Training convolutional networks of threshold neurons suited for low-power hardware implementation

Johannes Fieres, Johannes Schemmel, Karlheinz Meier

Abstract— **Convolutional neural networks are known to be powerful image classifiers. In this work, a method is proposed for training convolutional networks for implementation on an existing mixed digital-analog VLSI hardware architecture. The binary threshold neurons provided by this architecture cannot be trained using gradient-based methods. The convolutional layers are trained with a clustering method, locally in each layer. The output layer is trained using the Perceptron learning rule. Competitive results are obtained on hand-written digits (MNIST) and traffic signs. The analog hardware enables high integration and low power consumption, but inherent error sources affect the computation accuracy. Networks trained as suggested are highly robust against random changes of synaptic weights occuring on the hardware substrate, and work well even with only three distinct weight values (-1, 0, +1), reducing computational complexity to mere counting.**

I. 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], [18], [14], [20]. 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.

Convolutional neural networks apply the idea of hierarchical feature detectors in a biologically inspired way. 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 [2], [9], [12], [7]. In some reported approaches the proposed training strategies require to a high degree the intuition and the skill of a human supervisor, who has to pre-determine the set of features to be recognized in each intermediate layer [2], [12], [19]. Since for many problems it is not obvious *a priori* which features are optimally suited for the detection of the object(s) in question, this step can involve time-consuming manual adjustments.

Much research has been spent on how the hidden layers can be trained automatically. On the one hand, there are straight-forward, fully supervised methods, like error back-propagation [9], which can get computationally demanding for large multi-layer networks. On the other hand, various approaches have been proposed for the emergence of a hierarchical set of feature detectors by self-organization. Several un-supervised learning rules have been applied to all or some network layers, often some sort of correlation learning (e.g., Hebbian-type [11], self-organizing maps and auto-encoding [12], competitive learning [2]). In order to classify natural images, hybrid learning strategies have been successfully employed where the intermediate layers are trained by self-organization and only a simple classifier on top is trained by some supervised method [12].

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 Figure 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 sucessfully for speeding up a convolutionary network several years ago [8]. 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 [16]. A prototype chip is available and has been applied successfully to real-world tasks [4]. Moreover, a system allowing the parallel operation of multiple of those chips is existing [1]. In the chosen hardware architecture, neural computations are performed by analog circuits,

The authors are with the Kirchhoff Institute for Physics, Ruprecht-Karls University, Heidelberg, Germany. Email: {fieres, schemmel, meierk}@kip.uni-heidelberg.de

enabling high integration and low power consumption. In contrast, communication between neurons is digital, making data transport unaffected by noise and allowing to interface and interconnect the chips by standard digital equipment [1].

In the present work, it is evaluated how a convolutional neural network which is implemented on this hardware architecture can be trained. One difficulty is rooted in the hard-threshold activation function of the featured neuron model. Since this function lacks a meaningful derivative, well-proven gradient-based methods like back-propagation cannot be applied. Most other supervised and non-supervised automated training methods reported in the field of convolutional networks assume continuous neuron outputs and therefore can as well not be applied without heavy modifications. Here, for the hidden layers, a self-organization process is proposed which extracts features from the training data by means of a simple clustering technique bearing resemblance to competitive learning. The output units are trained supervisedly using the Perceptron learning rule.

As pointed out earlier, in the considered hardware architecture weight accumulation[1] and comparison with the threshold is carried out by analog electronics. This implies that the computation result is never exact. Sources of errors include for example inherent device variations or temporal noise. Although some of the fixed-pattern variations can be measured and compensated for by calibration routines [16], a training strategy is preferred which is robust against perturbations in the calculation.

In section II, the used network and the proposed training methods are detailed. In section III, experimental results are shown, obtained on two different recognition tasks: Hand-written digits and traffic signs. The robustness of the network against weight perturbation is evaluated. Moreover, it is tested how the network's performance is affected by coarse weight quantization, which, in the shown case, implies an enormous reduction of computational complexity. The paper closes with a critical discussion and an outlook in section IV.

II. Network Details and Training

The network consists of the input layer, four hidden layers, and one output layer (Figure 1b). The synaptic weights in the hidden layers are adjusted by self-organization, whereas the output layer is trained in a supervised way.

