A convolutional neural network tolerant of synaptic faults for low-power analog hardware

Johannes Fieres, Karlheinz Meier, and Johannes Schemmel

Ruprecht-Karls University, Heidelberg, Germany
fieres@kip.uni-heidelberg.de
http://kip.uni-heidelberg.de/vision

Abstract. Recently, the authors described a training method for a convolutional neural network of threshold neurons. Hidden layers are trained by by clustering, in a feed-forward manner, while the output layer is trained using the supervised Perceptron rule. The system is designed for implementation on an existing low-power analog hardware architecture, exhibiting inherent error sources affecting the computation accuracy in unspecified ways. One key technique is to train the network on-chip, taking possible errors into account without any need to quantify them. For the hidden layers, an on-chip approach has been applied previously. In the present work, a chip-in-the-loop version of the iterative Perceptron rule is introduced for training the output layer. Influences of various types of errors are thoroughly investigated (noisy, deleted, and clamped weights) for all network layers, using the MNIST database of hand-written digits as a benchmark.

1 Introduction

Many models of the human visual system assume a hierarchical set of feature detectors to play a fundamental role in invariant object recognition [13, 17, 12, 18]. The idea is that a visual representation of a natural object is composed of a number of smaller shapes which, each taken by themselves, appear more invariant under transformations than the entire object as a whole. Using a hierarchical system, where complex features are inferred from the presence or absence of many simpler features, recognition can be performed more robustly and with less computational effort compared to learning each single visual representation of the whole object.

Inspired by biology, convolutional neural networks apply this idea in the engineering field: The first layer usually detects simple features, e.g., oriented line segments. By successive feature extraction through the layer hierarchy, more and more complex shapes, and finally entire objects can be recognized in higher layers. Such networks have been shown to be robust image classifiers, provided the details of the network topology are correctly chosen and an appropriate training strategy is applied [3, 8, 16, 6, 11].

While convolutional neural networks usually possess a huge absolute number of computable connections, they make heavy use of a concept called *weight*

sharing, reducing the actual amount of adjustable parameters: Large groups of neurons have identical weights. Thus, the same operation ("compute the dot-product with a given weight vector") must be applied over and over again to different data. Moreover, the tree-like connection topology (see Fig. 1) makes parallel evaluation straight-forward. These facts lead naturally to the idea to employ a dedicated, possibly parallely working, hardware device optimized for this simple type of operation. A custom hardware solution was applied successfully for speeding up a convolutionary network several years ago [7]. Nowadays, where almost arbitrary amounts of computing speed can be bought off-the-shelf, e.g., in form of Linux clusters, the development of non-standard computing devices is motivated more by the desire for small, low-power solutions, as needed for example in mobile applications, or when economic mass production justifies the development effort (e.g., in automotive industry). In the long run, custom analog solutions could be an alternative to standard processors in terms of production yield, especially in the field of "soft" computing techniques: While inevitable defects occurring on microchips constitute a serious issue in large digital designs, one can envisage trainable systems working as well on imperfect, or even partly damaged, substrates.

A mixed-signal analog/digital Very-Large-Scale Integration neural network architecture has been developed by some of the authors [15]. A prototype chip is available and has been applied successfully to real-world tasks [4]. Connecting multiple chips by digital links allows smooth scalability[1]. The authors proposed a convolutional network implementation for this hardware architecture [2]. The threshold neurons provided by the hardware render gradient based methods inapplicable. Instead, a mixture of self-organized clustering and Perceptron learning is employed. The present paper focuses on techniques for assuring robustness of the algorithm against variations of the hardware substrate.

2 General Setup and Training

Input layer. In order to be processed by the hard-threshold network, the grey-level input images are transformed into a binary representation by applying a threshold at half the maximum pixel value.

