Application of Artificial Neural Networks to Optical Character Recognition

by

Osama Abdel-Wahhab Ahmed

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

June, 1994

Order Number 1360389

Application of artificial neural networks to optical character recognition

Ahmed, Osama Abdel-Wahhab, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Application of Artificial Neural Networks to Optical Character Recognition

BY

Osama Abdel-Wahhab Ahmed

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Electrical Engineering

June 1994


بسم الله الرحمن الرحيم

"Praise be to God, the Merciful, the Compassionate, the Lord of the Two Worlds, and blessing and peace upon the Prince of Prophets, our master Muhamed, whom God bless and preserve with abiding and continuing peace and blessings until the Day of the Faith ! "

King Fahd University of Petroleum & Minerals
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by **Osama Abdel-Wahhab Ahmed**
under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the college of Graduate Studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in ELECTRICAL ENGINEERING.

Thesis Committee

Prof. Maher Sid-Ahmed (Advisor)

Dr. M. S. El-Hennawey (Member)

Dr. Selim, Shokri Zaki (Member)

Dr. S. Z. Al-Akhdhar (Member)

Dr. Ahmed Yamani (Member)

Department Chairman

Dean, College of Graduate Studies

4-7-94
Date

This thesis is dedicated to

My Mother who taught me how to give.

ACKNOWLEDGMENT

I wish to thank my thesis advisor Professor. **Maher Sid-Ahmed** for his continuos guidance and encouragement. I wish to thank all my thesis committee Dr. Selim, Shokri Zaki, Dr. M. S. El-Hennawey, Dr. Ahmed Yamani, and Dr. S. Z. Al-Akhdhar, for their valuable suggestions. I would like also to thank King Fahd University of petroleum & Minerals for supporting me during my work.

TABLE OF CONTENT

LIST OF TABLES

Table Page

LIST OF FIGURES

THESIS ABSTRACT

Name : Osama Abdel-Wahhab Ahmed

Title: Application of artificial neural networks to optical characters recognition.

Major Field: Electrical Engineering

Date of Degree: June 1994

In this work we examine shape descriptors and neural network classifiers for the recognition of Arabic characters. There are a total of 29 characters in the Arabic alphabet. However, since the shape of character changes depending on its position in the word

(beginning, middle, end, and stand-alone) we end up with more than 29 shapes.

Shape descriptors are studied in this work. In particular the moment invariant approach due to Hu.[3], is studied and examined, with some modifications, on the Arabic alphabet.

A new algorithm for training the feed forward neural network is developed in this thesis. This algorithm is shown to be faster and more stable than other schemes presented in the literature.

The thesis presents the classification of shapes through shape descriptors and feed forward neural network classifiers. Testing on Arabic characters of different sizes and orientation is carried out. Four separate neural networks were trained for each character position with the seven moment invariants of Hu.[3] as an input. The output is trained to give five bits representing the ASCII code of Arabic letters plus one bit to serve as a parity check. Results of using the new training algorithm for the feed-forward neural network are presented. Very high recognition rate of Arabic fonts is achieved

خلاصة الرسالة

اسم الطالب الكامل : أسامة عبد الوهاب أحمد.

عنوان الدراسة : استخدام الشبكات العصبية في مجال تعرف الالة علي الاحرف العربية آليا.

التخصص : هندسة كهربائية.

تاريخ الرسالة : محرم ١٤١٥.

يختص هذا البحث بامكانية استخدام واصفات الشكل مع المصنفات المبنية علي فكرة الشبكات العصبية في مجال تعرف الالة علي الاحرف العربية آليا. من المعروف ان حروف الهجاء العربية تسعة و عشرون حرفا و لكن يعتمد شكل الحرف علي مكانة في الكلمة من حيث كونة فى البداية أو فى الوسط أو فى النهاية أو بمفردة.

فى هذا البحث درسنا واصفات الشكل و خصوصا طريقة العزوم الغير متغيرة. و لقد تم فى هذا البحث اختبار و تعديل طريقة العزوم الغير متغيرة لتلائم الحروف الابجدية.

و تم فى البحث أيضا بناء خوارزمية جديدة لتدريب الشبكات العصبية ذات التغذية الامامية. و اثبتت النتائج أن هذه الخوارزمية أسرع و أكثر استقرارا من الطرق الاخرى المستخدمة من قبل.

و قد تم بناء نظام مكون واصفات الشكل مع الشبكات العصبية ذات التغذية الامامية للتعرف على الاحرف العربية و تم اختباره على مختلف الاشكال و الاحجام و الميول, و النتائج مدونة فى هذا البحث. و قد دربت أربع شبكات عصبية ذات تغذية أمامية لكل وضع من أوضاع الحرف فى الكلمة باعتبار السبع عزوم الغير متغيرة كمدخل, و الخرج مكون من خمس خوينات من الكود الثنائى يمثلون نظام الشيفرة للحروف العربية بالاضافة الي خوينة سادسة تعمل كخوينة تكافؤ لغرض اكتشاف الاخطاء التى قد تقع أثناء عملية التعرف على الكتابة.

درجة الماجستير فى العلوم

حامعة الملك فهد للبترول و المعادن

الظهران , المملكة العربية السعودية

التاريخ

CHAPTER 1

INTRODUCTION

Optical character recognition, or OCR, is the branch of computer science devoted to the development of algorithms and hardware for translating images of text into machine-readable forms. OCR can offer important improvement in computer productivity because while data can be processed very quickly by computers, data entry process is still very slow and tedious and has been considered as the real bottleneck in data processing.

Optical character recognition (OCR) has been a subject of great interest to many computer scientists, engineers, and people from other disciplines. Intensive research has made OCR an efficient means of entering data directly into the computer and capturing information from data sheets, books, and other machine-printed or handwritten materials. Such capabilities greatly widen the applications of computers in areas like automatic reading of texts and data, man computer communications, language processing, and machine translation. Indeed, present OCR machines have been used to process very large volumes of type- and hand-written and printed data generated by large corporations and government agencies, for example, bank and

postal services, credit card and insurance companies, telephone and electricity companies, medical, taxation, and finance departments which handle millions of accounts and payments each year. OCR has the advantages of little human intervention and higher speed in both data entry and text processing, especially when the data already exist in machine-readable fonts. In fact, OCR has become one of the most successful application of modern technology in the field of applied pattern recognition and artificial intelligence.

Fortunately, market demand for OCR is very strong even though word processors are prevalent. For example, a dozen leading companies sell or are preparing to sell hand-printed character readers. So far these sophisticated machines are not prevalent yet, but it is certain that if the price and performance meet the requests of the users, these machines will be widely used in offices as a very natural man-machine interface. The accumulation of OCR knowledge is reducing the gap between the users and makers, which is also helped by the rapid development of computer technology.

OCR SYSTEM

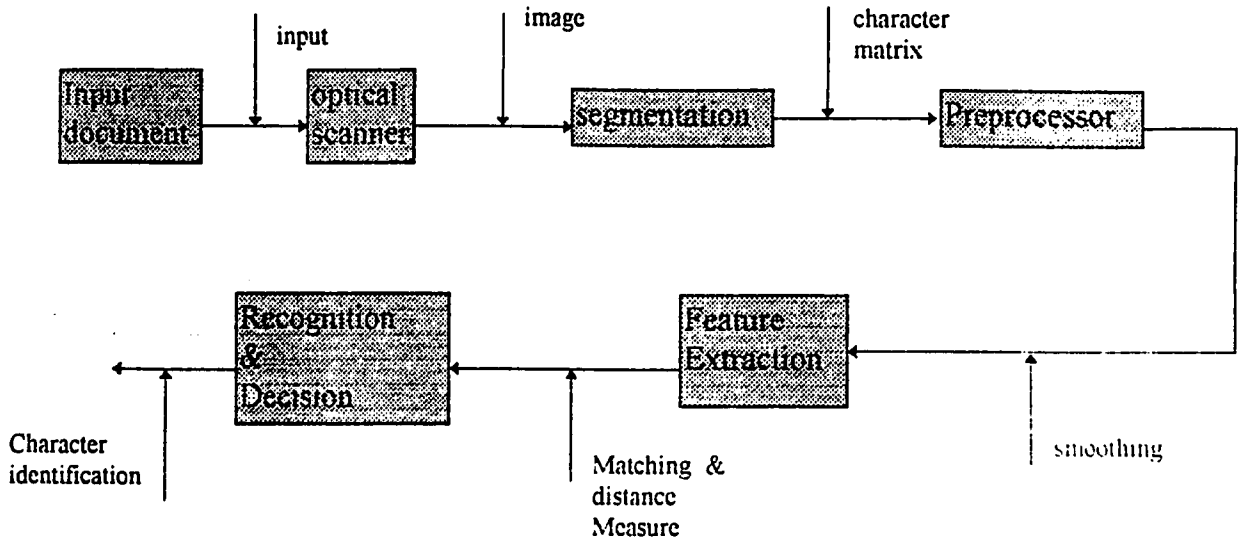

Figure 1 O.C.R. system

A block diagram of an OCR system is shown above in Figure 1. At the input end, characters typed or written on documents are scanned and digitized by an optical scanner to produce a digitized image. Depending on the complexity of the character shapes and the vocabulary involved, the size of the matrix, which reflects the resolution of a digitized character, varies to achieve speed and accuracy. The OCR system will start to locate the regions in which data have been entered, typed, printed, or written on the input documents. Once these regions are found, the data blocks are then segmented into character images. Instead of keeping the images in multigray levels, it is common practice to convert them into binary matrices to save memory space and computational effort. Then characters in the form of binary matrices go through the

preprocessor to eliminate random noise, voids, bumps, and other spurious components which might still be with them.

In some cases, normalization in size, orientation, position, as well as other operations are performed to facilitate the extraction of distinctive features in the subsequent stage. Once the characteristics of the cleaned characters have been extracted, they are matched to a list of references and a knowledge base built during the learning process to classify the characters. In addition, distance measurements are used, as well as shape derivation, shape matching, and hierarchical feature matching in the form of decision trees. The decision-maker is strongly influenced by the types of features detected.

Preprocessing

When patterns are scanned and digitized, the raw data may carry a certain amount of noise, for example, a scanner with low resolution will produce touching line segments and smeared images. In order to eliminate unwanted noise which may cause severe distortions in the digital image and hence ambiguous features and poor recognition rates, a preprocessor is used to smooth the digitized characters. Essentially, smoothing performs the functions of both filling and thinning to eliminate noise, isolated pixels, breaks, or bumps. Normalization is applied to produce patterns of uniform size or linewidth, fixed boundaries along certain edges (e.g., top-left justification), or a preferred orientation (e.g., vertical).

Feature Extraction

In OCR applications, it is best to extract those features which will enable the system to discriminate correctly one class of characters from the others. Since characters are formed from line segments, many different types of shape features can be extracted and used to recognize the characters. We can classify features into two Main groups through global and structural analyses. A full review of feature extraction will be found in the following chapter.

Arabic O.C.R. :

Arabic script and font characteristics are different from English. These differences result in that a direct implementation of the recognition techniques used for English is not possible for Arabic.

- Arabic text is written from right to left and recognition is expected to be in the same direction to allow for connection with a speech synthesis machine or linguistic checking of the recognized words. Whereas English characters can appear in two shapes (upper and lower case), Arabic characters can have four different shapes, depending on the position in the word (beginning, middle, end or alone). Unlike English, most of the Arabic characters of a word are connected along a base line.
- Many Arabic characters have dots which are positioned at a suitable distance above or below the letter body. Dots can be single, double, or triple. Different Arabic letters can have the same body and differ in the number of dots identifying them.
- Some Arabic characters use special marks to modify the letter accent, such as Hamza (ء), Mada (~), again positioned at certain distances from the letter.

- Arabic uses another type of special character as short vowels, which are referred to as diacritics. Although different diacritics on the same characters could lead to different words, an Arabic reader is trained to deduce the meaning of undiacritieized text. This is why diacritics are not used in newspapers or in office correspondence. However, when diacritics are used they appear above or below the characters and are viewed as isolated characters.
- Arabic characters, as described by calligraphers, are composed of two parts, the crown, which identifies the character in any position (beginning, middle, end or alone), and the connection to neighboring characters.
- To preserve the beauty of Arabic script, some letters are positioned to overlap with the neighboring letters The degree of overlapping can vary according to the typeface and the typewriter design. For character recognition this could be a very important feature, as it is commonly used.

Objective:

In this thesis we examine two major blocks used in the building of complete Arabic optical character recognition solution. Namely,

1. Feature extraction.
2. Classification.

Arabic text is utilized as a test bed for these blocks. A complete Arabic Optical character recognition requires the implementation of at least five more blocks

1.Segmentation of each page into text, lines, graphics, and images.

2.Text segmentation into paragraphs ... etc.

3.Line/character segmentation

4.Recognition and separation of diacritics

5.Recognition of ligatures.

The last 5 blocks are not investigated in this work and are left for future extensions. Nonetheless, in depth study in feature extraction schemes is carried-out with special attention to the moment shape descriptors.

Seven of these descriptors are extracted for the 108 different shapes of the Arabic set style (نسخ Nasekh) and they are used as input data to a feed-forward neural network. An in depth investigation into efficient learning (or training) schemes for feed-forward neural network is presented. A new scheme for training the feed-forward neural network which is order of magnitude faster than the classical delta rule .back-propagation method will be presented. The thesis is organized into five chapters

Chapter 2: Literature survey.

In this chapter a review for the previous work in the area of optical character recognition was done for both English characters as well as Arabic

Chapter 3: Shape descriptors.

Shape descriptors is a function which reflects (represent) the shape of an image mathematically and independent on other factors like position, size,

orientation, . . etc. We concentrate on moments as a shape descriptor which was utilized as a feature extractor in our system.

Chapter 4: The feed-forward neural networks and training algorithms.

After extracting the required feature from an image the following step of an OCR system is usually recognition and decision. We concentrate on artificial neural network which was utilized as a classifier in our system. Artificial neural network are described by two features: 1) Topology 2) Training algorithm. For topology, the multi-layer feed-forward perceptrons was used. For training, a newly developed algorithm which is orders of magnitude faster than the ordinary delta rule was described.

Chapter 5: New efficient training algorithms.

A new algorithm for training the multi-layer feed-forward perceptrons was developed which overcomes the limitations and disadvantages of the algorithm discussed in chapter 4. The new algorithm shows faster convergence rate than the previous one.

Chapter 6: Results, conclusion and recommendations.

Partial results are presented after each chapter to verify the algorithms presented.

CHAPTER 2

LITERATURE REVIEW

In the early days of pattern recognition research, OCR was a very popular problem. One reason is that characters are handy and were regarded as a problem which could be easily solved. However, against what was the expectation of many people, great difficulty in solving this problem surfaced. Of course there are practical demands for such a research. which are common to all pattern recognition topics.

In the 1950's and the early half of the 1960's, researchers imagined an ideal OCR, even though they were aware of the great difficulty of the problem. So some resorted to basic research, and engineers took a more systematic approach. Such a trend was seen in template matching and the structure analysis method. Both methods have their own advantages. Template matching is very sensitive to positional change, but it can be very strong in the sense of global matching, While structure analysis method has the advantage of detecting the local stroke features of characters.

A-Template Matching Approach:

The template matching process is divided into two processes, superimposing an input shape to a template and measuring the degree of coincidence between the input shape and the template . Template matching has many types including:

1) Application of Information Theory:

A research on automatic generation of discriminate logic functions began at the IBM Watson Laboratory. Kamentsky and Liu [1], [2] gave such a scheme based on information theory. They gave a measure of discriminating power of a logic function, f, which was chosen randomly. Its configuration is $f(x_i, x_j, \ldots x_n)$ for which $x_i, x_j, \ldots x_n$, are randomly chosen pixels whose values are either 1 or 0. Extensive experiments were done. Their scheme was elaborated by introducing the concept of distance So an automatic design system for the discriminating functions was implemented.

2)Karhunen-Loeve (K-L) Expansion:

K-L Expansion is very attractive and widely used for data compression as well as for pattern recognition. When a data set of [h(x, a)] is given vectors, we can construct the covariance matrix and solve its eigen-vectors, which form the coordinates of the given pattern space. Iijima made a feature extraction system based on the consideration that for a given normalized pattern set denoted by

$$D = \{h(x, \alpha)\},$$

where α is an assigned/indexing number of the individual pattern, and he constructed the following mapping functions to real values:

$$J[\varphi(x)] = \frac{\int_D w(\alpha)(h(x,\alpha),\varphi(x)).d\alpha}{\|\varphi(x)\|} \qquad (1)$$

Where $v(\alpha)$ is the appearance probability of α

3)Series Expansion:

The most typical Series Expansion are moment and Fourier expansions.

a)Moment: Recognition of shapes, independent of position, size, and orientation in the visual field, has been a goal of research. From a practical point of view, in OCR in particular, orientation invariance is not as important as position and size. However, it is a very interesting research goal and work on it has been continuing since Hu's theoretical work [3].

Alt [4] conducted the first systematic experiment in 1962, in which one font of alphabet was used which was normalized in position, size, and slant. Moments up to the second were used as the normalization factors, and the third to sixth moments were used for classification. This was a pilot program and no recognition rate was given for basic experiment; however he was optimistic.

Cash and Hatamian [5] reported a very systematic and reliable experiment on the moment method at Bell Laboratories in 1987. They used central moments up to the third order and so ten moments were used for classification. They compared the performances of three typical similarity measures: Euclidian distance, cross correlation, and Mahalanobis distance. The results were as follows: All three similarities achieved a recognition rate over 95% for all six fonts, and the weighted and normalized cross correlation measure produced the best recognition rates: 99% for four of the six fonts for the high quality data sets.

b)Fourier series: Concerning the studies of Zahn and Roskie's paper [6] . The representation of Fourier coefficients of a boundary can be divided in two ways The first is based on cumulative angular function which is expanded to a Fourier series. The set { A_k, a_k, | k= 1,2. - - } is called the Fourier descriptor (FD) for a curve, where A_k and a_k are the amplitude and phase of the k^{th}-harmonic term. The other representation was proposed by Granlund [7] and developed by Persoon and Fu [8].

4) Feature Matching:

Spinrad [9] extracted primitive stroke features, and then tried to match simultaneously the arrangement of the strokes and their attributes also. The direction

of a stroke was quantized into eight levels and the position of the center of a stroke was also quantized directionwise into 16 viewing the center of gravity of the whole set of strokes, such as N (north) and NE (north-east). Thus, an 8×16 matrix was constructed and at each entry of the matrix a 3×3 submatrix was set, in which the row is the quantized distance from the center of gravity and the column is the quantized stroke length. Thus the total number of the elements was $8 \times 16 \times 3 \times 3 = 1152$. Therefore, 1152-dimensional vectors were constructed, and the correlation between the unknown input vectors and each template vector, corresponding to each class, was measured. The above operation is a somewhat mechanical expansion of the correlation.

Feature matching is still sensitive to stroke positions. In this sense, considerable efforts have been made for normalization. However, so-called linear normalization is not enough and so-called nonlinear normalization has been used. The basic idea is to measure busyness of lines/strokes and relocate the strokes so that the busyness become uniform based on the measurement.

5)Nonlinear Template Matching:

Assumed that a character consists of a sum/concatenation of vectors. We can formalize such a shape as

$C = \{(c_1, c_2, .. c_k, ... c_K)(i_k, j_K)\}$,

where c_k denotes the k^{th} line segment/vector and is further expressed as $C_k = (d_k, l_k)$. Here d_k and l_k are the direction and length of the vector respectively and (i_K, j_K) denote the coordinates of the terminal point. The above expansion is very flexible when each c_k and (i_K, j_K) are changed. Therefore, we need to impose some constraints on it to represent a set of templates. The constraints are described as ranges of direction, length, and terminal points of a vector.

To define "matching" against the template set. First an input image on the $I \cdot J$ matrix is defined simply as follows:

$$A = \{ a(i, j) \}, \qquad i = 1,2,....,I \quad j = 1,2,.....,J$$

where $a(i, j)$ is multilevel.

The similarity between A and B (set) is defined as follows:

$$S(A, B) = \max. \{ (A, B') \mid B' \in B \}$$

where (A, B') denotes inner product of vectors A and B', and B' is the transpose of vector B.

The size of the set of templates is enormous, which shows its degree of flexibility in some sense. but we face too big a problem on how to search the set to find the best match. An efficient technique for solving the problem is dynamic programming. Kovalevsky [10] in 1967, made matching by using dynamic programming.

6) Graphical Matching:

Stroke segments and their relationships are represented by a graph in strict mathematical sense. Therefore, graph isomorphism and subgraph isomorphism provide a basic matching theory [11], actually they are used when the number of nodes is small. Otherwise it still has the problem of complexity.

Strict and general formulation from a more practical point of view given by Ambler *et al.* [12]. A practical method was considered, namely the relaxation method invented by Rosenfeld *et al.* [13]. This was first applied to shape by Davis [14].

B. Structure Analysis Approach:

Template matching method is only appropriate for the recognition of printed characters. For hand-written characters we need another consideration due to the large variation of shape of handwritten characters. Structure analysis method has been applied to handwritten characters recognition. In structure analysis method, there is no mathematical principle, but it's strategy is:

A structure can be broken into parts, it can be descried by the features of these parts and relationships between these parts. Then the problem is how to choose features and relationships between them so that the description gives each character clear identification.

There are several viewpoints to systematically see the complete structure of a character. These are classified as follows:

- thinning line analysis,
- bulk decomposition,
- consecutive slits analysis/stream following,
- contour following analysis,
- background analysis.

Thinning line analysis has been the most intensively investigated. Certainly this is a very important approach and many OCR systems have been made based on it. The analysis is higher than the other analyses in terms of abstraction, except for bulk decomposition. Bulk decomposition can be regarded as being at the same level as thinning line analysis, except bulk decomposition is applicable to the general shape.

1)Thinning Line Analysis:

An observed line usually has a width greater than that of a pixel. So the line is eroded from both sides keeping some constraints so that the line is not broken and shortened (thinning). .

In 1960 Sherman [15] regarded characters as consisting of abstracted lines and constructed a graph. In his graph he ignored a node which has two outgoing lines. Therefore, feature nodes are endpoint, branching point, crossing point, and so on. However, this topological view is very important, because it can absorb terrible

variation of character shape. Beun [16] did an experiment on unconstrained numerals. He used other features, such as node position relationship, to resolve the degenerated class. The experimental results were a 91.7% correct recognition rate and a 2.6% substitution rate. Programs to describe bubble chamber pictures were written based on thinning by McCormick [17]. However, the first systematic and rigorous algorithm was given by Hilditch [18] in 1969. Since then, more than 30 variations of thinning algorithms have been proposed, some of which were compared by Tamura [19]. However, it is well known that we cannot obtain perfectly abstract lines for real lines, which include acute corners and/or intersections. This is a basic problem which cannot be avoided because of the local operations used. Therefore, for its actual application some post processing is done. Since it is an iterative process, it is time consuming. However, thinning is a basic preprocessing block in OCR technology as well as in the recognition of drawings and is widely used in many OCR systems.

2)Bulk Decomposition:

A letter "L" can be regarded as consisting of vertical and horizontal lines, for example. The decomposition can be regarded as a counterpart of principal component analysis in template matching and decomposition began from run-length coding. Consecutive black runs are analyzed and connected if certain conditions are satisfied. Otherwise, a new part/segment of a character is generated.

Engineering oriented method was considered by Spinrad [9]. It consisted of eight directed slits on a frame within which a character image was set. For example, a

vertical slit is moved from the left to the right on the frame, then for a letter "D," at some movement, the slit intersects the vertical stroke of the image of "D." Thus, we can detect its vertical stroke.

