RECOGNITION OF SYMBOLS AND CHARACTERS ON ENGINEERING DRAWINGS AND MAPS

FADY N. SAID

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AT
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

MAY 1996

ISBN 0-612-10891-0

Abstract

Recognition of Symbols and Characters on Engineering Drawings and Maps

Fady N. Said

The need for a Geographic Information System (GIS) that analyzes scanned paper engineering drawings and maps has led many researchers - in the fields of pattern recognition, image processing, and document analysis - to consider this commercial and industrial demand as a new challenge.

My aim is to recognize different kinds of graphical symbols and alphanumeric characters, found on Engineering Drawings and Maps, where the graphical symbols are used to indicate their location, and the alphanumeric characters constitute the engineer's explanations or legends. Having different types of engineering drawings and maps - airport maps, street maps, construction drawings, etc. - will add to the complexity of the problem, along with the noise introduced to the image of the scanned document, the different orientations and scales present, and the problem of segmenting touching or overlapping symbols or characters.

In trying to solve this problem, a dynamic set of hybrid neural network classifiers based on the modified error backpropagation algorithm has been built . The classifiers are combined with some segmentation algorithms and related image processing algorithms. The networks were trained and tested on different types of drawings and maps collected from different industrial sources. One network per symbol is built so that adding a new symbol to the set of symbols to be classified doesn't affect the previously trained weights of the networks, thus resulting in a set of *independent* training and testing networks. In addition, each network has its own number of hidden neurons, values of the learning and momentum rate, and the sigmoidal function's parameter that are chosen *automatically* by the system during the training phase. Moreover, the vertical and horizontal histograms are combined with the connected

components techniques to segment a paragraph of text into separate symbols and characters.

As for the architecture of the neural networks, a method for finding the neighborhood of the optimal number of hidden neurons is discussed in chapter 3. This method was designed by experimenting on adding and removing neurons from the hidden layer while training the network. Modifying the values of the parameters of the Modified Error Backpropagation is also employed. The network never diverged on any training set tried or fell in a local minima. Consequently, segmenting paragraphs of text, extracted from engineering drawings, into separate lines and patterns fails only when a pattern is connected to another, or when a pattern is distorted, as shown in chapter 4. The performance of the networks was tested on the **CEDAR** database of numerals, where a recognition rate of 94.02% and a reliability rate of 97.24% were achieved, with an error rate of 2.66% and a rejection rate of 3.32%. When recognizing 5 symbols on a (7000x6500) binarized map using Hausdorff's method alone, 464 False Positives occur. After combining the dynamic set of hybrid neural networks with Hausdorff's method, the number of False Positives dropped down to 16. This work, as a whole, stands as a solid basis in building a hybrid GIS that would be able to process other types of documents.

Acknowledgements

Sincerely, I would like to express my deepest respect, greatest appreciation and gratitude to my supervisor Professor Ching Y. Suen for his unique constructive approach in guiding me throughout the whole scope of my thesis research conducted at **CENPARMI**, an active scientific member in the National Networks of Centres of Excellence in Canada, and one of the best known research centres in the world founded and directed by Professor Suen. In a word, *I am indebted to him throughout my lifetime.*

I would like to thank the Natural Sciences and Engineering Research Council of Canada, NSERC, the National Networks of Centres of Excellence program of Canada, the Institute of Robotics and Intelligent Systems, IRIS, the software programming company TRIGONIX Inc., and the Centre de Recherche Informatique de Montréal, CRIM, for supporting me during my my work on the thesis.

This work could have been impossible without the cooperative contribution of the industry partners of **CENPARMI**. On top of my list comes **TRIGONIX Inc.** headed by Dr. Denis Asselin, Misers. André Bouchard and Martin Guillemette the two research analysts at **TRIGONIX**, to whom I am indebted for many fruitful discussions and comments on the specifications of the whole project. This research project was intended to create a paragraph segmentation algorithm combined with a neural network classifier to process and recognize paragraphs of text extracted from engineering drawings. The duration of the project was over full year (May 1994 - May 1995).

The continuity of my research could have been impossible without the second industrial collaboration that came this time from **CRIM**, Centre de Recherche Informatique de Montréal, a scientific research centre (located in the heart of the downtown) playing the leading edge role in prompting the latest technology achieved at the university level to the demanding industry. I would like to thank Dr. Ying Li, Mr. Joseph Harb and Mr. Mark Lalonde for their comments and suggestions. Certainly,

I would like to present my thankfulness to Mr. Eric Reiher for his special comments, and most importantly for being always available to meet and discuss the different parts of the project. In this research project we intended to create a hybrid and dynamic set of neural network classifiers which will be able to recognize user-defined symbols from maps. This joint project started in June 1995 and will continue until December 1996.

I am grateful to all my colleagues at **CENPARMI**, and to the system analysts who always answered my questions and solved my problems (sometimes after midnight!) Mr. Michael Assels, Mr. Paul Gill, Mr. Stan Swiercz, Mr. William Wong, and Mr. C.L. Yu. Special thanks goes to Mr. Nick W. Strathy who helped me a lot since my very first day at **CENPARMI**.

Specially, I am very thankful to all members of **CENPARMI**, Professors, post doctoral fellows, staff, and graduate students: Dr. Sabine Bergler, Dr. Louisa Lam, Dr. T.D. Bui, Dr. Zhisong Chen, Dr. Mohamed Cheriet, Dr. Menier Gildas, Dr. Didier Guillevic, Dr. Y.S. Huang, Dr. Tonis Kasvand, Dr. Adam Krzyzak, Dr. Ke Liu, Dr. Rejean Plamondon, Dr. R. Shinghal, Mrs. Pat Kierans, Mr. William Wong, Mr. C.L. Yu, Ms. Helen Bouzianis, Ms. Wanhua Chen, Ms. Myriam Coté, Ms. Jie Ding, Ms. Monica Etwaroo, Ms. Rong Fan, Ms. Guiling Guo, Ms. Bei Li, Ms. Irene Mazis, Ms. Christine Nadal, Ms. Daini Xie, Mr. Alan Bloch, Mr. Yousef Al Ohali, Mr. Hao Chen, Mr. Jianxing Yuan, Mr. Raymond Legault, Mr. Daqing Li, Mr. George Nassar, Mr. Joseph Said, Mr. Patrice Scattolin, Mr. Nick W. Strathy, Mr. Raymond Tao, Mr. Boulos Waked, Mr. Eddie Webb, and Mr. Masood M. Zadeh.

To *Big Joe*, my one and only one brother, goes my greatest appreciation and thankfulness for being there for me, always.

Dedication

To Rita, my eternal ♡.

This Thesis
is dedicated to
Our mothers Salma and Hiam
Our fathers Nassif and Antoine
Our sweet sister Sandra
Our brothers Joseph (*Big Joe*)
Sami and his wife Tina
and Samir.

Contents

List of Figures xi

List of Tables xiv

1 Introduction 1

1.1 The problem 1

1.2 The proposal 3

2 Related Work in the Literature 4

2.1 The neural network classifiers 4

2.1.1 Introduction 4

2.1.2 Evolution and advancement 6

2.1.3 Basic components 7

3 The Neural Network Classifier 9

3.1 Introducing the accumulated knowledge notion 9

3.2 The modified error backpropagation algorithm 11

3.2.1 Notations used 12

3.2.2 Improvements on the method of Krzyzak et al. 13

3.2.3 The general architecture of the network 15

3.3 Modifications introduced 15

3.3.1 Modifying constants and parameters 18

3.3.2 Modifying the formulas 18

3.3.3 Initialization of weights and biases 21

3.3.4 The error function and the minimization process 21
3.4 Dynamic versus predefined architectures 22
3.4.1 Locally or fully connected? 22
3.4.2 The input and output layers 22
3.5 The hidden layer 23
3.5.1 Introduction to the problem 23
3.5.2 The neighborhood of the optimal number of neurons in a hidden layer 25
3.5.3 Simulation results 26
3.5.4 Comparing the results 30

4 Segmentation Algorithms Used **31**
4.1 Segmenting paragraphs of text from engineering drawings 32
4.1.1 Line Segmentation 32
4.1.2 Pattern Segmentation and Normalization 32
4.2 Segmenting map symbols 39

5 Results Achieved **47**
5.1 The CEDAR database 48
5.1.1 Training and testing the network 48
5.2 Results of segmenting and recognizing alphanumeric characters 53
5.2.1 The neural network classifier 53
5.2.2 Results 54
5.3 Results of segmenting and recognizing symbols 66
5.3.1 Definitions 66
5.3.2 The architecture of the neural network classifier 66
5.3.3 Results 69

6 Conclusion **94**
6.1 Major accomplishments 94
6.1.1 The neighborhood of the optimal number of hidden neurons . 94

6.1.2 Segmentation of paragraphs of text 95
6.1.3 Building a dynamic set of hybrid neural network classifiers . . 95
6.2 Minor drawbacks 95
6.2.1 Weights and biases 95
6.2.2 Towards the optimal number of hidden neurons 96
6.2.3 Segmentation of paragraphs of text 96
6.2.4 Map symbol recognition 96

References **97**

A Current applications based on backpropagation **102**

B The User Interface **105**

List of Figures

1	Document analysis as a multi-level problem.	2
2	The architecture of the neural network.	16
3	The graph for finding the neighborhood of the optimal number of hid den neurons.	27
4	This graph shows that the network converged to 10 hidden neurons.	28
5	An original input paragraph.	33
6	An original input paragraph.	34
7	An original input paragraph.	34
8	The original input paragraph to be segmented into separate lines.	35
9	The results of segmenting the paragraph into lines.	35
10	Removing noise while segmenting the paragraph into lines of text.	36
11	Comparison between the *Basic Vertical histogram* and the *Connected Components* methods; the first three characters *S*, *T*, and *R* were correctly segmented by both methods.	37
12	In case we had a pattern that has unconnected parts (components), then a wrong segmentation may occur, which will result to a misclas sification in the testing phase.	38
13	An original region of the map to be processed, Ellipses are the symbols of interest.	41
14	The Ellipse symbol's matches, using Hausdorff's distance method alone, found in the region of the map.	42
15	The results after passing all the Ellipse matches to a set of NNCs to filter out the False Positives.	43

16 Another original region of the map to be processed, Filled Triangles are the symbols of interest. 44
17 The Filled Triangle symbol's matches, using Hausdorff's distance method alone, found in the region of the map. 45
18 The results after passing all the Filled Triangle matches to a set of NNCs to filter out the False Positives. 46
19 The graph of the error function for training a network made up of 18x18 input binary images, 40 hidden neurons and 10 output neurons. 49
20 The graph of the error function for training a network made up of 16x16 input binary images, 80 hidden neurons, and 10 output neurons. 52
21 A paragraph tested with its respective output. 56
22 Example of a paragraph that has 2 rejected patterns, *5*, and *d*. . . . 57
23 A paragraph showing some of the segmentation problems encountered. 59
24 This tested paragraph had only one rejected pattern. 60
25 Another paragraph containing a character that was mal-segmented. . 61
26 The character *Q* was rejected because it was new to the network. . . 62
27 This tested paragraph had 3 rejected patterns before improving the CCSA. 63
28 This tested paragraph was completely recognized after improving the CCSA. 64
29 This tested paragraph no rejected patterns. 65
30 The architecture of the neural network for each symbol. 67
31 The dynamic architecture of the combined neural networks. 68
32 A part of the map that was used for creating the training sets of the symbols. 70
33 A part of the map that was used for creating the training sets of the symbols. 71
34 A part of the map that was used for creating the training sets of the symbols. 72
35 A part of the map that was used for creating the training sets of the symbols. 73

36 A part of the map that was used for creating the training sets of the symbols. 74
37 The error function graph of the Cross symbol. 75
38 The error function graph of the Ellipse symbol. 76
39 The instances of the Ellipse's False Positives that caused errors. . . . 78
40 The instances of the Ellipse symbols that were rejected. 79
41 The error function's graph of the Transformer symbol. 80
42 Some instances of the Transformer symbols present in the training set. 81
43 The instances of the Transformer's False Positives that were rejected. 83
44 The instances of the Transformer symbols that were rejected. 84
45 The instances of the Transformer symbols that caused errors. 85
46 The error function's graph of the Filled Triangle symbol. 86
47 The instances of the Filled Triangle's False Positives that caused errors. 88
48 The instances of the Filled Triangle symbol that were rejected. 89
49 The instances of the Filled Triangle symbol that were misclassified. . 90
50 An ideal sample of the 5 symbols used. 92
51 The confusion matrix as shown by the user interface. 106
52 The input and output maps as shown by the user interface. 107
53 The interface allows the user to initialize the values of the parameters and the network's architecture. 108
54 The pop up window of the training phase. 109
55 The pop up window of the testing phase. 109
56 The input and output paragraphs as shown by the user interface. . . 110

List of Tables

1 The Confusion matrix of the testing set consisting of 3000 patterns from the CEDAR database. 29
2 The recognition results of 6 classifiers compared to NNC. 30
3 The Confusion matrix of the *bad* testing set, after training on the first six subdirectories from the **CEDAR** database. 50
4 The Confusion matrix of the *good* testing set. 51
5 The classification results of the *good* and the *bad* testing sets. 51
6 The Confusion matrix of the *bad* testing set. 53
7 The recognition results of the Binary Image classifier compared to NNC. 53
8 Performance of the Cross NNC on the testing set. 72
9 Performance of the Ellipse NNC on the testing set. 77
10 Performance of the Transformer NNC on the testing set. 80
11 Performance of the Filled Triangle NNC on the testing set. 87
12 Combining Hausdorff's algorithm with DSHNNC. 91
13 Improvements to the number of False Positives. 91

Chapter 1

Introduction

1.1 The problem

Document analysis can be viewed as a multi-level problem, as shown in Figure 1. It starts by categorizing documents into different classes. For example, a scanned engineering drawing and a scanned payment slip are both documents but they fall into two different classes because the structural contents differ in both. This leads to having subclasses in the same class. For instance, under the engineering drawings class we can have subclasses for electrical and mechanical engineering drawings respectively.

The second level of complexity of the problem is the development of a software system that can automatically extract and recognize predefined models (representing lines, symbols, or alphanumeric characters) by the user from the document class. This part can be divided into the following steps: scanning and preprocessing, segmentation, and recognition.

The third level of complexity in analyzing documents is simply building a knowledge-based and a data retrieval system that can handle queries on the previously created database in the second level.

Specifically, we are tackling the second level of complexity in an attempt to present a hybrid system that can segment and recognize user predefined models in maps and engineering drawings, and that can recognize paragraphs of text containing alphanumeric characters and some special symbols.

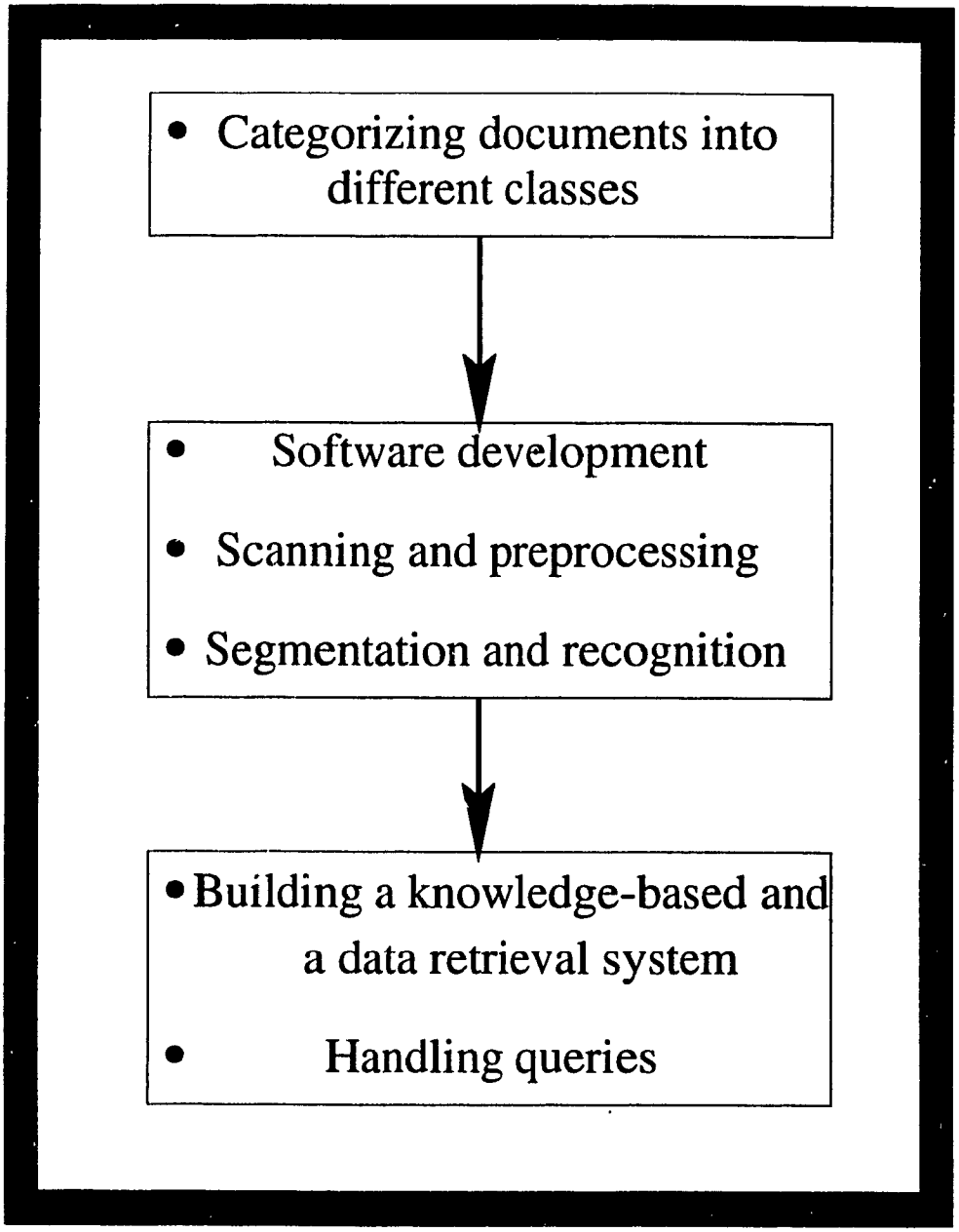


Figure 1: Document analysis as a multi-level problem.

1.2 The proposal

I claim that the key to solve the document analysis problem is to start by building a main classifier that classifies the documents into classes, then building a *class-system* for each document class respectively.

It is proposed to develop a hybrid neural network classifier (NNC') that can adapt its learning (training) process to different document subclasses in the same class, and attain a good performance status on new documents belonging to the same class. The NNC is based on the modified version of the error backpropagation training algorithm, described in Chapter 3. The architecture of the neural network (NN) is built in a dynamic way in which the number of neurons in the hidden layer is not fixed throughout the training process of the network. Moreover, the user has the ability to add new models (of characters, digits, or special symbols) to be extracted and recognized.

As for the segmentation and extraction of the model from the scanned document, we have used two methods. The first method was the *histogram* method that we used to segment paragraphs into separate lines. The second method was the *connected components* method combined with some intelligent case dependent techniques which was used in segmenting an extracted line of text into separate patterns (characters, digits, or special symbols). As a third method, the *directed Hausdorff distance* method was subsequently adapted and used for segmenting and extracting symbols from maps and Engineering drawings. Consequently, each model was then presented to a neural network classifier. The first two segmentation methods were implemented by me and used in the TRIGONIX[1] project, while the third was implemented at CRIM[2] by Eric Reiher and was used in the CRIM project solely

[1]TRIGONIX Inc. a software company, located in Montréal, specialyzing in processing engineering drawings and maps from different industrial sources.

[2]CRIM, Centre de recherche informatique de Montréal, a centre of excellence active in bridging the gap between the industry's needs and the universities' latest researches.

Chapter 2

Related Work in the Literature

The problem of automatic recognition of handwritten characters, digits, and symbols found on different types of documents has long been one of the important and challenging topics that many well known international researchers have tackled ever since the mid sixties.

2.1 The neural network classifiers

2.1.1 Introduction

For many decades, one of the goals of mankind has been to develop intelligent machines. We expect these machines to perform all complex and dangerous tasks so that we can enjoy a more fruitful life. The era of machine making began with the discovery of simple machines such as lever, wheel, and pulley. Many other inventions followed thereafter. Nowadays, engineers and scientists are trying to develop *intelligent machines*. Artificial neural networks stand as an example of such machines that have great potential to further improve the quality of our life.

The recent field of neural networks has its roots in the recognition that the brain performs computations in a different manner than do conventional digital computers. Computers are extremely precise and fast at executing sequences of instructions that have been formulated for them. A human information processing system is composed

of neurons switching at speeds about a million times slower than computer gates. Yet, humans are more efficient than computers at computationally complex tasks such as speech understanding, face recognition, and shape recognition. Unfortunately, the understanding of biological neural systems is not developed enough to address the issues of functional similarity that may exist between the biological and man-made neural systems. Although computers outperform both biological and artificial neural systems for tasks based on precise and fast arithmetic operations, artificial neural systems represent the promising new generation of information processing networks.

The ability of a neural network to perform computations is based on the aim that we can reproduce some of the flexibility and power of the human brain by artificial means. Moreover, the network computation is performed by a dense mesh of computing nodes and connections. They operate collectively and simultaneously on most or all data and inputs. *Neurons* are the basic processing elements of neural networks, (sometimes referred to as nodes or units). Neurons can be considered, in some cases, as thresholds that fire when their total input exceeds certain bias levels. They usually operate in parallel and are configured in regular architectures. Also, they are often organized in layers, where feedback connections within the same layer or toward adjacent layers are allowed. Each connection strength is expressed by a numerical value called a *weight*, which can be changed in the process of training the network.

Most neural networks must be taught or *trained*. They learn new associations, new patterns and symbols, and new functional dependencies. The *Learning* process corresponds to parameter changes. To sum up, a programmer can consequently select the architecture of the network, specify the characteristics of the neurons and initial weights, and choose the training algorithm for the network. Then, patterns from the *training set* are presented to the network as *input* so that it can acquire knowledge from the input. As a result, the network assimilates the information that can later be recalled by the network in the *testing process*.

2.1.2 Evolution and advancement

The field of neural networks, also referred to as *neurocomputing*, has an interesting past history and a solid promising future. It all started with McCulloch and Pitts, [48], in 1943 when they presented the *first formal model of an elementary computing neuron*, and included all necessary elements to perform logic operations so that the neuron functions as an arithmetic computing element. Even though at a time the model was not widely used for the vacuum tube computing hardware description, yet it laid the groundwork of the future developments.

Accordingly in 1949, the *Hebbian learning rule* was introduced by Donald Hebb, [20]. It is a method for updating neurons connections. Hebb stated that information can be stored in connections, and postulated the learning technique that had profound impact on the future developments in this field. Then, in 1958, Frank Rosenblatt, [50], introduced the *perceptron* network which was a trainable machine capable of learning to classify certain patterns by modifying its connections. His intelligent idea laid the groundwork for the basic machine learning algorithms that we still use nowadays.

Consequently, in 1960, Bernard Widrow and Marcian Hoff, [60], invented the *Widrow-Hoff learning rule* which minimized the mean squared error during training involving pattern classification. Unfortunately, research in the field of neural networks went through a slowdown because the existing machine learning theorems were too weak to support more complex computational problems, because of the modest computational resources available then. Later in 1969, Minsky and Papert disclosed the deficiencies of the perceptron model in their book called **Perceptrons**, [47]. Most of the researchers left the field except for a few including Teuvo Kohonen, Stephen Grossberg, James Anderson, and Kunihiko Fukushima.

Nevertheless, important pioneering research was conducted by some researchers. For instance, in 1980, the Japanese researcher Kunihiko Fukushima developed the *neocognitron*, a class of neural network architectures, [12], [10], [13], and [11]. Also Tuevo Kohonen developed in 1977, [30], the *associative memory* models and improved it in later publications in 1982, [31], 1984, [32], and 1988, [33]. In parallel, Stephen Grossberg developed the *adaptive resonance* networks in 1974, 1982, [17], [18]. In

1982, John Hopfield introduced the *recurrent* neural network architecture for associative memories, [26], [27]. Last but not least, S. Dreyfus pointed out the fact, in [7] and [8], that the first authors of the optimization approach for multilayer feedforward networks were Bryson and Ho in 1969, and Kelly in 1969 too.

A decade after the other, the use of neural networks in providing solutions to different applications is becoming a factual directive especially in reasoning, decision making, quality control, speech and vision systems. Add to this the fact that most universities have introduced some introductory and advanced courses that teach the principles, theory, and the algorithms' implementations of the different types of networks. I believe that neural networks will still span not only within the coming three or four decades producing new methods and algorithms, but also it will even keep on expanding in parallel to the biological understanding of the functioning of the human brain as well. The coming discoveries of new neural-based algorithms and techniques will prove my claim, because if we were to create a *human-like* machine, we should emphasize on combining the artificial intelligence principles applied in creating smart expert systems in a neural network environment. An environment that is capable to learn, generalize and most importantly *adapt* itself to new sets of data (scenes, objects, maps, symbols, etc).

2.1.3 Basic components

An artificial neural network is composed of an input layer, one or multiple hidden layers, and an output layer. The input layer is composed of units which have values (signals). They are connected, most of the time, with a bias value to the units of the layer following it. In other words, each unit receives input from the previous layer and computes its output signal, which in turn will be used as input to the following layer. Thus, we end up having three types of units:

- input units which receive data from outside the system,
- output units which send back the system's results, and
- hidden units between the input and output layers.

Calculations are executed further from one layer to another, until the results of the output layer are calculated. Then, in the training phase, using the error backpropagation algorithm, an error signal is calculated and is propagated back to the latter hidden layers, and thus adjusting the weights of the connections between the layers according to a selected learning rule. Training the network is categorized into two processes, *supervised* and *unsupervised* learning.

The process of training the network by providing it with the input patterns and their desired output is called *supervised* learning. On the other hand, the process of presenting the patterns without their desired output to the network and the network consequently classifying the patterns into different clusters is called *unsupervised* learning.

Chapter 3

The Neural Network Classifier

3.1 Introducing the accumulated knowledge notion

The terms *brain* and *mind* are often used interchangeably. When we speak of the brain, we are at times referring to the cognitive functions of the mind. On the other hand, the mind has no physical structure. Thus, cognition and the mind still largely inhabit the realm of philosophers and are only just beginning to emerge as subjects of scientific study. First, the brain was looked at as being similar to a complex telephone exchange, but this analogy changed when computers appeared.

In his book **Circuits of the mind**, Leslie Valiant states explicitly that the brain doesn't actually resemble the electronic computer, rather it is the mind, [56]. The *neurocortex*, the brain's outer layer, which has a large size relative to the rest of the brain is the main distinction between the human brain and other animal brains. It is in this folded layer, which is about 200 cm^2 in area and on average 2 mm thick, that functions associated with the mind's higher activities seem to occur, [56]. The neurocortex contains around 10^8 neurons, each with about 40,000 connections with other neurons on its output extensions (referred to as dendrites) and about the same number on its input extensions (axons). However, we should consider these configurations of the number of neurons, and their respective number of connections

as an estimate, because they differ in numbers from one human to another, especially the number of connections.

Within this context of huge number of neurons and diverse ways of interconnections, we, humans, are able to think, act, and communicate. Without the neurocortex, humans become incapable of doing explicitly anything. I claim that ever since these neurons are created in the brain of the infant (before being born), they start their own encrypted *journey* of building connections between one another through which sending and receiving signals between any two connected neurons is established. Connections between those neurons are extremely essential. Without them, neurons are incapable to communicate. Without communicating, neurons fail to learn or respond. As an analogy, neurons that fail to learn become like the memory locations which the central processing unit, CPU, of a computer fails to address and make use of. Thus, because of the everyday learning process that the brain passes through, new connections between neurons are established while some old connections are eliminated or lost.

Connections between neurons are perceived to exist in either of the following manners:

- some connections are considered to contribute to the desired results, and thus they will be strengthened by increasing the weight value, and
- some connections don't contribute to the desired results, and thus they will weaken by decreasing their weight value.

This process of updating the weights of the connection between neurons is widely labeled as the *process of learning.* This very important discovery by the neurologists have inspired many researchers, who later became the pillars of the **Neural Networks** field, to invent many neural network architectures with different types of connections and formulas for updating the connections' weights.

Still the brain can label and record, or retrieve and verify instances in a totally undiscovered manner. We can just speculate and give viewpoints on how a brain *functions* but not more. Take for instance, the classical problem of differentiating between cats and dogs. The brain can easily differentiate between the two, while it

remains as a complex problem for computers to do so. Apart from that, the human brain can recall past instances quite precisely. Not only this, but also it can add to its knowledge new instances of the same class (like two different types of animals, subclasses, that fall under the same class of say cats), or create a new class for every first-time processed instance. This leads us to conclude that once an instance is processed and validated by a human, it is labeled by his/her mind and recorded in his/her brain.

This ongoing and resident process is the cornerstone of the brain's main task. In other words, the brain and the mind together yield a new notion called the *accumulated knowledge.*

3.2 The modified error backpropagation algorithm

Backpropagation is the basis for training a supervised neural network. Static backpropagation is used to produce an instantaneous mapping of a static (time independent) input to a static output. These networks are used to solve static classification problems such as optical character recognition (OCR).

At the core of all backpropagation methods is an application of the chain rule for ordered partial derivatives to calculate the sensitivity that a cost function has with respect to the internal states and weights of a network. In other words, the term backpropagation is used to imply a backward pass of error to each internal node within the network, which is then used to calculate weight gradients for that node. Learning progresses by alternating propagation forward the activations and backward propagating the instantaneous errors.

We have implemented the modified error backpropagation algorithm (EBPA) to build a classifier for our problem. Actually, we first implemented the standard algorithm and then we went for the modified one in order to compare the results and see the difference. It was obvious that the modified algorithm used to converge with less iterations, while the standard EBPA took more iterations and sometimes it failed because of the local minima problem, and this was clearly detected while testing the convergence of the network on different training sets. However, after introducing

our modifications to the EBPA, this situation no longer detected, and thus the local minima problem was solved.

3.2.1 Notations used

Below is a list of the variables used throughout this chapter.

- $sample^j$ is the value of the input unit at the jth neuron.
- W_{12}^{ij} is the value in the weight matrix, at time T+1, between the jth input unit and the ith hidden unit in the hidden layer. Note that T+1 is the time when the current pattern is being presented to the network, time T when the previous pattern was presented, time T-1 when the pattern before the last one was presented.)
- b_{12}^{i} is the value in the bias vector, at time T+1, that connects with the ith neuron in the hidden layer.
- W_{11}^{ij} and W_{10}^{ij} are the values in the two weight matrices, at time T and T-1 respectively, between the jth input unit and the ith hidden unit in the hidden layer.
- b_{11}^{i} and b_{10}^{i} are the values in the two bias vectors, at time T and T-1 respectively, that connects with the ith neuron in the hidden layer.
- W_{22}^{ij} is the value in the weight matrix, at time T+1, between the jth hidden unit and the ith output unit in the output layer.(Please note that a similar declaration applies to W_{21}^{ij} and W_{20}^{ij}.)
- b_{22}^{i} is the value in the bias vector, at time T+1, that connects to the ith neuron in the output layer.(Please note that a similar declaration applies to b_{21}^{i} and b_{20}^{i}.)
- *nin* is the number of input units in the input layer.
- *nhid* is the number of hidden neurons in the hidden layer.

- *nout* is the number of units in the output layer.
- hid^i is the output calculated at the ith neuron in the hidden layer.
- out^i is the output calculated at the ith neuron in the output layer.
- *perr* is the error calculated of the current pattern being presented to the network.
- *terr* is the total error calculated in one complete iteration, that is, the total error of all the patterns found in the training set.
- $\Delta_{out}{}^i$ is the error signal calculated at the ith neuron in the output layer.
- $\Delta_{hid}{}^i$ is the error signal calculated at the ith neuron in the hidden layer.
- η is the learning rate.
- λ is the constant term used in the activation function.
- α is the constant term for the momentum we used.

3.2.2 Improvements on the method of Krzyzak et al.

Even though we have seen so many papers related to using the backpropagation algorithm for classifying patterns, let us refer to a paper entitled *Unconstrained Handwritten Character Classification Using Modified BackPropagation Model* by A. Krzyzak et al, [34]. At this stage, we have to admit that we didn't implement all the contents of the paper, because we wanted to introduce our own modifications to the error backpropagation algorithm, EBPA, which are described fully in this subsection.

Accordingly, we will start by briefly presenting the contents of the paper, then we will either defend or reject the modifications presented in the paper. Finally, we will present our work and show the modifications to the EBPA. This will help in introducing a new strategy in finding the neighborhood of the number of the hidden neurons, which will discussed thoroughly in a try to answer questions such as: "Why is it so important to find the neighborhood of the number of hidden neurons? How to find this neighborhood?" etc.

The modifications introduced in [34] consist of using a new delta error signal for the output neurons (units):

$$\Delta_{\mathbf{out}} = (\mathbf{desired} - \mathbf{out})$$

which means that the authors eliminated, from the standard formula, the term:

$$(\mathbf{1} - \mathbf{out}) \times \mathbf{out}$$

and deduced that by doing so, *"the local minima problem is eliminated"*. They supported their statement by saying that in the standard formula:

$$\Delta_{\mathbf{out}} = (\mathbf{desired} - \mathbf{out}) \times (\mathbf{1} - \mathbf{out}) \times \mathbf{out}$$

$\Delta_{\mathbf{out}}$ can be zero not only when **desired** = **out**, but also when **out** = 0 or 1. So the derivatives are zero, which means that the network will lose its learning ability. This new formula was tried, and after lots of testings and experiments, we found that it is only trying to bridge the gap of a specific situation that they are trying to solve which is the case when the **desired** values equal either 0 or 1. So, we started studying its generality, in the sense of: Would this formula still prove to be more efficient than the standard one if the desired outputs are mapped to [0.05, 0.95] instead of [0, 1]?

After mapping the desired outputs to [0.05, 0.95], the problem of having **desired** values equals 0 or 1 was solved, which was one of the problems they were facing. Moreover, the actual output **out** was mapped to the range [0.05, 0.95], which was another reason for the authors to support their need for having this new formula.

We did not stop at this theoretical level, but also implemented the alternative idea and compared the results. The comparison proved our suspicion that the mapping criterion of **desired** and **out** would better fit to solve the problem which they were trying to avoid.

