Industry
Canada
CRC

A NOVEL APPROACH TO THE APPLICATION OF HIGHER ORDER NEURAL NETWORKS TO IMAGE CLASSIFICATION

by

Fivos Hatsivasiliou and Kenneth L. Sala

CRC REPORT NO. 96-005

May 1996
Ottawa

The work described in this document was sponsored by the Department of National Defence under Task 1410-233.

Canadä

TK
5102.5
C673e
#96-005
c.b
S-Gen

A Novel Approach to the Application of Higher Order Neural Networks to Image Classification

by

Fivos Hatsivasiliou and Kenneth L. Sala
Communications Systems Research
Antennas and Integrated Electronics

The work described in this document was sponsored by the Department of National Defence under Task 1410-233.

COMMUNICATION RESEARCH CENTRE, INDUSTRY CANADA
CRC REPORT NO. 96-005

Canadä

May 1996
Ottawa

Abstract

The applicability of higher order neural networks to the classification of low resolution imagery is investigated. A novel boundary detection and encoding methodology is developed in order to significantly reduce the large number of third order interconnection weights which must be found during the training stage of the neural network. The higher order neural classifier can be trained by presenting only one sample image per class and so enables rapid network learning with a minimal requirement for training data. An extensive, MatLab compatible toolbox developed specifically to implement this approach is described and documented along with the algorithms employed in the image boundary detection and encoding process.

Resumé

L'applicabilité des réseaux neurals d'ordre élévé à la classification des images à faible résolution est étudié. Une nouvelle méthodologie de détection des conditions limites et d'encodage est proposée pour réduire substanciellement le grand nombre de poids d'interconnection de troisième ordre que l'on détermine pendant le stage d'apprentissage d'un réseau nouveau. Le classificateur neural d'ordre élevé peut apprendre en le présentant un seul échantillon d'image et permet ainsi un apprentissage accéléré du réseau avec un minimum de données. Un large utilitaire, compatible à MatLab, a été développé pour l'éxecution de cette approche et est décrite et documentée avec les algorithmes employés pour la détection des conditions limites et les procédures d'encodage.

Executive Summary

This report summarizes research conducted at the Communications Research Center from 1993 to 1995 concerning the applicability of higher order neural networks to the classification of low resolution imagery. In order to significantly reduce the large number of third order interconnection weights which must be found during the training stage of the neural network, a novel boundary detection and encoding methodology is developed. The higher order neural classifier can be trained by presenting only one sample image per class and so enables rapid network learning with a minimal requirement for training data. An extensive, MatLab compatible toolbox developed specifically to implement this approach is described and documented along with the algorithms employed in the image boundary detection and encoding process.

Table of Contents

1. Introduction

In the preponderance of environments, both military and civilian, in which some form of imaging technology is used to provide intelligence through automated target and scene classification, the observer has little if any control over the degree of image variability or degradation arising from various types of geometric transformations (scaling, object position, viewing angle) or from the presence of noise and occultation in the imaging process. The ability, therefore, to recognize and classify an object independently of such variations in the image presents one of the most critical and challenging aspects in the design of an automated target recognition system [2,7,20,21,22,32,35,36].

The application of various types of neural networks to image classification has become one of the principal thrusts by which researchers are attempting to develop automated target recognition systems, particularly within military contexts involving, for example, synthetic aperture radar (SAR) imagery, visible and infrared photography, and sonar signal classification (several excellent examples [5,6,7,13,21,32,33,35,37,49] of this type of research may be found in the 1995 Special Issue on Automatic Target Recognition, Neural Networks, vol. 8, no. 7/8; see also Roth [36], Bachmann et al [1], Dudani et al [10], Smith and Wright [42]). There are several advantages to the use of neural networks for pattern recognition and image classification with the principal ones being: (a), artificial neural networks possess the ability to "learn" classification criteria not necessarily known a priori by adaptively training the network on known examples; (b), neural networks are parallel processing systems and so offer the potential for real-time, in-line automated target classification in the case, e.g., of high resolution, land SAR imagery; (c), neural networks are distributed processing systems and thus exhibit, in general, excellent fault tolerance (see the reviews by Rogers et al [35] and Roth [36]).

The most common approach in the applications of neural networks to image classification consists of two stages; (1) a feature vector representing a specific image is extracted and, (2), using the feature vector as its input, a trained neural network classifies the image (i.e., the feature vector) as belonging to some particular class of objects. Although it is possible for neural networks to learn various types of invariance strictly through the training process, it has been found by many researchers that such an approach is impractical in terms of training times and amount of training data required and, more importantly, is ineffective in arriving at trained networks which are accurate and robust in their tolerance to image variations [2,4,22,35,36,38]. The almost universally preferred approach has been to derive image features which are explicitly invariant to various types of image variability such as the scale, position, or in-plane angle of the object in the image plane [2,3,4,7,14,15,17-20,26,27,39,46,47,50]. One such approach, in particular, is based upon the calculation of 'moment invariants' [14,15] using different types of basis functions for the feature extraction and work on this technique will be presented in a separate report.

An alternative and fundamentally different approach to incorporating invariance into neural network image classification involves the use of higher order neural networks (HONN's) [2,17,28,30,34,40,41,43-45]. Succinctly, such networks employ an input distribution layer which combines more than one image pixel (two in a second order network, three in a third order network, etc.) in such a way that, through the imposition of symmetries in the network weights, the subsequent processing by additional layers of artificial neurons remains invariant to certain types of geometric variations in the input image. In this manner, a HONN builds the desired invariances directly into the network architecture and so avoids entirely the need for any form of feature extraction, an operation which is frequently computationally very intensive.

This report describes research performed at the Communications Research Center (CRC) between October, 1993, and December, 1995, concerning the applicability of higher order neural networks (HONN's) to the identification and classification of synthetic aperture radar (SAR) imagery. A novel approach involving image boundary detection and encoding in combination with a third order neural network was developed and tested. The implementation of this technique led to the development of an extensive HONN, MatLab-compatible toolbox which, along with several image array processing algorithms, are described and documented in this report.

The second part of this report gives a general overview of the theory and limitations of HONN's and their applicability to image classification. The third part, which is the most extensive, describes in some detail the research performed, the various algorithms developed, and the implementation of the MatLab functions. Also described is the validation of the proposed network topology and the boundary encoding methodology using a classical recognition problem. The final part of the report summarizes the conclusions and offers a brief discussion of the problems and the drawbacks encountered with the present approach along with some suggestions concerning future directions for this research. An appendix lists the annotated MatLab functions written to implement the present work and includes additional software documentation for the pre- and post-processing algorithms.

2. Image Classification using Higher Order Neural Networks

A schematic representation of a conventional, three-layer, feedforward neural network (a multilayer perceptron) is shown in figure 2.1. The initial layer to the network serves only as a distribution layer which, in the example shown of a fully connected (first order) network, simply distributes each input value x_i to each and every neuron in the first (processing) layer of the network. By convention, the weights w_{ij} are considered 'attached' to this first processing (input) layer, the weights v_{ij} to the hidden layer, and the weights u_{ij} to the network's output layer. Although this type of neural network can be trained using any of several learning algorithms, the predominant technique in use at present is the backpropagation algorithm along with several variations and refinements to the original algorithm [38].

A higher-order neural network (HONN) [8,30,34,43-45] differs from a conventional, first order network chiefly in the role played by the input distribution layer. Specifically, the distribution layer in a HONN combines input values into pairs, triplets, etc. of values,

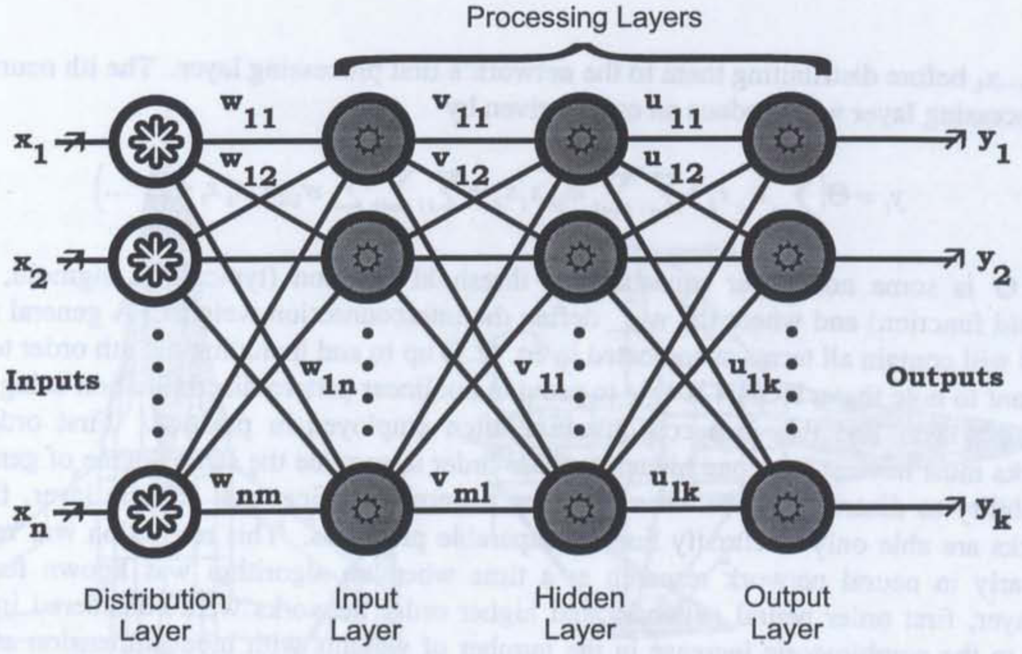

Figure 2.1 : A three layer, perceptron neural network with a first order distribution layer.

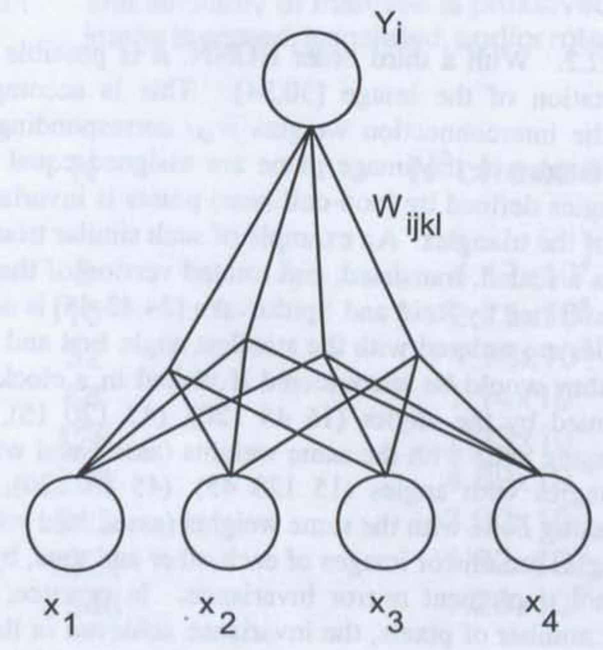

Figure 2.2 : A strictly third order neural network with one processing element and $N^2 = 4$.

$x_i x_j x_k \ldots x_t$, before distributing them to the network's first processing layer. The ith neuron in the first processing layer will produce an output given by

$$y_i = \Theta\left(\sum_j w_{ij}x_j + \sum_j \sum_k w_{ijk}x_j x_k + \sum_j \sum_k \sum_l w_{ijkl}x_j x_k x_l + \ldots\ldots\right) \tag{2.1}$$

where Θ is some non-linear squashing or threshold function (typically a sigmoid, tanh, or threshold function) and where the $w_{ijk\ldots}$ define the interconnection weights. A general nth order HONN will contain all terms as indicated in eq. (2.1) up to and including the nth order term. It is important to note that a HONN is able to provide nonlinear pattern discrimination using only one processing layer and this is a configuration often employed in practice. First order neural networks must have at least one hidden layer in order to provide the same degree of generality in their ability to discriminate between arbitrary pattern classifications. Single layer, first order networks are able only to classify linearly separable problems. This restriction was recognized very early in neural network research at a time when no algorithm was known for training multilayer, first order neural networks and higher order networks were considered impractical owing to the combinatoric increase in the number of weights with input dimension as outlined below (Minsky and Papert [29]).

In applications involving HONN's, it is more common to consider only *strictly* nth order HONN's, i.e., a HONN which contains *only* the nth order term in equation (2.1). A strictly third order HONN, for example, combines only triplets of input values according to

$$y_i = \Theta\left(\sum_j \sum_k \sum_l w_{ijkl}x_j x_k x_l\right) \tag{2.3}$$

as illustrated in figure 2.2. With a third order HONN, it is possible to achieve invariance to scale, position, and rotation of the image [30,34]. This is accomplished by imposing the following condition: the interconnection weights w_{ijkl} corresponding to triplets of pixels ijk which define similar triangles in the image plane are assigned equal values. The property of similarity between triangles defined by (non-collinear) points is invariant to any change in scale, translation, or rotation of the triangles. An example of such similar triangles common to a pair of images, one of which is a scaled, translated, and rotated version of the other, is shown in figure 2.3. The convention described by Reid and Spirkovska [34,43-45] is adopted for the ordering of the angles, i.e., the angles are ordered with the smallest angle first and the next two angles listed in the order in which they would be encountered if visited in a clockwise direction. Thus the three triangles represented by the angles (15 45 120), (45 120 15), and (120 15 45) are all connected to the processing node with the same weights (associated with the (15 45 120) triplet) whereas the three triangles with angles (15 120 45), (45 15 120), and (120 45 15) are all connected to the processing node with the same weights (associated with the (15 120 45) triplet). These two sets of triangles are mirror images of each other and thus, by treating them as distinct, such a HONN does not implement mirror invariance. In practice, since the image plane is represented by a finite number of pixels, the invariance achieved in this manner is approximate; perfect invariance is approached only as the pixel size, for a fixed image, is decreased. A major advantage of a third order HONN is that, since the network's output is unchanged by scale, translation, and rotation of the image, it is necessary to use only a single view of each object

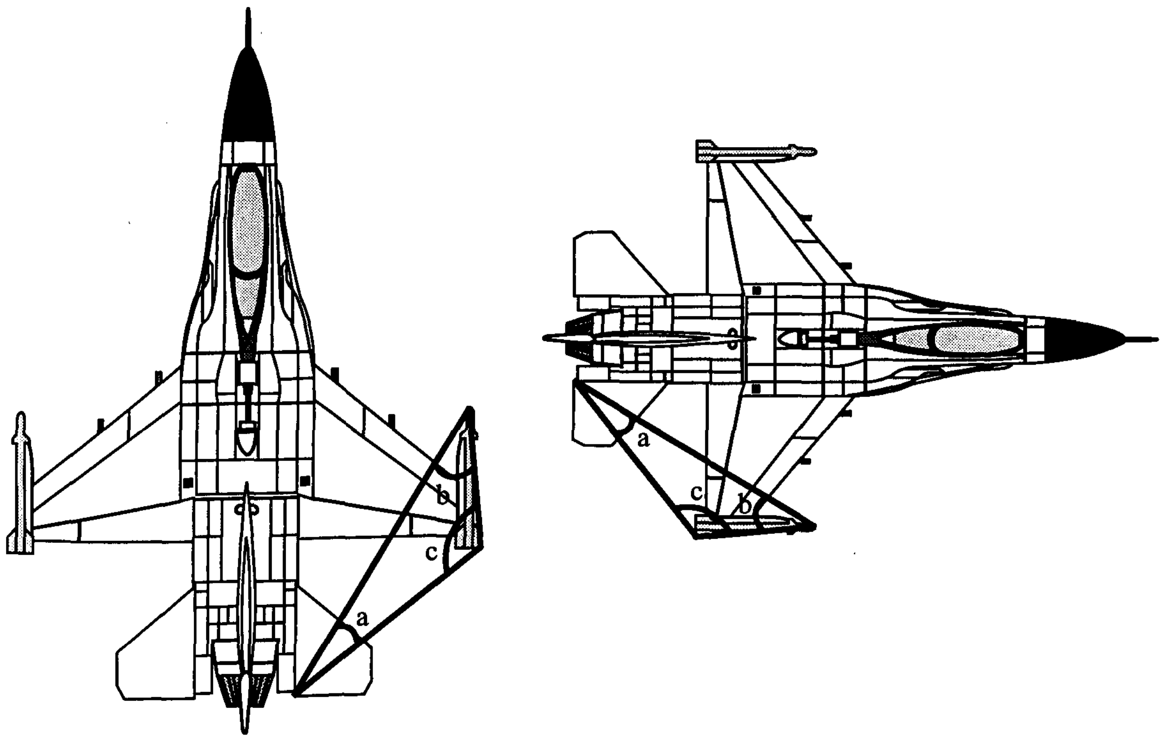

Figure 2.3 : The similarity of triangles is preserved when an image is scaled, translated, and/or rotated.