After Spinrad, Pavlidis [20] extended his approach from the theoretical point of view. He proposed the k^{th} integral projection. All the black runs for any scan can be labeled by number, for example, from the left: 1, 2, 3, and so on. The k^{th} integral projection is just the sum of the k^{th} labeled black runs. The projection method is resistance against noise, i.e., raggedness of boundaries of character images.

3) Stream Following Analysis:

This method is very simple and is very strong against variations of shape. Perotto called the description a morpho-topological description. However, we need to note because of the simplicity, considerably different shapes are identified as the same. Therefore, we need another raster scanning horizontally, in order to distinguish among "U", " ∩", and " _". For "+" and "T," diagonal scanning is necessary. Nadler [38] proposed such scanning. The description of stream following is based only on the topology of each slit. However, there is no noise removing process and it only works well for ideal patterns. Nadler [22], in 1974, gave a simple algorithm of stream following analysis, in which he used a small window of 2×1. The final description is graphlike and intuitive, being slightly redundant. The algorithm is given by the transition table/diagram of an automaton whose number of state is eight, while the original paper described a seven-state automaton. From the general point of view in

image processing some algorithms had been developed based on the same idea and uses run length encoding [23].

4) Contour Following Analysis:

Feature events along a contour of a closed shape is a circulant description. the term of contour following is used in a broad sense, i.e. including both boundary tracing and line (thin) following. Historically speaking, the first contour following mechanism was given by using an analog device, called a flying spot scanner. This was developed by Greanias *et. al.* [25] and was fully used in the IBM 1287 OCR system. It was very fast and flexible. Furthermore it could absorb the coarseness of the boundary. On other hand, its computer counterpart was very slow suffered from boundary noise. The first problem of contour following is Coarseness on the grid plane. Some smoothing techniques have been developed. A 3×3 window averaging process is one of them. A direct averaging technique on a contouring curve was done by Freeman [26] and Gallus *et al.* [27].

The second problem is feature selection along a contour. The third problem is how to segment a contour. The third is the most difficult and fundamental problem, as mentioned before.

The first reliable technique for the segmentation problem was given by Rosenfeld and Johnston [28]. Analytically a curvature is given by a function of local derivatives, but actually it has the global nature suggested by Freeman. This was extensively examined by Freeman and Davis [29]. The corner feature plays a crucial

role. For example, sometimes differentiating between "O" and "D" and between "5" and "S." So considerable effort has been spent in this area. However, the methods developed do not always give stable and consistent results. Therefore an alternative approach was taken, one based on a polygonal approximation [30], which is global and very strong against noise. Considerable work has been done on polygonal approximation methods. The work of Pavlidis and Horowitz [31] is very general. It is not constrained to make a connection. In this sense, it gives a global approximation which meets human intuition. Other polygonal approximation algorithms can be found in [32] .

5)Background Analysis:

Glucksman took an approach of background analysis. Each background pixel takes a four-digit code. This maybe a binary, ternary, or higher-order code. Classification was done using the feature vector, each element of which is a number of the ternary coded pixels. The histogram of the background features, ternary codes, was used. Theoretically, the feature vector has 81 dimensions, but 30 were sufficient in practice. An experiment on alphabetic characters of nine fonts and 52 classes of uppercase and lowercase letters was done. The total number of samples was 26643, a large number. The data sets were provided by Casey of the IBM Watson Research Laboratories. The correct recognition, rejection, and misclassified rates were 96.8%, 0.3%, and 2.9% respectively.

Munson [33] proposed an interesting method, also based on background analysis using contour following. A character image's boundary is traced and some points are marked as extreme points. Using these extreme points, a convex hull is constructed and connected regions adjacent to both the boundary and convex hull are detected as concavity regions. Enclosures are detected when tracing the boundary. Concerning the detection of the extreme points, however, only a simple procedure was given. That procedure involves tracing a boundary, clockwise with a right hand system. Points turned to the right were searched and consecutive extreme points were connected with a straight line so that they lie inside the image.

Such a procedure is local and generates many extreme points; therefore some filtering technique must be provided to find global extreme points. No further structural analysis was done and both concavity and enclosure were used as a part of the components of the feature vector. An experiment on a hand-printed 46-character was conducted, which gave a 97% correct reading rate for test data and a 3% of error rate. The data size was not specified.

LEARNING:

Researchers have been striven to incorporate new expertise into preprocessing, feature extraction, and classification stages of their methods, and they have experimented new approaches such as expert system, neural networks, mathematical morphology, or any combination of them.

Mathematical morphology:

Michell and Gallies [34] use the tool of mathematical morphology to extract cavity features as the starting input to their specialized digit recognizer. Thirty-three numeral model were "painstakingly crafted" with an iterative refine-and-test methodology over several thousand digits. Classification is performed by a symbolic model matching process.

Expert system:

For totally unconstrained characters some expert system have been developed by R. M. Brown, T. M. Fay, and C. L. Walker, [35]. Suen *et al.* introduced a multiple-expert system [36]. They report a very encouraging results. They combine the expertise of these experts to enforce their strengths and to suppress their weaknesses.

Neural Networks:

Recently, the use of Neural Networks to recognize characters and different types of patterns has resurfaced (full description is found in chapter 4).

Very good results were recently reported with neural networks. In [37] Krzyzak *et al.* first extracted features from the contours of numerals: 15 complex Fourier descriptors from the outer contours and simple topological

features from the inner contours. These features are then presented as a three-layer back-propagation networks. Le Cun *et al.* [38] achieved excellent results with a back-propagation networks using size-normalized images as direct input.

Arabic OCR :

Amin, *et. al.* [39] presented a method for the recognition of multifont Arabic texts as follows: digitization, line separation, word separation, segmentation of a word into characters, identification of each character, and recognition of the word, A vertical and horizontal histogram as a classifier was used, but poor results < 85%. El-Sheikh, Talaat *et. al.* [40] used Fourier descriptors from the coordinate sequences of the outer contour of each character. A topological classifier is used to classify the stress mark over or under the character contour. A reject option is introduced by the classifier so that incorrectly segmented characters are detected.

In Amin, Adnan *et al.* [41] Character recognition scheme is divided into three phases: the digitization process, segmentation of words into characters, and identification of characters. Character recognition was achieved despite several impeding properties of the Arabic script, especially the connectivity of characters.

El Gowely, Khaled *et al.* [42] used three interleaved phases. The segmentation phase attempts to produce an initial set of characters from the connected text according to a set of predefined rules. The output is then passed to a preliminary

classification phase that attempts to label the unknown characters into one of ten possible classes according to a set of rules that acquire their parameter values through learning. The last phase contains a more elaborate set of rules that recognize characters within each class. This recognition phase is designed to allow errors during segmentation and/or classification to be rectified through an adaptive recognition technique. Impedovo, *et al.* [43] recognized handwritten Arabic numerals using Fourier descriptors. The classification phase is based on a new similarity definition introduced from one of the Banach algebra of continuous and bounded plane curves. The learning phase is developed using a man-machine interactive system in which the role of man as character-source and the role of the machine as recognizer are partially interchangeable. El-Khaly, *et al.* [44] investigated the use of moment-invariant descriptors are investigated for the purpose of recognition of individual characters. An algorithm for separation of individual characters was developed.

Sami El-Dabi, *et al.* [45] involves a statistical approach for character recognition. This approach uses 'Accumulative Invariant Moments' as an identifier, which helped in the segmentation of connected and overlapping Arabic characters. However, Invariant Moments proved to be very sensitive to slight changes in a character shape. The recognition zone was defined based on the mean and standard deviation for the moments of a large sample of each character. However, this zone was increased, using an empirical multiplier, to improve recognition rate. In [15] the character is segmented into primary and secondary parts (dots and zigzags). The secondary parts of the

character are then isolated and identified separately, thereby reducing the number of classes from 28 to 18. The moments of the horizontal and vertical projections of the remaining primary characters are then calculated and normalized with respect to the zero-order moment. Simple measures of the shape are obtained from the normalized moments. A 9-D feature vector is obtained for each character. Classification is accomplished using quadratic discriminate functions.

Few paper appeared for on-line character recognition: In El-Wakil, *et al.* [46] features that are found to be independent of the writer style are represented as a list (vector) of integer values, while those that are subjected to more variations are represented using a Freeman-like chain code. This mixing of the representation combined with a hierarchical organization of the characters proved to be useful in reducing recognition time. In Al-Emami, *et al.* [47] the length and the slope of each segment was found, and the slope categorized into one of four directions. In the learning process, specifications on the strokes of each character are fed to the computer. In the recognition process, the parameters of each stroke are found and select the collection of strokes which best matches the features of one of the stored characters. In Al-Yousefi, *et al.* [48] a hierarchical system for the recognition of on-line Arabic mathematical symbols is developed. A precedence grammar has also been developed for the recognition of one-dimensional Arabic mathematical formulas. The set of characters comprising formulas includes sixteen isolated letters, ten digits and eleven mathematical symbols. Several features are used in the recognition process,

where different features are utilized sequentially in different stages of the hierarchical system.

CHAPTER 3

SHAPE DESCRIPTORS

Moments and other shape descriptors have been utilized as pattern features in a number of applications to achieve invariant recognition of two-dimensional image patterns [51]-[58]. A number of techniques have been developed to extract features which are invariant with respect to object translation, scale change, and rotation. Hu [51] first introduced moment invariants in 1961, based on methods of algebraic invariants.

Teague [59] suggested the notion of orthogonal moments to recover the image from moments based on the theory of orthogonal polynomials, and has introduced *Zernike moments,* which allow independent moment invariants to be constructed easily to an arbitrarily high order. Other orthogonal moments are *Legendre moments,* making use of Legendre polynomials. In [60] *rotational moments* are used to extend the definition of moment invariants to arbitrary order in a manner which ensures that their magnitudes do not diminish significantly with increasing order. More recently, the

notion of *complex moments* [61], [62] has been introduced as a simple and straightforward way to derive moment invariants.

Types of Moments:

Assume that the real image intensity function $f(x, y)$ is piece-wise continuous and has bounded support.

A. Geometric Moments :

The geometric moments of order $(p+q)$ of $f(x, y)$ are defined as

$$M_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p . y^q . f(x,y).dx.dy \qquad (1)$$

where $p, q = 0, 1, 2, \ldots, \infty$. The above definition has the form of the projection of the function $f(x, y)$ onto monomial $x^p y^q$.

B. Legendre Moments :

The Legendre moments of order $(m+n)$ are defined as

$$\lambda_{mn} = \frac{(2m+1)(2n+1)}{4} \cdot \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P_m(x).P_n(y).f(x,y).dx.dy \qquad (2)$$

where m, n = 0, 1, 2, . . ., ∞. The Legendre polynomials $\{P_m(x)\}$ [64] are a complete orthogonal basis set on the interval [-1, 1]:

$$\int_{-1}^{1} P_m(x)P_n(x).\mathrm{dx} = \frac{2}{(2m+1)}\delta_{mn} \qquad (3)$$

The n^{th}-order Legendre polynomial is

$$P_n(x) = \sum_{j=0}^{n} a_{nj}x^j = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n \qquad (4)$$

C. Zernike Moments :

The complex Zernike moments of order n with repetition l are defined as

$$A_{nl} = \frac{n+1}{\pi}\int_{0}^{2\pi}\int_{0}^{\infty}[V_{nl}(r,\theta)]^*.f(r.\cos\theta, r.\sin\theta).r.dr.d\theta \qquad (5)$$

where $n = 0, 1, 2, \ldots, \infty$. and l takes on positive and negative integer values subject to the conditions

$$n - |l| = \text{even}, \qquad |l| \le n. \tag{6}$$

The symbol * denotes the complex conjugate. The Zernike polynomials

$$V_{nl}(x, y) = V_{nl}(r.\cos\theta, r.\sin\theta) = R_{nl}(r)\, e^{il\theta} \tag{7}$$

are a complete set of complex-valued functions orthogonal on the unit disk $x^2 + y^2 \le 1$:

$$\int_0^{2\pi}\int_0^1 \left[V_{nl}(r,\theta)\right]^* . V_{mk}(r,\theta) r.dr.d\theta = \frac{\pi}{n+1}\delta_{nm}.\delta_{kl} \tag{8}$$

The real-valued radial polynomials $\{R_{nl}(r)\}$ satisfy the relations

$$\int_0^1 R_{nl}(r).R_{ml}(r).r.dr = \frac{1}{2(n+1)}\delta_{nm} \tag{9}$$

and are defined as

$$R_{nl}(r) = \sum_{s=0}^{(n-|l|)/2} (-1)^s \cdot \frac{(n-s)!}{s!(\frac{(n+|l|)}{2}-s)!(\frac{(n-|l|)}{2}-s)!} r^{n-2s}$$

$$= \sum_{\substack{k=|l| \\ n-k=even}}^{n} B_{n|l|k} r^k .$$

(10)

D. Pseudo-Zernike Moments :

A related orthogonal set of polynomials in x, y, and r to Zernike polynomials was derived in [67] which has properties analogous to those of Zernike polynomials. This set of polynomials differs from that of Zernike in that the real-valued radial polynomials are defined as

$$R_{nl}(r) = \sum_{s=0}^{n-|l|} (-1)^s \cdot \frac{(2n+1-s)!}{s!(n-|l|-s)!(n+|l|+1-s)!} r^{n-s}$$

$$= \sum_{k=|l|}^{n} S_{n|l|k} r^k$$

(11)

where $n = 0, 1, 2, \ldots, \infty$. and l takes on positive and negative integer values subject to $|l| \leq n$ only.

This set of pseudo-Zernike polynomials contains $(n+1)^2$ linearly independent polynomials of degree $\leq n$, whereas the set of Zernike polynomials contains only $(n+1)(n+2)/2$ linearly independent polynomials of degree $\leq n$ due to the additional condition of

$n - |l| =$ even.

E. Rotational Moments :

The rotational moments of order n with repetition l are defined as

$$D_{nl} = \int_0^{2\pi}\int_0^{\infty} r^n e^{-il\theta} f(r.\cos\theta, r.\sin\theta) r.dr.d\theta \qquad (12)$$

where $n = 0, 1, 2, \ldots, \infty$. and l takes on any positive and negative integer values.

F. Complex Moments :

The notion of complex moments was recently introduced in [61] as a simple and straightforward way to derive moment invariants. The complex moments of order $(p+q)$ are defined as

$$C_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x+iy)^p.(x-iy)^q.f(x,y).dx.dy \qquad (13)$$

where p, $q = 0, 1, 2, \ldots, \infty$. and $i = \sqrt{-1}$.

In polar coordinates, the complex moment of order $(p+q)$ can be written as

$$C_{nl} = \int_{0}^{2\pi}\int_{0}^{\infty} r^{p+q} e^{-i(p-q)\theta} f(r.\cos\theta, r.\sin\theta) r.dr.d\theta \qquad (14)$$

Noise Sensitivity

Moment invariants using various schemes based on the different moment types as defined above have been shown to provide perfect invariance properties under noise-free condition. However, in the presence of noise, the computed invariant moments are expected not to be strictly invariant. Cho-Huak Teh [65] show that complex and geometric moments less affected by noise than other moments.

Invariant Moments:

Using nonlinear combinations of *geometric moments*, Hu[51] derived a set of invariant moments which has the desirable properties of being invariant under image translation, scaling, and rotation.

Getting the central geometric moments defined as

$$M_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\bar{x})^p(y-\bar{y})^q . f(x,y).dx.dy \tag{15}$$

Where

$$\bar{x} = \frac{M_{10}}{M_{00}} \quad , \quad \bar{y} = \frac{M_{01}}{M_{00}} \tag{16}$$

Where $M_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p . y^q . f(x,y).dx.dy$

Central moments has the property of translation invariance

From the second- and third-order moments, a set of seven invariant moments, which is invariant to translation and rotation has been derived by Hu[51]:

$$\phi 1 = M_{20} + M_{02}. \tag{17}$$

$$\phi 2 = (M_{20}\text{-}M_{02})^2 + 4M_{11}^2 \tag{18}$$

$$\phi 3 = (M_{30}\text{-}3M_{12})^2 + (3M_{21} - M_{03})^2. \tag{19}$$

$$\phi 4 = (M_{30}+M_{12})^2 + (M_{21}+M_{03})^2. \tag{20}$$

$$\begin{aligned}\phi 5 = {} & (M_{30}\text{-}3M_{12})\,(M_{30}+M_{12})\,[(M_{30}+M_{12})^2 \\ & - 3(M_{21}+M_{03})^2] + (3M_{21} - M_{03})(M_{21}+M_{03}) \\ & *[3(M_{30}+M_{12})^2 - (M_{21}+M_{03})^2].\end{aligned} \tag{21}$$

$$\begin{aligned}\phi 6 = {} & (M_{20} - M_{02})[(M_{30}+M_{12})^2 - (M_{21}+M_{03})^2] \\ & + 4\,M_{11}(M_{30}+M_{12})(M_{21}+M_{03}).\end{aligned} \tag{22}$$

$$\begin{aligned}\phi 7 = {} & (3M_{21} - M_{03})(M_{30}+M_{12})[(M_{30}+M_{12})^2 \\ & - 3(M_{21}+M_{03})^2] + (3M_{21} - M_{03})(M_{21}+M_{03}) \\ & *[3(M_{30}+M_{12})^2 - (M_{21}+M_{03})^2].\end{aligned} \tag{23}$$

These functions can be normalized to make them invariant under a scale change by using the normalized central moments instead of the central moments .

The normalized central moments are defined by:

$$m_{pq} = \frac{M_{pq}}{M_{00}^{a}} \tag{24}$$

where

$$a = (p+q)/2 + 1 \tag{25}$$

which when substituted in equations (18) to (24) gives a set of seven moments, which is invariant to translation, scale change, and rotation.

The Φ's have large dynamic ranges. Thus it was found that [68] it is more practical to deal with the logarithm of the magnitude of the Φ's . the seven invariant moments we shall use are :

$$\Phi 1 = \log | \ m_{20} + \ m_{02} | \tag{26}$$

$$\Phi 2 = \log | \ (m_{20}-m_{02})^2 \ + \ 4m_{11}{}^2 \ |. \tag{27}$$

$$\Phi 3 = \log | (m_{30} - 3m_{12})^2 + (3m_{21} - m_{03})^2 |. \quad (28)$$

$$\Phi 4 = \log | (m_{30} + m_{12})^2 + (m_{21} + m_{03})^2 |. \quad (29)$$

$$\Phi 5 = \log | (m_{30} - 3m_{12})(m_{30} + m_{12}) [(m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2] + (3m_{21} - m_{03})(m_{21} + m_{03}) * [3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2] |. \quad (30)$$

$$\Phi 6 = \log | (m_{20} - m_{02})[(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2] + 4\, m_{11}(m_{30} + m_{12})(m_{21} + m_{03}) |. \quad (31)$$

$$\Phi 7 = \log | (3m_{21} - m_{03})(m_{30} + m_{12})[(m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2] + (3m_{21} - m_{03})(m_{21} + m_{03}) * [3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2] |. \quad (32)$$

For digital images the continuos image intensity function $f(x , y)$ is replaced by a matrix where x and y are the discrete locations of the image pixels. The integrals in equation (1) and equation (15) are approximated by the summations:

$$M_{pq} = \sum_{x=0}^{m} \sum_{y=0}^{n} x^p y^q . f(x,y) \tag{34}$$

$$M_{pq} = \sum_{x=0}^{m} \sum_{y=0}^{n} (x - \bar{x})^p (y - \bar{y})^q . f(x,y) \tag{35}$$

where m and n are the dimensions of the image.

Any Arabic font can take on different shape depending on its position in a connected segment of a word. Four different shapes for each character arise; viz.: separated, beginning, middle, end. For style type Naskh (نسخ) the fonts are shown in Figure 2. The value Φ's is tabulated for all Arabic letters found in style (نسخ)in the four different cases (separated, beginning, middle, end) in Tables 6, 7, 8, 9.

ا	ـا	ـا	ا	ضـ	ـضـ	ـض	ض
بـ	ـبـ	ـب	ب	طـ	ـطـ	ـط	ط
تـ	ـتـ	ـت	ت	ظـ	ـظـ	ـظ	ظ
ثـ	ـثـ	ـث	ث	عـ	ـعـ	ـع	ع
جـ	ـجـ	ـج	ج	غـ	ـغـ	ـغ	غ
حـ	ـحـ	ـح	ح	فـ	ـفـ	ـف	ف
خـ	ـخـ	ـخ	خ	قـ	ـقـ	ـق	ق
دـ	ـدـ	ـد	د	كـ	ـكـ	ـك	ك
ذـ	ـذـ	ـذ	ذ	لـ	ـلـ	ـل	ل
رـ	ـرـ	ـر	ر	مـ	ـمـ	ـم	م
زـ	ـزـ	ـز	ز	نـ	ـنـ	ـن	ن
سـ	ـسـ	ـس	س	هـ	ـهـ	ـه	ه
شـ	ـشـ	ـش	ش	وـ	ـوـ	ـو	و
صـ	ـصـ	ـص	ص	يـ	ـيـ	ـي	ي

Figure 2 Arabic letters for different shapes

Effect of Digitization:

The moment invariants are truly invariant under image translation, scaling, and rotation only for a continuous image function. For our application the image functions are discrete. The set of invariant moments equations (26) to (32), because of digitization, is still invariant under image translation, but not so for image scale or image rotation changes.

We examine the stability of the results against rotation for letter ش with rotation 2°, 3°, 4°, 5°, 10°, 15° and the result is tabulated in Table 1. We notice a big variance although we had used double precision (64 bit precision). To avoid dealing with very high numbers we divided the scale of x and y in equation (35) by 10 (for example if we were to use a 256×256 pixel image, and the calculated difference and M_{03} which require that we multiply $f(x, y)$ with a number which can reach 256^3 i.e. 1677716 could be very large thus affecting round off error).

Table 1 Letter ش with rotation 2°, 3°, 4°, 5°, 10°, 15°

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
ش ٠	-0.664	-2.824	-5.336	-6.106	-12.092	-8.483	-11.903
ش ٢	-0.655	-2.783	-5.867	-5.357	-12.093	-6.758	-10.970
ش ٣	-0.655	-2.781	-5.881	-5.369	-12.172	-6.770	-10.995
ش ٤	-0.657	-2.781	-7.005	-4.907	-11.555	-6.396	-10.873
ش ٥	-0.656	-2.775	-6.904	-4.904	-11.083	-6.379	-10.880
ش ١٠	-0.662	-2.747	-5.173	-4.355	-9.120	-5.992	-10.203
ش ١٥	-0.670	-2.738	-4.708	-4.145	-8.590	-5.834	-9.106
Sd.Dv.	0.00528	0.02588	0.79929	0.61515	1.3838703	0.8126	0.79796
Mean	-0.65986	-2.77557	-5.83914	-5.02043	-10.957857	-6.65886	-10.70429
Sd./ M	-0.008	-0.00932	-0.13688	-0.12253	-0.1262902	-0.12203	-0.074546

We examined also the stability of the results against scale change for letter ċ , we used images with 64×64, 60×60, 50×50, 45×45, 40×40, 32×32 and the result is tabulated in Tables (2).

Table 2 Letter ċ with 65×65, 59×59, 45×45, 31×31, 29×29 Pixel images

size	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
ċ64×64	-0.704	-4.479	-6.123	-5.641	-12.175	-7.911	-11.535
ċ60×60	-0.699	-3.965	-6.465	-5.563	-11.603	-7.862	-12.045
ċ50×50	-0.701	-4.589	-5.978	-5.860	-12.731	-8.162	-11.782
ċ45×45	-0.709	-4.449	-5.773	-5.529	-11.389	-7.821	-11.285
ċ40×40	-0.705	-5.047	-6.371	-6.013	-12.431	-8.568	-12.301
ċ32×32	-0.702	-4.514	-5.900	-5.647	-11.590	-7.935	-11.554
Std. Dev.	0.0032	0.31525	0.24799	0.17195	0.491373	0.2585	0.339734
Mean	-0.70333	-4.50717	-6.10167	-5.70883	-11.9865	-8.04317	-11.75033
Std. / Mn		-0.06994	-0.04064	-0.03012	-0.040994	-0.03214	-0.028913

As seen from 1 and Table 2 that the seven moments invariants are not absolute invariant against size changes but this variances can be accepted .