And thus the formula becomes:

$$\Delta_{\mathbf{out}} = \eta \times \lambda \times \mathbf{out} \times (\mathbf{1} - \mathbf{out}) \times (\mathbf{desired} - \mathbf{out})$$

We also tried to use the formula:

$$\Delta_{\mathbf{out}} = (\mathbf{1} - \mathbf{out^2}) \times (\mathbf{desired} - \mathbf{out})$$

derived from substituting the term

$$\mathbf{out} \times (\mathbf{1} - \mathbf{out}) \text{ by } (\mathbf{1} - \mathbf{out^2})$$

At that time, our module was good enough, i.e. converging with a less number of iterations, so we preferred not to use it at the final stage even though the network

was also converging to a total error less than 0.005 per iteration.

Krzyzak *et al.* used the Fourier Descriptors to represent the outer contour of the pattern combined with extracting the number of "holes representing internal topological feature of the pattern" for its ability in dividing the class:
{ **0, 1, 2, 3, 4, 5, 6, 7, 8, 9** } directly into 3 subclasses:
{ **1, 2, 3, 5, 7** } , { **0, 4, 6, 9** } and { **8** } which stands respectively to a zero hole, one hole and 2 holes in the tested pattern.

3.2.3 The general architecture of the network

The input to the neural network was a (26x26) binary image of the pattern. The reason behind presenting the binary image of the pattern was that we wanted to build a network that will train on the image of the pattern rather on some features, because the network will have to accommodate itself to new types of symbols, which are completely unpredictable in size or shape. The network has only one hidden layer. The size of the hidden layer is dynamic (not fixed). However, the number of hidden neurons in the hidden layer is bounded between a maximum of 250 and a minimum of 5. As for the TRIGONIX project, as shown in Figure 2, the output layer consisted of a total of 43 output neurons, 26 neurons for the alphabets, 10 for the numerals, and 7 for a set of seven special symbols, { *(,), [,], /,* \, - } . In the CRIM project, we fixed the number of output neurons to 2, one for the symbol itself and the other for its False Positives.

3.3 Modifications introduced

Now that we have discussed the paper [34], we will emphasize on our modifications and improvements. Throughout this section, we will refer to the work done related to the TRIGONIX project. Figure 2 shows the network's architecture.

The output layer is made up of 43 units (neurons). The first 26 units are for the character set, then 10 units are reserved for the digit set, and the 7 units left are reserved for the 7 new symbols. On the classification or the testing phase, the unit

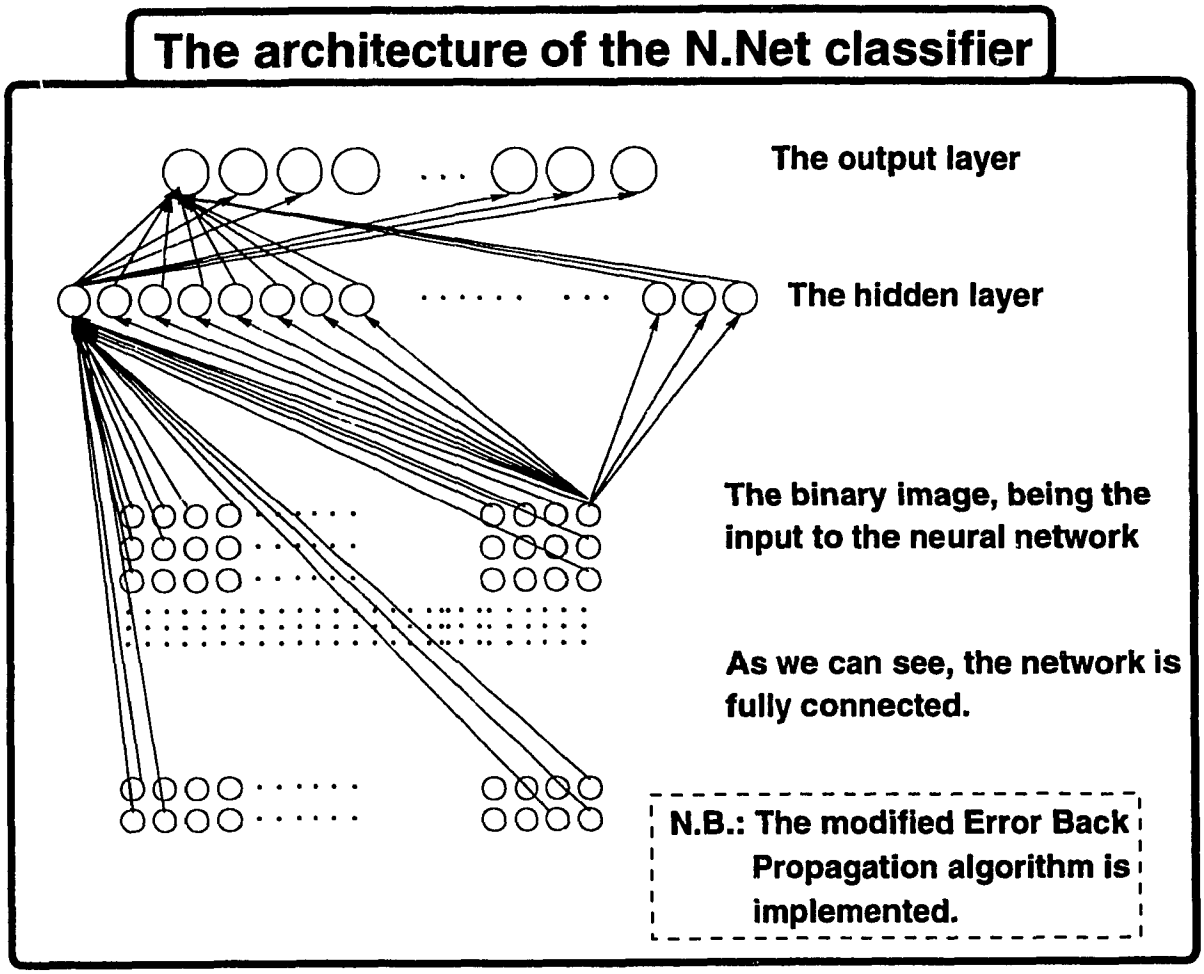


Figure 2: The architecture of the neural network.

with the higher output will win over the other units, thus leading to the classification of the currently tested pattern to its respective class.

In the same token, the input tc the network is only the (26x26) binary image of the pattern. No smoothing, removing noise, or any other kind of preprocessing to the image has been done. However, we have only normalized the size of the image into (26x26) in trying to decrease the number of connections between the input layer and the hidden layer, especially after we have decided to use the fully connected network. It is true that the locally connected network will lead to similar good results, and at the same time it will result in a smaller number of connections. The reason for choosing the full connection is to allow adding some statistical features to the input layer. It is known that a full connection is better for such a situation. Another reason for deciding on the fully connected network is to permit either increasing or decreasing the number of hidden neurons in the hidden layer. In the case of a locally connected network, we discovered that the problem of "forgetting some patterns that have been already learned in previous iterations" was a serious problem which we wanted to avoid, especially as we wanted to build a smart network that will not forget what it has learned in previous iterations.

Having decided on building such a smart network, the choice of training the network again on old weight matrices (that were the results of previous training batch processes) while keeping the same training set but adding to it some more samples was implemented. This experiment was done to see if the network would tend to forget some previously learned patterns. Fortunately, the problem of "forgetting what has been previously learned" was surpassed if we try to train the network to learn some other patterns.

The training samples of the patterns for this experiment were selected from text paragraphs extracted from different types of *engineering drawings* from **TRIGONIX Inc.**

3.3.1 Modifying constants and parameters

3.3.2 Modifying the formulas

Following are the steps of the algorithm implemented, including the modifications to the formulas.

First, the weight matrices and bias vectors of the hidden layer and the output layer are initialized into values in the range of [-0.75, 0.75]. Second, the pattern is presented to the network at the input layer. Third, the output of the hidden layer is calculated using this formula:

$$hid^i = f(\sum_{j=0}^{nin-1} (w_{11}{}^{ij} \times sample^j) - b_{11}{}^i) \qquad \forall i \in \{0, \ldots, nhid-1\} \tag{1}$$

where the activation function used is the sigmoid function:

$$f(x) = \frac{1}{1+e^{(-\lambda x)}} \tag{2}$$

Then, the output of the output layer is calculated using the formula:

$$out^i = f(\sum_{j=0}^{nhid-1} (w_{21}^{ij} \times hid^j) - b_{21}{}^i) \qquad \forall i \in \{0, \ldots, nout-1\} \tag{3}$$

and is then limited into the range [0.05, 0.95] using these conditions:

$$out^i = \begin{cases} 0.05 & \text{if } out^i < 0.05 \\ 0.95 & \text{if } out^i > 0.95 \\ out^i & otherwise \end{cases} \qquad \forall i \in \{0, \ldots, nout-1\} \tag{4}$$

Fourth, *perr* the error value of the pattern, currently being presented to the network, is calculated using the formula:

$$perr = \frac{1}{2}(desired^i - out^i)^2 \tag{5}$$

perr is the calculated error of the *sample* being presented with respect to its *desired* output. The total error per iteration is calculated and kept in *terr* using the formula:

$$terr = terr + perr \tag{6}$$

Consequently, the error signal of the output layer and of the hidden layer are computed. At the same time, the weights and biases of the output and hidden layers are updated respectively. The formula of the error signal of the output layer is the following:

$$\triangle_{out}{}^{i} = \eta \times \lambda \times out^{i} \times (1 - out^{i}) \times (desired^{i} - out^{i}) \qquad \forall i \in \{0, \ldots, nout - 1\} \tag{7}$$

The term $\eta \times \lambda$ was added to the formula of the standard error signal of the output layer, ([63], p. 189). The justification in keeping out^{i} depends on the fact that after mapping out^{i} into the region [0.05, 0.95] , the case of having $out^{i} = 0$ is eliminated. The constant term $\eta \times \lambda$ was added to the formula heuristically.
The weights of the output layer are adjusted in the following way;

$$w_{22}{}^{ij} = w_{21}{}^{ij} \times \eta \times \lambda \times \triangle_{out}{}^{i} \times hid^{j} + \alpha \times (w_{21}{}^{ij} - w_{20}{}^{ij}) \tag{8}$$

$$w_{20}{}^{ij} = w_{21}{}^{ij} \tag{9}$$

$$w_{21}{}^{ij} = w_{22}{}^{ij} \tag{10}$$

where w_{22}^{ij} belongs to the final results' weight matrix, and w_{20}^{ij} & w_{21}^{ij} belong to the two matrices that are used in calculating the **momentum term** . For more explanation on the notation, please refer to the subsection entitled **Notations used.**
Another modification made to the algorithm, was the updating of the values of the bias vectors, while training, using the formulas:

$$b_{22}{}^{i} = b_{21}{}^{i} - \Delta_{out}{}^{i} + \alpha \times (b_{21}{}^{i} \times b_{20}{}^{i}) \tag{11}$$

$$b_{20}{}^{i} = b_{21}{}^{i} \tag{12}$$

$$b_{21}{}^{i} = b_{22}{}^{i} \tag{13}$$

At the beginning, this modification (of the bias vectors) was implemented without any previous knowledge of its impact on the performance of the network, having only the intention that the bias plays an important factor in calculating the outputs of the hidden and output layers respectively, which was derived from the fact that the **Matlab** software package has the utility of generating random bias vectors and updating them in the training process. So, after generating these random biases, we modified them while training the network, but the novelty of this approach is that these biases are modified the same way the weight matrices are updated.
Consequently, the error is propagated back to the hidden layer using the error signal formula:

$$\delta_{hid}{}^{i} = \eta \times \lambda \times hid^{i} \times (1 - hid^{i}) \times \sum_{k=0}^{nout-1} (\Delta_{out}{}^{k} \times w_{22}{}^{ki}) \qquad \forall i \in \{0, \ldots, nhid - 1\} \tag{14}$$

The original formula was modified by replacing the term $1/2 \times (1 - hid^{i(2)})$ ([63], p. 189) by the term $\eta \times \lambda \times hid^{i} \times (1 - hid^{i})$. Then the weight matrices of the hidden layer are calculated using the following modified formula;

$$w_{12}{}^{ij} = w_{12}{}^{ij} \times \Delta_{hid}{}^{i} \times sample^{j} + \alpha \times (w_{11}{}^{ij} - w_{10}{}^{ij}) \tag{15}$$

The same formula is applied for updating the bias vectors of the hidden layer, with two minor changes in (11) where $\Delta_{out}{}^{i}$ is replaced by $\delta_{hid}{}^{i}$, and b^{i}_{22} is replaced by b^{i}_{12}, (same for b^{i}_{10} and b^{i}_{11}).
To sum up, all of the steps mentioned above are applied repetitively to each sample pattern in the training set, thus constituting **one iteration**. After each iteration, we

check whether the total error *terr* is less than or equal to a desired minimum total error per iteration (set by the user), in this case *0.005* . Hence, the training process will terminate when $terr < 0.005$.
Finally, the values of η, λ, and α were also modified during the training process.

3.3.3 Initialization of weights and biases

Though the pace of the research was conducted to accomplish the two (TRIGONIX and CRIM) projects and write my thesis, there emerged an experimental idea which can be summarized by saying: there is (there must be) a relation between the number of neurons in a layer L and the initialization of its respective weight and bias matrices. Throughout this thesis, the initial weights and biases fall in the range of [-0.75, 0.75].

3.3.4 The error function and the minimization process

The error function is calculated by taking the sum squared of the difference between the desired and the actual output. Consequently, the minimization process adopts the *Steepest Descent* algorithm (which is the first order method). Its advantages are that it's easy to be implemented on digital computers, and that it displays good results on well behaved functions in reaching the neighborhood of function's minimum very quickly. On the other hand, its disadvantage is that the progress of minimization is very slow near the minimum point. This is related to the path followed in the steepest descent minimization. This fact forces us to search for another alternative or another minimization algorithm to adopt. One of these methods is the *Newton's Method*, which is a second degree method. One of its major advantages is that if the current position (point) on the curve is close to the minimum, then it converges very quickly.

One could directly conclude that a combination of both methods will prove to be very practical. In other words, starting with the *Steepest Descent* method when we are away from the minimum, and switching to *Newton's Method* when we reach the neighborhood of the minimum enable us to reach the minimum quickly.

3.4 Dynamic versus predefined architectures

3.4.1 Locally or fully connected?

This is totally related to the type of input to the network. In other words, if the input to the network is a binary or grey-scale image only, then locally connected is preferable. On the other hand, if the input has some statistical features, then fully connected is better. However, if a combination of both inputs is to be used, then it is hard to choose which one is better, except after experimenting on both. Here we have to keep in mind that the convergence of the network is faster in the case of a locally connected because the number of connections will be obviously less than in the case of a fully connected.

3.4.2 The input and output layers

The input to the network is a normalized two dimensional, 2D, array of the pattern's binary image. Many experiments have been conducted on different sizes of the 2D array. When normalizing the patterns into (16x16) 2D array, the network used to converge faster because it had less connections. But, when the dimension of the array was set to (24x24), the network used to take more time per iteration, and thus longer to converge. The reason is simply because the latter (24x24) dimension employs more computations to an additional 320 input neurons, especially that all the networks are fully connected.

Two approaches were implemented concerning the output layer. The first one was implemented in the **TRIGONIX** project. The output layer consisted of N number of neurons corresponding to an N number of patterns. As for the **CRIM** project, since the objective was to build a dynamic NNC that is capable to add new or eliminate old patterns, one NNC per pattern was built and the output layer consisted of two neurons only, corresponding to the pattern itself and the other corresponding to its False Positive[1].

[1] For a definition of the False Positive, please refer to chapter 5, section 5.3.1.

3.5 The hidden layer

While creating the architecture of any artificial neural network that has one or more hidden layers, the designer is faced with the problem of deciding on the number of neurons to be set for each hidden layer. After lots of heuristic trials on different numbers of neurons, the decision will depend on the performance of the network. Throughout this section, a method of finding the neighborhood of the optimal number of hidden neurons for an error backpropagation neural network with a single hidden layer is proposed. Simulation results are done on recognizing unconstrained handwritten numerals.

3.5.1 Introduction to the problem

During the past 2 decades, there has been a large number of researchers who have dedicated their work to the building and discovering of new notions and axioms in neurology and in artificial neural networks. Many new networks have been introduced (e.g. SOM, BAM, ART, CPN,...), especially those which can contain hidden layers. Moreover, some effort has been devoted to the analysis of the hidden layers, see e.g. [14], [16], and [29].

On page 31 of his book **Digital Neural Networks**, Dr. S.Y. Kung [35] depicts the fact that: *"The number of hidden-units is directly related to the capabilities of the network. For the best network performance(e.g. generalization), an optimal number of hidden-units must be properly determined."* However, the author neither described how to find that *"optimal number of hidden-units"* nor provided some hints on how to proceed. And since the book was published in 1993, we took it for granted that if there was anybody who wrote about this critical point, then the author with his huge and selective references must have at least cited or referenced it when he talked about this problem. Some of the constrained guidelines will be presented in order to find the number of hidden units in the hidden layer. Some other papers addressed this issue but, to our knowledge, none of them tackled this subject the way we intend to. Also, during my attendance of a tutorial entitled **Neural Networks for Industry** given by professor Geoffrey Hinton, [24], who has made a considerable contribution

to the field, I had the chance to describe the method to him. He was impressed with my approach and asked me for results and whether the network will converge or not. Professor Hinton agreed cautiously on my theoretical blind prunning of hidden neurons, because he reserved his final opinion about it to see its experimental results. The rest of this chapter describes the intuition behind its theory, and the simulation results. Explicitly, the problem that we are trying to solve is directly related to the architecture of the network. Thus, in trying to optimize the number of neurons in a hidden layer, we are actually optimizing the size of the network - its architecture. Once the architecture of the network has been optimized, convergence will be achieved more quickly while training because the size of the hidden layer will be optimized and consequently, the performance of the network will be improved.

Up to this stage, the burden is left with the designer to decide on how many neurons to choose for a hidden layer, which is done most of the time heuristically. Yet, whether the number of neurons chosen was optimal, one can hardly judge! Even though the performance of the network might be good and acceptable, we have to seek an optimal architecture of the network. It can be proved that the performance of the network with an optimal architecture will always be better, and in the worst case the same, but never worse. So why not optimize the size of the hidden layers, i.e. the architecture of the network?

Consequently, let us go back to the roots of Neural Networks, to Neurology, and study [51] how our brain functions when learning to solve a problem. Then, we will try to mimic the formulas (or relations) we have deduced, implement those relations and check the results achieved. First, let's raise the following question: "Do we think, focus, concentrate the same way when we read a daily magazine, or when we read a letter from our manager stating that we are fired?" Certainly not, because the latter will force us to reserve and use more memory of our mind in trying to think why we were fired, the reasons behind that, and its financial consequences on us, for example. This will cause us to use more "neurons" in order to get some quick answers to the latter questions. In contrast, if you are reading or just flipping the pages of a magazine, then you don't need to exert much pressure or use lots of "neurons" because such tasks are easily executed. In the same token, there are plenty of examples that

support our claim, like preparing for a final exam versus watching a movie, etc.

3.5.2 The neighborhood of the optimal number of neurons in a hidden layer

The state of the art of our method is the following: *there is a relation between the number of hidden neurons and the process of learning*, and that forms the basis of our research. In other words, if the error function is decreasing towards the minimum over a certain period of iterations (which was set to 4 during simulation), then this means that the current number of hidden neurons is good. But, let us try to optimize it by decreasing the number of the hidden neurons. How can we decrease its value? There are two methods, one prunes the hidden neurons that were not learning during that period of iterations, [25] and [29], and the other chops off a constant number of neurons from the hidden layer. As a matter of fact, the first method proves to be more practical since we will not lose some neurons that have previously learned well. To do this, we need to keep track of all the neurons at hand, and for simplicity, we decided to choose the second alternative. However, even if we are losing some knowledge in blind pruning, this knowledge will be gained throughout the next iterations. Furthermore, as far as the first method is concerned, the connections will be changed so that calculating the *perr* and *terr* will result in the same values of the currently finished iteration, but not for the following iteration.

In this approach, after decreasing the number of hidden neurons, we start again the study of the curvature of the error function over the same period of iterations. In case the error function starts to diverge away from the minimum of the function, then we increase the number of hidden neurons by a certain number of neurons (not necessarily equal to the number of neurons deleted in the case of a converging function, as explained above). This process takes place while the network learns the patterns from the training set. However, there must be a lower bound and an upper bound for the number of hidden neurons, set by the user. During our simulation, we set the upper bound as 120 and the lower bound as 5. Therefore, during training, we are not only interested in letting the error function reach its minimum, but also be able to

optimize the number of hidden neurons and end up with an optimal architecture of the network which is capable of displaying good results at the generalization level.

3.5.3 Simulation results

The performance of the network was examined by recognizing unconstrained handwritten digits from the CEDAR database, which contains digits collected from the U.S. Postal Services' dead letters. Here, we see that it is worthless to compare the final recognition rate with our previous experiments on the network because the number of hidden units in the hidden layer used to be chosen heuristically, whereas in these simulation results the number of hidden neurons is automatically decided by the method. The reason for having a neighborhood of the optimal solution, rather than the optimal solution is also stated.

Accordingly, while studying the curvature of the error function to decide on whether to add (or remove) more neurons to (or from) the hidden layer, the number of iterations was set to 4 in case of convergence, which means that we have to reduce the number of neurons in the hidden layer by 10. On the other hand, in case of divergence, the number of iterations was set to 3 iterations, implying that we have to increment the number of the hidden units by 5. Note that the number of neurons to be added (5) or removed (10) was fixed in our implemention, but it need not be fixed this way.

Figure 3 shows the graph of finding the neighborhood of the optimal number of hidden neurons. As one may notice, the initial number of hidden neurons was 80, then it reached the value of 20 twice until it ended up oscillating between 30 and 40. (Note that only in Figure 3 adding or removing neurons was set to 10.) Consequently, after many experiments with our method such as changing the initial value of the lower and upper bounds, changing the number of iterations for either or both of the converging and diverging situations, and training the network on different sets of data, we finally agreed that finding the optimal number of hidden neurons cannot be achieved at the generic level, because the process of finding it is related to the special context in which the network (having the hidden layer) was trained. One of the most important factors is related to the quality and quantity of the training sets. In other words, training

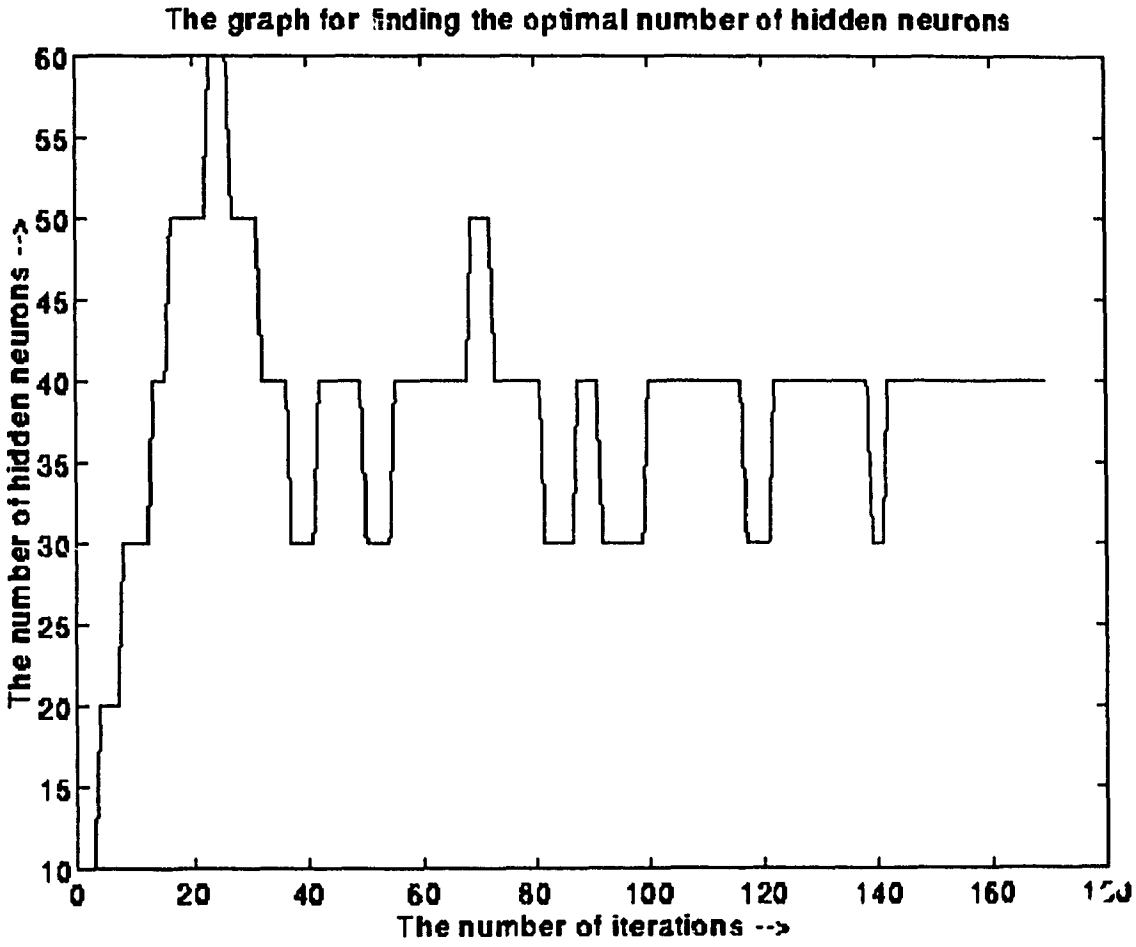


Figure 3: The graph for finding the neighborhood of the optimal number of hidden neurons.

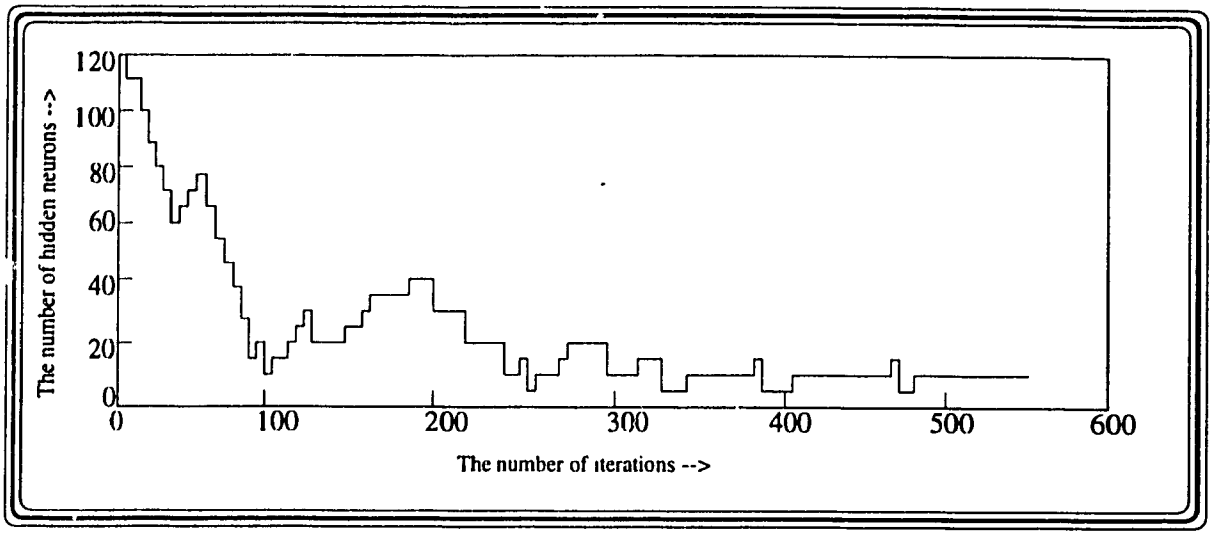


Figure 4: This graph shows that the network converged to 10 hidden neurons.

the same network on two different training sets may (sufficiently and not necessarily) lead to two different optimal solutions. Here, it is noteworthy to declare that the optimal solution for the larger training set is to be considered. In our simulation, the training set was made up of 2000 digits consisting of 200 patterns for each digit, mainly the *bad* set added to it some patterns from the *good* set. For instance, the *bad* set consisted of 193 samples of the numeral 5, so we added 7 more samples from the *good* set. As for the other numerals, the first 200 samples were considered.

Moreover, Figure 4 represents the graph of the number of hidden units with respect to the number of iterations. These results were achieved after we removed 97 patterns from the previous training set, which even humans could not recognize correctly. When the error function was decreasing or increasing slowly, we kept track of the amount of change in the value of the error function (starting from the first iteration until the current iteration) and depending on it, the number of hidden neurons was incremented or decremented. Thereafter, we continue the training process.

Even though the optimal solution that we got in this case is *10 hidden units* only in the single hidden layer, yet the existing oscillation in the graph raises the following question: Is it true that the number of hidden units should be at least equal to the number of clusters (in our case 10 clusters), and are we able to reduce it further?

	0	1	2	3	4	5	6	7	8	9	Rej	Total	Error
0	291	1	0	0	0	0	4	0	0	0	4	300	5
1	0	285	2	1	0	0	3	0	3	0	6	300	9
2	2	0	281	2	0	0	0	2	0	0	13	300	6
3	2	1	2	280	0	2	0	1	2	0	10	300	10
4	0	0	1	0	281	2	0	1	0	4	11	300	8
5	2	1	2	0	0	278	1	0	1	2	13	300	9
6	2	0	2	2	1	0	286	0	1	0	6	300	8
7	0	0	0	1	0	0	0	284	0	6	9	300	7
8	0	0	2	4	0	2	0	0	275	2	15	300	10
9	0	0	0	3	2	0	0	3	0	281	11	300	8
										Total:	98	3000	80

Table 1: The Confusion matrix of the testing set consisting of 3000 patterns from the CEDAR database.

Presently, although some may directly disagree with the idea, we leave the answer to this question for future research.

Now that the neural network classifier, NNC, has been trained, we will test its performance on a testing set made up of a new set of 3000 patterns, from the CEDAR database. For each pattern, i.e. numeral, the first 300 samples are chosen from the *bad* set in case it contained more than 300 samples, otherwise more samples are taken from the *good* set in order to have 300 samples per pattern.

The threshold for rejection was set at 45%. Table 1 shows the confusion matrix and the results achieved in terms of the error, rejection, recognition, and reliability rates.

80 patterns were recognized incorrectly resulting in an error rate of 2.66%, 98 rejected patterns resulting in a rejection rate of 3.32%, so the recognition rate was 94.02% and the reliability was equal to 97.24%.

Such results are considered to be good and can further be improved if some statistical features are employed. For instance, Table 1 shows that 6 instances of the numeral **7** were missrecognized as a **9**, and that 3 instances of **9** were missrecognized as **7** resulting in 9 missclassification errors between the 2 classes. Such problems could be avoided when the number of holes, breakpoints and endpoints, calculated for each instance, are used for training and testing.

Classifier	Recognition
NNC	97.24
Binary Image	96.61
Binary Polynomial	96.72
Histogram	97.38
Morphology	97.75
Chaincode	98.05
GSC	98.23

Table 2: The recognition results of 6 classifiers compared to NNC.

3.5.4 Comparing the results

Compared to the performance of other researchers' classifiers, [41], NNC performs fairly well as can be seen from Table 2. The *bad* set of the **CEDAR** database was used since it is the worst testing set. The classifiers chosen are based on many approaches. The features extracted include horizontal and vertical projection profiles, the number of pixels of the pattern, contour information, number of end points, break points, circles, straight and slant lines, concave and convex curves, etc. The classification methods range from neural networks, binary polynomial, histogram-based, dynamic programming, and tree classfiers. The NNC has no features extracted, which is similar to the Binary Image classifier that had a recogntion rate of 96.61%, while the NNC achieved a better performance of 97.24%. Lee and Srihari, in [41], claim that it is hard on most classifiaction algorithms to generalize when no extracted *features* are employed. To probe further, please refer to [19], [37], [41], [44], [54], and [55]. In a word, although the NNC's performance is not the highest among all of the cited work of other researchers, yet it is the highest between those classifiers that have the same input as the NNC, a binary image of the symbol only.

Chapter 4

Segmentation Algorithms Used

Segmentation is the art of separating a certain pattern from its background and neighborhood. The problem of segmentation has always been a hard and challenging topic to tackle. Nick W. Strathy of CENPARMI has conducted research in trying to solve this problem and has contributed to the field in many papers published in highly technical international conferences and journals, in addition to his Master's thesis, [53].

Considering his suggestions, I chose to use the vertical and horizontal histograms, and the connected components segmentation algorithms for the TRIGONIX project. This project deals with the segmentation of paragraphs and the recognition of alphanumeric characters extracted from engineering drawings.

On the other hand, a modified version of the Hausdorff's distance method [49], developed by Eric Reiher of CRIM, was used to extract symbols from the maps to be presented to the neural networks classifiers, NNC. To probe further on the Hausdorff's distance method, please refer to [28].

4.1 Segmenting paragraphs of text from engineering drawings

Figures 5, 6, and 7 are samples of the paragraphs that are passed to the NNC to be recognized. These paragraphs, as we can see, contain different font types and sizes which add to the complexity of the problem.