A. Input layer

In order to be processed by the hard-threshold network, the grey-level input images are transformed into a binary representation. For the hand-written digits, this is done by applying a threshold at half the maximum pixel value. For the grey-value traffic sign photographs, a Laplacian-of-Gaussian filter (LoG) is applied, which is a simple contrast detector [6], with subsequent thresholding to mark regions with high contrast (see Figure 5).

B. Intermediate layers

One network layer consists of a stack of *feature planes*, each of which is a regular grid of neurons (see Figure 1a). A group of neurons from adjacent planes within the same layer, located at the same grid position, shall 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. In this sense, each plane computes a convolution with the preceding layer, while thresholding the result according to the steplike activation function.

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 *blurring*) layers (see Figure 1b). The S/C notation is adopted from [2].[2] The S-layers have plastic synaptic weights being adjusted during training, whereas the C-layers' neurons have all their weights fixed to 1. Another particularity of C-layers is that the number of feature planes is the same as for the preceding S-layer, and that neurons in a given C-plane receive input only from the corresponding plane in the preceding S-layer. This way, a C-layer spatially blurs the S-layer's output. The blurring operation promotes the emergence of invariant recognition since it results in local shift invariance [2] and thus in higher-order invariances in higher network layers. Moreover, in the C-layers, layer dimensions are decreased about a pre-defined factor. Together with the connection topology, this leads to growing *receptive fields* in higher-layer neurons. The receptive field of a neuron denotes the region on the input layer from which this neuron eventually receives input. Scaling is applied to a hypothetical full-sized C-layer by evaluating only some of the rows and columns in the layer grid and discarding the rest. For example, at a scaling factor of $1/a$, where a is integer, only each ath row and column are kept. The size of the blurring kernel (i.e., the input region) is always chosen greater than a, so no information is lost in the sub-sampling step. In order to compute cells at the border, two strategies are applied: For the hand-written digits, parts of the input region falling outside the previous layer's dimensions are filled with the background value (-1). For the traffic sign images, where pseudo line-endings occur at the image border, the S-type layers are cropped to not contain border neurons.

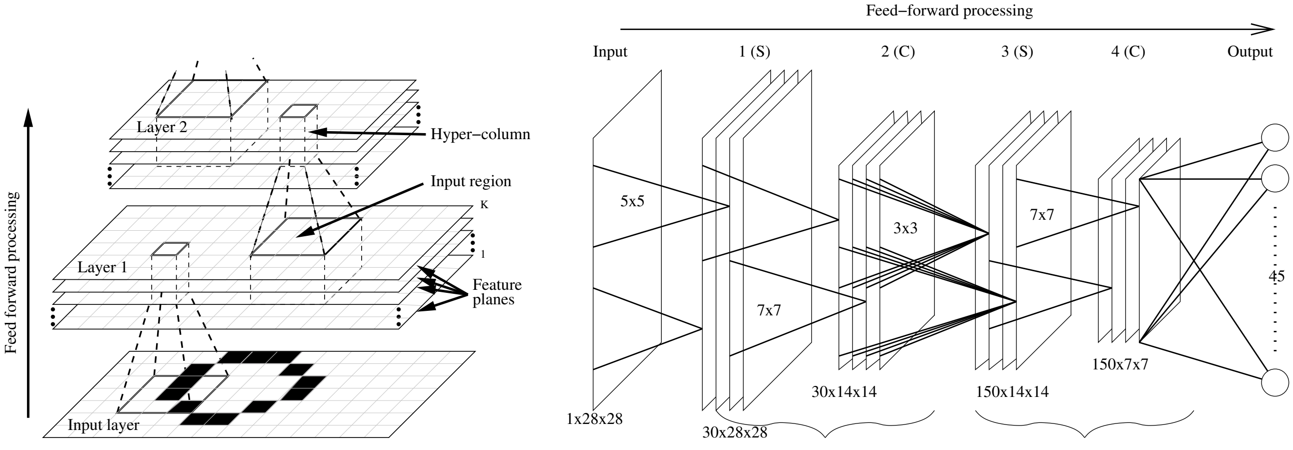
All neurons compute their output according to

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

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

[1] The usual multiply-and-accumulate operation breaks down to mere accumulation in networks of hard-threshold neurons.