N	**N^2 choose 3**
8	4.17×10^{4}
16	2.75×10^{6}
32	1.78×10^{8}
64	1.14×10^{10}
128	7.33×10^{11}
256	4.69×10^{13}
512	3.00×10^{15}
1024	1.92×10^{17}
2048	1.23×10^{19}

Table 2.1 : The variation of "N^2 choose 3" with N.

defining a class in order to train the network. This minimizes the data requirement for training and also greatly shortens the training times of the network itself [34].

As attractive as the above capabilities of a HONN are, there exists one major disadvantage to HONN's, namely the greatly increased number of weights required in comparison to the conventional case of a first order neural network. For a third order HONN and an image plane of NxN pixels, the maximum number of possible combinations of three pixels is "N^2 choose 3" which is given by the binomial coefficient

$$\binom{N^2}{3} = \left(\frac{N^2!}{3!(N^2-3)!}\right) = \frac{N^2(N^2-1)(N^2-2)}{6} \tag{2.3}$$

Note that the leading term in this expression is N^6 (it would be N^4 for a second order network and N^2 for a first order network). Table 2.1 shows the value of "N^2 choose 3" for several different values of N. Clearly, storage requirements alone would prohibit the use of HONN's for values of N ~ 128 or greater. It should be noted, however, that these numbers represent a maximum amount, corresponding to the use of every pixel in the NxN image plane. In practice, only a subset of this plane is used (i.e., pixels with non-zero values). It is then possible to train the weights w_{ijkl} by first setting all of them to zero and then training only on those weights which correlate with three non-zero pixels in the image plane. Weights corresponding to pixel triplets having at least one zero value will remain zero and can be ignored a priori, thus somewhat reducing the number of weights which need to be determined.

Much of the research to date involving HONN's has concentrated on finding practical methods to reduce significantly the number of weights that must be calculated and trained. Some of the methods proposed include coarse gridding [45], grouping of similar triangles by limiting the resolution of the angle measurements [12,34,43], pruning of triplet sets [34,40,43], and the use of image boundaries [16,24,25,51]. In the present work, we pursue the latter approach using boundary representation of the images but go even further in that the number of boundary points is reduced by using algorithmic encoding for regular boundary lines. The next section provides more details of this approach.

3. Experimental Design and Results

The overall design and methodology adopted in the present approach to combining image boundary detection and encoding with a third order neural network classifier is outlined in the flow chart representation shown in figure 3.1. There are two distinct parts to the overall process of classifying a given test image. The left-hand part of the flowchart in figure 3.1 refers to a processing sequence which is a one-time only calculation, i.e., for a given resolution NxN, it is

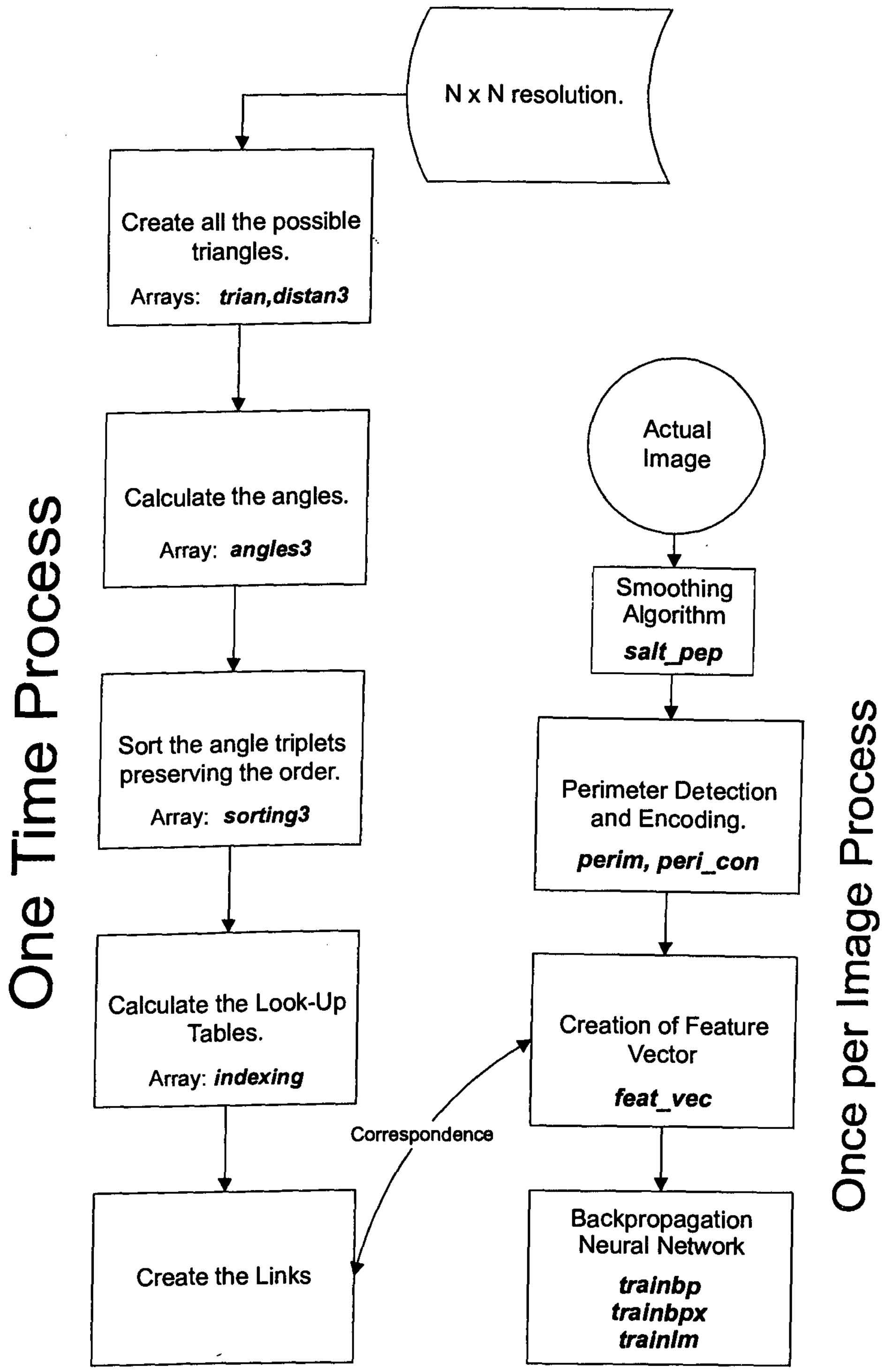

Figure 3.1 : Process flowchart for the sequence of array calculations.

necessary to calculate and sort the angles once and only once and then to store these results in a lookup table. Thereafter, the required calculations are as outlined in the right-hand part of the flowchart which refers to a specific test image. Several specific algorithms have been developed for each of the flowchart blocks since each block represents a complex operation and these are now described in sequence.

3.1 One Time Processing of Image Plane Arrays

The one time array calculations for a given resolution NxN of the frame under classification consist of determining a set of 5 arrays calculated in an ordered sequence. These arrays (for any given resolution in the array **segm**) have been given the following names: **trian**, **dist**, **ang**, **sort**, and **index**. All of the arrays corresponding to a given resolution NxN are grouped into one file under the name **higherNx.mat**. For example, the file **higher6x.mat** groups together the arrays calculated for a 6x6 resolution. The array **segm** refers to the resolution itself for which the HONN will be created. In this case, for demonstration purposes, a 6x6 array has been chosen. In the course of this work, HONN's have been constructed for resolutions up to 11x11. The only limitations are the pragmatic ones of memory requirements and processing time; in theory, all of the algorithms developed in this project can operate for an unlimited range of resolutions. An example (N=6) of the initial array **segm** is given in figure 3.2. The next step consists of finding all the possible triangles that can be formed using pixels from this array.

Creation of the possible triangles trian=*triangl3(*segm*).m*

The number of all possible triangles is given by the function ***combin(n,3)***, where n=NxN=36 for the example case. The number of rows of the **trian** array will be the number of different combinations of 36 taken 3 at a time with no repetitions as given by eq. (2.4), i.e., 7140. The function ***triangl3(f).m*** creates a lookup table with indices which correspond to the triangle coordinates formed from all possible combinations of 3 points. It is similar to ***triangl2.m*** but with the implementation of speed optimization. The format of the result is shown in figure 3.3. The convention used for the scanning pattern is to proceed clock-wise starting from the left-upper pixel of the input array, continuing with the first row, followed by the 2nd row, etc.

Calculation of the Distances dist=*distan3(*trian*).m*

The next step consists of the calculation of the lengths of the triangle sides. The function ***distan3(*trian*).m*** is used and the format of the output is a three-column array where the first column gives the length between the 1st and 2nd vertex of the triangle, the 2nd column gives the length between the 2nd and 3rd vertex of the triangle, and the third column gives the length between the 1st and the 3rd vertex. This routine is essentially the same as ***distan(f)*** with the only difference lying with the input format which now follows the above convention and is faster than ***distan2*** due to the speed optimization implemented. The output of this function (the array **dist**) for the 6x6 example is shown in figure 3.4.

```
0    0    0    0    0    0
0    0    0    0    0    0
0    0    0    0    0    0
0    0    0    0    0    0
0    0    0    0    0    0
0    0    0    0    0    1
```

Figure 3.2 : Example of initial array **segm** for N = 6.

```
1    1    1    2    1    3
1    1    1    2    1    4
1    1    1    2    1    5
1    1    1    2    1    6
..........................

3    1    3    2    6    6
3    1    3    3    3    4
3    1    3    3    3    5
```

i.e. the triangle that formed by the points with coordinates (1,1) , (1,2) and (1,6).

Figure 3.3 : Lookup table format for N = 6.

```
1.0000    1.0000    2.0000
1.0000    2.0000    3.0000
1.0000    3.0000    4.0000
1.0000    4.0000    5.0000
1.0000    1.4142    1.0000
............................
```

Figure 3.4 : Example of the output array **dist** for N = 6.

```
     0         0    3.1416
     0         0    3.1416
     0         0    3.1416
     0         0    3.1416
0.7854    1.5708    0.7854
0.7854    0.7854    1.5708
............................
```

Figure 3.5 : Example output array **ang** for N = 6.

Calculation of the included Angles ang=*angles3(dist).m*

In this step, the function ***angles3*(dist)** creates an array with the angles computed from the array **dist**, the three lengths that form the triangles. The result is an array with the same dimension as **dist**. The formula used is

$$\cos C = \frac{a^2 + b^2 - c^2}{2ab} \tag{3.1}$$

The format for the sequence of the angles is the same as described above for the function ***dist3***. This function incorporates a provision for the case where a side may be of zero length in order to prevent the generation of Matlab NaN or "not-a-number" results. This is the reason for the provision of the employment routine of ***zero_fil***. The output of the function ***angles3*(dist)** for the example N=6 is the array **ang** with the format shown in figure 3.5.

Sorting of the Angles sor=*sorting3*(ang).m

The term 'granularity' is introduced in this function and refers to the accuracy with which two angles will be considered different or equal. The underlying reason is that we deal with pixelized (i.e., sampled) images, consisting of discrete elements, and a subset of those pixels represents the vertex of the angles formed by the above functions. If the same image is scaled down, even without rotation, the scaled image will be represented by a smaller set of pixels and thus, owing to the discrete nature of image, the angles which are formed by the corresponding pixels may differ. Similar results can also follow from a slighted rotated (e.g., by 0.001 rad) representation of the initial image. Figures 3.6 and 3.7 illustrate the nature of the problems that may be encountered. Both sides of these figures represent an analysis of a 16x16 image. The left side shows a 2x14 linear bar aligned in parallel with the sensor (e.g., a camera) while the right side shows the kind of distortion which can be introduced by rotating the image.

For the purposes of this research, the granularity has been adjusted through experimentation to 0.001 rad. The higher the image resolution, the lower the granularity can be set. However, at the same time, the lower the granularity the more the calculated features will be different and the more discriminating the classification procedure can be, albeit at the cost of a noticeably increased processing load.

The function ***sorting3*(ang)** has as its input the **ang** array with all the possible angles that can be created at the given resolution as calculated in the above steps. In the first stage, ***sorting3*(ang)** operates on each row (row level only) and rearranges the angles that are given in every row in ***ang*** in such a way that the first is the smallest angle of the triplet but, at the same time, the order of the angles is preserved, i.e., (a,b,c)=(b,c,a)=(c,a,b). Numerically, a triplet consisting of the angles (60,30,90) will become (30,90,60) and **not** (30,60,90). The reason for the differentiation between the two is that they represent different features and, in the case where only one is encountered with the convention as described, the other will not be encountered as arising from any rotation, translation, or scaling transformation. The only way in which it can be encountered is through a mirroring. Mirroring invariance, though, is here considered as undesirable since it is not encountered under normal circumstances. In the second stage, the function ***sorting3*(ang)** rearranges the array created in the first stage in such a way that it groups

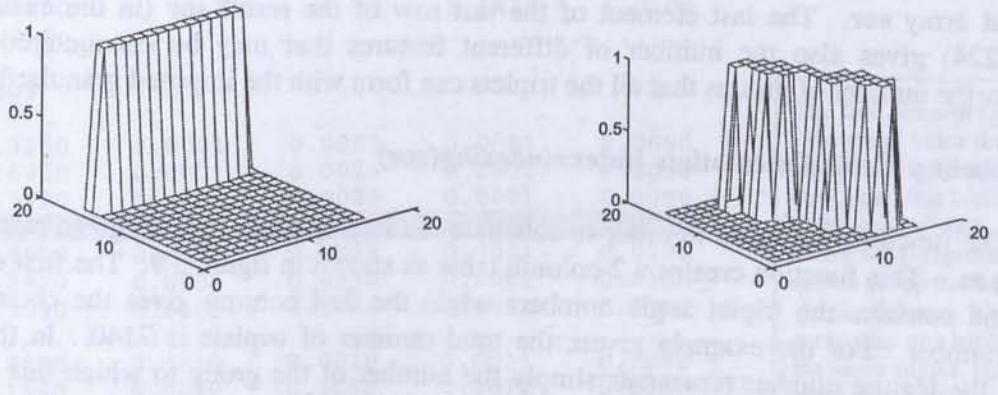

Figure 3.6 : Example of rotation induced distortion on a 16x16 image grid.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 3.7 : Binary valued arrays corresponding to figure 3.6.

all the triplets that have the same smallest angle into subgroups. In the next phase, it operates on each subgroup in order to rearrange the elements of the subgroup so that the triplets which have the same second angle are grouped together in ascending order. Figure 3.8 shows the format of the output array **sor**. The last element of the last row of the result **sor** (in the example, the number 224) gives also the number of different features that may be encountered since it represents the number of groups that all the triplets can form with the imposed granularity.

Lookup Table Calculation index=*indexing(*sor*)*

The function employed in order to calculate the lookup table, for the given resolution, is ***indexing.m.*** This function creates a 2-column table as shown in figure 3.9. The first column is sorted and contains the triplet angle numbers while the 2nd column gives the corresponding feature number. For the example given, the total number of triplets is 7140. In the second column, the feature number represents simply the number of the group to which this particular set belongs.

Batch Processing Operation *segm_bat(*segm*)*

All of the above matrices can be generated through a batch processing mode implemented with the function ***segm_bat(*segm*).*** Given in **segm** the segment array (i.e., a 10x10 array), ***segm_bat(*segm*)*** calculates each of the five necessary arrays in order to proceed with the HONN training.

3.2 Image Processing

The size of the feature vector for a given resolution is known after the creation of the lookup table as described in the one time array processing section. The size of this vector is also affected by the defined granularity, i.e. a larger feature vector will result when the granularity is set equal to 0 as compared to the case of a nonzero value of granularity since, in the latter case, comparable sets of triangles will be grouped under the same feature. For example, in the case of a 4x4 array, with 560 possible combinations of the pixels taken 3 at a time, only 38 triplets of ordered angles (groups) can be created and consequently the feature vector will be a 1x38 column array. For test purposes, a 6x6 array will be used for which the number of features have been found to be 224. The test images will be the letters **c** and **t** which constitute a standard reference test pattern as commonly employed in the relevant literature[38]. The binary patterns representing these images are given in figure 3.10.