4.1.1 Line Segmentation

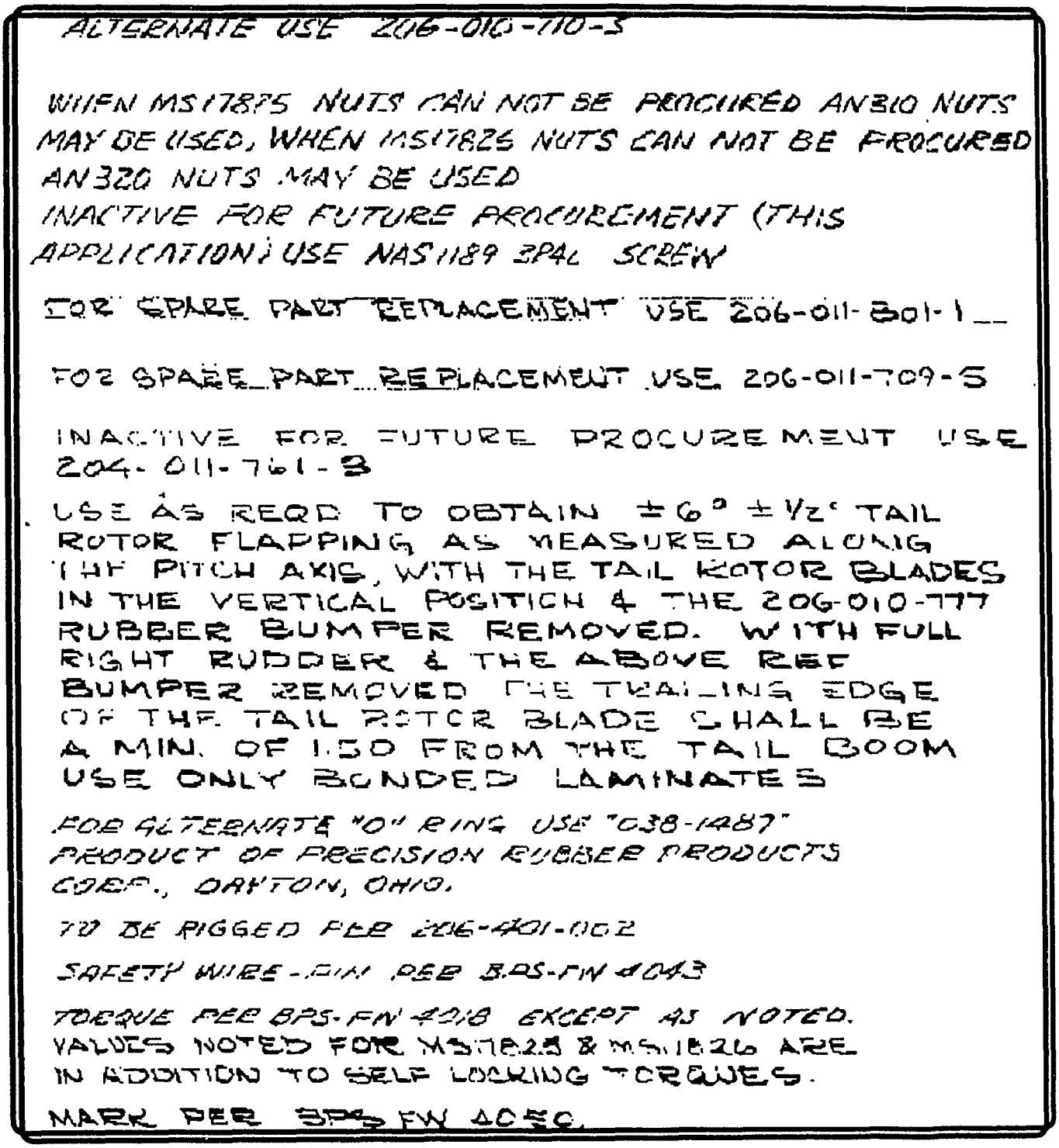
Since we will be receiving paragraphs containing more than one line, we had to break the paragraph into separate lines. The method used is simply the *Basic Horizontal Histogram* of the whole input paragraph. Figure 8 shows a sample of the input paragraphs, and figure 9 shows the result of segmenting the paragraph into lines. **Removing noise** that might be present between lines (like dots, dashes, and straight lines resulting from improper scanning or from underlining some words of the currently segmented line) is done efficiently.
Figure 10 shows the result of segmenting a paragraph that has an underlined word.

4.1.2 Pattern Segmentation and Normalization

Now after segmenting the paragraph into separate lines of text, we start segmenting each line of text into separate patterns. Then, we normalize it into a 26x26 array to be presented later as input to the neural network classifier.

Pattern segmentation

No matter how efficient and good the performance of a stand alone classifier is, the absence of a precise pattern segmentation algorithm causes the recognition rate of that classifier to drop down. As for segmenting each line into *single* separate patterns, we tried two methods, the *Basic Vertical Histogram* method, and the *Connected Components* method. Figure 11 shows the results of both methods when experimented on words containing patterns that lie below or above other patterns. We can see that


ALTERNATE USE 206-010-710-3

WHEN MS17825 NUTS CAN NOT BE PROCURED AN310 NUTS MAY BE USED, WHEN MS17826 NUTS CAN NOT BE PROCURED AN320 NUTS MAY BE USED
INACTIVE FOR FUTURE PROCUREMENT (THIS APPLICATION) USE NAS1189 3P4L SCREW

FOR SPARE PART REPLACEMENT USE 206-011-801-1

FOR SPARE PART REPLACEMENT USE 206-011-709-5

INACTIVE FOR FUTURE PROCUREMENT USE 204-011-761-3

USE AS REQD TO OBTAIN ±6° ±½° TAIL ROTOR FLAPPING AS MEASURED ALONG THE PITCH AXIS, WITH THE TAIL ROTOR BLADES IN THE VERTICAL POSITION & THE 206-010-777 RUBBER BUMPER REMOVED. WITH FULL RIGHT RUDDER & THE ABOVE REF BUMPER REMOVED THE TRAILING EDGE OF THE TAIL ROTOR BLADE SHALL BE A MIN. OF 1.50 FROM THE TAIL BOOM USE ONLY BONDED LAMINATES

FOR ALTERNATE "O" RING USE "038-1487" PRODUCT OF PRECISION RUBBER PRODUCTS CORP., DAYTON, OHIO.

TO BE RIGGED PER 206-401-002

SAFETY WIRE - CIM PER BPS-FW 4043

TORQUE PER BPS-FW 4218 EXCEPT AS NOTED. VALUES NOTED FOR MS17825 & MS17826 ARE IN ADDITION TO SELF LOCKING TORQUES.

MARK PER BPS FW 4050.

Figure 5: An original input paragraph.

Figure 6: An original input paragraph.

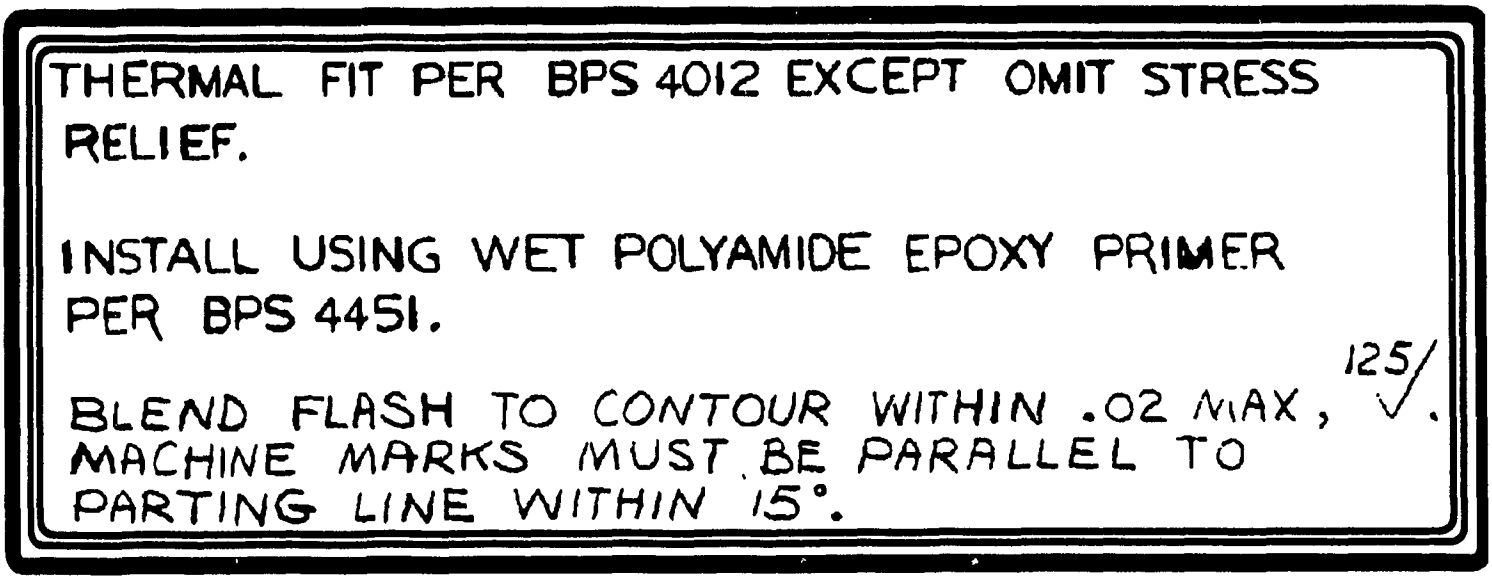


Figure 7: An original input paragraph.

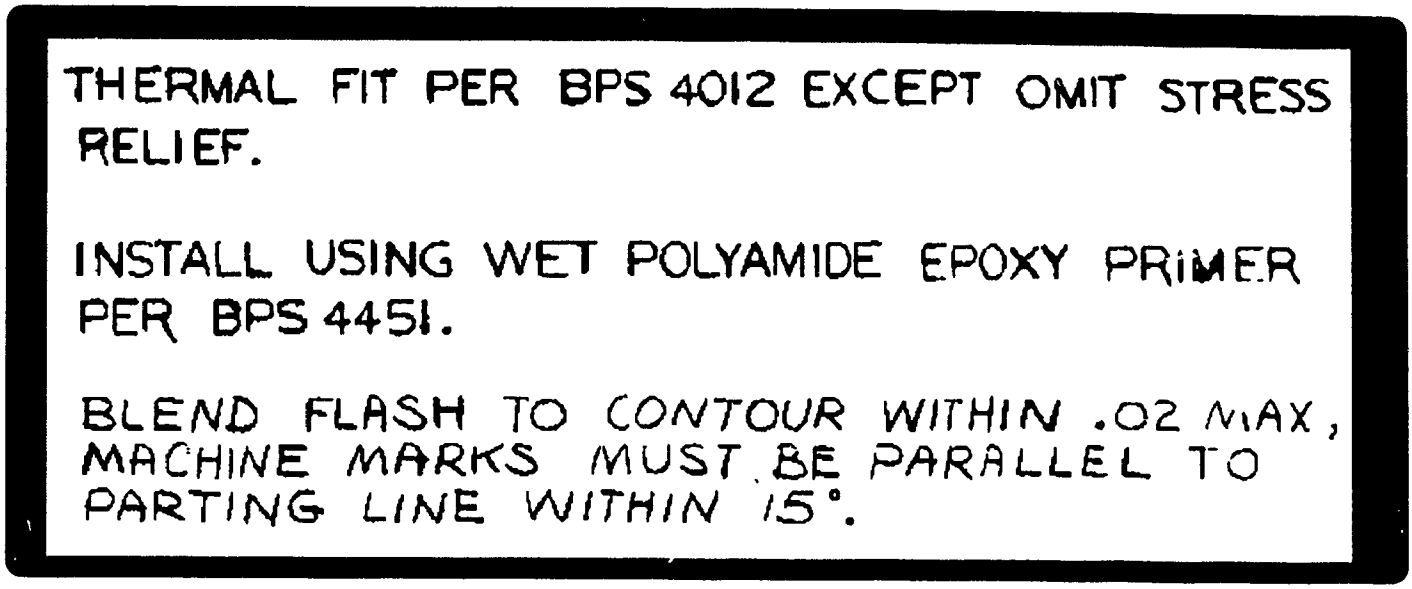


Figure 8: The original input paragraph to be segmented into separate lines.

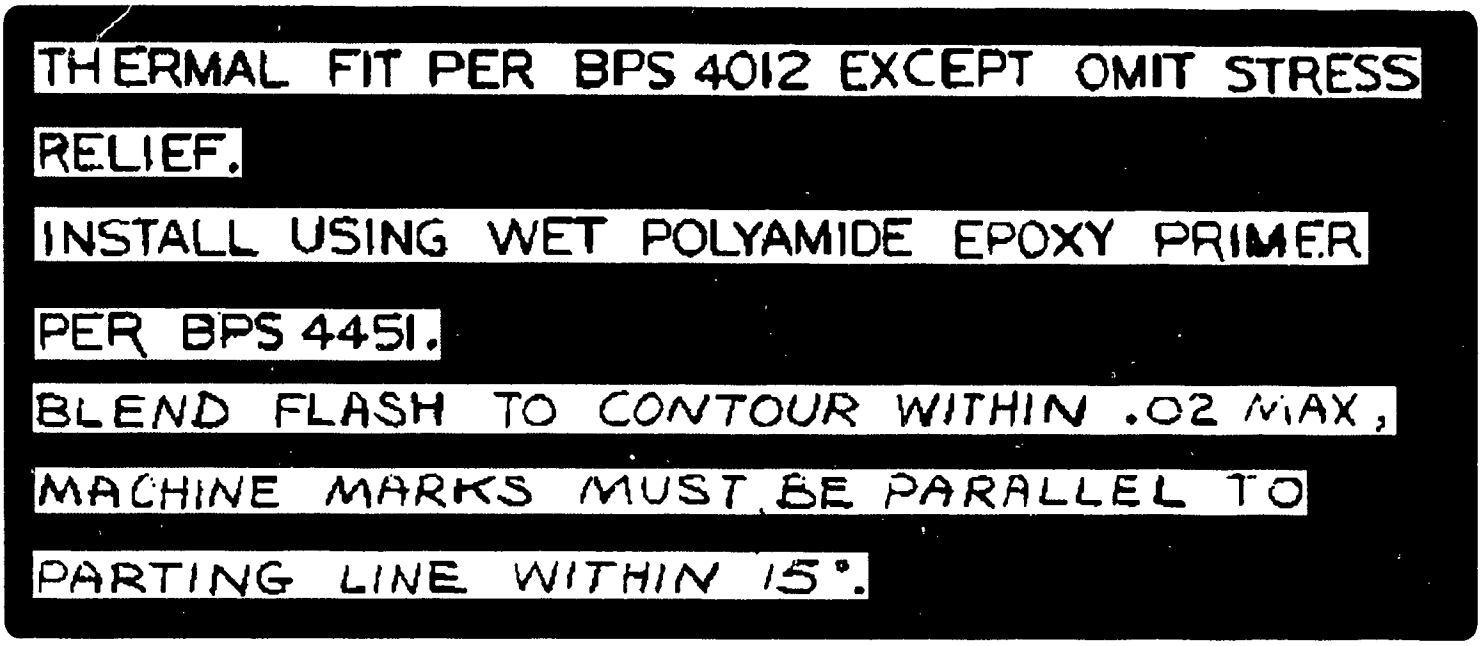


Figure 9: The results of segmenting the paragraph into lines.

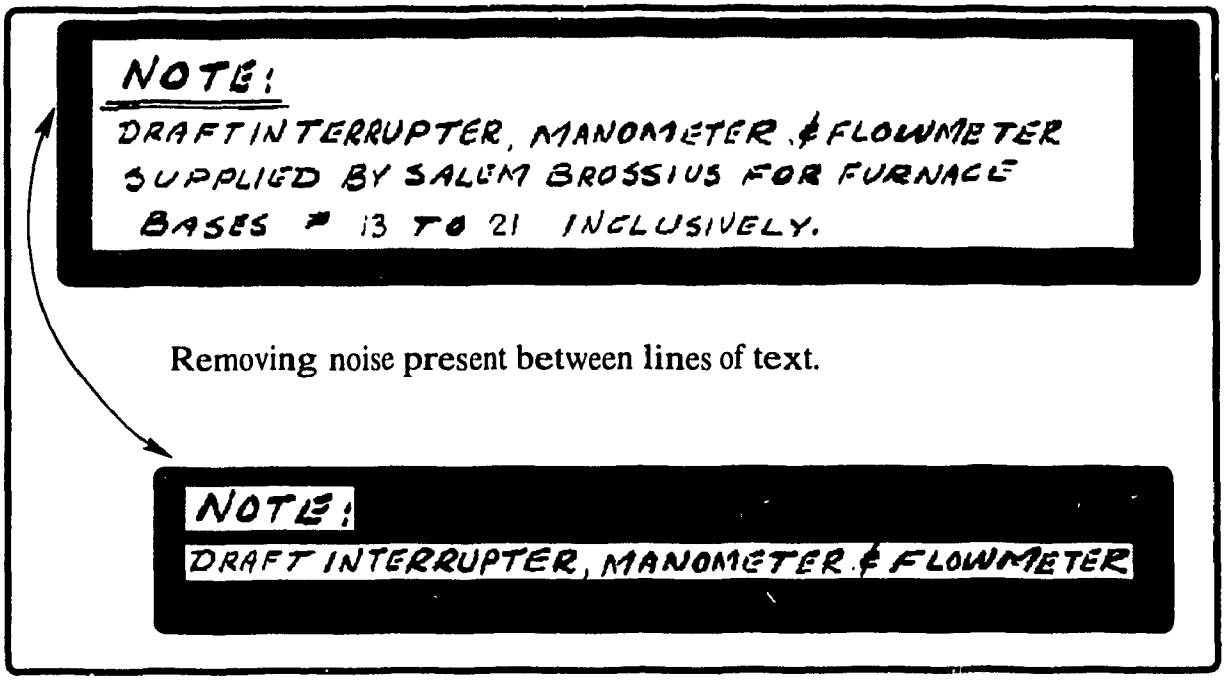


Figure 10: Removing noise while segmenting the paragraph into lines of text.

the *Basic Vertical Histogram* method resulted in a segmentation error while trying to segment the three consecutive patterns *E*, *S*, and *S* into one pattern - which is wrong. On the other hand, the *Connected Components* method was able to correctly segment them into three separate patterns.

Even though the *Connected Components* method is better than the *Basic Vertical Histogram* method, yet it has a defect. If the patterns had holes (i.e. disconnected) as in figure 12, then it will produce wrong results.

Pattern normalization

Since the input to the neural network classifier is a (26x26) binary image, we had to normalize the segmented patterns into that size. The method used for normalization is quite simple; it maps the black pixels in the segmented pattern into their corresponding pixels in the (26x26) binary image. It assumes that the (26x26) binary image is initialized to be blank before each normalization process. The *normalization function* maps the coordinates (a_1, b_1) of each black pixel in the original 2D-array to

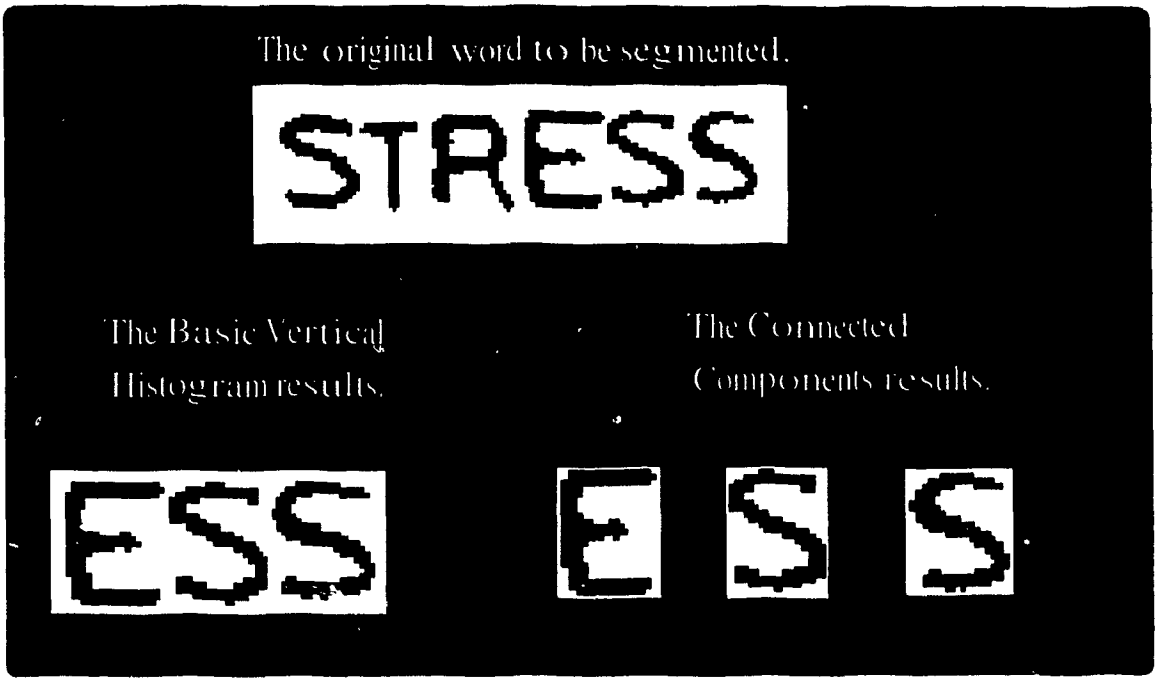


Figure 11: Comparison between the *Basic Vertical histogram* and the *Connected Components* methods; the first three characters *S*, *T*, and *R* were correctly segmented by both methods.

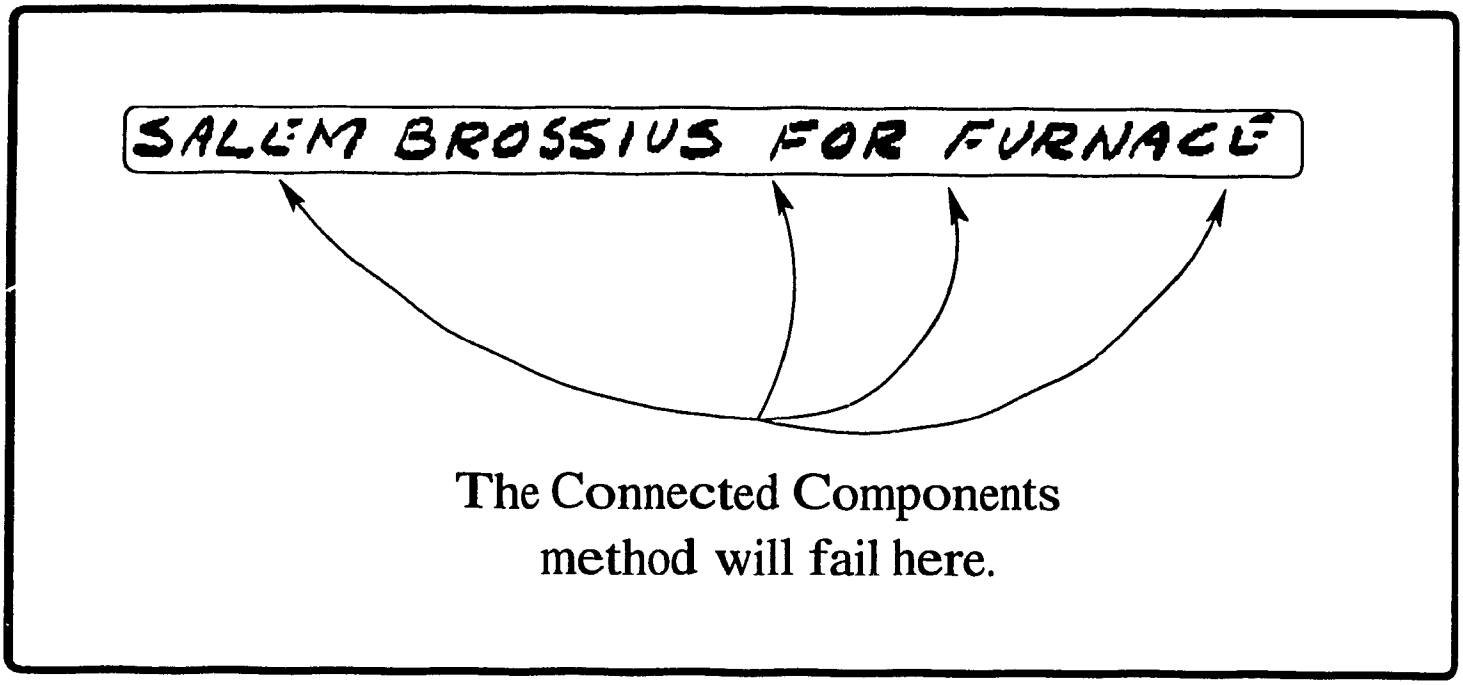


Figure 12: In case we had a pattern that has unconnected parts (components), then a wrong segmentation may occur, which will result to a misclassification in the testing phase.

its corresponding coordinates (a_2, b_2) in the desired (26x26) 2D-array. The equations of a_2, and b_2 are the following;

$$a_2 = (des_{row} \times a_1 / org_{row})$$

$$b_2 = (des_{col} \times b_1 / org_{col})$$

4.2 Segmenting map symbols

Figures 13 and 16 show samples of the maps that need to be processed. Figures 14 and 17 display the results after applying the Hausdorff's distance method only. Notice the large number of False Positives present in these two figures. The reason for having so many False Positives is because of the the Hausdorff's algorithm parameters are set in order not to miss any symbol from the map. Yet, some symbols are missed occasionally.

However, to solve the problem of eliminating the False Positives we combined the Hausdorff's algorithm with a set of NNCs to decrease the number of False Positives and to reconfirm the classification of the symbols into their respective classes, [49]. Figures 15 and 18 show the results of combining the Hausdorff's distance method with the DSHNNC.

Briefly, the Hausdorff distance measures the degree of mismatch between two sets of points by measuring the distance of a point of one set that is farthest away from any point of the other, and vice versa. This distance can be used to determine the degree of mismatch between two objects that are superimposed on one another. The Hausdorff distance is not sensitive to symbols touching or overlapping each other. It can thus naturally be used to first locate possible candidates. The symbols can be located in different scales and orientations. The scaling range starts from 30% less than the original model of the symbol up to 30% larger than the original model of the symbol with a 5% scaling step. For each scale value, the model is rotated 22.5°, starting from 0° to 337.5°, searching for possible matches after which they are horizontally rotated before they are passed to the DSHNNC. Then, each symbol is normalized into a square binary image that should not be less than (16x16) because the image of the symbol would lose part of its information, especially that they are

originally ranging from (256x200) to (50x50) but not limited to this range.

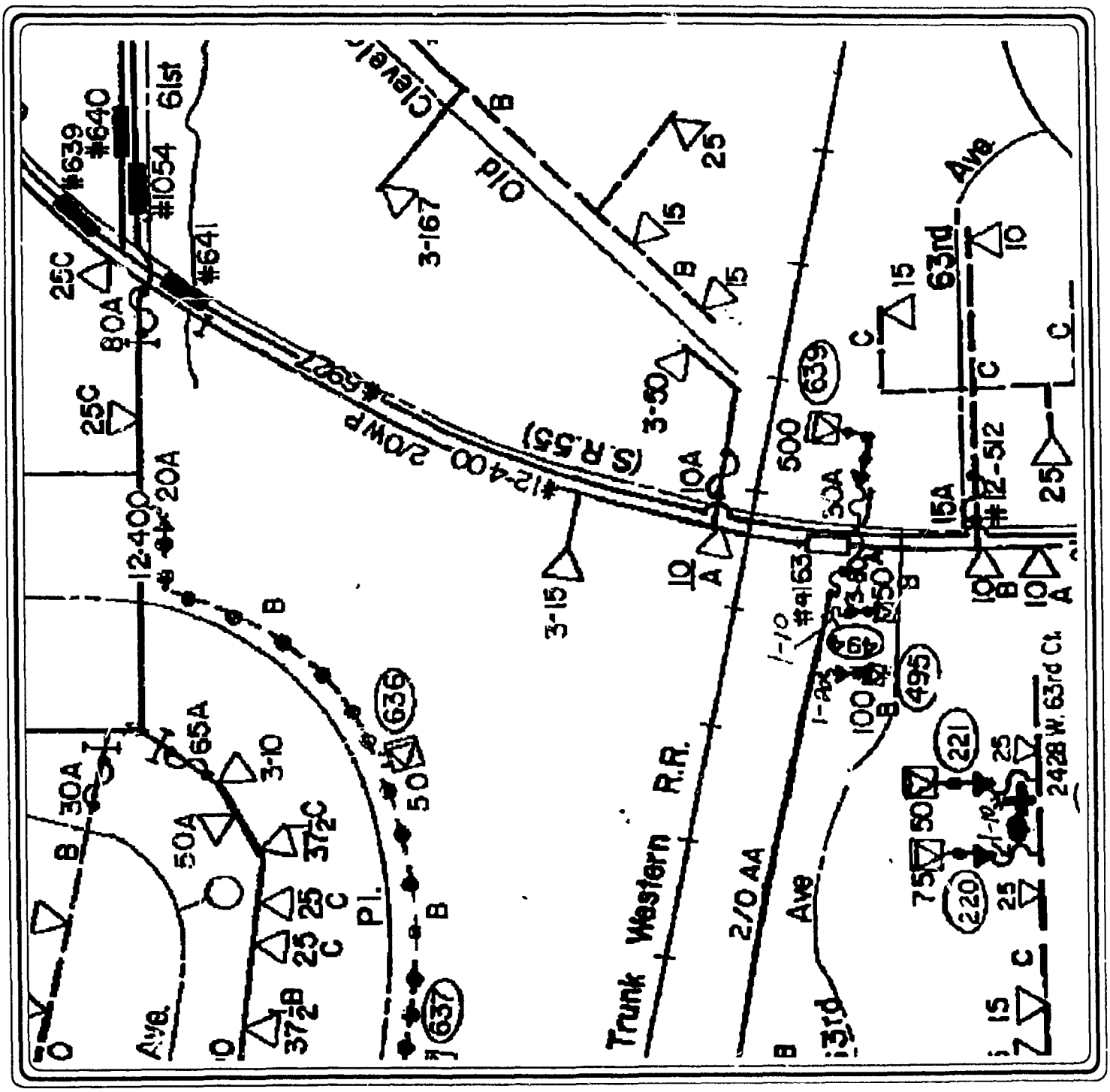


Figure 13: An original region of the map to be processed, Ellipses are the symbols of interest.

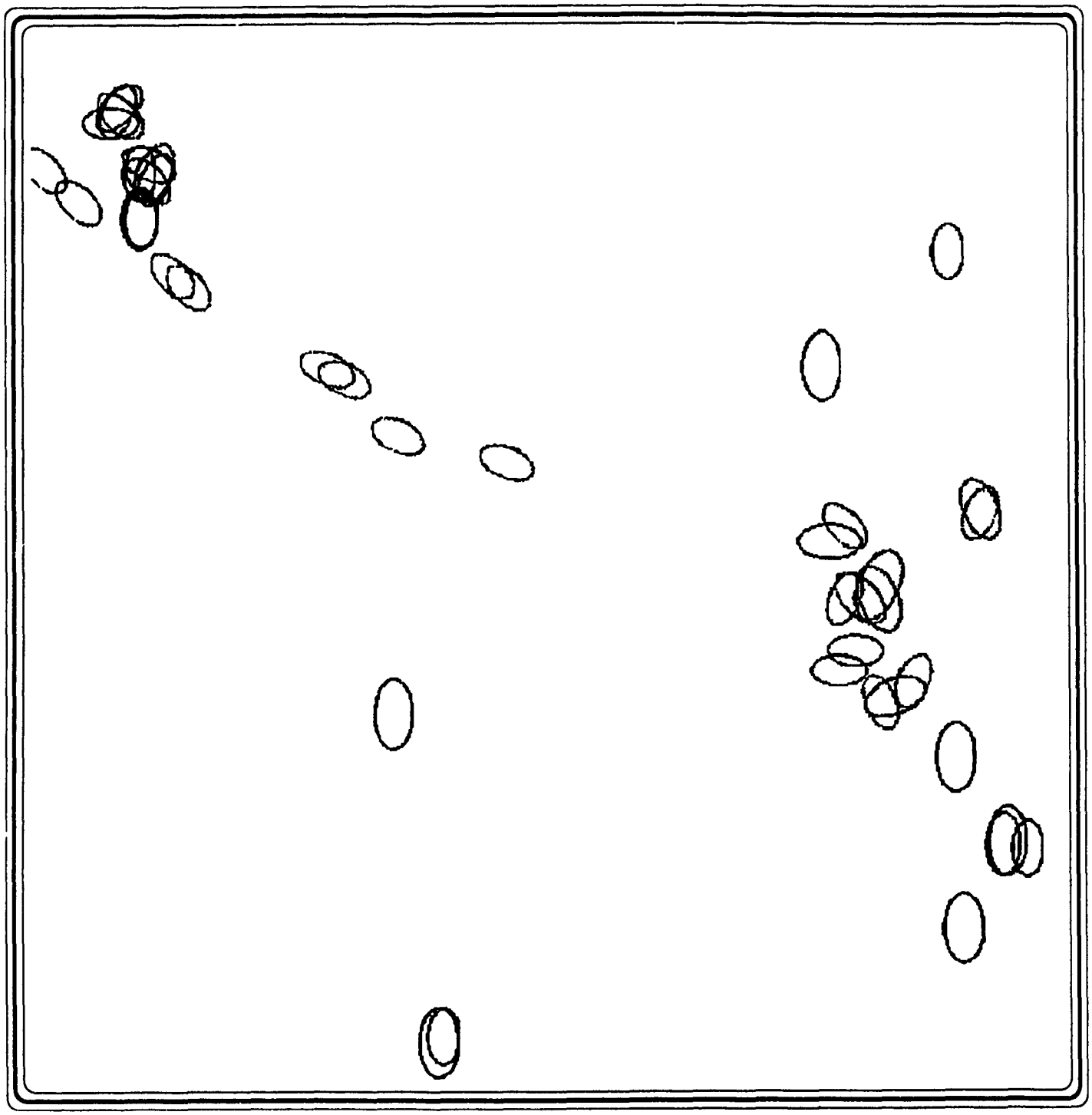


Figure 14: The Ellipse symbol's matches, using Hausdorff's distance method alone, found in the region of the map.

