VLSI IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK FOR A REAL TIME DETECTION PROBLEM

Marco Krips, Anton Kummert

Communication Theory, Department of Electrical and Information Engineering, University of Wuppertal, Rainer-Gruenter-Strasse 21, 42119 Wuppertal, Germany, phone: +49-202-4391961, e-mail: krips@uni-wuppertal.de

ABSTRACT

Real time image processing is an interesting field of application for artificial neural networks as well as for very large scale integrated (VLSI) circuits. This paper shows how the advantage of parallel computing of neural network algorithms can be realized by the potentials of VLSI circuits in order to design a real time detection and tracking system for video image processing.

A real time localization and tracking algorithm has been developed for detecting human hands in video-images. Due to the real time aspect, a single-pixel-based classification is aspired, so that a continuous data stream can be processed. Consequently, no storage of full images or parts of them is necessary. The classification, whether a pixel belongs to a hand or to the background, is done by analysing the RGB-values of a single pixel by means of an artificial neural network. To obtain the full processing speed of the neural network a parallel operating hardware solution is realised.

1. INTRODUCTION

Safety aspects in industrial facilities are of growing importance. In some fields approved methods like safety light barriers or safety light curtains are not applicable. An example for such a complicated workplace can be found at a press brake, where workpieces have to be handled even close to the danger area. The herein before mentioned systems would make it impossible to guarantee an undisturbed workflow. Hence, new methods for workplace safeguarding are needed. One basic approach will be introduced subsequently.

In this study a safeguard for a press brake's workplace will be investigated. The tracing is done by a video based system that detects human's hands, which are in danger mostly, and evaluates if there is a risk of bodily harm for the operator. This paper is limited to the discussion of the algorithms for the detection problem and of a hardware implementation of these algorithms.

Based on its colour, a real-time localization algorithm has been developed. In contrast to many other industrial applications reference image techniques turned out to be less successful for our problem. This is due to the fact that many different hand shapes have to be compared, whereas in other applications the actual position of workpieces can be compared easily with reference positions. Certainly, the use of a large database filled with reference images could be a solution, however, real-time processing would be impossible with such an approach.

Taking into account short admissible processing times, we concentrated our research on pixel based operations. In our system, each pixel provided by a video camera is classified, whether it belongs to a hand or to the background. The classification is done by analysing the RGB-values by means of an artificial neural network.

The theory of neural networks is not discussed in detail since this would be out of the scope of this work. Nevertheless, references are added for further information.

2. ARTIFICIAL NEURAL NETWORK

There exist many types and architectures of neural networks (NN), e.g. Perceptron [1], RBF [2] or self-organizing map [3]. It can be shown that several architectures and network classes, respectively, are eligible for the presented hand-detection problem. Nevertheless, regarding to the best ratio of performance to network dimension (hardware effort) this paper focuses on special multilayer feedforward network structures.

2.1. Network architecture

The neuron is the basic information processing unit that is fundamental to the operation of a neural network. Figure 1 shows the model of a neuron as used in this work. Attention should be paid to the handling of the "bias" value b. The latter is taken into account as an additional weighted input signal of the neuron. This input signal has the constant value "1" and is multiplied by the weight $w_{nj} = b_j$. So one gets a simplified mathematical description of the neuron.

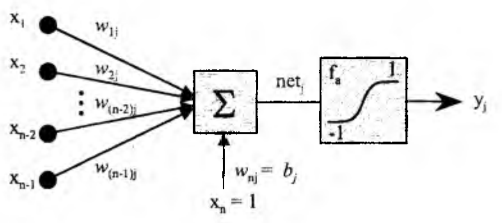


Figure 1. Nonlinear model of a neuron

In mathematical terms, we can describe a neuron j by writing the following set of equations:

$$net_j = \sum_{i=1}^{n} x_i w_{ij} \qquad (1)$$

and

$$y_j = f_a(net_j) \qquad (2)$$

where x_1, x_2, ..., x_n are the input signals; w_{1j}, w_{2j}, ..., w_{nj} are the synaptic weights of neuron j; net_j is the signal after the summing node; f_a is the nonlinear activation function; and y_j is the output signal of the neuron.