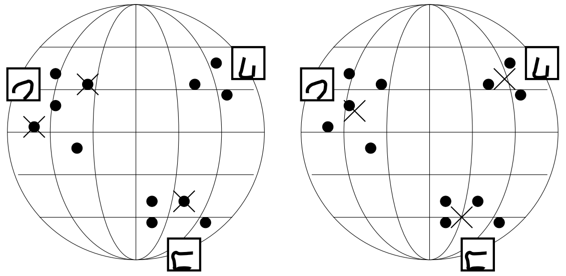
[2] Fukushima named the layer types after the Simple and Complex cells found in the mammalian visual cortex by Hubel and Wiesel (1968).



(a)

(b)

Fig. 1. Network architecture **(a)** Detailed view of the input layer and the first two hidden layers. Layers are organized in feature planes, which are regular grids of neurons (neurons are represented by small squares). Neurons receive input connections from a local neighborhood in the previous layer. Each feature plane is tuned to detect a specific feature. By successive feature extraction through the layer hierarchy, more and more complex shapes are detected. **(b)** Overall schematic. Processing left-to-right for layout reasons. The hidden layers are alternatingly of the 'S' (feature extracting) and 'C' (blurring) type. S-layer neurons are fully connected to their input region, while C-neurons get input from corresponding planes only. Groups of layers denoted by braces are referred to as S/C layer pairs. Shown layer dimensions and convolution kernel sizes refer to the MNIST experiments.



(a)

(b)

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

$I_i \in \{-1, 1\}$, and $\beta(x)$ is the bipolar step function (1 for $x > 0$, -1 otherwise).[3]

The threshold t is not incorporated in the training process (see below), and 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 independently for each layer.

Training of the S-layers proceeds bottom-up, i.e., layer n (n odd) is trained after training of layer $n-2$ has been completed. Assume that layer n, consisting of K feature planes, is to be trained. Since weight sharing is applied, only the weights of one prototype hyper column must be identified, which is then duplicated to the full layer dimensions. As a result, the trained layer can recognize every feature at every position. For each training image (index j) and for each grid position in layer n (indices x, y), there is one input vector $\mathbf{I}^j_{xy}$ to the hyper column at that position. (Remember: in S-layers, all neurons within a hyper column receive the same input vector.) Let $\mathcal{I} := \{\mathbf{I}^j_{xy}/||\mathbf{I}^j_{xy}||\}$ be the set of all those (normalized) input vectors.[4] The vectors in $\mathcal{I}$ all 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, in the proposed approach, K clusters in $\mathcal{I}$ are identified and the cluster centers are employed 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 well known K-Means clustering algorithm is used (see e.g. [5]). The angle between two vectors is taken as the distance measure. for clustering.

In detail, the procedure is as follows: At the start, the cluster centers $\mathbf{w}_k, k = 1..K$, are initialized with vectors randomly drawn from $\mathcal{I}$. Then, repeatedly, the following two steps are performed in each epoch:

1) For each vector $\mathbf{i} \in \mathcal{I}$, assign $\mathbf{i}$ to the cluster $\tilde{k}$, the center of which ($\mathbf{w}_{\tilde{k}}$) has the smallest angle to $\mathbf{i}$ ($\tilde{k} = \mathrm{argmax}_{k=1}^{K}(\mathbf{i} \cdot \mathbf{w}_k)$).

[3] The hardware architecture described in [16] uses 0/1 as possible neuron outputs. Before transferring the trained network onto the hardware, the weights and thresholds must be converted accordingly, which is always possible by a simple transformation.

[4] In practice, division by $||\mathbf{I}^j_{xy}||$ can be omitted when constructing $\mathcal{I}$. Since bipolar vectors of a given dimension have fixed length, equations involving vectors from $\mathcal{I}$ are just scaled by a constant factor, leaving the clustering result invariant.

2) Update each cluster center $\mathbf{w}_k$ to be the center of mass of all vectors being assigned to the kth cluster. Re-normalize $\mathbf{w}_k$ to unit length.