Khotanzad [66] normalized the image width and length [-1, +1] such that equation (34) will be

$$\mathcal{M}_{pq} = \sum_{x=-1}^{+1} \sum_{y=-1}^{+1} x^p y^q . f(x,y) \qquad (12)$$

We implemented Khotanzad algorithm but the stability worsened as shown in Table 3 with respect to Table 2, for example Standard deviation/Mean for $\Phi(1)$ is =-0.00455 for without normalization and = -0.14121 with normalization.

Table 3 Using Khotanzad normalization for letter ځ with 65×65, 59×59, 45×45, 31×31, 29×29 Pixel images

size	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
ځ64×64	-1.688	-6.459	-8.856	-8.459	-17.210	-11.925	-16.347
ځ60×60	-1.622	-6.283	-8.851	-8.299	-17.072	-11.696	-15.985
ځ50×50	-1.471	-5.874	-8.292	-7.840	-17.375	-11.469	-14.907
ځ45×45	-1.388	-6.133	-7.717	-7.213	-16.513	-11.152	-13.678
ځ40×40	-1.287	-5.692	-7.648	-7.048	-14.748	-10.347	-13.444
ځ32×32	-1.090	-5.453	-6.791	-6.123	-12.957	-9.425	-11.623
Std. Dev.	0.20113	0.34558	0.73074	0.80171	1.612029	0.865073	1.616487
Mean	-1.42433	-5.98233	-8.02583	-7.497	-15.97917	-11.00233	-14.33067
Std. / Mn	-0.14121	-0.05777	-0.09105	-0.10694	-0.100883	-0.078626	-0.112799

An in-depth study of the effects of sampling, digitizing, and quantization noise on moment invariants will be found in C. H. Teh and R. T. Chin. [63].

T. Chin suggests the use of Simpson's rule as a better approximation for equations (1) and (15) rather than the traditional double summation methods. We implement this method and tested it for rotation for letter ش with rotation 2°, 3°, 4°, 5°, 10°, 15° and the result is tabulated in Table 4. We have examined also the stability against scale change for letter ج with scale change, we use images with 65 × 65, 59 × 59, 45 × 45, 31 × 31, 29 × 29 and the result is tabulated in

Table 5. Simpson's method gives little improvement.

Table 4 Using Simpson's rule for letter ش with rotation 2°, 3°, 4°, 5°, 10°, 15°

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
0°	-0.657	-2.779	-5.252	-6.415	-12.533	-8.777	-12.317
2°	-0.650	-2.738	-5.826	-5.192	-11.975	-6.581	-10.701
3°	-0.651	-2.741	-5.917	-5.214	-11.275	-6.617	-10.802
4°	-0.651	-2.744	-7.029	-4.746	-10.983	-6.215	-10.681
5°	-0.650	-2.736	-6.826	-4.753	-10.663	-6.220	-10.729
10°	-0.657	-2.707	-5.017	-4.236	-8.863	-5.832	-10.073
15°	-0.666	-2.697	-4.576	-4.050	-8.379	-5.699	-8.932
Std.	0.00547	0.02476	0.84393	0.72451	1.42186	0.95802	0.93213
Mean	-0.65457	-2.73457	-5.77757	-4.94371	-10.6673	-6.563	-10.605
Sd./ M	-0.00836	-0.00905	-0.14607	-0.14655	-0.13329	-0.14597	-0.0879

Table 5 Using Simpson's rule for letter ځ with 65×65, 59×59, 45×45, 31×31, 29×29 Pixel images

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
ځ64×64	-0.702	-4.532	-5.913	-5.562	-12.166	-7.853	-11.304
ځ60×60	-0.698	-4.217	-6.109	-5.501	-11.499	-7.750	-11.421
ځ50×50	-0.697	-4.914	-5.959	-5.929	-12.292	-8.389	-11.906
ځ45×45	-0.708	-4.239	-5.674	-5.576	-11.354	-7.787	-11.347
ځ40×40	-0.701	-5.283	-5.862	-5.877	-12.267	-8.563	-11.767
ځ32×32	-0.702	-4.599	-5.606	-5.517	-11.185	-7.910	-11.283
Std. Dev.	0.003543	0.374406	0.170048	0.174088	0.458553	0.31498	0.241919
Mean	-0.70133	-4.63067	-5.85383	-5.66033	-11.7938	-8.042	-11.5047
Std. / Mn	-0.00505	-0.08085	-0.02905	-0.03076	-0.03888	-0.03917	-0.02103

A Complete set of the seven moments for all Arabic characters in their four positions are listed in tables A, B, C, D. The variances between the moments of each letter is large enough such that it can be distinguished by the classifier.

Table 6 beginning letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
أ	-0.693	-2.341	-5.649	-5.235	-10.877	-6.969	-10.788
ب	-0.632	-2.295	-4.325	-4.541	-10.348	-6.073	-8.975
ت	-0.650	-2.159	-3.999	-4.442	-8.663	-5.769	-9.917
ث	-0.667	-2.530	-4.191	-4.785	-9.281	-6.302	-10.013
ج	-0.612	-3.840	-4.510	-4.370	-9.048	-7.438	-8.899
ح	-0.590	-3.999	-4.714	-4.658	-9.382	-6.671	-9.737
خ	-0.656	-3.531	-4.060	-4.076	-8.224	-6.774	-8.398
د	-0.587	-2.324	-5.366	-4.577	-9.642	-5.942	-9.776
ذ	-0.655	-3.184	-4.696	-4.096	-9.375	-5.791	-8.495
ر	-0.624	-2.518	-3.999	-4.193	-8.542	-5.704	-8.369
ز	-0.677	-3.868	-5.475	-5.098	-10.384	-7.374	-11.627
س	-0.634	-3.635	-3.699	-3.325	-6.876	-5.287	-7.224
ش	-0.606	-3.401	-3.562	-3.400	-7.343	-5.225	-6.908
ص	-0.606	-3.035	-3.913	-3.544	-7.292	-5.459	-7.813
ض	-0.648	-3.726	-4.114	-4.050	-8.273	-5.926	-8.292
ط	-0.602	-2.266	-3.987	-3.913	-8.539	-5.300	-7.872
ظ	-0.586	-2.223	-4.453	-4.176	-8.749	-5.361	-8.570

	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
ع	-0.605	-3.488	-4.128	-4.299	-9.217	-6.043	-8.521
غ	-0.632	-3.562	-4.156	-4.138	-8.762	-6.422	-8.311
ف	-0.608	-2.423	-3.843	-4.159	-8.738	-5.551	-8.176
ق	-0.620	-2.405	-4.121	-4.402	-10.710	-5.606	-8.664
ك	-0.604	-2.873	-3.756	-3.841	-8.074	-5.504	-7.671
ل	-0.643	-2.419	-3.781	-4.368	-8.949	-5.582	-8.465
م	-0.582	-2.552	-3.610	-3.672	-7.317	-4.989	-8.194
ن	-0.615	-2.633	-3.282	-3.995	-8.455	-5.436	-7.639
ه	-0.499	-1.968	-3.242	-3.285	-6.572	-4.288	-7.040
و	-0.644	-3.513	-4.378	-4.446	-9.808	-6.240	-8.861
ى	-0.611	-2.884	-4.394	-3.720	-8.437	-5.429	-7.788

Table 7 Isolated letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1. أ	-0.672	-2.546	-6.027	-4.745	-10.732	-6.805	-10.145
2. ب	-0.670	-3.747	-5.117	-4.261	-8.972	-6.663	-9.449
3. ت	-0.634	-4.916	-4.638	-4.673	-9.401	-8.688	-9.602
4. ث	-0.630	-3.938	-3.859	-4.122	-8.126	-6.148	-8.737
5. ج	-0.655	-3.158	-4.485	-3.912	-8.153	-5.522	-8.483
6. ح	-0.638	-3.134	-5.240	-4.290	-9.079	-5.959	-9.551
7. خ	-0.647	-2.280	-4.963	-4.938	-9.999	-6.101	-10.087
8. د	-0.616	-2.395	-4.856	-4.001	-8.456	-5.282	-8.897
9. ذ	-0.654	-3.026	-5.887	-4.304	-10.003	-5.867	-9.414
10. ر	-0.654	-2.426	-4.978	-4.243	-8.919	-5.660	-9.148
11. ز	-0.675	-2.466	-5.773	-4.192	-9.600	-5.855	-9.208
12. س	-0.657	-3.500	-3.769	-3.820	-7.618	-5.720	-8.488
13. ش	-0.605	-2.351	-3.726	-4.072	-8.392	-5.330	-8.004
14. ص	-0.631	-3.061	-4.170	-4.518	-9.536	-6.064	-8.873
15. ض	-0.661	-3.059	-4.500	-5.272	-10.159	-6.886	-11.503
16. ط	-0.652	-4.794	-4.833	-4.682	-9.926	-7.091	-9.464
17. ظ	-0.607	-4.176	-4.142	-4.093	-8.211	-6.182	-9.546

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
18. غ	-0.678	-3.447	-5.418	-4.447	-9.389	-6.455	-10.058
20. ف	-0.671	-3.423	-4.189	-3.899	-8.073	-5.752	-8.116
21. ق	-0.662	-2.901	-4.766	-4.313	-8.856	-5.780	-9.729
22. ك	-0.588	-2.981	-4.226	-3.605	-7.623	-6.338	-7.734
23. ل	-0.667	-2.978	-4.460	-4.430	-9.563	-6.143	-8.885
24. م	-0.679	-2.472	-5.496	-4.101	-9.892	-6.152	-8.901
25. ن	-0.615	-2.271	-4.565	-4.235	-8.638	-5.422	-9.510
26. ه	-0.690	-2.443	-4.885	-4.370	-10.040	-6.857	-8.999
27. و	-0.669	-2.415	-5.147	-4.562	-9.598	-6.654	-9.539
28. ي	-0.572	-2.335	-3.717	-3.690	-7.440	-4.869	-7.752

Table 8 Ending letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1. أ	-0.637	-2.449	-3.539	-4.254	-8.152	-5.479	-9.232
2. ب	-0.656	-3.477	-5.605	-4.353	-9.369	-6.159	-9.732
3. ت	-0.660	-2.984	-3.456	-3.977	-7.801	-5.556	-7.898
4. ث	-0.645	-4.760	-3.579	-4.117	-8.004	-6.513	-8.360
5. ج	-0.551	-2.399	-4.176	-3.720	-7.706	-5.279	-8.062
6. ح	-0.580	-4.058	-3.548	-3.446	-7.655	-5.635	-6.952
7. خ	-0.585	-2.351	-5.029	-5.043	-10.570	-6.255	-10.103
8. د	-0.623	-2.905	-3.275	-3.161	-6.387	-4.640	-7.086
9. ذ	-0.582	-1.948	-3.417	-3.278	-6.722	-4.302	-6.847
10.ر	-0.582	-2.930	-4.238	-4.190	-8.409	-6.024	-9.235
11.ز	-0.584	-2.392	-3.456	-3.394	-7.122	-4.963	-6.881
12.س	-0.555	-2.248	-3.433	-5.448	-9.899	-7.043	-10.549
13.ش	-0.586	-3.083	-3.112	-3.678	-7.578	-5.247	-7.096
14.ص	-0.554	-2.299	-3.657	-3.906	-7.688	-5.060	-8.795
15.ض	-0.629	-3.207	-4.174	-4.554	-9.109	-6.223	-9.034
16.ط	-0.627	-3.219	-3.889	-3.468	-7.165	-5.079	-7.686
17.ظ	-0.606	-2.563	-3.720	-3.618	-7.294	-4.900	-8.034

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
18. ع	-0.581	-2.413	-4.539	-3.949	-8.471	-5.160	-8.264
19. غ	-0.590	-2.108	-4.612	-5.010	-9.928	-6.082	-10.026
20. ف	-0.665	-2.825	-3.826	-3.939	-7.991	-6.178	-7.955
21. ق	-0.602	-2.379	-3.538	-3.785	-7.795	-5.127	-7.496
22. ك	-0.555	-3.326	-2.908	-3.184	-6.260	-5.263	-6.673
23. ل	-0.593	-2.310	-2.887	-3.461	-6.675	-4. 647	-7.022
24. م	-0.559	-2.138	-3.542	-3.528	-7.096	-4.756	-7.486
25. ن	-0.599	-2.786	-4.052	-3.820	-7.893	-5.213	-7.921
26. ه	-0.613	-2.247	-3.563	-3.458	-6.969	-4.582	-10.576
27. و	-0.591	-2.733	-3.418	-3.525	-6.998	-4.899	-8.024
28. ى	-0.565	-2.551	-3.486	-3.396	-6.944	-4.763	-7.040

Table 9 Center letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1. أ	-0.665	-2.492	-4.110	-4.410	-8.986	-5.784	-8.727
2. ب	-0.583	-2.119	-4.323	-3.742	-7.781	-4.804	-8.523
3. ت	-0.619	-2.817	-3.500	-3.824	-7.500	-5.240	-8.102
4. ث	-0.617	-3.105	-3.389	-3.780	-7.368	-5.333	-8.272
5. ج	-0.634	-3.835	-3.906	-3.848	-7.737	-5.773	-8.343
6. ح	-0.648	-4.933	-3.880	-4.046	-8.177	-6.562	-8.144
7. خ	-0.598	-2.720	-3.208	-3.594	-7.217	-5.004	-7.092
8. د	-0.608	-2.502	-3.442	-3.400	-6.821	-4.652	-8.839
9. ذ	-0.627	-2.391	-3.893	-3.961	-7.891	-5.162	-8.843
10. ر	-0.592	-2.566	-3.856	-3.965	-8.470	-6.350	-7.889
11. ز	-0.626	-3.549	-4.125	-3.946	-8.837	-6.428	-7.986
12. س	-0.618	-6.421	-3.202	-3.083	-6.228	-6.354	-7.184
13. ش	-0.564	-4.976	-3.197	-3.289	-7.434	-6.854	-6.536
14. ص	-0.564	-2.644	-3.537	-3.328	-7.429	-4.867	-6.770
15. ض	-0.637	-2.636	-3.975	-3.923	-8.276	-5.318	-7.909
16. ط	-0.587	-3.794	-3.562	-3.263	-7.299	-5.296	-6.688
17. ظ	-0.573	-4.349	-3.748	-3.386	-7.019	-6.038	-7.246

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
18. ع	-0.589	-3.814	-3.294	-3.327	-6.646	-5.239	-7.320
19. غ	-0.590	-5.355	-3.289	-3.541	-7.001	-6.798	-7.320
20. ف	-0.576	-2.450	-3.147	-3.444	-6.742	-4.670	-7.714
21. ق	-0.600	-3.150	-3.535	-4.023	-7.936	-5.638	-7.969
22. ك	-0.578	-3.338	-3.202	-3.543	-7.351	-5.730	-6.948
23. ل	-0.583	-2.756	-3.093	-3.397	-6.712	-4.788	-6.923
24. م	-0.634	-3.868	-3.596	-3.499	-7.049	-5.471	-7.995
25. ن	-0.517	-1.842	-2.778	-3.191	-6.187	-4.115	-6.827
26. ه	-0.479	-1.734	-4.462	-3.969	-8.282	-4.914	-8.406
27. و	-0.589	-2.994	-4.075	-4.657	-9.468	-7.446	-9.053
28. ى	-0.591	-3.390	-4.289	-3.756	-7.839	-5.478	-8.081

CHAPTER 4

NEURAL NETWORKS

Neural networks is a computational paradigm which solves problems with massive interconnections of simple processors, and several models have been proposed for classification problems such as discriminating between underwater sonar returns, forming text-to-phoneme rules, and so on.

Classifiers determine which of M classes is most representative of an unknown static input pattern containing N input elements. In an image classifier the inputs might be the gray scale level of each pixel for a picture and the classes might represent different objects. Unlike the traditional classifier, in which, strong assumptions are typically made concerning underlying distributions of the input elements, the neural net classifier makes no assumptions concerning the shape of underlying distributions but focuses on errors that occur where distributions overlap. It may thus be more robust than classical techniques and work well when inputs are generated by nonlinear processes and are heavily skewed.

Input values are fed in parallel to the first stage via N input connections. Each connection carries an analog value. After classification is complete, only that output corresponding to the most likely class will be "high"; other outputs will be "low". If the correct class is provided, then this information and the classifier outputs can be fed back to the first stage of the classifier to adapt weights using a learning algorithm. Adaptation will make a correct response more likely for succeeding input patterns that are similar to the current pattern.

Single Layer Perceptron:

A perceptron that decides whether an input belongs to one of two classes (denoted A or B) is shown in Figure 3.

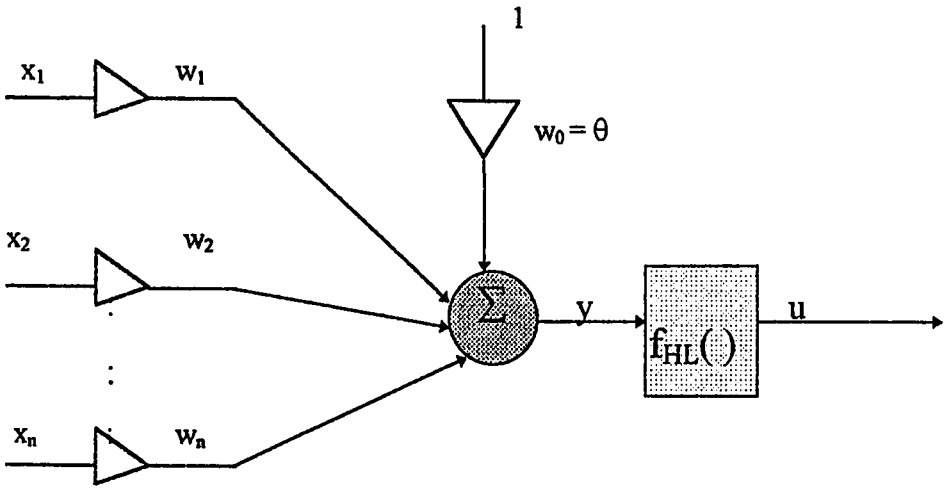

Figure 3 Perceptron

The perceptron forms two decision regions separated by a hyperplane. In this case inputs above the boundary line lead to class A responses and inputs below the line lead to class B responses.

The single node computes a weighted sum of the input elements, subtracts a threshold (θ) and passes the result through a hard limiting nonlinearity such that the output y is either +1 or -1. The decision rule is to respond class A if the output is +1 and class B if the output is -1.

Rosenblatt [77] developed the first perceptron convergence procedure for adjusting weights. First the weights and the threshold value are initialized to small random non-zero values. Then a new input with N continuous valued elements is applied to the input and the output is computed. Connection weights are adapted only when an error occurs.

Rosenblatt [77] proved that if the inputs presented from the two classes are separable (that is they fall on opposite sides of some hyperplane), then the procedure converges and positions the decision hyperplane between those two classes. This decision boundary separates all samples from the A and B classes. One problem with the perceptron convergence procedure is that decision boundaries may oscillate continuously when inputs are not separable and distributions overlap. The least mean square (LMS) is a modification to the perceptron convergence procedure which

minimizes the mean square error between the desired output of a perceptron-like net and the actual output [80,81,71].

The LMS algorithm is identical to the perceptron convergence procedure except the hard limiting nonlinearity is made linear or replaced by a threshold-logic nonlinearity. Weights are thus corrected on every trial by an amount that depends on the difference between the desired and the actual input. A classifier that uses the LMS training algorithm could use desired outputs of 1 for class A and 0 for class B. During operation the input would then be assigned to class A only if the output was above 0.5.

The decision regions formed by perceptrons are similar to those formed by maximum likelihood Gaussian classifiers which assume inputs are uncorrelated and distributions for different classes differ only in mean values. The perceptron training algorithm makes no assumptions concerning the shape of underlying distributions but focuses on errors that occur where distributions overlap. It may thus be more robust than classical techniques and work well when inputs are generated by nonlinear processes and are heavily skewed and non-Gaussian. The Gaussian classifier makes strong assumptions concerning underlying distributions and is more appropriate when distributions are known and match the Gaussian assumption. The adaptation algorithm defined by the perceptron convergence procedure is simple to implement and doesn't require storing any more information than is present in the weights and the threshold. The Gaussian classifier can be made adaptive [73], but extra information must be stored and the computations required are more complex.

Multi-Layer Perceptron:

When classes cannot be separated by a hyper plane, the perceptron convergence procedure is not appropriate. Distributions for the two classes for the exclusive OR problem are disjoint and cannot be separated by a single straight line. This problem was used to illustrate the weakness of the perceptron by Minsky and Papert [75]. Multi-layer perceptrons overcome many of the limitations of single-layer perceptrons

Multi-layer perceptrons are feed-forward nets (No weight feed back) with one or more layers of nodes between the input and output nodes as seen in Figure 4 These additional layers contain hidden units or nodes that are not directly connected to both the input and output nodes. The capabilities of multi-layer perceptrons stem from the nonlinearities used within nodes. If nodes were linear elements, then a single-layer net with appropriately chosen weights could exactly duplicate those calculations performed by any multi-layer net. From intersections of the half-plane regions formed by each node in the first layer of the multi-layer perceptron, a two-layer perceptron can form any, possibly unbounded, convex region in the space spanned by the inputs. Each node in the first layer behaves like a single-layer perceptron and has a "high" output only for points on one side of the hyperplane formed by its weights and offset. A logical AND operation in the output node is performed and results in a decision region

that is the intersection of all the half-plane regions formed in the first layer. These convex regions have at the most as many sides as there are nodes in the first layer.

A three-layer perceptron can form arbitrarily complex decision regions and can separate the meshed classes. It can form regions as complex as those formed using mixture distributions and nearest-neighbor classifiers [71].

Kolmogorov [74] states that any continuous function of N variables can be computed using only linear summations and nonlinear but continuously increasing functions of only one variable. It effectively states that a three layer perceptron with $N(2N+1)$ nodes using continuously increasing nonlinearities can compute any continuous function of N variables. A three-layer perceptron could thus be used to create any continuous likelihood function required in a classifier.