Next, the network architecture has to be examined. A multilayer feedforward network consists at least of an input and an output layer. Between these, several hidden layers can occur. Each layer can consist of several neurons. The architecture of the neural networks used in our context is presented in the following. First of all, the input layer has to be considered.

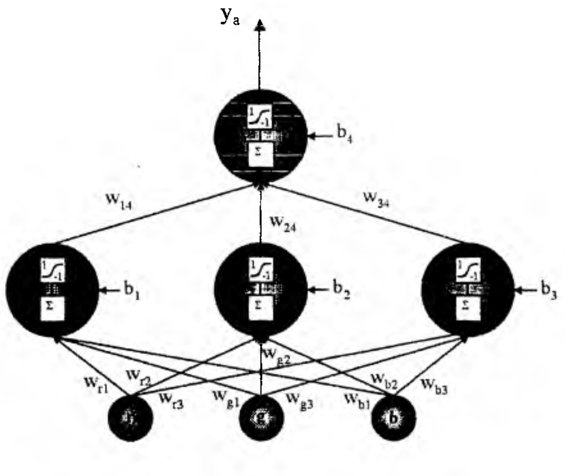


Figure 2. Architecture of the neural network

The number of input neurons is specified by the basic raw data, which is provided by a video camera. There are 3 neurons needed for the three colour values (RGB) of each single pixel. It can be shown that this colour representation is to be most appropriate to detect human hands compared to other systems e.g. the Yuv colour space [4]. As mentioned before we restricted our analysis to single pixel-based algorithms. Therefore, the result of the neural network should be a 1 if the presented pixel has the colour of a hand and 0 otherwise. Hence, only one output neuron is needed.

The next aspect, which has to be investigated, is the number of hidden layers and neurons within these layers. Tests proved that even neural networks consisting of one hidden layer with three neurons produce outstanding results as shown subsequently.

A scheme of the basic architecture is shown in Figure 2. For easier understanding the biases are not drawn as additional weighted inputs, but as "real" biases.

Besides the architecture, the learning algorithm used for training is important for convergence speed and the results of the neural network, respectively.

2.2. Supervised Learning

The most common algorithm for multilayer feedforward network training is backpropagation. Since this algorithm has been presented in [5], it has been modified in various aspects. In this context only a few algorithms should be mentioned, e.g. conjugate gradient algorithms [6] or the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update [7]. The advantage of all these different variations of the standard backpropagation algorithm relies on the fact that they are mostly more efficient concerning a better convergence and reduced training time, respectively. During the development phase for designing the neural networks by means of the software tool MATLAB® two variants proved to be the best solutions for our problem, namely the Marquardt-Levenberg algorithm [8] and RPROP [9].

The Marquardt-Levenberg algorithm (ML) is an approximation to Newton's method while backpropagation is a steepest descent algorithm. It minimizes the sum of squares of errors over all inputs. For Newton's method the weight-updates are computed by

$$\Delta \mathbf{w} = -\left[\nabla^2 E(\mathbf{w})\right]^{-1} \nabla E(\mathbf{w}) \qquad (3)$$

where $\nabla^2 E(\mathbf{w})$ is the Hessian matrix and $\nabla E(\mathbf{w})$ is the gradient of the error function $E(\mathbf{w})$ that depends on the parameter vector $\mathbf{w}$ and has the form of a sum of squares. The Marquardt-Levenberg algorithm avoids computing the Hessian matrix. Instead, an approximation by means of the Jacobian matrix is used. The Jacobian matrix can be computed by a standard backpropagation technique that is much less complex than computing the Hessian matrix.