The procedure stops if either only a small fraction of patterns switched their cluster assignments in the previous epoch, or a pre-defined number of epochs has elapsed. The exact definition of the termination criterion does not seem critical. In the experiments presented, a fraction of 0.5% and a maximum of 100 epochs is used. In most observed cases, the algorithm terminates after less than 50 epochs. Because the MNIST data set is very large, to save computing time only the first 200 samples of each class are used for clustering. Taking the first 400 instead does not improve results. For the traffic signs, all images in the training set are included. Input vectors which are entirely inactive (all components equal -1) are not considered for clustering, since they are not assumed to contain meaningful features.

In order to understand this training technique in terms of well-known neural network paradigms, it is instructive to have a look at the probabilistic version of the K-Means algorithm, which would produce similar results: Here, the cluster centers are updated after each single pattern presentation. In each iteration step, the angles between a randomly selected normalized input vector $\mathbf{i} \in \mathcal{I}$ and each neuron's unit-length weight vector are evaluated and the best-matching neuron $\tilde{k}$ is updated according to

$$\mathbf{w}_{\tilde{k}} \leftarrow \mathbf{w}_{\tilde{k}} + \epsilon \mathbf{i}. \qquad (2)$$

After each such update, $\mathbf{w}_{\tilde{k}}$ is re-normalized. The constant ϵ defines the speed of learning. The update scheme (2) with successive weight normalization resembles Oja's variant of the Hebbian learning rule. Adding the winner-takes-all scheme by which only the best matching neuron is updated, the prototype hyper column is trained according to a competitive learning scheme which is in turn known to be able to perform data clustering [15], [3].

C. Output layer

The output layer is a pairwise linear classifier, fully connected to the last intermediate layer. Each output cell is trained only with patterns from two selected classes in order to separate them from each other. For c pattern classes, and considering every possible combination of two classes, there are $c(c-1)/2$ 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. If two classes receive the same number of votes, the respective pattern is counted as mis-classified. For the traffic-sign problem, a different output encoding scheme is applied (see section III).

The output neurons are trained using simple 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:

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

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 always correct for a pre-defined number of consecutive pattern presentations, or a maximum number of iterations is reached. The required number of consecutive correct patterns is *ad-hoc* set to 10,000 in all of the experiments, resulting almost always in termination after about 1-2 million pattern presentations for the MNIST data set (3-4 Million for the expanded MNIST data set).

D. Network 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 parameters needed for one adjacent S/C layer pair are defined here: 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. The factor by which the layers are scaled down in the C-layers shall be denoted by α

III. Results

A. Recognition Performance

1) Hand-written digits (MNIST): The MNIST data set [10] consists of 28x28 pixel grey-value images of the hand-written digits "0" through "9". 60,000 images are provided for training, 10,000 for testing. In the experiments reported here, 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. The validation set is used for parameter adjustments. With the parameters found to perform best on the validation set (Table I), 10 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 1.71%±0.07% (best network: 1.56%, worst network: 1.84%). The error given is the standard deviation within the ensemble of the 10 networks.

In order to give an impression of how the applied training method yields a hierarchical set of feature detectors, examples of detected features are shown. The spatial distributions of the synaptic weights of the first network layer are shown in Figure 3(a). The 5x5 weights of 20 selected neurons are visualized in a grey-scale coded scheme, (white=negative,

TABLE I

Parameters used in the experiments.

	K	D_S	T_S	D_C	T_C	α
MNIST						
Layers 1-2	30	5	0.5	7	1	0.5
Layers 3-4	150	3	0.4	7	0	0.5
Traffic signs						
Layers 1-2	25	5	0.5	11	1	0.33
Layers 3-4	100	3	0.5	11	3	0.33

TABLE II
CLASSIFICATION ERRORS [% FALSE PATTERNS]

Training set expansion	Weight quantization	Error (Avg, 10 runs) [%]	Best [%]	Worst [%]
MNIST				
None	None	**1.71±0.07**	1.56	1.84
Elastic trans.	None	**1.48±0.10**	1.26	1.65
None	-1, 0, +1	**2.80±1.06**	1.83	5.29
None	-1, 0, +1 (on-chip)	**1.80±0.10**	1.61	1.95
Traffic signs				
None	None	**2.2±1.0**	0	4.0

