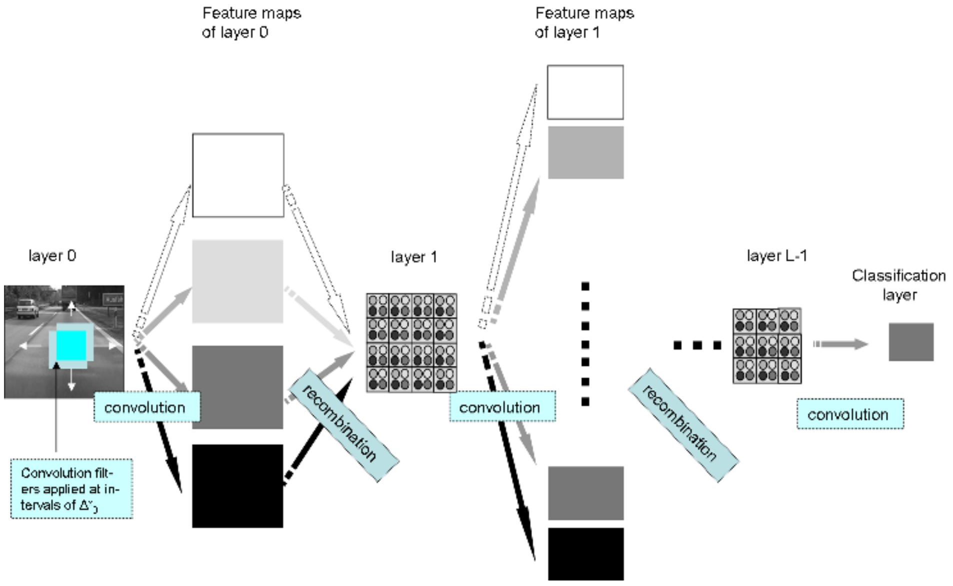

Fig. 3. Sketch of the SCNN architecture for whole-image searches. SCNN model. The input layer consists of the whole image, and successive layers are correspondingly enlarged. Input filters of the SCNN translate into convolution filters: convolution results of a layer with its input filters are called *feature maps*. The recombination of feature maps into the next layer is defined by the connectivity of the SCNN. Identical shades of gray of hidden layer neurons and feature maps indicate this. Instead of converging to a single output neuron, the SCNN now converges to a layer where each neuron represents the output of one SCNN classifier. The number of input filters (i.e., feature maps), hidden layer neurons and similar SCNN parameters are examples.

layer due to weight-sharing. By inference, the whole image needs to be convolved *only once* with all input filters in order to produce a classification result at each position within an image. Please see fig. 3 for details of the convolutional architecture and fig. 1 for details on whole-image search. Furthermore, since the input image is usually subsampled by the input filters of the first network layer (always the case when $\Delta_0^x \geq 2$, see fig. 1), only the convolutions with these input filters contribute significantly to the total processing time. The whole-image classification problem then reduces to filtering with a limited number of (usually nonseparable) filters; if real-time performance is desired, the mask size should be small (typically, sizes of 5, 7 or 9 are chosen).

The SCNN model presented here belongs to the class of convolutional neural networks which were originally proposed in [7]. A crucial difference is that no implicit subsampling of feature maps is performed, whereas in [7], feature maps are successively filtered and subsampled until they converge onto one neuron, the output of which is combined with similar neurons to form a classification output. In the SCNN model, subsampling is performed if the step sizes in one layer are chosen larger than 1, but the choice of subsampling filters is not defined a priori

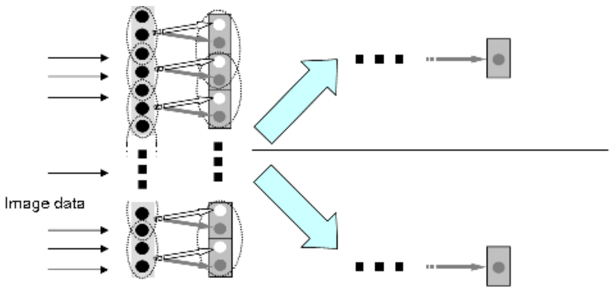

Fig. 4. Exemplary SCNN architecture for feature base learning. The horizontal line indicates that the processing streams converging onto the two (or more) output neurons do not have any connections in common.