The weight-updates are computed accordingly to

$$\Delta\mathbf{w} = \left[J^T(\mathbf{w})J(\mathbf{w}) + \mu\mathbf{I}\right]^{-1} J^T(w)\mathbf{e}(w) \tag{4}$$

where $J(\mathbf{w})$ is the Jacobian matrix. The weight updating can be described as follows:

Whenever a step would result in an increased $E(\mathbf{w})$ the parameter μ is increased by a constant factor β and it is decreased by division by β, whenever a step reduces $E(\mathbf{w})$. So, if μ is large, the algorithm becomes similar to steepest decent, whereas for small μ the algorithm becomes similar to Gauss-Newton.

A further algorithm has been'analysed for the learning task namely the resilient propagation (RPROP) algorithm. RPROP performs a local adaptation of the weight-updates $\Delta\mathbf{w}$ according to the behaviour of the error function.

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & ,\text{if } \dfrac{\partial E(t-1)}{\partial w_{ij}}\dfrac{\partial E(t)}{\partial w_{ij}} > 0 \wedge \dfrac{\partial E(t)}{\partial w_{ij}} > 0 \\ \Delta_{ij}(t) & ,\text{if } \dfrac{\partial E(t-1)}{\partial w_{ij}}\dfrac{\partial E(t)}{\partial w_{ij}} > 0 \wedge \dfrac{\partial E(t)}{\partial w_{ij}} < 0 \\ -\Delta w_{ij}(t-1) & ,\text{if } \dfrac{\partial E(t-1)}{\partial w_{ij}}\dfrac{\partial E(t)}{\partial w_{ij}} < 0 \\ -\operatorname{sgn}\left(\dfrac{\partial E(t)}{\partial w_{ij}}\right)\Delta_{ij}(t) & ,\text{else} \end{cases} \tag{5}$$

By this, an individual update value Δ_{ij} is computed for each weight, which does not depend on the unforeseeable influence of the magnitude of the partial derivative but only depends on the behaviour of its sign.

$$\Delta_{ij}(t) = \begin{cases} \eta^{+}\Delta_{ij}(t-1) & ,\text{if } \dfrac{\partial E(t-1)}{\partial w_{ij}}\dfrac{\partial E(t)}{\partial w_{ij}} > 0 \\ \eta^{-}\Delta_{ij}(t-1) & ,\text{if } \dfrac{\partial E(t-1)}{\partial w_{ij}}\dfrac{\partial E(t)}{\partial w_{ij}} < 0 \\ \Delta_{ij}(t-1) & ,\text{else} \end{cases} \tag{6}$$

The update value is increased by the factor η^{+} whenever the partial derivative of the error function has the same sign for two successive iterations. The update value is decreased by the factor η^{-} whenever the partial derivative changes its sign. The update value remains the same if the derivative is zero. When the weights are oscillating, the weight change will be reduced. If the derivative retains its sign, the update value itself is slightly increased in order to accelerate convergence in shallow regions.

While ML and RROP are suitable for supervised learning, a sample image and its corresponding output-reference or teacher, respectively, is needed. Due to the aspect that the network learning is done by a software based simulation tool on a workstation it is not recommendable to use a whole frame since this would lead to enormously long learning times. For this, a sample image is clipped from a whole frame provided by the video camera.

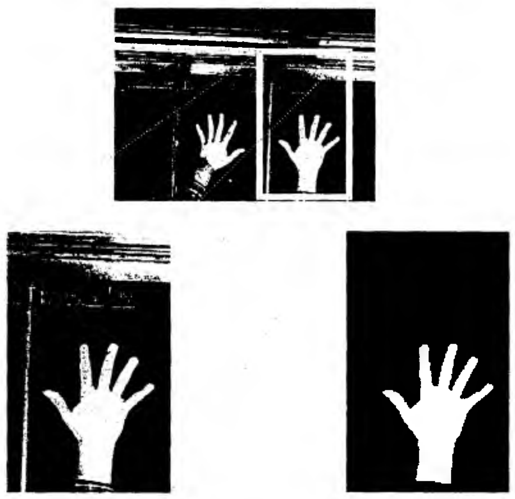


Figure 3. Clipped training pattern and corresponding output reference (teacher) image

The next step is the extraction of the output reference (teacher). For doing this the binary image is manually created by means of graphic software.