Smoothing Algorithm

The first step in the processing of the target image is the employment of a "salt and pepper" algorithm. The underlying idea is to specify a Boolean function on a neighborhood centered at a pixel p and to assign to this pixel a value of either 1 or 0 depending on the spatial arrangement and binary values of its neighbors. The 3x3 neighborhood mask is shown in figure 3.11. This algorithm is called "salt and pepper" because (a), it eliminates isolated 1's, and, (b), it

```
      1.0000         0          0      3.1416     1.0000
      2.0000         0          0      3.1416     1.0000
      3.0000         0          0      3.1416     1.0000
      4.0000         0          0      3.1416     1.0000
     35.0000         0          0      3.1416     1.0000
...................................................

  1.0e+003 *

      4.3250    0.0001     0.0029     0.0001     0.0090
      4.6460    0.0001     0.0029     0.0001     0.0090
      4.9420    0.0001     0.0029     0.0001     0.0090
      5.2480    0.0001     0.0029     0.0001     0.0090
      5.4950    0.0001     0.0029     0.0001     0.0090
      0.2520    0.0001     0.0030     0.0001     0.0100
      1.5590    0.0001     0.0030     0.0001     0.0100
...................................................
      3.8380    0.0010     0.0010     0.0011     0.2240
      4.1960    0.0010     0.0010     0.0011     0.2240
      4.4840    0.0010     0.0010     0.0011     0.2240
      4.5850    0.0010     0.0010     0.0011     0.2240
      4.7950    0.0010     0.0010     0.0011     0.2240
      4.8840    0.0010     0.0010     0.0011     0.2240
```

The first element (5248) is the serial index that this triplet had in the input array (**ang**), i.e., this triplet is the 5248th row of **ang**. The last element (9) represents the unique group to which this triplet belongs. We have more than one triplet with the same angles, but all of them are characterized by this number (9).

Figure 3.8 : Example of the output array **sor** for N = 6.

```
           1             1
           2             1
           3             1
           4             1
.........................
        5245            87
        5246            27
        5247             1
        5248             9
        5249            16
        5250            28
.........................
        7136             1
        7137             1
        7138             1
        7139             1
        7140             1
```

The same triplet # is used here for demonstration, as in figure3.8: Format of the output (array **sor**).

Figure 3.9 : Example of the lookup table created by **indexing**.

1	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

1	1	1	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Figure 3.10 : Binary values arrays representing the letters 'C' and 'T'.

a i-1, j-1	b i-1, j	c i-1, j+1
d i, j-1	p i, j	e i, j+1
f i+1, j-1	g i+1, j	h i+1, j+1

Figure 3.11 : Neighborhood mask for the 'salt and pepper' smoothing algorithm.

eliminates small 'bumps' along straight-edge segments. For each pixel the following expression is evaluated

$$B=p\cdot[(a+b+d)\cdot(e+g+h)+(b+c+e)\cdot(d+f+g)] \tag{3.3}$$

In order to avoid boundary condition problems at the edges, the test image is framed in an (N+2)x(N+2) array where the perimeter extra columns and rows are filled with 0's. The MatLab function written for this function is ***salt_pep(C,t)***, where C is the array and t represents a threshold value used to binarize the array C. Further parameter information is given in the following sections dealing with the perimeter detection algorithms.

Perimeter Detection

It has been proven that, for feature-based identification using HONN's, it is sufficient to deal with the features of the perimeter points only [3,9,16,23-25,30,43-45,51]. For the detection of boundary points, there are several techniques in use, mainly based on local array operators (usually 3x3). Figures 3.12 and 3.13 illustrate a 'test' image consisting of a 4x4 square of value '1' placed upon a 16x16 image grid which will be used in the following approaches to boundary detection for illustration puposes

A representative, local array operator is the Sobel operator given by the following equation:

$$S(x)=\left|(\mathbf{x})*\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\right|+\left|(\mathbf{x})*\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}\right| \tag{3.4}$$

where **x** is the 3x3 array representing the image and where the operator is applied locally on every pixel. Figures 3.14 and 3.15 show the edge detection result using the Sobel operator on the test image of figure 3.12. An alternate set of operators are those proposed by Plessey Semiconductors and implemented in hardware by the same company (PDSP 16401 2-D Edge Detector). This set consists of four operators and the result is the largest of the four convolutions of the input image with each of the four operators:

Horizontal

$$1.5*\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix}$$

Vertical

$$1.5*\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}$$

(Diagonal) 45^0

$$\begin{bmatrix}2 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -2\end{bmatrix}$$

(Diagonal) -45^0

$$\begin{bmatrix}1 & -1 & -2\\ 1 & 0 & -1\\ 2 & 1 & 0\end{bmatrix}$$

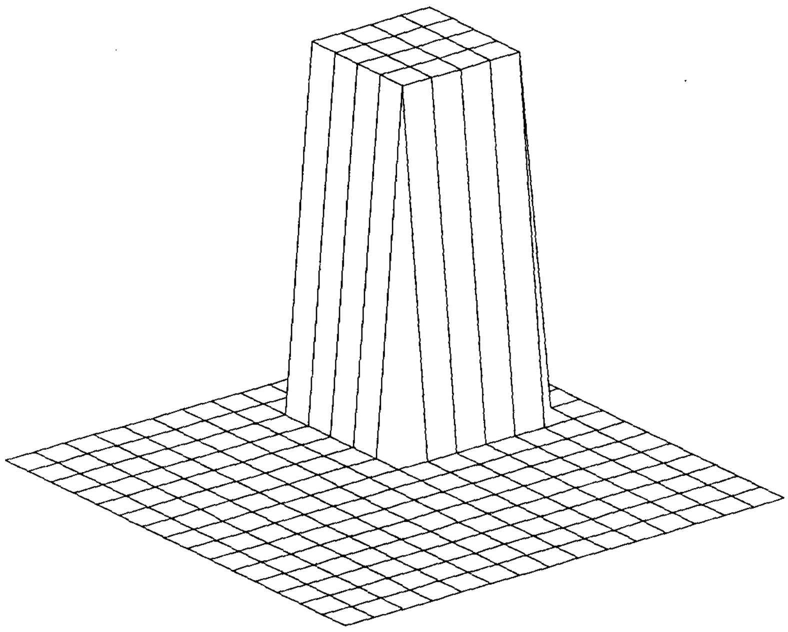

Figure 3.12 : Test image for boundary detection algorithms.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.13 : Binary array representation for the test image of figure 3.12.

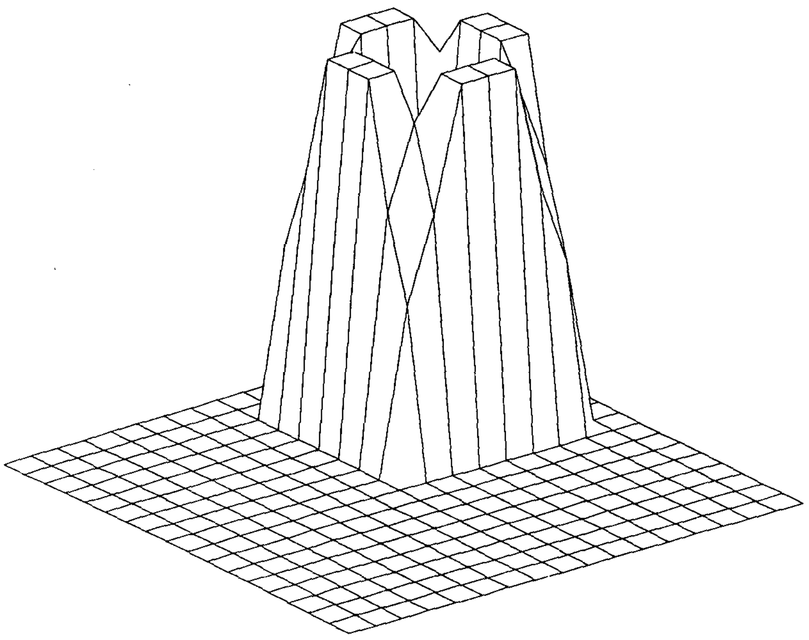

Figure 3.14 : Sobel operator boundary detection for the test image of figure 3.12.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	3	4.5	4.5	4.5	3	2	0
0	0	0	0	0	0	0	0	3	4	4.5	4.5	4.5	4	3	0
0	0	0	0	0	0	0	0	4.5	4.5	0	0	0	4.5	4.5	0
0	0	0	0	0	0	0	0	4.5	4.5	0	0	0	4.5	4.5	0
0	0	0	0	0	0	0	0	4.5	4.5	0	0	0	4.5	4.5	0
0	0	0	0	0	0	0	0	3	4	4.5	4.5	4.5	4	3	0
0	0	0	0	0	0	0	0	2	3	4.5	4.5	4.5	3	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.15 : Binary array representation of the boundary detected image of figure 3.14.

The above operators detect vertical, horizontal, and diagonal edges while the factor 1.5 is used to offset the differences between points lying on horizontal or vertical axis and points on a diagonal edge (1.5 is the approximation used in hardware for $\sqrt{2}$). Results for the application of the Plessey operators on the test image are shown in figures 3.16 and 3.17.

Due to the nature of the relatively low resolution images used in this study, such algorithms are considered to be inappropriate in the present case since such edge detection operators are essentially differential operators and tend to produce a "double edge" phenomenon. In a high resolution environment, this may be acceptable but is highly undesirable when the image resolution is low.

An algorithm much more suited to low resolution images has been developed based upon the examination of a 3x3 image area and is given in figure 3.18. The truth-table given in figure 3.19 shows under which conditions the pixel (x,y) (with intensity F(x,y)) is considered to be a point on the perimeter. One further condition for this point to be a point of the perimeter is to have an intensity value larger than a defined threshold which, in the case of a binary image, would be F(x,y)=1. The following function can be derived from this table:

$$\overline{\mathrm{F}}_{per} = \bar{a}b\bar{c}d\cdot(e\bar{g}h+egh)+\bar{a}bcd\cdot(e\bar{g}h+egh)+ab\bar{c}d\cdot(e\bar{g}h+egh)+abcd(e\bar{g}h+egh) \qquad (3.6)$$

Simplifying leads to

$$\overline{F_{per}} = e\cdot h\cdot b\cdot d \Leftrightarrow F_{per} = \overline{ehbd} = \bar{e}+\bar{b}+\bar{h}+\bar{d} \qquad (3.7)$$

and, with the threshold condition that F(x,y) =1, the final equation can be derived as

$$\mathbf{F_{per}= F(x,y)AND\{NOT(e)\ OR\ NOT\ (b)\ OR\ NOT\ (h)\ OR\ NOT(d)\}} \qquad (3.8)$$

The corresponding M-function that implements the above is ***perim(f)***. The result of using this edge detection algorithm on the test image of figure 3.12 is shown in figures 3.20 and 3.21.

Perimeter Encoding

A further algorithm has been developed in order to "encode" the points of the perimeter. This "perimeter encoding" reduces the number of the points from which features of the images need to be extracted in such a manner that, as proved in this work, the remaining points after the encoding can produce all the features necessary for pattern classification. The function for the encoding has been derived using the same neighborhood 3x3 area as shown in figure 3.22.

We encode the points of the perimeter that constitute corners and we reject the perimeter points that lay on contiguous, straight line segments of the perimeter. Under the above assumptions and using a table similar to that shown in figure 3.22, the corresponding function that performs this operation has been found to be

$$\overline{\mathrm{F}}_{enc\_per} = \mathrm{F}(x,y)\cdot\overline{((\mathrm{a}\cdot i)+(c\cdot g)+(de)+(bh))}. \qquad (3.9)$$

It should be noted here that the function as described applies to the array that represents the perimeter points only and not to the initial image. The corresponding M-function that

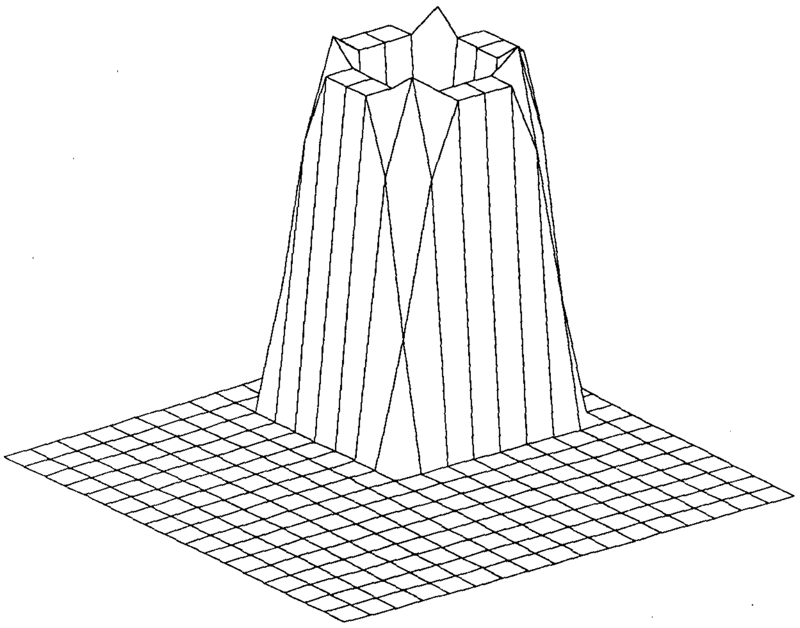

Figure 3.16 : Plessey operators boundary detection for the test image of figure 3.12.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1.4	3.2	4	4	4	3.2	1.4	0
0	0	0	0	0	0	0	0	3.2	4.2	4	4	4	4.2	3.2	0
0	0	0	0	0	0	0	0	4	4	0	0	0	4	4	0
0	0	0	0	0	0	0	0	4	4	0	0	0	4	4	0
0	0	0	0	0	0	0	0	4	4	0	0	0	4	4	0
0	0	0	0	0	0	0	0	3.2	4.2	4	4	4	4.2	3.2	0
0	0	0	0	0	0	0	0	1.4	3.2	4	4	4	3.2	1.4	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.17 : Binary array representation of the boundary detected image of figure 3.16.

a (x-1,y-1)	b (x-1,y)	c (x-1,y+1)
d (x,y-1)	f(x,y)	e (x,y+1)
g (x+1,y-1)	h (x+1,y)	i (x+1,y+1)

Figure 3.18 : The 3x3 boundary detection mask developed for low resolution images.

	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd	abcd
eghi	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0001	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0010	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0011	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0100	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0110	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0111	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1000	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1001	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1010	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0
1011	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0
1100	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1110	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0
1111	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0

Figure 3.19 : The 'truth table' giving the 256 possible combinations corresponding to the detection mask given in figure 3.18.

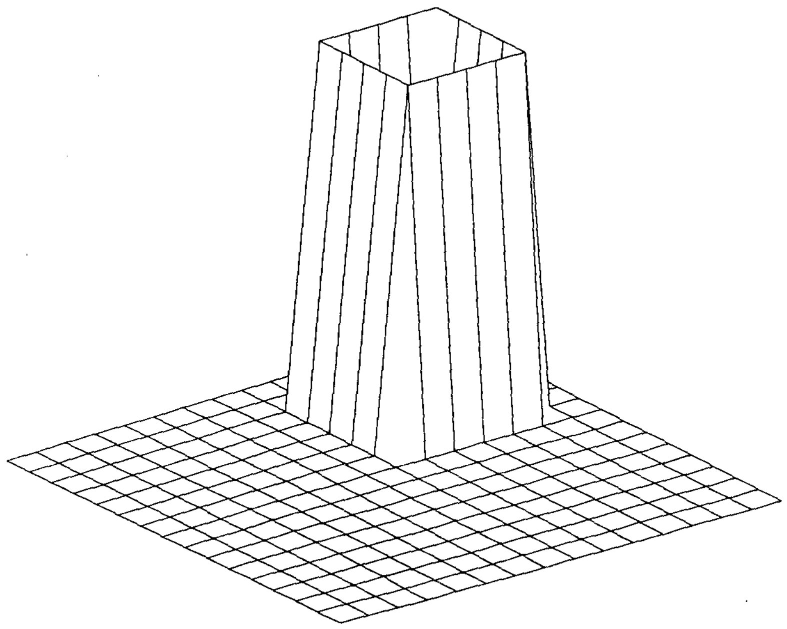

Figure 3.20 : The boundary as detected by the array mask of figure 3.18 for the test image of figure 3.12.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 3.21 : Binary array representation of the boundary detected image of figure 3.20.

a (x-1,y-1)	b (x-1,y)	c (x-1,y+1)
d (x,y-1)	f(x,y)	e (x,y+1)
g (x+1,y-1)	h (x+1,y)	i (x+1,y+1)

Figure 3.22 : The 3x3 array for encoding of the boundary detected by the array mask of figure 3.18.