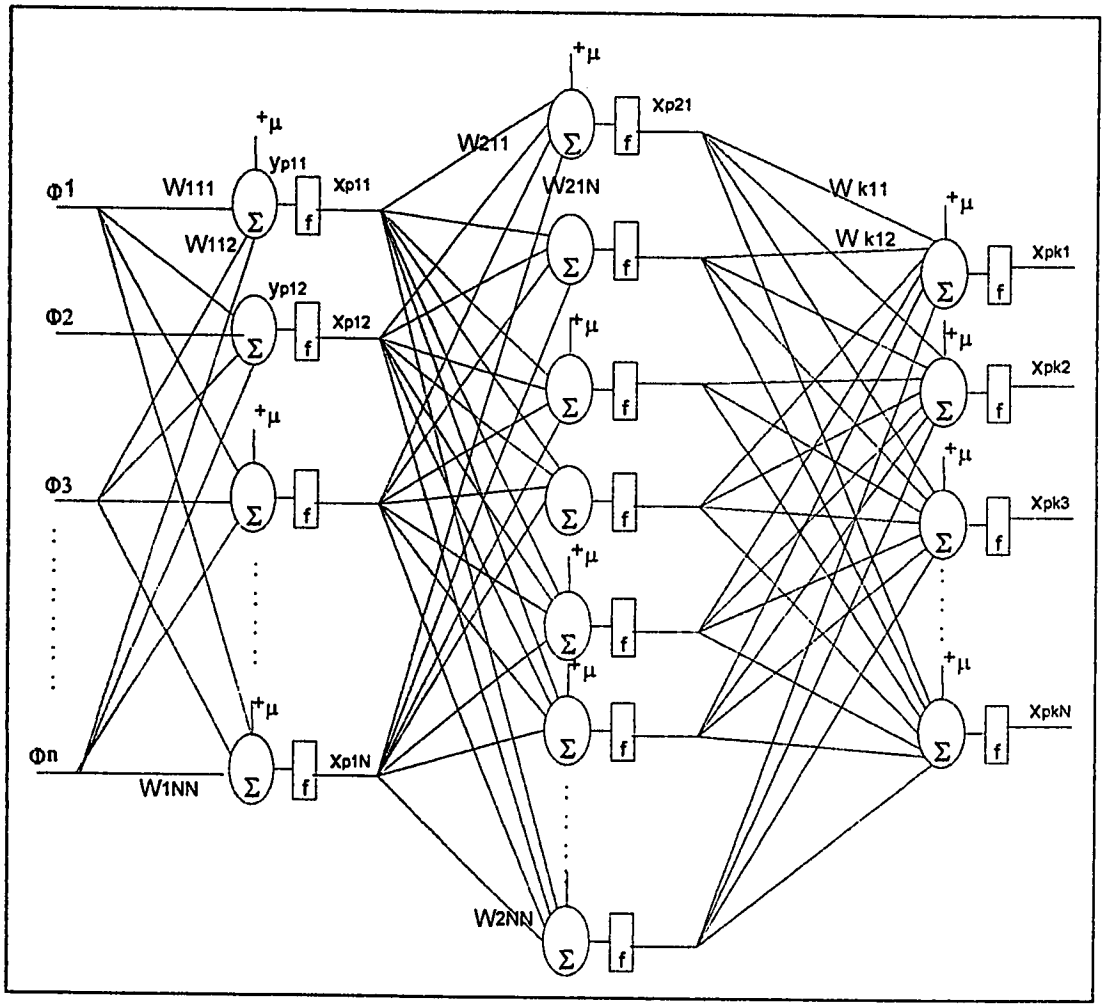

Figure 4 Multi-layer Perceptron.

where w_{ijk} is the weight of layer i from node k at layer $(i-1)$ to node j at layer i.

f is a nonlinear function .

The number of second layer nodes required in the worst case is equal to the number of disconnected regions in input distributions. The number of nodes in the first layer must typically be sufficient to provide three or more edges for each convex area

generated by every second-layer node. There should thus typically be more than three times as many nodes in the second as in the first layer.

TRAINING

Although it cannot be proven that the training algorithms converge as with single layer perceptrons, they have been shown to be successful for many problems [78].

The problem of finding a set of weights for a fixed-size networks which performs the desired mapping exactly for some training set is known as the *loading problem.* It has been shown that the loading problem is NP-complete [72, 84]. So if we have a very large problem, e.g., if the dimension of the input space is very large, then it is unlikely that we will be able to determine if a weight solution exists in a reasonable amount of time. Learning algorithms like back-propagation are based on a gradient search, which is a greedy algorithm that seeks out a local minimum and thus may not yield the exact mapping. The back-propagation algorithm is a generalization of the LMS algorithm. It uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and the actual net outputs. The desired output of all nodes is typically "low" unless that node corresponds to the class the current input is from in which case it is "high". The net is trained by initially selecting small random weights and internal thresholds and then presenting all training data repeatedly. Weights are adjusted after every trial using side information

specifying the correct class until weights converge and the cost function is reduced to an acceptable value. An essential component of the algorithm is the iterative method that propagates error terms required to adapt weights back from nodes in the output layer to nodes in lower layers.

It is well known that finding local weight solutions using back-propagation is extraordinarily slow. Attempts to speed learning include variations on simple gradient search [83,86], line search methods [82], and second-order methods[79,85].

However, they usually introduce additional parameters which are difficult to determine, must be varied form one problem to another, and if not chosen properly can slow the rate of convergence.

Kalman Filter Algorithm:

Robert Scalero [89] reduces the problem to a system of linear equations that relates the weight vector W_{jk} to the summation outputs y_{pjk} and the node inputs $x_{p,j-1,k}$.

The inputs to first layer nodes and outputs from output layer nodes are specified by the training pattern and its associated desired response o_{pk}, respectively and the desired summation outputs d_{pjk} for output layer nodes are also known through the inverse function of the desired response o_{pk}. However, none of the node I/O are known and must be estimated.

The estimate of the desired summation outputs d_{pjk} is

$$d_{pjk} = y_{pjk} + \mu e_{pjk}. \quad (1)$$

Where μ is the step size and e_{pjk}. Is the error signal which is for the output layer is

$$e_{pLk} = f'(y_{pLk})(o_{pk} - x_{pLk}).$$

and for hidden layer

$$e_{pLk} = f'(y_{pLk}).\sum_{i}(e_{p,j\_1,i}.w_{j+1,i,k}).$$

When using these estimates in the system of linear equations, the solution of the weight vector w_{jk} at each node will not be an exact one. It is then necessary to update the desired summation output estimates d_{pjk} with respect to the error produced at the output of the network. As training patterns are applied to the network, these estimates improve and consequently, so do the weight vectors.

The entire training set is run through the network and the error signals are calculated and used to update the desired summation output estimates d_{pjk}. At this point, new weights are calculated by using the system of linear equations at each node. The training set is then run through the network again and the process is continued until convergence is reached.

It is not necessary to wait for the entire training set to be run through the network before an update is performed, but that updates may be performed for each training pattern (iteration).

Formulation:

The problem is to minimize, with respect to the summation outputs y_p, the total mean-squared error E given by

$$E = \sum_{p=1}^{M} (d_p - y_p)^2 \tag{2}$$

where M is the number of training patterns.

Minimizing this error by taking partial derivatives of E with respect to each weight and equating them to zero. The result will be a set of $N+1$ linear equations where $N+1$ is the number of weights in the neuron. Minimizing the error E with respect to a weight w_n produces

$$\frac{\partial E}{\partial w_n} = -2\sum_{p=1}^{M} (d_p - y_p)\frac{\partial y_p}{\partial w_n} = 0 \tag{3}$$

for $n = 0$ through N. The summation output y_p is the inner product of the input vector x_p and weight vector w.

$$y_p = \sum_{i=0}^{N} w_i . x_{pi}$$

Taking the partial derivative of the summation output y_p with respect to a weight w. Equation (3) can be written as

$$\sum_{p=1}^{M} x_{pn} w^T x_p = \left[\sum_{p=1}^{M} x_{pn} x_p^T w \right] \tag{4}$$

for $n = 0$ through N.

Rewrite the right-hand side of equation (4) as :

$$\sum_{p=1}^{M} \sum_{i=0}^{N} w_i x_{pi} x_{pn} = \sum_{p=1}^{M} x_{pn} \sum_{i=0}^{N} w_i x_{pi}$$

Then changing the right summation to a vector multiplication:

$$\sum_{p=1}^{M} x_{pn} \mathbf{W}^T x_p = \left[\sum_{p=1}^{M} x_{pn} x_p \mathbf{W} \right] \tag{5}$$

for $n = 0$ through N.

Defining

$$R = \sum_{p=1}^{M} x_p x_p^T \tag{6}$$

and

$$p = \sum_{p=1}^{M} d_p x_p^T \tag{7}$$

equation (4) can be expressed in matrix form

$$p = R\,w\,. \tag{8}$$

We can solve equation (8) for w by using one of the standard techniques for solving a system of linear equations

$$w = R^{-1} p\,.$$

Instead of running all the training patterns through the network before each update is performed, this poses no problem when the training set consists of a small number of patterns. Scalero [89] modified the correlations in equations (6) and (7) and selected the training patterns to the network randomly.

Redefine equations (6) & (7) as

$$R(t) = \sum_{k=1}^{t} x(k)x^T(k)$$

$$p(t) = \sum_{k=1}^{t} d(k)x(k) \tag{9}$$

$$R(t) = \sum_{k=1}^{t} x(k)x^T(k)$$

$$p(t) = \sum_{k=1}^{t} d(k)x(k) \tag{9}$$

where t is the number of the present iteration starting at $t = 1$.

Equation (9) are estimates of the correlation matrix and correlation vector, respectively, and they improve with increasing time. The weight at each node can now be updated at every iteration (training pattern) since one does not have to wait M training patterns for these estimates to be available.

Robert Scalero [89] inserted a forgetting factor b to overcome a problem that can be described as follows with the exception of the first layer, estimates of one layer are based upon data that layer receives from the previous layer. Since the previous layer is untrained at the beginning, the correlation estimates are not as good as they will be at or near the end of the training period. These inaccurate older estimates, however, are still strongly included in the estimation process later on when the network has gotten smarter. This would, at best, increase the training time of the network.

then equation (9) became

$$R(t) = \sum_{k=1}^{t} b^{t-k} x(k)x^T(k)$$

$$P(t) = \sum_{k=1}^{t} b^{t-k} d(k)x(k) \tag{10}$$

Where b is a constant less than 1.

The forgetting factor will allow the new information to dominate while the correlation estimates from earlier training will have negligible effect on the current estimates.

The recursive form of equation (10) is

$$R(t) = b\,R(t-1) + x(t)\,x^{T}(t)$$

$$P(t) = b\,P(t-1) + d(t)\,x(t). \qquad (11)$$

Specifying the system correlations this way eliminates the problem associated with large numbers of training patterns. In fact, for very large training sets, convergence is often achieved long before all the training patterns have been presented to the network. To obtain a recursive equation for the inverse autocorrelation matrix $R^{-1}(t)$, Scalero [89] used the Kalman filter [87], [88] after making some modifications. Thus, for the j^{th} layer we have the following equations:

The Kalman gain vector:

$$k_j(t) = \frac{\boldsymbol{R}_j^{-1}(t-1)\boldsymbol{x}_{j-1}(t)}{b_j + \boldsymbol{x}_{j-1}^{T}\boldsymbol{R}_j^{-1}(t-1)\boldsymbol{x}_{j-1}(t)} \qquad (12)$$

The update equation for the inverse matrix:

$$\boldsymbol{R}_j^{-1}(t)=\left[\boldsymbol{R}_j^{-1}(t-1)-\boldsymbol{k}_j(t)\boldsymbol{x}_{j-1}^T\boldsymbol{R}_j^{-1}(t-1)\right]/b_j \tag{13}$$

The update equation for the weight vectors in the output layer:

$$w_{Lk}(t) = w_{Lk}(t - 1) + k_L(t)(d_k - y_{Lk}) \tag{14}$$

And the update equation for the weight vectors in the hidden layers:

$$w_{jk}(t) = w_{jk}(t - 1) + k_j(t)e_{jk}(t)\mu_j \tag{15}$$

where t is the present iteration number, and k is the node in the layer. Constants b_j and μ_j are the forgetting factor and back-propagation step size, respectively, in the j^{th} layer.

The algorithm:

1) Initialization

- Randomize all weights in the network.
- Initialize the inverse matrix $\boldsymbol{R}^{-1}$.

•Equate the node offset $x_{j-1,0}$ of every node to some nonzero constant for layers j = 1 through L.

2)Select training pattern.

Randomly select an input/output pair to present to the network. The input vector is x_0 and desired network output vector is o.

3) Run selected pattern through the network.

For each layer j from 1 through L, calculate the summation output

$$y_{jk} = \sum_{i=0}^{N} (x_{j-1,i} \cdot w_{jki}) \tag{16}$$

and the function output

$$x_{jk} = f(y_{jk}) = \frac{1 - exp(-a.y_{jk})}{1 + exp(-a.y_{jk})} \tag{17}$$

for every node k. N is the number of inputs (not including the offset) to a node, and constant a is the sigmoid slope.

4) Invoke Kalman filter equations.

For each layer j from 1 through L, calculate the Kalman gain

$$k_j = \frac{R_j^{-1} x_{j-1}}{b_j + x_{j-1}^T R_j^{-1} x_{j-1}} \tag{18}$$

and update the inverse matrix

$$R_j^{-1} = \left[R_j^{-1} - k_j x_{j-1}^T R_j^{-1} \right] / b \tag{19}$$

5)Backpropagate error signals.

Compute the derivative of $f(y_{jk})$ using

$$f'(y_{jk}) = \frac{2a.\exp(-a.y_{jk})}{\left[1 + \exp(-a.y_{jk})\right]^2} \tag{20}$$

Calculate error signals in the output layer, where $j = L$, by evaluating

$$e_{Lk} = f'(y_{Lk})\,(o_k - x_{Lk}) \tag{21}$$

for every node k.

For the hidden layers, starting at layer $j = L - 1$ and decrementing through $j = 1$, find error signals by solving

$$e_{jk} = f'(y_{jk}) \sum_{i} (e_{j+1,i} w_{j+1,i,k}) \tag{22}$$

for every node in the j^{th} layer.

6) Find the desired summation output.

Calculate the desired summation output at the layer by using the inverse function

$$d_k = \frac{1}{a} ln \frac{1+o_k}{1-o_k} \tag{23}$$

for every k^{th} node in the output layer.

7)Update the weights.

The weight vectors in the output layer L are updated by

$$w_{Lk} = w_{Lk} + k_L (d_k - y_{Lk}) \tag{24}$$

for every k^{th} node.

For each hidden layer j = 1 through L - 1, weight vectors are updated by

$$w_{jk} = w_{jk} + k_j e_{jk} \mu_j . \tag{25}$$

for every k^{th} node.

8)Stopping criteria

- Use the mean-square error of the network output as a convergence test or
- Run the algorithm for a fixed number of iterations .

This algorithm was tested on different sets of data using a program developed in the C programming language (see Appendix A). The error function is defined as the sum of the square of the error. The iterations are terminated when the sum reaches a value smaller than 0.01. The result for Arabic letters is shown in Figure 2 for separated, beginning, middle, and end characters, (one example for each distinct output).

CHAPTER 5

IMPROVED TRAINING ALGORITHM

Numerical experience with the Kalman algorithm, Eq. (4-13), indicates that it is sensitive to the effects of computer roundoff and is susceptible to an accuracy degradation due to the differencing of positive terms. For example, suppose that two numbers that agree to four digits are differenced and if the original numbers were only known to, say, six-digit accuracy, then the difference will have only two-digit accuracy.

Potter [91] recognized that numerical accuracy degradation is often accompanied by a computed covariance matrix that loses its positive definiteness (Note: A symmetric matrix X ,with dimension n × n, is positive definite if and only if $a^T X a > 0$ for all n vectors a with $||a|| > 0$). By reformulating the Kalman algorithm in terms of a square root covariance matrix we can preserve nonnegativity of the computed covariance (which is only an implicit quantity in the algorithm). It turns out that introduction of the covariance square root does indeed improve numeric accuracy and

stability. Examples illustrating the algorithm's numerical deterioration are discussed in [92].

From matrices elementary, if matrix X can be written as $X = Y \times Y$ with Y is a square matrix, we say that Y is a square root of X. It was proved [92] that every positive definite matrix has a square root

In our problem, set

$$R = S \times S^{T} \qquad \mathcal{R} = \check{S} \times \check{S}^{T} \tag{1}$$

where $\mathcal{R}$ is the postriory estimate of the inverse of the covariance matrix

$$R = \sum_{p=1}^{M} x_p x_p^{T}$$

S is the square root of the inverse of the covariance matrix R.

$\check{S}$ is the square root of the postriory estimate of the inverse of the covariance matrix R

Then the kalman equation for the estimate of the covariance matrix R:

$$\mathcal{R} = R - K.x.R$$

Where K is the gain of the kalman filter

will become

$$\Re = \breve{S}.\breve{S}^T = S.S^T - S.S^T.x^T.F^{-1}.S.S^T$$
$$\Re = S.\left[I - v.F^{-1}.v^T\right].S^T \qquad (2)$$

where

$$v = (xS)^{\mathrm{T}} \qquad \text{and} \qquad F = v^{\mathrm{T}}v + 1 \qquad (3)$$

From equation (2) one can define

$$\breve{S} = S.(I - vF^{-1}v^{\mathrm{T}})^{1/2} \qquad (4)$$

From matrices elementary Householder [93] { $I - vF^{-1}v$ } could be factored as

$$I - vF^{-1}v^{\mathrm{T}} = (I - \alpha vv^{\mathrm{T}})^2 \qquad (5)$$

where α is a root of the quadratic equation

$$(v^{\mathrm{T}}v)\,\alpha^2 - 2\alpha + F^{1} = 0 \qquad (6)$$

From matrices elementary, The roots of equation $I - vF^{-1}v^{\mathrm{T}} = (I - \alpha vv^{\mathrm{T}})^2$ are

$\alpha = \dfrac{1}{\sqrt{F}(\sqrt{F} \pm 1)}$ and to avoid cancellation problems we choose

$$\alpha = \frac{1}{\sqrt{F}(\sqrt{F}+1)} \tag{7}$$

When equation (5) is substituted into equation (4) one obtains

$$\breve{S} = S - \frac{\sqrt{F}}{\sqrt{F}+1} K v^T \tag{8}$$

$$K = S\, v / F \tag{9}$$

Equations (8) & (9) can be used with the update equation for the weight vectors in the output layer:

$$w_{Lk}(t) = w_{Lk}(t - 1) + k_L(t)(d_k - y_{Lk}) \tag{10}$$

and the update equation for the weight vectors in the hidden layers:

$$w_{jk}(t) = w_{jk}(t - 1) + k_j(t) e_{jk}(t) \mu_j \tag{11}$$

where t is the present iteration number, and k is the node in the layer.

correlation estimate are not as good as they will be near the end of the training. These bad estimate are still strongly included in the estimation process later on. To overcome this problem we have to insert a forgetting factor b so equations (3) & (8) will be

$$F = \mathbf{v}^{\mathrm{T}}\mathbf{v} + b \tag{12}$$

$$\breve{\mathbf{S}} = \left\{\mathbf{S} - \frac{\sqrt{F}}{\sqrt{F}+1}\mathbf{K}\mathbf{v}^{T}\right\}b^{-1} \tag{13}$$

Where b is < 1.

The algorithm:

1) Initialization:

- Randomize all weights in the network.
- Initialize the matrix S by small non-zero random number.
- Equate the node offset $x_{j-1,0}$ of every node to some nonzero constant for layers $j = 1$ through L.

2)Select training pattern.

Randomly select an input/output pair to present to the network. The input vector is $\mathbf{x}_0$ and desired network output vector is $\mathbf{o}$.

2)Select training pattern.

Randomly select an input/output pair to present to the network. The input vector is $\mathbf{x_0}$ and desired network output vector is o.

3) Run selected pattern through the network.

For each layer j from 1 through L, calculate the summation output

$$y_{jk} = \sum_{i=0}^{N} (x_{j-1,i} \cdot w_{jki}) \tag{14}$$

and the function output

$$x_{jk} = f(y_{jk}) = \frac{1 - exp(-a.y_{jk})}{1 + exp(-a.y_{jk})} \tag{15}$$

for every node k. N is the number of inputs (not including the offset) to a node, and constant a is the sigmoid slope.

4) Invoke filter equations.

For each layer j from 1 through L, calculate the Kalman gain

$$v_j^T = X_j S_j \tag{16}$$

The predicted residual variance inverse will be

$$\sigma_j = 1 / (v_j^T v_j + b_j) \tag{17}$$

Then the unweighted kalman gain will be

$$K_j = S_j v_j \tag{18}$$

5) Update the inverse matrix

Calculate:

$$\gamma_j = \frac{\sigma_j}{1+\sqrt{\sigma_j}} \tag{19}$$

Then

$$S_{j+1} = [S_j - (\gamma_j K_j) v_j^T] / b_j \tag{20}$$

6)Backpropagate error signals.

Compute the derivative of $f(y_{jk})$ using

$$f'(y_{jk}) = \frac{2a.exp(-a.y_{jk})}{\left[1+exp(-a.y_{jk})\right]^2} \tag{21}$$

Calculate error signals in the output layer, where $j = L$, by evaluating

$$e_{Lk} = f'(y_{Lk})\,(o_k - x_{Lk}) \tag{22}$$

for every node k.

For the hidden layers, starting at layer $j = L - 1$ and decrementing through $j = 1$, find error signals by solving

$$e_{jk} = f'(y_{jk}) \sum_i (e_{j+1,i} w_{j+1,i,k}) \tag{23}$$

for every node in the j^{th} layer.

7) Find the desired summation output.

Calculate the desired summation output at the L^{th} layer by using the inverse function

$$d_k = \frac{1}{a} ln \frac{1+o_k}{1-o_k} \tag{24}$$

for every k^{th} node in the output layer.

8)Update the weights.

The weight vectors in the output layer L are dated by

$$W_{Lk} = W_{Lk} + k_L (d_k - y_{Lk}) \tag{25}$$

for every k^{th} node.

9)Stopping criteria :

- Use the mean-square error of the network output as a convergence test or
- Run the algorithm for a fixed number of iterations or
- Split the data into two sets: a training set which used to train the network, and test set which is used to measure the performance of the network. If the error in the test set is below required threshold then stop.

10)If convergence is not reached go back to step 2.

We have implemented this algorithm using the C language and carried out test on different data sets.

To illustrate the immunity of this method against accumulation of round off error, we tested the program on a fixed set of data (27 samples with 7 input each and used one hidden layer with 16 nodes). The plot of the results for both algorithms as shown in Fig 5 and 6.

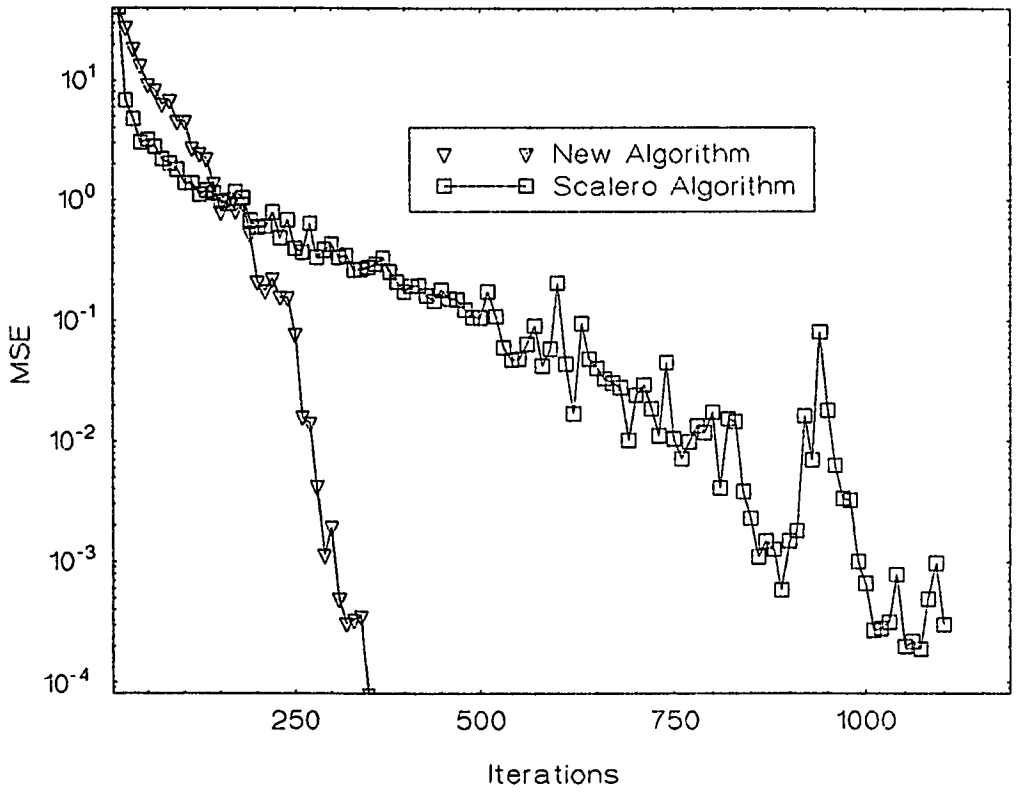

Figure 5 Training time for both algorithms for random data.