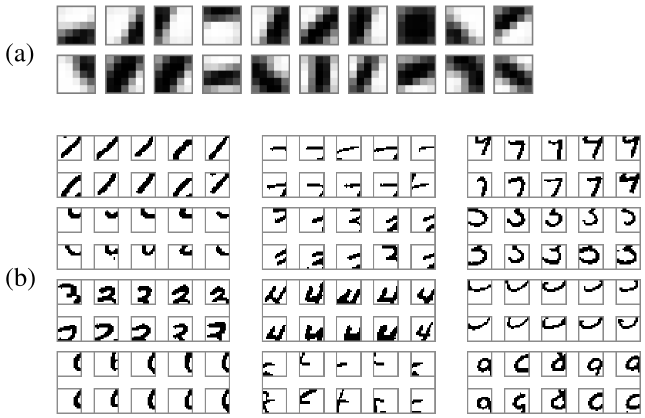


Fig. 3. Features detected in the S-layers after training. **(a)** Spatial distributions of synaptic weights of 20 selected layer-1 neurons. **(b)** For 12 selected layer-3 neurons, receptive field samples are shown which make the neuron fire. See text for more details.

black=positive). Obviously, oriented edges and lines are the preferred stimuli in this network layer. In the second recognition stage (layer 3), the plain weight values are inconvenient to interpret. Instead, effective receptive field stimuli are shown for selected neurons, i.e., image patches which cause strong activation. For this, the internal activation ($\mathbf{w} \cdot \mathbf{I}$ in (1)) of each neuron is recorded for 1,000 random test images. In Figure 3(b), for 12 selected neurons the ten receptive field stimuli are shown which evoke the strongest activations throughout the considered data set, each ordered from top-left to bottom-right. Apparently, features characteristic for digits (line-endings, loops, and other specific stroke segments) are recognized in layer 3. Note that until this point no explicit knowledge about the classification task has entered the system. The investigations suggest that the proposed method of feature extraction in the intermediate layers can be very differentiated, such that—in the ideal case—the output layer merely has to choose from the features presented to it.

It has been stated that the MNIST training set may be too small to infer generalization properly, which gives raise to expand the data set by applying spatial deformations to the original training images [9], [17]. In this work, a deformation based on Gaussian displacement kernels is chosen, simply because a corresponding software program was available. In particular, each image from the MNIST training set is subject to the following transformation. For each pixel position $\mathbf{r}' = (x', y')$ of the transformed image, the corresponding position in the source image $\mathbf{r}$ is computed according to

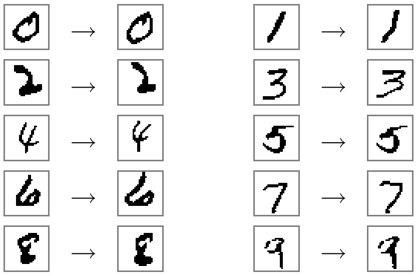


Fig. 4. Expanding the training data set by elastic transformations. Shown are examples of the original and transformed images (binarized versions).

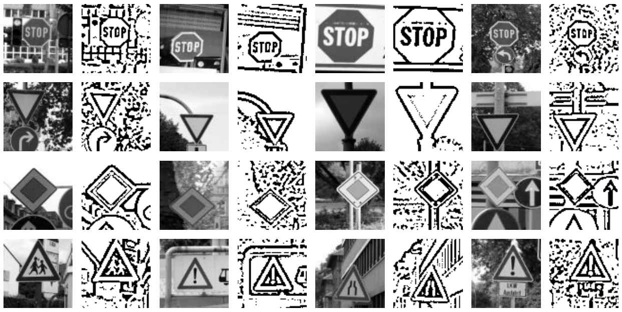


Fig. 5. Samples of the traffic sign data set. Each row shows samples of a different class. Each image is displayed as original grey-value image (left, 75x75pixels) and LoG preprocessed version (right, 73x73pixels) as fed into the network.