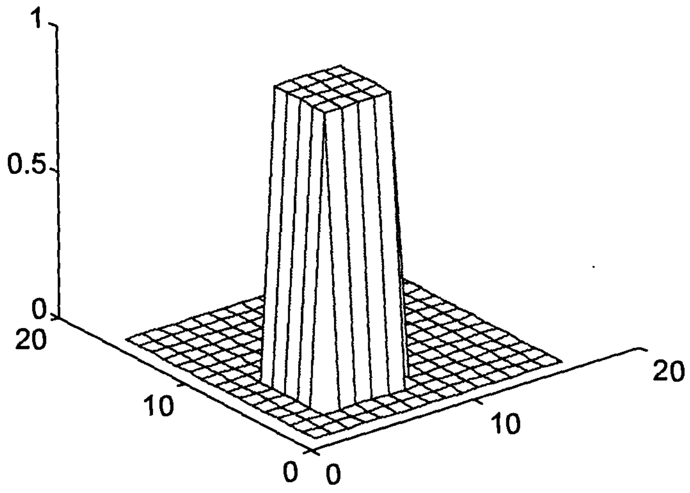

Figure 3.23 : The test image used to demonstrate the combination of boundary detection and encoding.

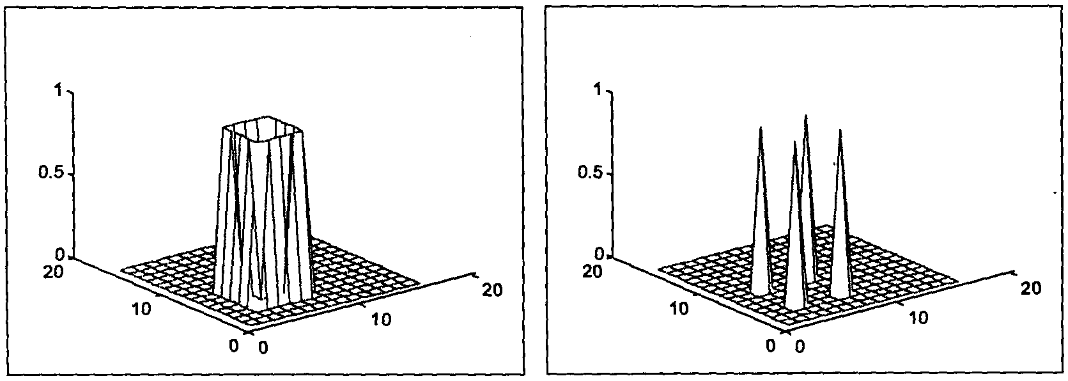

Figure 3.24 : The boundary detected (left) image and the boundary encoded image (right) corresponding to the test image of figure 3.23 and the detection/encoding array masks of figures 3.18 and 3.22.

implements the above is ***peri_con(f)***. An example of this operation of perimeter encoding is given using a 16x16 unity array shown in figure 3.23 is given by figures 3.24.

Applying the perimeter detection function only, we get the left pattern of figure 3.24 (with 16 pixels) while, applying the additional perimeter encoding scheme on this pattern, we get the right pattern with 4 pixels that can characterize the initial image. It should also be noted here that, as a guideline, the encoding technique is practical only when it is applied to images with a size of at least 10x10. Otherwise, for lower resolutions, it would cause significant reduction to the spatial information contained.

Calculation of the feature vector feat_vecX=*feat(*X,sor,index*)*

For the calculation of the feature vector for a binary image, the input arguments for the function ***feat_vec*** consist of three arrays. The first one, **X**, is the binary image resulting from the application of function ***perim*** (or ***peri_con*** if the resolution allows the employment of this function). The other two arrays, **sor** and **index**, are calculated for the particular resolution in this processing stage. The function ***feat_vec*** creates the links between the particular sets of triplets that can be formed from the pixels (with value 1) in the image and the corresponding feature number that has been assigned in this combination. For the letters **c** and **t,** the corresponding feature vectors are the variables **feat_c1** and **feat_t1** (included in the **higher6x.mat**) and are illustrated in figure 3.25.

3.3 Employment of Neural Networks

The proposed network consists of X binary inputs and Y outputs. X is the number of features that have been found in the array processing stage while Y is the number of classes to be classified. For the T-C problem in a 6x6 analysis, the number of inputs is 224 and the number of outputs 2 (since the result might be either C or T). A general, three layer (i.e., one hidden layer) feedforward neural network was given earlier in figure 2.1. Note, however, that the number of layers actually used is dependent upon the specific problem as will be explicitly indicated below. In all cases, due to the nature of the desired outputs, the output transfer function (i.e., for the output layer) used was exclusively the log-sigmoid function illustrated in figure 3.26.

Networks trained by the backpropagation algorithm were used throughout the course of this work and, as will be shown, give acceptable solutions. However, the backpropagation training algorithm can lead to a local rather than a global error minimum. If the local error minimum is not satisfactory, a network with more neurons and/or layers may perform better. For comparison purposes and solution optimization, the training of the networks has been performed employing the three different techniques offered by the Matlab neural network toolbox, namely: ***trainbp, trainbpx,*** and ***trainlm***. ***Trainbp*** employs the "classical" backpropagation algorithm with the assorted problems of local minima and slow training; the two other algorithms are improved variations of the classical one. ***Trainbpx*** employs backpropagation with the use of momentum and an adaptive learning rate. ***Trainlm*** trains the network with the Levenberg-Marquardt algorithm and is much faster than the othe two methods. However, the ***trainlm*** approach incurs memory limitation problems since it uses much more memory, typically as much memory as ***trainbp*** multiplied by **S*Q** where S is the number of neurons in the output layer and

```
feat_cl =                           feat_tl
       1                                   1
       0                                   0
       0                                   0
       0                                   0
       0                                   .
       .                                   1
       1                                   .
       .                                   0
       0                                   0
```

Both arrays have 224 rows.

Figure 3.25 : The feature vectors corresponding to the images 'C' (left) and 'T' (right).

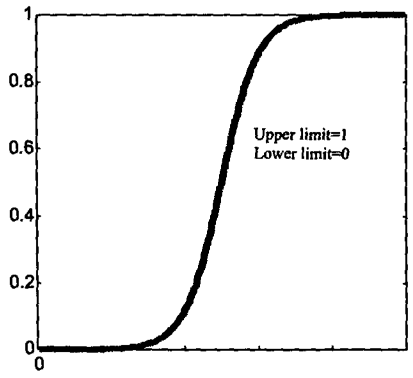

Figure 3.26 : The sigmoid transfer function used for the processing neurons.

```
T =
       1         0
       0         1
```

The first column refers to the letter 't' while the second column refers to the letter 'c'.

Figure 3.27 : Array of target vectors for the 'T-C problem'.

Q is the number of training vectors in the training set. The function ***initff*** is used for initialization of the network weights and biases while the function ***simuff*** is used to simulate the trained network. For further documentation and details for any of these functions, the reader is referred to the MatLab neural network toolbox user's guide.

T-C Problem Solution with HONN

The T-C problem has been presented previously and refers to the invariant discrimination of the letters C and T in a 6x6 resolution. The number of inputs is 224 and the number of outputs is 2. For this problem, a single-layer network will be used. **T** is the array with the target vectors shown in figure 3.27. The variables **weightl** and **biasl** are the initial weight and bias used and were calculated with the function ***initff*** as indicated in figure 3.28. The function ***trainlm*** with parameters given by figure 3.29 was found to be the fastest for this particular problem. It should be noted that the matrix P used as input is different than the P used in the initialization stage (i.e. the calculation of the initial weights and biases). The format of the P (as has been also retained in the higher6x.mat files) is shown in figure 3.30. The set training goals were attained with ***trainlm*** in only 5 epochs as evidenced by figure 3.31. In contrast, training with the ***trainbp*** algorithm was unsuccessful due to the trapping of the function in a local error surface as indicated by figure 3.32. In both cases, the training was conducted with only one training input, the matrix P (i.e. the actual feature vectors of the letters **c** and **t**). This is the unique characteristic of the HONNs. The invariance is encoded in the architecture so it is not necessary to present to the network various views of the same object.

After the training of the NN, the validity of the methodology can be verified using the patterns shown in figure 3.33. The t1 is the initial representation for letter **t**, the t_enl is a scaled and translated **t**, while the t_45 is a 45 degrees rotated and translated **t**. Corresponding names have been given to the representations of **c**. The function ***simmuf*** is used to simulate the network performance with the results given in figure 3.34.

Alphabet Recognition using HONN

The next experiment was to verify the validity of the methodology using a more accurate representation of the letters and, ideally, to be able to recognize all the 26 letters of the alphabet. While for the T-C problem there are references in the literature, for the full alphabet problem there is no reported success with HONN's. The way in which the problem has been approached is to represent all of the letters of the alphabet with a 7x5 resolution as can be seen in figure 3.35. The resolution in which those characters will be examined has been set to 8x8, so the characters will be presented to the network "framed" in a 8x8 window. For this resolution, a new set of array processing steps has to be taken, as has been described, in order to calculate the initial arrays that correspond to the 8x8 analysis. All the results, following the convention made for the filenames, have been stored under the name **higher8x.mat**. An 8x8 frame, with the given granularity, results in 441 discrete features so that the feature vector representing the features for a letter is a column vector with 441 rows. Consequently, the number of inputs for the network will be 441 and the outputs will be 26 (i.e., the number of letters in the alphabet). One layer networks have been found to be inadequate for this application as can be seen in figure 3.36. The training for this one layer network has been performed with the ***trainbp*** function since the

```
    [weight1,bias1] = INITFF(P,S1,'logsig')
      P  - 224x2 matrix of input vectors, where for the initialization
only each ith row of P must contain expected min and max values for the
ith input, i.e. 0 and 1.
    Si - Size of ith layer, here S1=2 (2 neurons output-layer)

P=
       0      1
       0      1
       .......
       0      1
       0      1
```

Figure 3.28 : Initialization values for the input vectors, training weights, and bias values.

```
[weight4,bias4,TE,TR] = TRAINLM(weight1,bias1,'logsig',P,T,TP)
   P - RxQ (224x2) matrix of input vectors.
   T  - SxQ (2x2)matrix of target vectors.
   TP - Training parameters (optional).
Returns new weight and bias and
   TE - the actual number of epochs trained.
   TR - training record: [row of errors]

Training parameters are:
   TP(1) - Epochs between updating display, here = 5.
   TP(2) - Maximum number of epochs to train, here = 15000.
   TP(3) - Sum-squared error goal, here = 0.001.
   TP(4) - Minimum gradient, here = 0.0001
   TP(5) - Initial value for MU, default = 0.001.
   TP(6) - Multiplier for increasing MU, default = 10.
   TP(7) - Multiplier for decreasing MU, default = 0.1.
   TP(8) - Maximum value for MU, default = 1e10.
Missing parameters and NaN's are replaced with defaults.
```

Figure 3.29 : Training parameters employed for the MatLab neural network algorithm **trainlm**.

```
P=
       1      1
       0      0
       0      0
       .......
       1      0
       0      0
       .......
       0      0
       0      0
```

The first column is the feature vector of t (i.e., feat_t1) and the second column is the feature vector for c (i.e., feat_c1).

Figure 3.30 : The feature vectors for the images of 'C' and 'T'.

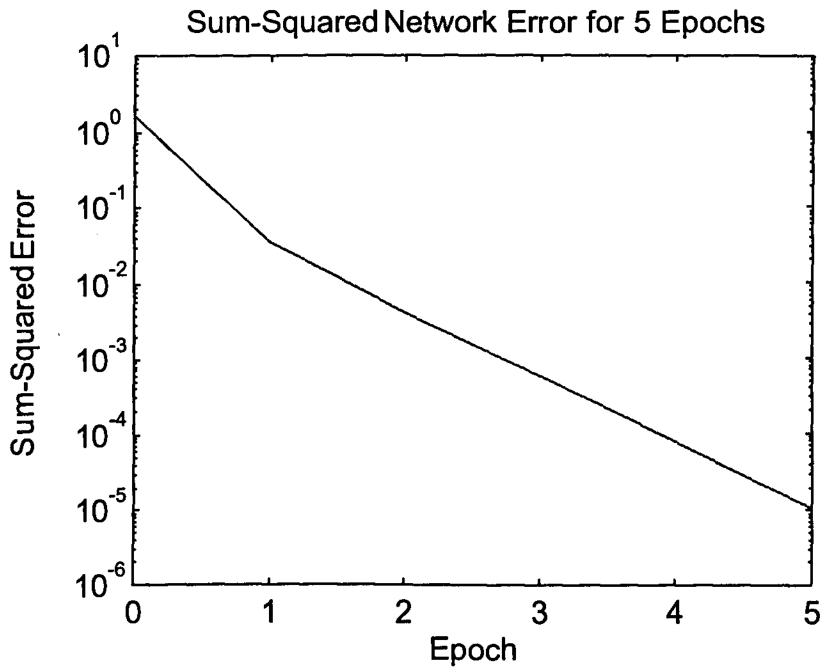

Figure 3.31 : Successful training of the network using **trainlm**.

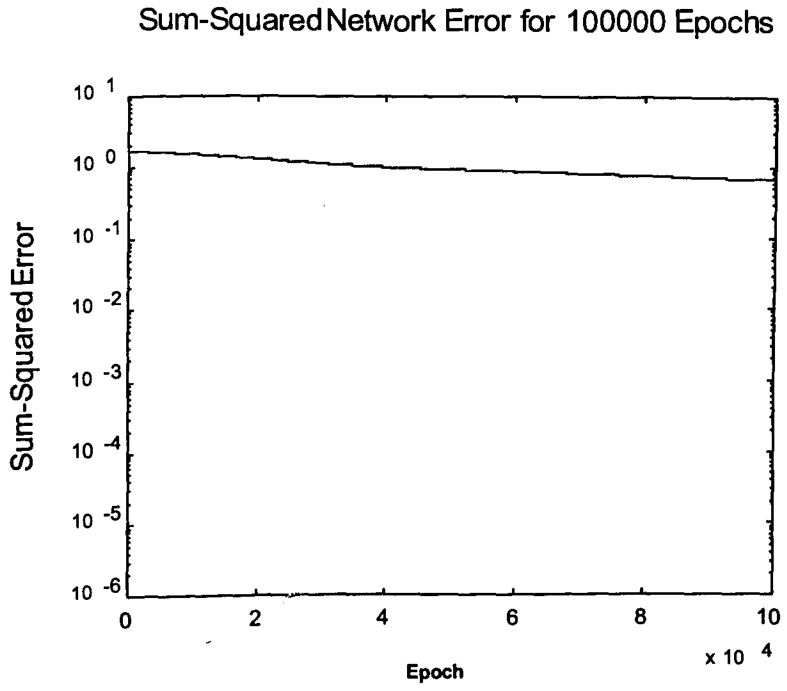

Figure 3.32 : Example of an unsuccessful training run using the algorithm **trainbp**.

```
t1 =
     1     1     1     0     0     0
     0     1     0     0     0     0
     0     1     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

t_enl =
     1     1     1     1     1     0
     0     0     1     0     0     0
     0     0     1     0     0     0
     0     0     1     0     0     0
     0     0     1     0     0     0
     0     0     0     0     0     0

t_45 =
     0     0     1     0     0     0
     0     1     0     0     0     0
     1     1     1     0     0     0
     0     0     0     1     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

```
c1 =
     1     1     0     0     0     0
     1     0     0     0     0     0
     1     1     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

c_enl =
     1     1     1     0     0     0
     1     0     0     0     0     0
     1     0     0     0     0     0
     1     1     1     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

c_45 =
     0     1     0     0     0     0
     1     0     0     0     0     0
     0     1     0     1     0     0
     0     0     1     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Figure 3.33 : Binary array representation of various test images used to validate the trained network.

```
simuff (feat_vec(c1,sor,index),weight4,bias4,'logsig')
ans =
    0.0060
    0.9940

simuff (feat_vec(c_enl,sor,index),weight4,bias4,'logsig')
ans =
    0.0004
    0.9966

simuff (feat_vec(c_45,sor,index),weight4,bias4,'logsig')
ans =
    0.0060
    0.9940

simuff (feat_vec(t1,sor,index),weight4,bias4,'logsig')
ans =
    0.9942
    0.0058

simuff (feat_vec(t_enl,sor,index),weight4,bias4,'logsig')
ans =
    0.8940
    0.0368

simuff (feat_vec(t_45,sor,index),weight4,bias4,'logsig')
ans =
    0.9852
    0.0127
```

Figure 3.34 : Network results for the test images of figure 3.33.