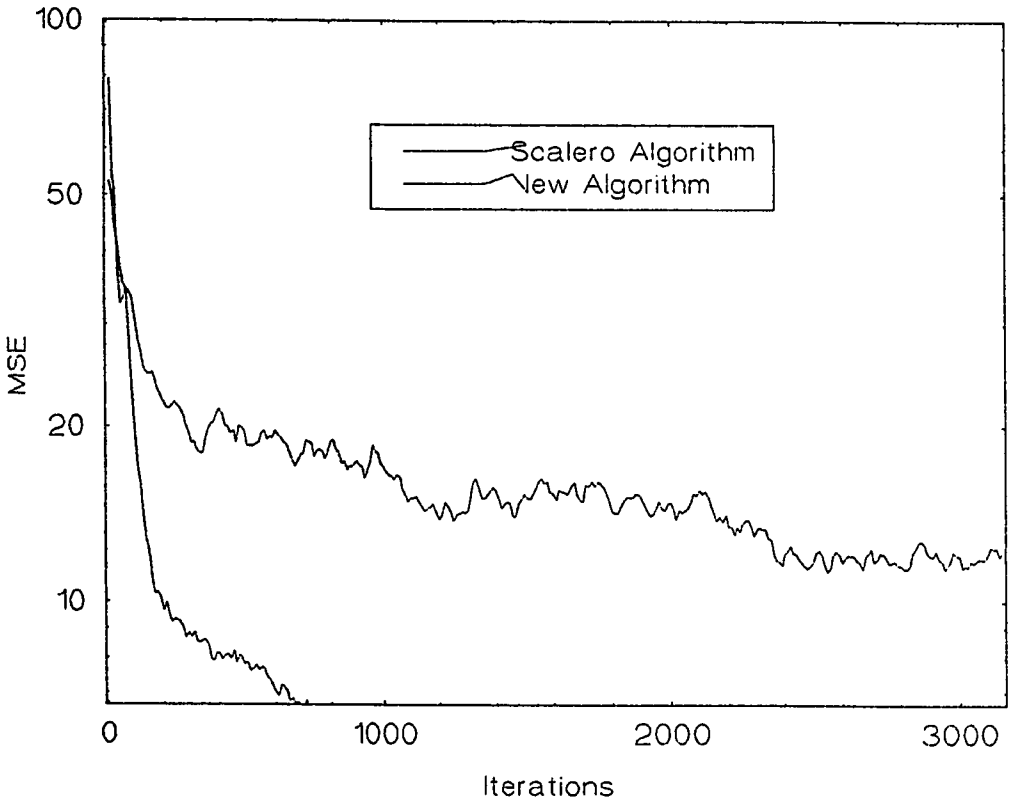

Figure 6 Training time for both algorithms for random data.

We noticed that when the number of iterations gets larger the accumulation of the round-off error gets larger and the difference between the two algorithms becomes obvious. Also, we notice that while we need to run the first algorithm several times due to abnormal program termination: divide by zero error (instability), this never happened in the second algorithm. Also the first algorithm goes more frequently to the unstable case when the forgetting factor b gets smaller, while the second algorithm does not have this problem and allows us to adjust the forgetting factor b to the best value. We summarized the differences in Table 10.

Table 10 comparison between Scalero and the new algorithms

	Scalero Method	The New Method
Forgetting Factor	Very limited change region	Any value
Effect of dynamic range	Becomes unstable for moderate and large dynamic range	Unaffected
Stability	Frequently Becomes Unstable	Never becomes Unstable

Special configuration used for the 128 Arabic fonts:

An information about the position of the character in the word is available after word segmentation process (position means if the character lies in the beginning, middle, end, or separate), The easiest word segmentation process make segmentation either from a space or a mid-line.

Our system used this available information by constructing four separate Neural-Networks ,as shown in Figure 7 Each one is trained separately by one type of fonts beginning, middle, end, or separate. In the recognition phase one and only one of them are activated by the word segmentation process. If character is segmented via space before it and mid-line after it then this character is beginning character and so on.

Each Neural-Network has 7 nodes in the input layer and 5 or more nodes in the output layer. We have 27 distinct output so 5 nodes in the output layer is enough but we take more for two purposes:

- Correct the errors generated in the word segmentation process by considering each segmentation failure as another character like لم فى ...etc.
- Use redundant bits for errors detection and correction.

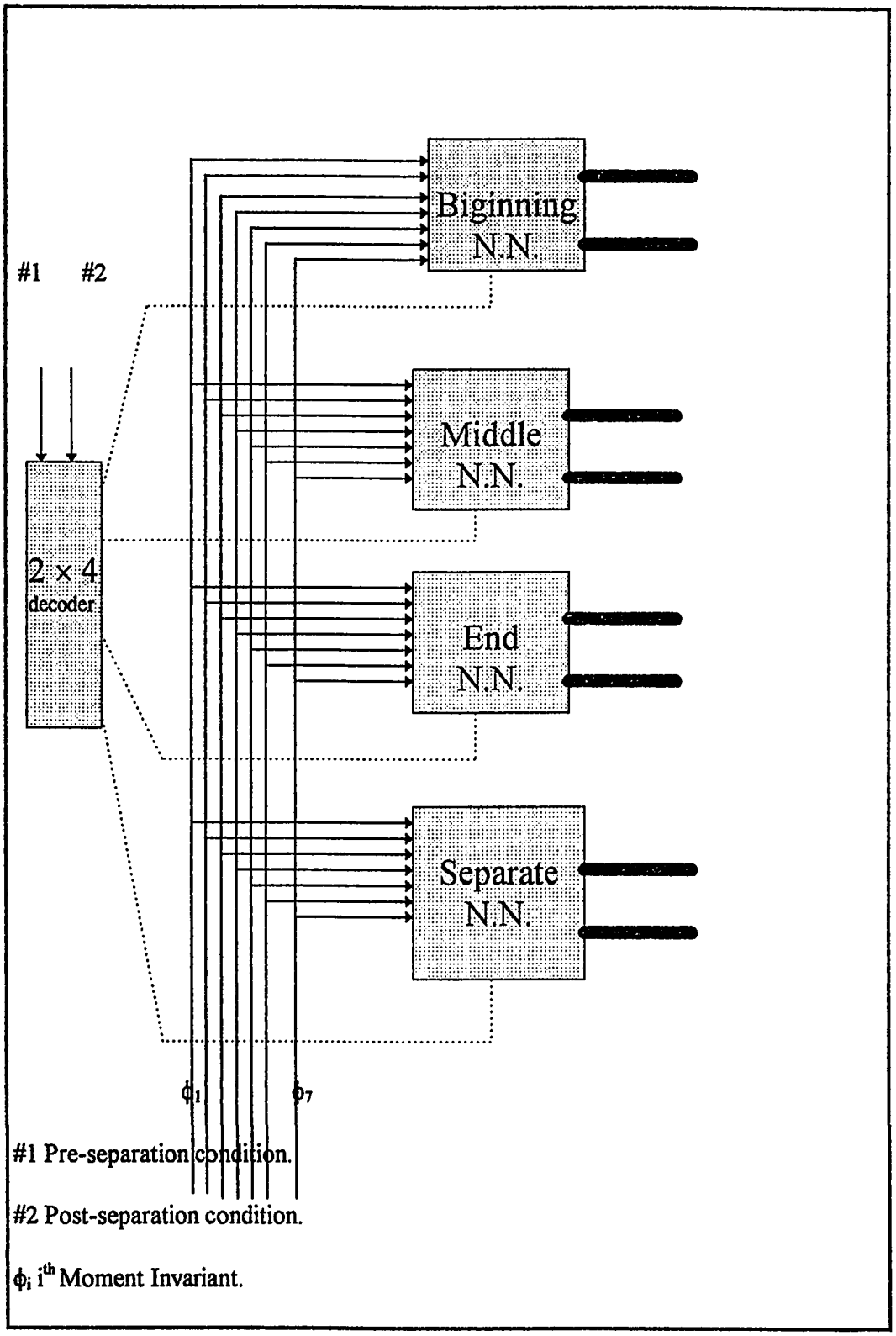

Figure 7 Arabic O.C.R. classifier

CHAPTER 6

RESULTS AND RECOMMENDATIONS

In this chapter we will summarize the results obtained in chapters 3, 4, and 5 We shall also provide outlines for reasons taken for the selection of the various approaches. It is to be pointed out that the intention of this research is to study two main blocks in OCR as pertaining to Arabic fonts; viz.:

1. Feature extraction
2. Classification

Other blocks that could be included but not limited to :

1. graph and picture separation;
2. title and paragraph separation;
3. line and word separation;
4. font separation;
5. dictionary for word verification;

are not studied in this work. The reason is that such a project is normally handled by a large group of researchers, each examining one aspect of the work. We opted to examine the first two blocks and we leave the rest for future research in this area.

1) Feature Extraction:

1-1 Moment Invariants were chosen for our purpose for the following reasons:

- Moment Invariants which are extracted from geometrical moments were proved to be better in noise immunity than other types of moments[95].
- Moment Invariants do not require any type of preprocessing unlike Fourier Descriptor and others.
- Moment Invariants are more flexible: Allow rotation and shift in character image.
- Moment Invariants depend on global shape, basic shape and diacritics, (more suitable for Arabic characters, as mentioned in El-khaly [93]).

1-2 Special modification needed for Moment Invariants are:

- Moment Invariants were made invariant to size by replacing Central moments by Normalized moments as discussed in chapter 3.
- As was indicated in chapter 3, the Moment Invariants can have a large dynamic range. Therefore, the logarithm of the absolute value of the

Moment Invariants was taken in place of the Moment itself. $\psi(i) = \log |\Phi(i)|$

1-3 To decrease the effect of digitization on the properties of moment invariants (scale, shift, rotation) the following were done:

- To relax the computation (avoid dealing with huge numbers), the x and y scale was divided by a scale factor (10 was chosen for our case) as mentioned in chapter 3 (a fixed number for all sizes).
- The Simpson rule was chosen since it is a better approximation to the double integration as suggested in [94]. Comparison results are found in chapter 3.

1-4 To facilitate the work of Neural networks: we noticed that each moment invariant has an offset value that is large with respect to the variations in the numbers. This is obvious if you would examine Table 6, Table 7, Table 8 Ending letters, and Table 9 Center letters for example shows the following number for $\Phi(7)$

-8.727,-8.523,-8.102,-8.272,-8.343,-8.144,-7.092,-8.839,-8.843,-7.889,-7.986,-7.184

for the following letters ا ب ت ث ج ح خ د ذ ر ز س If we were to normalize we would obtain 0.987, 0.9642, 0.9166, 0.936, 0.944, 0.921, 0.8023, 1, 1, 0.8925, 0.903, 0.8128. However if an offset value of -8 is subtracted before normalization we will obtain 0.727,-0.523,-0.102, -0.272, -0.343, -0.144, +0.908, -0.839, -0.843, +0.111, +0.014, +0.816.

An experiment was made to test the effect of removing this offset, a Neural-Network with 11 nodes in the hidden layer was trained by three complete sets of Arabic alphabet for sizes 61×61, 59×59, and 55×55 and a set of Arabic alphabet for size 57×57 was used as a recognition test (The number of hidden nodes is intentionally small to show the effect of removing the offset). A comparison between convergence time and correct recognition are provided in Figure 8, Table 11 respectively. It was found that if we eliminate these offset values an improvement in training and recognition phases is achieved.

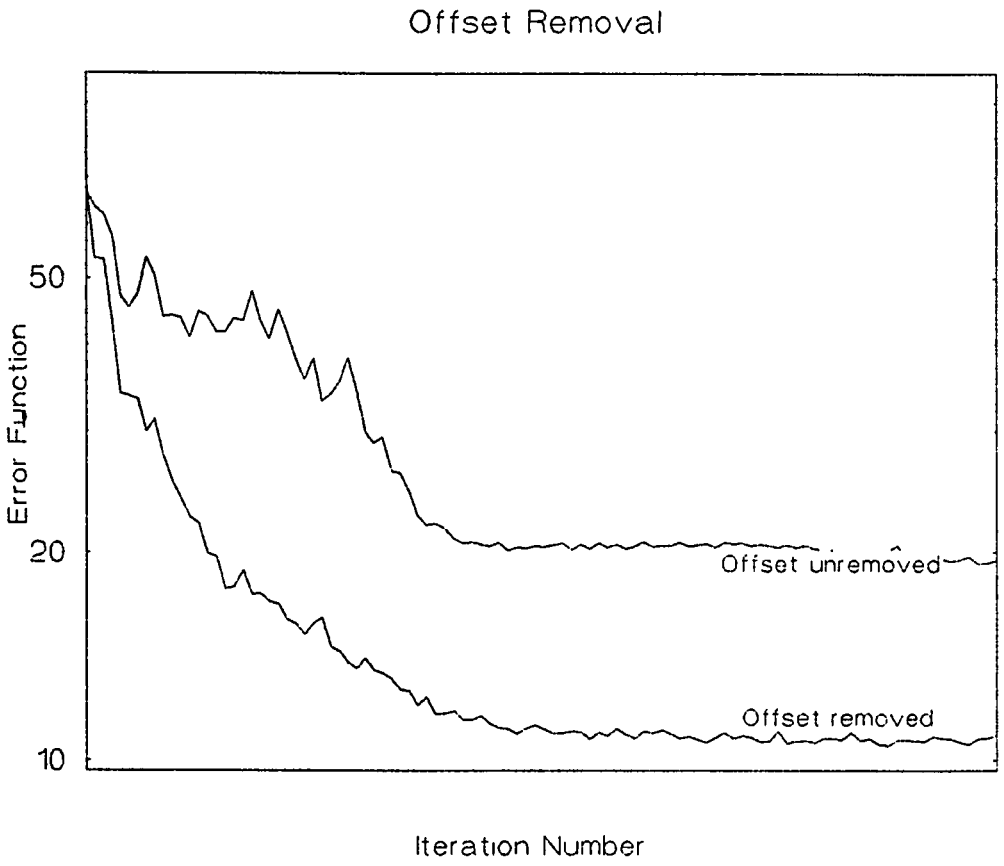

Figure 8 Effect of removing offset on training time.

Table 11 Effect of offset removal		
	Offset not removed	Offset removed
Number of erroneous bits	13 (2%)	6 (0.93%)
Number of erroneous letters	11 (10%)	5 (4.6%)
Number of detected erroneous letters	9 (82%)	4 (80%)

The offset component was subtracted from the moment invariants as follows:

$\Phi 1 = \Phi 1 + 0.6$.

$\Phi 2 = \Phi 2 + 2$.

$\Phi 3 = \Phi 3 + 4$.

$\Phi 4 = \Phi 4 + 4$.

$\Phi 5 = \Phi 5 + 8$.

$\Phi 6 = \Phi 6 + 6$.

$$\Phi 7 = \Phi 7 + 8 .$$

After this process it was found that almost all moment invariant have a range between +1 and -1 for all letters.

The complete data set is found in Appendix A for all letters.

2) Classification:

We choose the feed-forward neural network for the following reasons:

- Recognition phase requires less computation time than other classifiers such as the K-mean.
- Outperform other classifiers, especially when noise is present [96].

2-1 The Configuration of the selected Feed-Forward Neural Network is as follows:

- Number of layers = 3: It was proved in [97] that a 3 layer feed-forward neural networks can divide the N-dimensional input space into any arbitrary shape regions.
- Number of nodes in the input layer = 7 (seven moments).
- Number of nodes in the hidden layer = 21: In [98] Shih-Chi proved that m neurons in the hidden layer can divide the N-dimensional input space into N m-polytopes where $(m+1) \leq N \leq P(m, n)$ where $P(m, n)$ is the permutation of m with n. So for $N = 32$ distinct output the number of hidden layer is between $31 \geq m \geq 9$ (31 will be, for sure, sufficient). In order to accelerate the calculations in the recognition phase, we tried the following number of nodes in the hidden layer: 21, 15, 11, and 9. The results are attached. Note that Lippman [99]recommended that we should have 3 times as many nodes in the second (hidden) layer as in the first (input) layer.

- Number of nodes in the output layer = 6:
    1 Five nodes represent the binary code of the letter from 0 to 32.
    2 One node acts as a parity check that we introduced to detect any single error in one character during the recognition phase.
- The input to the first layer was the seven moment invariants for each letter, while the output to the last layer was forced to be +0.5 for " 1 " or -0.5 for " 0 ". The reason for such a selection is to force the values obtained from the sigmoid function to lie in the linear region in order to accelerate the training phase

2-3 Training Phase:

As discussed in chapter 5, The improved method was implemented and used for its superior convergence.

An extensive study about the value of the forgetting value (b) appropriate in our character recognition system was performed. A value of b of 0.99, 0.98, 0.97, 0.96, 0.95, 0.94 was tested. All of these values of b gave an exponential decay as shown in Figure 9, which has a behavior of relatively fast decay at the beginning for the most recent values and then slow decay after that. We require a function to have a large forgetting value at the beginning of the iteration when the layers are untrained and the value of the weights matrix and the correlation estimate matrix are completely random, and to have a small forgetting value at or near the end of the iteration when

the layers are trained and the weight matrix and the correlation estimate matrix have good estimate. After a study, an incremental forgetting factor b was chosen starting from 0.94 to 0.98 with step 0.00003. It exhibits the required behavior as seen in Figure 10. A comparison between this function and a usual constant value of b of 0.94, 0.98, and our method in convergence of training of the Arabic character set was done and the results are illustrated in Figure 10.

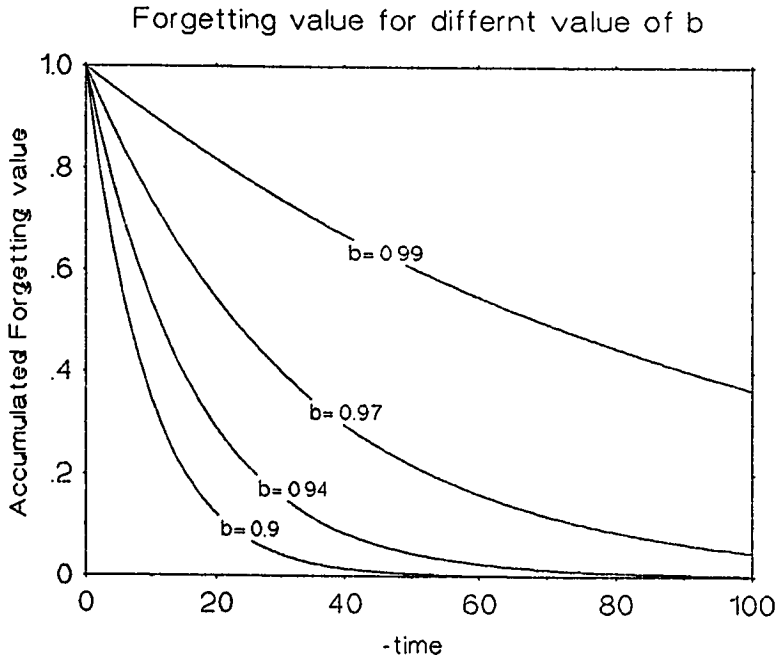

Figure 9 Forgetting function for various values of b.

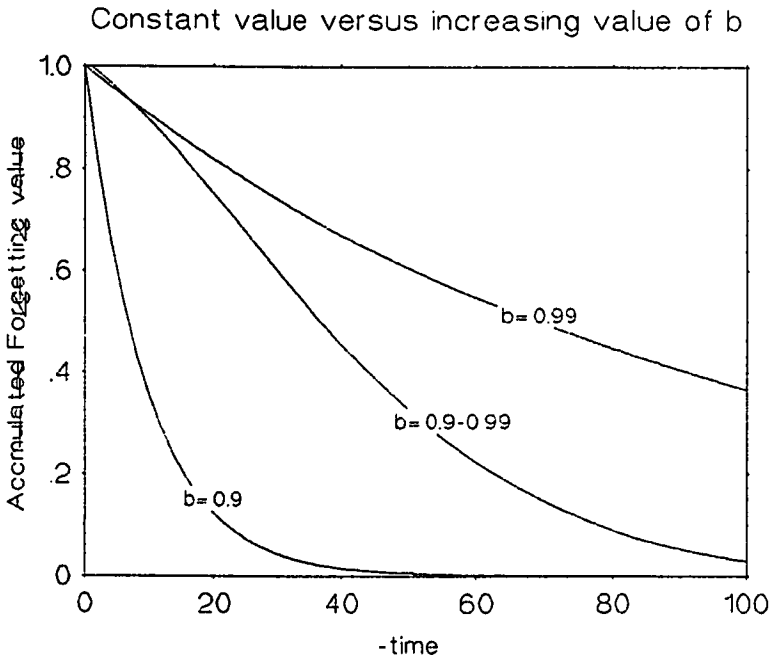

Figure 10 Forgetting function for constant b versus variable b.

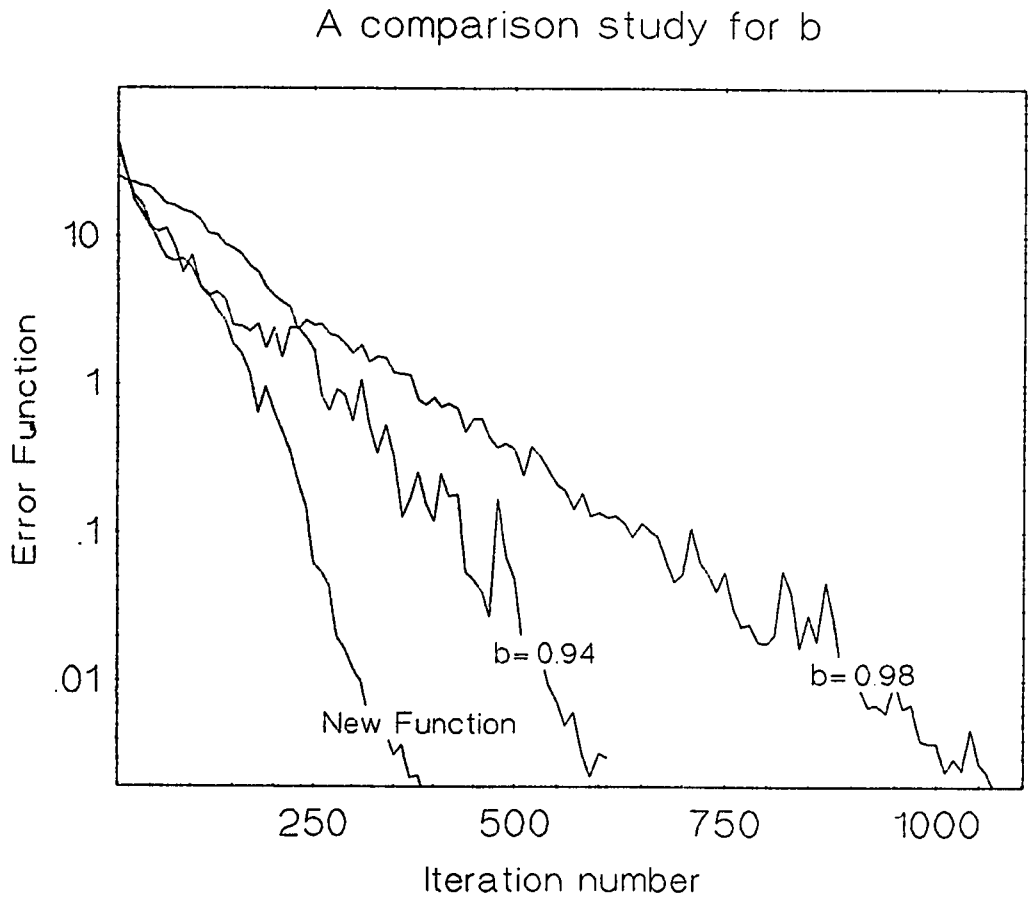

Figure 11 Training time for constant b versus variable b.