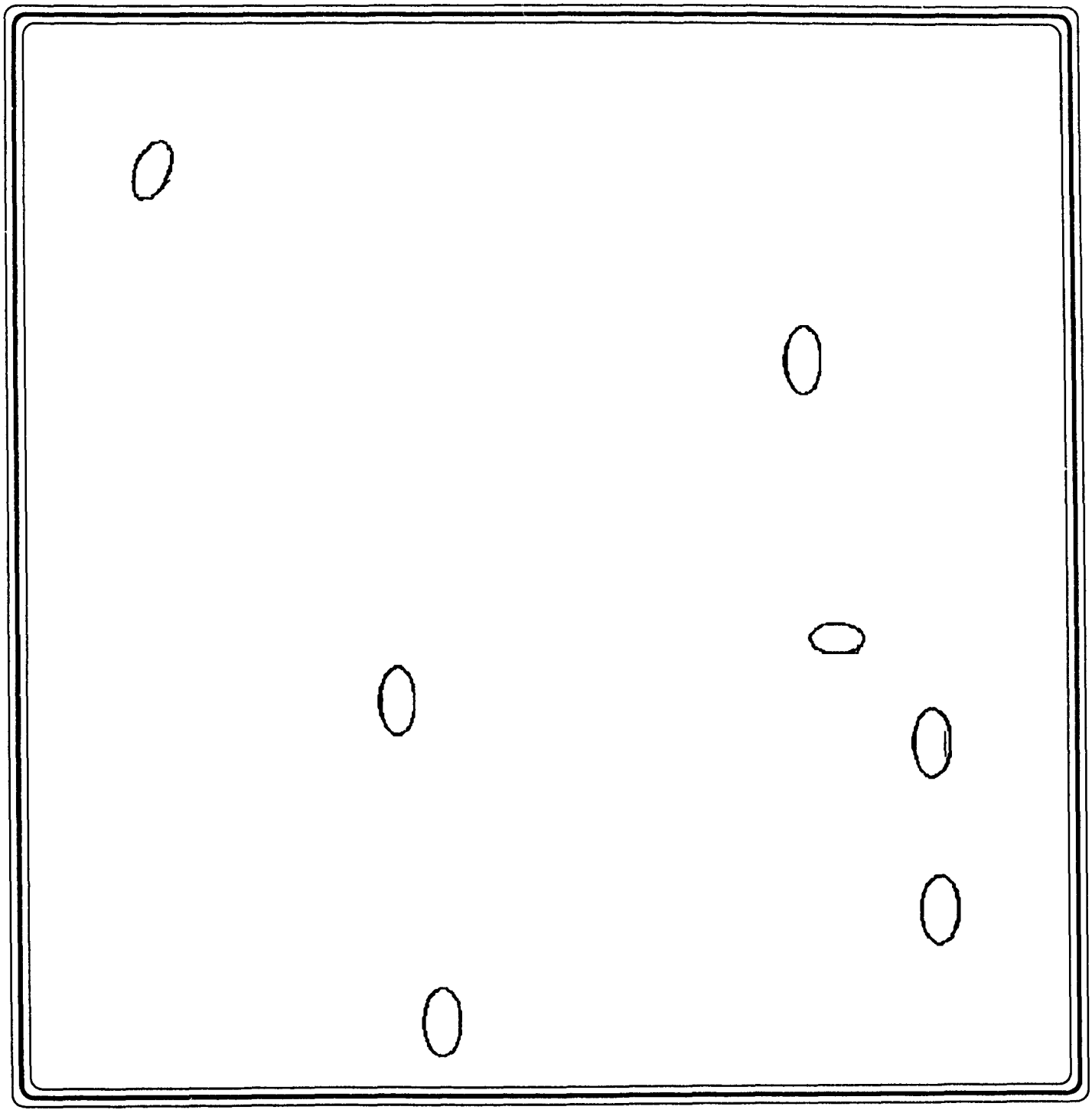


Figure 15: The results after passing all the Ellipse matches to a set of NNCs to filter out the False Positives.

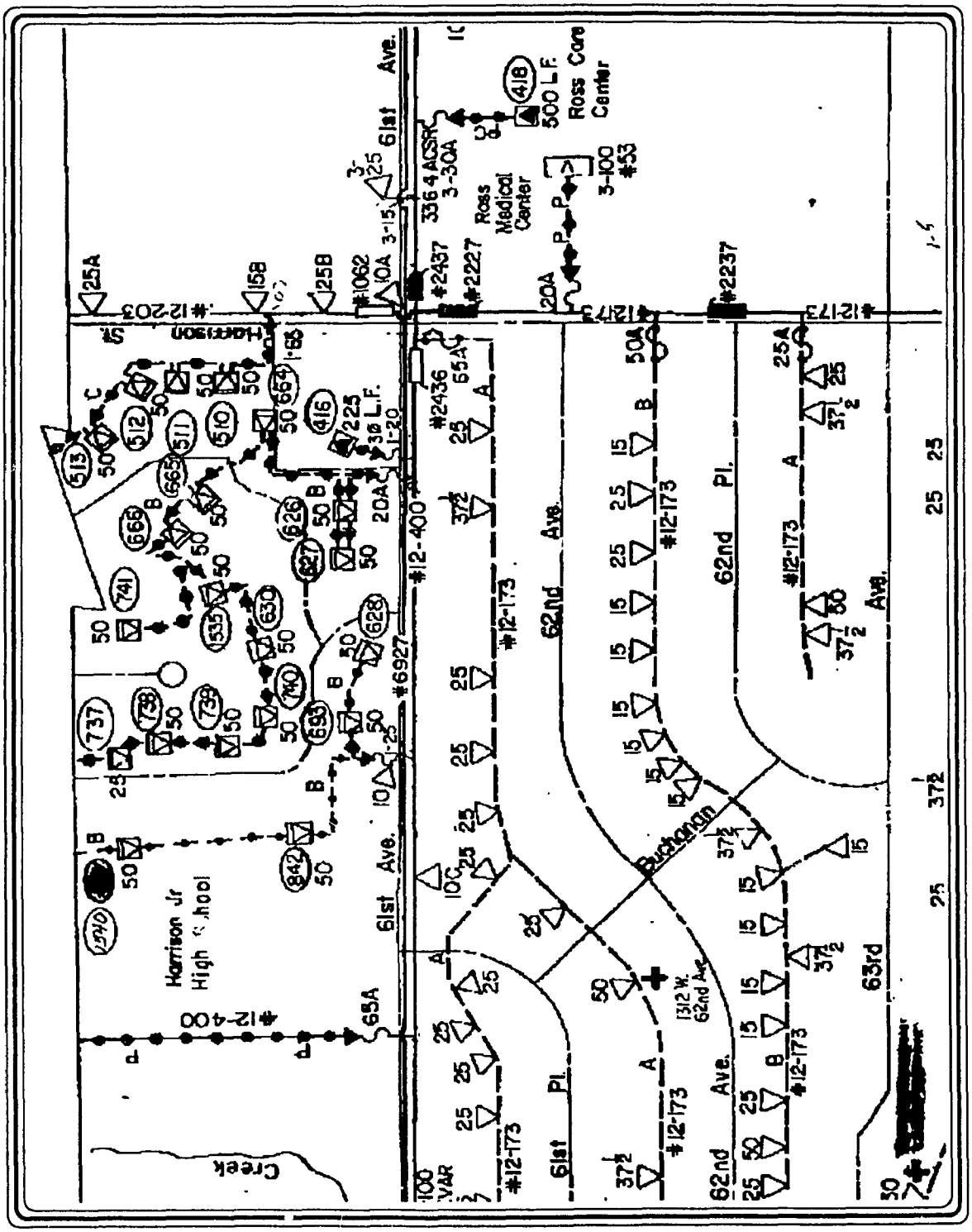


Figure 16: Another original region of the map to be processed, Filled Triangles are the symbols of interest.

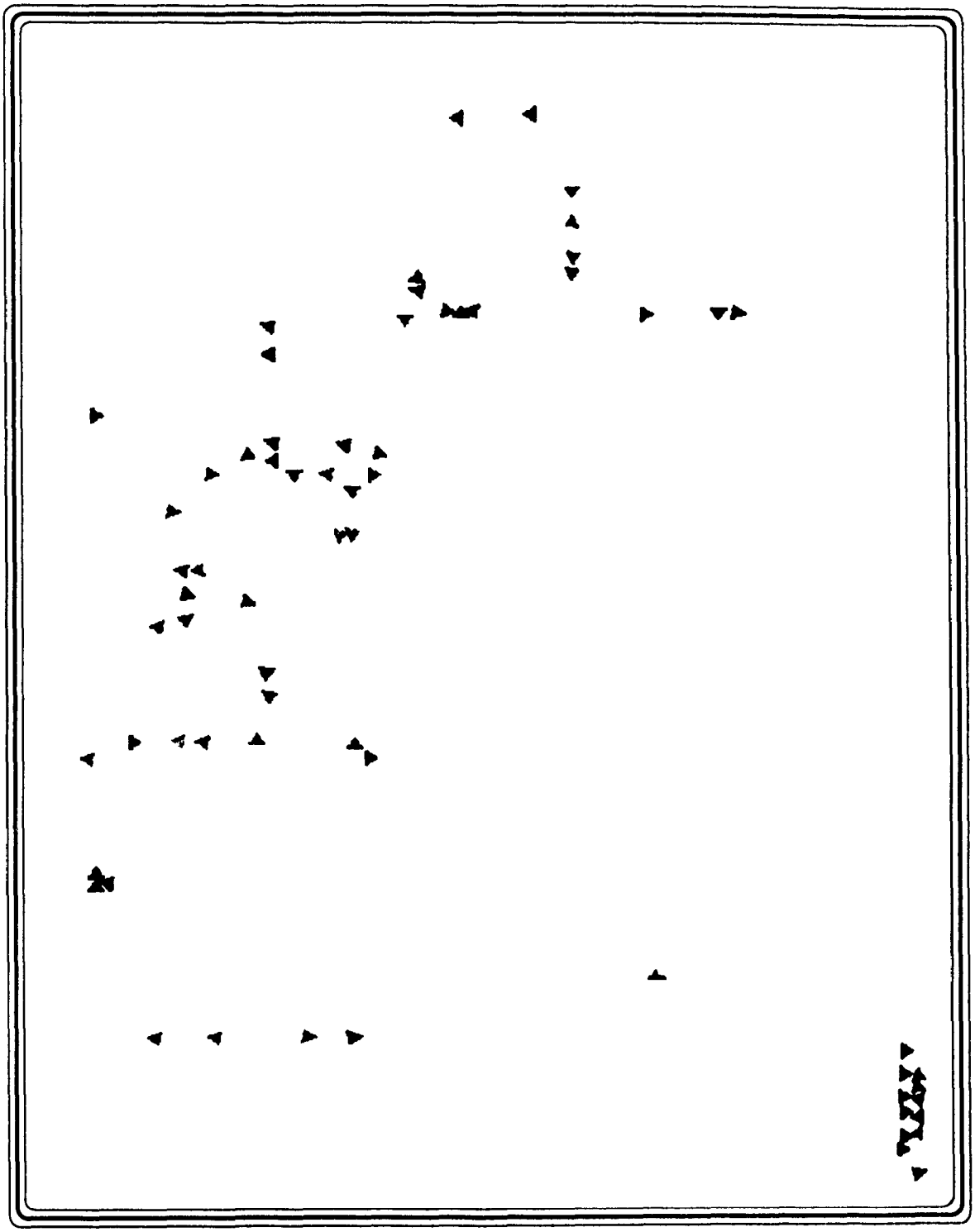


Figure 17: The Filled Triangle symbol's matches, using Hausdorff's distance method alone, found in the region of the map.

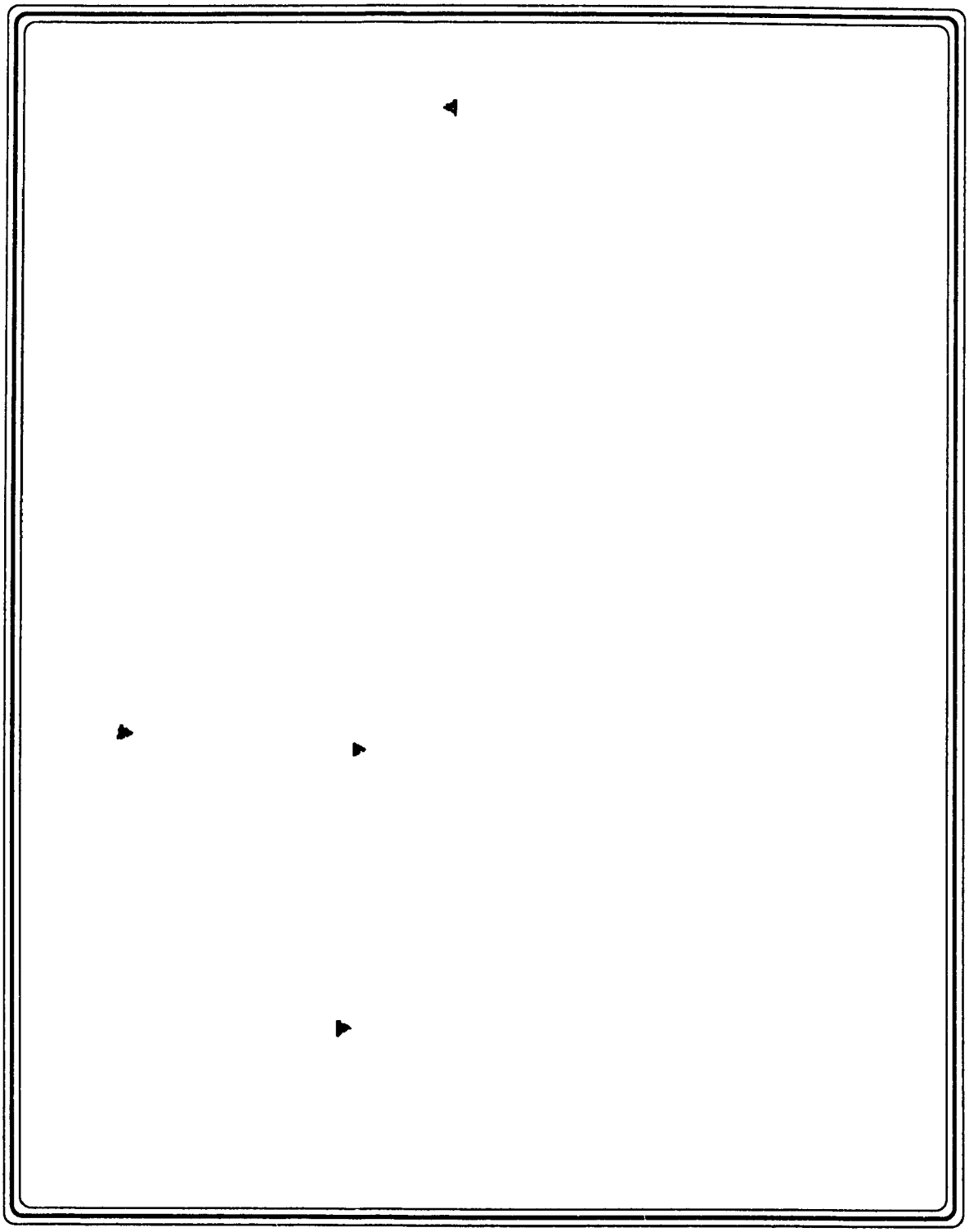


Figure 18: The results after passing all the Filled Triangle matches to a set of NNCs to filter out the False Positives.

Chapter 5

Results Achieved

Throughout this chapter, recognition results will be presented on the following data:

- the **CEDAR** database of numerals
- paragraphs of text from the **TRIGONIX** database, and
- special map symbols from the **CRIM** database.

A user interface was built and Figures of the interface utilities are presented in Appendix B.

Please recall that a threshold is used in the testing phase. When the values of the output neurons are calculated, we find the two neurons that have the highest values compared to the others. Next, we check if the difference between these two units is greater than the given threshold, then the current pattern is rejected, else it is classified into the same class of the neuron that has the higher value between the last two.

Training and testing the network was done on a SPARC station 10 that has 64 MB (mega bytes) of RAM (random access memory), with 50 MHz speed. It is important to note that the machine was not fully dedicated to our examinations meaning that the machine was connected to a network of other machines from which other users used to login to that machine too.

Please note that the reason for having spikes is related to changing the values of the eita, alpha and lambda parameters, and the number of hidden neurons. In the

case of changing either one or more of the parameters (eita, alpha, lambda), the error function is less affected and the spikes would be small, as they appear, for example, in Figure 20. On the other hand, when the number of hidden neurons is changed, the spikes are higher especially when the number of hidden neurons is incremented, as it explicitely shows in Figure 19.

5.1 The CEDAR database

The **CEDAR** database of numerals is made available by the *Center of Excellence for Document Analysis and Recognition*, a research center, at the State University of New York at Buffalo, directed by Dr. Sargur N. Srihari. This database consists of numerals extracted from zip-codes of the United States Postal Services office.

5.1.1 Training and testing the network

Many different architectures of the network were implemented and tested to check whether the network will converge and be able to generalize well. Hereafter, only two will be shown.

The first examination process

The architecture of the network is the following:

- an (18x18) binary image is presented to the network as input,
- 40 neurons[1] constitute the hidden layer, and
- 10 output neurons constituting the output layer.

The network was trained on 11550 patterns collected from the first six directories of the **CEDAR** database. The network achieved convergence after 8 hours of training, and Figure 19 shows the error function (Y-Axis) with respect to the number of iterations (X-Axis). Later, we examined the performance of the network on a subset of

[1] chosen from the neighborhood of the optimal number of hidden neurons

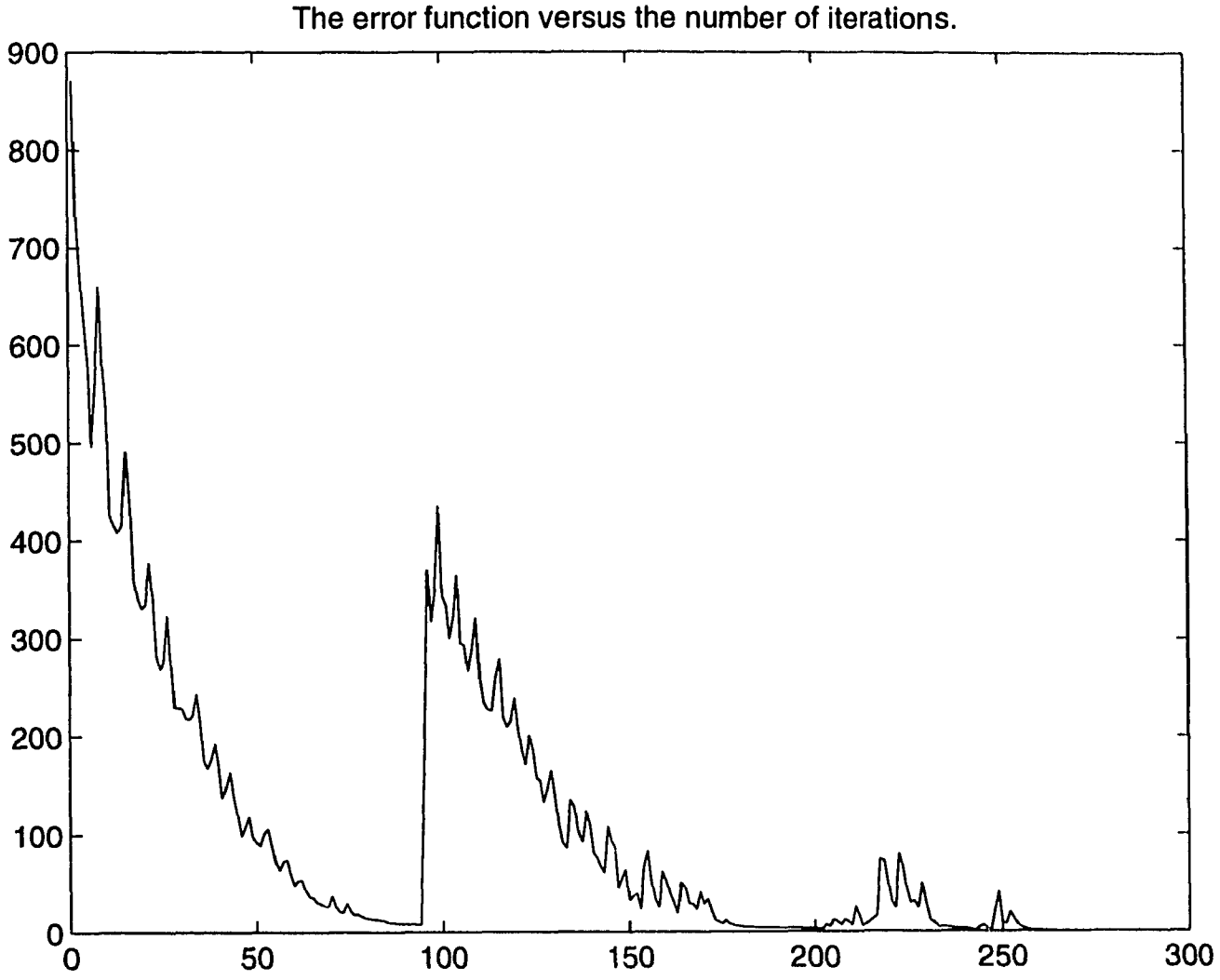


Figure 19: The graph of the error function for training a network made up of 18x18 input binary images, 40 hidden neurons and 10 output neurons.

	0	1	2	3	4	5	6	7	8	9	Rej	Total	Error
0	335	1	1	1	0	2	1	0	1	0	13	355	7
1	0	275	1	0	0	0	0	1	0	0	12	289	2
2	3	1	189	3	1	1	2	0	4	0	20	224	15
3	0	0	2	186	0	4	1	0	2	0	13	211	9
4	0	1	0	0	172	0	0	0	0	1	9	183	2
5	0	0	1	3	2	101	1	0	1	1	7	117	9
6	2	0	2	0	1	0	232	0	1	0	7	245	6
7	1	0	1	1	3	0	0	198	2	2	13	221	10
8	0	1	0	1	3	0	0	0	171	1	14	191	6
9	0	0	0	1	4	0	0	2	9	166	15	214	16
										Total:	123	2227	82

Table 3: The Confusion matrix of the *bad* testing set, after training on the first six subdirectories from the **CEDAR** database.

the *bad* testing set from the **CEDAR** database containing 2227 numerals. Table 3 shows the confusion matrix of the testing set.

After setting the threshold to a value of 0.5, out of the 2227 patterns 82 patterns were misclassified resulting in an error rate of 3.68%, and 123 patterns were rejected resulting in a rejection rate of 5.52%. Thus, the recognition rate was 90.79%, and the reliability rate was 95.24%.

The second examination process

We carried another *examination process* (training and testing) using the following architecture:

- a (16x16) binary image is presented to the network as input,
- 80 neurons[2] constitute the hidden layer, and
- 10 output neurons constituting the output layer.

[2]chosen from the neighborhood of the optimal number of hidden neurons

	0	1	2	3	4	5	6	7	8	9	Rej	Total	Error
0	344	0	0	0	0	0	1	0	1	0	9	355	2
1	0	284	1	0	0	0	0	0	0	0	4	289	1
2	2	0	195	1	1	0	0	0	1	0	24	224	5
3	0	0	4	179	0	5	0	0	1	1	18	208	11
4	0	1	0	0	171	0	1	1	1	1	7	183	5
5	1	0	0	2	0	96	0	0	3	0	15	117	6
6	3	0	1	0	2	0	231	0	0	0	8	245	6
7	0	0	0	0	5	1	0	208	1	1	4	220	8
8	0	0	0	3	1	1	0	0	174	1	11	191	6
9	0	0	0	0	5	1	0	2	0	165	8	181	8
										Total:	108	2213	58

Table 4: The Confusion matrix of the *good* testing set.

Test set	Error	Rejection	Recognition	Reliability
Good set	2.62 %	4.88 %	92.50 %	97.24 %
Bad set	4.46 %	6.46 %	89.08 %	95.43 %

Table 5: The classification results of the *good* and the *bad* testing sets.

We trained the network on the full **CEDAR** training set, made up of 18,468 patterns. The set is made up of 2866 patterns for the *0*, 2544 patterns for the *1*, 2047 patterns for the *2*, 1731 patterns for the *3*, 1676 patterns for the *4*, 1459 patterns for the *5*, 1722 patterns for the *6*, 1616 patterns for the *7*, 1453 patterns for the *8*, and 1354 patterns for the *9*. Figure 20 shows the error function of the training process.

The network was tested on the two different testing sets from the **CEDAR** database, the *good* set and the *bad* set. The *good* set consists of a total of 2213 patterns, while the *bad* testing set is made up of 2599 samples. The threshold for rejection was set to a value equal to 0.5. Tables 4 and 6 show the confusion matrices for the *good* set and the *bad* set.

Results of the error, rejection, recognition and reliability rates are shown in Table 5. A comparison between the accuracy of the NNC and other researchers' classifiers (same as in section **Comparing the results**, Chapter 3) is shown in Table 7, where the top first 2 choices are considered while classification.

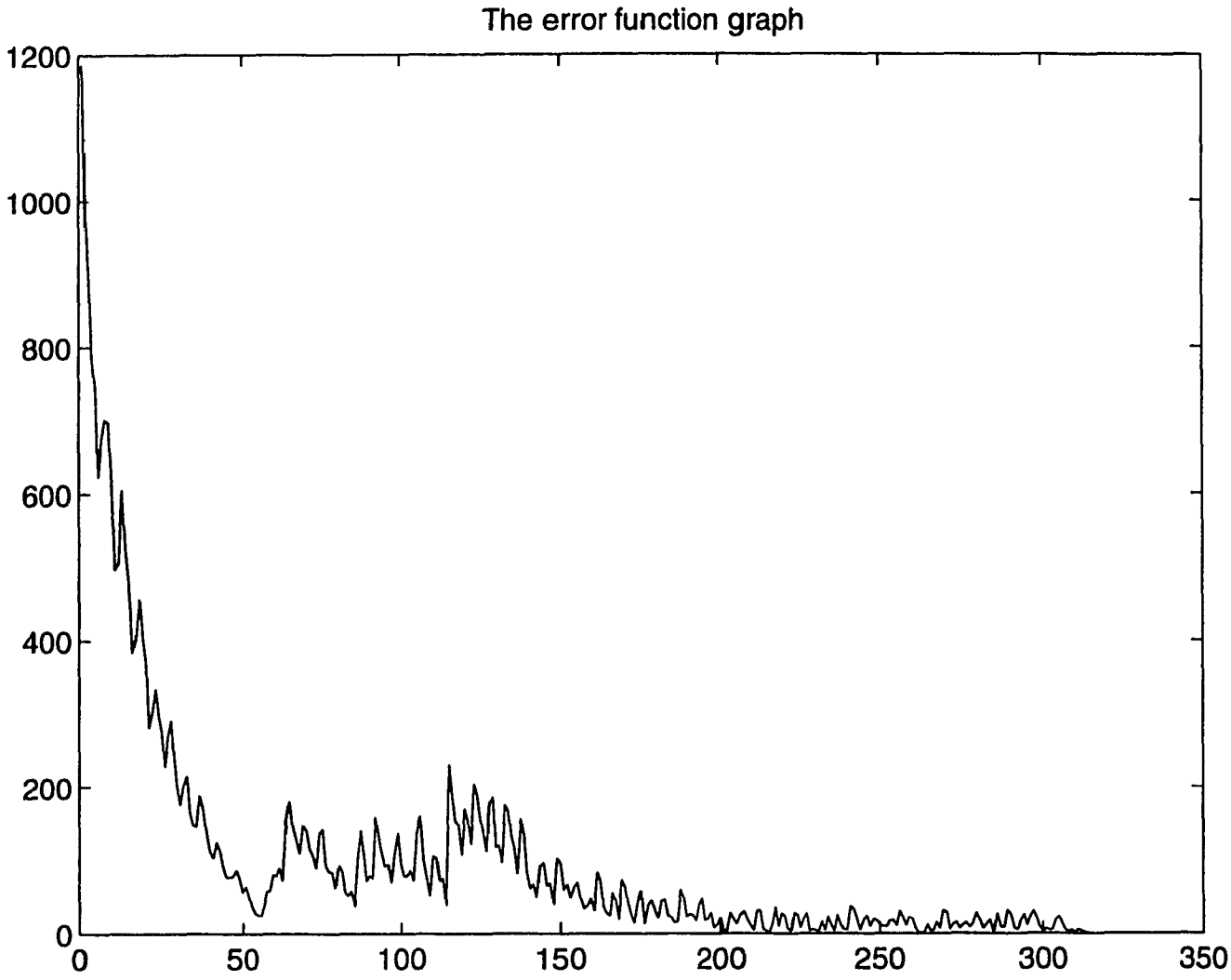


Figure 20: The graph of the error function for training a network made up of 16x16 input binary images, 80 hidden neurons, and 10 output neurons.

	0	1	2	3	4	5	6	7	8	9	Rej	Total	Error
0	407	0	0	1	1	2	6	1	2	1	11	432	14
1	1	331	2	0	0	0	2	1	0	0	8	245	6
2	3	1	253	2	2	1	1	1	1	0	30	295	12
3	1	0	5	215	0	5	0	0	2	2	30	260	15
4	0	1	0	0	210	2	2	2	1	3	13	234	11
5	3	2	0	7	0	155	0	0	3	0	23	193	15
6	3	1	2	1	3	0	255	0	0	0	16	281	10
7	1	0	0	0	5	1	0	219	1	3	11	241	11
8	0	0	0	3	1	2	0	1	195	1	14	217	8
9	0	0	0	1	8	1	0	3	1	185	12	201	14
										Total:	168	2599	116

Table 6: The Confusion matrix of the *bad* testing set.

Classifier	*good set*	*bad* set
NNC	97.24	95.43
Binary Image	98.33	96.61

Table 7: The recognition results of the Binary Image classifier compared to NNC.

5.2 Results of segmenting and recognizing alphanumeric characters

5.2.1 The neural network classifier

We followed the same terminology as described in **The Neural Network Trainer** section. The only thing different is that the testing process will use the final results of the weight and bias matrices (calculated in the training process) to test and classify patterns form the testing set of paragraphs. In the testing process, the pattern is either misclassified, rejected, or classified correctly. A threshold value, *thvalue* decided by the user, is used for rejection. This is done in the following manner: We first calculate the output vector *out* and find the first and second units with higher output values, *max1* and *max2* respectively. Then,

if (max1-max2 < thvalue) $\Rightarrow$ *the pattern is rejected*

else if(code $\neq$ *maxcode1)* $\Rightarrow$ *the pattern is misclassified*
else the pattern is classified correctly.

where *code* is the current pattern's identification (read from the testing set), *maxcode1* is the index to the unit that has the highest output in the output vector *out*, *max1* and *max2* are the two highest values of the neurons in the output layer, and *thvalue* holds the value of the threshold used (0.25).

While testing the performance of the neural network classifier, we found that the network had conflicts between *I* and *1*, and between *O* and *0* (zero). A solution to that problem was implemented and it is described below. It proved to decrease the error rate of misclassification.

Differentiating between *I* and *1*

After calculating the values of *max1* and *max2*, we first check if any of them corresponds to the neuron of *I* or *1*. If so, then we check whether the pattern that preceded it was a character or numeral. If it was a character (or a space), then the pattern is classified as an *I*, otherwise it is classified as a *1*.

Differentiating between *O* and *0*

Second, if *max1* and *max2* didn't correspond to neither *I* nor *1*, we do the same thing as for *O* and *0*. However, if the pattern that preceded it was a character (or a space), then the pattern is classified as an *O*, else it is classified as a *0*.

5.2.2 Results

Since the results of line and pattern segmentations were already shown in chapter 4 entitled **Segmentation**, only the results of classification will be presented in this subsection. The neural network classifier was *trained on a training set* made up of **2245** patterns that were extracted from more than 40 sample paragraphs. It took the network *9 hours and 14 minutes* to converge. The performance of the neural network classifier was examined on 12 new sample paragraphs. Results showed a recognition

rate of 92.46%, an error rate of 3.22%, and a rejection rate of 4.32% per segmented patterns, and not per processed paragraphs. The problems faced are mainly related to pattern segmentation, that is some parts of the patterns which caused an error or rejection, have been cut or partially distorted. Moreover, the absence of a huge database that contains instances of each pattern for each *font* that is used at different scanning levels has its impact on the performance of the network.

However, we should consider that the neural network classifier was also trained on a small training set which is not uniformly distributed. By uniform distribution we mean the following:
Given a data set **S** made up of X patterns, then:

$$\exists \textit{ exactly } N \textit{ samples } \forall \textit{ pattern } P_i, \textit{ where } i \in \{1, \ldots, X\} \tag{16}$$

Figures 21, 22, ..., and 29 represent the paragraphs that were used for testing and their respective results. The ampersand @ character symbol is not considered as a symbol to be recognized in this application, but it is used to point out that the current pattern has been *rejected* by the classifier, and thus **it will be saved in a file for further training in the future**. In addition, the punctuation marks were not considered to be part of the patterns set that we intended to segment and recognize throughout this application. Please note that the paragraph in Figure 21 contains some symbols like; *(,) / and* - . Notice that we had a misclassification in this case; *D* was misclassified with an *O*, the fourth word in line three.

In Figure 22, Pattern *5* was rejected because initially the training set didn't have more than 15 samples. Pattern *d* is simply a lower case and wasn't even considered for training, and it is very critical and important to have it rejected rather than misclassified.

In Figure 23, we had a problem of segmentation because we had a blank line cutting the paragraph from the top to the bottom, in the middle, and passing through some patterns. Segmentation errors emerged when we were segmenting them. Certainly any mistake in segmentation will directly result in a recognition error of the pattern. Example of those cut patterns are *b* the first character of the fourth word in

The input paragraph:

DISJ. 6.9 kV-CB/A2 (5334-T41)
TRANSFORMATEUR
DÉCL. OU PERTE DE 250 VCC

Its respective output:

DISJ 6 9 KV-CB/A@ (5334-T41)
TRANSFORMATEUR
DECL OU PERTE OE 250 VCC

Figure 21: A paragraph tested with its respective output.

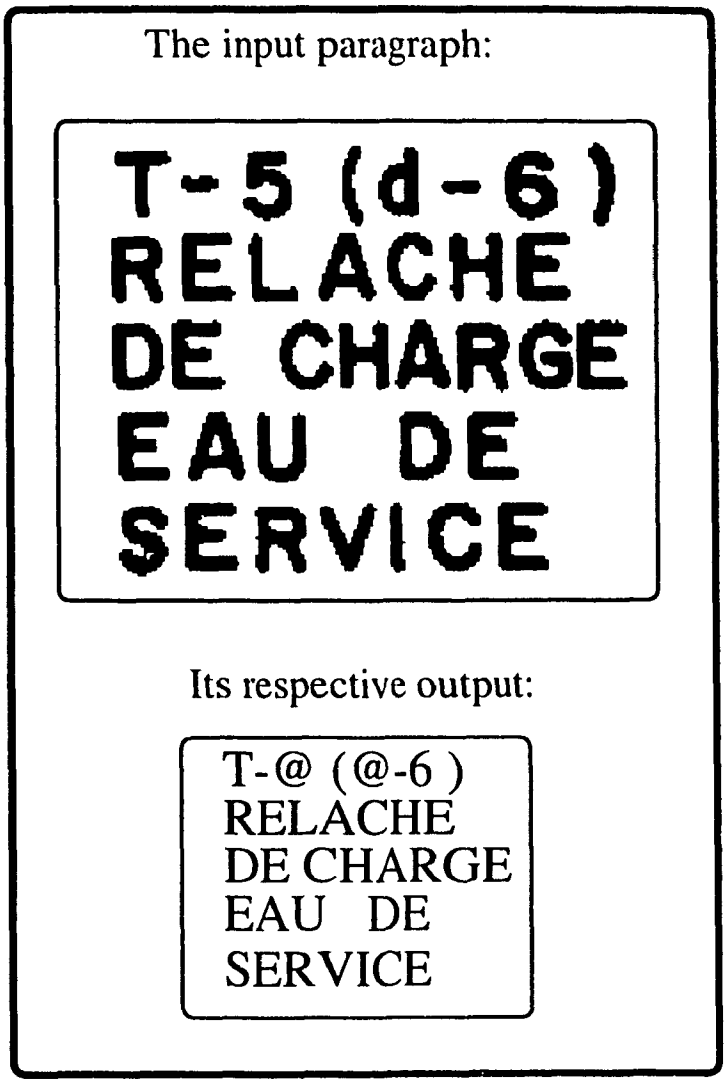


Figure 22: Example of a paragraph that has 2 rejected patterns, *5*, and *d*.

the seventh line, E the fourth character of the third word in the fifth line, and R the sixth character of the second word in the second line. In the same token, Figure 25 had one character, A in the second line, rejected because it was not fully connected.

