



Modified high-order neural network for invariant pattern recognition

Evgeny Artyomov, Orly Yadid-Pecht *

The VLSI Systems Center, Ben-Gurion University, P.O. Box 653, Beer Sheva 84105, Israel

Received 3 March 2003; received in revised form 20 August 2004

Abstract

A modification for high-order neural networks (HONN) is described. The proposed modified HONN takes into account prior knowledge of the binary patterns that must be learned. This significantly reduces hence computation time as well as memory requirements for network configuration and weight storage. An “approximately equal triangles” scheme for weight sharing is also proposed. These modifications enable the efficient computation of HONNs for image fields of greater than 100×100 pixels without any loss of pattern information.

© 2004 Elsevier B.V. All rights reserved.

Keywords: HONN; High-order neural networks; Invariant pattern recognition; Binary patterns

1. Introduction

Invariant pattern recognition is known as a highly complex and difficult problem. Different techniques were developed to cope with this problem, which can be divided to two distinguished stages (Wood, 1996): invariant feature extraction and feature classification. An example of the former is the moment descriptors (Wong et al., 1995), and of the latter is the multilayer perceptron (Chaudhuri and Bhattacharya, 2000). In some

algorithms, which implement neural networks, these two stages can be combined. Invariant pattern recognition using neural networks is a particularly attractive approach because of its similarity with biological systems.

Neural networks used for invariant pattern recognition are classified by the way that invariance is achieved (Barnard and Casasent, 1991): invariance by training, invariant feature spaces, or invariance by structure. Invariance by training is achieved by presenting the network a predefined set of patterns under different transformations, such as rotation (Wood, 1996). The second class, known as invariant feature spaces, works by classifying invariant features (Mahmoud and Mohamed, 2000). Invariance

* Corresponding author. Tel.: +972 8646 1512; fax: +972 8647 7620.

E-mail address: oyp@ee.bgu.ac.il (O. Yadid-Pecht).

by structure is achieved by the structure of the neural network; good examples are the recognition (Fukushima, 2001) and high-order neural networks (HONN) (Spirkovska and Reid, 1992). This article will concentrate on the HONN and its modifications.

In third-order networks, which are a special case of the HONN, invariance is built into the network structure. This enables fast network learning with only one view of each pattern presented at the learning stage. However, the number of interconnections in the network grows exponentially, and this prevents the use of this method for image fields larger than 18×18 pixels (Spirkovska and Reid, 1992).

Several other solutions have been proposed to minimize the number of the HONN interconnections. Weight sharing by similar triangles (Spirkovska and Reid, 1992) reduces the number of individual weights and achieves translation, rotation and partial scale invariance. Weight sharing by “approximately similar triangles” (Perantonis and Lisboa, 1992; He and Siyal, 1999; Takano et al., 1994) further reduces the number of individual weights and achieves translation rotation, limited scale, and limited distortional invariance. Coarse coding (Spirkovska and Reid, 1993) reduces the input field by obtaining a few coarsely coded images from the original image. In non-fully interconnected HONN (Spirkovska and Reid, 1990), the number of interconnections is reduced by subsampling. A neural network that could learn high-order correlations was also proposed (Guler, 2001).

Except for coarse coding, all these methods partially solve the problem of the HONN interconnections, but still do not work with larger images. With the coarse coding scheme, on the other hand, very large input image fields can be computed, but the network loses its capacity to learn patterns after its size is reduced. Consequently, the research community in the field of invariant pattern recognition largely abandoned the HONN method.

In this paper, a modification for the third-order network is described. The proposed modification takes into account prior knowledge of the binary patterns that must be learned. By eliminating idle loops, the network achieves significant reduc-

tions in computation time as well as in memory requirements for network configuration and weight storage. Better recognition rates (compared to conventionally constructed networks with the same input image field) are attained by the introduction of a new “approximately equal triangles” scheme for weight sharing. The modified configuration enables efficient computation of image fields larger than 100×100 pixels without any loss of image information—an impossible task with any previously proposed algorithm.