During learning the sample image is presented to the network recurrently. After each recursion an error signal is computed and propagated backwards through the network to adjust the weights and biases. As performance function the mean square error (MSE) was used. The Marquardt-Levenberg algorithm (MSE = 0.00521778) provides a slightly better result as RPROP (MSE = 0.00566031) as presented in [10].

Nevertheless, RPROP has the advantage that it needs less memory during training and additionally generates weights, which can be better modified for implementation. Therefore we used RPROP for network training.

A discussion of the graphical results follows in section 4 together with the results of the hardware realization of the neural network.

3. VLSI Design

The model of the neural network for hardware implementation has been described by VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language) [11]. The choice of VHDL was made due to its flexibility and the fact that it is a standard language (IEEE 1076). Furthermore, it is a target device independent designing tool, i.e. with one design description many device architectures can be included. For testing we used an Field Programmable Gate Array (FPGA). The advantage of an FPGA can be seen in the fact that it can be easily reconfigured since its configuration will be lost if power supply is switched off. Furthermore the latter becomes more and more standard solution in different application fields, e.g. research and development, prototyping and areas where short "time to market" windows or updating via Internet are desirable.

Naturally, hardware implementations require modifications of the neural network architecture compared to software solutions.

First, the data type of the neural network's weight and bias coefficients has to be changed from real type into integer. The latter can be used immediately in VHDL. To be more specific, it can be easily represented as std_logic_vector type, which is the most suitable data type for digital processing. Secondly the nonlinear activation function must be modified. In the software based neural network the hyperbolic tangent function (tanh) has been used. This function is not a standard object for an implementation in an FPGA. Therefore, an approximation has to be applied.

Moreover the input signal range has to be adjusted to a defined 8 Bit representation of the RGB-signals i.e. to the interval [0, 255]. In contrast to this the software-based neural network was computed for an input signal range [0, 1], hence the factor 255 has to be taken into account for the weights and the activation function.

3.1. Neural Network Modifications

The first aspect, which has to be handled, is the data type conversion. It is insufficient to round the weights directly since many coefficients lie in the interval [0, 1]. So the rounding operation would cause the elimination of connections by rounding the weights to zero, if $w_{ij} < 0.5$. In order to avoid this, weights are multiplied by a constant factor α.

In other words coefficient values in the interval [0, 1] are transformed into (α+1) quantization levels. Thus the coefficients in the integer data type are computed by

$$w_{\text{int}} = round(\alpha \cdot w_{real}) \ . \tag{7}$$

The second problem for hardware implementation is the tanh-function. Obviously, a classical solution to this problem is the use of a look-up-table (LUT). Since this LUT is required for every neuron of the neural network its size has major influence on the required gate size of the FPGA. Resource sharing is not an acceptable option due to parallel processing and the very short processing times, which are aspired. Consequently, we concentrated our attention on the minimization of the LUT.

The tanh- function has been linearly quantized to provide many quantization levels around zero and less at the extreme values 1 and –1. This behaviour is worthwhile since a special sensitive fragmentation around zero is needed whereas this is not necessary in upper and lower saturation regions. Furthermore the latter can be separated by the borders –L and L, for which we defined L=2.6467 which corresponds to 99% of the maximum value 1. This means that the function has to be quantized in the input interval [-2.6467, 2.6467] only. Other arguments result in –1 and 1, respectively.

Another important aspect, which has to be taken into account, is to take care that zero gets its own quantization level. This is required because a zero output represents a "non-existent connection". A different behaviour of the LUT would lead to serve errors since an "additional connection" would be added.

The number of quantization levels has been fixed on the basis of tests with different LUTs each with its own quantization level for zero. As Criteria for quality the following value Q_a was defined

$$Q_a = \frac{1}{N} \sum_{x=-N/2}^{N/2} \left[LUT\left(\frac{x}{1000}\right) - \tanh\left(\frac{x}{1000}\right) \right]^2 \tag{8}$$

where LUT(x) is the output of the LUT for the given argument x and N is the number of samples.