$$\mathbf{r} = \mathbf{r}' + \sum_{g=1}^{G} \mathbf{d}_g \exp\left(-||\mathbf{r}' - \mathbf{r}_g||^2/\sigma^2\right),$$

where $\mathbf{r}_g$ and $\mathbf{d}_g$ are the positions and maximum displacements of each of the G Gaussian kernels, and σ is the (uniform) kernel width. The pixel $\mathbf{r}'$ in the transformed image is assigned the grey-value of the original image at position $\mathbf{r}$. Grey-values at non-integer pixel positions are inferred by bi-linear interpolation. The constants $\mathbf{d}_g$ and $\mathbf{r}_g$ are drawn from a uniform distribution, separately for each image to be transformed. Every image from the training set is transformed, doubling the training set from 60,000 to 120,000. The parameters used are $d_g^{x,y} \in [-3, 3]$ (3 pixels maximum displacement), $r_g^{x,y} \in [0, 28]$ (every position on a 28x28 image is a possible kernel center), $\sigma = 8$, and $G = 3$ (3 Gaussian kernels used). These values were chosen *ad-hoc* by visual judgment of the transformation results, and have not been subject to optimization. Deformation is applied to the original grey-value MNIST images, before binarization. Examples of the transformed images are shown in Figure 4. With training the output units using the expanded training set, the average classification error on the test set goes down to 1.48% (see Table II).

2) Traffic Signs: In order to illustrate that the training method works invariant of position and scale, and also with noise present, the network is applied to photographs of traffic signs. 400 pictures of 4 classes of traffic signs (100 per class) were taken and cropped to 75x75 pixels dimension, while making sure that the images still show variances in shift and scale. The images are pre-processed as described in section II. The preprocessed images are 73x73 pixels in size. Figure 5 shows examples. The images are available from the authors on request. The data set is split randomly in two parts (300/100) serving as training and test data sets, respectively. Network parameters are shown in Table I, bottom two rows. Parameter optimization is done on the test set. This may be not 100% clean, but introcuding an extra validation set would have decreased the already small training set even further.

Here, the pairwise linear classifier used for the MNIST images does not produce good results, probably due to the smaller number of image classes. Instead, each output unit is trained to separate one given class from all the others. Per class, five units are trained, making an overall number of 4x5=20 output units. On the test set, voting among these units determines the answer of the network. If two or more classes receive an equal number of votes, the image is supposed to be mis-classified. Training and evaluation of the network is conducted ten times. The average, best, and worst achieved classification errors are 2.2%, 0.0%, and 4.0%, respectively. The standard deviation within the ensemble of ten networks is 1.0%.

B. Robustness against random weight perturbation

In analog computing devices various phenomena influence the accuracy of the computation result. In the following investigation it is assumed that variations in the network's weight values constitute the dominating effect in the considered hardware architecture. In particular, the effective synapse strengths are supposed to deviate from the values the chip was configured with by a random amount, drawn from a normal distribution of a given width. Throughout this paragraph, the unperturbed weight vectors are assumed to be scaled to unit maximum norm (the strongest weight has an absolute value of 1), and the thresholds are realized by one or many additional constant inputs with weights of absolute value not larger than 1.

Two scenarios are considered: In the first scenario, the weights of the hidden S-type layers of a *completely trained* network are perturbed and the influence on the classification performance is measured. This corresponds to loading a network fully trained in software onto the chip. The output layer is assumed to be evaluated in software and thus is not subject to perturbation. Results are shown in the diagram in Figure 6, upper curve. The shaded area corresponds to no perturbation. A perturbation width of 0.02 of the maximum weight value results in only a small performance loss. With stronger perturbations, the average error and sandard deviation both start to increase. Although in the chosen view the curve seems very steep, it should be noted that the network's performance never breaks down completely. Even when the width of the perturbation reaches as much as 50% of the maximum weight value, approx. 93% of all patterns are correctly recognized on average.

In the second scenario, an on-chip training approach is simulated: The weights of a layer are perturbed immediately after training that layer. Consecutive layers (including the output units) are trained with the patterns produced by the *already perturbed* previous layer. Since succeeding layers "learn" to live with the imperfections of the previous one, more robust behavior is expected as in the first scenario. Figure 6, lower curve, shows the results. Until a perturbation width of 0.2, the classification performance deviates only marginally from a network without perturbation (shaded area). Only very strong perturbations with a width of 0.5 result in significant performance loss. The on-chip approach is 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. The on-chip approach is not easily applicable when multiple chips are operated in parallel, because here an identical behavior of all chips is preferred.