2. HONN architecture

The output of a third-order network can be described by the following equation:

$$y_i = f \left(\sum_a \sum_b \sum_c w_{iabc} x_a x_b x_c \right) \tag{1}$$

where i is the output index, w is the weight associated with a particular triangle, y is the actual output, x is a binary input, and a , b , and c are the indices of the inputs.

A schematic description of this network is shown in Fig. 1.

In the training phase, a perceptron-like rule is used:

$$\Delta w_{iabc} = \eta (t_i - y_i) x_a x_b x_c \tag{2}$$

where t is the expected training output, y is the actual output, η is the learning rate, and x is a binary input.

The best-known problem of the third-order network is that the number of possible triangles grows exponentially with image size. Each triangle has a different weight and thus a large number of

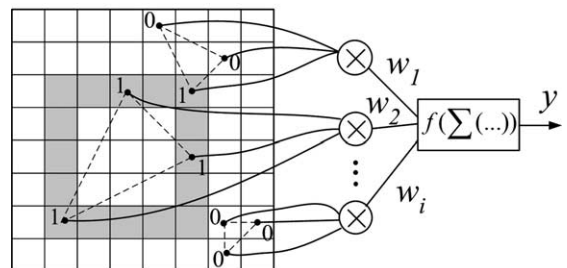


Fig. 1. Schematic description of a third-order network.

Table 1
The number of triangles as a function of input image size

| Input image size | Number of triangles |
|------------------|-------------------------|
| 4×4 | 560 |
| 8×8 | 41 664 |
| 12×12 | 487 344 |
| 20×20 | 10 586 800 |
| 40×40 | 681 387 200 |
| 100×100 | 1.6662×10^{11} |
| 256×256 | 4.6910×10^{13} |

weights must be stored. The number of triangles (NoT) can be calculated with the following equation:

$$\text{NoT} = \frac{\text{IN}!}{(\text{IN} - 3)!3!} \quad (3)$$

where IN is the number of input nodes. The number of possible triangles for different input sizes is given in Table 1.

As can be seen in Table 1, the number of triangles quickly exceeds the limits of any current hardware. A few techniques to reduce the number of weights have been proposed in the literature (as described in Section 1), but they do not reduce computation time. Conversely, the coarse coding technique of Spirkovska and Reid (1990) significantly reduces the input field (and thus the computation time), but the reduced net adversely affects its network learning capacity.

The problem of large computational demands arises for HONN methods because the network is constructed in the preprocessing stage before the learning phase. At this stage, all possible triangles are computed and pointers to the weights are stored (He, 1999; Spirkovska and Reid, 1992), along with the array of weights themselves. Thus, a minimum of two memory bytes is required for each pointer. For example, an input field of 40×40 pixels will require 1.3628×10^9 bytes to store the entire vector of pointers. The memory and computation requirements are enormous. To work with large input patterns, significant network modifications are required.

3. The proposed modified HONN method

To improve the network efficiency, a modified HONN is proposed. This network is not intended

for grayscale input patterns, so the input image is binary: pixels corresponding to an edge or contour of the object have the value “1” and all other pixels have the value “0”.

Using Eq. (1), each value of “0” gives a product of “0”. Therefore, only active triangles—those with all three pixels corresponding to an object edge or contour—can influence the result. In addition, the weights of inactive triangles are not updated and remain “0” during the learning process.

Following this observation, the network is modified to disregard all inactive triangles during the construction phase, thereby eliminating idle loops from the computation. With this modification, the network configuration depends strictly upon the input patterns that must be learned.

In addition, an “approximately equal triangles” scheme of network construction is introduced to improve the network performance. This scheme, combined with the “approximately similar triangles” scheme presented by Perantonis and Lisboa (1992) for weights sharing, means that approximately similar triangles with approximately equal areas will share the same weight. Theoretically, this scheme will negatively influence scale invariance, but it will greatly improve network performance with regard to rotation, distortion, and number of learned classes.