The paragraph in Figure 24 had only one rejected pattern. The reason is related to the small size of the training set. Such problems will not be faced later because these patterns will be presented to the network for training.

If a new character or symbol was presented to the network in the testing phase, the network will tend to reject it as is the case with Q in Figure 26. This character will be saved in a file for future training of the network.

After many experiments on the **TRIGONIX** database of paragraphs, we found out that most of them contained patterns that were not fully connected, which was a result of scanning and binarizing the engineering drawings and maps. That is why some characters were rejected, because they were not fully connected. In trying to avoid this problem, a heuristic approach was introduced to the standard connected components segmentation algorithm, CCSA. Simply, it checks if the minimum distance, d_{min}, between two distinct labeled regions, reg_i and reg_j, is less than 3 pixels width, then the two regions are connected and considered as *one* region, as in Formula 17.

$$if\ d_{min}\ between\ reg_i\ and\ reg_j\ <\ 3\ \ then\ reg_i\ and\ reg_j\ are\ merged\ into\ reg_i. \quad (17)$$

Figure 27 shows the testing results before the modification was introduced to CCSA where some patterns were rejected. On the other hand, Figure 28 shows the testing results of the same pattern after modifying CCSA where all the previously rejected patterns were correctly recognized. Figure 29 shows the results of another tested paragraph.

The input paragraph:

SAUF INDICATIONS CONTRAIRES, TOUS LES NUMÉROS D'INSTRUMENTS SONT PRÉCÉDÉS DU Nº U.S.I. 67340.

UTILISER CE BOYAU LORS DE MANIPULAT DU FILTRE MODÉRATEUR.

VOIR AUSSI EN b-7 POUR VENTILATIOI GÉNÉRALE

Its respective output:

SAUF INDICATIONS @ONTRAIRES TOUS LES NUMEROS D INSTF@UMENTS SONT PRECEDES DU NO U S I 67340
UTILISER CE BOYAU LORS DE MANIPULAT DU FILTRE MODI@RATEUR
VOIR AUSSI EN I@-7 POUR VENTILATIOI GENERALE

Figure 23: A paragraph showing some of the segmentation problems encountered.

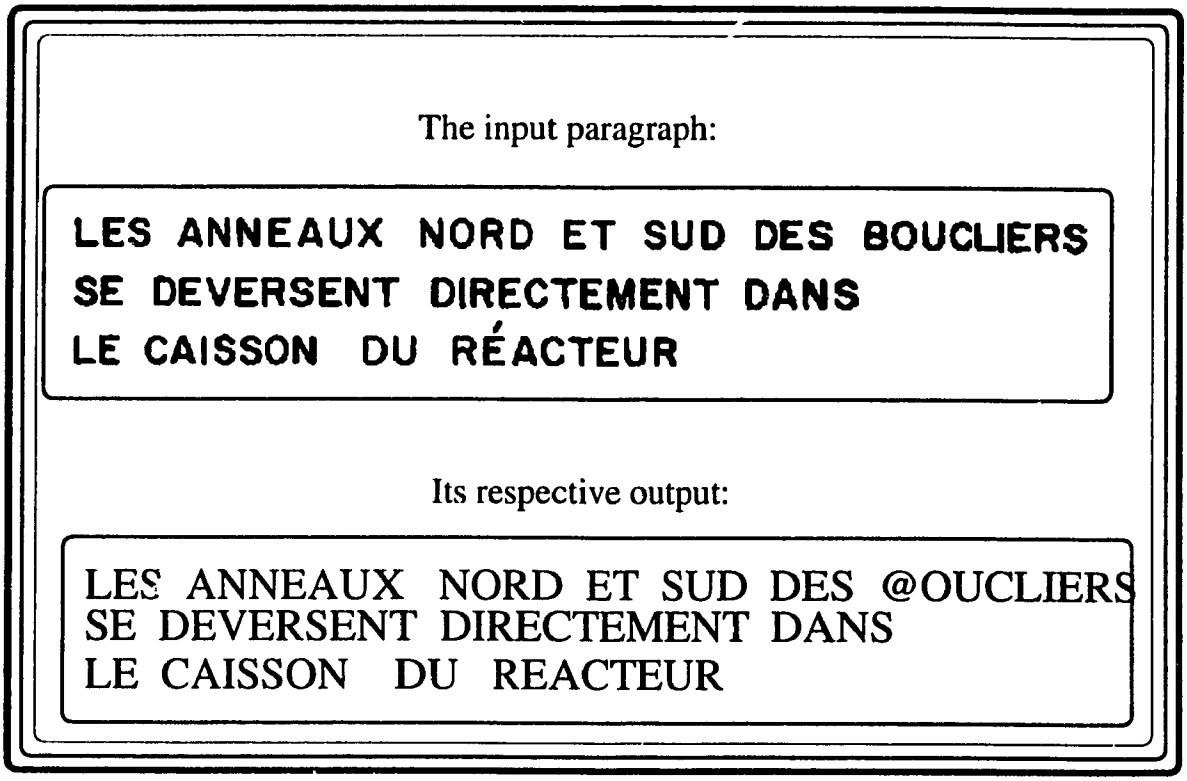


Figure 24: This tested paragraph had only one rejected pattern.

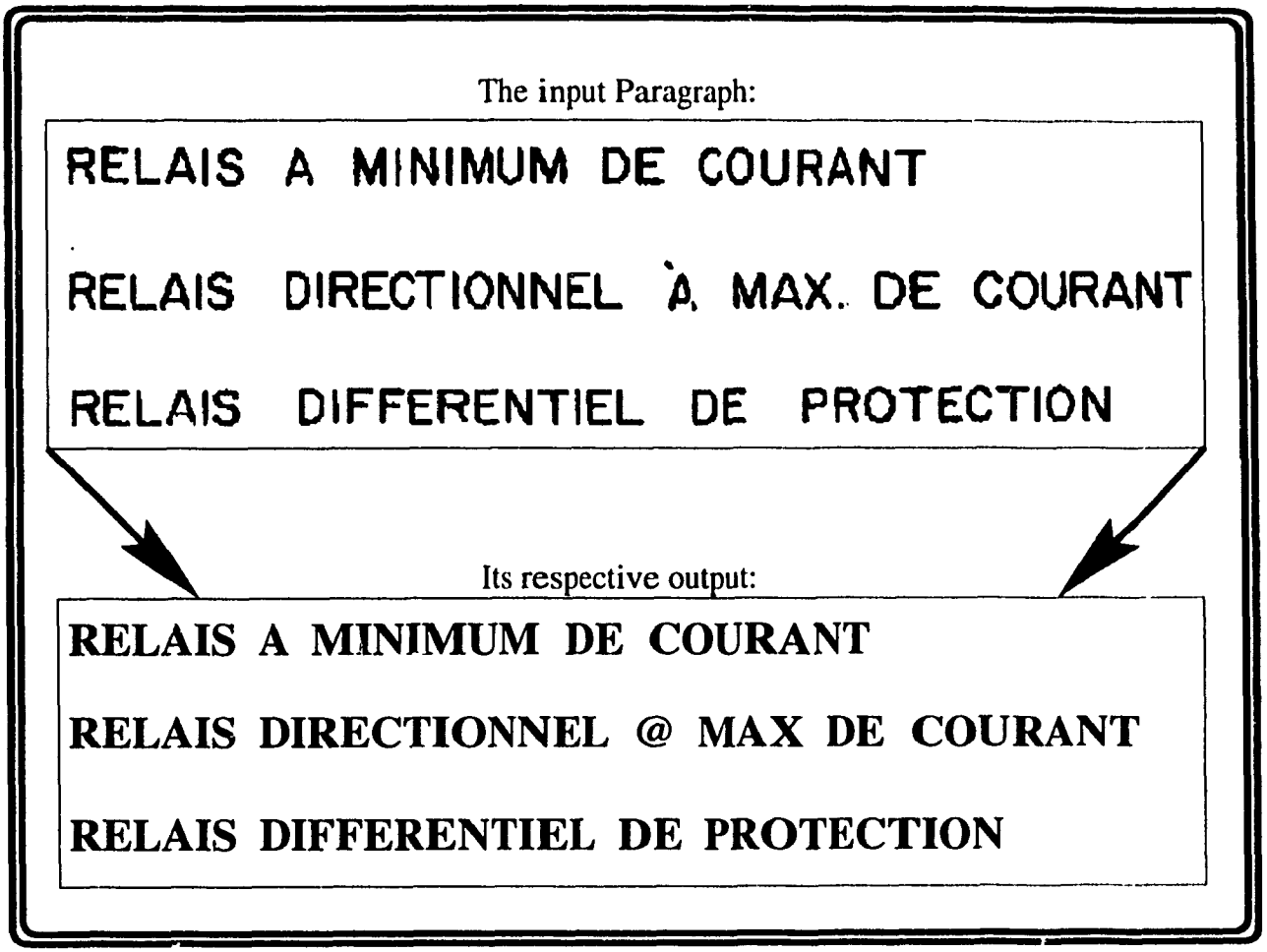


Figure 25: Another paragraph containing a character that was mal-segmented.

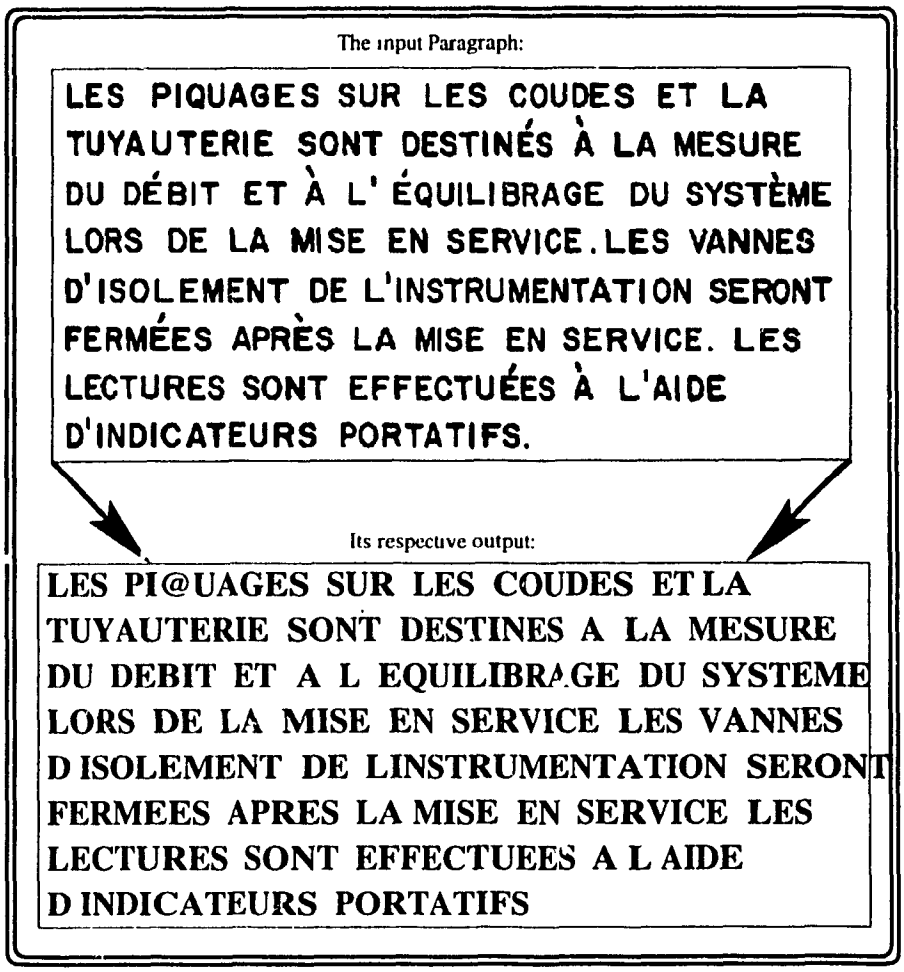


Figure 26: The character Q was rejected because it was new to the network.

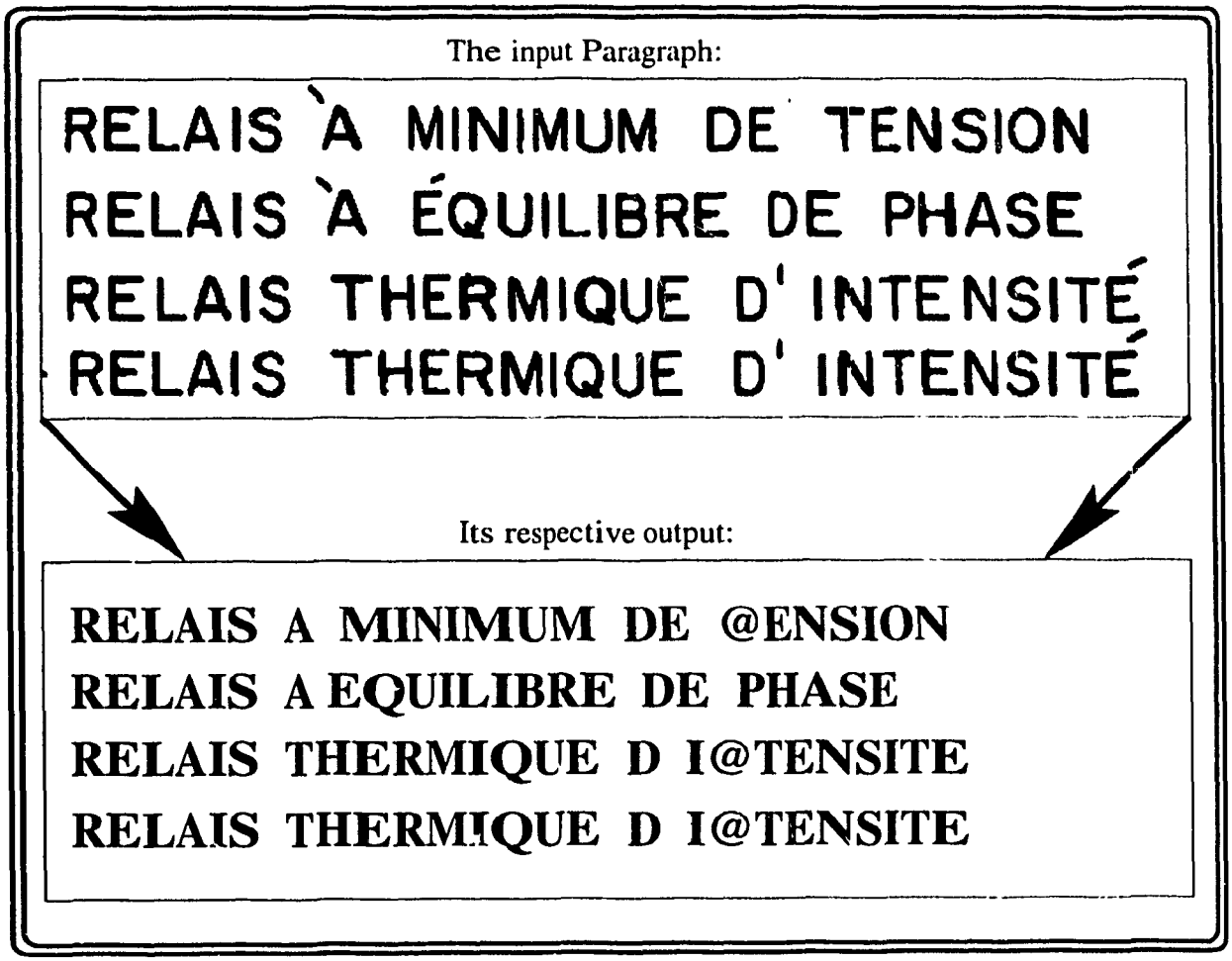


Figure 27: This tested paragraph had 3 rejected patterns before improving the CCSA.

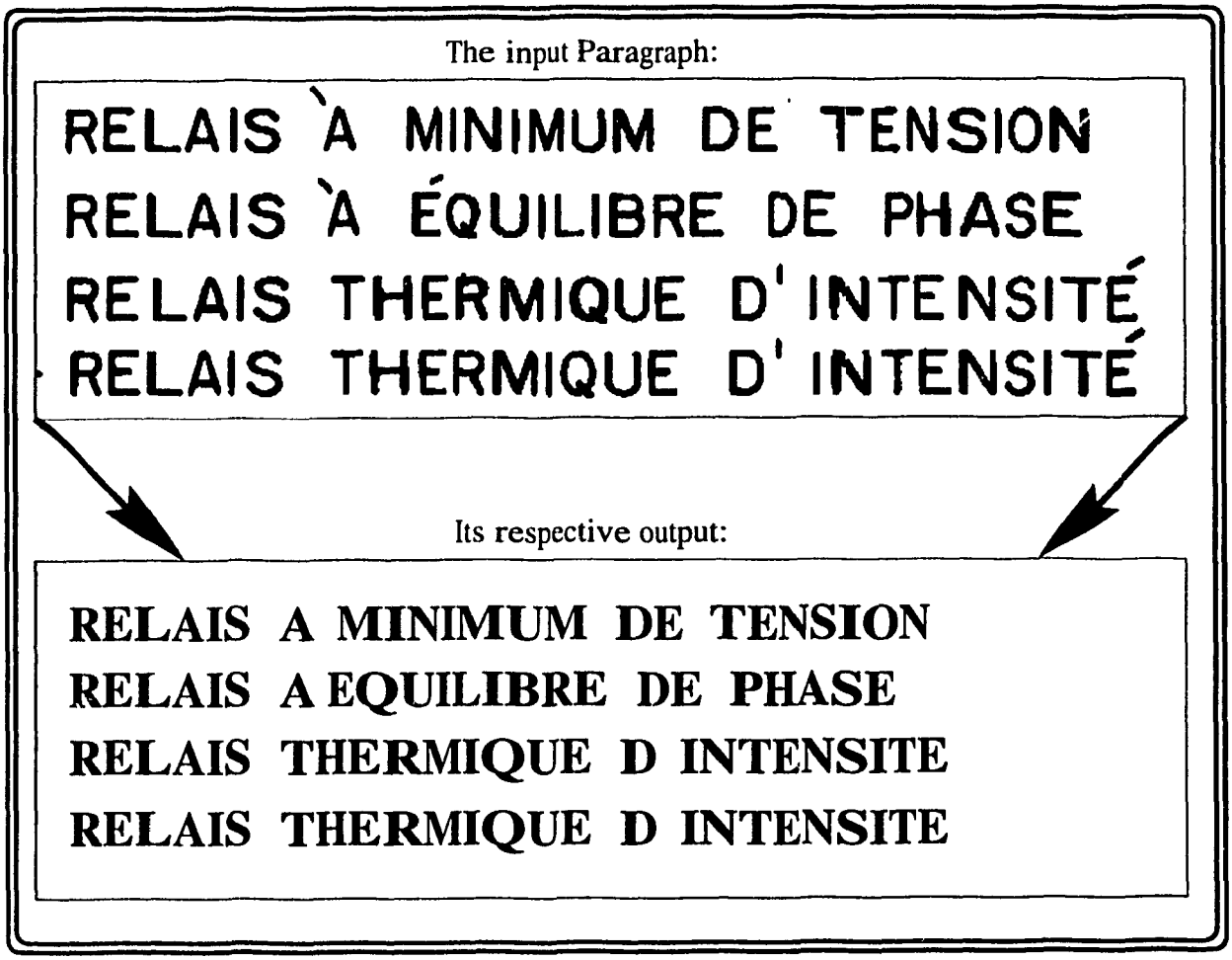


Figure 28: This tested paragraph was completely recognized after improving the CCSA.

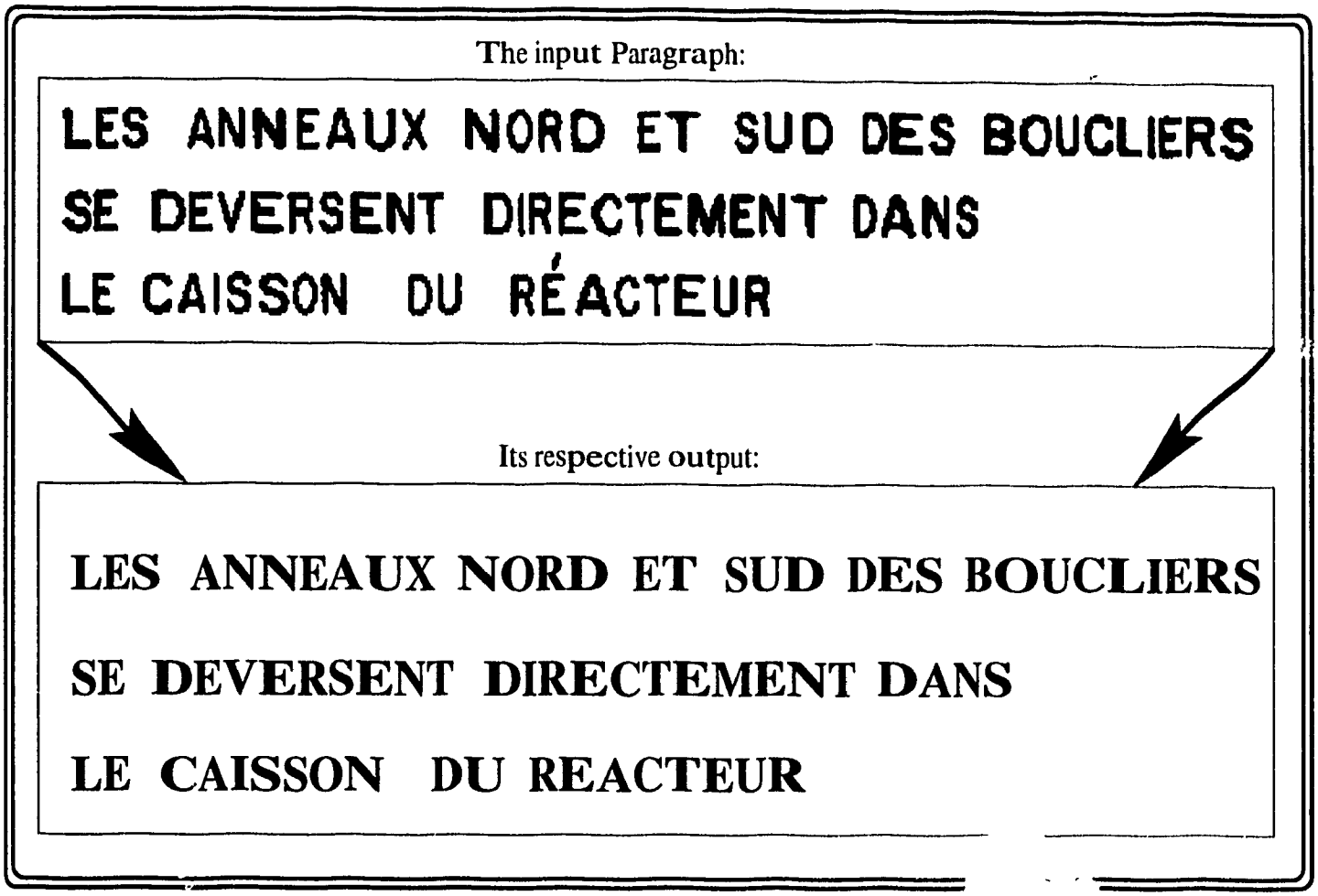


Figure 29: This tested paragraph no rejected patterns.

5.3 Results of segmenting and recognizing symbols

5.3.1 Definitions

The following definitions are used throughout this section:

- **False Positive**: Before combining the Hausdorff method with the neural network classifier, we used to have some patterns being labeled as a symbol (say Cross) but in fact it was not. This led to a false matching of the symbol with extracted data (which is close enough in shape to that symbol) from the map. Since the extracted data is not the actual symbol, we labeled it as a False Positive and trained the network to learn how to separate the class of the False Positive from the class of its actual symbol.

- **Rej**: It stands for rejection. When the difference between the output of the two neurons in the output layer is less than a given threshold TH, the pattern is rejected. For example, given a network whose output layer consists of 2 neurons N1 and N2. In the testing phase, when a pattern P is presented to the network, N1 and N2 have calculated values of V1 and V2 respectively. P is rejected if the difference between V1 and V2 is less than TH.

5.3.2 The architecture of the neural network classifier

Figure 30 shows the general architecture of our network where the number of units in the input, hidden or output layers can be modified. The output layer is made up of 2 units (neurons). One unit is for the symbol itself and the second is for its False Positives. In the classification or the testing phase, the unit with the higher output will win over the other unit, thus leading to classifying the currently tested pattern as its respective class. In case the difference between the outputs of the two units is less than a certain threshold, the pattern will be rejected. The input to the network is a normalized (26x26) binary image of the pattern.

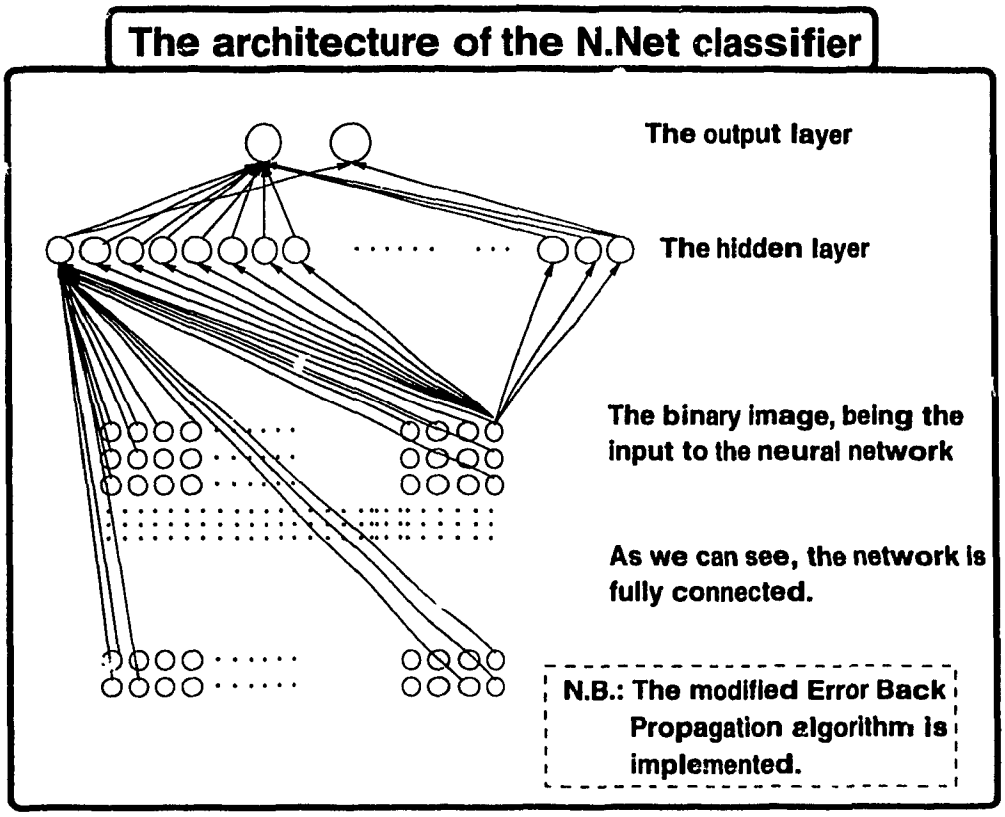


Figure 30: The architecture of the neural network for each symbol.

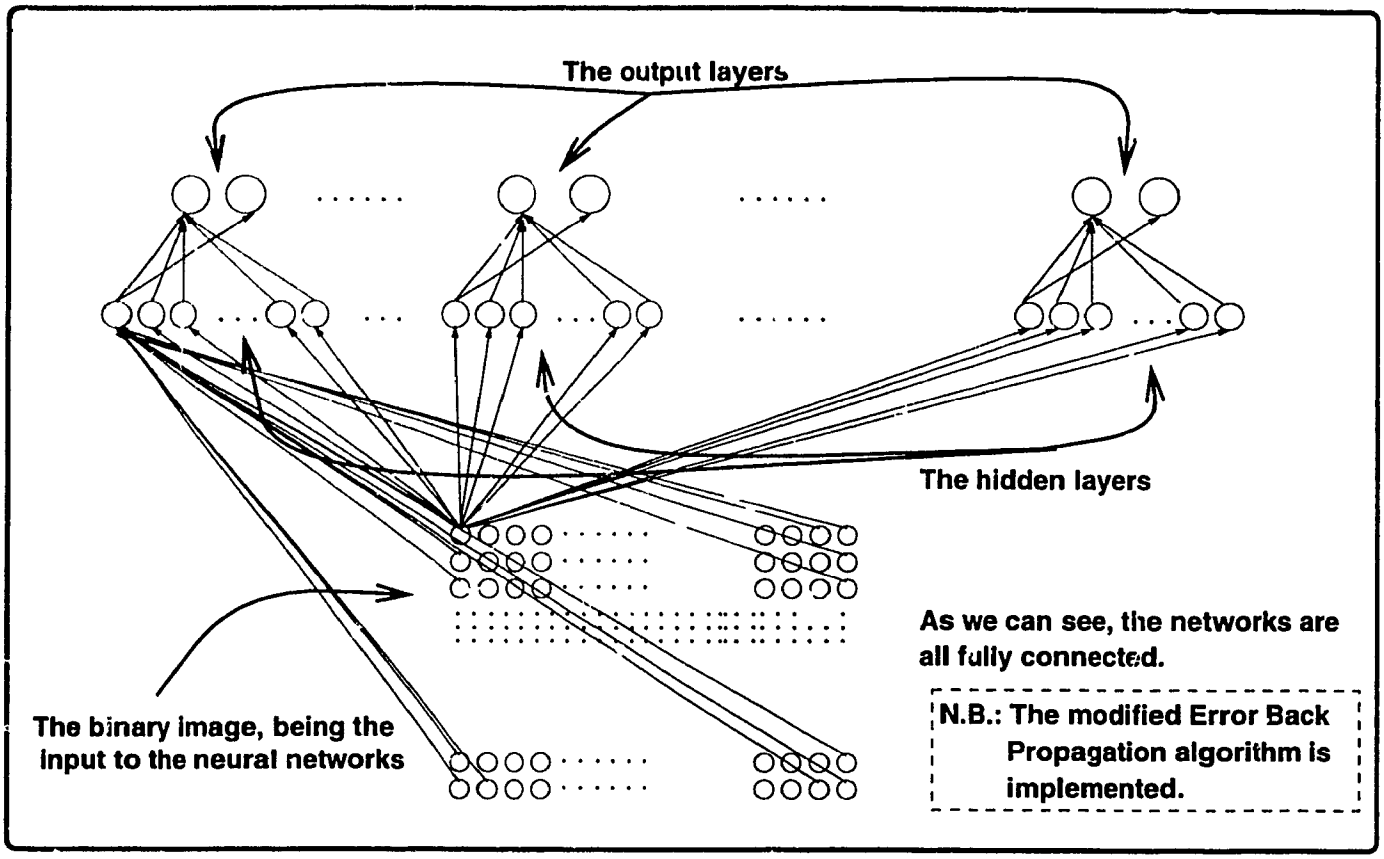


Figure 31: The dynamic architecture of the combined neural networks.

Later, when the network of each symbol will be combined with the Hausdorff's system, the architecture of the networks will be implemented as shown in Figure 31. Since adding a new symbol to be classified is optional, we chose to build one network per symbol. *Thus, if a new symbol, S_i, is to be added, all we have to do is to create an additional network Net_i associated with S_i.* In the training phase, Net_i will be trained on a data set representing samples of S_i and its False Positives.

This approach is better than having one neural network whose output layer consists of neurons where each neuron corresponds to its respective symbol. In such a case, adding a new symbol, will cause a change in the network's connections (architecture) and thus forces us to train the network again on the data set which should contain the samples of all the symbols.

5.3.3 Results

Please note that all of the training and testing sets were produced by the Hausdorff's method. Figures 32, 33, 34, 35, and 36 show parts of the map that was used for creating the training sets of the 5 symbols, shown in Figure 50. Parts of the map that was used for testing are shown in Figures 13 and 16.

Results of the *Cross* symbol

Training was done on a data set made up of 49 patterns, consisting of:

- 10 patterns representing the Cross symbol, and
- 39 patterns being False Positives.

Figure 37 shows the graph of the error function. Note that the network converged after 16 iterations with a total error value per iteration less than 0.035 per iteration. The network took 21 minutes and 46 seconds to converge. Even though the network was trained on a SPARC station 10, yet the load average was ranging between 2.14, 1.99, and 1.84. This means that if we train the network on a machine that is dedicated to our application, then the processing time will be much less.
Testing was done on a data set made up of 87 patterns, consisting of:

- 5 patterns representing the Cross symbol, and
- 82 patterns being False Positives.

Table 8 shows the confusion matrix of the testing set after presenting it to the network.

Results show that 82 False Positives (originally considered as Cross symbols by the Hausdorff method) dropped down to 4, and that 5 of them were rejected. On the other hand, one of the Crosses was considered as a False Positive. The recognition rate was 88.51% {(73+4)/(87)}, and the reliability was equal to 93.90%.