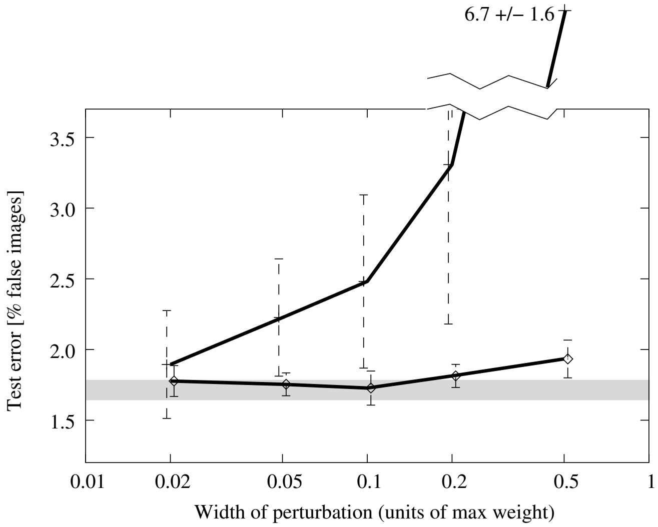


Fig. 6. Recognition performance on the MNIST test set with random weight perturbation. The shaded area corresponds to network performance without perturbation (1.71% ± 0.07%). Upper curve: Perturbation of fully trained network. Lower curve: Perturbation with on-chip training approach (see text). Errors shown are the standard deviations among 10 training runs. X-axis is logarithmic for convenience.

C. Weight quantization

The experiments in the previous paragraph provide evidence that layers trained with the clustering approach are quite robust against random weight perturbations especially when employing an on-chip training approach. For understanding where this high degree of robustness comes from, a look at the distribution of weights in the trained convolutional layers is helpful. Figure 7 (solid line) shows a bimodal distribution, as usually observed after Hebbian-type re-inforcement learning with a weight saturation condition [11] (cf., Formula (2) + normalization): Most synaptic weights are either near their maximum or near their minimum allowed weight value. As a result, the global preference of a neuron (whether it likes an input to be on or off) is rather little affected by small or medium weight changes. As an indication of this behavior, the overall shape of the weight distribution vanishes only for very strong perturbations (Fig-

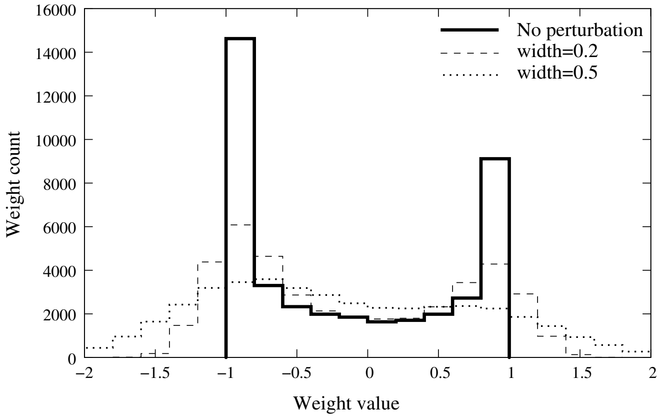


Fig. 7. Solid line: Distribution of synaptic weights of stages 1 and 3 after training (histogram bin-size 0.2). Weights are forced to the interval [-1,1] by scaling the weight vectors to unit maximum norm after training. Strong weight values (excitatory and inhibitory) dominate. This pronounced bimodal distribution is not easily destroyed by perturbation: Dashed line: After strong perturbation (width=0.2) Dotted line: After very strong perturbation (width=0.5).

ure 7, dashed and dotted lines).

These observations raised the question if weights being able to code only extreme values (-1, 0, +1) are sufficient for the implementation of the hidden layers. Such quantization reduces a neuron's task to plain pattern matching ("how many pixels are correct?") while requiring only most basic computing operations. Additional training runs were conducted where, after training, the weights of the hidden layers are quantized by the following scheme:

$$w \leftarrow \begin{cases} +1 & \text{if } w > 0.5 \\ -1 & \text{if } w < -0.5 \\ 0 & \text{else} \end{cases} .$$

Like in the previous experiments, the weight change is applied either to a completely trained network, or applied successively to each layer, where consecutive layers, including the output layer, are trained with patterns produced by the quantized layer(s) ("on-chip" approach). The results (Table II) confirm that the performance is not severely affected by the quantization. Again, as expected the performance loss is smaller when using the on-chip training approach.