The Neural-Network was trained by three complete sets of Arabic alphabet style (نسخ) for different sizes 61×61, 59×59, 55×55 (Size changes were chosen because it has the most effect on the values of the moment invariants due to digitization effect. The number of node of the hidden layer was tried as 26, 21, 15, 11, 9.

2-4 Recognition Phase:

- A program to implement the feed-forward phase and to detect any possible occurrence of single error (per character) was developed.
- Different set of Arabic alphabet for size 57×57 (neither included in the training set nor in the stopping criteria) was used
- The results are tabulated in Table 12.

Table 12 Effect of changing # of nodes in the hidden layer					
Number of nodes in the hidden layer	26	21	15	11	9
Number of erroneous bits	0 0%	0 0%	0 0%	2 0.3%	9 1.3%
Number of erroneous letters	0 0%	0 0%	0 0%	2 1.8%	8 7.4%
Number of detected erroneous letters	0 0%	0 0%	0 0%	2 100%	7 88%

Recommendations and Proposal for Future Work:

The process of Arabic OCR involves a number of distinct steps which are outlined in the thesis. We examined 2 of these steps viz.:

1. Feature extraction
2. Classification using the feed-forward neural networks.

However, this does not present a complete solution to the Arabic OCR problem.

The following are recommended:

1. A study of other feature extraction methods that depend on topology of the characters such as vectors, strokes, loops, etc. rather than some mapping scheme,.
2. An examination of other classifiers.
3. Search for solutions to the other modules that form a complete solution to the OCR problem, such as graph and picture separation .....etc.

APPENDICES

TABLE (1)
beginning letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1.أ	-0.065	-0.421	-1.835	-1.731	-4.340	-1.740	-3.518
2.ب	-0.010	-0.410	-0.147	-0.272	-0.583	-0.289	-0.695
3.ت	-0.031	-0.198	0.539	-0.134	-0.663	0.373	0.062
4.ث	-0.070	-0.831	0.239	-1.059	-1.592	-1.269	-1.651
5.ج	0.032	-1.758	-0.183	-0.317	-0.572	-0.279	-1.429
6.ح	0.065	-1.344	-0.161	-1.829	-3.267	-2.086	-2.855
7.خ	-0.034	-0.905	0.298	-0.122	-0.115	0.264	-0.289
8.د	0.033	-0.519	-0.646	-0.766	-1.484	-0.181	-2.103
9.ذ	-0.044	-2.104	-0.264	0.018	-0.401	-0.142	-0.170
10. ر	-0.011	-0.321	0.115	0.212	0.376	0.612	-0.881
11. ز	-0.066	-1.378	-1.410	-0.581	-1.603	-0.551	-2.036
12. س	-0.030	-0.757	0.531	0.858	1.388	1.388	1.416
13. ش	0.011	-1.008	0.773	0.930	1.264	1.378	1.760
14. ص	0.006	-0.879	0.155	0.715	0.870	0.072	1.080
15. ض	-0.039	-1.853	0.130	0.173	0.095	0.247	0.232
16. ط	-0.001	-0.491	0.368	0.318	0.382	1.045	0.591
17. ظ	0.031	-0.399	0.113	-0.026	0.014	0.102	-0.888
18. ع	0.015	-1.002	0.065	0.074	0.077	0.304	-0.147
19. غ	-0.014	-1.736	-0.006	0.026	-0.187	-0.118	-0.060
20. ف	0.001	-0.697	0.436	0.124	0.313	0.718	0.172
21. ق	0.002	-0.640	0.334	-0.256	-0.233	0.069	-0.791
22. ك	0.014	-0.718	0.703	0.210	0.368	-0.380	0.603
23. ل	-0.036	-0.609	0.689	-0.379	-1.018	0.314	-0.230
24. م	0.042	-0.704	0.612	0.649	1.261	1.282	0.736
25. ن	-0.012	-0.277	0.876	0.041	0.482	0.323	-0.060
26. ه	0.128	-0.062	1.089	1.145	2.144	2.015	2.074
27. و	-0.015	-1.604	-0.129	-0.828	-1.326	-0.671	-1.847
28. ي	0.021	-1.371	-0.334	0.636	0.364	0.670	0.754

TABLE (2)
Center letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1.أ	-0.062	-0.821	-1.142	-0.883	-1.931	-0.419	-2.307
2.ب	-0.065	-2.140	-0.958	-0.082	-0.735	-0.408	-0.772
3.ت	-0.012	-1.884	-0.602	-0.731	-1.402	-0.677	-2.225
4.ث	-0.015	-3.071	0.285	-0.104	-0.052	-0.751	-0.407
5.ج	-0.046	-1.872	0.144	0.170	0.291	-0.065	-0.078
6. ح	-0.018	-2.486	-0.180	-0.089	-0.350	-0.371	-0.401
7. خ	-0.032	-0.370	-0.670	-1.050	-1.916	-0.257	-2.696
8. د	-0.017	-0.588	-0.801	0.263	-0.391	0.577	-0.047
9. ذ	-0.043	-1.448	-0.828	0.129	-0.290	-0.058	-0.502
10. ر	-0.049	-0.586	-0.765	0.121	-0.332	0.528	-0.373
11. ز	-0.087	-0.668	-1.773	-0.072	-1.076	0.153	-1.246
12. س	-0.046	-1.483	0.436	0.426	0.853	0.417	-0.069
13. ش	-0.004	-0.348	0.667	0.479	0.388	1.211	1.041
14. ص	-0.014	-0.905	0.018	-0.021	-1.611	0.482	-0.023
15. ض	-0.045	-0.761	-0.485	-0.593	-1.362	-0.299	-1.225
16. ط	-0.031	-1.733	-0.902	-0.524	-1.452	-0.726	-1.339
17. ظ	0.024	-1.803	-0.125	0.403	0.155	-0.755	0.501
18. ع	-0.052	-1.838	-0.019	0.114	0.112	0.047	-0.183
19. غ	-0.068	-2.400	-1.115	-0.408	-1.192	-0.625	-1.672
20. ف	-0.057	-1.351	-0.096	0.438	0.194	0.638	0.575
21. ق	-0.052	-1.173	-1.016	-0.055	-0.600	0.299	-1.271
22. ك	0.035	-0.658	0.097	0.723	0.953	0.785	1.009
23. ل	-0.065	-1.286	-0.506	-0.441	-1.509	-0.276	-0.929
24. م	-0.069	-0.648	-1.836	-0.020	-1.072	-0.502	-1.129
25. ن	0.004	-0.437	-0.447	-0.069	-0.442	0.338	-0.517
26. ه	-0.085	-0.600	-1.064	-0.363	-1.936	-0.362	-1.080
27. و	-0.052	-0.548	-0.849	-0.539	-1.306	-0.468	-1.504
28. ي	0.068	-0.381	0.540	0.320	0.746	1.130	-0.088

TABLE (3)
End letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1. أ	-0.043	-0.661	0.625	-0.543	-0.594	0.113	-0.734
2. ب	-0.041	-1.618	-1.427	-0.234	-1.099	-0.047	-1.484
3. ت	-0.067	-0.651	0.610	-0.165	-0.036	0.450	-0.171
4. ث	-0.032	-2.375	0.530	-0.156	-0.029	-0.376	-0.279
5. ج	0.046	-1.396	-0.355	0.255	0.204	0.386	-0.942
6. ح	0.020	-1.519	0.445	0.799	0.560	1.017	1.416
7. خ	0.016	-0.593	-0.679	-2.005	-3.382	-1.636	-3.758
8. د	-0.032	-1.287	0.635	0.772	1.300	0.564	1.346
9. ذ	0.011	-0.144	0.671	0.870	1.304	1.697	1.588
10. ر	0.039	-0.616	0.099	0.296	-0.016	0.864	0.471
11. ز	0.008	-0.747	0.505	0.835	0.452	0.928	1.503
12. س	0.085	0.078	1.011	-0.436	-0.317	0.433	-0.282
13. ش	0.020	-0.736	1.036	0.553	1.216	1.150	1.176
14. ص	0.067	-0.037	0.652	0.566	1.174	1.492	-0.006
15. ض	-0.007	-0.917	-0.104	-0.121	-0.376	0.391	-0.393
16. ط	-0.013	-1.585	-0.089	0.628	0.896	-0.400	-0.075
17. ظ	0.025	-0.721	0.553	0.592	1.162	1.215	0.238
18. ع	0.022	-0.935	-1.586	-0.180	-2.084	0.348	-1.064
19. غ	0.029	-0.205	-0.473	-0.460	-1.219	0.110	-0.992
20. ف	-0.072	-0.565	0.119	0.093	0.168	0.509	-0.239
21. ق	0.012	-0.480	0.677	0.607	-0.616	1.159	1.249
22. ك	0.061	-0.550	1.148	0.932	1.593	0.328	1.930
23. ل	0.007	-0.592	1.229	0.700	1.321	1.307	1.615
24. م	0.050	-0.282	0.489	0.395	0.402	0.192	0.806
25. ن	0.016	-1.302	0.074	0.526	0.710	0.875	0.634
26. ه	-0.012	-0.454	0.510	0.697	1.300	1.469	-0.051
27. و	0.017	-1.137	0.819	0.724	1.473	1.145	0.991
28. ي	0.049	-0.636	0.626	0.885	1.637	1.540	0.737

TABLE (4)
Separated letters

letter	Φ(1)	Φ(2)	Φ(3)	Φ(4)	Φ(5)	Φ(6)	Φ(7)
1. أ	-0.076	-0.719	0.199	-0.268	-1.739	0.054	-0.303
2. ب	0.045	-0.176	0.079	0.677	1.054	1.589	-0.273
3. ت	-0.014	-1.370	0.585	0.251	0.670	0.566	-1.112
4. ث	-0.008	-1.927	0.784	0.312	0.852	-0.088	0.144
5. ج	-0.022	-2.085	0.183	0.206	0.353	-0.306	0.046
6. ح	-0.031	-1.597	0.340	0.148	0.341	0.339	0.052
7. خ	0.005	-1.183	1.040	0.438	0.977	0.832	1.068
8. د	-0.011	-0.743	0.503	0.638	1.169	1.230	0.820
9. ذ	-0.016	-0.611	0.213	0.138	0.200	0.760	0.119
10. ر	0.018	-0.800	0.267	0.345	0.299	0.551	0.602
11. ز	-0.014	-2.677	-0.192	0.472	0.042	-0.408	0.596
12. س	-0.011	-2.023	0.816	0.970	1.831	0.957	1.432
13. ش	0.038	-1.911	0.776	0.790	1.400	0.080	1.444
14. ص	0.059	-0.168	0.708	1.022	1.142	1.562	1.880
15. ض	-0.019	-0.224	0.257	0.297	0.466	0.819	0.372
16. ط	0.022	-1.370	0.428	0.790	0.892	0.702	1.378
17. ظ	0.047	-1.270	0.470	0.526	0.660	0.048	0.978
18. ع	0.037	-0.779	0.982	0.899	1.834	1.507	1.026
19. غ	0.022	-1.649	0.763	0.518	1.155	0.693	0.246
20. ف	0.034	-0.708	0.963	0.690	1.500	1.334	0.948
21. ق	-0.002	-2.756	0.650	-0.018	0.294	-0.397	-0.574
22. ك	0.040	-3.929	1.088	0.622	0.840	-1.455	1.465
23. ل	0.033	-1.826	1.166	0.656	1.567	0.727	-0.263
24. م	-0.021	-0.948	0.599	0.665	1.263	1.189	0.874
25. ن	0.105	0.116	1.485	1.165	2.422	2.211	2.204
26. ه	0.157	0.256	-0.985	-0.133	-1.181	0.742	-0.716
27. و	0.016	-1.278	0.011	-0.088	-0.868	-0.064	-0.134
28. ي	0.041	-1.483	-0.019	0.496	0.731	0.755	-0.150

This program calculates the weights of a Neural Network for any number of inputs and any number of outputs and any number of layers with any number of neurons Kalman Filter Method


```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include<time.h>
#define a 0.2          /* segmind slope */
#define b 0.9          /* Forgetting value */
#define st_size .85
#define no_iter 100000
#define ERROR  0.0001
#define fd(i)  (1.0-y[i]*y[i])*a/2   /* Define derivative.*/
float fun(float *w,float *xp,int *d);   /* function to simulate the feed-forward part */
```

```
int  M,*NL,*NS,L;
float *y,theta,*wi;

void main()
{
    int  z, m, n, Nt1, Nt2, Nt3, ii, *d, I, j, N, xd, ind, Nt, N1, N2, qq;
    float *w, q, xt, error, x, xrx, sum, *R, *dd, *kl, *kL, *delta, *xp, *net, ne;
    double E;
    long int iter = 0;
    char file_name[18],file_name2[18],ch;
    FILE *fptr,*fptr2;

    clrscr();
    printf("\nDo you wish to use previously trained weights? (y or n)-->");
    while(((ch=getch())!='y')&&(ch!='n'));
    putch(ch);
    switch(ch)
    {
     case 'y':
          printf("\nEnter file name -->");
          scanf("%s",file_name);
```

```
fptr=fopen(file_name,"r");
if(fptr==NULL)
   {       printf("No such file exists.");
           exit(1);
    }          .
fscanf(fptr,"%d ",&L);
if(!(NL=(int *)malloc(L*sizeof(int))))
{
           printf("Out of memory.NL\n");
           exit(1);
   }
if(!(NS=(int *)malloc((L-2)*sizeof(int))))
{
           printf("Out of memory.NS\n");
           exit(1);
   }
for(i=0;i<L;i++)  fscanf(fptr,"%d ",&NL[i]);
NS[0]=NL[0]*NL[1];
for(i=1;i<(L-2);i++)  NS[i]=NS[i-1]+NL[i]*NL[i+1];
N=NS[L-3]+NL[L-2]*NL[L-1];    /* Total of weights. */
/* Assigning memory for weights. */
```

```
        if(!(w=(float *)malloc(N*sizeof(float))))
            {
                printf("Out of memory.w\n");
                exit(1);
            }
        for(i=0;i<N;i++)  fscanf(fptr,"%f ",&w[i]);
        fscanf(fptr,"%f ",&theta);
        fclose(fptr);
        break;
case 'n':
        printf("\nEnter number of hidden layers-->");
        scanf ("%d", &L) ;
        L+=2; /*adding input and output layers. */
        if(!(NL=(int *)malloc(L*sizeof(int))))
        {
                printf("Out of memory.NL\n");
                exit(1);
            }
        if(!(NS=(int *)malloc((L-2)*sizeof(int))))
        {
                printf("Out of memory.NS\n");
```

```
            exit(1);
    }
printf("Enter number of nodes in input layer-->");
scanf("%d",&NL[0]);
for(i=1;i<=(L-2);i++)
{
  printf("Enter number of nodes in hidden layer %d-->",i);
  scanf("%d",&NL[i]);
}
printf("Enter number of nodes in output layer-->");
scanf("%d",&NL[L-1]);
NS[0]=NL[0]*NL[1];
for(i=1;i<(L-2);i++)  NS[i]=NS[i-1]+NL[i]*NL[i+1];
N=NS[L-3]+NL[L-2]*NL[L-1]; /* Total # of weights.
/* Assigning memory for weights. */
if(!(w=(float *)malloc(N*sizeof(float))))
    {
            printf("Out of memory.w\n");
            exit(1);
    }
randomize();
```

```
        for(i=0;i<N;i++)
            w[i]=(float)random(N)/(float)N;
        theta=0.1;
 }
gotoxy(1,10);
printf("Enter file name for storing trained weights-->  ");
scanf("%s",file_name);
ind=access(file_name,0);
while(!ind)
{
 gotoxy(1,12);
 printf("File exists.  Wish to overwrite? (y or n)-->");
 while(((ch=getch())!='y')&&(ch!='n'));
 putch(ch);
 switch(ch)
  {
     case 'y':
    ind=1;
    break;
     case 'n':
    gotoxy(1,7);
```

```
        printf("                                    ");
        gotoxy(1,10);
        printf("                                    ");
        gotoxy(1,10);
        printf("Enter file name-->");
        scanf("%s",file_name);
        ind=access(file_name,0);
      }
  }
for(i=1,Nt=0;i<L;i++) Nt+=NL[i];   /* Total number of neurals.*/
/* Assigning memory to *net, *z, *delta.*/
if(!(net=(float *)malloc(NL[L-1]*sizeof(float))))
    { printf("Out of memory.net\n");exit(1); }
if(!(y=(float *)malloc(Nt*sizeof(float))))
    {
    printf("Out of memory.y\n");
    exit(1);
    }
if(!(delta=(float *)malloc(Nt*sizeof(float))))
    {
    printf("Out of memory.delta\n");
```

```
    exit(1);
    }

for(i=0,Nt=0;i<L-1;i++)  Nt+=NL[i];
if(!(kl=(float *)malloc(Nt*sizeof(float))))
    {
    printf("Out of memory.kl\n");
    exit(1);
    }
if(!(kL=(float *)malloc(Nt*sizeof(float))))
    {
    printf("Out of memory.kL\n");
    exit(1);
    }
for(i=0,N1=0;i<L-1;i++)  N1+=NL[i]*NL[i];
if(!(R=(float *)malloc(N1*sizeof(float))))
    {
    printf("Out of memory.R\n");
    exit(1);
    }
randomize();
```

```
for(i=0;i<N1;i++)
   R[i]=(float)random(N1)/(float)N1;
 printf("\nEnter file name for stored Data--> ");
 scanf("%s",file_name2);
 fptr2=fopen(file_name2,"r");
if(fptr2==NULL)
   {printf("file %s does not exist. ",file_name2);exit(1);}
/* Determining the size of the data.*/
 M=0; ind=1;
while(1)
{
 for(i=0;i<NL[0];i++)
 {
   if((fscanf(fptr2,"%f ",&xt))==EOF) /* input data.*/
    {
   ind=0;
   break;
    }
 }
if(ind==0) break;
for(i=0;i<NL[L-1];i++)
```

```
    fscanf(fptr2,"%d ",&xd);  /* desired output.*/
M++;
}
printf("\n# of data points=%d\n",M);
rewind(fptr2);
/* Assigning memory to *xp, *d  */
if(!(xp=(float *)malloc((M*NL[0])*sizeof(float))))
    {
     printf("Out of memory.xp\n");
    exit(1);
    }
if(!(d=(int *)malloc((M*NL[L-1])*sizeof(int))))
    {
    printf("Out of memory.d\n");
    exit(1);
    }
if(!(dd=(float *)malloc((M*NL[L-1])*sizeof(float))))
    {
    printf("Out of memory.dd\n");
    exit(1);
    }
```

```
/* Reading in the data.*/
 for(i=0; i<M; i++)
 {
  for(j=0;j<NL[0];j++)    fscanf(fptr2,"%f ",&xp[j*M+i]);
  for(j=0;j<NL[L-1];j++)
     {
    fscanf(fptr2,"%d ",&d[j*M+i]);
    dd[j*M+i]=(log((0.5+d[j*M+i])/(1.5-d[j*M+i])))/a;
     }
 }
fclose(fptr2);

fptr=fopen(file_name,"w");
clrscr();
gotoxy(1,1);
printf("Press q to exit and save latest update for weights.\n");

while(iter<no_iter)
{
 if(kbhit() !=0) { if(getch()=='q') ;  break;}
 iter ++;
```

```
for(i=0;i<M;i++)
{
    qq=random(M);
    for(z=0,N2=0;z<L-1;z++)   N2 += NL[z];

    /* Forward propagation. */
    wi=w;
    for(z=0;z<NL[1];z++) /* From input layer to first */
    {           /* hidden layer.           */
     for(j=0,ne=theta;j<NL[0];j++)  ne += *(wi++)*xp[j*M + q q];
     E=(double)exp(-(double)a*ne);
     y[z]=(float)(1.0-E)/(1.0+E);
    }
    Nt1=NL[1]; Nt2=0;
    for(n=2;n<L-1;n++)    /* From layer n-1 to layer n.*/
    {
     for(z=0;z<NL[n];z++)
     {
      m=Nt1 + z;
      for(j=0,ne=theta;j<NL[n-1];j++) ne += *(wi++)*y[j + Nt2];
      E=(double)exp(-(double)a*ne);
```

```
    y[m]=(float)(1.0-E)/(1.0+E);
   }
   Nt1+=NL[n];
   Nt2+=NL[n-1];
  }
   for(z=0;z<NL[L-1];z++)
   {
    m=Nt1+z;
    for(j=0,net[z]=theta;j<NL[L-2];j++) net[z]+=*(wi++)*y[j+Nt2];
    E=(double)exp(-(double)a*net[z]);
    y[m]=(float)(1.0-E)/(1.0+E);
   }
  Nt1=0;
  for(z=1;z<(L-1);z++)
    Nt1+=NL[z];
  for(z=0;z<NL[L-1];z++)       /* delta's for output layer.*/
  {
   ii=Nt1+z;
   error=d[qq+z*M]-0.5-y[ii];
   delta[ii]=error*fd(ii);
  }
```

```
for(m=0;m<(L-2);m++)        /* delta's by back propagation.*/
{
 Nt2=Nt1-NL[L-2-m];
 for(z=0;z<NL[L-2-m];z++)
 {
  ii=Nt2+z;
  sum=0.0;
  n=NS[L-3-m]+z;
  for(j=0;j<NL[L-1-m];j++)
  sum+=delta[Nt1+j]*w[n+j*NL[L-2-m]];
  delta[ii]=fd(ii)*sum;
 }
 Nt1=Nt2;
}
for(z=0,xrx=b;z<NL[0];z++)
{
 for(j=0,kl[z]=0.0,kL[z]=0.0;j<NL[0];j++)
  {
  kl[z]+=R[j+z*NL[0]]*xp[qq+j*M];
  kL[z]+=R[z+j*NL[0]]*xp[qq+j*M];
  }
```

```
 xrx+=kl[z]*xp[qq+z*M];
}
for(z=0;z<NL[0];z++)  kl[z]/=xrx;

for(z=0;z<NL[0];z++)
   for(j=0,n=z*NL[0];j<NL[0];j++)
      R[j+z*NL[0]] =(R[j+n]- kL[j]*kl[z])/b;
Nt1=NL[0]*NL[0];  Nt2=NL[0]; Nt3=0;
for(m=1;m<(L-1);m++)
{
 for(z=0,xrx=b;z<NL[m];z++)
 {
    for(j=0,kl[z+Nt2]=0.0,kL[z+Nt2]=0.0;j<NL[m];j++)
   {
         kl[z+Nt2]+=R[j+Nt1+z*NL[m]]*y[Nt3+j];
         kL[z+Nt2]+=R[z+Nt1+j*NL[m]]*y[Nt3+j];
   }
   xrx+=kl[z+Nt2]*y[Nt3+z];
 }
for(z=0;z<NL[m];z++)          kl[z+Nt2] /=xrx;
for(z=0;z<NL[m];z++)
```

```
    for(j=0,n=Nt1+z*NL[m];j<NL[m];j++)
            R[n+j] =(R[n+j]- kL[j+Nt2]*kl[z+Nt2])/b;
  Nt1+=NL[m]*NL[m];
  Nt2+=NL[m];
  Nt3+=NL[m];
 }
 wi=w;
 for(z=0;z<NL[1];z++)
    for(j=0;j<NL[0];j++)
            *(wi++) +=kl[j]*delta[z]*st_size;
 Nt1=NL[0],Nt2=NL[1];
 for(n=2;n<L-1;n++)    /* From layer n-1 to layer n.*/
 {
   for(z=0;z<NL[n];z++)
            for(j=0;j<NL[n-1];j++)
                    *(wi++) +=kl[Nt1+j]*delta[z+Nt2]*st_size;
    Nt1+=NL[n-1];
    Nt2+=NL[n];
 }
 for(z=0;z<NL[L-1];z++)
   for(j=0;j<NL[L-2];j++)
```

```
                    *(wi++) +=kl[Nt1+j]*(dd[qq+z*M]-net[z]);
        }
        q=fun(w,xp,d);
        gotoxy(1,8);
        printf(" Error function= %f at iteration # %-5d",q,iter);
        if(q<ERROR)break;
      }
      fprintf(fptr,"%d ",L);
      for(i=0;i<L;i++)   fprintf(fptr,"%d ",NL[i]);
      for(i=0;i<N;i++)   fprintf(fptr,"%f ",w[i]);
      fprintf(fptr,"%f ",theta);
      fclose(fptr);
      printf ("\nError=%f ",q);
      printf("\n File name used to store weights is %s",file_name);
      printf("\n File name for the training data is %s",file_name2);
}

/*  Generating the function. */
float  fun(float *w,float *xp,int *d)
{
    float net,error,q=0.0;
```

```
double E;
int k,j,i,n,m,Nt1,Nt2;
for(k=0;k<M;k++)
{
    wi=w;
    for(i=0;i<NL[1];i++)  /* From input layer to first */
    {               /* hidden layer.          */
        for(j=0,net=theta;j<NL[0];j++)        net+=*(wi++)*xp[j*M+k];
        E=(double)exp(-(double)a*net);
        y[i]=(float)(1.0-E)/(1.0+E);
    }
    Nt1=NL[1]; Nt2=0;

    for(n=2;n<L;n++)   /* From layer n-1 to layer n.*/
    {
      for(i=0;i<NL[n];i++)
      {
        m=Nt1+i;
        for(j=0,net=theta;j<NL[n-1];j++) net+=*(wi++)*y[j+Nt2];
        E=(double)exp(-(double)a*net);
        y[m]=(float)(1.0-E)/(1.0+E);
```

```
            }
            Nt1+=NL[n];
            Nt2+=NL[n-1];
        }
        for(i=0;i<NL[L-1];i++)   /* Calculating the error. */
        {
            error=d[k+i*M]-0.5-y[Nt2+i];
            q+=error*error;
        }
    }/*k-loop*/
    q/=2;
    return q;
}
```