3.1. Triangle classification method

For the association of a triangle with a particular weight, the sets of possible values of the two smallest triangle angles (α, β) and the triangle area (S) are partitioned into bins defined by

$$(k - 1)w \leq \alpha < kw \quad (4a)$$

$$(l - 1)w \leq \beta < lw \quad (4b)$$

$$(m - 1)s \leq S < ms \quad (4c)$$

where w is the angular tolerance, α and β are the smallest angles of the triangle such that $\alpha \leq \beta$, s is the triangle area tolerance, S is the area of the computed triangle, and k , l , and m are the bin indices associated with the two angles and the triangle area, respectively.

Each triangle class is associated with a corresponding weight and is represented by the three variables k , l , and m . The number of individual weights depends strictly on the angular tolerance w and the triangle area tolerance s . Larger values of w and s result in a smaller number of weights. When the “approximately similar triangles” scheme is used alone, triangle area computation is not required and m is omitted.

3.2. The proposed network construction

The modified algorithm for network construction can be described as follows:

1. All the patterns that must be learned are loaded.
2. For each pattern, an array consisting of the x and y coordinates of the object contour (or boundary) pixels is constructed. The number of pixels that belong to the contours of an object is usually a small percentage of the total number of image pixels. A set of such arrays for different input patterns is shown in Fig. 2.
3. Using the arrays of the pixel coordinates, the angles and areas of all the presented triangles are computed and classified to associate each of them with a particular weight. To compute these values for each triangle, the three corner coordinates must first be selected from the array. The following code insures that all possible triangles are taken into account:

```

for ( $a = 0$ ;  $a < M - 2$ ;  $a++$ )
  for ( $b = a + 1$ ;  $b < M - 1$ ;  $b++$ )
    for ( $c = b + 1$ ;  $c < M$ ;  $c++$ )
      {
// triangle angle and area computation based
// on three coordinates:
// ( $x_a, y_a$ ), ( $x_b, y_b$ ), ( $x_c, y_c$ ).
...
// triangle classification
...
      }

```

where a , b and c are indices of the array of coordinates corresponding to a pixel index, M is the number of contour pixels in the particular pattern.

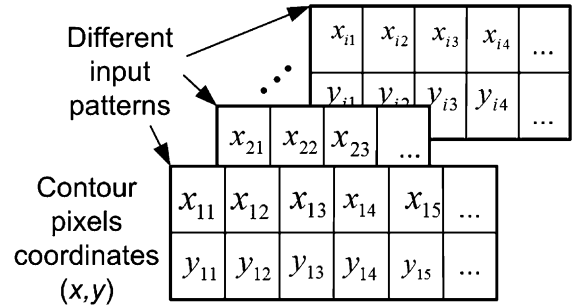


Fig. 2. Arrays of the pixel coordinates of the object contours. The indices of x_{ij} and y_{ij} are pattern number (i) and pixel index (j).

During the classification step (described in Section 3.1), each triangle class is associated with a corresponding weight and is represented by three variables k , l , and m . The array of triangle classes is constructed as shown in Fig. 3.

The variable n is the number of triangles from the particular pattern that correspond to the particular triangle weight index (class). The subscripts in n_{ij} correspond to pattern number (i) and weight index (j). The parameters α and β are the two smallest angles of each triangle, S is the triangle area, and k , l , and m are the bin indices associated with two angles and the triangle area, respectively.

After construction, only the array of triangle classes presented in Fig. 3 must be stored in memory. Two bytes are sufficient to save each number in the array, and in most cases, one byte is enough to store the integers k , l , and m (depending on the angle and area tolerances used).

3.3. Network training

The previously constructed array of triangle classes (Fig. 3) is used as the basis for learning in the training phase.