All presented results were obtained using a software implementation of the described networks. This is in part 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 (cf., section IV). On the other hand, this way the properties of the network model and training method are evaluated isolated from the particularities of a concrete hardware substrate, and reproducability is ensured.

IV. Discussion and Outlook

It is shown that a neural network consisting of binary threshold neurons, suited for being implemented in mixed-signal low-power hardware, trained using simple strategies, can produce good results on two natural image classification tasks.

The presented methods are robust against random weight perturbations, especially when employing an on-chip training approach. This property facilitates the implementation in highly integrated low-power analog circuits which usually exhibit a certain degree of inaccuracy. During the experiments it turned out that even very coarse weight quantization does not severely affect the feature-extracting property of the hidden layers. Experiments using only three distinct weight values (-1, 0, +1) are able to produce results comparable to using continuous weight values. In networks where both the neuron's outputs and the weights are quantized in this rigorous way, the normal multiply-and-accumulate operations followed by the evaluation of an activation function reduce to mere counting and subsequent thresholding. It is remarkable that a powerful feature extraction system can work using only such simple computing instructions.

The error rates achieved on the MNIST data set do not yet quite reach the best rates reported for convolutional networks trained by back-propagation. The best values found are 0.95% [10] (without preprocessing the images, e.g., de-slanting, and without expanding the training set, e.g., elastic distortions), respectively 0.4% [17] (with expansion of the training set by elastic distortions). This may be due to information loss introduced when binarizing the images. For comparison, a continuous-valued linear classifier trained using MSE was found to achieve about 14.5% error rate on the binarized images in contrast to reported 12% for the unprocessed grey-value images [10]. However, the performance of the presented system can probably be improved, e.g., by optimizing the parameters for the elastic transformations or by employing a more sophisticated read-out mechanism than the pairwise linear classifier.

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. Ample evaluations of the hardware system (possible speed gain, power consumption per image, effects on classification performance) will be conducted. However, due to limitations of the prototype chip (a maximum of 128 inputs per neuron), the evaluation of large networks as shown in this work may have to be postponed for a future larger implementation of the architecture.

In the presented system, the neuron thresholds are not included in the automatic training, but are left as meta-parameters to be chosen appropriately. Although the choice turns out not to be too critical, it remains a time-consuming task. Methods will be evaluated to determine the optimal thresholds automatically.

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. He also likes to thank A. Kovatcheva for help with taking the traffic sign photographs. 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] K. Fukushima: Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks 1, 119-130 (1988)

[3] S. Haykin: Neural networks: a comprehensive foundation. 2nd ed., Prentice Hall, New Jersey (1999)

[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] B. Jähne: Digital Image Processing. 6th ed., Springer Verlag Berlin, Heidelberg, New York (2005)

[7] 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)

[8] 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)

[9] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324 (1998)

[10] Y. LeCun: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist

[11] R. Linsker: From basic network principles to neural architecture. (Series of 3 papers) Proc. Natl. Sci. USA 83, 7508-7512 (1983)

[12] C. Neubauer: Evaluation of convolutional neural networks for visual recognition. Transactions on Neural Networks 9(4), 685-696 (1998)

[13] M.W. Oram, D.I. Perret: Modeling visual recognition from neurobiological constraints. Neural Networks (7) 945-972 (1994)

[14] M. Riesenhuber, T. Poggio: Hierarchical Models of Object Recognition in Cortex. Nature Neuroscience 2 , 1019-1025 (1999)

[15] D. E. Rumelhart, D. Zipser: Feature discovery by competitive learning. Cognitive Science, 9, 75-112 (1985)

[16] 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)

[17] 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)

[18] K. Tanaka: Inferotemporal cortex and object vision. Annu. Rev. Neurosci 19, 109-139 (1996)

[19] J. Teichert, R. Malaka: A Component Association Architecture for Image Understanding. Lecture Notes in Computer Science 2415 (ICANN 2002), 125-130.

[20] S. Ullmann, S. Soloviev: Computation of pattern invariance in brain-like structures. Neural Networks 12, 1021-1036, (1999)