This program reads BMP image file and calculate the seven Moment Invariants

Note: The BMP reader is taken from:

"Super charged bitmaped image" Steve Rimme. Wincrest/McGraw Hill.


```
#include "stdio.h"
#include "dos.h"
#include "alloc.h"
#include "math.h"
#include "f.c"

#defineGOOD_READ        0       /* return codes */
#defineBAD_FILE    1
#defineBAD_READ  2
#defineMEMORY_ERROR  3
```

```
#defineSCREENWIDE       320
            /* mode 13 screen dimensions */
#defineSCREENDEEP       200
#defineSTEP         32    /* size of a step when panning */
#defineHOME         0x4700        /* cursor control codes */
#defineCURSOR_UP 0x4800
#defineCURSOR_LEFT      0x4b00
#defineCURSOR_RIGHT     0x4d00
#defineEND          0x4f00
#defineCURSOR_DOWN      0x5000
#defineRGB_RED          0
#defineRGB_GREEN 1
#defineRGB_BLUE  2
#defineRGB_SIZE  3
#definepixels2bytes(n)((n+7)/8)
#definegreyvalue(r,g,b)     (((r*30)/100)+((g*59)/100)+((b*11)/100))

typedef struct {
      int width,depth,bytes,bits;
      int background;
      char palette[768];
```

```
        int (*setup)();
        int (*closedown)();
        } FILEINFO;

typedef struct {
        char id[2];
        long filesize;
        int reserved[2];
        long headersize;
        long infoSize;
        long width;
        long depth;
        int biPlanes;
        int bits;
        long biCompression;
        long biSizeImage;
        long biXPelsPerMeter;
        long biYPelsPerMeter;
        long biClrUsed;
        long biClrImportant;
        } BMPHEAD;
```

```
char *farPtr(char *p,long l);
char *getline(unsigned int n);
char *mono2vga(char *p,int width);
char *ega2vga(char *p,int width);
char *rgb2vga(char *p,int width);
int dosetup(FILEINFO *fi);
int doclosedown(FILEINFO *fi);
int putline(char *p,unsigned int n);

char masktable[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};

FILEINFO fi;
char *buffer=NULL;

void main(argc,argv)
        int argc;
        char *argv[];
{
        FILE *fp;
        static char results[8][16] = {  "Ok",
```

```
                    "Bad file",
                    "Bad read",
                    "Memory error",
                    };
    char path[81];
    int r;

    if(argc > 1) {
    strmfe(path,argv[1],"BMP");
    strupr(path);
    if((fp = fopen(path,"rb")) != NULL) {
        fi.setup=dosetup;
        fi.closedown=doclosedown;
        r=unpackbmp(fp,&fi);
        printf("\%s",results[r]);
        fclose(fp);
    } else printf("Error opening %s",path);
    } else puts("Argument:      path to a BMP file");
}

/* unpack a BMP file */
```

```
unpackbmp(fp,fi)
        FILE *fp;
        FILEINFO *fi;
{
        BMPHEAD bmp;
        char *p,*pr;
        int i,n;

        /* set a default monochrome palette */
        memcpy(fi->palette,"\000\000\000\377\377\377",6);

        /* get the header */
        if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)==sizeof(BMPHEAD)) {

        /* check the header */
        if(!memcmp(bmp.id,"BM",2)) {

                /* set the details */
                fi->width=(int)bmp.width;
                fi->depth=(int)bmp.depth;
                fi->bits=bmp.bits;
```

```
/* work out the line width */
if(fi->bits==1) fi->bytes=pixels2bytes(fi->width);
else if(fi->bits==4) fi->bytes=pixels2bytes(fi->width)<<2;
else if(fi->bits==8) fi->bytes=fi->width;
else if(fi->bits==24) fi->bytes=fi->width*3;

/* round up to an even dword boundary */
if(fi->bytes & 0x0003) {
        fi->bytes |= 0x0003;
        ++fi->bytes;
}

/* get the palette */
if(fi->bits > 1 && fi->bits <=8) {
        n=1<<fi->bits;
        for(i=0;i<n;++i) {
            fi->palette[(i*RGB_SIZE)+RGB_BLUE]=fgetc(fp);
            fi->palette[(i*RGB_SIZE)+RGB_GREEN]=fgetc(fp);
            fi->palette[(i*RGB_SIZE)+RGB_RED]=fgetc(fp);
            fgetc(fp);
```

```
        }
}
else if(fi->bits==24) {
        for(i=0;i<256;++i)
           memset(fi->palette+(i*RGB_SIZE),i,RGB_SIZE);
}

/* allocate a line buffer */
if((p=malloc(fi->bytes)) != NULL) {
        if((fi->setup)(fi) != GOOD_READ) {
        free(p);
        return(MEMORY_ERROR);
        }

        /* find the start of the image data */
        fseek(fp,bmp.headersize,SEEK_SET);

        /* read all the lines */
        for(i=0;i<fi->depth;++i) {
              if(fread(p,1,fi->bytes,fp) != fi->bytes) {
                    freebuffer();
```

```
        free(p);
        return(BAD_READ);
}

/* translate the line types into VGA */
switch(fi->bits) {
        case 1:
                pr=mono2vga(p,fi->width);
                if(pr != NULL) {
                        putline(pr,fi->depth-1-i);
                        free(pr);
                }
                else {
                        freebuffer();
                        free(p);
                        return(MEMORY_ERROR);
                }
                break;
        case 4:
                pr=ega2vga(p,fi->width);
                if(pr != NULL) {
```

```
                putline(pr,fi->depth-1-i);
                free(pr);
            }
            else {
                freebuffer();
                free(p);
                return(MEMORY_ERROR);
            }
            break;
        case 8:
            putline(p,fi->depth-1-i);
            break;
        case 24:
            pr=rgb2vga(p,fi->width);
            if(pr != NULL) {
                putline(pr,fi->depth-1-i);
                free(pr);
            }
            else {
                freebuffer();
                free(p);
```

```
                                        return(MEMORY_ERROR);
                                    }
                                    break;
                                }
                            }
                            (fi->closedown)(fi);
                            free(p);
                            return(GOOD_READ);
                        } else return(MEMORY_ERROR);
                } else return(BAD_FILE);
                } else return(BAD_READ);
}

/* convert a monochrome line into an eight bit line */
char *mono2vga(p,width)
        char *p;
        int width;
{
        char *pr;
        int i;
```

```
        if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;++i) {
                if(p[i >> 3] & masktable[i & 0x0007])
                  pr[i]=1;
                else
                  pr[i]=0;
        }
        return(pr);
        } else return(NULL);
}

/* convert a four bit line into an eight bit line */
char *ega2vga(p,width)
        char *p;
        int width;
{
        char *pr;
        int i,j=0;

        if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;) {
```

```
            pr[i++]=(p[j] >> 4) & 0x0f;
            pr[i++]=p[j] & 0x0f;
            ++j;
        }
        return(pr);
        } else return(NULL);
}

/* convert an RGB line into an eight bit line */
char *rgb2vga(p,width)
        char *p;
        int width;
{
        char *pr;
        int i;

        if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;++i) {
            pr[i]=greyvalue(p[RGB_RED],p[RGB_GREEN],p[RGB_BLUE]);
            p+=RGB_SIZE;
        }
```

```
        return(pr);
        } else return(NULL);
}

/* this function is called before an image is unpacked */
dosetup(fi)
        FILEINFO *fi;
{
        union REGS r;

        if(!getbuffer((long)fi->width*(long)fi->depth,fi->width,fi->depth))
          return(MEMORY_ERROR);

        r.x.ax=0x0013;
        //int86(0x10,&r,&r);

        //setvgapalette(fi->palette,256,fi->background);

        return(GOOD_READ);
}
```

```
/* This function a called after an image has been unpacked. It must
   display the image and deallocate memory. */
doclosedown(fi)
        FILEINFO *fi;
{
        union REGS r;
        int c,i,j,v,n,e ,x=0,y=0;
        char *aa;
        FILE *mom;
        long double
m00=0,m01=0,m10=0,m11=0,m02=0,m20=0,m30=0,m03=0,m21=0,m12=0;
                                /*normalized central moments*/

        long double f[7],jj,ii,b,ax,ay,vv;  /*invarient moments*/
        e=pow(2,fi->bits)-1;
        n=fi->width;
        for(i=0;i<SCREENDEEP;++i)
        {
                c=y+i;
                if(c>=fi->depth) break;
                aa=getline(c);
```

```
        ii=((long double)i*2.0/fi->depth)-1.0;
        for(j=0; j<n ;j++)
        {
                v=(int)aa[j]/e;
                vv=(long double)v;
                jj=((long double)j*2.0/n)-1.0;
                m00 += vv;
                m10 +=vv*jj;
                m01 +=vv*ii;
        }
}
ax=m10/m00;
ay=m01/m00;
for(i=0;i<SCREENDEEP;++i)
{
        c=y+i;
        if(c>=fi->depth) break;
        aa=getline(c);
        ii=((long double)i*2.0/fi->depth)-1.0-ay;
        for(j=0; j<n ;j++)
        {
```

```
                    v=(int)aa[j]/e;
                    vv=(long double)v;
                    jj=((long double)j*2.0/n)-1.0-ax;
                    m20 +=jj*jj*vv;
                    m30 +=jj*jj*jj*vv;
                    m11 +=ii*jj*vv;
                    m21 +=ii*jj*jj*vv;
                    m02 +=ii*ii*vv;
                    m12 +=ii*ii*jj*vv;
                    m03 +=ii*ii*ii*vv;
                }
        }
b=m00*m00;
m20=m20/b;
m02=m02/b;
m11=m11/b;
b=powl(m00,2.5);
m30=m30/b;
m12=m12/b;
m21=m21/b;
m03=m03/b;
```

```
f[0]=log10l(fabsl(m20+m02))+2;
f[1]=log10l(fabsl((m20-m02)*(m20-m02)+4*m11*m11))+4;
f[2]=log10l(fabsl((m30-3.0*m12)*(m30-3.0*m12)+(3.0*m21-m03)*(3.0*m21-
m03)))+6;
f[3]=log10l(fabsl((m30+m12)*(m30+m12)+(m21+m03)*(m21+m03)))+6;
f[4]=log10l(fabsl((m30-3.0*m12)*(m30+m12)*((m30+m12)*(m30+m12)-
3.0*(m21+m03)*(m21+m03))+(3.0*m21-m03)*(m21+m03)*
(3.0*(m30+m12)*(m30+m12)-(m21+m03)*(m21+m03))))+12;
f[5]=log10l(fabsl((m20-m02)*((m30+m12)*(m30+m12)-(m21+m03)*(m21+m03))
+4*m11*(m30+m12)*(m21+m03)))+8;
f[6]=log10l(fabsl((3.0*m21-m03)*(m30+m12)*((m30+m12)*(m30+m12)-
3.0*(m21+m03)*(m21+m03))+(3.0*m12-m30)*
(m21+m03)*(3.0*(m30+m12)*(m30+m12)-(m21+m03)*(m21+m03))))+13;
for(i=0;i<7;i++) printf("%Lf ",f[i]);
        freebuffer();
        //r.x.ax=0x0003;
        //int86(0x10,&r,&r);
        return(GOOD_READ);
}
```

```
/* get one extended key code */
GetKey()
{
        int c;

        c = getch();
        if(!(c & 0x00ff)) c = getch() << 8;
        return(c);
}

/* set the VGA palette and background */
setvgapalette(p,n,b)
        char *p;
        int n,b;
{
        union REGS r;
        int i;

        outp(0x3c6,0xff);
        for(i=0;i<n;++i) {
        outp(0x3c8,i);
```

```
        outp(0x3c9,(*p++) >> 2);
        outp(0x3c9,(*p++) >> 2);
        outp(0x3c9,(*p++) >> 2);
        }
        r.x.ax=0x1001;
        r.h.bh=b;
        int86(0x10,&r,&r);
}

/* make file name with specific extension */
strmfe(new,old,ext)
        char *new,*old,*ext;
{
        while(*old != 0 && *old != '.') *new++=*old++;
        *new++='.';
        while(*ext) *new++=*ext++;
        *new=0;
}

/* if you don't use in the memory manager, these functions
  will stand in for it */
```

```
#if !MEMMANGR

/* return a far pointer plus a long integer */
char *farPtr(p,l)
        char *p;
        long l;
{
      unsigned int seg,off;

        seg = FP_SEG(p);
        off = FP_OFF(p);
        seg += (off / 16);
        off &= 0x000f;
        off += (unsigned int)(l & 0x000fL);
        seg += (l / 16L);
        p = MK_FP(seg,off);
        return(p);
}

/* save one line to memory */
```

```
putline(p,n)
        char *p;
        unsigned int n;
{
        if(n >= 0 && n < fi.depth)
           memcpy(farPtr(buffer,(long)n*(long)fi.width),p,fi.width);
}

/* get one line from memory */
char *getline(n)
        unsigned int n;
{
        return(farPtr(buffer,(long)n*(long)fi.width));
}

#pragma warn -par
getbuffer(n,bytes,lines)
        unsigned long n;
        int bytes,lines;
{
        if((buffer=farmalloc(n)) == NULL) return(0);
```

```
        else return(1);
}

freebuffer()
{
        if(buffer != NULL) farfree(buffer);
        buffer=NULL;
}
#endif /* !MEMMANGR */
```