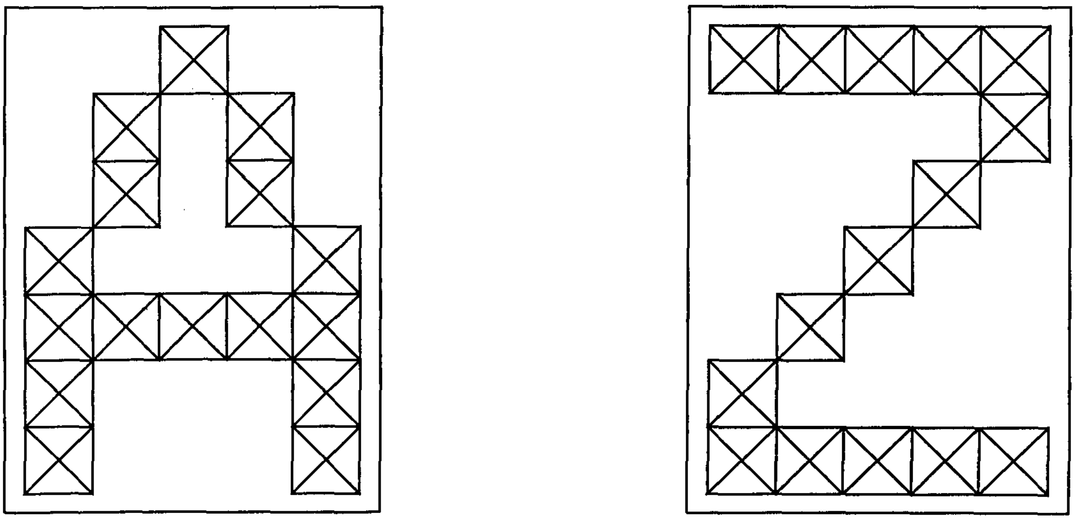

Figure 3.35 : Examples of letters defined on a 7x5 matrix.

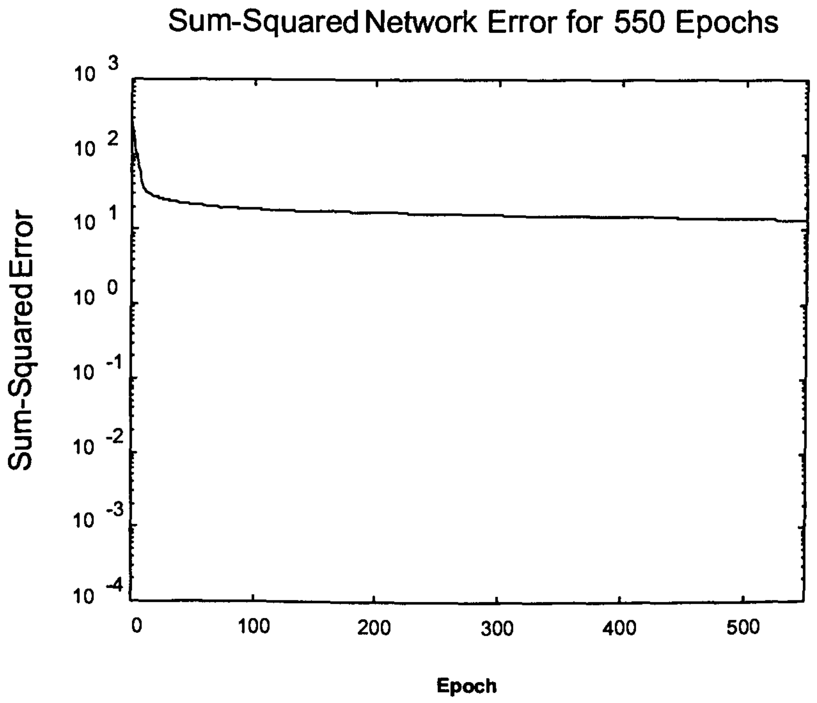

Figure 3.36 : Unsuccessful training of a single layer network.

memory requirements for ***trainlm*** are more than what was available. The variables **weightl**, and **biasl** are the initial weight and bias used and were calculated with the function ***initff*** as given in figure 3.37. In both cases (***trainbp, trainbpx***), the training was performed with **[weight_1,bias_11,TE,TR]=** ***trainbp***(x)(**weight_1,bias_1,'logsig',feat_alp,targets,TP**) where **targets** is a 26x26 array which contains the targets, i.e., one column for each letter as shown in figure 3.38. **Alp_feat** is a 441x26 array that contains the feature vectors for each letter as extracted with the function ***feat_vec.m***. For example, the first column of the matrix which represents the higher order feature vector for the letter **a** has been calculated using the function ***feat_vec*(char_dis(alphabet(:,1)), sor,index).** Training with ***trainbx*** performed a bit better but was still inadequate as can been seen in figure 3.39 for which ***trainbpx*(weight_1,bias_11,'logsig',alp_feat,targets,TP).**

The employment of a three layer network solved the problem and the optimum number of neurons in the hidden layer was found to be 18, figure 3.40. Figure 3.41 shows the response of the network during the first training epochs. The performance of the network is exceptional (recognition rate 100%) as long as it operates in a noiseless environment and the letters are presented in any translational variation within the 8x8 frame. There is some degradation in the performance when the network operates with rotated and scaled representations of the letters, which is expected, due to the low resolution and the accompanying problems mentioned in the section describing the sorting of the angles. The performance falls to 87 % which at the same time shows the performance of the net with this particular type of noise. Figure 3.42 shows a typical case where the network failed to recognize the letter C. The network was trained with the features of the letter as in the leftmost block but, for input, used the scaled representation pictured in the middle block. The result is as shown in figure 3.43, i.e., the network in this particular case didn't find any close proximity with any of the letters although the best candidate (with a very low score though =0.0164) was the letter O. The similarity, and the reason why this type of confusion was expected stems from the way the letter C was scaled. The conclusion is that, in low resolutions, experimentation with granularity must be carried out in order to force the network to perform accurately.

Feature Extraction of SAR Data.

There is the general assumption that HONNs, by definition, can perform only with binary representations of the images to be classified. This results in a loss of dynamic information since the representation now is limited to black and white while a typical SAR image can have more than thousands of gray levels. Hence it is essential to find a way to encode the grayness variation in the HONN architecture. Figure 3.44 shows a target as acquired by SAR radar and segmented in a 25x32 frame. Every pixel is encoded with 4 bytes, and the values in this particular segment vary between 17 and 7863 as can be seen in figure 3.45 The same target after a binarization process is shown in figure 3.46 while the corresponding representation is given in figure 3.47. The loss of information is obvious and future efforts have to be directed towards this problem. The approach proposed in this work, in order to take advantage of the dynamic range encoded in SAR images is implemented in the function ***feat_gry*(f,sorted,index)** which generates the corresponding feature column vector for the gray scale array given in (**f**). The algorithm works by firstly creating a lookup table with indices corresponding to the triangle coordinates that form all the combinations of the non-zero points of (**f**) taken 3 at a time. The result is expected to have

```
[weight_1,bias_1] = INITFF(P,S1,'logsig')

        P  - 441x2 matrix of min-max input vectors.
        S1=26
  P=
      0      1
      0      1
      ........
      0      1
      0      1
```

Figure 3.37 : Initialization values for the training run.

```
targets =
   1     0     0     0     0     0     0     0     .
   0     1     0     0     0     0     0     0     .
   0     0     1     0     0     0     0     0     .
   0     0     0     1     0     0     0     0     .
   0     0     0     0     1     0     0     0     .
   0     0     0     0     0     1     0     0     .
   0     0     0     0     0     0     1     0     .
   0     0     0     0     0     0     0     1     .
   0     0     0     0     0     0     0     0     .
   0     0     0     0     0     0     0     0     .
   0     0     0     0     0     0     0     0     .

   .     .     .     .     .     .     .     .     .
```

The first column is the desired output for the letter a, the second column is for the letter b, etc.

Figure 3.38 : The format of the 26x26 targets array.

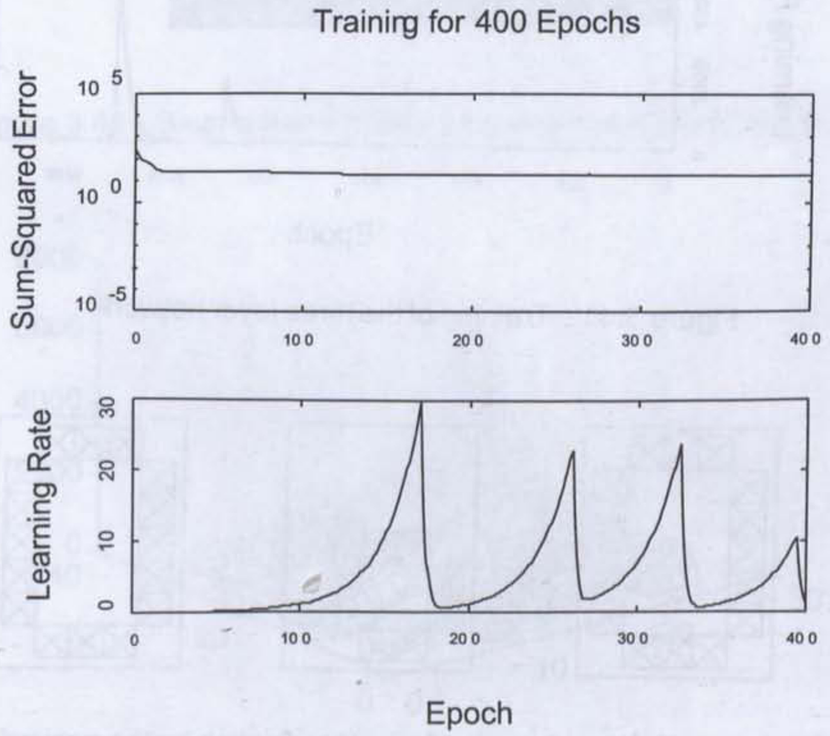

Figure 3.39 : Unsuccessful network training using trainbx and a single layer network.

Sum Squared Error for a 2-layer Back-Propagation Network		
# of neurons (hidden layer)	**SSE**	**EPOCHS**
10	>2	10,000
12	1.4	10,000
16	0.9	10,000
18	0.000998228	10,000

Figure 3.40 : Optimization of the number of hidden neurons.

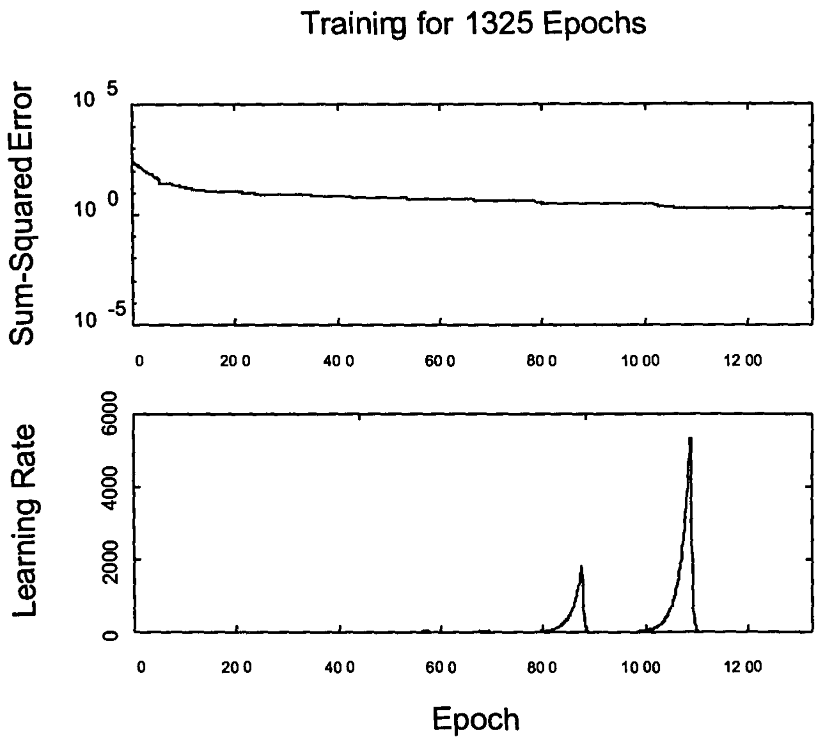

Figure 3.41 : Training of the three layer network.

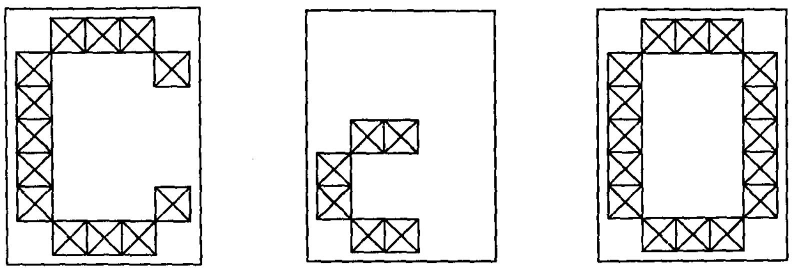

Figure 3.42 : An example of misclassification by the network for a scaled 'C' as the letter 'O'.

```
ans =
  Columns 1 through 7
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0035    0.0000
  Columns 8 through 14
    0.0000    0.0000    0.0105    0.0000    0.0007    0.0000    0.0046
  Columns 15 through 21
    0.0164    0.0001    0.0000    0.0001    0.0005    0.0000    0.0000
  Columns 22 through 26
    0.0069    0.0000    0.0000    0.0042    0.0000
```

Figure 3.43 : Results for the classification of the scaled 'C' (center image) of figure 3.42.

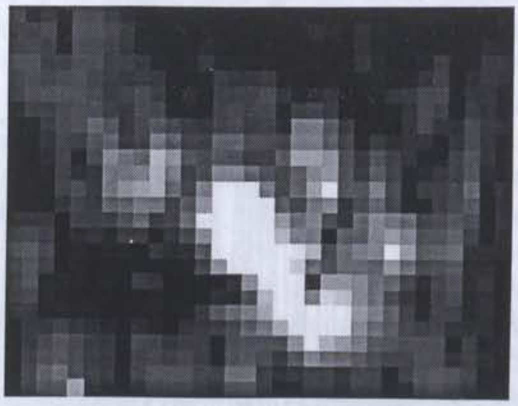

Figure 3.44 : Segmented SAR image consisting of 25x32 pixels.

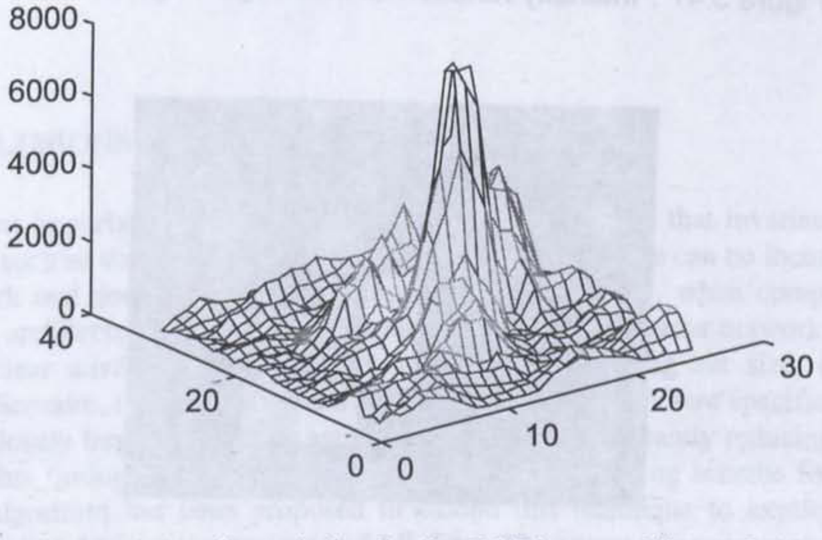

Figure 3.45 : Intensity variation for the SAR image of figure 3.44.

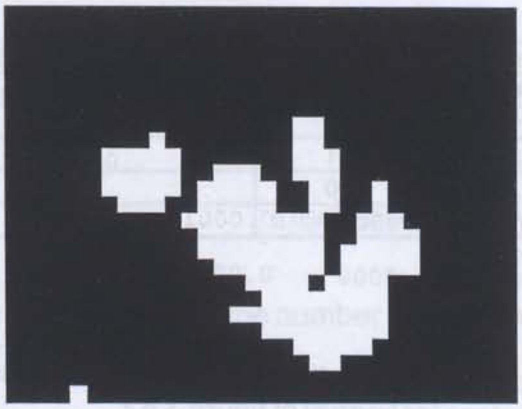

Figure 3.46 : Conversion of the original SAR image of figure 3.44 to a binary valued image.