The algorithm for network training can be described as follows:

1. Zero array of weights $W = (w_{ij})$ with the size of $\text{NoP} \times \text{NoW}$ is constructed.
2. Output y_i is computed according to the following equation:

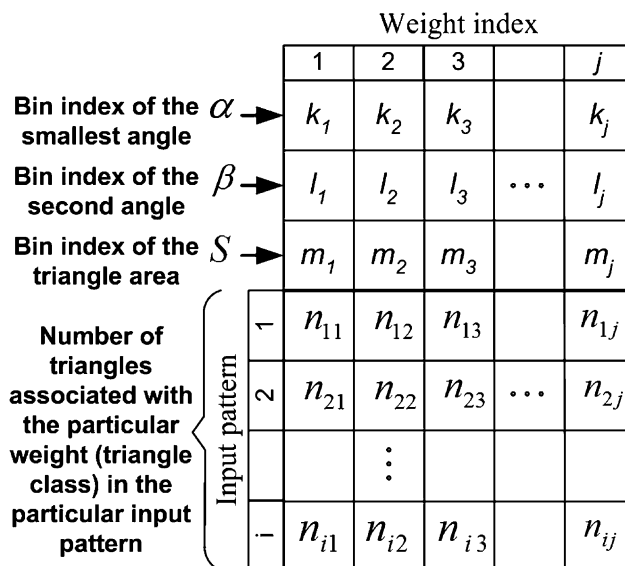


Fig. 3. General array for classifying triangles by weight index.

$$y_i = f \left(\sum_j w_{ij} n_{kj} \right) \quad (5)$$

where i is the output index, j is the weight index, k is the pattern index, w is the weight, and n is the number of triangles that correspond to the particular triangle class (i.e., the particular weight). Computation of this output takes into account only the information that is presented in the weights array and in the triangle array (Fig. 3). A graphical representation of the output computation is shown in Fig. 4.

- All weights are updated according to the following equations.

If $N_{kl} > 0$, then

$$\Delta w_{il} = \eta(t_i - y_i) \quad (6a)$$

and if $N_{kl} = 0$, then

$$\Delta w_{il} = 0 \quad (6b)$$

where t is the expected training output, y is the actual output, and η is the learning rate.

- Stages 2 and 3 are repeated until the training phase is complete.

After the training phase is complete, only the array of learned weights and the corresponding

coefficients k , l , and m that represent the equivalence class (from the upper three rows of the array in Fig. 3) must be saved. Each of the numbers in the weight array requires four bytes in memory.

3.4. Recognition

The algorithm for recognition can be described as follows:

- The pattern intended for recognition is loaded.
- An array is constructed of coordinates of contour pixels. This array should have the form presented in Fig. 2, but with only one pattern.
- A one-dimensional zero array N is constructed with a length equal to the number of learned weights (NoW). This array is a counter for the triangles in the pattern that correspond to the particular triangle class (i.e., the particular weight).
- To classify the triangles, the coefficients k , l , and m are computed for all possible triangles as described in Sections 3.1 and 3.2. After each computation, the newly found k , l , and m are compared with the ones previously saved in the training stage (in the upper part of the array in Fig. 3). If a matching class for the triangle is

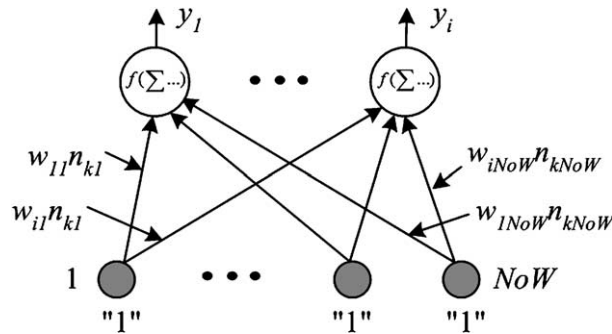


Fig. 4. Graphical view of the modified network.

found, the counter N corresponding to that triangle class position is increased by one. Thus, a nonzero array of counters N is built during classification:

- Using the weights array W from the training stage and the array of counters N , the outputs y_i are computed with the following equation:

$$y_i = f\left(\sum_j w_{ij}n_j\right) \tag{7}$$

3.5. The advantages of the modified network

The proposed network modification significantly reduces computational time in the construction, training, and recognition phases. Memory requirements are minimized, making the hardware requirements more feasible. In addition, recognition results are better than those obtained with conventionally constructed networks.