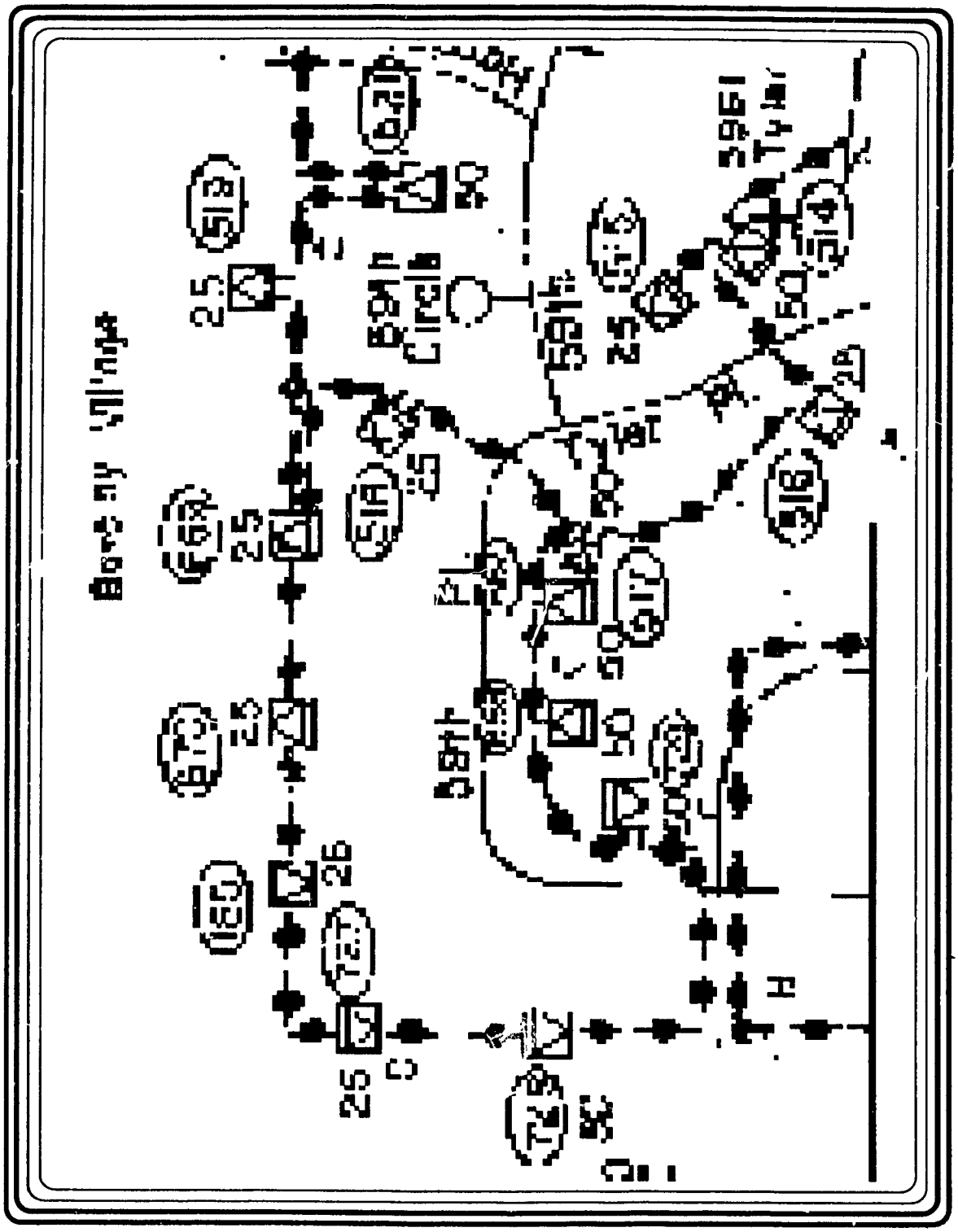


Figure 32: A part of the map that was used for creating the training sets of the symbols.

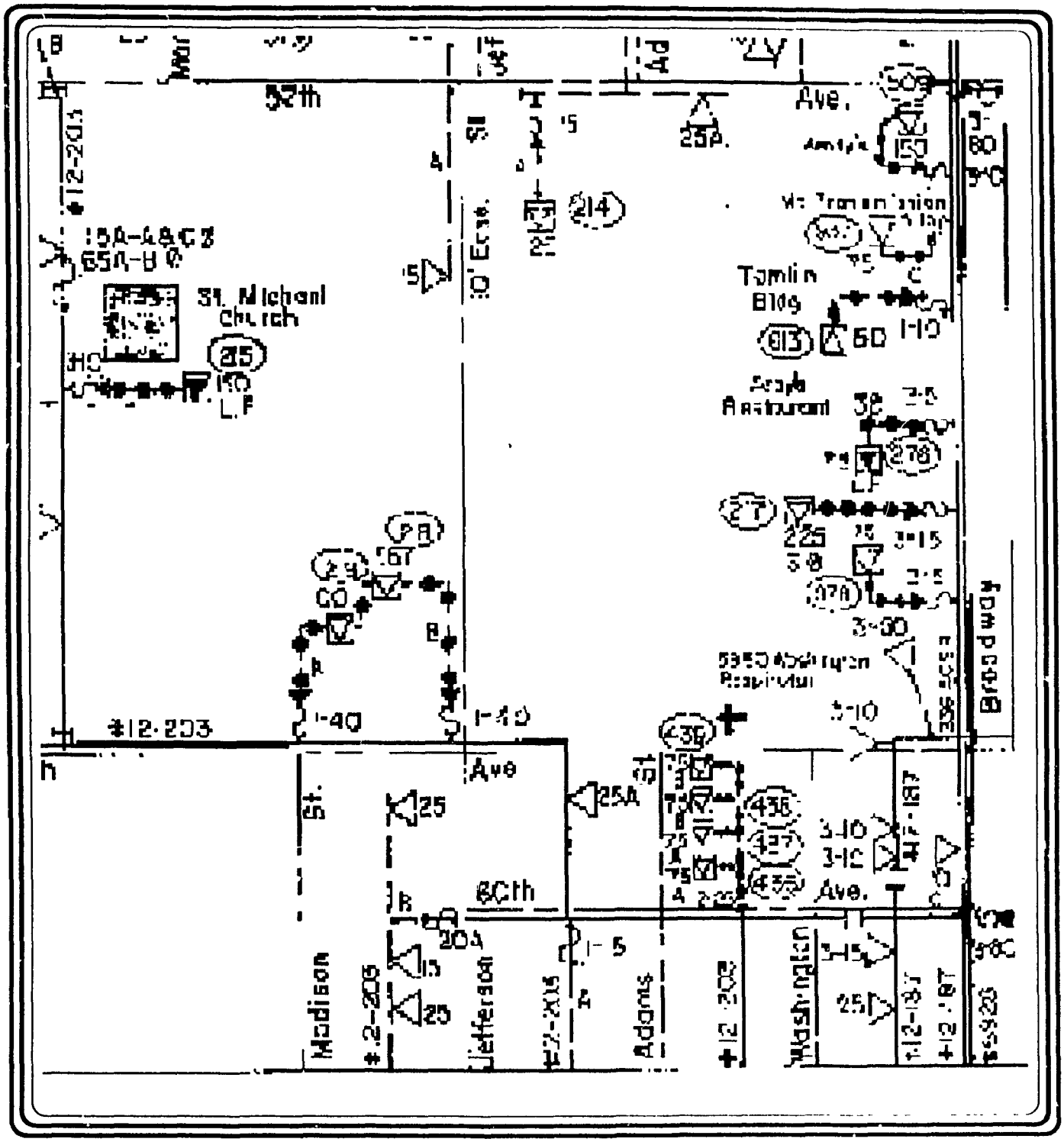


Figure 33: A part of the map that was used for creating the training sets of the symbols.

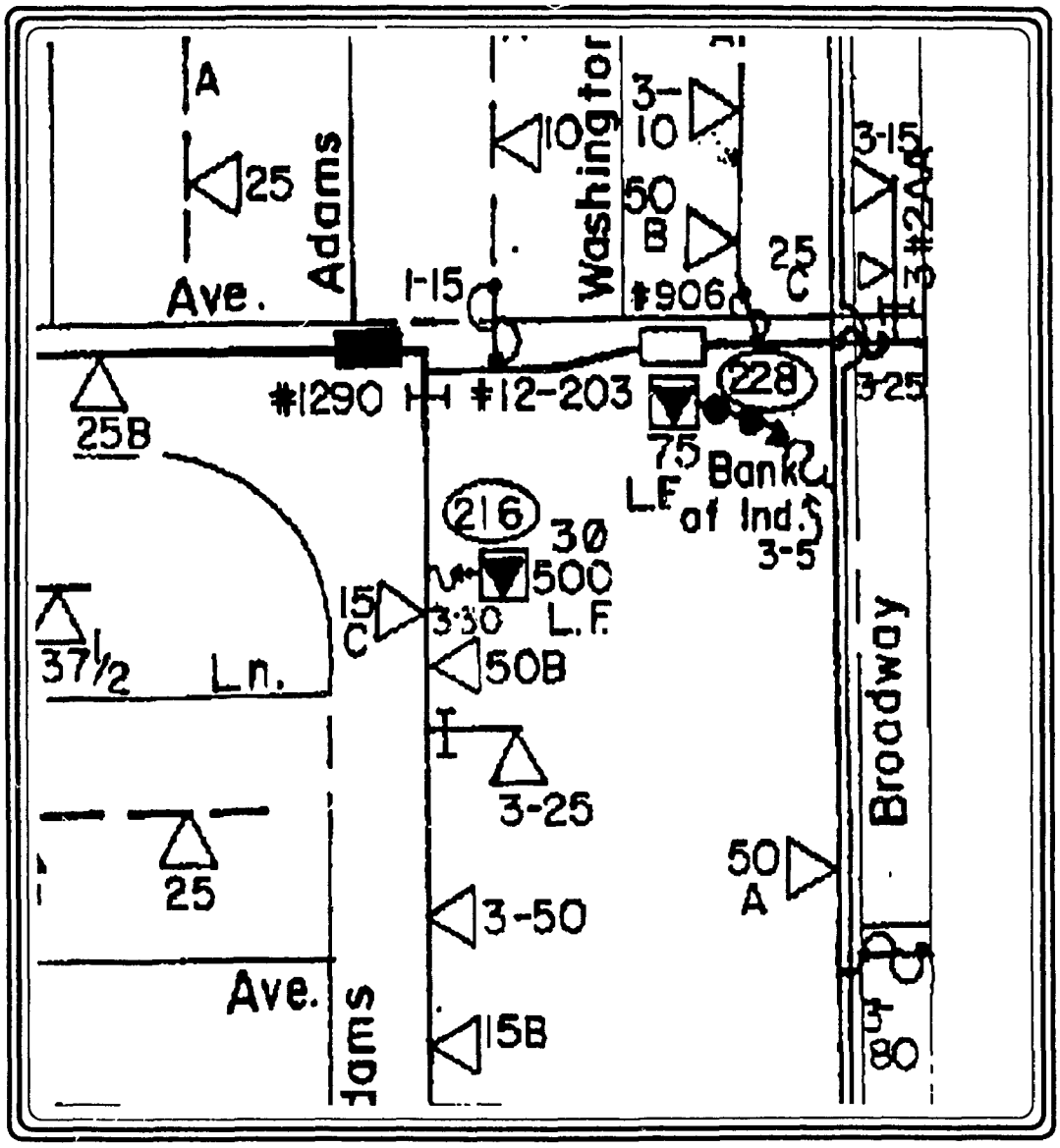


Figure 34: A part of the map that was used for creating the training sets of the symbols.

	False Positive	Cross	Rejected
False Positive	73	4	5
Cross	1	4	0

Table 8: Performance of the Cross NNC on the testing set.

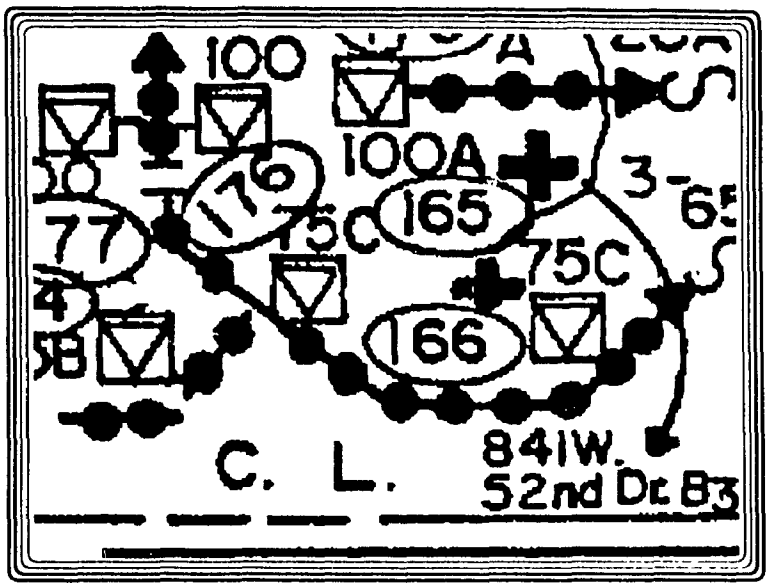


Figure 35: A part of the map that was used for creating the training sets of the symbols.

Results of the *Ellipse* symbol

Training was done on a data set made up of 135 patterns, consisting of:

- 61 patterns representing the Ellipse symbol, and
- 74 patterns being False Positives.

Figure 38 shows the graph of the error function. Note that the network converged after 16 iterations with a total error value per iteration less than 0.005 per iteration. The network took 35 minutes and 34 seconds to converge.
Testing was done on a data set made up of 178 patterns. The set consisted of:

- 48 patterns representing the Ellipse symbol, and
- 130 patterns being False Positives.

Table 9 shows the confusion matrix of the testing set after presenting it to the network.

Results show that the 130 False Positives (originally considered as Ellipse symbols by the Hausdorff method) dropped down to 12, and that 10 of them were rejected.

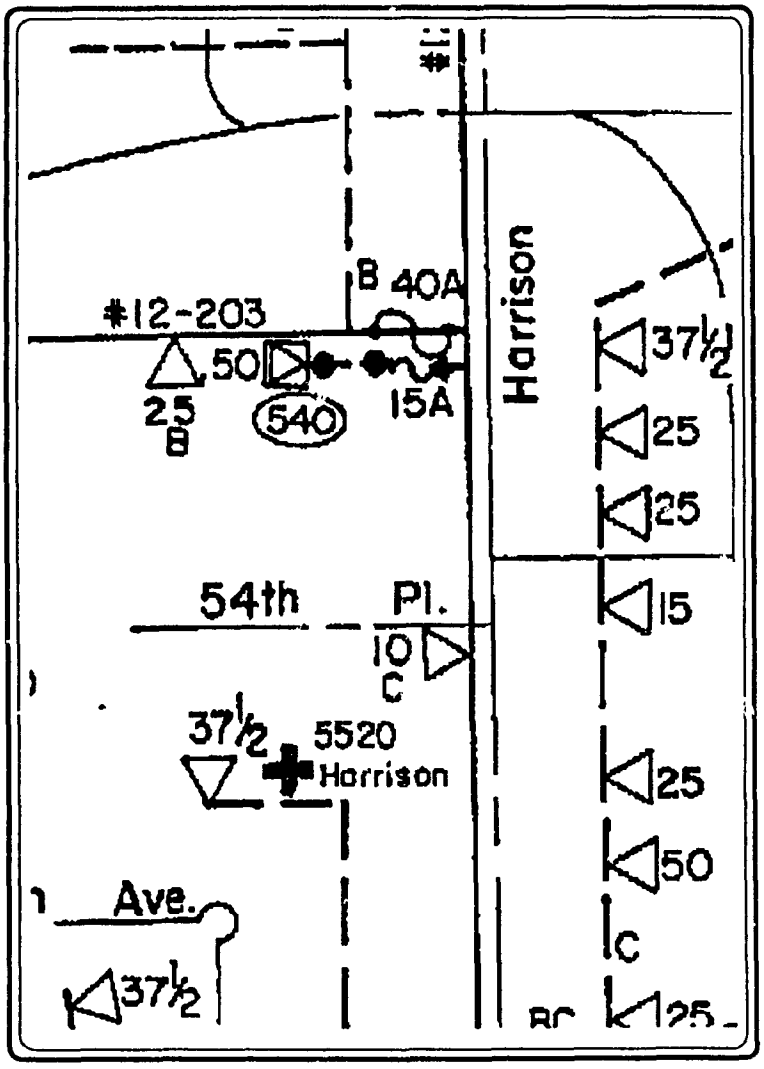


Figure 36: A part of the map that was used for creating the training sets of the symbols.

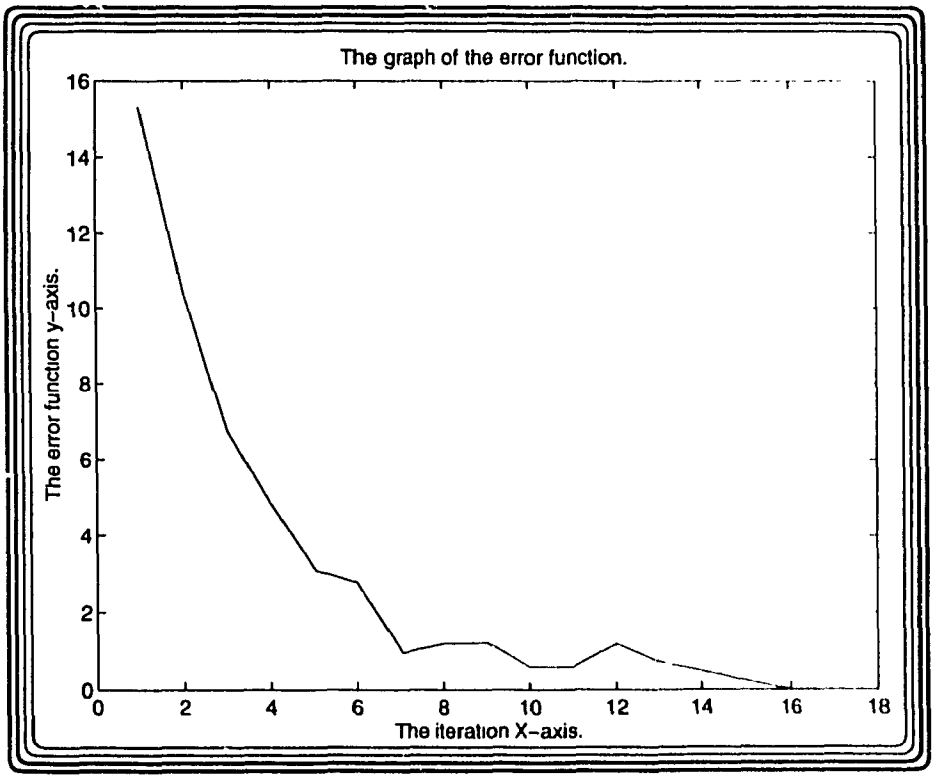


Figure 37: The error function graph of the Cross symbol.

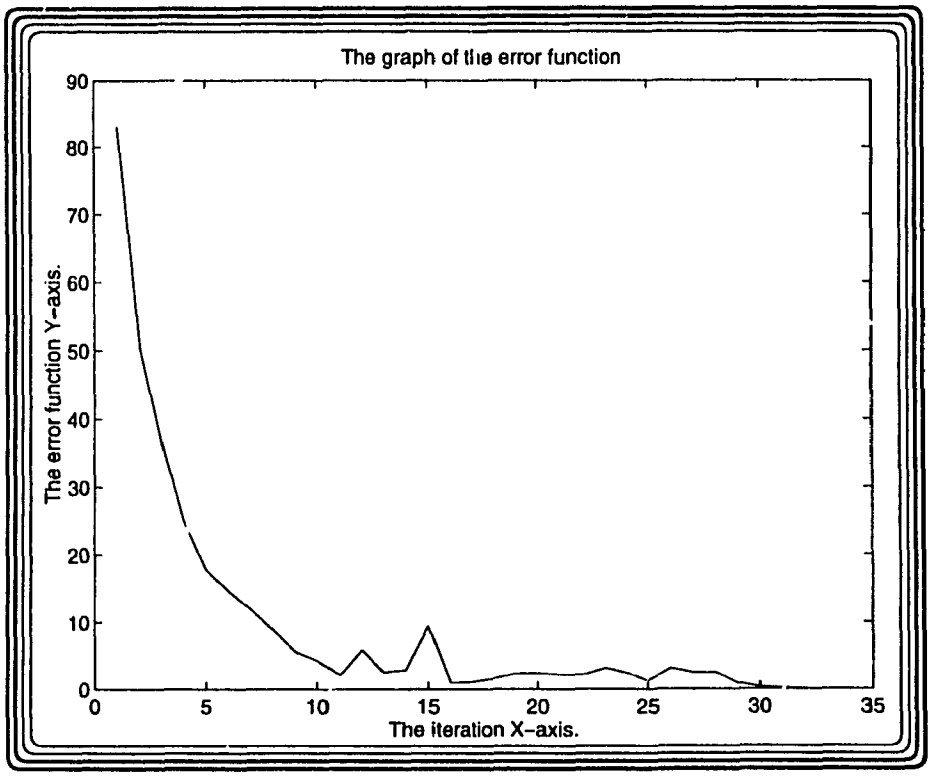


Figure 38: The error function graph of the Ellipse symbol.

	False Positive	Ellipse	Rejected
False Positive	108	12	10
Ellipse	0	11	7

Table 9: Performance of the Ellipse NNC on the testing set.

The rest (i.e. the 108 patterns) were identified correctly as being False Positives. It is good that none of the Ellipse patterns were recognized as being False Positives, and that's why we see a zero in the intersection of the first column (FP) with the second row (E). However, 7 of the Ellipse patterns were rejected. The recognition rate was 83.71%, and the reliability was equal to 92.25%. Figure 39 shows the patterns of the False Positives that resulted in an error, whereas Figure 40 shows the patterns of the Ellipse symbol that were rejected. Such problems will be eliminated when training is done on a large data set.

Results of the *Transformer* symbol

Training was done on a data set made up of 178 patterns, consisting of:

- 48 patterns representing the Transformer symbol, and
- 130 patterns being False Positives.

Figure 41 shows a graph of the error function. Note that the network converged after 34 iterations, with a total error value per iteration less than 0.005 per iteration. The network took 43 minutes and 17 seconds to converge. Figure 42 shows some instances of the Transformer's symbols found in the training set, that were extracted by Hausdorff's method and labeled manually by the user. Notice the presence of some instances of the Transformer's False Positives that were labeled as Transformer's symbols.

Testing was done on a data set made up of 117 patterns, consisting of:

- 51 patterns representing the Transformer symbol, and
- 66 patterns being False Positives.

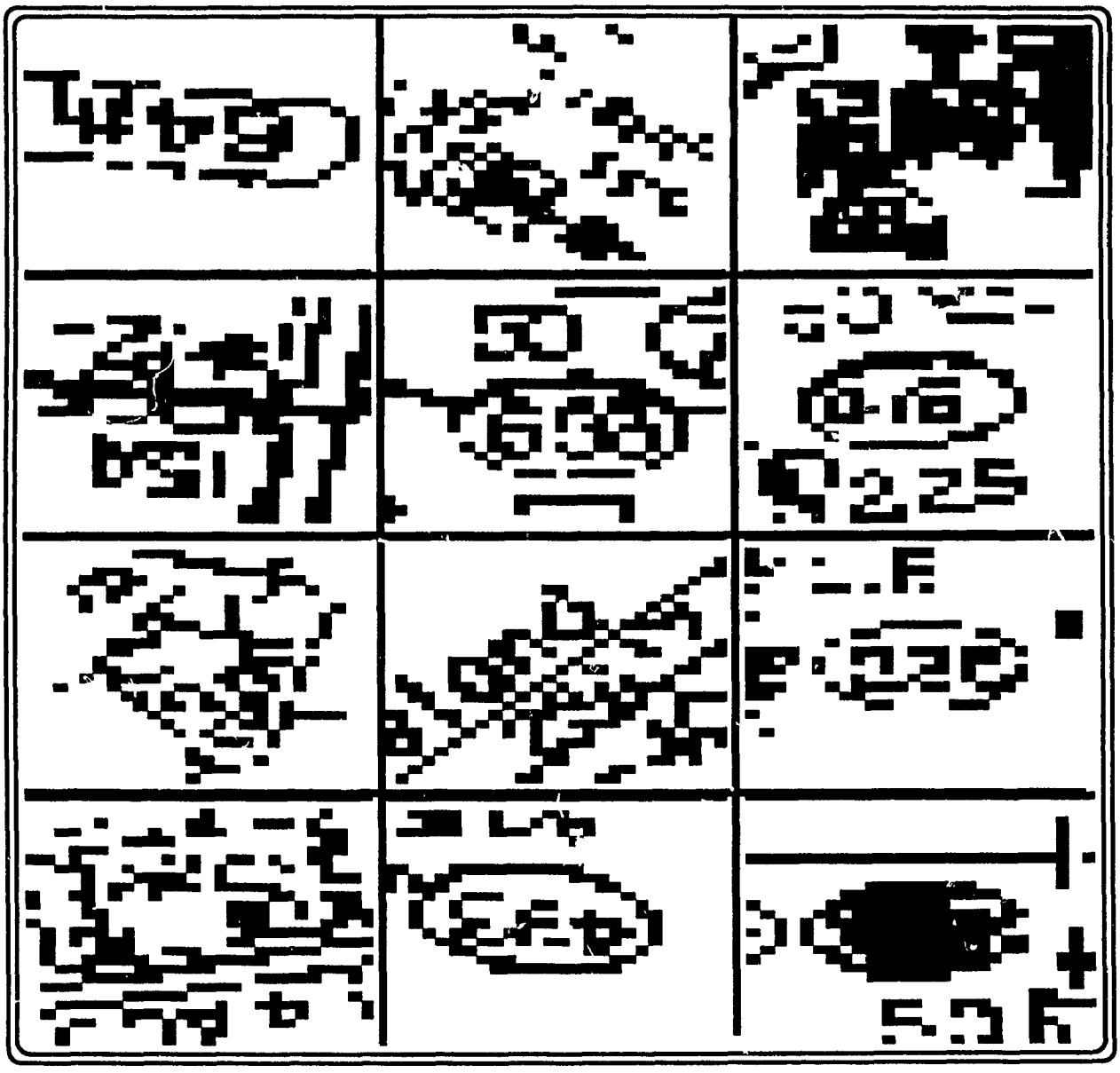


Figure 39: The instances of the Ellipse's False Positives that caused errors.

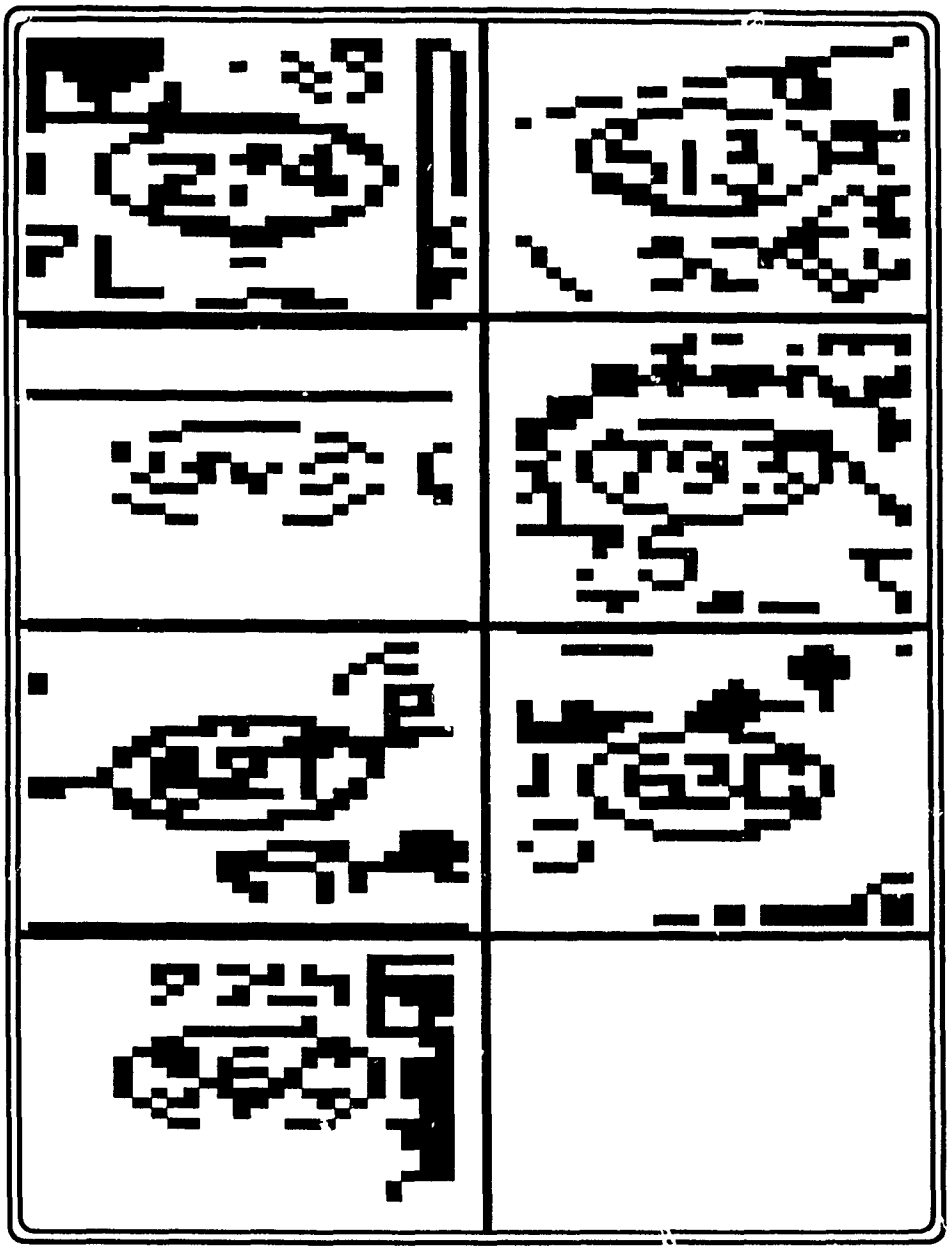


Figure 40: The instances of the Ellipse symbols that were rejected.

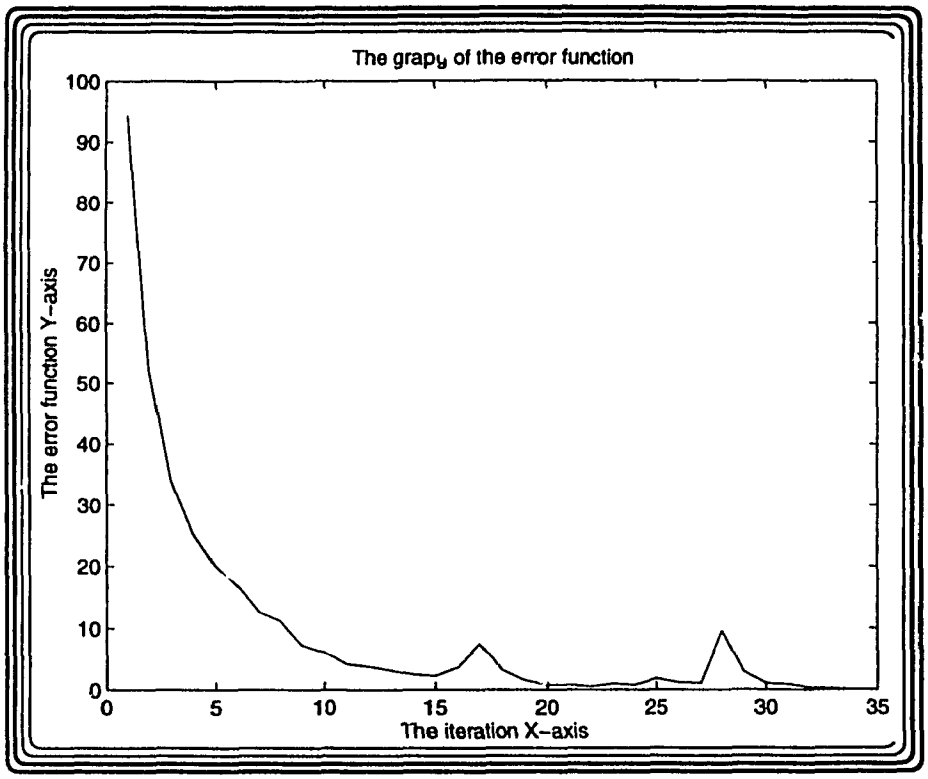


Figure 41: The error function's graph of the Transformer symbol.

	False Positive	Transformer	Rejected
False Positive	61	2	3
Transformer	20	21	10

Table 10: Performance of the Transformer NNC on the testing set.

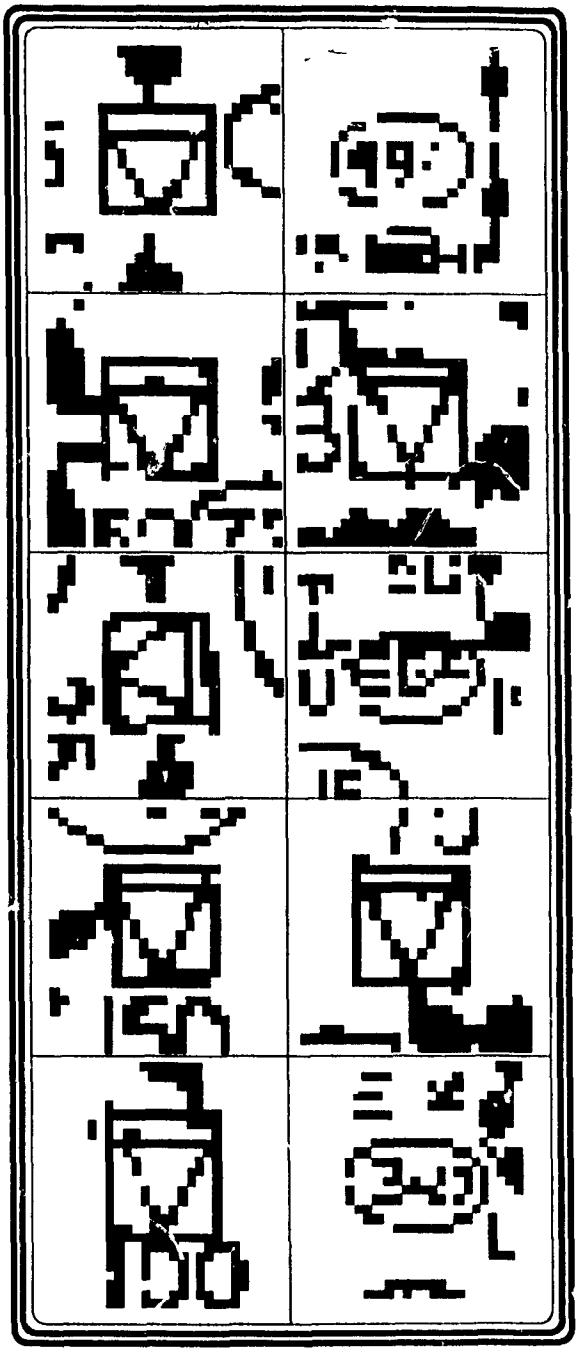


Figure 42: Some instances of the Transformer symbols present in the training set.