Hidden layers. Network layers consist of a stack of *feature planes*, each of which is a rectangular grid of neurons (see Fig. 1). A group of neurons from adjacent planes within the same layer, located at the same grid position, will be referred to as a *hyper column*. The neurons in a given hyper column receive input connections from the same local neighborhood of neurons in the preceding layer (which will be called their *input region*), but are each tuned to detect a different feature. All hyper-columns within a given layer are identical with respect to their neurons' synaptic weights (*weight sharing*), while receiving inputs from shifted input regions, depending on their own grid position.

Four layers of this kind are present, where—alternatingly—the odd-numbered are S-type (or *recognition*) layers, and the even-numbered are C-type (or *blur-*

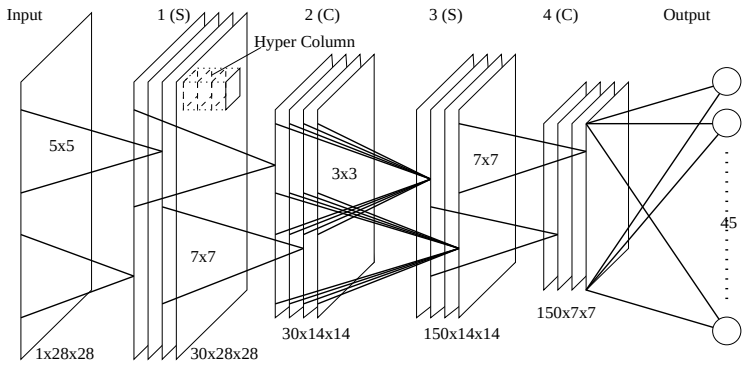

Fig. 1. Network topology Layers are organized in feature planes, which are regular grids of neurons. Neurons receive input connections from a local neighborhood in the previous layer.

ring) layers. The S/C notation is adopted from [3].[1] The S-layer weights are adjusted during training, whereas the C-layer neurons have all their weights fixed to 1. Another particularity of C-layers is that neurons receive input only from the corresponding plane in the preceding S-layer. This way, a C-layer spatially blurs the S-layer's activation pattern. The blurring operation results in local shift invariance [3] and thus in higher-order invariances in higher network layers. In the C-layers, layer dimensions are sub-sampled by a factor of 2. Together with the connection topology, this leads to growing *receptive fields* in higher-layer neurons. The receptive field of a neuron denotes the area in the input layer from which this neuron eventually receives input. For computing border cells, the previous layer is padded with the background value (-1).

All neurons compute their output according to

$$O = \beta\left(\mathbf{w} \cdot \mathbf{I} - t\right), \tag{1}$$

where $\mathbf{w} = [w_1, \ldots, w_N]$ and t are the neuron's weights resp. threshold, $\mathbf{I} = [I_1, \ldots, I_N]$ are the current input values $I_i \in \{-1, 1\}$, and $\beta(x)$ is the bipolar step function (1 for $x > 0$, -1 otherwise).[2] The threshold t is not incorporated in the training process (see below), but is set explicitely afterwards. For the S-layers, t is set to a fraction of the respective neuron's maximally achievable activation $t = T_S \sum_i |w_i|$. For C-layers, $t = T_C$. The constants T_S and T_C are optimized for each layer.

Training of the S-layers proceeds bottom-up. Assume that layer n, consisting of K feature planes, is to be trained. Only the weights of one prototype hyper column are identified, which is then duplicated to the full layer dimensions (weight sharing). For each training image (index j) and for each grid position

[1] Fukushima named the layer types after the Simple and Complex cells found in the mammalian visual cortex by Hubel and Wiesel (1968).

[2] The hardware [15] uses 0/1 neurons. Before transferring the weights to the hardware, they are converted accordingly which is possible by a simple transformation.