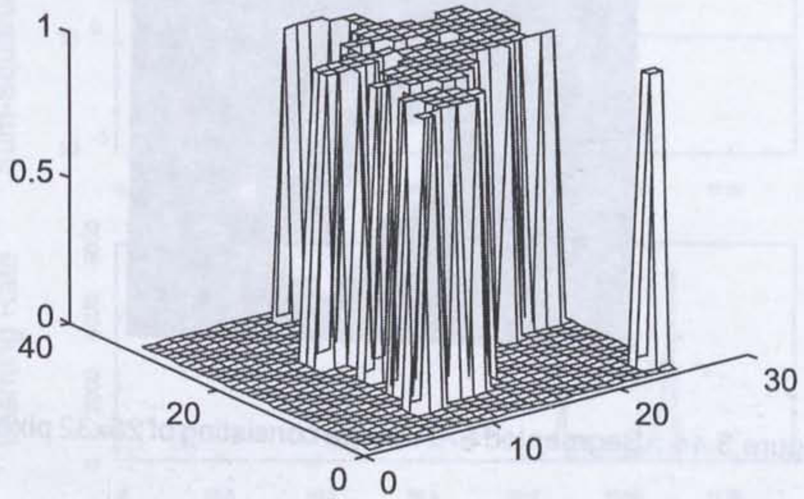

Figure 3.47 : Intensity variation for the image of figure 3.46.

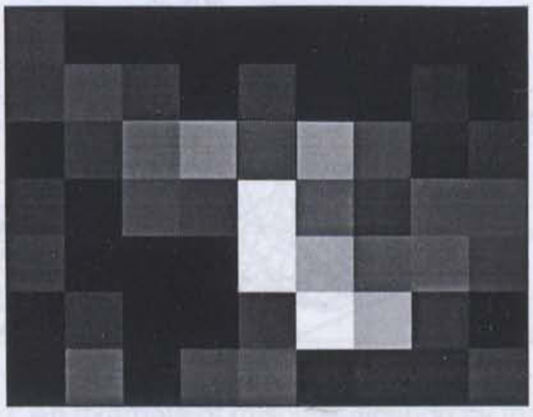

Figure 3.48 : Resizing of the 25x32 SAR image of figure 3.44 to a 6x8 image.

combin*(k,3)** rows, where **k** is the number of non-zero elements. Then, it gives the feature vector number, provided the array **index**, that represents the final result of the ***indexing3 function on an array MxN. The final column vector is formed by multiplying the normalized values of the three pixels. Since two or more triplets could correspond to the same feature number (forming similar triangles but having different intensities), the maximum result of the internal multiplication is kept in order to represent the magnitude of the particular feature. The 'sorted' parameter in the input argument stands for the array given after applying the ***sorting3*** function (on the initial array). The result is the feature vector of the target, but the values are not binary, as has been done so far, but range between 0 and 1 thus giving information not only for the existence or not of the particular triplet but for its intensity as well.

In order to demonstrate the operation of the above proposed algorithm, the same target shown in figure 3.44 has been chosen. In order to deal with the 25x32 resolution, which with the existing means is considered to be 'high', the target has been resized (using ErgoVista software) to a 6x8 representation as is shown in figure 3.48. The ratio of compression is approximately 4. The target is examined in the context of the 8x8 processing that has been performed and the results have been stored in the **higher8x.mat** file. The feature vector is expected to have 441 elements but this time their values are not limited to 0's or 1's since on top of that they can take any value in between. The function is used with the format ***feature_target_sar*** = ***feat_gry*(target_sar,sor,index)** where **target_sar** is the target framed in a 8x8 frame as given in figure 3.49. The result of the above operation has the format shown in figure 3.50.

Attention has to be paid to the way the normalization is implemented. The way the function has been encoded is that it takes into consideration as a maximum value the maximum element of the given segment which varies from segment to segment. The other way is to provide the maximum value of all the targets set, if it is known. Due to the limited number of available targets, the first approach has been adopted. If, in the near future, more data is available, the necessary changes can be made easily since they have been included in the function by means of comments. The training of the network has no practical value at the time being since there is no other data with which to compare or train.

4. CONCLUSIONS

The most important advantage of the HONN architecture is that invariance to geometric transformation such as scaling, translation, and rotation of the image can be incorporated directly into the network and does not need to be learned. Consequently, when compared with other neural network architectures such as backpropagation trained first order networks, HONNs have demonstrated clear advantages in terms of training time, training set size, and recognition accuracy. Furthermore, the algorithms and methodology developed here specifically address the issue of anomalously large numbers of network weights by significantly reducing the number of necessary weights through a novel boundary detection and encoding scheme for the image. A further novel algorithm has been proposed to extend this technique to explicitly include the dynamic information (gray scale) present in SAR data. The extent of experimentation using SAR

```
target_sar

 Columns 1 through 6
      655        735         93         51        193        261
      341        930        835        138        742        555
      125        434       1805       1041       1118       2193
      846        294        137       1420       6193        993
      427        245        231        554       2457       3688
      179        848        574        623        554        589
        0          0          0          0          0          0
        0          0          0          0          0          0
 Columns 7 through 8
      153        281
      231        104
      694        711
     2814       1235
     1144        600
      171        895
        0          0
        0          0
```

Figure 3.49 : Array representation of the scaled down SAR image of figure 3.48 with added rows of zeros to yield an 8x8 image.

```
feature_target_sar
    0.1042
         0
    0.0153
    0.0079
    0.0047
    0.0421
..........
         0
    0.0145
    0.0016
    0.0017
    0.0057
    0.0139
         0
    0.0034
..........
    0.0051
         0
         0
    0.0009
         0
    0.0110
```

A "0" means that the particular combination has not been encountered. Values other than "0" give the 'weight' of the prominent triplet that corresponds to the particular feature number. The dimension of this column vector is 441x1.

Figure 3.50 : Feature vector for the gray scale SAR image of figure 3.48.

imagery in this work was limited owing to a lack of sufficient numbers of suitable images. However, the proposed framework is very promising and has been validated with the employment of classical pattern classification problems. The application of a HONN combined with the proposed boundary detection and encoding technique for alphabet recognition has clearly demonstrated the efficiency with which this topology can be utilized.

The principle limitation which remains for the application of HONN's, even with the perimeter detection technique developed her, is that the size of the input field is limited (pragmatically to images containing, at most, a few thousand pixels). This is a technical limitation rather than an algorithmic one since it is governed by available memory size and computing throughput. The HONN architecture by itself can perform with unlimited input size. For SAR, post-segmented images which represent typical targets, an acceptable maximum size would be ~ 40x40, a number which, in the present environment, appears well suited to the types of images encountered in practice. Future efforts in this field should be directed towards the further reduction of the input size by means of compression and encoding either with conventional methods or with fractal geometry. Two specific directions worthy of serious consideration and additional research would be, (a), the development of HONN's for which the input variables are derived directly from the parameters which characterize fractal compression transformations of the images and, (b), the application of first order neural networks for which the feature vector is extracted from an explicit determination of the number, type, and intensity contents of similar triangles formed by the image pixels.

Appendix: Listing of the MatLab M-Functions

function y = angles(f);

```
% Angles(f)
% Creates an array, given in (f), containing the angles corresponding to
% the 3 lengths that form the triangles.
% The result is an array with the same dimension as (f), i.e., (Nx3).
% Care is taken to ensure that none of the 3 lengths are equal
% to 0, i.e., a*b*c .ne. 0, otherwise the result would be
% NaN type (Not-a-Number).  The routine zero_fil is used to disallow
% this possibility.
%
% Dr. Fivos Hatzivasiliou - June 1994
echo off
aux1=zero_fil(f); % Create a new array aux1 with NO null elements.
[m,n]=size(aux1); % The dim of the new array are <= dim(f).
aux2=aux1.^2;
a=aux1(1:m,1); b=aux1(1:m,2); c=aux1(1:m,3);
a2=aux2(1:m,1); b2=aux2(1:m,2); c2=aux2(1:m,3);
C=acos( (a2+b2-c2)./(2*a.*b) );
B=acos( (a2+c2-b2)./(2*a.*c) );
% A=acos( (c2+b2-a2)./(2*b.*c) );
y=abs([pi-(B+C) B C])
% The abs function is used to ensure that no imaginery part will be in the result.
```

function y = angles3(f);

```
% Angles3(f)
% Creates an array, given in (f), containing the angles corresponding to
% the 3 lengths that form the triangles.
% The result is an array with the same dimension as (f), i.e., (Nx3).
% Care is taken to ensure that none of the 3 lengths are equal
% to 0, i.e., a*b*c .ne. 0, otherwise the result would be
% NaN type (Not-a-Number).  The routine zero_fil is used to disallow
% this possibility.
%
% Dr. Fivos Hatzivasiliou - Sept 1995
echo off
aux1=f; %aux1=zero_fil(f); % Create a new array aux1 with NO null  elements.
[m,n]=size(aux1); % The dim of the new array are <= dim(f).
aux2=aux1.^2;
a=aux1(:,1); b=aux1(:,2); c=aux1(:,3);
a2=aux2(:,1); b2=aux2(:,2); c2=aux2(:,3);
C=acos( (a2+b2-c2)./(2*a.*b) );
B=acos( (a2+c2-b2)./(2*a.*c) );
```

```
% A=acos( (c2+b2-a2)./(2*b.*c) );
y=abs([pi-(B+C) B C])
% The abs function is used to ensure that no imaginery part will be in the result.
```

function y = aray_ser(f);

```
% Aray_ser(f)
% Converts the (MxN) array (f) to a vector (MxN,1) suitable for application as
% as direct input to a third order HONN.  The result is a vector as described above.
%
% Dr. Fivos Hatzivasiliou - May 1994
echo off
[m,n]=size(f);
G(m*n)=0; % Auxilliary vector from 2D to 1D.
for x=1:m
   for y=1:n
     G((x-1)*n +y)=f(x,y);
   end
end
k=m*n;
y=G;
```

function y = char_dis(f);

```
% Char_dis(f)
% Creates a 7x5 array (lines x columns) with (f) the column-vector containing
% the 35 elements which represents an alphabet letter, i.e., the alphabet(:,1) represents
% the letter a, the alphabet(:,2) the letter b, and so on.  The character is also displayed
% using the function plotchar.
%
% See also plotchar, prprob.
%
% Dr. Fivos Hatzivasiliou - Sept 1995
echo off
[m,n]=size(f);  % The initial assumption is that n=1 (column vector)
i=1;
for x=1:7
   for y=1:5
         G(x,y)=f(i);
         i=i+1;
   end
end
plotchar(f);
y=G;
```

```
function y = chr_feat(f,sorted,index);
% Chr_feat(f,sorted,index)
% Creates an array with 26 columns where each column represents the feature vector
% of the corresponding alphabet letter, i.e., the 1st column represents the feature vector
% for the letter a, the 2nd column for the letter b, and so on.  The character is also
% displayed using the function plotchar.
% In (f) is given the alphabet array (35x26), for a 7x5 analysis.
% The 'sorted' parameter in the input argument stands for the array given after applying the
% sortingX.m function (on the INITIAL array).  This array has been given a name
% such as sor, in the higherXX.mat file (e.g., an analysis 8x8 is higher8x.mat).
% See also feat_vec, char_dis.
%
% Dr. Fivos Hatzivasiliou - Sept 1995
echo off
[m,n]=size(f);  % The initial assumption is that n=1 (column vector)
no_of_feat=sorted(combin(64,3),5); % See notes.Find the size (mX1) of the feature vector
G(no_of_feat,26)=0;
i=1;
for x=1:26
        A(8,8)=0; % Here its only for 8x8 analysis.
        A(1:7,1:5)=char_dis(f(:,x));
        pause(1);
        G(:,x)=feat_vec(A,sorted,index);
end
y=G;


function y = combin(n,k)
% Combin(n,k)
% Gives the number of different combinations of n different
% elements taken k at a time without repetitions ('n choose 3').
% Uses the function n!/k!(n-k)!=n(n-1)(n-2)...(n-k+1)/1*2*...*k.
% (See also Factoria.m)
%
% Dr. Fivos Hatzivasiliou - Feb.94
y=permut(n,k)/factoria(k);
%  Ref: Advanced Engineering Mathematics, Kreyszig, p.1191


function y = distan(f);
% Distan(f)
% Gives the lengths for all possible triangles formed from the binary image given in f.
%
% Dr. Fivos Hatzivasiliou - Feb. 1994
echo off
```

```
a=neur_3v3(f);
[m,n]=size(a);
b(1:m,1:6)=a(1:m,5:10); % See function Neur_3v3 for the format
clear a;          % Here in the b there are only the coordinates
                           % a1,b1 a2,b2 a3,b3 for the triangles.
c(1:m,1:6)=[b(1:m,1)-b(1:m,3) b(1:m,2)-b(1:m,4) b(1:m,3)-b(1:m,5) b(1:m,4)-b(1:m,6) b(1:m,5)-
b(1:m,1) b(1:m,6)-b(1:m,2)];
clear b;
c=c.^2;
d(1:m,1:3)=[c(1:m,1)+c(1:m,2) c(1:m,3)+c(1:m,4) c(1:m,5)+c(1:m,6)];
clear c;
result=sqrt(d);
y=result;
```

function y = distan2(f);

```
% Distan2(f)
% Gives the lengths for all possible triangles formed from the data given in (f)
% in the following form [x1 y1 x2 y2 x3 y3].
% This routine is the same as distan(f) differing only in the input format.
%
% Dr. Fivos Hatzivasiliou - June 1994
echo off
[m,n]=size(f);
b=f;                       % See function triangle2 for the format
                           % Here in the b there are only the coordinates
                           % a1,b1 a2,b2 a3,b3 for the triangles.
c(1:m,1:6)=[b(1:m,1)-b(1:m,3) b(1:m,2)-b(1:m,4) b(1:m,3)-b(1:m,5) b(1:m,4)-b(1:m,6) b(1:m,5)-
b(1:m,1) b(1:m,6)-b(1:m,2)];
clear b;
clear f;
c=c.^2;
d(1:m,1:3)=[c(1:m,1)+c(1:m,2) c(1:m,3)+c(1:m,4) c(1:m,5)+c(1:m,6)];
clear c;
result=sqrt(d);
y=result;
```