The results given in Table 1 show that even a LUT with only 15 (LUT 15) quantization levels provides sufficient results so that larger LUTs with

31 or 511 levels are not necessary. These results are validated by the binary output results [10] of the neural network as well. For every neuron a separate LUT is needed on the chip hence, a LUT as small as possible is recommendable as mentioned before. In our case the network consists of 4 neurons this means that also 4 LUTs are needed.

Table 1. Quality of LUTs (N=12000)

	LUT 15	LUT 31	LUT 511
Q_a	1.2E-3	2.07E-4	4.74E-6

Figure 4 shows the characteristic of the LUT 15 for implementation, which takes all necessary modifications, as different input range of the RGB-values [0, 255] instead of [0, 1], $\alpha = 10$, and the new output range [-255, 255], into account. This LUT is used for the hidden neurons.

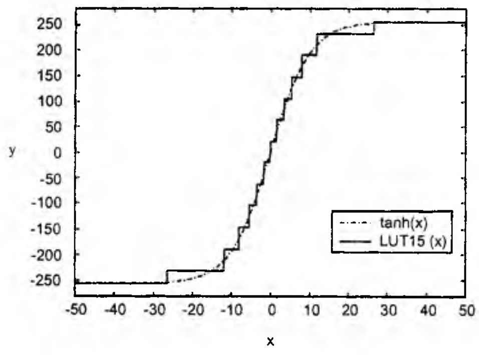


Figure 4. Approximation of tanh

For the output neuron the range of output values of the LUT has been reduced to "0" and "1". So a further processing unit, a threshold decision-maker, can be saved, i.e. the threshold is defined within the output LUT.

4. Detection Results

The evaluation of binary result images of the neural network is based on a comparison between optimal (used for training) and simulated results by using the quality measure

$$\gamma_e = \sum_i \sum_j XOR(t_{i,j}, r_{i,j}) \qquad (9)$$

where $t_{i,j}$ is the pixel of the corresponding output reference (teacher) and $r_{i,j}$ is the equivalent pixel of the computed result image. The value γ_e is called error subsequently.

4.1. Binary Result Images

The binary result image of the ideal network as used in software based simulations includes 199 errors (Figure 5 b). These errors correspond to an error rate of 0.702 % with respect to the learning sample image (Figure 5 a). The results of the implementable NNs are slightly inferior to the original NN. The error values γ_e= 221 (LUT 15) and γ_e= 204 (LUT 511) correspond to an error rate of 0.780 % and 0.720 %, respectively. The modifications applied after the learning phase do not affect the performance of the NN.