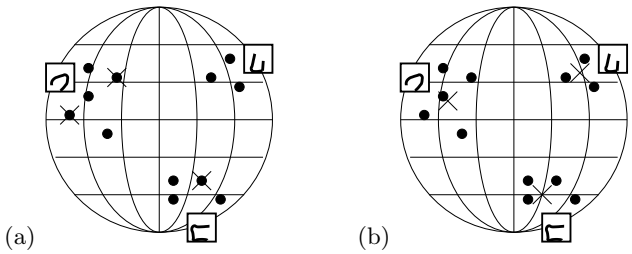

Fig. 2. Schematic visualization of the clustering process. Dots represent the input vectors $\mathcal{I}$, located on a hyper sphere, crosses represent the weight vectors w_k. Input vectors are clustered around typical shape features. **(a)** Before training, w_k are initialized with random members from $\mathcal{I}$. **(b)** After training, w_k have settled in cluster centers.

in layer n (indices x, y), there is one input vector $\mathbf{I}^j_{xy}$ to the hyper column at that position. Let $\mathcal{I} := \{\mathbf{I}^j_{xy}/||\mathbf{I}^j_{xy}||\}$ be the set of all those (normalized) input vectors. These vectors lie on a unit hyper-sphere in the input space, and, since the training images contain the objects to be recognized, the vectors are likely to be clustered around shape features which are typical for the objects in question and which can be recognized in layer n. Consequently, K clusters in $\mathcal{I}$ are identified and the cluster centers are set as the weight vectors of the K neurons in the prototype hyper column (see Figure 2 for an illustration). This consideration does not depend on certain shape features to appear always at the same positions on the input layer, since $\mathcal{I}$ includes data from all grid positions.

For clustering, the K-Means algorithm is used (see e.g. [5]), where the angle between two vectors is taken as the distance measure. This training scheme is equivalent to competitive Hebbian learning [14]. The algorithm stops if either only a small fraction (0.5%) of patterns switched their cluster assignments in the previous epoch or a maximum of 100 epochs has elapsed. The exact definition of the termination criterion does not seem critical. Only a subset of the training images (200 per class) is considered for clustering. Taking 400 per class instead does not improve results.

Output layer. The output layer is a pairwise linear classifier, fully connected to the last intermediate layer. Each output cell is trained to discriminate only two classes. For 10 pattern classes, and considering every possible combination of two classes, there are 45 output units. When evaluating an unseen pattern, each unit votes for one of the two classes it was trained with. The class receiving the most overall votes wins. Training is done using standard Perceptron learning, where after each pattern presentation, a neuron's weights and threshold $\mathbf{v} = [w_1, \ldots, w_N, t]$ are updated according to:

$$\mathbf{v} \leftarrow \begin{cases} \mathbf{v} - O\mathbf{J}, \text{ if } O \text{ is incorrect} \\ \mathbf{v}, \text{ if } O \text{ is correct} \end{cases}, \tag{2}$$

where $\mathbf{J} = [I_1, \ldots, I_N, -1]$ is the current input vector plus an additional constant component to account for the bias t in $\mathbf{v}$, and O is the neuron's current

output. Patterns are presented in random order and the training is terminated if the output is correct for a pre-defined number of consecutive pattern presentations, The required number of consecutive correct patterns is 10,000 in all of the experiments, resulting almost always in termination after about 1-2 million pattern presentations.

Meta-parameters. For a concrete network implementation, various details regarding the network topology, as well as the threshold constants T_S and T_C, must be fixed. The meta-parameters needed for one adjacent S/C layer pair are defined now: K denotes the number of feature planes per layer. For the S-layers, the input region is a square of hyper-columns of size $D_S \times D_S$. C-neurons have a circular input region with diameter D_C with all their weights fixed to 1. The threshold parameters T_S and T_C have already been discussed above.

3 On-Chip Training

Before operating the hardware, the neural weight values are loaded onto the network chip. Due to substrate imperfections and inherent device variations of the analog computing units, network results generally differ from the ones expected from exact computation. Here, it is assumed that variations in the weight values constitute the dominating effect in the considered hardware architecture. More specifically, we assume that the actual *effective weights* on the chip differ from the explicitely *programmed weights*, according to some distortion model. In order to account for these weight perturbations, on-chip training techniques are employed. On-chip training approaches are able to compensate only for fixed-pattern errors (i.e., perturbations which do not vary over time), which however, according to experiences with the prototype chip, seem to play the major role.