(i.e., Gaussian smoothing) but learned by the network, too. It should be stressed here that subsampling *is* possible in the SCNN model, but at this point it seems more practical to let the SCNN *learn* the downsampling filters as well. A second difference is that feature maps are recombined into layers in the SCNN model after each convolution, whereas in [7], feature map outputs are recombined only in the last layer of the network. Recombination after each convolution inflicts a small computational cost, but it can be expected that it results in more reliable detection of conjunction features similar to the object detection architectures of [9,12]. Lastly, there exists a direct mechanism of enforcing diversity among the learned features of the SCNN, which guarantees without the need for additional experiments that informative and non-redundant features are learned which, in addition, capture a significant part of the input's local variance (by the local PCA property).

4 Feature Base Learning

An interesting application is motivated by the observation made in the previous section that the computational load is biggest during the simulation of the first layer. If two networks had identical processing in that layer, they could be used simultaneously for whole-image search while the convolutions of the input image would only need to be computed once. Stated in different terms, it would be interesting to find out if there is a *common feature base* for two or more object classes, i.e., a preprocessing of the input which is suited for representing all of the object classes under consideration. Therefore, it is investigated if and how a common feature base for N object classes can be learned only from available examples. It is tested by experiment whether it is possible to achieve classification rates comparable to separately trained classifiers. In the formalism of SCNNs, there exists a straightforward approach: N networks are trained independently from each other using methods given in section 2.2, but after each iteration of the learning algorithm, a weight-sharing constraint is enforced between the filters in the input layers of all networks.[1] An alternative interpretation is a single network which converges onto N output neurons by non-interfering processing streams from a common first hidden layer. Please see fig. 4 for a visualization. Model selection is performed using the sum $\sum_{i=0}^{N} E^i_{\mathrm{MSE}}(D^i_{\mathrm{val}})$ of the mean squared errors of each individual classifier on its validation dataset D^i_{val}.

5 Experiments

Two types of experiments are conducted: the speed of the system is (empirically, not theoretically) determined using selected SCNN topologies, and tests of topology-dependent classification performance are conducted. The latter task is performed off-line using data from real-world classification tasks. (see fig. 5).

5.1 Classification Tasks and Training Data Generation

Most experiments described here are based on the problem of car classification in real-world video traffic scenes (see, e.g., [3]). For a few experiments, the problem of traffic sign classification is considered in addition. However, this problem is not a present focus of investigations, therefore training data are much less rigorously selected and tested, and results may not be very generalizable.

Object classifiers are trained to distinguish objects from background. Training data are generated by marking rectangular regions of interest (ROIs) that contain objects. Objects are enclosed as tightly as possible. Negative examples are also created manually, although their choice is more ambiguous. Some representative training examples for cars and traffic signs are shown in fig. 5.

Due to the fact that whole-image searches usually sample the image at step sizes $\Delta_0^{x/y} > 1$, the classification must be invariant to certain transformations (especially small translations and rescalings). This requirement is encoded into the training examples. Let us define some notation: a *training example* consists of a *class label* and a *region of interest* (ROI) within a specified image. The ROI either does or does not enclose an object: to indicate this, the class label is set to 1 for an object and to -1 otherwise. A *training dataset* D contains N examples. Before using a dataset for training, a defined number of transformations is applied to the ROI of each example, creating the *transform dataset* D^{tr}.

First of all, the transformations to be applied must be specified as well as the degree of invariance which the classification should have with respect to these transformations. Let us assume that each transformation $f_t^{\alpha}, t \in [0, \dots, T-1] : D \mapsto D^{\mathrm{tr}}$ can be continuously parametrized by a single parameter α, and that a total of T different transformations exists. Let $f_t^{\alpha=0}$ denote the identity transform. Then a limit $\alpha_t^{max} > 0$ must be specified, stating the range of parameters $I_t = [-\alpha_t^{max} \dots \alpha_t^{max}]$ in which classification invariance should hold. Further assuming that all transforms commute (fulfilled for translation, rotation and scaling in two dimensions), we obtain a map

[1] Note that the dimensions of the input layers need not be identical, only the dimensions of the input filters in the input layer.

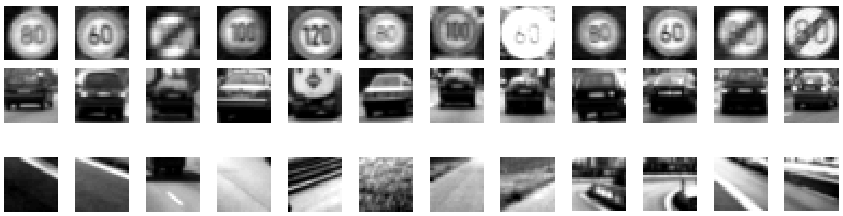

Fig. 5. Positive and shared negative examples for the car/traffic sign object classes