This program calculates the weights of a Neural Network

for any number of inputs and any number of outputs

and any number of layers with any number of neurons

Kalman Filter Method


```
float fun(float *w,float *xp,int *d);
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include<time.h>
#define a 0.2            /*segmind slope */
#define b1 0.94
#define st_size 40
#define no_iter 4000
#define ERROR  0.001
#define fd(i)  (1.0-y[i]*y[i])*a/2   /* Define derivative.*/
int  M,*NL,*NS,L;
```

```
float *y,theta,*wi;


void main()
{
 int   z,m,n,Nt1,Nt2,Nt3,ii,*d,st=st_size;
 float *w,q,Q=0,xt,error,x,xrx,sum,*R,*kl,*dd,*v,*delta,*xp,*net,*mx,ne,b=b1;
 double E;
 int i,j,N,xd,ind,Nt,N1,N2,qq;
 long int iter=0;
 char file_name[18],file_name2[18],file_name3[18],ch;
 FILE *fptr,*fptr2,*fptr3;


 clrscr();
 printf("\nDo you wish to use previously trained weights? (y or n)-->");
 while(((ch=getch())!='y')&&(ch!='n'));


 putch(ch);
 switch(ch)
 {
  case 'y':
    printf("\nEnter file name -->");
```

```
scanf("%s",file_name);
fptr=fopen(file_name,"r");

if(fptr==NULL)
    { printf("No such file exists."); exit(1); }
fscanf(fptr,"%d ",&L);
if(!(NL=(int *)malloc(L*sizeof(int))))
{ printf("Out of memory.NL\n");exit(1); }
if(!(NS=(int *)malloc((L-2)*sizeof(int))))
{ printf("Out of memory.NS\n");exit(1); }
for(i=0;i<L;i++)  fscanf(fptr,"%d ",&NL[i]);
NS[0]=NL[0]*NL[1];
for(i=1;i<(L-2);i++) NS[i]=NS[i-1]+NL[i]*NL[i+1];
N=NS[L-3]+NL[L-2]*NL[L-1];   /* Total of weights. */
/* Assigning memory for weights. */
if(!(w=(float *)malloc(N*sizeof(float))))
    {  printf("Out of memory.w\n");  exit(1); }
for(i=0;i<N;i++)   fscanf(fptr,"%f ",&w[i]);
fscanf(fptr,"%f ",&theta);
fclose(fptr);
break;
```

```
case 'n':
 printf("\nEnter number of hidden layers-->");
 scanf ("%d", &L) ;
 L+=2; /*adding input and output layers. */
 if(!(NL=(int *)malloc(L*sizeof(int))))
 { printf("Out of memory.NL\n");exit(1); }
 if(!(NS=(int *)malloc((L-2)*sizeof(int))))
 { printf("Out of memory.NS\n");exit(1); }
 printf("Enter number of nodes in input layer-->");
 scanf("%d",&NL[0]);
 for(i=1;i<=(L-2);i++)
 {
   printf("Enter number of nodes in hidden layer %d-->",i);
   scanf("%d",&NL[i]);
 }
 printf("Enter number of nodes in output layer-->");
 scanf("%d",&NL[L-1]);
 NS[0]=NL[0]*NL[1];
 for(i=1;i<(L-2);i++)  NS[i]=NS[i-1]+NL[i]*NL[i+1];
 N=NS[L-3]+NL[L-2]*NL[L-1]; /* Total # of weights.
```

```
 /* Assigning memory for weights. */
 if(!(w=(float *)malloc(N*sizeof(float))))
       {  printf("Out of memory.w\n");  exit(1); }
 randomize();
 for(i=0;i<N;i++)
      w[i]=(float)random(N)/(float)N;
 theta=0.1;
}

gotoxy(1,10);
printf("Enter file name for storing trained weights-->  ");
scanf("%s",file_name);
ind=access(file_name,0);
while(!ind)
{
 gotoxy(1,12);
 printf("File exists.  Wish to overwrite? (y or n)-->");
 while(((ch=getch())!='y')&&(ch!='n'));
 putch(ch);
 switch(ch)
  {
```

```
        case 'y':
            ind=1;
            break;
        case 'n':
            gotoxy(1,7);
            printf("                                    ");
            gotoxy(1,10);
            printf("                                    ");
            gotoxy(1,10);
            printf("Enter file name-->");
            scanf("%s",file_name);
            ind=access(file_name,0);
      }
}
gotoxy(1,15);
printf("Enter file name for storing output results-->  ");
scanf("%s",file_name3);
ind=access(file_name3,0);
while(!ind)
{
 gotoxy(1,17);
```

```
printf("File exists.  Wish to overwrite? (y or n)-->");
while(((ch=getch())!='y')&&(ch!='n'));
putch(ch);
switch(ch)
 {
      case 'y':
       ind=1;
       break;
      case 'n':
       gotoxy(1,12);
       printf("                                    ");
       gotoxy(1,15);
       printf("                                    ");
       gotoxy(1,15);
       printf("Enter file name-->");
       scanf("%s",file_name3);
       ind=access(file_name3,0);
  }
}

for(i=1,Nt=0;i<L;i++)  Nt+=NL[i];   /* Total number of neurals.*/
```

```
/* Assigning memory to *net, *z, *delta.*/
if(!(net=(float *)malloc(NL[L-1]*sizeof(float))))
   { printf("Out of memory.net\n");exit(1); }
if(!(y=(float *)malloc(Nt*sizeof(float))))
   { printf("Out of memory.y\n");exit(1); }
if(!(delta=(float *)malloc(Nt*sizeof(float))))
   { printf("Out of memory.delta\n");exit(1); }

for(i=0,Nt=0;i<L-1;i++)  Nt+=NL[i];
if(!(kl=(float *)malloc(Nt*sizeof(float))))
        {  printf("Out of memory.kl\n");  exit(1); }
if(!(v=(float *)malloc(Nt*sizeof(float))))
        {  printf("Out of memory.v\n");  exit(1); }

for(i=0,N1=0;i<L-1;i++)  N1 +=NL[i]*NL[i];
if(!(R=(float *)malloc(N1*sizeof(float))))
        {  printf("Out of memory.R\n");  exit(1); }
randomize();
for(i=0;i<N1;i++)
        R[i]=(float)random(N1)/(float)N1;
```

```
printf("\nEnter file name for stored Data-->  ");
scanf("%s",file_name2);
fptr2=fopen(file_name2,"r");
if(fptr2==NULL)
        {printf("file %s does not exist. ",file_name);exit(1);}
/* Determining the size of the data.*/
if(!(mx=(float *)malloc((NL[0])*sizeof(float))))
    { printf("Out of memory.xp\n");exit(1); }
for(i=0;i<NL[0];i++) mx[i]=0.0;
 M=0; ind=1;
while(1)
{
 for(i=0;i<NL[0];i++)
 {
  if((fscanf(fptr2,"%f ",&xt))==EOF) /* input data.*/
        { ind=0;  break; }
  if(fabs(xt)>mx[i]) mx[i]=fabs(xt);
 }
 if(ind==0) break;
 for(i=0;i<NL[L-1];i++)   fscanf(fptr2,"%d ",&xd);  /* desired output.*/
M++;
```

```
}
printf("\n# of data points=%d\n",M);

rewind(fptr2);
/* Assigning memory to *xp, *d  */
if(!(xp=(float *)malloc((M*NL[0])*sizeof(float))))
    { printf("Out of memory.xp\n");exit(1); }
if(!(d=(int *)malloc((M*NL[L-1])*sizeof(int))))
    { printf("Out of memory.d\n");exit(1); }
if(!(dd=(float *)malloc((M*NL[L-1])*sizeof(float))))
    { printf("Out of memory.dd\n");exit(1); }
/* Reading in the data.*/
 for(i=0; i<M; i++)
 {
  for(j=0;j<NL[0];j++)
  {
          fscanf(fptr2,"%f ",&xt);
          xp[j*M+i]=xt;//mx[j];
  }
  for(j=0;j<NL[L-1];j++)
     {
```

```
        fscanf(fptr2,"%d ",&d[j*M+i]);
        dd[j*M+i]=(log((0.5+d[j*M+i])/(1.5-d[j*M+i])))/a;
      }

 }
fclose(fptr2);

fptr=fopen(file_name,"w");
fptr3=fopen(file_name3,"w");
clrscr();
gotoxy(1,1);
printf("Press q to exit and save latest update for weights.\n");

while(iter<no_iter)
{
 if(kbhit() !=0) { if(getch()=='q') ;  break;}
 iter ++;
 for(i=0;i<M;i++)
 {
  qq=random(M);
```

```
for(z=0,N2=0;z<L-1;z++)    N2 += NL[z];


 /* Forward propagation. */
wi=w;
for(z=0;z<NL[1];z++)  /* From input layer to first */
{              /* hidden layer.              */
 for(j=0,ne=theta;j<NL[0];j++)  ne+=*(wi++)*xp[j*M+qq];
 E=(double)exp(-(double)a*ne);
 y[z]=(float)(1.0-E)/(1.0+E);
}
Nt1=NL[1]; Nt2=0;
for(n=2;n<L-1;n++)     /* From layer n-1 to layer n.*/
{
 for(z=0;z<NL[n];z++)
 {
  m=Nt1+z;
  for(j=0,ne=theta;j<NL[n-1];j++) ne+=*(wi++)*y[j+Nt2];
  E=(double)exp(-(double)a*ne);
  y[m]=(float)(1.0-E)/(1.0+E);
 }
 Nt1+=NL[n];
```

```
 Nt2+=NL[n-1];
}
 for(z=0;z<NL[L-1];z++)
 {
  m=Nt1+z;
  for(j=0,net[z]=theta;j<NL[L-2];j++) net[z]+=*(wi++)*y[j+Nt2];
  E=(double)exp(-(double)a*net[z]);
  y[m]=(float)(1.0-E)/(1.0+E);
 }



Nt1=0;
for(z=1;z<(L-1);z++)
  Nt1+=NL[z];


for(z=0;z<NL[L-1];z++)       /* delta's for output layer.*/
{
 ii=Nt1+z;
 error=d[qq+z*M]-0.5-y[ii];
```

```
 delta[ii]=error*fd(ii);
 }

for(m=0;m<(L-2);m++)        /* delta's by back propagation.*/
{
 Nt2=Nt1-NL[L-2-m];
 for(z=0;z<NL[L-2-m];z++)
 {
  ii=Nt2+z;
  sum=0.0;
  n=NS[L-3-m]+z;
  for(j=0;j<NL[L-1-m];j++)
       sum+=delta[Nt1+j]*w[n+j*NL[L-2-m]];
  delta[ii]=fd(ii)*sum;
 }
 Nt1=Nt2;
}

for(z=0,xrx=b;z<NL[0];z++)
```

```
{
 for(j=0,v[z]=0.0;j<NL[0];j++)  v[z]+=R[z+j*NL[0]]*xp[qq+j*M];
 xrx+=v[z]*v[z];
}
xrx=1/xrx;
xrx/=(1+sqrt(xrx));

for(z=0;z<NL[0];z++)
  {
   for(j=0,kl[z]=0;j<NL[0];j++)  kl[z] +=R[j+z*NL[0]]*v[j];
   kl[z]*=xrx;
   for(j=0,n=z*NL[0];j<NL[0];j++)
        R[j+n] =(R[j+n]- v[j]*kl[z])/b;
  }
Nt1=NL[0]*NL[0];  Nt2=NL[0]; Nt3=0;
for(m=1;m<(L-1);m++)
{
 for(z=0,xrx=b;z<NL[m];z++)
 {
   for(j=0,v[z+Nt2]=0.0;j<NL[m];j++)
      v[z+Nt2]+=R[z+Nt1+j*NL[m]]*y[Nt3+j];
```

```
    xrx+=v[z+Nt2]*v[z+Nt2];
  }
xrx=1/xrx;
xrx/=1+sqrt(xrx);

for(z=0;z<NL[m];z++)
{
  for(j=0,kl[z+Nt2]=0,n=Nt1+z*NL[m];j<NL[m];j++) kl[z+Nt2] +=R[n+j]*v[j+Nt2];
  kl[z+Nt2]*=xrx;
  for(j=0,n=Nt1+z*NL[m];j<NL[m];j++)
        R[n+j] =(R[n+j]- v[j+Nt2]*kl[z+Nt2])/b;
}
Nt1+=NL[m]*NL[m];
Nt2+=NL[m];
Nt3+=NL[m];
}
wi=w;
for(z=0;z<NL[1];z++)
  for(j=0;j<NL[0];j++)
    *(wi++) +=kl[j]*delta[z]*st;
Nt1=NL[0],Nt2=NL[1];
```

```
for(n=2;n<L-1;n++)    /* From layer n-1 to layer n.*/
 {
   for(z=0;z<NL[n];z++)
        for(j=0;j<NL[n-1];j++)
          *(wi++) +=kl[Nt1+j]*delta[z+Nt2]*st;
    Nt1+=NL[n-1];
    Nt2+=NL[n];
 }

for(z=0;z<NL[L-1];z++)
 for(j=0;j<NL[L-2];j++)
    *(wi++) +=kl[Nt1+j]*(dd[qq+z*M]-net[z]);
if(b<.98) b+=.000003;
 }
 q=fun(w,xp,d);
 Q+=q;
 if((iter%10)==0)
 {
 fprintf(fptr3,"%d ",iter);
 fprintf(fptr3,"%f\n",Q/10);
 Q=0;
```

```
  }
  gotoxy(1,8);
  printf(" Error function= %f at iteration # %-5d",q,iter);
  if(q<ERROR)break;
}
fclose(fptr3);
fprintf(fptr,"%d ",L);
for(i=0;i<L;i++)        fprintf(fptr,"%d ",NL[i]);
for(i=0;i<N;i++)        fprintf(fptr,"%f ",w[i]);
fprintf(fptr,"%f ",theta);
fclose(fptr);
printf ("\nError=%f ",q);
printf("\n File name used to store weights is %s",file_name);
printf("\n File name for the training data is %s",file_name2);
}

/* Generating the function.*/
float  fun(float *w,float *xp,int *d)
{
 float net,error,q=0.0;
 double E;
```

```
int k,j,i,n,m,Nt1,Nt2;
for(k=0;k<M;k++)
{
 wi=w;
 for(i=0;i<NL[1];i++)  /* From input layer to first */
 {                /* hidden layer.          */
  for(j=0,net=theta;j<NL[0];j++)       net+=*(wi++)*xp[j*M+k];
  E=(double)exp(-(double)a*net);
  y[i]=(float)(1.0-E)/(1.0+E);
 }
Nt1=NL[1]; Nt2=0;

for(n=2;n<L;n++)    /* From layer n-1 to layer n.*/
{
 for(i=0;i<NL[n];i++)
 {
       m=Nt1+i;
       for(j=0,net=theta;j<NL[n-1];j++) net+=*(wi++)*y[j+Nt2];
       E=(double)exp(-(double)a*net);
       y[m]=(float)(1.0-E)/(1.0+E);
  }
```

```
  Nt1+=NL[n];
  Nt2+=NL[n-1];
 }

 for(i=0;i<NL[L-1];i++)   /* Calculating the error. */
 {
  error=d[k+i*M]-0.5-y[Nt2+i];
  q+=error*error;
 }
}/*k-loop*/
q/=2;
return q;
}
```

REFERENCES

[1] L. A. Kamentsky and C. N. Liu, "Computer-automated de: of multifont print recognition logic," *IBM Journal of Research and development*, 7, no. 2, pp. 2-13, 1963.

[2] L. A. Kamentsky and C. N. Liu, "A theoretical and experimental study of a model for pattern recognition," in *Computer and Science*, J. T. Tou and R. H. Wilcox, Eds. New York: Spa Books, 1964.

[3] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, vol. IT-8, pp. 179-187, Feb. 1962.

[4] F. L. Alt, "Digital pattern recognition by moments," in *Optical Character Recognition*, G. L. Fischer *et al.*, Eds. Washington, DC: Mc Greger & Werner, 1962, pp. 159-179.

[5] G. L. Cash and M. Hatamian, "Optical character recognition by the method of moments," *CVGIP*, vol. 39, pp. 291-310, 1987.

[6] C. T. Zahn and R. Z. Reskies, "Fourier descriptors for plane closed curves," *IEEE Transactions on Computer*, vol. C-21, pp. 269-281, Mar. 1972.

[7] G. H. Granlund, "Fourier preprocessing for hand printed character recognition," *IEEE Transactions on Computer*, vol. C-21, pp. 195-201, Feb. 1972.

[8] E. Persoon and K. S. Fu, "Shape discrimination using Fourier descriptors," *IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-7*, pp. 170-179, Mar. 1977.

[9] R. J. Spinrad, "Machine recognition of hand printing," *Journal of Info. and Control*, vol. 8, pp. 124-142, 1965.

[10] V. A. Kovalevsky, "An optimal algorithm for the recognition of some sequences," *Cybernetics*, vol. 3, no. 4, 1967.

[11] J. R. Ullman, "An algorithm for subgraph isomorphism," *Journal of Association of Computer Machinery*, vol. 23, no. 1, pp. 31-42, Jan. 1976.

[12] A. P. Ambler, H. G. Barrow, C. M. Brown, P. M. Burstall, and R. J. Popplestone, "A versatile computer-controlled assembly system," *Artificial Intelligence*, vol. 6, pp. 129-156, 1975.

[13] A. Rosenfeld, R. Hummel, and S. Zucker, "Scene labeling by relaxation operations," *IEEE Transactions on Systems, Man and, Cybernetics*, vol. SMC-6, pp. 420-433, 1976.

[14] L. Davis, "Shape matching using relaxation techniques," *IEEE Transactions on Pattern And. Machine Intell.*, vol. PAMI- 1, pp. 60-72, Jan. 1979.

[15] H. Sherman, "A quasi-topological method for the recognition of line patterns," in *Info. Process., Proceedings UNESCO* Conference (Paris), 1959.

[16] M. Beun, "A flexible method for automatic reading of handwritten numerals," *Philips Tech. Rev.*, vol. 33, no. 4, Part 1, pp. 89-101; Part 11, pp. 130-137, 1973.

[17] R. H. McComick, "The Illinois pattern recognition computer-Illiac Ill," IRE Transactions on Electron. Computer, vol. EC-12, no. 5, 1963.

[18] C. J. Hilditch, "Linear skeleton from square cupboards," *in Machine Intelligence IV*, B. Meltzer and D. Michie, Eds. Edinburgh, University Press, 1969, pp. 403-420.

[19] H. Tamura, "A comparison of line thinning algorithms from digital computer view point," in *Proceedings of 4th Intenational Joint Conference on Pattern Recognition*, 1978, pp. 715-719.

[20] T. Pavlidis, "Computer recognition of figures through decomposition," *Journal of Info. Control.* vol. 12, pp. 526-537, 1968 .

[21] P. G. Perotto, "A new method for automatic character recognition," *IEEE Transactions on Electron. Computer,* vol. EC-12, pp. 521-526, Oct. 1963.

[22] M. Nadler, "Sequentially-local picture operators," in *Proceedings 2nd IJCPR*, 1974, pp. 131-135.

[23] C. B. Shelman, "The application of list processing techniques to picture processing," *Pattern Recognition*, vol. 4, pp. 201-210, 1972.

[25] E. C. Greanias, P. E. Meagher, R. J. Norman, and P. Essinge "The recognition of handwritten numerals by contour analysis," *IBM Journal of Research and development*, vol. 7, pp. 2-13, 1963.

[26] H. Freeman, "On the digital computer classification of geometric line patterns," in *Proceedings of Natl. Elect. Conference*, vol. 18, 1962, pp. 312-324.

[27] G. Gallus and P. W. Nourath, "Improved computer chromosome analysis incorporating preprocessing and boundary analysis," *Physics . Med. Biol.*, vol. 15, pp. 435-445, 1970.

[28] A. Rosenfeld and E. Johnston, "Angle detection on digital curves," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-5, pp. 610-614, Nov. 1975.

[29] H. Freeman and L. S. Davis, "A corner finding algorithm for chain coded curves," *IEEE Transactions on Computer*, vol. C-26, pp. 297-303, 1977.

[30] T. Pavlidis and F. Ali, "Computer recognition of hand written numerals by polygonal approximations," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-5, pp. 610-614, Nov. 1975.

[31] T. Pavlidis and S. L. Horowitz, "Segmentation of plane curve," *IEEE Transactions on Computer*, vol. C-23, pp. 860-870, Aug. 1974.

[32] T. Pavlidis, Structural Pattern Recognition. New York: Springer, 1977.

[33] M. D. Levine, "Feature extraction: A survey," Proceedings 57, pp. 1391-1419, Aug. 1969.

[34] B. T. Mitchell and A. M. Gillies, "a model-based computer vision system for recognizing hand written ZIP codes," *Machine Vision and Applications*, vol. 2, pp. 231-243, 1989.

[35] R. M. Brown, T. M. Fay, and C. L. Walker, "Hand-printed symbol recognition system," *Pattern Recognition*, vol. 21, no. 2, pp. 91-118. 1988.

[36] C. Y. Suen, *et al.* "Recognition of totally unconstrained handwritten numerals based on the concept of multiple experts" in *Proceedings IEEE*, June pp. 1162-1180, 1992.

[37] A. Krzyzak, W. Dai, and C. Y. Suen, "Unconstrained hand-written character classification using modified back-propagation model," in *Proceedings of Intenational Workshop Frontiers in Handwriting Recognition*, Apr. 1990, pp. 155- 166.

[38] Y. Le Cun *et al.*, "Constrained neural networks for unconstrained handwritten digit recognition," *Proceedings of Intenational Workshop Frontiers in Handwriting Recognition* Apr. 1990, pp. 145- 154.

[39] Amin, Adnan; Masini, Gerald, "Machine Recognition Of Multi-Font Printed Arabic Texts". *Proceedings . International Conference on Pattern Recognition 8th.* Publ by IEEE, New York, NY, USA. Available from IEEE Service Cent (Cat n 86CH2342-4), Piscataway, NJ, USA p 392-395, 1986

[40] El-Sheikh, Talaat S.; Guindi, Ramez M. "COMPUTER RECOGNITION OF ARABIC CURSIVE SCRIPTS" *Pattern Recognition* v 21 n 4 1988 p 293-302

[41] Amin, Adnan; Mari, Jean Francois " Machine recognition and correction of printed Arabic text." *IEEE Transactions on Systems, Man and Cybernetics* v 19 n 5 Sep-Oct 1989 p 1300-1305

[42] El Gowely, Khaled; El Dessouki, Ossama; Nazif, Ahmed "Multi-phase recognition of multi-font photoscript Arabic text." *Proceedings . International Conference on Pattern Recognition* v 1. Publ by IEEE, IEEE Service Center, Piscataway, NJ, USA (IEEE cat n 90CH2898-5). p 700-702 1990

[43] Impedovo, S.; Dimauro, G. "An interactive system for the selection of handwritten numeral classes." *Proceedings . International Conference on Pattern Recognition* v 1. Publ by IEEE, IEEE Service Center, Piscataway, NJ, USA (IEEE cat n 90CH2898-5). p 563-566 1990

[44] El-Khaly, F.; Sid-Ahmed, M. A. "Machine recognition of optically captured machine printed Arabic text." *Pattern Recognition.* v 23 n 11 1990 p 1207-1214

[45] Sami El-Dabi, Sherif; Ramsis, Refat; Kamel, Aladin" Arabic character recognition system. A statistical approach for recognizing cursive typewritten text." *Pattern Recognition* v 23 n 5 1990 p 485-495.

[46] El-Wakil, Mohamed S.; Shoukry, Amin A " On-line recognition of handwritten isolated Arabic characters." *Pattern Recognition* v 22 n 2 1989 p 97-105

[47] Al-Emami, Samir; Usher, Mike "On-line recognition of handwritten Arabic characters." *IEEE Transactions on Pattern Analysis and Machine Intelligence* v 12 n 7 July 1990 p 704-710

[48] Al-Yousefi, H.; Udpa, S. S. " Recognition of Arabic characters." *IEEE Transactions on Pattern Analysis and Machine Intelligence* v 14 n 8 Aug 1992 p 853-857.

[51] M.-K. Hu, "Pattern recognition by moment invariants," *Proc. IRE,* vol. 49, p. 1428, Sept. 1961.

[52] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Inform. Theory,* vol. IT-8, pp. 179-187, Feb. 1962.

[53] F. W. Smith and M. H. Wright, "Automatic ship photo interpretation by the method of moments," *IEEE Transactions on Comput.,* vol. C-20, pp. 1089-1094, Sept. 1971.

[54] K. Udagawa, J. Tofiwaki, and K. Sugino, "Normalization and recognition of two-dimensional patterns with linear distortion by moments, " *Electron. Commun. Japan,* vol. 47, no. 6, pp. 34-46, 1964.

[55] R. G. Casey, "Moment normalization of handprinted characters," *IBM J. Res. Develop.,* vol. 14, pp. 548-557, Sept. 1970.

[56] S. A. Dudani, K. J. Breeding, and R. B. McGhee, "Aircraft identification by moment invariants," *IEEE Transactions on Comput.*, vol. C-26, pp. 39-46, Jan. 1977.

[57] S. S. Reddi, "Radial and angular moment invariants for image identification," *IEEE Transactions on Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 240- 242, Mar. 1981.

[58] F. A. Sadjadi and E. L. Hall, "Three-dimensional moment invariants," *IEEE Transactions on Pattern Anal. Machine Intell.*, vol. PAMI-2, pp. 127-136, Mar. 1980.

[59] M. R. Teague, "Image analysis via the general theory of moments," *J. Opt. Soc. Amer.*, vol. 70, pp. 920-930, Aug. 1980.

[60] J. F. Boyce and W. J. Hossack, "Moment invariants for pattern recognition," *Pattern Recognition Lett.*, vol. 1, no. 5-6, pp. 451-456,m July 1983.

[61] Y. S. Abu-Mostafa and D. Psaitis, "Recognitive aspects of moments invariants," *IEEE Transactions on Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 698-706, Nov. 1984.

[62] Y. S. Abu-Mostafa and D. Psaitis, "Image normalization by complex moments," *IEEE Transactions on Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 46-55, Jan. 1985.

[63] C.-H. Teh and R. T. Chin, "On digital approximation of moment invariants," *Comput. Vision, Graphics, Image Processing*, vol. 33, pp. 318-326, 1986.

[64] R. Courant and D. Hilbert, *Methods of Mathematical Physics,* Vol. 1 New York: Interscience, 1953.

[65] CHO-HUAK THE, and Roland T. Chin, "On Image Analysis by the Methods of Moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 4, July 1988.

[66] A. Khotanzad, and Jiin-Her Lu, "Classification of Invariant Image Representations Using a Neural Network," *IEEE Transactions on Acoustics, Speach, and Signal Processing*, Vol. 38, No. 6, June 1990.

[67] A. B. Bhatia and E. Wolf, *Porc. Camb. Phil. Soc., vol.* 50, pp. 40-48, 1954.

[68] Tien C. H., "A Note on Invariant Moments in Image Processing," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 12, December 1981.

[71] R. 0. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* John Wiley & Sons, New York (1973).

[72] A. lituivi and R.L.Rivest "Training a 3-node neural network is NP-complete" In *Proceedings of the computational learning theory (COLT) conference,* pp. 9-18. Morgan Kaufmann, 1988.

[73] F L. Lewis, *Optimal Estimation,* John Wiley & Sons, New York (1986).

[74] G.G. Lorentz, 'The 13th Problem of Hilbert," in F. E. Browder (Ed.), *Mathematical Developments Arising from Hilbert Problems,* American Mathematical Society, Providence, R.I. (1976).

[75] M. Minsky, and S. Papert, *Perceptrons: An Introduction to Computational Geometry,* MIT Press (1969).

[76] T. Parsons, *Voice and Speech Processing,* McGraw-Hill, New York (1986).

[77] R. Rosenblatt, *Principles of Neurodynamics, New* York, Spartan Books (1959).

[78] D.E. Rumelhart, G.E. Hinton, and R. Williams, "Learning Internal Representations by Error Propagation" in D. E. Rumelhart & J. L. McCielland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol.* 1: *Foundations. MIT* Press (1986).

[79] S. Becker and Y. le Cun, "Improving the convergence of back-propagation learning With second-order methods," *Technical Report CRG*-TR-88-5, U. of Toronto, Toronto, Canada, 1988.

[80] B. Widrow, and M. E. Hoff, "Adaptive Switching Circuits," 1960 *IRE WESCON Conv. Record, Part* 4, 96-104, August 1960.

[81] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, New Jersey (1985).

[82] D. R. Hush and J.M. Salas, "Improving the learning of back-propagation with the gradient reuse algorithm," *In proceedings of the IEEE International conference on Neural Networks*, volume 1, pages 441-448, 1988.

[83] R.A. Jacobs, "Increased rate of convergence through learning rate adaptation," Neural Networks, 1(4):294-308, 1988.

[84] J.S. Jadd, Neural Network Design and the complexity of learning, MIT Press, Cambridge, MA, 1990.

[85] R.L. Watrous, "Learning algorithms for connectional networks: applied gradient methods of nonlinear optimization," In Proceedings IEEE 1st International conference on Neural Networks, volume 2, pages 619-628, 1987.

[86] D.C. Plaut, S.J. Nowlan, and G.E. Hinton, "Experiments on learning back-propagation," Technical Report CMU-CS-86-128, Carnegie-Mellon University, Pittsburgh, PA, 1986.

[87] J. G. Proakis, Digital Communications. New York: McGraw-Hill, 1983.

[88] S. S. Haykin, Adaptive Filter Theory. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[89] R. S. Scalero, and N. Tepedelenlioglu, "A Fast New Algorithm for Training Feedforward Neural Networks,' *IEEE Transactions on Signal Processing*, Vol. 40, No. 1, January 1992.

[91] Battin, R. H., "Astronaytical Guidance," pp. 338-339. McGraw-Hill, NewYork (1964).

[92] Gerald J. Bierman, " Factorization Methods for Discrete Sequential Estimation," Academic Press, (1977).

[93] El-Khaly, F.; Sid-Ahmed, M. A. "Machine recognition of optically captured machine printed Arabic text." *Pattern Recognition.* Vol. 23, No. 11 1990 p 1207-1214

[94] C.-H. Teh and R. T. Chin, "On digital approximation of moments invariants," *Computer. Vision, Graphics, Image Processing,* vol. 3 pp. 318-326, 1986.

[95] Cho-Huak Teh and R. T. Chin, "On Image Analysis by moments" *IEEE Transactions on pattern analysis and Machine Intelligence*, Vol. 10. No. 4, July 1988.

[96] Khotanzad, A. Jiin-Her Lu, "Classification of invariant Image Representations Using a Neural Network" *IEEE Transactions on Acoustics, speech, and signal processing,* Vol. 38, No. 6, June 1990.

[97] R. O. Duda and P. E. Hart, Pattern Classification and scene analysis, Jone wily & Sons, New York (1973).

[98] Shih-Chi, H. And Yih-Fang, H. "Bounds on the number of hidden neurons in multilayer perceptrons" *IEEE Transactions on Neural Networks*, Vol. 2, No. 1, January 1991.

[99] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoustics, Speech and Signal Processing Magazine*, 4(2):4-22, April 1987.