function y = distan3(f);

```
% Distan3(f)
% Gives the lengths for all possible triangles formed from the data given in (f)
% in the following form [x1 y1 x2 y2 x3 y3].
% This routine is the same as distan(f) differing only in the input format.
% This function is an optimized version of distan2(f).
%
```

```
% Dr. Fivos Hatzivasiliou - June 1994
echo off
[m,n]=size(f);
b=f;                          % See function triangle2 for the format
clear f;             % Here in the b there are only the coordinates
                              % a1,b1 a2,b2 a3,b3 for the triangles.
c(m,6)=0;
c(:,:)=[b(:,1)-b(:,3) b(:,2)-b(:,4) b(:,3)-b(:,5) b(:,4)-b(:,6) b(:,5)-b(:,1) b(:,6)-b(:,2)];
clear b;

c=c.^2;
d(m,3)=0;
d(:,:)=[c(:,1)+c(:,2) c(:,3)+c(:,4) c(:,5)+c(:,6)];
clear c;
result=sqrt(d);
y=result;
```

```
function y = drawing(G)
% Drawing(G)
% Splits the graph-window into 4 sub-windows and plots the array (G)
% in the top left quarter of the screen, the perim(G) in the top right,
% the peri_con(G) in the bottom left and, in the bottom right,
% a comparative plot.
%
% Dr. Fivos Hatzivasiliou - March 94
clg;
subplot(221),mesh(G);
title('Original Image');
p=perim(G);  % Perimeter.
subplot(222),mesh(p);
title('Perimeter');
c=peri_con(p); % Encoded (reduced) perimeter.
subplot(223),mesh(c);
title('Encoded Perimeter');
x=[sum(sum(G)) sum(sum(p)) sum(sum(c))];
for i=1:3
  yy(i)=combin(x(i),3);
end
subplot(224),semilogy(yy);
ylabel('Number of Triangles');
y=x;
```

```
function y = factoria(x)
% Factoria(x)
% Gives the factorial function of x.
% If x is < 150 then it uses the function x!=x(x-1)(x-2)...1.
% If x>150, then it uses the Stirling Formula approximation
%
% Dr. Fivos Hatzivasiliou - Feb.94
if x<=150,
  m=1;
  for i=1:x
    m=m*i;
  end
else
m=(x/exp(1))^x * sqrt(2*x*pi);  % Stirling Formula
end
y=m
%  Ref: Advanced Engineering Mathematics, Kreyszig, p.1192


function y = feat_gry(f, sorted,index);
% Feat_gry(f,sorted,index)
% Generates the corresponding feature column vector for
% the gray scale array given in (f).
% Feat_gry.m first creates a lookup table with indices the triangle coordinates
% that form all the combinations of the non-zero points of (f) taken 3 at a time.
% The result will have combin(k,3) rows, where k is the # of non-zero elements.
% This routine then gives the feature vector #, provided the array index,
% representing the final result of the indexing.m function on an array mXn.
% Normally this lookup table has been given a name like ind (in the mat file).
% The final column vector is formed by multiplying the normalized values of the three pixels.
% Since two or more triplets could correspond to the same feature (forming similar triangles but
% having different intensities), the max result of the internal multiplication is kept in order to
% represent the magnitude of the particular feature.
% The 'sorted' parameter in the input argument stands for the array given after applying the
% sorting2.m function (on the INITIAL array).  In the same way, this array is given
% a name such as ss, in the higherXX.mat file (e.g., for an analysis 4x4).
%
% Dr. Fivos Hatzivasiliou - Sept 1995
echo off
threshold=0.1; % The threshold is defined as the minimum 'intensity' value, that would
          %represent the existense of useful information (i.e. targets)
aux3=f';      % Conversion from array to vector. The invertion (') is being used in order the result
aux4=aux3(:); % to comply with the way we've defined the scanning pattern. (all the columns in
1st row, 2nd row...)
[m,n]=size(f);  % The initial assumption is that m=n
```

```
maxim=max(aux4);    % Find the max "intensity" OR maxim= maximum value given as
                    % parameter.
size_of=m*n;
no_of_feat=sorted(combin(size_of,3),5); % See notes.Find the size (mX1) of the feature vector
non_zero=find(f'); % Creates a vector with the non-zero elements of f,
                    % e.g. if f is 4x4 could give, 3, 13, 15, 16 .
[mm,nn]=size(non_zero);
if mm>2 & maxim>threshold % to avoid situations with just 2 active-pixels in the array.
                          % (i.e. No possible triangle), or no information at all.
k=1;  i=1;
aux4=aux4 ./ maxim;  % Normalisation.
feat_vec(no_of_feat)=0; % Creation of the feature vector, lenght = the # of features.
for x=1:size_of
   for y=x+1:size_of %
         for z=y+1:size_of %
               if aux4(x)&aux4(y)&aux4(z)
              G(i,1:4)=[k x y z];
                 G(i)=k; % Since we are interested only in the # of the combination.
                  %aux5(i)=index(k,2);
                   aux5=index(k,2);
                  feat_vec(aux5)=max([feat_vec(index(k,2))  aux4(x)*aux4(y)*aux4(z)]);
             i =i+1; % The above command accomodates the function that selects the most
               end        % prominent (maximum) value for the feature in the feature vector.
               k=k+1;  % So for all the similar triangles, only one value will be taken
         end                 % into account, the maximum.
   end
end
end % For the IF loop1
i=i-1;
y=feat_vec'
% To create a column vector (one column, # of features rows).
```

```
function y = feat_vec(f, sorted,index);
% Feat_vec(f,sorted,index)
% Generates the corresponding feature column vector for the binary array given in (f).
% Feat_vec.m first creates a lookup table with indices the triangle coordinates
% that form all the combinations of the non-zero points of (f) taken 3 at a time.
% The result will be combin(k,3) rows, where k is the # of non-zero elements.
% This routine then gives the feature vector #, provided the array index,
% representing the final result of the indexing.m function on an array MxN.
% Normally this lookup table is given a name such as ind (in the Matlab file).
% The 'sorted' parameter in the input argument stands for the array given after
% applying the sorting2.m function (on the INITIAL array).  In the same way, a name such
% as ss is given in the higherXX.mat file (e.g., for an analysis 4x4).
```

```
%
% Dr. Fivos Hatzivasiliou - June 1995
echo off
aux3=f';     % Conversion from array to vector. The invertion (') is being used in order the result
aux4=aux3(:); % to comply with the way we've defined the scanning pattern. (all the columns in
1st row, 2nd row...)
[m,n]=size(f); % The initial assumption is that m=n
size_of=m*n;
no_of_feat=sorted(combin(size_of,3),5); % See notes.Find the size (mX1) of the feature vector
non_zero=find(f'); % Creates a vector with the non-zero elements of f,
                   % e.g. if f is 4x4 could give, 3, 13, 15, 16 .
[mm,nn]=size(non_zero);
if mm>2 % to avoid situations with just 2 active-pixels in the array. (No possible triangle)
k=1; i=1;
feat_vec(no_of_feat)=0; % Creation of the feature vector, lenght = the # of features.
for x=1:size_of
   for y=x+1:size_of %
         for z=y+1:size_of %
                  if aux4(x)&aux4(y)&aux4(z)
                 G(i,1:4)=[k x y z];
                    G(i)=k; % Since we are interested only in the # of the combination.
                     aux5(i)=index(k,2);
                     feat_vec(index(k,2))=1;
                i =i+1;
                  end
                  k=k+1;
          end
    end
end
end % For the IF loop1
i=i-1;
y=feat_vec'
% To create a column vector (one column, # of features rows).
```

```
function y=file_inp(name)
% File_inp('name')
% Opens and reads the file named 'name'.
%
% Dr. Fivos Hatzivasiliou
fid=fopen(name);
a=input('How many rows ?');
b=input('How many columns ?');
matrix=fread(fid,[a,b],'float');
fclose(fid);
y=matrix;
```

```
function y = grouping(f);
% Grouping(f)
% Given in (f) the 3 angles that form the triangles containing no
% zero elements since the function zero_fil was applied, this routine
% creates a new array with the same or smaller dimension than f, (Nx3),
% eliminating all the triplets having the same elements taking into account
% the relative order (e.g., a b c is different from a c b).
%
% Dr. Fivos Hatzivasiliou - July 1994
echo off
[m,n]=size(f); % The dim of the new array will be <= dim(f).
n=1;k=1;
aux(m)=0;       % Auxilliary vector m.
for i=k:m-1
        if abs(f(i,1)-f(i+1,1)) <= 0.0001,
        %       C(1:n+1,1:3)=[C(1:n,1:3)' f(i,1:3)']'; % The ' operation is very time consuming
                ind=ind+1;
        else,
                aux(n)=i;
                n=n+1;
        end
end
        if abs(f(m-1,1)-f(m,1)) > 0.0001,  % Boundary Condition, since in the
                aux(n)=m;                  % above if-loop there is no provision
        end                                % for the m-th element.
y=aux;
```

```
function y = indexing(f);
% Indexing(f)
% Creates a 2-column indexing table with the following structure:
% the first column is sorted and contains the triplet #, while the 2nd
% column gives the corresponding feature vector position.
% (f) is the result of the sorting2 routine and has in the first column the initial
% index of the triplet, (e.g., 11, for a 6x6, means the 11th combination; see also
% triangl2.m routine), while in the fifth column the no. of the feature vector,
% or set of angles, to which it corresponds.
%
% Dr. Fivos Hatzivasiliou - June 1995
echo off
[m,n]=size(f);
[Y,I]=sort(f(1:m,1));
G(m,2)=0;
  G(:,2)=f(I(1:m),5); % Sorting regarding the 1st element
  G(:,1)=[1:m]';
y=G;
```

```
function y = peri_con(f)
% Peri_con(f)
% Perimeter edge detection and encoding using our developed algorithm where (f)
% is the binary input array (dim MxN). When the perimeter is found, another
% algorithm is applied in order to find the vertex points of the perimeter.
% In order to avoid boundary condition problems, an auxilliary array dim m+2,n+2 is used.
% The result is another array (MxN) with 1's showing the perimeter vertex points.
%
% Dr. Fivos Hatzivasiliou - March 1994
echo off
% thres_points=
% The max number of perim. points that won't be converted to perimeter vertex points
[m n]=size(f).
G(m+2,n+2)=0;
G(2:m+1,2:n+1)=f;
T(m+2,n+2)=0;
C=T;  % Declaration of auxiliary arrays.
for x=2:m+1
 for y=2:m+1
  T(x,y)=(G(x,y)&(~G(x+1,y)|~G(x,y+1)|~G(x,y-1)|~G(x-1,y) ) );
 end
end
%points=sum(sum(T)); % Gives the number of the perimeter points (Binary Image).
% if points <= thres_points, C=T;
% else
 for x=2:m+1
  for y=2:n+1
      if T(x,y)==1,
       C(x,y)=~( (T(x-1,y-1) & T(x+1,y+1)) | (T(x-1,y+1) & T(x+1,y-1)) | (T(x,y-1)&T(x,y+1))
| (T(x-1,y)&T(x+1,y)) );
    end
  end
 end
%end
y=C(2:m+1,2:n+1);


function y = perim(f)
% Perim(f)
% Perimeter edge detection using our developed algorithm where (f)
% is the binary input array (dim MxN). There is also a provision for the points
% lying diagonally to be multiplied with the factor 1.414 (sqrt 2).
% In order to avoid boundary condition problems an auxilliary array dim m+2,n+2 is used.
% The result is another array (MxN) with 1's showing the perimeter points.
```

```
%
% Dr. Fivos Hatzivasiliou - Feb. 1994
echo off
[m n]=size(f);
G(m+2,n+2)=0;
G(2:m+1,2:n+1)=f;
T(m+2,n+2)=0;
for x=2:m+1
  for y=2:m+1
    T(x,y)=(G(x,y)&(~G(x+1,y)|~G(x,y+1)|~G(x,y-1)|~G(x-1,y) ) );
  end
end
%for x=2:m+1
%  for y=2:n+1
%    if (T(x-1,y-1) & ~T(x-1,y) & ~T(x,y-1) ) | ( T(x-1,y+1) & ~T(x,y+1) & ~T(x-1,y) )
%       T(x,y)=T(x,y)*1.414;
%    end
%  end
%end
y=T(2:m+1,2:n+1);
```

```
function y = permut(n,k)
% Permut(n,k)
% Gives the number of different permutations of n different
% things taken k at a time without repetitions ('n choose 3').
% Uses the function n!/(n-k)!=n(n-1)(n-2)...(n-k+1).
% (See also Factoria.m)
%
% Dr. Fivos Hatzivasiliou - Feb.94
m=1;
for i=n-k+1:n
    m=m*i;
end
y=m;
%  Ref: Advanced Engineering Mathematics, Kreyszig, p.1190
```

```
function y = salt_pep(C,t)
% Salt_pep(C,t)
% Removes from the array C (dim MxN), the isolated 1's and smooths the 'bumps'.
% Initially binarizes the C with  threshold t (i.e., if C(i,j)<=t then =0 else =1)
% Ref. Robotics pp.341.
% The result is a filtered binary array with the same dimensions.
%
```

```
% Dr. Fivos Hatzivasiliou - Feb. 1994
echo off
[m n]=size(C);
v(m+2,n+2)=0;
L(m+2,n+2)=0;
megisto=max(max(C)); %
v(2:m+1,2:n+1)=C;  %
 for i=2:m+1     %
  for j=2:n+1   %
   if v(i,j)<=t
     v(i,j)=0;
   else
     v(i,j)=1;
   end
  end
 end
 for i=2:m+1
   for j=2:n+1
     if v(i,j)==1
       a=v(i-1,j-1); %
       b=v(i-1,j);  %
       c=v(i-1,j+1);
       d=v(i,j-1);
       e=v(i,j+1);
       f=v(i+1,j-1);
       g=v(i+1,j);
       h=v(i+1,j+1);
       L(i,j)=( ( (a|b|d) & (e|g|h) ) | ( (b|c|e) & (d|f|g) )  ) | (C(i-1,j-1) > .2*megisto);
     else       % The array  L is used only to preserve the elements
       L(i,j)=0;  % of v during the loop
     end
   end
 end
y=L(2:m+1,2:n+1);     % The result has the same dim. mxn.
```

```
function [trian,dist,ang,sor,index] = segm_bat(f);
% Segm_bat(f)
% Given in (f) the segment array (i.e. a 10x10 array), we generate the five arrays
% required to proceed with HONN training.
% Attention must be paid to the way granularity is defined in the employed m.files.
%
% Dr. Fivos Hatzivasiliou Sept. 1995
echo off
trian=triangl3(f);
```

```
dist=distan3(trian);
ang=angles3(dist);
sor=sorting2(ang);
index=indexing(sor);
```

```
function y = sorting3(f);
% Sorting3(f)
% Rearranges the angles that are given in (f), in such a way that, in every row,
% the first is the smallest angle of the triplet but, at the same time,
% the order of the angles is preserved (e.g., (a,b,c)=(b,c,a)=(c,a,b)).
% Finally, it sorts the rows, deleting at the same time all the rows having
% the form 0.001 3.14 0.000 etc., i.e., any 'line' triangles (180 0 0 ).
%
% Dr. Fivos Hatzivasiliou - June 1995
echo off
granularity=0.001;
f=abs(f); % To eliminate the elements -0.00000....
[m,n]=size(f);
aux1=f;
for i=1:m
        if abs(f(i,1)-f(i,3))<granularity,      % First, we try to preserve the order.
         f(i,1)=aux1(i,1); % Then, we replace the values of i.e. i,1 and i,3 with one (i.e. i,1),
because
         f(i,2)=aux1(i,1); % we want the further procedure to deal with exact equal values. Some
inequalities
         f(i,3)=aux1(i,2); % result from the routine angles, since employement of the cos function
is made.
        elseif abs(f(i,2)-f(i,3))<granularity,
         f(i,1)=aux1(i,2);
         f(i,2)=aux1(i,2);
         f(i,3)=aux1(i,1);
        elseif abs(f(i,1)-f(i,2))<granularity,
         f(i,1)=aux1(i,1);
         f(i,2)=aux1(i,1);
         f(i,3)=aux1(i,3);
        end
        aux2=f(i,1:3); % Local aux. vector.
        [x,ind]=min(f(i,1:3)); % Sorting procedure starts here.
        if ind==2,
          f(i,1)=aux2(2);
          f(i,2)=aux2(3);
          f(i,3)=aux2(1);
        elseif ind==3,
          f(i,1)=aux2(3);
          f(i,2)=aux2(1);
       f(i,3)=aux2(2);
        else
        end
end
aux1=f; % Section that sorts the rows.
[Y,I]=sort(f(1:m,1));
l=1;  mm=1;
clear f;
I(m+1)=m+1; %Trick
aux1(m+1,1)=4; % Trick (Because 4>3.14, the max. number can be found in the array, the sort
procedure
                 % will take place also for the last set of elements.
aux2(m+1,2)=4;
aux3(m,5)=0; % Attempt to optimize the procedure, pre-specifying the array size.
for i=1:m
  aux2(i,1:3)=aux1(I(i),1:3); % Sorting regarding the 1st element

        if aux1(I(i+1),1) - aux2(i,1) > granularity % define granularity
                [Y_SUB,I_SUB]=sort(aux2(l:i,2));
                for ind=l:i
                  aux3(ind,1)=I(I_SUB(ind-l+1)+l-1);
                  aux3(ind,2:4)=aux2(I_SUB(ind-l+1)+l-1,1:3); % Because refers to A subset
                  if ind>2
                        if aux3(ind,3) - aux3(ind-1,3) > granularity % Define granularity
                        mm=mm+1; % The index (mm) here changes for the rows in WITHIN
                                    the same subset
                        end     % i.e. for the rows with the
                  end
                  aux3(ind,5)=mm;
                end
           mm=mm+1; % The index here changes for the rows WITHIN different subsets.
           l=i+1;
        end

end
y=aux3;
```

function y = tempor(f);

```
% Tempor(f)
% Creates a lookup table with indices the triplet #'s which form all possible combinations
% of 3 points plus the serial index #.  The result will have combin(m*n,4) rows.
%
% Dr. Fivos Hatzivasiliou - June 1995
```

```
echo off
[m,n]=size(f);  % The initial assumption is that m=n
size_of=m*n;
No_of_triangles=combin(size_of,3);
i=1;
for x=1:size_of
   for y=x+1:size_of %
         for z=y+1:size_of %
             G(i,1:4)=[i x y z];
             i=i+1;
         end
    end
end
i=i-1;
%tim=fix(clock)-tim;     % This timer gives the elapsed calculation time
y=G;
```