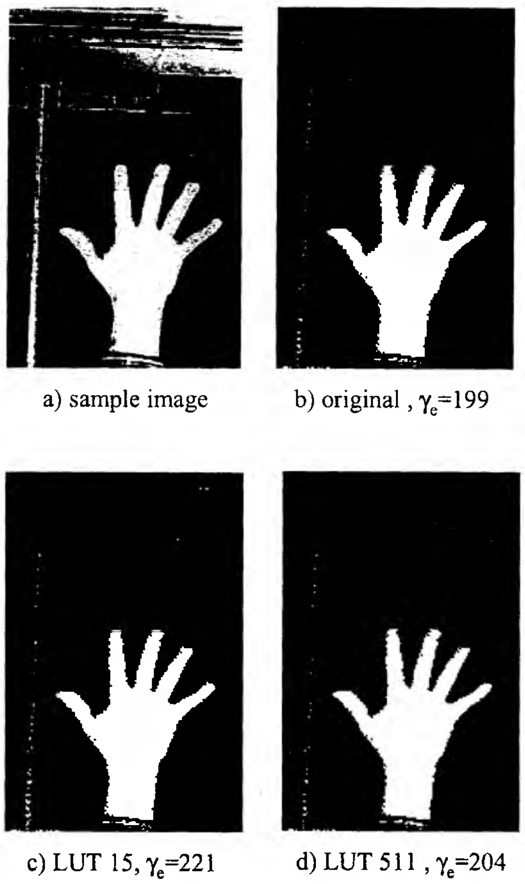
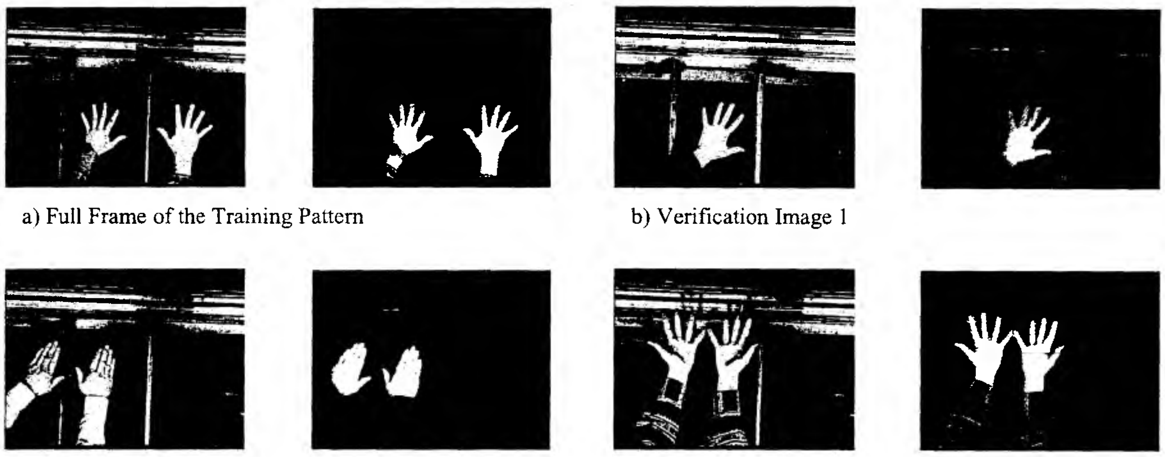

a) sample image
b) original , γ_e=199
c) LUT 15, γ_e=221
d) LUT 511 , γ_e=204

Figure 5. Binary result images (training pattern)

The results in Figure 5 show that there are a few parts marked as belonging to the hand, which definitely do not match. However, these misclassifications are of size 1 to 4 pixels in diameter. So these errors can be deleted by means of morphological operations like opening, erosion and dilation [12]. The morphological operations are performed very quickly, since they are applied to binary images



a) Full Frame of the Training Pattern

b) Verification Image 1

c) Verification Image 2

d) Verification Image 3

Figure 6. Detection results of full images: Image including training pattern (a), Images including non-trained hands (b, c, d)

only. So all regions outside the hand representing classification errors can be deleted without affecting the detection of the hand.

4.2. Generalisation of the Detection Problem

Generally, learning strategies could lead to solutions, which only work well for inputs from the learning data set whereas other inputs could lead to misclassifications. Nevertheless, the neural network should be useable for many different images of the same acquisition situation and detection always should be possible. This is called generalisation. This aspect can be easily checked by presenting different images to the neural network and analysing the results.

In Figure 6 a) the binary output of the neural network for the full input frame (288×384 pixels) of the clipped training pattern is shown. It can be seen that the detection of both hands is possible.

In addition, the generalization behaviour has been tested with pairs of hands that have not been trained. The results achieved by the NN are also excellent (Figure 6 b, c, d). Furthermore, the classification errors in each binary output can be deleted by morphological operations, which have not been applied yet.

4.3. Hardware Results

The results provided by the FPGA realisation of the neural network and those given by the original MATLAB® based solution without any modifications applied, differ in less than 0.02% for an image of size 288×384.

Certainly, the processing times depend on the target device. Processing time defines the time to get the result for a whole frame without applying morphological operations.