Table 10 shows the confusion matrix of the testing set after presenting it to the network.

Results show that the 66 False Positives (originally considered as Transformer symbols by the Hausdorff method) dropped down to 2, and that 3 of them were rejected. The rest (61) were identified correctly as being False Positives. One drawback shows in the number of Transformers (20) that were recognized as False Positives. The reason is that severe noise was present in both the training and the testing sets. The recognition rate was 70.09% and the reliability was equal to 78.85%. Figure 43 shows the Transformer's False Positives that were rejected. Although the Transformer symbol shows in all of its instances, shown in Figure 44, they were rejected because of the severe noise present.

Results of the *Filled Triangle* symbol

Training was done on a data set made up of 1077 patterns, consisting of:

- 72 patterns representing the Filled Triangle symbol, and
- 1005 patterns being False Positives.

Figure 46 shows the graph of the error function. Note that the network converged after 34 iterations, with a total error value per iteration less than 0.005 per iteration. The network took 1 hour, 49 minutes and 26 seconds to converge.
Testing was done on a data set made up of 499 patterns. The set consisted of:

- 32 patterns representing the Filled Triangle symbol, and
- 467 patterns being False Positives.

Table 11 shows the confusion matrix of the testing set after presenting it to the network. Results show that 467 False Positives (originally considered as Filled Triangle symbols by the Hausdorff method) dropped down to 5, and that 14 of them were rejected. Figure 47 shows the False Positive patterns that were misclassified. The rest (448) were identified correctly as being False Positives. We lost 19 Filled Triangles, shown in Figure 49, because they were recognized as being False Positives,

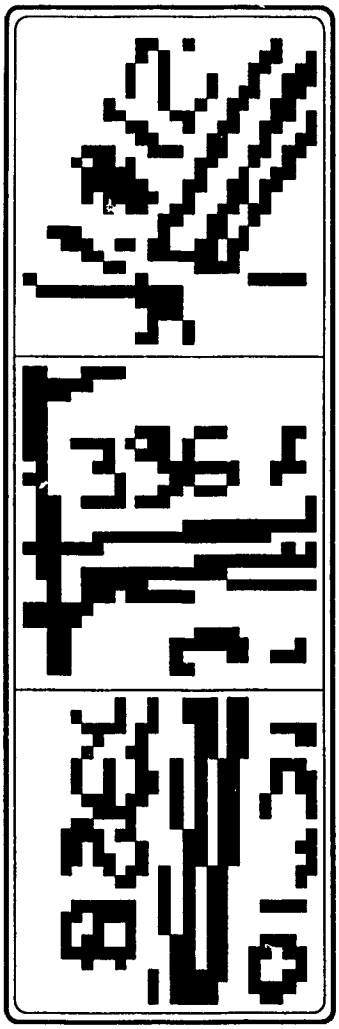


Figure 43: The instances of the Transformer's False Positives that were rejected.

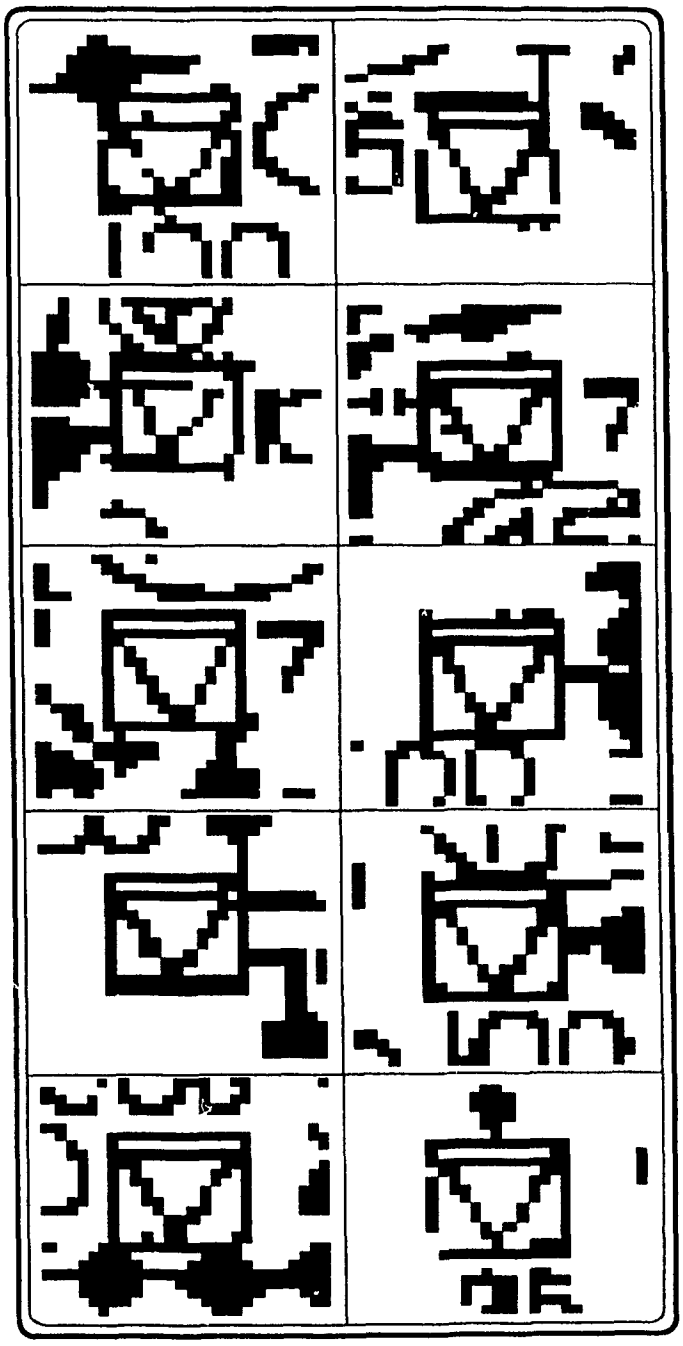


Figure 44: The instances of the Transformer symbols that were rejected.

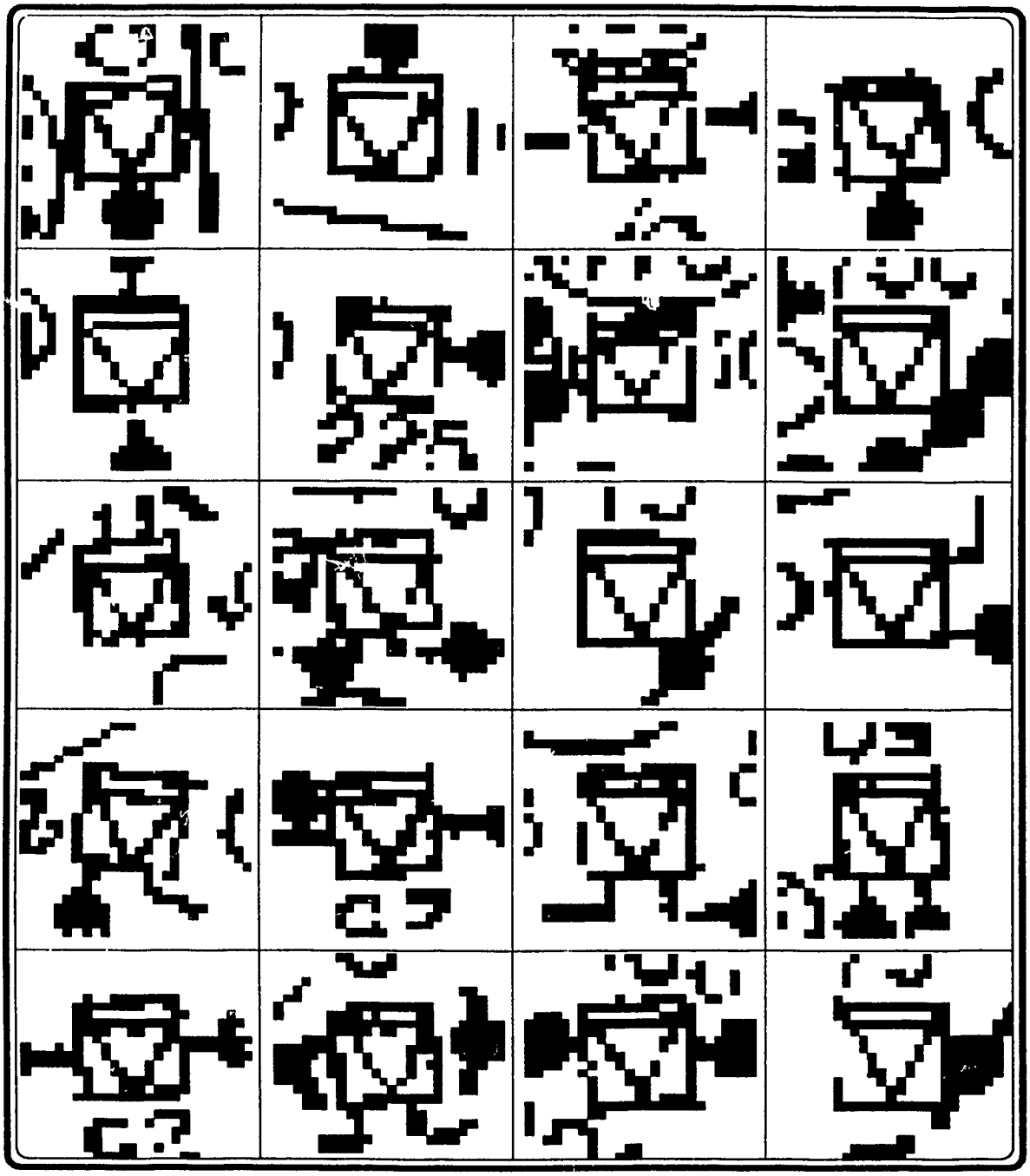


Figure 45: The instances of the Transformer symbols that caused errors.

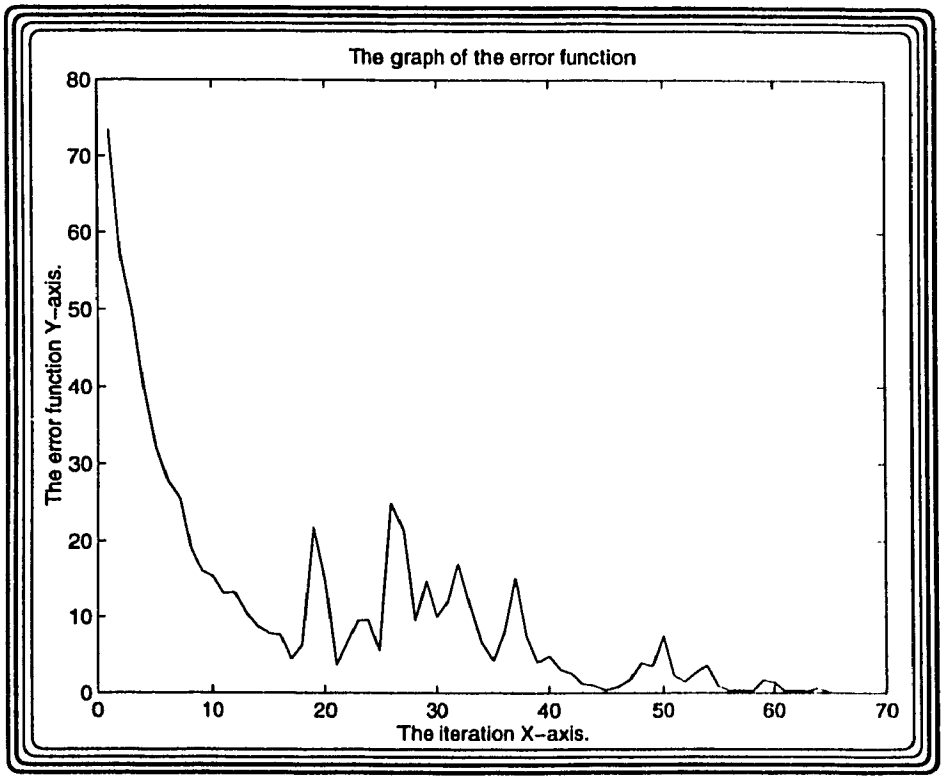


Figure 46: The error function's graph of the Filled Triangle symbol.

	False Positive	Filled Triangle	Rejected
False Positive	448	5	14
Filled Triangle	19	5	8

Table 11: Performance of the Filled Triangle NNC on the testing set.

and 8 patterns because they were rejected. Figure 48 show the patterns of the Filled Triangle that were rejected. The recognition rate was 90.78% and the reliability was equal to 94.97%.

Previous Results achieved

Table 12–(a) shows the results that were achieved when the neural networks object codes were incorporated into the main interface code (during October, 1995) at CRIM when the last voting decision was taken by the Hausdorff's system. Figure 50 shows the ideal samples of the symbols that is used by the Hausdorff algorithm.
We can see the improvements reached compared to the performance of the Hausdorff's distance method alone as shown in Table 12–(b). If we compare the number of *False Positives* as in Table 13, we notice that it has dropped down from 464 to 16 after combining the Hausdorff's method with the dynamic set of hybrid neural network classifiers, DSHNNC.

The currently modified networks were not yet incorporated and tested with Hausdorff's method at CRIM, after which the final recognition results will further improve because of the multiple expert (the neural networks and Hausdorff's) system where there are cases when the system, at CRIM, takes the last decision based on the neural networks' response and the Hausdorff's calculated parameters. This means that the DSHNNC will be used to help minimize the number of False Positives, [49]. Out of the latter presented samples of the passed symbols to the DSHNNC, as in Figures 39, 40, 47, 48, and 49, it is explicitly seen that these samples should be preprocessed and cleaned before presenting them to DSHNNC for either training or testing and which

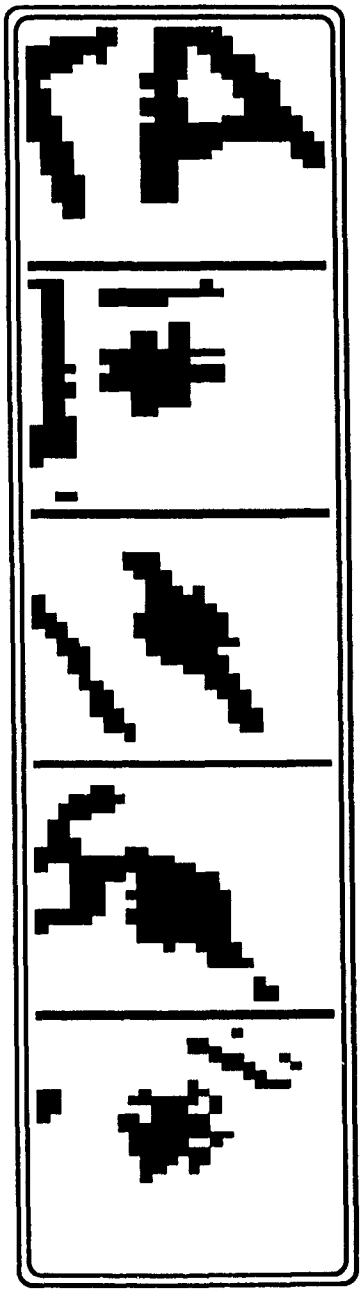


Figure 47: The instances of the Filled Triangle's False Positives that caused errors.

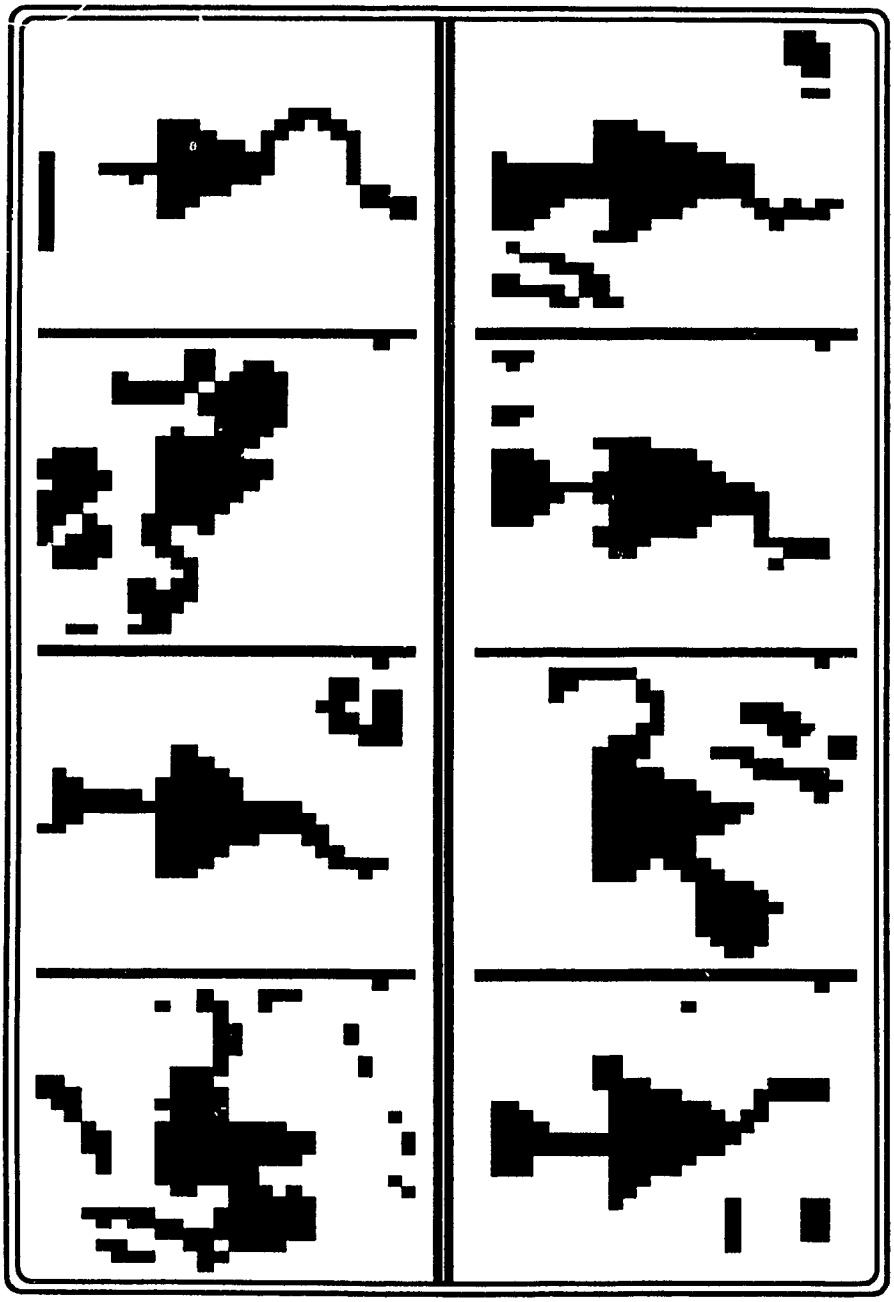


Figure 48: The instances of the Filled Triangle symbol that were rejected.

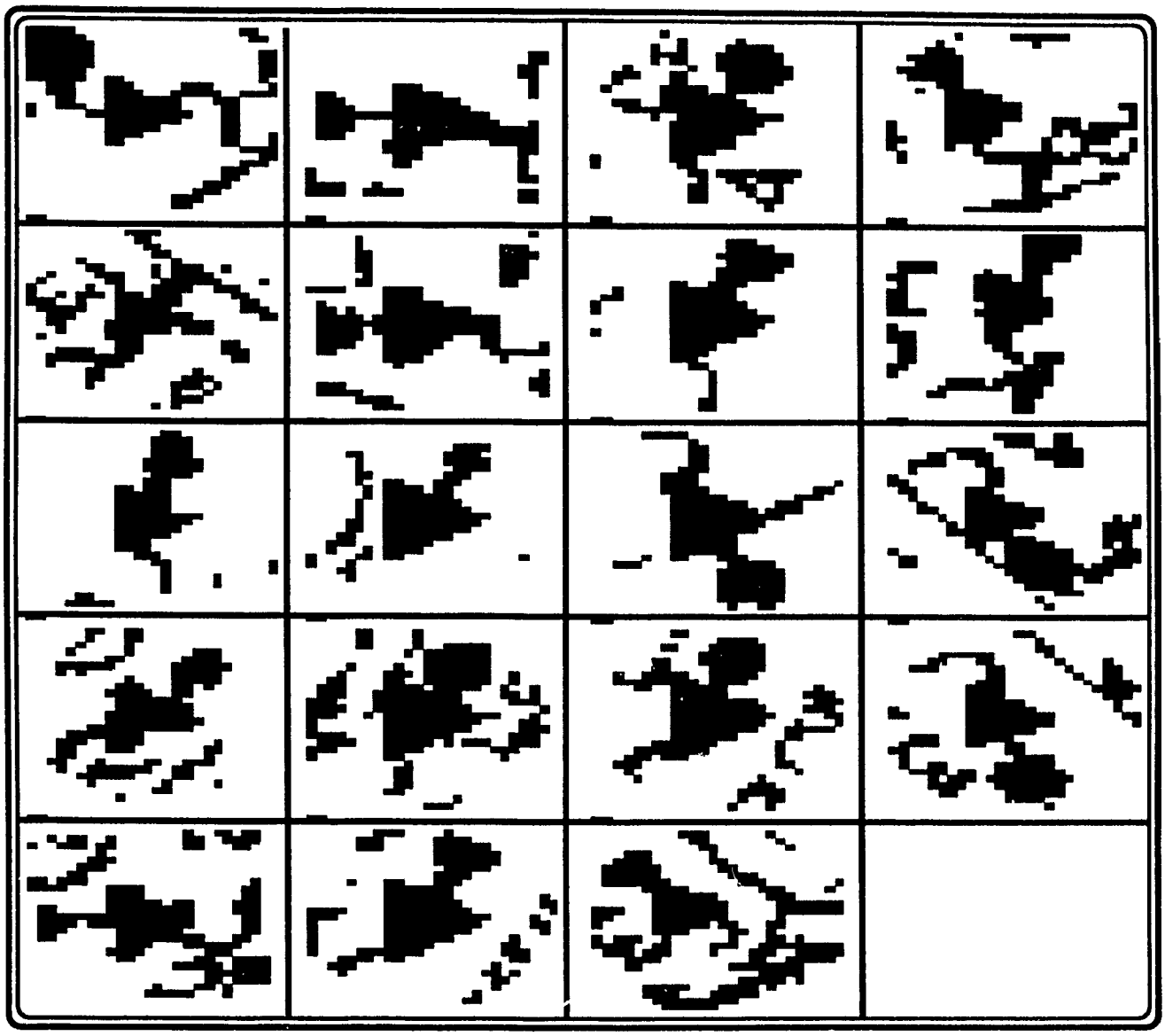


Figure 49: The instances of the Filled Triangle symbol that were misclassified.

(a) Hausdorff + DSHNNC			
Symbol	Correct	Missed	False Positive.
Cross	4	1	3
Ellipse	48	4	7
Transformer	25	4	3
Filled Triangle	15	18	3
VTransform	10	8	0
(b) Hausdorff's Method alone			
Symbol	Correct	Missed	False Positive.
Cross	5	0	4
Ellipse	50	2	162
Transformer	25	4	86
Filled Triangle	32	1	178
VTransform	18	0	34

Table 12: Combining Hausdorff's algorithm with DSHNNC.

The False Positives table		
Symbol	Hausdorff + DSHNNC	Hausdorff
Cross	3	4
Ellipse	7	162
Transformer	3	86
Filled Triangle	3	178
VTransform	0	34
Total	16	464

Table 13: Improvements to the number of False Positives.

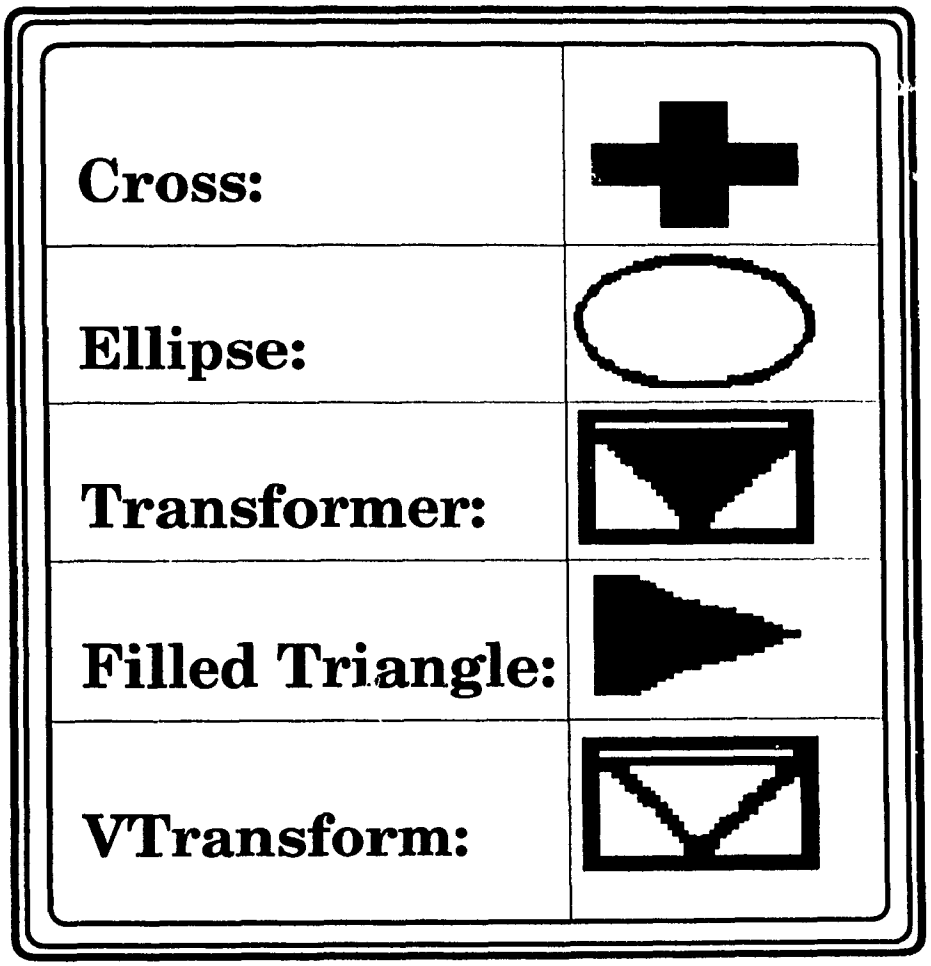


Figure 50: An ideal sample of the 5 symbols used.

stands as a future direction.

Chapter 6

Conclusion

6.1 Major accomplishments

6.1.1 The neighborhood of the optimal number of hidden neurons

In conclusion, the method proposed here is a *fast technique* for finding the neighborhood of the optimal number of hidden neurons in a hidden layer. After this stage, if finding the optimal number is the designer's aim, then only one process is left to be done, which is simply running the same proposed method with the new upper and lower bounds of the neighborhood, and setting the number of neurons to be added or removed to 1 neuron only. This will eventually lead to the optimal number of hidden neurons.

At the beginning, the idea of finding an optimal number of hidden neurons was perceived for its relational dependency with the way the human brain responds to tasks of different complexity levels. Later, after many experimental results, a neighborhood of the optimal number of hidden neurons was explicitly detected, and a fast method of finding it is proposed while attaining convergence to the network being trained.

To sum up, this method proves to be reliable with the modified error backpropagation algorithm. It can be embedded in any other similar algorithms. Thus as a

future directive, we seek to generate some formulas that govern this criterion. We also intend to extend our work to study a way to deal with addition and deletion of one (or more) hidden layer to a network, and its effect on the curvature of the error function and on the general performance of the network.

6.1.2 Segmentation of paragraphs of text

A combination of the vertical and horizontal histograms and the connected components segmentation algorithms is presented as a trial to quickly segment paragraphs of texts, extracted from engineering drawings and maps.

6.1.3 Building a dynamic set of hybrid neural network classifiers

A method of building a dynamic set of hybrid neural network classifiers, DSHNNC, is presented. Adding a new pattern to be classified is fairly easy. All we have to do is to train its respective newly created NNC and add the NNC to the DSHNNC architecture. As for the case when deciding to remove a pattern, not willing to recognize it anymore, then only the NNC associated with that pattern will only be removed form the DSHNNC architecture. In this way, the training process over the data sets of the patterns is completely pattern dependent. In other words, training is done for each pattern's NNC separately.

6.2 Minor drawbacks

6.2.1 Weights and biases

The idea of having a relation between the number of neurons in a given layer and the initialization of its respective weight and bias matrices remains as a future work for us to accomplish, or for other researchers to pursue.

6.2.2 Towards the optimal number of hidden neurons

Even though we were able to find some relations and map them into conditions (below the state of a formula), yet we still believe that we must do some future research on this promising criterion in order to reach some formulas and prove their consistency and generality not only with the error backpropagation algorithm, but also with the other networks that have hidden layers.

6.2.3 Segmentation of paragraphs of text

The current segmentation method proves to be fast and efficient with untouching characters, however it becomes less efficient when it encounters touching characters. More research as to segmenting touching characters on the same line or from two consecutive lines is to be employed as a future direction.

6.2.4 Map symbol recognition

While examining the performance of the dynamic sets of the hybrid neural network classifiers, DSHNNC, we suffered a lot from two problems. First, the instances of the symbols were full of noise. This led to having unclean sets of databases for training and testing, which is the second problem faced. Thus, new types of preprocessing techniques are to be sought for such types of noise.

References

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, *Image coding using wavelet transform*, **IEEE Transactions on Image Processing**, Vol. 1, pp. 205-220, 1992.

[2] D. Ballard, *Generalizing the Hough transform to detect arbitrary shapes*, **Pattern Recognition**, Vol. 13, No. 1, pp. 111-122, 1981.

[3] E. Cohen, J.J. Hull, and S.N. Srihari, *Control Structure for Interpreting Handwritten Addresses*, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 16, No. 2, pp.1049-1055, October 1994.

[4] C. Dachet, *Automatic reading of technical drawings: symbol extraction by geometrical and morphological methods*, **Proceedings of Vision Interface'94**, pp 127-136, Banff Alberta, May 18-20, 1994.

[5] H. Demuth and M. Beale, **Neural network tool box user's guide for Matlab**, The Math Works Inc, 1992.

[6] G. Dreyfus, *Neural networks for the automatic recognition of handwritten digits*, **Applications of Neural Networks**, edited by H.G. Schuster, Vol. 3, p. 35-59, Fed. Republic of Germany, 1992.

[7] S. Dreyfus, *The numerical solution of variational problems*, **Mathematical Analysis and Applications**, Vol. 5, No. 1, pp. 30-45, 1962.

[8] S. Dreyfus, *Artificial neural networks, back propagation and the Kelly-Bryson gradient procedure*, **Journal of Guidance and Control Dynamics**, Vol. 13, No. 5, pp. 926-928, 1990.

[9] J.A. Freeman and D.M. Skapura, **Neural networks: algorithms, applications, and programming techniques**, Addison-Wesley Inc., New York, 1992.

[10] K. Fukushima, *Analysis of the process of visual pattern recognition by the neocognitron*, **Neural Networks**, Vol. 2, No. 6, pp. 413-421, 1989.

[11] K. Fukushima and T. Imagawa, *Recognition and segmentation of connected characters with selective attention*, **Neural Networks**, Vol. 6, No. 1, pp. 33-45, 1993.

[12] K. Fukushima and S. Miyaka, *Neucognitron: A self-organizing neural Nnt-work model for a mechanism of pattern recognition unaffected by shift in position*, **Biological Cybernetics**, Vol. 36, No. 4, pp. 193-202, 1980.

[13] K. Fukushima, M. Okada, and K. Kiroshige, *Neucognitron with dual C-Cell layers*, **Neural Networks**, Vol. 7, No. 1, pp. 41-53, 1994.

[14] G. Mirchandani, and W. Cao, *On hidden Nodes for neural nets*, **IEEE Transactions on Circuits and Systems**, vol. 36, No. 5, pp 661-664, May 1989.

[15] G.M. Georgiou, *Comments on 'On hidden nodes in neural nets'* , **IEEE Transactions on Circuits and Systems**, Vol. 38, No. 11, pp 1410, November 1991.

[16] R.P. Groman and T.J. Sejnowski, *Analysis of hidden units in a layered network trained to classify sonar targets*, **Neural Networks**, vol. 1, pp75-89, 1988.

[17] S. Grossberg, *Classical and intrumental learning by neural networks*, **Progress in Theoretical Biology**, vol. 3, pp. 51-141, Academic press, New York, 1974.

[18] S. Grossberg, **Studies of mind and brain: neural principles of learning perception, development, cognition, and motor control**, Reidel press, Boston, 1982.

[19] T.M. Ha and H. Bunke, *Handwritten numeral recognition by perturbation method*, **Proceedings of the fourth International Workshop on Frontiers in Handwriting Recognition**, pp. 97-106, December 1994.

[20] D.O. Hebb, **The organization of behaviour, a neuropsychological theory**, Wiley, New York, 1949.

[21] D. Heller and P.M. Ferguson, **Motif programming manual**, O'Reilly and Associates Inc, Second edition, 1994.

[22] F. Herget, W. Finnoff and H.G. Zimmerman, *A comparison of weight elimination methods for reducing complexity in neural networks*, **International Joint Conference on Neural Networks**, Vol. 3, pp 980-987, Baltimore, Maryland, June 7-11, 1992.

[23] G. Hinton, *Connectionist learning procedures*, **Artficial Intelligence**, Vol. 40, pp. 185-234, 1989.

[24] G. Hinton, *How neural networks learn from experience*, **Scientific American**, pp. 144-151, September, 1992.

[25] G. Hinton, **Neural networks for the industry**, Information Technology Research Center and PRECARN Associates Inc., University of Toronto, Toronto, February 19-20, 1996.

[26] J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, **Proceedings of National Academic Science**, Vol. 79, pp. 2554-2558, 1982.

[27] J. Hopfield, *Neurons with graded response have collective computational properties like those of two state neurons*, **Proceedings of National Academic Science**, Vol. 81, pp. 3088-3092, 1984.