Hidden Layers. As detailed in section 2, network layers are trained sequentially, bottom-up. Weight training (clustering) happens in software, based on the output of the previous layer. If, after training a given layer, the trained weights are loaded onto the chip, the chip's result can be used as training input for consecutive layers. Since this way succeeding layers "learn" to live with the imperfections of the previous one, more robust behavior is expected compared to loading a completely software-trained network onto the chip.

Output layer. The straight-forward method just described does not work for the output layer because no further layer exists which could compensate for possible errors. Thus, the output layer must be configured such that the effective weights on the hardware (in contrast to the programmed weights) are optimal. For this aim, the Perceptron learning algorithm is applied in a "chip-in-the-loop" fashion: Let $\hat{\mathbf{v}}$ be the effective weight vector after the hardware has been configured with the programmed weights $\mathbf{v}$. Then, the update rule (2) is applied, with the difference that the actual output O is now computed on the hardware:

$$O = O(\hat{\mathbf{v}}(\mathbf{v})). \tag{3}$$

If the algorithm converges, $\hat{\mathbf{v}}$ will be optimal. Note that no explicit knowledge about $\hat{\mathbf{v}}$ is necessary, i.e., there is no need for quantitative error analysis.

Certainly, the crucial part is "if the algorithm converges", which is not guaranteed any more even if the data are linearly separable. For example, a malfunctioning synapse could cause the corresponding weight to grow infinitely if one ore more patterns keep being classified uncorrectly due to this synapse failure. However, heuristically, no ill behavior is observed in the presented experiments.

Although all 32,000 synapses of the network chip can be reconfigured in about a tenth of a millisecond, updating the weights after each single pattern is not very effective, due to the necessary data transfer to and from the hardware, and other overhead associated with one network run. Minimal training speed can be achieved by evaluating a few thousand patterns at once and then updating the weights with the accumulated modification. Nevertheless, in the experiments presented here, the cingle-pattern update rule is used.

4 Results

Performance with ideal synapses. The MNIST data set [9] is used as a benchmark recognition problem. It consists of 28x28 pixel grey-value images of the hand-written digits "0" through "9". Samples are shown in Fig. 3. 60,000 images are provided for training, 10,000 for testing. For finding the optimal meta-parameters, the training images are split further into a training set (50,000) and a validation set (10,000), each with equal distribution of digit classes. With the parameters found to perform best on the validation set (Table 1), 100 networks are trained with the patterns of the training and validation set combined, and tested on the test set. The average error rate obtained on the test set is 1.74%±0.10% (best network: 1.49%, worst network: 1.97%). Here, the error is given as the standard deviation within the ensemble of the 100 networks.

Performance with faulty synapses. The network's robustness is tested by artificially applying three different kinds of synaptic errors to the programmed weights.
"Noise": All effective synaptic weights are subject to perturbations by adding random normally distributed offsets to the programmed weights.
"Delete": A given fraction of all weights is set to 0, corresponding to disabled synapses.

	K	D_S	T_S	D_C	T_C
Layers 1-2	30	5	0.5	7	1
Layers 3-4	150	3	0.4	7	0

Table 1. Topology and training meta-parameters

Fig. 3. Sample digits from the MNIST data base (binarized version)