Table 2. FPGA utilization and processing times

Type of FPGA	**xc4013e**	**xcv100**
CLBs / Slices	**574 CLBs**	**418 Slices**
Utilization	**99.7 %**	**34 %**
Delay t_{delay}	**408.6 ns**	**71 ns**
Processing time t_{proc}	**< 45 ms**	**< 10ms**

Due to the VHDL entry of the VLSI circuit design the target device can be changed easily. In Table 2 the results of two implementations are compared. It reflects that as target device not only the latest technology is necessary, but that it would improve enormously the performance of the detection system. The detection system in terms of the neural network can be implemented in a Xilinx® xc4013e FPGA as well as in a Xilinx® Virtex® xcv100. The detection results concerning the error y_e are identically. The differences appear in the delay time t_{delay} and the processing speed. The delay between input signal and corresponding output signal differ in about 337 ns between the two realisations (Table 2). The processing time t_{proc} for a whole image of 288×384 pixels is less than 45 ms for the xc4013 and for the Virtex® implementation t_{proc} is 4.5 times less, respectively.

Nevertheless, the use of recent technology with a hardware overhead is recommended since the morphological operations have to be implemented additionally and last but not least strategies for system testing [13] have to be build in as well. This is an important issue as this detection system can be used as one component in a video-based safety facility. So testing is not only important during designing but also during the normal duty to guarantee accurate detection and tracking for the whole lifetime of the system.

5. Conclusion

In this paper a pretentious detection and tracking problem is presented, which can be solved in real-time by using a neural network. The presented NN permits reliable real-time detection combined with a very compact network architecture. It is obvious that a challenging problem like hand tracking cannot be solved completely by simply applying a neural network. However, the misclassifications can be eliminated by a further processing step, namely binary morphological operations.

A hardware solution using an FPGA has the big advantage that the fundamental idea of a NN, the parallel information processing, can be used. Moreover, a fast update of the FPGA's configuration can be done e.g. with new coefficients, if a new training is necessary.

The presented system is only the first step of the implementation of a neural network in an FPGA. The next step is the integration of the hardware into the training phase to speed up the learning process and to increase the flexibility of the system.

References

[1]. Minsky M., Papert S.: Perceptrons - An Introduction to Computational Geometry. MIT Press, Cambridge, Mass., 1969

[2]. Haykin S.: Neural Networks - A Comprehensive Foundation. Macmillan College Publishing Company, New York, 1994

[3]. Kohonen T.: The Self-Organizing Map. Proc. of the IEEE, Vol.78, No.9, 1990, pp. 1464-1477

[4]. Littmann E., Ritter H.: Adaptive Color Segmentation - A Comparison of Neural and Statistical Methods. IEEE Trans. on Neural Networks, vol. 8, 1997, pp. 175-184

[5]. Rumelhart D.E., McClelland J.L.: Parallel Distributed Processing - Exploration in the Microstructure of Cognition. Vol.1: Foundations, MIT Press, Cambridge, Mass., 1986

[6]. Hagan M.T., Demuth H.B., Beale M.H.: Neural Network Design. PWS Publishing, Boston, Mass., 1996

[7]. Dennis J.E., Schnabel R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice Hall, Englewood Cliffs, NJ, 1983

[8]. Hagan M.T., Menhaj M.B.: Training Feedforward Networks with the Marquardt Algorithm. IEEE Trans. on Neural Networks, Vol.5, No.6, 1994, pp. 989-993

[9]. Riedmiller M., Braun H.: A direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. Proc. of the International Conference on Neural Networks, 1993, pp. 586-591

[10]. Krips M., Kummert A., Velten J.: FPGA-Implementation of a Neural Network for real-time hand-detection in video-images, Proc. of International Conference on Signal Processing Applications & Technology (ICSPAT'00), Dallas, Tx, USA, 2000

[11]. Ashenden P.J.: The Designer's Guide to VHDL. Morgan Kaufmann Publishers, San Francisco, 1999

[12]. Haralick R.M., Sternberg S.R., Zhuang X.: Image Analysis Using Mathematical Morphology. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-9, No.4, 1987, pp. 532-550

[13]. Yarmolik V.N.: Fault Diagnosis of Digital Circuits. John Wiley & Sons, Chichister, England, 1990