4. Experimental results

This section presents experimental results for the modified HONN resources and compares them with the conventional HONN construction method. The simulations were performed on a Pentium 4 computer (1700MHz), and the programs for both the modified HONN and conventional HONN were written in Visual C++. Seven different object classes with both 60×60 and

170×170 pixel patterns were used in the construction and recognition stages. These patterns (the contours of actual keys) are shown in Fig. 5. One object from each class was used in the training phase and 14 rotated patterns of each object were used in the recognition phase. Examples of the patterns used in the recognition stage are shown in Fig. 6.

Table 2 presents a comparison of the demands made for computational resources by 60×60 and 170×170 input fields. The reduction in the number of computational steps achieved with the modified network is four orders of magnitude for an input field of 60×60 and five orders of magnitude for an input field of 170×170 . This difference will increase further with increasing image size. In addition, the memory requirements are significantly reduced: only 81 340 bytes are required for the angular tolerance in this example, since only the weight array must be saved.

Tables 3 and 4 summarize a set of experiments conducted on both 60×60 and 170×170 input patterns to compare the recognition rates for different numbers of classes. Pattern rotation introduces a distortion and a new set of triangles is produced that differs from the triangle set of the original pattern. Such distortions lead to reductions in the recognition rates.

The tabulated data demonstrate that the various angular and area tolerances give quite similar results. The configuration with the smallest number of weights is thus generally preferred, because smaller weight numbers give faster algorithms.



Fig. 5. Patterns used for network training.

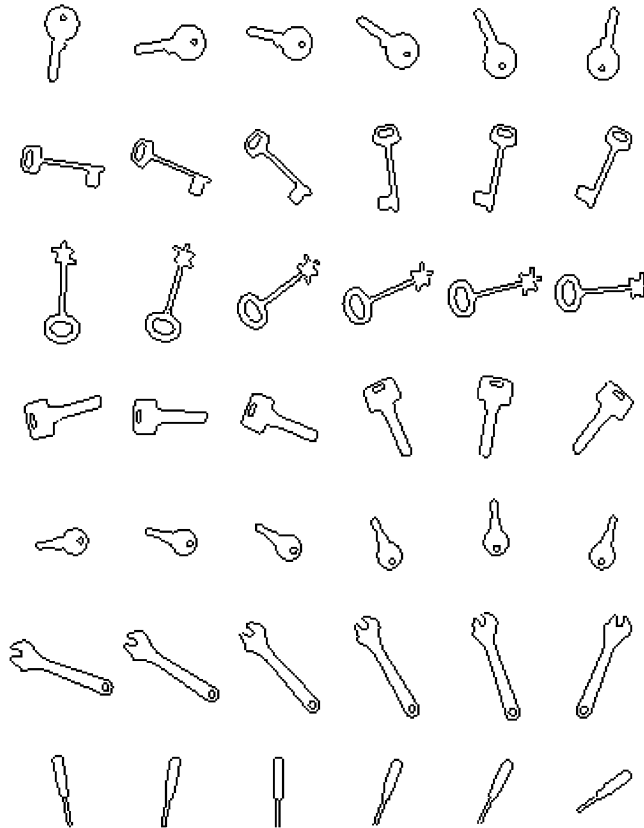


Fig. 6. Examples of patterns used in the recognition phase.

For comparison, a few results of the “approximately similar triangles” scheme are provided in Table 5. Only the results for the best configuration are shown, but even these have much lower recognition rates than the modified HONN. The rates in Table 5 would be even lower for larger input fields. In this scheme, similar triangles with very large differences in size are associated with the same triangle class; therefore, several different object classes may be associated with a single triangle class. Since each of these object classes will not be associated its own unique triangle set, recognition rates will be lowered.

A set of noisy patterns was synthesized to check the response of the network to noise. Examples of noisy patterns are shown in Fig. 7; in this case, about 0.5% of all pixels had inversed values. Table 6 shows the results for experiments that were conducted for a single network configuration with added noise.

As can be seen in Table 6, this network is very sensitive to the noise. It will not work with noisy patterns without prior noise removal.

The experimental data demonstrates that the modified HONN method enables the computation of large input fields without imposing impractical

Table 2

Comparison of the demands of HONN for computational resources under the following conditions: the “approximately similar triangles” scheme was used alone, the network was trained for the first five pattern classes, w was set at $\pi/180$, and m was not used

| Input field size | 60 × 60 | | 170 × 170 | |
|--|---|---|--|---|
| | Conventional net | Modified net | Conventional net | Modified net |
| Computational steps (number) | 7.8×10^9 | 13.8×10^5 | 4.02×10^{12} | 4.5×10^7 |
| Weight number | 5810 | 5810 | 5810 | 5810 |
| Memory required to store weight pointers (bytes) | $2 \times 7.8 \times 10^9 = 15.5 \times 10^9$ | 0 | $2 \times 4.02 \times 10^{12} = 8.04 \times 10^{12}$ | 0 |
| Memory required to store weights (bytes) for a network with five outputs | $5 \times 2 \times 5810 = 58100$ | $5 \times 2 \times 5810$ [weights] + $2 \times 2 \times 5810$ [k, l] = 81340 | $5 \times 2 \times 5810 = 46480$ | $5 \times 2 \times 5810$ [weights] + $2 \times 2 \times 5810$ [k, l] = 81340 |
| Total memory requirements (bytes) | 15.5×10^9 | 81340 | 8.04×10^{12} | 81340 |

Table 3

Recognition rate for a 60 × 60 pattern with various angular tolerances (w), area tolerances (s), and numbers of trained classes

| Angular tolerance | Area tolerance | Number of trained classes | | | | | | Weight number |
|-------------------|----------------|---------------------------|----|----|----|----|----|---------------|
| | | 2 | 3 | 4 | 5 | 6 | 7 | |
| $\pi/60$ | 20 | 100 | 95 | 95 | 85 | 80 | 80 | 8935 |
| | 50 | 100 | 93 | 90 | 88 | 78 | 80 | 3775 |
| | 100 | 100 | 95 | 80 | 71 | 72 | 72 | 2047 |
| $\pi/30$ | 20 | 100 | 95 | 95 | 87 | 82 | 82 | 2576 |
| | 50 | 100 | 93 | 88 | 75 | 75 | 72 | 1087 |
| | 100 | 100 | 95 | 85 | 72 | 74 | 74 | 589 |
| $\pi/20$ | 20 | 100 | 95 | 88 | 80 | 80 | 79 | 1217 |
| | 50 | 100 | 95 | 83 | 78 | 70 | 63 | 535 |
| | 100 | 100 | 91 | 83 | 73 | 58 | 60 | 291 |

Table 4

Recognition rate for a 170 × 170 pattern with various angular tolerances (w), area tolerances (s), and numbers of trained classes

| Angular tolerance | Area tolerance | Number of trained classes | | | | | | Weight number |
|-------------------|----------------|---------------------------|-----|----|----|----|----|---------------|
| | | 2 | 3 | 4 | 5 | 6 | 7 | |
| $\pi/60$ | 20 | 100 | 100 | 95 | 95 | 93 | 92 | 82368 |
| | 50 | 100 | 100 | 93 | 90 | 90 | 89 | 30953 |
| | 500 | 100 | 95 | 92 | 90 | 88 | 87 | 3428 |
| $\pi/30$ | 20 | 100 | 100 | 95 | 95 | 95 | 87 | 22023 |
| | 50 | 100 | 96 | 83 | 86 | 84 | 86 | 8897 |
| | 500 | 100 | 93 | 88 | 85 | 80 | 78 | 982 |
| $\pi/20$ | 20 | 100 | 98 | 90 | 88 | 87 | 88 | 11003 |
| | 50 | 100 | 98 | 88 | 86 | 86 | 84 | 4445 |
| | 500 | 100 | 95 | 85 | 85 | 73 | 66 | 498 |

Table 5

Recognition rates for a 60 × 60 pixel pattern using the “approximately similar triangles” scheme alone

| Angular tolerance (w) | Number of trained classes | | | | Weight number |
|---------------------------|---------------------------|----|----|----|---------------|
| | 2 | 3 | 4 | 5 | |
| $\pi/225$ | 100 | 80 | 75 | 60 | 8533 |



Fig. 7. Examples of noisy patterns.

Table 6

Recognition rates for 60 × 60 pixel patterns with added noise

| Angular tolerance (w) | Area tolerance (s) | Number of trained classes | | |
|---------------------------|------------------------|---------------------------|----|----|
| | | 2 | 3 | 4 |
| $\pi/20$ | 20 | 90 | 62 | 21 |

demands on computing resources. Translation invariance is built into the network, and thus 100% translation invariance is achieved. The “approximately equal triangles” scheme influences the scale invariance capabilities of the network negatively, however, since similar triangles that are different in size will not share the same weight.

Much better recognition results can be achieved for data sets in which the object classes differ significantly in form as well as in size.

5. Conclusions

A modified high-order neural network for efficient invariant object recognition has been presented. The proposed modification achieves significant reductions in computation times and memory requirements. With the proposed modified HONN, large input patterns (greater than 100×100 pixels) are processed without extremely large computational resources. Computation times are improved by using prior knowledge of patterns in the learning stage to eliminate idle loops. By using the “approximately equal triangles” scheme, the performance of the network is also improved significantly. Further enhancements may be possible by combining our modified method with previously proposed methods such as coarse coding (Spirkovska and Reid, 1993) or sub-sampling (Spirkovska and Reid, 1990). Additional research is planned on improving the performance of the modified HONNs with rotated, scaled, and distorted patterns.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful notes.

References

- Barnard, E., Casasent, D., 1991. Invariance and neural nets. *IEEE Trans. Neural Networks* 2 (5), 498–507.
- Chaudhuri, B.B., Bhattacharya, U., 2000. Efficient training and improved performance of multilayer perceptron in pattern classification. *Neurocomputing* 34 (1–4), 11–27.
- Fukushima, K., 2001. Recognition of partly occluded patterns: A neural network model. *Biol. Cybernet.* 84, 251–259.
- Guler, M., 2001. A model with an intrinsic property of learning higher order correlations. *Neural Networks* 14, 495–504.
- He, Z., 1999. Invariant Pattern Recognition with Higher-Order Neural Networks. M.Sc. thesis, School of Electrical and Computer Engineering, Nanyang Technological University, Singapore.
- He, Z., Siyal, M.Y., 1999. Improvement on higher-order neural networks for invariant object recognition. *Neural Process. Lett.* 10, 49–55.
- Mahmoud, I.K., Mohamed, M.B., 2000. Invariant 2D object recognition using the wavelet modulus maxima. *Pattern Recogn. Lett.* 21, 863–872.
- Perantonis, S.J., Lisboa, P.J.G., 1992. Translation, rotation, and scale invariant pattern recognition by higher-order neural networks and moment classifiers. *IEEE Trans. Neural Networks* 3 (2), 241–251.
- Spirkovska, L., Reid, M.B., 1990. Connectivity strategies for higher-order neural networks applied to pattern recognition. In: *Proceedings of IJCNN*. Vol. 1, San Diego, pp. 21–26.
- Spirkovska, L., Reid, M.B., 1992. Robust position, scale, and rotation invariant object recognition using higher-order neural networks. *Pattern Recogn.* 25 (9), 975–985.
- Spirkovska, L., Reid, M.B., 1993. Coarse-coded higher-order neural networks for PSRI object recognition. *IEEE Trans. Neural Networks* 4 (2), 276–283.
- Takano, M., Kanaoka, T., Skrzypek, J., Tomita, S., 1994. A note on a higher-order neural network for distortion invariant pattern recognition. *Pattern Recogn. Lett.* 15 (6), 631–635.
- Wong, W.H., Siu, W.C., Lam, K.M., 1995. Generation of moment invariants and their uses for character recognition. *Pattern Recogn. Lett.* 16, 115–123.
- Wood, J., 1996. Invariant pattern recognition: A review. *Pattern Recogn.* 29 (1), 1–17.