$$\tau : D \mapsto D^{\text{tr}}; r \mapsto (f_1^{\alpha_1} \circ \cdots \circ f_T^{\alpha_T})(r), \alpha_k \in I_k, r \in D.$$

that is applied a defined number of times to each example in D. From the results, the transform dataset is created: it is therefore larger than the original dataset of examples. In the implementation presented here, a certain invariance to scaling and translation is required. Translation is modelled by two transformations, one for horizontal and one for vertical translations. The parameters α_x and α_y of both transformations are interpreted as the percentage of an ROI's width or height by which it should be shifted. The single scaling transform enlarges or reduces an ROI's width and height by a factor of α_{sc} while ensuring that the center of the ROI stays constant. In addition, it is required for the map τ that each transformation result must completely contain the original example.

D^{tr} is generated from labeled data by applying τ 9 times per example and uniformly drawing from the parameter intervals $I_{x,y}$, I_{sc} defined by $\alpha_{sc}^{max} = \sqrt{2}$,$\alpha_{x,y}^{max} = 10$. From the transform dataset, three disjunct datasets D_{train}, D_{val} and D_{test} are created which contain 2000 examples each, half of them positive. The image content within the ROIs is up- or downsampled to a fixed size of 25x25 pixels. Whenever necessary, appropriate smoothing and bicubic interpolation are performed. For later experiments, three additional car datasets are created from D^{tr} where each ROI is shifted by 50% of its width to the left. The idea behind this classification task is to make detection more robust by checking if the "left-shifted" object classifier indeed finds half a car at the left of a detected car.

5.2 Off-Line Classification Performance of Single SCNNs

Due to the weight-sharing constraint (see section 2.2), the number of free parameters in an SCNN is greatly reduced. The choice of an appropriate topology is therefore crucial since the number of free parameters in the network depends directly on it. Since the correct choice of NN topology for a given classification task is still, in general, an unsolved problem, a number of experiments was conducted to identify suitable topologies. The search space can be reduced by the requirement that small input filters should be used in the input layer, as well as by the architectural constraints (1), (2) and (3) which SCNNs must obey.

In each experiment, a certain SCNN topology is trained 6 times using a different random seed each time, and the best classification result CE(D_{test}) is

Table 1. Best classification errors of various SCNN topologies for cars (C), cars shifted left (L) and traffic signs (TS) (errors are given in percent). Cells and layers are quadratic so only one dimension of their sizes is given. In row 0, the result for two fully connected reference networks of MLP type is given. SCNNs 1-5 demonstrate the effects of varying input filter sizes and numbers. Notable is the improvement when allowing 4x4 input filters as in SCNN 5. Rows 6 and 7 give results for the feature base learning (using topology 4, see text) of two and three object classes. For comparison, the last row shows the results of the SNoW-architecture [10] using the Winnow update rule.

Nr.	dimensions	filter size	nr.filters	conn.	free param.	$min_{\text{exp}}\text{CE}(D_{\text{test}})$
0	25-5-1	(25)	-	15650	15650	5.5 (C), 5.4 (L), 6.7(TS)
1	25-9-1	21-9-x	3-1-x	36162	4050	6.8
2	25-18-1	9-18-x	2-1-x	26568	648	7.8
3	25-30-1	7-30-x	3-1-x	45000	1341	6.7
4	25-22-1	5-22-x	2-1-x	12584	584	6.8(C),5.8(L),11.4(TS)
5	25-44-1	5-44-x	4-1-x	50336	2336	6.3
6	feature base learning using SCNN 4					7.3(C),5.9 (L), 10.9 (TS)
7	feature base learning using SCNN 4					6.5(C),11.0 (TS)
8	SNoW, std. parameters, 50 cycles					13.6

taken to be a measure of that topology's learning capacity. As a baseline, a fully connected NN with one hidden layer is identically trained on the car and traffic sign classification problems. Table 1 gives an overview over representative SCNN topologies as well as the fully connected reference networks. When considering SCNNs with one hidden layer, two ways to improve classification results were identified: increasing the size, or alternatively the number of input filters in the input layer. Obviously, both operations lead to a larger number of free parameters. The best topology found in this way has filter sizes of 5x5 pixels in the input layer, yet it is not quite compatible with real-time requirements since it requires 16 convolutions of the input image in the input layer alone. It uses 80000 connections, although the actual number of free parameters is 2384[2]. Classification performance is only slightly worse than that of the reference network despite the fact that the number of free parameters is much lower.[3] If real-time capability is desired, SCNN 4 is the topology to choose. Although using a much smaller number of connections and free parameters than topology 5, it achieves only slightly worse classification performance. Please see section 5.4 for speed measurements.