[28] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge, *Comparing Images Using the Hausdorff Distance*, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 15, No. 9, pp. 850-863, September 1993.

[29] N. Indukhya and S.M. Weiss, *Heuristic configuration of hidden units in Back-propagation Neural Networks*, **International Joint Conference on Neural Networks**, Vol. 2, pp A-961, Seattle, WA, July 8-12, 1991.

[30] T. Kohonen, **Associative memory: a system-theoretical approach**, Springer-Verlag, Berlin, 1977.

[31] T. Kohonen, *A simple paradigm for the self-organized formation of structured feature maps*, in S. Amari and M. Arbib (eds.), **Competition and cCooperation in Neural Networks**, Vol. 45, Springer-Verlag, Berlin, 1982.

[32] T. Kohonen, **Self-organization and associative memory**, Springer-Verlag, Berlin, 1984.

[33] T. Kohonen, *The 'neural' phoneic typewriter*, **IEEE Computer**, Vol. 27, No. 3, pp. 11-22, 1988.

[34] A. Krzyzak, W. Dai, and C.Y. Suen, *Unconstrained Handwritten Character Classification Using Modified Back propagation Model*, in C.Y. Suen (ed.), **Frontiers in Handwriting Recognition**, p. 145-151, CENPARMI, Concordia University, 1990.

[35] S.Y. Kung, **Digital neural networks**, PTR Prentice-Hall Inc, First edition, 1993.

[36] L. Lam and C.Y. Suen, *A theoratical analysis of the application of majority voting to pattern recognition*, bf Proceedings of the Twelfth International Conference on Pattern Recognition, Vol. II, pp. 418-420, 1994.

[37] L. Lam and C.Y. Suen, *Optimal combinations of pattern classifiers*, **Pattern Recognition Letters**, No. 16, pp. 945-956, 1994.

[38] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, *Backpropagation applied to handwritten zipcode recognition*, **Neural Computation**, Vol. 1, No. 4, pp. 541-551, 1989.

[39] Y. Le Cun, **Generalization and network design strategies**, Technical report CRG-TR-89-4, University of Toronto, Toronto, 1989.

[40] Y. Le Cun, B. Boser, D. Henderson, and R.E. Howard, *Constrained neural network for unconstrained handwritten digit recognition*, in C.Y. Suen (ed), **Frontiers in Handwriting Recognition**, p. 145-151, CENPARMI, Concordia University, 1990.

[41] D.S. Lee and Sargur N. Srihari, *Handprinted digit recognition: A comparison of algorithms*, **Proceedings of the third International Workshop on Frontiers in Handwriting Recognition**, pp. 153-162, May 1993.

[42] M. Levine, **Vision in Man and Machine**, McGraw Hill, Inc., New York, 1982.

[43] B. Li, F.N. Said, and C.Y. Suen, *An automated method for extracting handwritten data from payment slips*, **Fourth Annual IRIS-PRECARN Conference**, pp 136-137, Toronto, Canada, June 21-23, 1994.

[44] K. Lui,, Y.S. Huang, and C.Y. Suen, *Image classification by classifier combining technique*, **Proceedings of the SPIE Conference on Neural and Stochastic Methods in Image and Signal Processing III**, pp. 338-346, 1994.

[45] Y. Lu, *Machine printed character segmentation - an overview*, **Pattern Recognition**, Vol. 28, No. 1, pp 67-80, 1995.

[46] M. de la Mazar, *SPLIT net: dynamically adjusting the number of hidden units in a neural network*, **Artificial Neural Networks**, edited by T. Kohonen, K. Makisara, O. Simula, and J. Kangas, vol. 1, pp 647-651, 1991.

[47] M. Minsky and S. Papert, **Perceptrons: an introduction to computational geometry**, MIT press, 1969.

[48] W.S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, **Bulletin of Mathematical Biophysics**, Vol. 5, pp. 115-133, 1943.

[49] E. Reiher, F.N. Said, Y. Li, and C.Y. Suen, *Map symbol recognition using directed Hausdorff distance and a neural network classifier*, to appear in the **Proceedings of the 18th ISPRS Congress**, Vienna, July 9-19, 1996.

[50] F. Rosenblatt, **The perceptron: a theory of statistical separability in cognitive systems**, Cornell aeronautical laboratory inc., Technical report No. VG-1196-G-1, 1958.

[51] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by backpropagating errors, **Nature**, Vol. 323, pp. 533-536, 1986.

[52] F.N. Said, D. Asselin, and C.Y. Suen, *Recognition of alphanumeric characters on engineering drawings*, **Fifth Annual IRIS-PRECARN Conference**, pp. 78-79, Vancouver Canada, June 13-15, 1995.

[53] N.W. Strathy, **A method for segmentation of touching handwritten numerals**, Concordia University, Montréal, Québec, September 1993.

[54] C.Y. Suen, R. Legault, C. Nadal, M. Cheriet, and L. Lam, *Building a new generation of handwriting recognition systems*, **Pattern Recognition Letters**, Vol. 14, No. 4, April 1993.

[55] C.Y. Suen, C. Nadal, T.A. Mai, R. Legault, and L. Lam, *Computer recognition of unconstrained handwritten numerals*, bf Proceedings of the IEEE, Vol. 80, No. 7, pp. 1162-1180, 1992.

[56] L.G. Valiant, **Circuits of the mind**, Oxford University press, New York, 1994.

[57] D. Wang and S. Srihari, *Analysis of form images*, **Proceedings of the First International Conference on Document Analysis and Recognition**, pp 181-191, 1991.

[58] P.S.P. Wang, **Character and handwriting recognition, expanding frontiers**, World scientific publishing Co., 1991.

[59] W.X. Wein, H. Liu, A. Jennings, *Self-Generating neural networks*, **International Joint Conference on Neural Networks**, Vol. 4, pp 850-855, Baltimore Maryland, June 7-11, 1992.

[60] B. Widrow and M. E. Hoff, *Adaptive switching sircuits*, **1960 IRE Western Electric Show and Convention Record**, No. IV, pp. 96-104, August 23, 1960.

[61] L. Xu, A. Krzyzak, and C.Y. Suen, *Methods of combining multiple classifiers and their applications to handwriting recognition*, **IEEE transactions on Systems, Man, and Cybernetics**, Vol. 22, No. 3, pp. 418-435, 1992.

[62] Y.H. Yu, A. Samal, and S. Seth, *Isolating symbols from connection lines in a class of engineering drawings*, **Pattern Recognition**, Vol. 27, No. 1, pp. 391-404, 1994.

[63] J.M. Zurada, **Introduction to artificial neural systems**, West publishing company, 1992.

Appendix A

Current applications based on backpropagation

To sense the degree of importance of the backpropagation algorithm, this Appendix presents a summarized survey of some selected neural networks software systems, found in the literature. In addition, this Appendix helps in broadening the view of the reader to a wide variety of different applications where neural networks techniques and algorithms can be used to solve problems.

Please note that the companies listed in this Appendix are not the sole companies that produced application using the backpropagation algorithm. However, they are some of those active *software* companies that are recognized by the computer magazines.

BuildNet, a software package of **Adaptive Solutions**[1], allows the user create custom products using backpropagation. It offers a command-line and Motif style interface, as well as an interface C library functions.

Propagator, an application developed by **ARD CORP.**,[2] is a backpropagation neural network development system with a graphical interface. It features speed, simplicity, training with validation, optional input noise, three dynamic graphs, up to 5 layers and 32,000 nodes per layer, data scaling, and C/C++ source code generation.

[1] Adaptive Solutions a software company in Beaverton, Ore. 97006, (800) 48-CNAPS

[2] ARD CORP., a software company in Columbia, Md 21045, (800) 969-2731

It comes with a free technical support and money-back guarantee is included. It can run under Sun, Windows and Macintosh.

Guffy software[3] developed the *BP++* which is a C++ neural network library for constructing backpropagation neural networks. It includes the source code for. BPWin is a Windows package for interactively creating and training backpropagation neural networks that can be saved and later embedded using BP++.

NnetLib 'C' Programmers Library, developed by **Lever Software Systems**[4], contains functions and data structures that implement neural networks specifically for C programmers. It currently supports backpropagation, Hopfield, Kohonen, and counterpropagation. It runs under Windows and DOS.

havBpNet++, developed by **hav.Software**[5], is an embeddable feedforward and recurrent backpropagation C++ class library. It includes double precision with special scaling for high resolution data and large dynamic range. The source code is available. It is available for DOS, Windows (QW and DLL), and UNIX (PC, SUN, HP, RS6000, and SGI) platforms.

Neural Network Toolbox 2.0, developed by **the MathWorks Inc**[6], is a collection of MATLAB functions for designing and simulating neural networks. The toolbox includes learning rules, transfer functions, and training and design procedures for the perceptron learning rule, the Widrow-Hoff rule, and variations of backpropagation. It also includes unsupervised training functions that use associative learning rules for competitive layers, feature map layers, and Hopfield networks. The Toolbox integrates with MATLAB, a numeric computing visualization environment, and runs under various platforms.

NeuroDimension Inc.[7] developed *NeuroSolutions* which is a Windows based neural network simulation environment that supports static, fixed-point, and trajectory learning through backpropagation, recurrent backpropagation, and backpropagation through time. It provides the flexibility needed to construct a wide range of

[3]Guffy Software a software company in Rochester, N.Y. 14624, (716) 594-2836.
[4]Lever Software Systems a software company in Utica, N.Y. 13501, (800) 638-7250.
[5]hav.Software a software company located in Richmond, Texas 77406-0354, (713) 341-5035.
[6]The MathWorks Inc. a software company located in Natick, Mass. 01760, (508) 653-1415.
[7]NeuroDimension Inc. a software company in Pittsburgh, Pa. 15276, (412) 338-6779.

learning paradigms and network topologies.

Appendix B

The User Interface

A user interface was built using the C programming language and under the Motif/UMIX environment. Figures 51 and 52 show samples of the output generated by the system. Figure 51 shows the window in which the confusion matrix shows the recognition results of a certain testing experiment, and Figure 52 shows the map being processed.

The system has other utilities like:

- changing the initial values of the parameters used in the training phase, Figure 53
- training the network as shown in Figure 54
- testing the performance of the network, Figure 55
- processing engineering drawings, Figure 56, and
- adding/removing symbols from the DSHNNC, payment slips, etc.

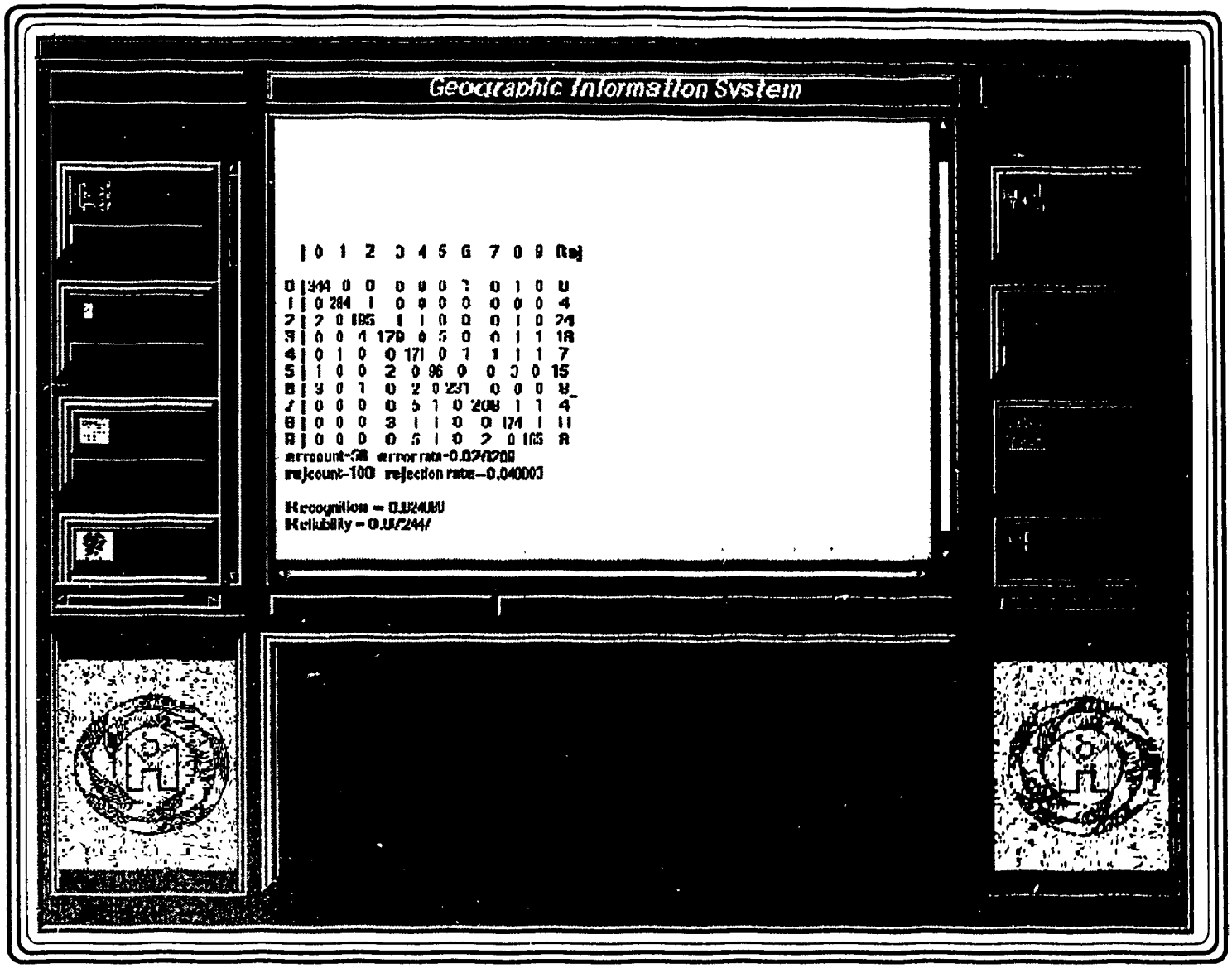


Figure 51: The confusion matrix as shown by the user interface.

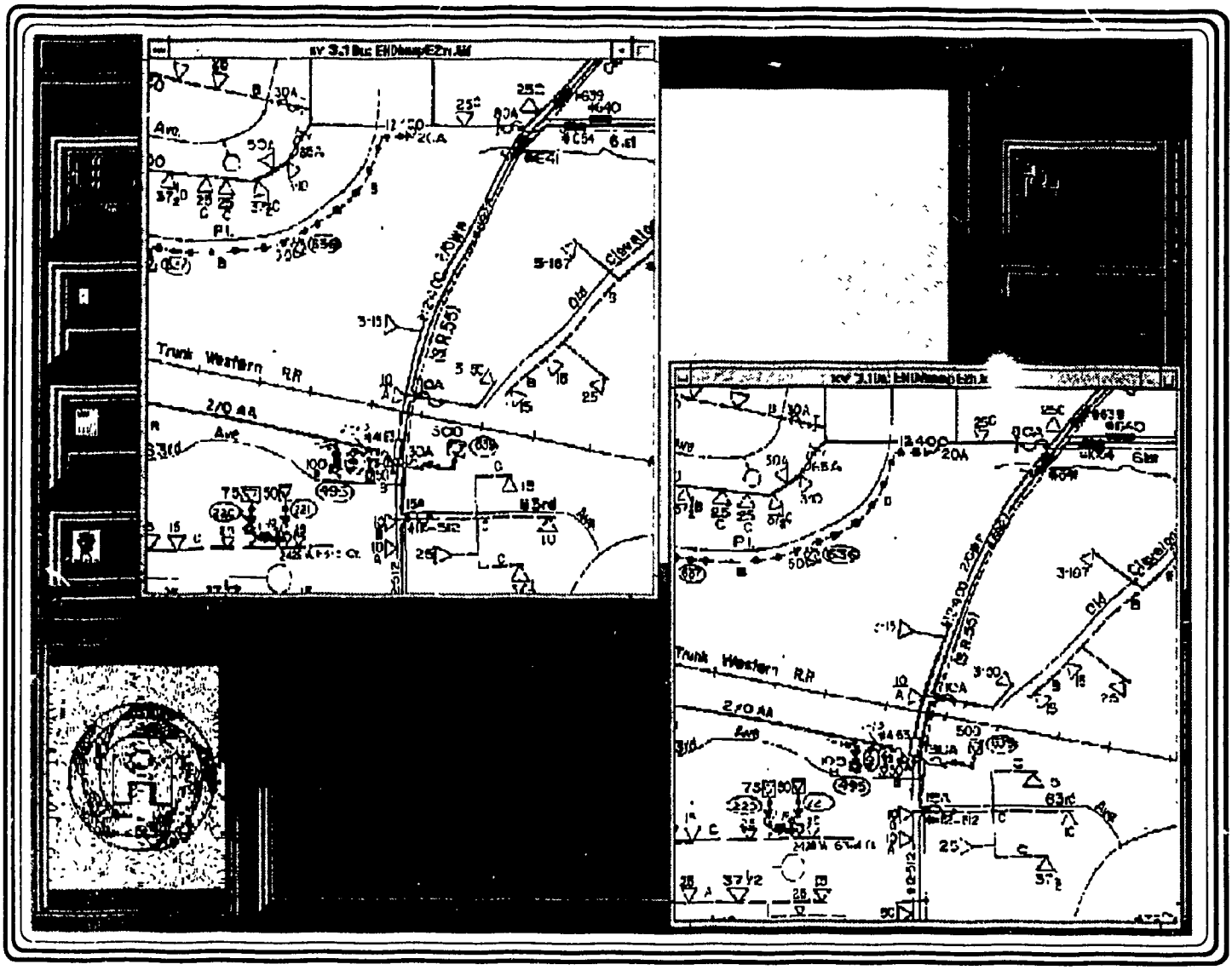


Figure 52: The input and output maps as shown by the user interface.

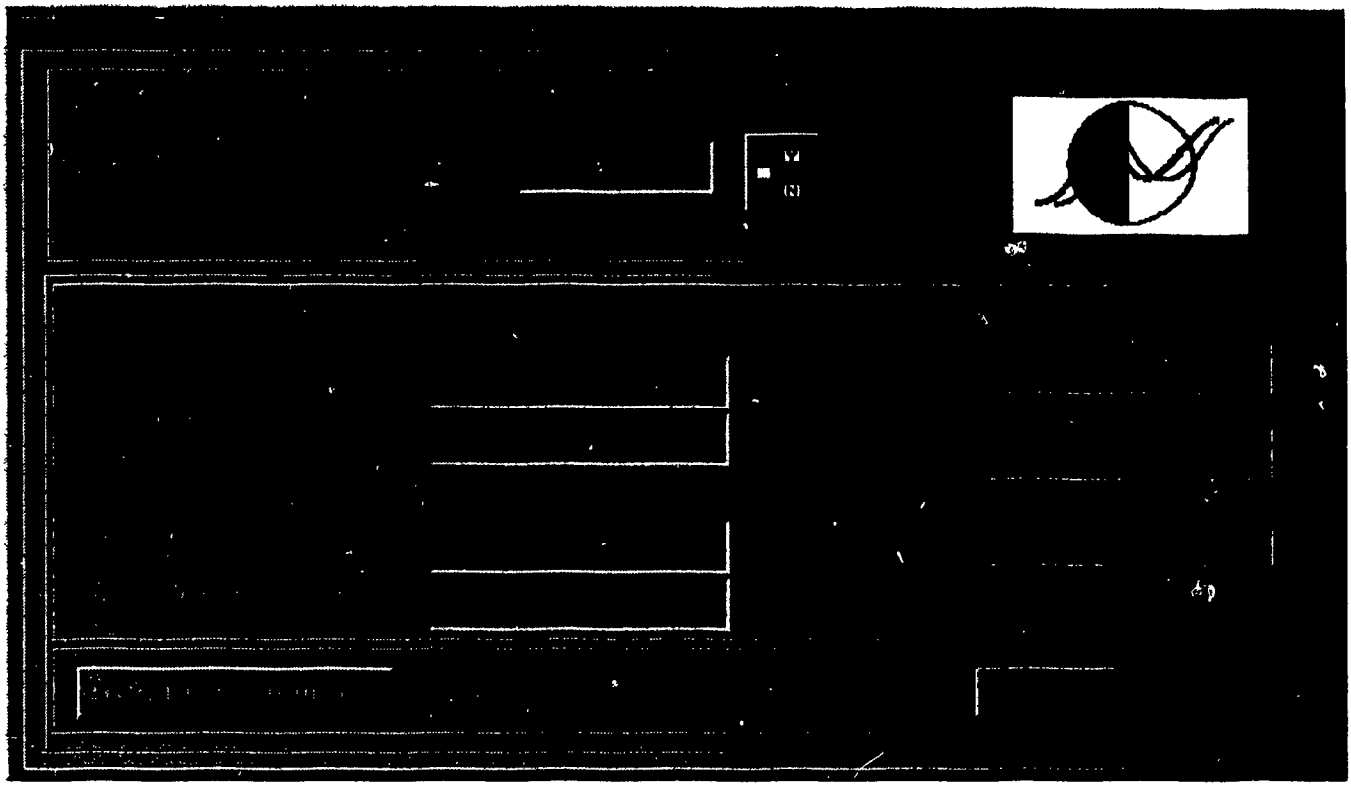


Figure 53: The interface allows the user to initialize the values of the parameters and the network's architecture.

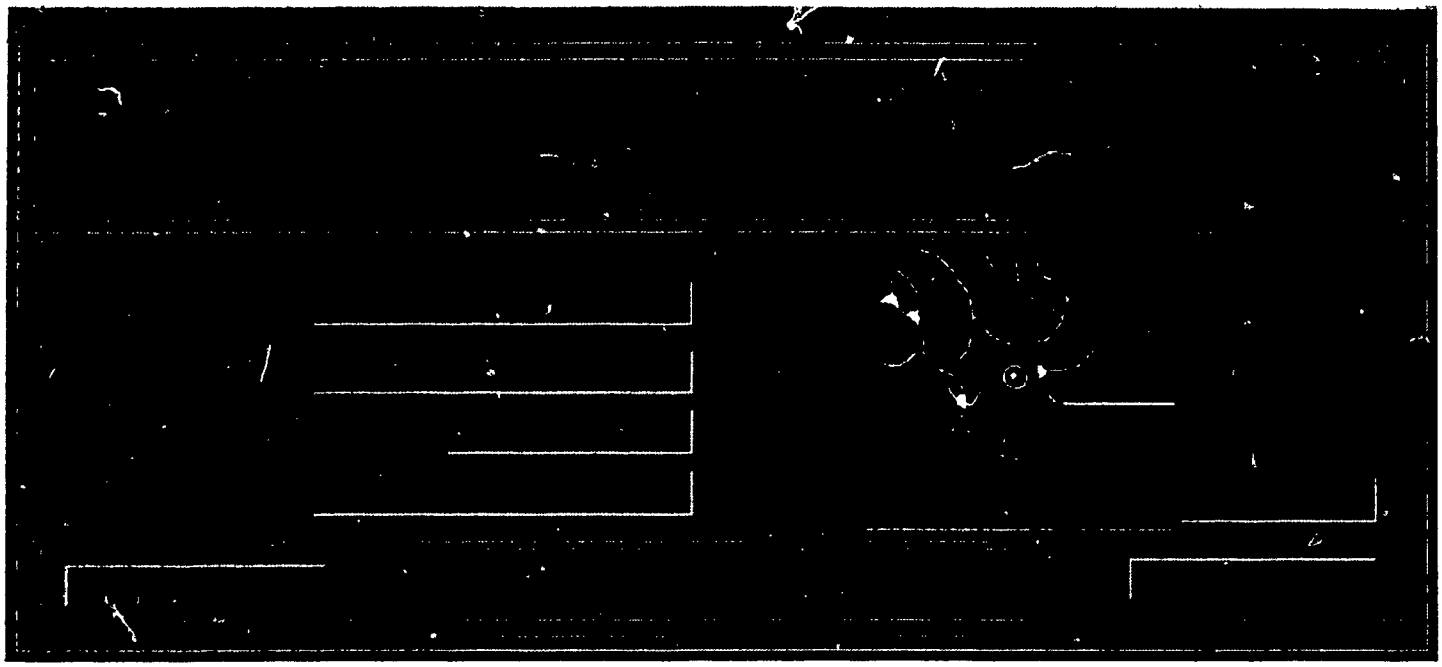


Figure 54: The pop up window of the training phase.

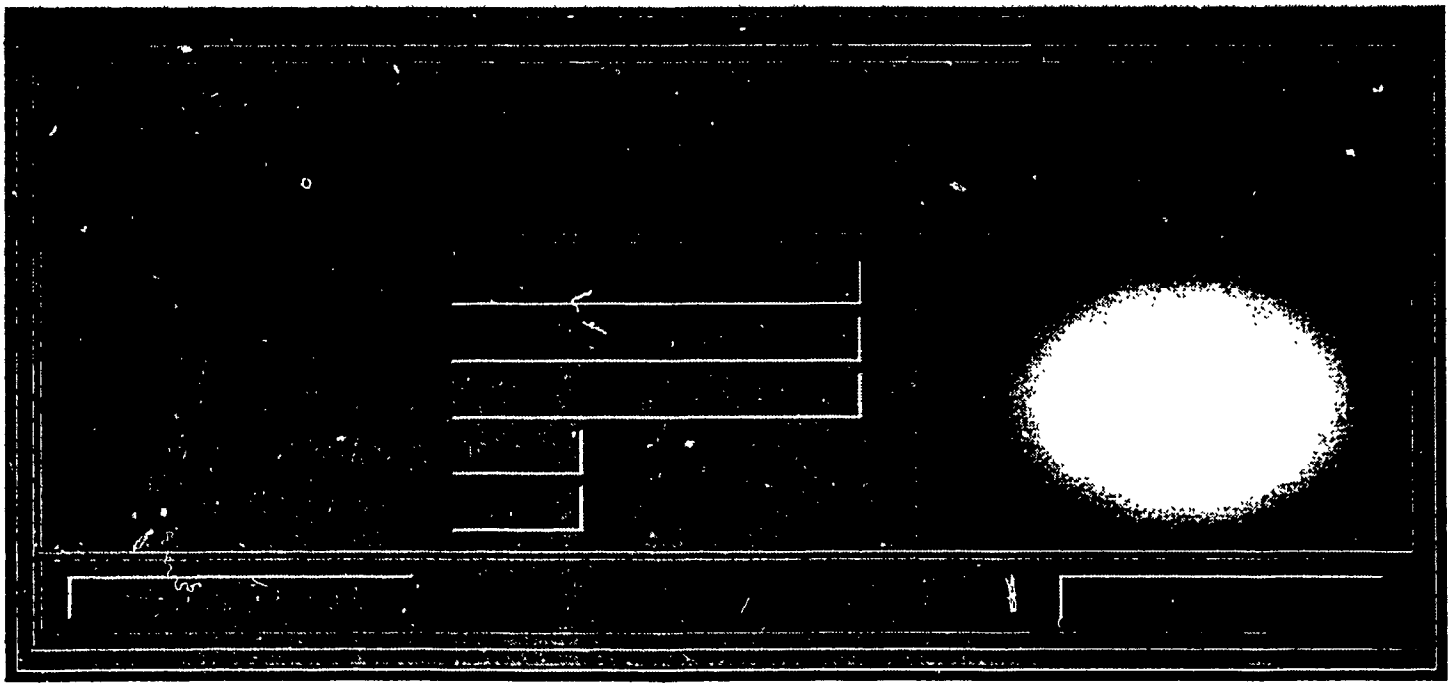


Figure 55: The pop up window of the testing phase.

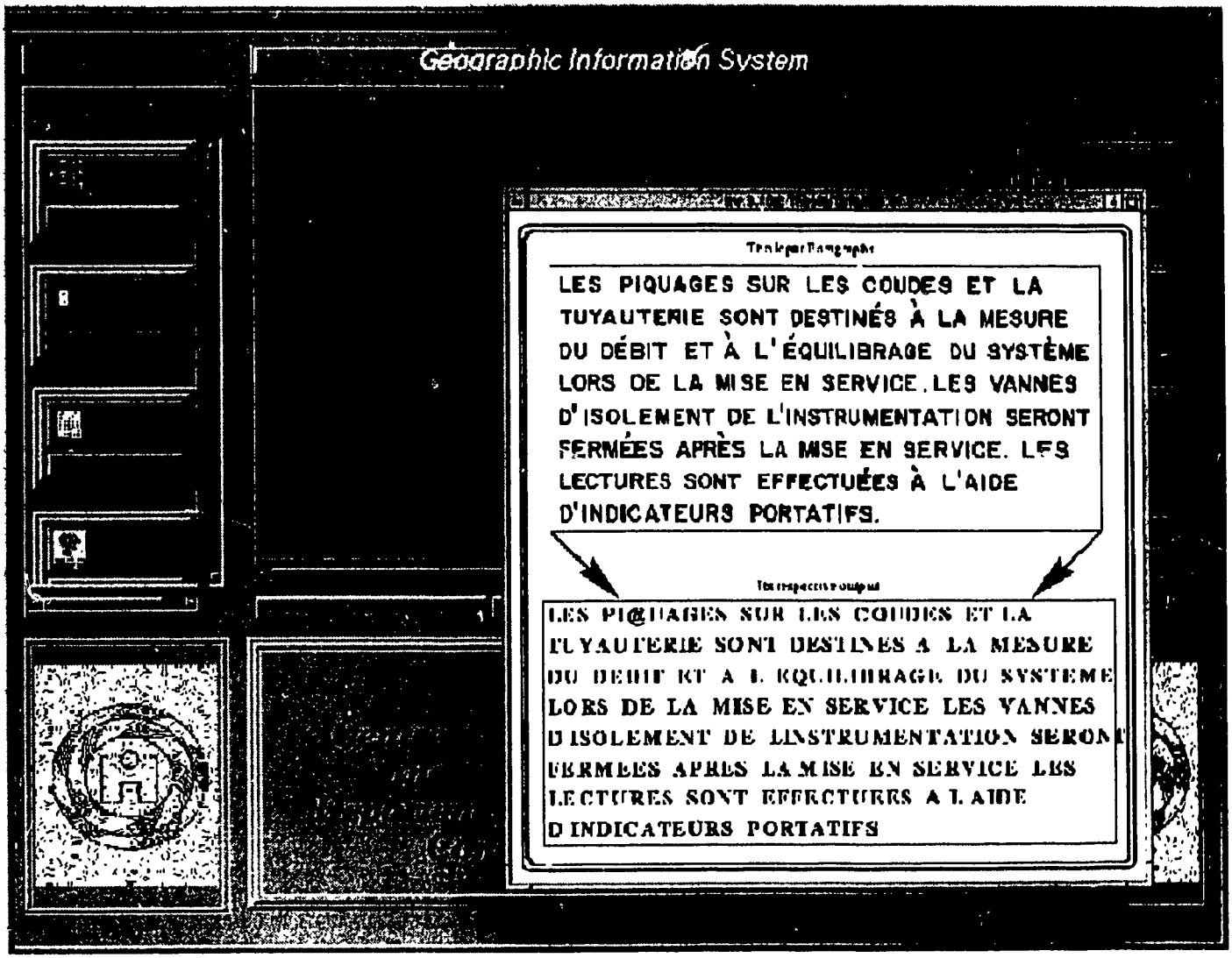


Figure 56: The input and output paragraphs as shown by the user interface.

Index

Mai, T.A., 101

accumulated knowledge, 11
Acknowledgements, v
adaptive resonance, 6
Anderson, J., 6
Antonini, M., 97
architecture of NNC, 66
Asselin, D., v, 100
associative memory, 6

Ballard, D., 97
Bouchard, A., v
brain, 9–11, 94
Bryson, A.E., 7
Bunke, H., 98

Cao, W., 98
CEDAR database, 30, 47, 48, 51
CENPARMI, v, vi, 99, 100
Cheriet, M., 101
claim, 10
Cohen, E., 97
confusion matrix, 50, 69, 73, 82
CRIM, v, 3, 87
CRIM database, 47

Dachet, C., 97
de la Mazar, M., 100
Dedication, vii
Demuth, H., 97
dendrites, 9
document analysis, 1
Dreyfus, G., 97
Dreyfus, S., 7
DSHNNC, 87, 91, 96

engineering drawings, iii, 105

False Positive, 66
Freeman, J.A., 97
Fukushima, K., 6, 97

geographic information system, iii
Georgiou, G.M., 98
Grossberg, S., 6, 98
Guillemette, M., v

Ha, T.M., 98
Harb, J., v
Hausdorff's method, 31, 66
Hebb, D., 6
Hebb, D.O., 98
Heller, D., 98
Herget, F., 98
hidden neurons, 26

Hinton, G., 98
Ho, Y.C., 7
Hoff, M., 6
Hoff, M.E., 101
Hopfield, J., 7, 99
Huang, Y.S., 100

Indukhya, N., 99
IRIS, v

Kasvand, T., ii
Kohonen, T., 6, 99
Krzyzak, A., ii, 99, 101
Kung, S.Y., 99

Lalonde, M., v
Lam, L., 99, 101
Le Cun, Y., 99
Lee, D.S., 30, 100
Legault, R., 101
Li, B., 100
Li, Y., v
line segmentation, 32
Lui, K., 100

machine used, 47
map symbols segmentation, 39
maps, iii, 105
Martin, G., ii
McCulloch, W.S., 6, 100
mind, 9, 11
Minsky, M., 6, 100
Mirchandani, G., 98

Nadal, C., 101
neocognitron, 6
neural network classifier, 3
neural network classifiers, iii
neurocortex, 9, 10
neurons, 5, 9, 13, 21, 24
neurons interconnections, 10
NSERC, v

Papert, S., 6
pattern normalization, 36
pattern segmentation, 32
Pitts, W.H., 6

recurrent networks, 7
Reiher, E., vi, 3, 31, 100
Rejection, 66
Rosenblatt, F., 6, 100
Rumelhart, D.E., 100

Said, F.N., i-iii, 100
Said, J.N., vi
segmentation, 31
spikes, 47
Srihari, S.N., 30, 48, 97, 100, 101
Strathy, N.W., vi, 31, 100
Suen, C.Y., ii, v, 99 101
supervised learning, 8

TRIGONIX database, 47, 58
TRIGONIX Inc., v, 3, 31

uniform distribution, 55
unsupervised learning, 8

user interface, 47, 105

Valiant, L., 9, 101

Wang, D., 101
Wang, P.S.P., 101
Wein, W.X., 101
Widrow, B., 6, 101

Xu, L., 101

Yu, Y.H., 101

Zurada, J.M., 101