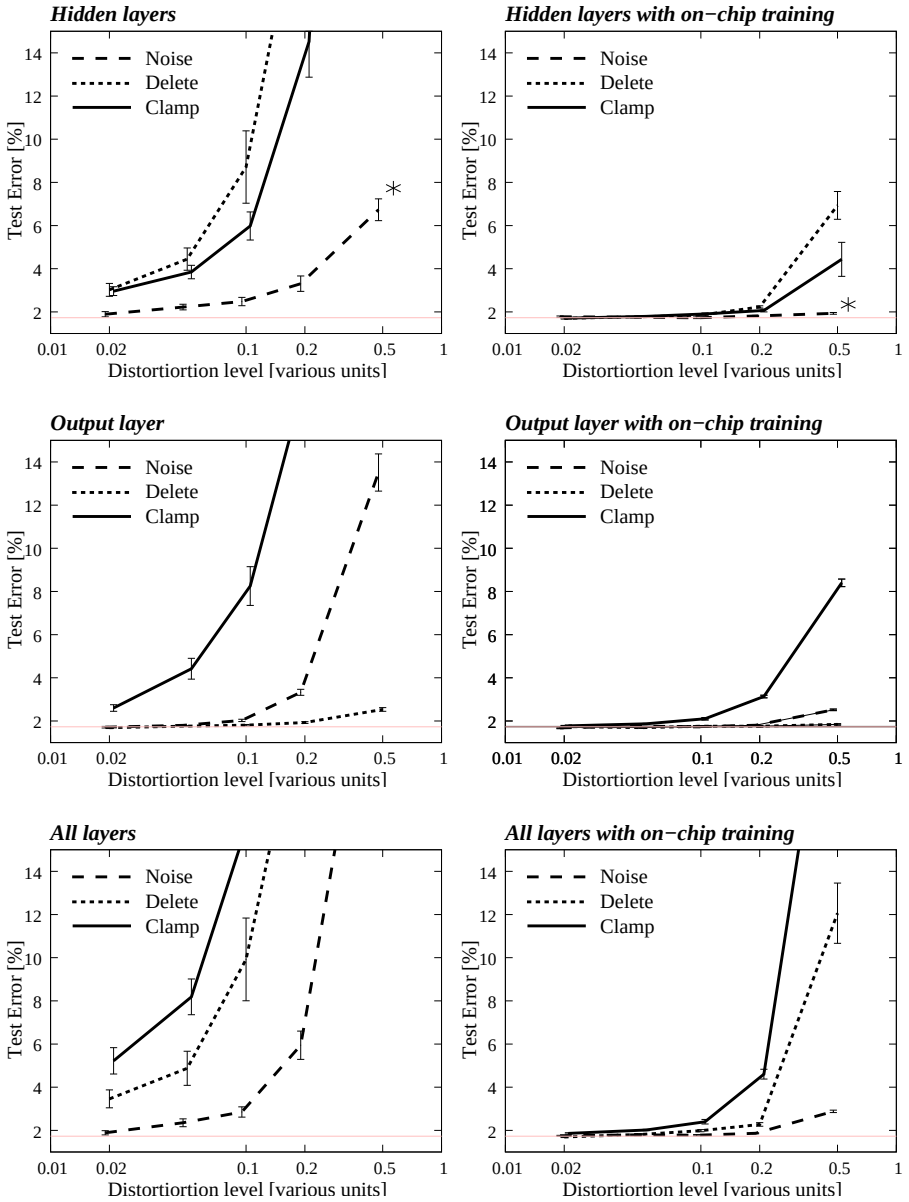

Fig. 4. Different types of synaptic errors in the hidden layers (top), the output layer (middle), and all layers (bottom). Left: Errors are applied to completely trained network. Right: Errors are incorporated in the training (on-chip approach). The x-axis denotes the width of weight perturbation for the "noise" error type, resp. the fraction of affected synapses for the "delete" and "clamp" error types. Logarithmic scale for convenience. The shaded line corresponds to network performance with ideal synapses ($1.74\% \pm 0.01\%$). Here, errors are given as errors of the average value of an ensemble ($\mathrm{stdev}/\sqrt{\#\mathrm{trials}}$). * Marked curves appear also in [2]