function y = triangl2(f);

```
% Triangl2(f)
%Creates a lookup table with indices the triangle coordinates which form all possible
% combinations of 3 points.  The result will have combin(m*n,3) rows.
%
% Dr. Fivos Hatzivasiliou - June 1994
echo off
[m,n]=size(f);  % The initial assumption is that m=n
size_of=m*n;
No_of_triangles=combin(size_of,3);
i=1;
for x=1:size_of
   for y=x+1:size_of %
         for z=y+1:size_of %
             G(i,1:3)=[x y z];
             i=i+1;
         end
    end
end
i=i-1
%tim=fix(clock)-tim;     % This timer gives the elapsed calculation time
% pause
aux1=ceil(G/n); % Calculation of the row index of each element
aux2=G-(aux1-1)*n; %Calculation of the column index
G(1:i,1)=aux1(1:i,1);
G(1:i,2)=aux2(1:i,1);
G(1:i,3)=aux1(1:i,2);
G(1:i,4)=aux2(1:i,2);
```

```
G(1:i,5)=aux1(1:i,3);
G(1:i,6)=aux2(1:i,3);
% from this part on we try to arrange the triangles in a clockwise direction.
for ind=1:i
        x(1:6)=G(ind,1:6);
        if x(1)==x(3)
        elseif (x(3)==x(5)) | ( (x(2)==x(4)) & (x(6)>x(4)) )
                G(ind,3:6)=[x(5) x(6) x(3) x(4)];
        elseif ( ((x(2)-x(4))/(x(1)-x(3)))*(x(5)-x(1)) + x(2) ) < x(6)
                G(ind,3:6)=[x(5) x(6) x(3) x(4)];
        end
end
y=G;
```

```
function y = triangl3(f);
% Triangl3(f)
% Creates a look-up table with indices the triangle coordinates which form all possible
% combinations of 3 points.  The result will have combin(m*n,3) rows.
% This routine is an optimized version of the routine triangl2.m above
%
% Dr. Fivos Hatzivasiliou - Sept. 1995
echo off
[m,n]=size(f);  % The initial assumption is that m=n
size_of=m*n;
No_of_triangles=combin(size_of,3);
G(No_of_triangles,3)=0;
i=1;
for x=1:size_of
   for y=x+1:size_of %
        for z=y+1:size_of %
            G(i,1:3)=[x y z];
            i=i+1;
        end
   end
end
i=i-1
%tim=fix(clock)-tim;    % This timer gives the elapsed calculation time
% pause
aux1=ceil(G/n); % Calculation of the row index of each element
aux2=G-(aux1-1)*n; %Calculation of the column index
G(:,1)=aux1(:,1);
G(:,2)=aux2(:,1);
G(:,3)=aux1(:,2);
G(:,4)=aux2(:,2);
```

```
G(:,5)=aux1(:,3);
G(:,6)=aux2(:,3);
% from this part on we try to arrange the triangles in a clockwise direction.
for ind=1:i
        x(1:6)=G(ind,1:6);
        if x(1)==x(3)
        elseif (x(3)==x(5)) | ( (x(2)==x(4)) & (x(6)>x(4)) )
                G(ind,3:6)=[x(5) x(6) x(3) x(4)];
        elseif ( ((x(2)-x(4))/(x(1)-x(3)))*(x(5)-x(1)) + x(2) ) < x(6)
                G(ind,3:6)=[x(5) x(6) x(3) x(4)];
        end
end
y=G;
```

function y = zero_fil(f);

```
% Zero_fil(f)
% Given in (f) the 3 lengths that form the triangles, this routine creates a new array
% with the same or smaller dimension than f, (Nx3), by eliminating all triangles for
% which there is one (or more) zero lengths.
% This routine is used to ensure that the 3 lengths must be different than 0,
% i.e., a*b*c .ne. 0, otherwise there is no triangle.
%
% Dr. Fivos Hatzivasiliou - June 1994
echo off
[m,n]=size(f); % The dim of the new array are <= dim(f).
pro=(prod(f'))';
ind=1;
for i=1:m
        if pro(i) ~= 0,
                C(ind,1:3)=f(i,1:3);
                ind=ind+1;
        end
end
y=C;
```

REFERENCES

1. Bachmann C. M., Musman S. A., and Schultz A., "Lateral inhibition neural networks for classification of simulated radar imagery", Int. Joint Conf. Neural Networks, Baltimore, MD, June 1992, vol. II, pp. 115-120.

2. Barnard E. and Casaent D., "Invariance and neural nets", IEEE Trans. Neural Networks **2**, no. 5, pp. 498-508, Sept. 1991.

3. Bebis G. N. and Papadourakis G. M., "Object recognition using invariant object boundary representations and neural network models", Pattern Recognition **25**, no. 1, pp. 25-44, 1992.

4. Belkasim S. O., M. Shridhar, and M. Ahmadi, "Pattern recognition with moment invariants: A comparative study and new results", Pattern Recognition **24**, no. 12, pp. 1117-1138, 1991.

5. Bradski G. and S. Grossberg, "Fast-learning Viewnet architectures for recognizing three-dimensional objects from multiple two-dimensional views", Neural Networks **8**, no. 7/8, pp. 1053-1080, 1995 (1995 Special Issue on Automatic Target Recognition).

6. Casasent D. and S. Natarajan, "A classifier neural net with complex-valued weights and square-law nonlinearities", Neural Networks **8**, no. 7/8, pp. 989-998, 1995 (1995 Special Issue on Automatic Target Recognition).

7. Casasent D. P. and L. M. Neiberg, "Classifier and shift-invariant automatic target recognition neural networks", Neural Networks **8**, no. 7/8, pp. 1117-1129, 1995 (1995 Special Issue on Automatic Target Recognition).

8. Chang C., J. Lin, and J. Y. Cheung, "Polynomial and standard higher order neural network", IEEE Int. Conf. on Neural Networks, Vol. 2, pp. 989-994, 1993.

9. Chen K., "Efficient parallel algorithms for the computation of two-dimensional image moments", Pattern Recognition **23**, no. 1/2, pp. 109-119, 1990.

10. Dudani S. A., Breeding K. J., and McGhee R. B., "Aircraft identification by moment invariants", IEEE Trans. Computers **C-26**, no. 1, pp. 39-45, Jan. 1977.

11. Fukushima K., "Analysis of the process of visual pattern recognition by the neocognitron", Neural Networks **2**, no. 6, pp. 413-420, 1989.

12. Ghosh J. and Y. Shin, "Efficient higher-order neural networks for classification and function approximation", Int. J. Neural Systems **3**, no. 4, pp. 323-350, 1992.

13. Grossberg S., E. Mingolla, and J. Williamson, "Synthetic aperture radar processing by a multiple scale neural system for boundary and surface representation", Neural Networks **8**, no. 7/8, pp. 1005-1028, 1995 (1995 Special Issue on Automatic Target Recognition).

14. Hu M., "Pattern recognition by moment invariants", Proc. IRE **49**, no. 9, pg. 1428, Sept. 1961.

15. Hu M., "Visual pattern recognition by moment invariants", IRE Trans. Information Theory **IT-8**, no. 2, pp. 179-187, Feb. 1962.

16. Jiang X. Y. and H. Bunke, "Simple and fast computation of moments", Pattern Recognition **24**, no. 8, pp. 801-806, 1991.

17. Kanaoka T., R. Chellappa, M. Yoshitaka, and S. Tomita, "A higher-order neural network for distortion invariant pattern recognition", Pattern Recognition Lett. **13**, no. 12, pp. 837-841, Dec. 1992.

18. Khotanzad A. and Y. H. Hong, "Rotation invariant image recognition using features selected via a systematic method", Pattern Recognition **23**, no. 10, pp. 1089-1101, 1990.

19. Khotanzad A. and Lu J. H., "Distortion invariant character recognition by a multi-layer perceptron and back-propagation learning", IEEE Conf. Neural Networks, San Diego, CA, July 1988, vol. I, pp. I625-I632.

20. Khotanzad A. and Lu J., "Classification of invariant image representations using a neural network", IEEE Trans. Acoustics, Speech, and Signal Processing **38**, no. 6, pp. 1028-1038, June 1990.

21. Koch M. W., M. M. Moya, L. D. Hostetler, and R. J. Fogler, "Cueing, feature discovery, and one-class learning for synthetic aperture radar automatic target recognition", Neural Networks **8**, no. 7/8, pp. 1081-1102, 1995 (1995 Special Issue on Automatic Target Recognition).

22. Lenz R., "Group invariant pattern recognition", Pattern Recognition **23**, no. 1/2, pp. 199-217, 1990.

23. Leu J., "Computing a shape's moments from its boundary", Pattern Recognition **24**, no. 10, pp. 949-957, 1991.

24. Li B. and J. Shen, "Fast computation of moment invariants", Pattern Recognition **24**, no. 8, pp. 807-813, 1991.

25. Li. B., "A new computation of geometric moments", Pattern Recognition **26**, no. 1, pp. 109-113, 1993.

26. Li Y., "Applications of moment invariants to neurocomputing for pattern recognition", Electronics Lett. **27**, no. 7, pp. 587-588, March 1991.

27. Li Y., "Reforming the theory of invariant moments for pattern recognition", Pattern Recognition **25**, no. 7, pp. 723-730, 1992.

28. Lisboa P. J. G. and S. J. Perantonis, "Invariant pattern recognition using third-order networks and Zernike moments", IEEE Int. Joint Conf. on Neural Networks 1991, Vol. 2, pp.1421-1425.

29. Minsky M. L. and S. Papert, **Perceptrons**, MIT Press, Cambridge, MA, 1969.

30. Perantonis S. J. and Lisboa P. J. G., "Translation, rotation, and scale invariant pattern recognition by high-order neural networks and moment classifiers", IEEE Trans. Neural Networks **3**, no. 2, pp. 241-251, March 1992.

31. Prokop R. J. and A. P. Reeves, "A survey of moment-based techniques for unoccluded object representation and recognition", CVGIP Graphical Models and Image Processing **54**, no. 5, pp. 438-460, Sept. 1992.

32. Ranganath H. S., D. E. Kerstetter, and S. R. F. Sims, "Self partitioning neural networks for target recognition", Neural Networks **8**, no. 7/8, pp. 1475-1486, 1995 (1995 Special Issue on Automatic Target Recognition).

33. Ravichandran A. and B. Yegnanarayana, "Studies on object recognition from degraded images using neural networks", Neural Networks **8**, no. 7/8, pp. 481-488, 1995 (1995 Special Issue on Automatic Target Recognition).

34. Reid M. B., L. Spirkovska, and E. Ochoa, "Rapid training of higher-order neural networks for invariant pattern recognition", Int. Joint Conf. on Neural Networks IJCNN (Washington D. C.), 1989, Vol. 1, pp. I689-I692.

35. Rogers S. K., J. M. Colombi, C. E. Martin, J. C. Gainey, K. H. Fielding, T. J. Burns, D. W. Ruck, M. Kabrisky, and M. Oxley, "Neural networks for automatic target recognition", Neural Networks **8**, no. 7/8, pp. 1153-1184, 1995 (1995 Special Issue on Automatic Target Recognition).

36. Roth M. W., "Survey of neural network technology for automatic target recognition", IEEE Trans. Neural Networks **1**, no. 1, pp. 28-43, March 1990.

37. Rubin M. A., "Application of fuzzy Artmap and Art-emap to automatic target recognition using radar range profiles", Neural Networks **8**, no. 7/8, pp. 1109-1116, 1995 (1995 Special Issue on Automatic Target Recognition).

38. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation", in **Parallel Distributed Processing: Explorations in the Microstructures of Cognition,** Vol.1, MIT Press, Cambridge, MA, 1986, pp. 38-362.

39. Sadjadi F. A. and Hall E. L., "Three-dimensional moment invariants", IEEE Trans. Pattern Anal. Machine Intell. **PAMI-2**, no. 2, pp. 127-136, March 1980.

40. Schmidt W. A. C. and J. P. Davis, "Pattern recognition properties of various feature spaces for higher order neural networks", IEEE Trans. Pattern Anal. Machine Intell. **15**, no. 8, pp. 795-801, Aug. 1993.

41. Shin Y. and J. Ghosh, "The Pi-sigma network : an efficient higher-order neural network for pattern classification and function approximation", Int. Joint Conf. on Neural Networks IJCNN, 1991, Vol. 1, pp. I13-I18

42. Smith F. W. and Wright M. H., "Automatic ship photo interpretation by the method of moments", IEEE Trans. Computers **C-20**, no. 9, pp. 1089-1095, Sept. 1971.

43. Spirkovska L. and M. B. Reid, "Connectivity strategies for higher-order neural networks applied to pattern recognition", Int. Joint Conf. on Neural Networks IJCNN 1990 (San Diego), vol. 1, pp. I21-I26.

44. Spirkovska L. and M. B. Reid, "Robust position, scale, and rotation invariant object recognition using higher-order neural networks", Pattern Recognition **25**, no. 9, pp. 975-985, 1992.

45. Spirkovska L. and M. B. Reid, "Coarse-coded higher order neural networks for PSRI object recognition", IEEE Trans. Neural Networks **4**, no. 2, pp. 276-283, March 1993.

46. Teague M. R., "Image analysis via the general theory of moments", J. Opt. Soc. Am. **70**, no. 8, pp. 920-930, Aug. 1980.

47. Teh D. and Chin R. T., "On image analysis by the methods of moments", IEEE Trans. Pattern Anal. Machine Intell. **10**, no. 4, pp. 496-512, July 1988.

48. Troxel S. E., Rogers S. K., and Kabrisky M., "The use of neural networks in PSRI target recognition", IEEE Conf. Neural Networks, San Diego, CA, July 1988, vol. I, pp. 593-600.

49. Waxman A. M., M. C. Seibert, A. Gove, D. A. Fay, A. M. Bernardon, C. Lazott, W. R. Steele, and R. K. Cunningham, "Neural processing of targets in visible, multispectral IR and SAR imagery", Neural Networks **8**, no. 7/8, pp. 1029-1051, 1995 (1995 Special Issue on Automatic Target Recognition).

50. Wong R. Y. and Hall E. L., "Scene matching with invariant moments", Computer Graphics and Image Processing **8**, no. 1, pp. 16-24, 1978.

51. Zakaria M. F., L. J. Vroomen, P. J. A. Zsombor-Murray, and J. M. H. M. van Kessel, “Fast Algorithm for the computation of moment invariants”, Pattern Recognition **20**, no. 6, pp. 639-643, 1987.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) COMMUNICATIONS RESEARCH CENTRE 3701 CARLING AVENUE, P.O. BOX 11490, STN H OTTAWA, ONTARIO, CANADA K2H 8S2	2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable) UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

A NOVEL APPROACH TO THE APPLICATION OF HIGHER ORDER NEURAL NETWORKS TO IMAGE CLASSIFICATION(U)

4. AUTHORS (Last name, first name, middle initial)

HATZIVASILIOU, FIVOS AND SALA, KENNETH L.

5. DATE OF PUBLICATION (month and year of publication of document) MAY 1996	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.) 62	6b. NO. OF REFS (total cited in document) 51

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

TECHNICAL REPORT OCT. 93 - DEC. 95

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)
DEFENCE RESEARCH ESTABLISHMENT OTTAWA (DREO)
3701 CARLING AVENUE, OTTAWA, ONTARIO K1A 0Z2

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant) 1410 FE 233	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)
10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) CRC-RP-96-00 2	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNLIMITED

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

THE APPLICABILITY OF HIGHER ORDER NEURAL NETWORKS TO THE CLASSIFICATION OF LOW RESOLUTION IMAGERY IS INVESTIGATED. A NOVEL BOUNDARY DETECTION AND ENCODING METHODOLOGY IS DEVELOPED IN ORDER TO SIGNIFICANTLY REDUCE THE LARGE NUMBER OF THIRD ORDER INTERCONNECTION WEIGHTS WHICH MUST BE FOUND DURING THE TRAINING STAGE OF THE NEURAL NETWORK. THE HIGHER ORDER NEURAL CLASSIFIER CAN BE TRAINED BY PRESENTING ONLY ONE SAMPLE IMAGE PER CLASS AND SO ENABLES RAPID NETWORK LEARNING WITH A MINIMAL REQUIREMENT FOR TRAINING DATA. AN EXTENSIVE, MATLAB COMPATIBLE TOOLBOX DEVELOPED SPECIFICALLY TO IMPLEMENT THIS APPROACH IS DESCRIBED AND DOCUMENTED ALONG WITH THE ALGORITHMS EMPLOYED IN THE IMAGE BOUNDARY DETECTION AND ENCODING PROCESS.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

NEURAL NETWORKS, HIGHER ORDER NEURAL NETWORKS, IMAGE CLASSIFICATION, SAR IMAGERY

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM