

# PRACTICAL, COMPUTATION EFFICIENT HIGH-ORDER NEURAL NETWORK FOR ROTATION AND SHIFT INVARIANT PATTERN RECOGNITION

Evgeny Artyomov and Orly Yadid-Pecht

**Abstract:** In this paper, a modification for the high-order neural network (HONN) is presented. Third order networks are considered for achieving translation, rotation and scale invariant pattern recognition. They require however much storage and computation power for the task. The proposed modified HONN takes into account a priori knowledge of the binary patterns that have to be learned, achieving significant gain in computation time and memory requirements. This modification enables the efficient computation of HONNs for image fields of greater than  $100 \times 100$  pixels without any loss of pattern information.

**Keywords:** HONN, higher-order networks, invariant pattern recognition.

## 1. Introduction

Invariant pattern recognition using neural networks was found to be attractive due to its similarity to biological systems. There are three different classes that use neural networks for invariant pattern recognition [1], that differ in the way invariance is achieved, i.e. Invariance by Training [2], Invariant Feature Spaces, or invariance by Structure, good examples are: the Neocognitron and HONN [3].

In third-order networks, which are a special case of the HONN, invariance is built into the network structure, which enables fast network learning with only one view of each pattern presented at the learning stage. However, an exponentially growing amount of interconnections in the network does not enable its usage for image fields larger than  $18 \times 18$  pixels [3]. A few different solutions were proposed to minimize the number of the HONN interconnections. Weight sharing, by similar triangles [3]. Weight sharing by "approximately similar triangles" [4]-[5]. Coarse coding [6]. Non-fully interconnected HONN [7]. All these methods partially solve the problem of the HONN interconnections but do not help with larger images. 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 a priori knowledge of the binary patterns that must be learned. By eliminating idle loops, the network achieves significant reductions 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  $100 \times 100$  pixels without any loss of image information — an impossible task with any previously proposed algorithm.

## 2. HONN architecture

Following equation describes the output of a third-order network:

$$y_i = f\left(\sum_j \sum_k \sum_l w_{ijkl} x_j x_k x_l\right), \quad (1)$$

where  $w$  is the weight associated with a particular triangle,  $y$  is the output and  $x$  is a binary input,  $j$ ,  $k$ , and  $l$  are the indices of the inputs.

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

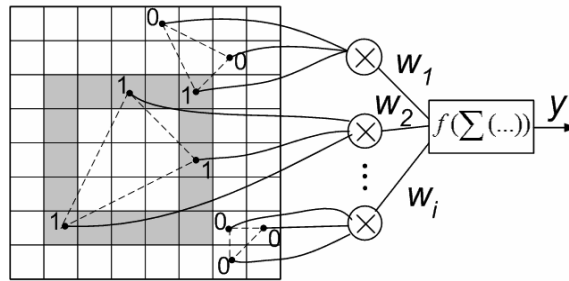


Fig.1. Schematic description of a third-order network

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

$$\Delta w_{ijkl} = \eta(t_i - y_i)x_j x_k x_l, \quad (2)$$

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

The number of triangles ( $NoT$ ) can be calculated with the following equation:

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

where  $IN$  is the number of input nodes.

For image fields  $100 \times 100$  and  $256 \times 256$  the number of triangles will be  $1.6662 \times 10^{11}$  and  $4.6910 \times 10^{13}$  accordingly. As can be seen, the number of triangles grows very fast off 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.

The problem of large computational demands arises since the network is constructed in the pre-processing stage before the learning phase. At this stage, all possible triangles are computed and pointers to the weights are saved [8]. In addition to the pointers, the weight array is also stored. At least two memory bytes are required for each pointer. If, for example, an input field of  $100 \times 100$  pixels is given, the total number of bytes required to store the entire vector of pointers is  $3.3324 \times 10^{11}$  bytes. The memory and computation requirements are enormous. To work with large input patterns, significant network modifications are required.

### 3. The proposed modified HONN method

As noted before, the input pattern is binary: edge or contour pixel has the value "1" and all other pixels have the value "0". As can be seen from equation (1), each product with pixel value "0" will give "0" as a result. This means that only active triangles (in which all pixels belong to an object contour) will influence the result. In addition, the weights that belong to the inactive triangles will not be updated and will keep "zero" value during the learning process.

Following this observation, the network can be modified and all inactive triangles can be disregarded during the construction phase, which eliminates the idle loops from the computation. With this modification, the network configuration strictly depends on the input patterns that have to be learned.

In addition, to improve network performance regarding rotation, distortion and a number of learned classes we introduce an "approximately equal triangles" scheme for network construction. This scheme, in addition to the "approximately similar triangles" scheme presented in [4] for weights sharing, adds triangle area equality. This means that "approximately similar triangles" with "approximately equal" areas will share the same weight.

#### 3.1 The proposed network construction

The modified algorithm for network construction can be described as follows: 1. Load all patterns that must be learned. 2. Run through each image and save the coordinates of the contour (boundary) pattern pixels to the

different arrays. A set of such arrays is shown in Fig. 2. 3. Compute angles of all presented triangles and classify them in order to associate with a particular weight.

Indices  $X_{im}$ ,  $Y_{im}$  and  $n_{ij}$  correspond to pattern number ( $n$ ), pixel number ( $m$ ), weight index ( $j$ ) and pattern number ( $i$ ). The variable  $n_{ij}$  is the number of triangles from the particular pattern that correspond to the particular triangle weight index (class).

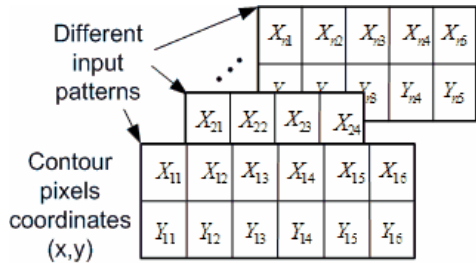


Fig.2. Arrays of the pixel coordinates of the object contours

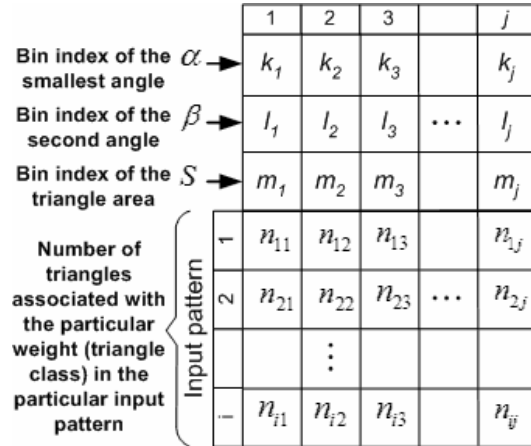


Fig.3. General array for classifying triangles by weight index.

The presented method of classification is based on “approximately equal triangles”. For the association of a triangle with a particular weight, the sets of possible values of the two smallest triangle angles ( $\alpha$ ,  $\beta$ ) and the triangle area ( $S$ ) are partitioned into bins defined by:

$$(k - 1) w \leq \alpha < kw, (l - 1) w \leq \beta < lw, (m - 1) s \leq S < ms, \tag{4}$$

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

During the classification step, 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. After construction, only the array of triangle classes presented in Fig.3 must be stored in memory.

### 3.2 Network training

The previously constructed array of triangle classes (Fig. 3) is used as the basis for learning in the training phase. In addition, a zero matrix of weights ( $\mathbf{W}$ ) with the size of  $NoP \times NoW$  is constructed. Where  $NoW$  is the number of individual weights,  $NoP$  is the number of training patterns.

Output computation takes into account only information presented in the weights array ( $\mathbf{W}$ ) and in the triangle array ( $\mathbf{N}$ ) (Fig. 3). It follows the next equation for a particular input image:

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

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).

All weights are updated only once after each iteration, according to the next equation:

$$\Delta w_{il} = \eta(t_i - y_i) \text{ , if } N_{kl} > 0 \text{ ; } \Delta w_{il} = 0 \text{ , if } N_{kl} = 0, \tag{6}$$

where  $t$  is the expected training output,  $y$  is the actual output,  $\eta$  is the learning rate.

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.

### 3.3 Recognition

The algorithm for recognition can be described as follows: 1. Load pattern intended for recognition. 2. Construct an array of coordinates of contour pixels (as in the construction stage). 3. Construct a zero matrix ( $\mathbf{N}$ ) with the size equal to  $1 \times NoW$ . This is a counter for triangles in the image, which correspond to the particular weight. 4. Run through the coordinate array and compute coefficients  $k$ ,  $l$  and  $m$  for all possible triangles as was described in 3.1. After each computation, compare the newly found  $k$ ,  $l$  and  $m$  with the ones previously saved (upper part of the array from Fig.3). If a matching class for the triangle is found, the counter corresponding to that triangle class position is increased by one ( $n_{ij} = n_{ij} + 1$ ). Thus, during classification the nonzero one-dimensional matrix of counters ( $\mathbf{N}$ ) is built. 4. Compute outputs according to equation (5), using the weights array ( $\mathbf{W}$ ) built during construction phase and the triangle counters ( $\mathbf{N}$ ) built in the beginning of recognition phase.

## 4. Experimental results

To study the performance of the modified network and compare computational resources with the conventional network, seven different object classes with  $60 \times 60$  and  $170 \times 170$  pixels were prepared. One object from each class was used in the training phase and 14 rotated patterns of each class were used in the recognition phase. Pattern examples are shown in Fig.4.



Fig.4. Pattern examples

The comparison for computational resource demands for  $60 \times 60$  and  $170 \times 170$  input fields are presented in Table 1.

As can be seen from the table, the gain achieved with the modified network in computational steps amount is four orders of magnitude for an input field  $60 \times 60$  and five orders of magnitude for an input field of  $170 \times 170$ . This gain will be more significant with image size increase. In addition, the memory resources are minimized also.

Table 1: Comparison of the computational resources demands  
("approximately similar triangles" scheme is used alone, the network was trained for first five pattern classes,  $w = \pi/180$ ,  $m$  - not used).

Input field size	60 x 60		170 x 170	
	Conventional	Modified	Conventional	Modified
Computational steps (number)	$7.8 \times 10^9$	$13.8 \times 10^5$	$4.02 \times 10^{12}$	$4.5 \times 10^7$
Total memory requirements (bytes)	$15.5 \times 10^9$	81340	$8.04 \times 10^{12}$	81340

Table 2: Recognition rate for a varying number of trained classes, angular and area similarities.  
Input pattern:  $60 \times 60$  pixels.

Tolerance		Number of trained classes						Weight number
Angular ( $w$ )	Area ( $S$ )	2	3	4	5	6	7	
$\pi/60$	10	100	95	94	84	80	80	17286
$\pi/60$	20	100	95	95	85	80	80	8935
$\pi/20$	10	100	95	91	87	82	80	2502
$\pi/20$	20	100	95	88	80	-	-	1217

Table 3: Recognition rates of the net with the "approximately similar triangles" scheme alone.  
Input pattern: 60 x 60 pixels.

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

For comparison with the "approximately similar triangles" scheme, a few results are provided in Table 3. Results for the best configuration are shown only, but even this shows much worse recognition rates. The cause for this is that similar triangles with very large difference in size are associated with the same triangle class, as a result, some object classes will be associated with the same triangle class, preventing from the objects to have an individual triangle set associated with it.

From the experimental data provided, it can be seen that our method enables the possibility of large input field computation without significant resource demands. Translation invariance is built into the network, thus 100% translation invariance is achieved. All experimental data are provided for this particular data set. For other data sets, where object classes differ significantly in size and in form, much better recognition results can be achieved.

## 5. Conclusions

A modified High-Order Neural Network for efficient invariant object recognition has been presented. The proposed modification achieves significant gain in computation time and memory requirements. The gain in computation time is achieved by eliminating the idle loops, by taking a priori knowledge of training patterns. With the proposed modified HONN, large input patterns can be processed without large computation demands. Performance of the network is improved also significantly, by using the "approximately equal triangles" scheme.

## References

- [1] Barnard E., D. Casasent. 1991. Invariance and neural nets. IEEE Transactions on Neural Networks, vol. 2, no. 5, pp. 498-507.
- [2] Wood J. 1996. Invariant pattern recognition: a review. Pattern Recognition, vol. 29, no.1, pp.1-17.
- [3] Spirkovska L., M.B. Reid. 1992. Robust position, scale, and rotation invariant object recognition using higher-order neural networks. Pattern Recognition, Vol.25, No. 9, pp. 975-985.
- [4] Perantonis S.J., P.J.G. Lisboa. 1992. Translation, Rotation, and Scale Invariant Pattern Recognition by Higher-Order Neural Networks and Moment Classifiers. IEEE Transactions on Neural Networks, Vol.3, No. 2, pp. 241-251.
- [5] He Z., M.Y. Siyal. 1999. Improvement on Higher-Order Neural Networks for Invariant Object Recognition. Neural Processing Letters, Vol. 10, pp 49-55.
- [6] Spirkovska L., M.B. Reid. 1993. Coarse-Coded Higher-Order Neural Networks for PSRI Object Recognition. IEEE Transactions on Neural Networks, Vol. 4, No. 2, pp. 276-283.
- [7] Spirkovska L., M.B. Reid. 1990. Connectivity Strategies for Higher-order Neural Networks applied to Pattern Recognition. Proceedings of IJCNN, Vol. 1, San Diego, pp. 21 - 26.
- [8] He Z. 1999. Invariant Pattern Recognition with Higher-Order Neural Networks. Master Thesis, School of Electrical and Computer Engineering, Nanyang Technological University, Singapore.

## Authors Information

**Evgeny Artyomov**<sup>1</sup> - e-mail: [artemov@bgumail.bgu.ac.il](mailto:artemov@bgumail.bgu.ac.il)

**Orly Yadid-Pecht**<sup>1,2</sup> - e-mail: [oy@ee.bgu.ac.il](mailto:oy@ee.bgu.ac.il)

1. The VLSI Systems Center, Ben-Gurion University, Beer Sheva, Israel.

2. Dept of Electrical and Computer Engineering, University of Calgary, Alberta, Canada