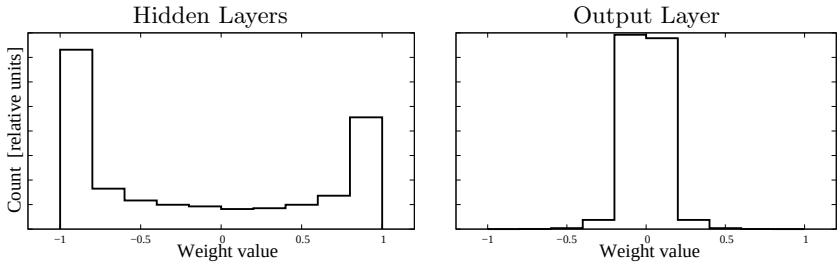

Fig. 5. Distribution of synaptic weights typically observed in the hidden layers and the output layer. Weights are forced to the interval [-1,1] by scaling each neuron's weight vector to unit maximum norm after training. Histrograms are over all weights in one trained network.

"Clamp": A given fraction of all weights is set to the extreme positive or negative value (-1,1, each 50% chance), corresponding to clamped synapses, e.g., as caused by electric shortcuts.
All presented results are obtained using a software implementation of the described network model. This is mostly due to the fact that the parallelly ongoing development of the hardware system has not quite kept pace with the requirements of the presented application. In particular, the prototype chip is too small for large networks as used here (see also section 5). Looking at it positively, this way the properties of the network model and training method are evaluated isolated from the particularities of a concrete hardware substrate, and reproducibility is ensured.

All three error types are applied separately to the hidden layers and the output layer, and, in a third setting, to the entire network. All settings are first evaluated when applying the synapse errors to a fully trained network (Fig. 4, left diagrams), and then with employing the on-chip training (Fig. 4, right diagrams). Each data point represents the average test error rate from a series of 10 independently trained and distorted networks. Clearly, the on-chip training approach is to a large extent able to compensate for the tested synaptic errors.

It is interesting to observe that the hidden layers show relatively high sensitivity to the deletion of synapses, but can cope quite well with large amounts of noise, while the output layer behaves the opposite way. This fact can be understood from the different training strategies applied: The hidden layers are trained by correlation-based learning, which is known to tend to produce extreme synaptic weights, c.f., [10]. Fig. 5, left hand side, shows a typical weight distribution in the hidden layers. In such a bi-modal weight distribution, adding noise will not easily destroy the overall behavior of a neuron, but setting synapses to zero (i.e., deleting), is very likely to strongly affect a neuron's behavior. On the other hand, a typical weight distribution in the output layer is depicted in Fig. 5, right side. Here, most weights are close to zero, so deleting synapses will with a high probability do not much harm to a neuron, while adding noise with an absolute width will likely alter a synapse's stength by a large relative factor.

It should be noted that the low sensitivity of the output layer to synapse deletion might also be promoted by the large number of synapses present per neuron: Each output neuron receives more than 7,000 inputs which are highly inter-correlated, because of the blurring C-layers, neighboring pixels on a feature plane tend to be in an equal state. When working with pruned networks in the future (see Outlook), this effect can be better quantified.

5 Conclusion and Outlook

A neural network consisting of binary threshold neurons, trained using a combination of self-organization and supervised learning, is applied to hand-written digits classification. With regard to the envisaged analog hardware implementation, robustness to computation errors is required. Therefore, the influences of various modes of synaptic malfunction are thoroughly evaluated. Two on-chip approaches are described for coping with fixed-pattern errors. Although initially not obvious, a simple chip-in-the-loop version of the Perceptron learning rule produces satisfying results. With the on-chip learning, the network shows to be remarkably resistant to unknown, but temporally invariable, synaptic errors. However, it should be noted that even with spontaneous synaptic errors which were not seen during training, the performance degrades gracefully (Fig. 4, plots to the left). For example, even with 10% randomly deleted synapses in all layers, still over 90% of all digits are correctly classified.

The error rates achieved on the MNIST data set do not quite reach the best rates reported for convolutional networks trained by back-propagation, see [9] for a "high score". But taking into account the simplicity of the training methods, the low complexity of evaluating the network (threshold neurons), and the focus on robustness, the presented method can certainly be said to be competitive. Moreover, it has been shown previously that by adding a preprocessing stage (expanding the training data set by elastic distortions), the test error can be further decreased [2].