For unknown reasons, the inclusion of more hidden layers did not improve performance. Many-layered topologies were constructed by adding new layers onto well-performing SCNNs with one hidden layer. Notable was much slower overall learning convergence. It is therefore conceivable that training was not conducted sufficiently long. More research will have to be applied in order to shed light on this particular point.

[2] Note that it is the number of free parameters which determines the speed of whole-image classification.

[3] It was also shown that SCNN performance is slightly superior to that of an MLP with identical connectivity as well as an SCNN using supervised learning only.

5.3 Feature Base Learning Results

SCNN topology 4 given in table 1 is used for learning a common feature base for cars and traffic signs. It is not the best-performing topology that was found but comes very close to it; what is more, it allows real-time operation. Training is performed using the algorithm given in section 4. Results are given in table 1. It is evident that classification results are comparable to those of classifiers trained separately on their respective tasks. Observe that the feature base result for traffic signs has to be compared to traffic sign results of topology 4 in table 1, not to the reference network performance: the goal was to show that the performance of the individual classifiers can be reproduced by feature base learning. When choosing SCNN topologies with larger input filters, the performance of the reference network can be approached for traffic signs, too.

5.4 Online Performance

All tests were conducted using a 1.86Mhz Centrino processor. Images had a size of 360x288 pixels; convolutions were implemented in C++, and no use was made of the capabilities of the graphics hardware. Classification was performed at three spatial scales for each frame, where each scaled image was obtained by smoothing with a size-5 binomial filter and downsampling by a factor of 2. The best-performing SCNN topology 5 given in table 1 allows a frame rate of 7 frames per second (fps), whereas SCNN 4 allows 22 fps at the price of slightly inferior classification performance. When using three classifiers of topology 4 sharing a common preprocessing, a speed of 19 fps is attainable.

6 Discussion

The SCNN model is interesting in several respects: on the one hand, it demonstrates a successful combination of supervised and unsupervised learning rules; on the other hand, it offers very interesting possibilities for practical applications. Due to its real-time capability and the ability to search images simultaneously for several object classes using using feature base learning, it is suited for applications where scene analysis is performed, which usually consists of the recognition of more than one type of object. The driving idea behind the SCNN model was to reduce the need for ”manual” feature design. With SCNNs, *some* prior knowledge must still be provided in the form of the network topology: if it is known that, for example, that features of a certain size are characteristic of an object class, the input filters should be chosen accordingly. In many cases, input filter and step sizes are constrained by real-time requirements; once input filter and step sizes are fixed, the SCNN topology constraints are sufficient for removing most of the remaining ambiguities. As with all NNs, the correct choice of topology is an unsolved problem, although in practice one can simply take the SCNN with the largest number of parameters that is compatible with application constraints. As has been demonstrated, increasing the number of free parameters tends to improve classification performance.

The issue of extending SCNN topology successfully to more than one hidden layer is a current research topic: SCNNs with two or more hidden layers may be much more powerful in capturing local combination features; furthermore, it is intuitive that feature base learning can profit greatly from such topologies. The SCNN model itself could also be extended; in particular, shortcut connections which bypass one or more layers, and subsampling layers (as in LeCun's original proposal) suggest themselves. From a theoretical point of view, a detailed examination of the interplay between the supervised and unsupervised terms in the learning rule would be interesting; the relation of learned SCNN input filters to independent components seems to be worth investigating. Last not least, it is intended to use SCNN classifiers (possibly in conjunction with other modules) to build robust and fast object detection systems that reliably work in practice[4].

References

1. C. Garcia and M. Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, November 2004.
2. A. Gepperth. Visual object classification by sparse convolutional networks. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN) 2006*. d-side publications, 2006. accepted.
3. A. Gepperth, J. Edelbrunner, and T. Bücher. Real-time detection of cars in video sequences. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2005.
4. A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10:626–634, 1999.
5. C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.
6. B. Jähne. *Digital image processing*. Springer-Verlag, 1999.
7. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
8. R. D. Reed and R. J. Marks II. *Neural Smithing*. MIT Press, 1999.
9. M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.
10. D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May 1999.
11. M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata. Pedestrian detection using convolutional neural networks. In *Proceedings of the IEEE Symposium on Intelligent Vehicles*, pages 224–229, 2005.
12. H. Wersing and E. Körner. Unsupervised learning of combination features for hierarchical recognition models. In *Proceedings of the ICANN*, 2002.

[4] Training data as well as the C++ source code for simulating and training SCNNs are available under *http://www.neuroinformatik.rub.de/thbio/group/vision*.