The final aim is to implement the system on a mixed-signal hardware architecture. Setups for recognizing simple geometric shapes have been already run successfully on the prototype chip. This chip however features a maximum number of 128 inputs per neuron, which limits the number of feature planes in the first convolutional layer to 14 (corresponding to 3 x 3 x 14 = 125 inputs to the 3rd layer), and restricts also the number of possible inputs to the output layer. The evaluation of a network as shown in this paper has to be postponed for a larger implementation of the hardware architecture. However, in order to allow the most realistic evaluation of the hardware system using the prototype chip, methods are being developed for pruning the network size with comparably little drawback in performance.

Acknowledgments

This work was funded in part by the European Union, contracts nos. IST-2001-34712 (SenseMaker) and IST-2004-2.3.4.2 (FACETS). The first author was supported by a scholarship of the Landesgraduiertenförderung, Baden-Württemberg. We thank all persons who contributed to this work with useful comments.

References

1. J. Fieres, A. Grubl, S. Philipp, K. Meier, J. Schemmel, F. Schürmann: A platform for parallel operation of VLSI neural networks. Conference on Brain Inspired Cognitive Systems (BICS 2004), Stirling, Scotland (2004)
2. J. Fieres, J. Schemmel, K. Meier: Training convolutional neural networks of threshold neurons suited for low-power hardware implementation. Int. Joint Conference on Neural Networks (IJCNN 2006), Vancouver, CA, (accepted) (2006)
3. K. Fukushima: Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks 1, 119-130 (1988)
4. S. G. Hohmann, J. Fieres, K. Meier, J. Schemmel, T. Schmittz, F. Schürmann: Training Fast Mixed-Signal Neural Networks for Data Classification. Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04), 2647-2652, IEEE Press (2004)
5. J.-S. R. Jang, C.T. Sun, E. Mizutani: Neuro-Fuzzy and Soft Computing, Prentice-Hall (1997)
6. S. Lawrence, C.L. Giles, A.C. Tsoi, A.D. Back: Face recognition: a convolutional neural network approach. Transactions on Neural Networks 8(1) 98-113 (1997)
7. Y. LeCun, L. D. Jackel, B. Boser, J.S. Denker, H. P. Graf, I. Guyon, D. Henderson, R.E. Howard, W. Hubbard: Handwritten digit recognition: Applications of neural net chips and automatic learning. IEEE Communications Magazine, November 1989, 41-46 (1989)
8. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324 (1998)
9. Y. LeCun: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist
10. R. Linsker: From basic network principles to neural architecture. (Series of 3 papers) Proc. Natl. Sci. USA 83, 7508-7512 (1983)
11. C. Neubauer: Evaluation of convolutional neural networks for visual recognition. Transactions on Neural Networks 9(4), 685-696 (1998)
12. M.W. Oram, D.I. Perret: Modeling visual recognition from neurobiological constraints. Neural Networks (7) 945-972 (1994)
13. M. Riesenhuber, T. Poggio: Hierarchical Models of Object Recognition in Cortex. Nature Neuroscience 2 , 1019-1025 (1999)
14. D. E. Rumelhart, D. Zipser: Feature discovery by competitive learning. Cognitive Science, 9, 75-112 (1985)
15. J. Schemmel, S. Hohmann, K. Meier, F. Schurmann: A mixed-mode analog neural network using current-steering synapses. Analog Integrated Circuits and Signal Processing 38, 233-244 (2004)
16. P.Y. Simard, D. Steinkraus, J.C. Platt: Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Intl. Conf. Document Analysis and Recognition, 958-962 (2003)

17. K. Tanaka: Inferotemporal cortex and object vision. Ann. Rev. Neuroscience 19, 109-139 (1996)
18. S. Ullmann, S. Soloviev: Computation of pattern invariance in brain-like structures. Neural Networks 12, 1021-1036, (1999)