Industry Canada CRC

IMAGE CLASSIFICATION BY NEURAL NETWORKS USING MOMENT INVARIANT FEATURE VECTORS

by

Kenneth L. Sala

CRC REPORT NO. 97-002

February 1997
Ottawa

The work described in this document was sponsored by the Department of National Defence under Task 5BB14.

Canada

IMAGE CLASSIFICATION BY NEURAL NETWORKS USING MOMENT INVARIANT FEATURE VECTORS

by

Kenneth L. Sala

The work described in this document was sponsored by the Department of National Defence under Task 5BB14.

COMMUNICATION RESEARCH CENTRE, INDUSTRY CANADA
CRC REPORT NO. 97-002

Canada

February 1997
Ottawa

Abstract

An image classification system based upon the extraction of moment invariant feature vectors and an artificial neural network classifier is described. The moment invariant feature vectors are derived from the test images using series of orthogonal basis functions. Six different basis functions are studied which include four types of Zernike functions and two types of Walsh functions. Four different schemes for the normalization of the feature vectors are also investigated. The images used in the study possess random scales, lateral positions, and angles of orientation in the image plane. In addition, random noise with different signal-to-noise ratios is superimposed upon the images. The feature vector extraction technique employs the concept of moment invariants so that the feature vector components are independent of the image's scale, lateral position, and orientation. The neural network employed for the classification task is a multilayer perceptron network which is trained with the backpropagation algorithm. The performance of the overall classification system is determined by measuring the classification accuracy as a function of the signal-to-noise ratio of the test imagery. The work and the results presented in this study form the basis for a neural network based, image recognition system which will be employed in the classification of military, synthetic aperture radar (SAR) imagery of land targets.

Resumé

Un système de classification d'images fondé sur l'extraction des vecteurs caractéristiques à moment invariant et d'un classificateur de réseau neural est décrit. Les vecteurs caractéristiques à moment invariant sont obtenus par des images de référence en utilisant une série de fonctions orthogonales. Six fonctions orthogonales sont étudiées incluant quatre types de fonctions Zernike et deux types de fonctions Walsh. Quatre méthodes de normalisation de ces vecteurs caractéristiques sont aussi étudiées. Les images utilisées dans cette étude varient selon l'échelle, la position latérale et l'angle d'orientation du plan de l'image. En plus, un bruit de fond aléatoire ayant des rapports signal-bruit différents sera surimposé sur les images. La technique d'extraction des vecteurs emploie le concept des moments invariants afin que les composants de ces vecteurs soient indépendants de l'échelle, de la position latérale et de l'orientation de l'image. Le réseau neural employé pour le travail de classification est un réseau perceptron à couches multiples qui reconnaît l'algorithme de propagation. Le rendement du système de classification est déterminé en mesurant l'acuité de la classification en fonction du rapport signal-bruit des images de référence. Le travail et les résultats présentés dans cette étude forment la base d'un système de reconnaissance d'images fondé sur les réseaux neuraux et qui sera employé dans la classification d'images militaires de cibles terrestres obtenues par radar à ouverture synthétique (SAR).

Executive Summary

The work described in this report represents the successful completion of the first phase of a DND sponsored project which has as its fundamental objective the development of a neural network based classification system for SAR imagery of land targets. The central purpose of this initial phase of the project is to comprehensively characterize the methodology of using moment invariant feature vectors as the means of representing the imagery to the neural classifier. An image database consisting of binary valued, 2-D objects is employed in these studies in order to facilitate a determination of an optimum environment for image classification using this combination of moment invariant features and a multilayer perceptron, neural network classifier. Six different mathematical bases employed in the image feature vector extraction are studied. Each of these bases is examined for two types of training data sets, one containing only noiseless imagery and the second containing a mixture of noiseless and noisy imagery. In addition, four different schemes are investigated for the normalization of the feature vectors. Other results described include the measurement of the dependence of the neural network classifier upon training epoch size, the number of hidden neurons in the network architecture, and upon the network initialization process. A series of feature vector graphs is presented which illustrate the measured invariance of the feature vectors, their degradation with decreasing signal-to-noise ratio, similarities and differences between feature vectors for similar and dissimilar images, and the dependence of the feature vectors upon the normalization method. The specific environment and experimental process determined by this phase of the project shall form the basis for classification studies of actual SAR imagery during the second phase of the project.

Table of Contents

List of Figures and Tables

Chapter 2

Chapter 3

Chapter 4

Chapter 5

List of Acronyms, Symbols, and Abbreviations

Notation	Meaning	First Use or Definition
SZF	Standard Zernike Function(s)	pg. 7
PZF	Pseudo Zernike Function(s)	pg. 7
SZRP	Standard Zernike Radial Polynomial(s)	pg. 7
PZRP	Pseudo Zernike Radial Polynomial(s)	pg. 7
WRF	Walsh Radial Function(s)	pg. 7
HRF	Haar Radial Function(s)	pg. 7
$SZF_{nm}(r,\theta)$	Standard Zernike Function of Order {n,m}	pg. 11, eq. (2.10)
$PZF_{nm}(r,\theta)$	Pseudo Zernike Function of Order {n,m}	pg. 22, eq. (2.24)
$SZRP_{nm}(r,\theta)$	Standard Zernike Radial Polynomial of Order {n,m}	pg. 12, eq. (2.11)
$PZRP_{nm}(r,\theta)$	Pseudo Zernike Radial Polynomial of Order {n,m}	pg. 22, eq. (2.25)
$WAL(n,x)$	Walsh Function of Order n	pg. 35, eq. (2.39)
$WRF_n(r)$	Walsh Radial Function of Order n	pg. 38, eq. (2.44)
$HAR(n,x)$	Haar Function of Order n	pg. 42, eq. (2.48)
$HRF_n(r)$	Haar Radial Function of Order n	pg. 42, eq. (2.51)
β	Irradiance Normalization Constant	pg. 9
m_{ij}	Algebraic Moments	pg. 7, eq. (2.1)
μ_{ij}	Irradiance Normalized Central Moments	pg. 9, eqs. (2.6) and (2.9)
TIFF	Tagged Image File Format	pg. 53
SNR	Signal to Noise Ratio	pg. 63, eq. (3.1)
SN	Standard Normalization	pg. 71, eq. (3.3)
PN	Positive Normalization	pg. 71, eq. (3.4)
BN	Bipolar Normalization	pg. 71, eq. (3.5)
DN	Differential Normalization	pg. 72, eq. (3.6)
fv(i,j)	Normalized Feature Vector, ith vector, jth Component	pg. 71, eq. (3.2)
rfv(i,j)	Raw Feature Vector, ith vector, jth Component	pg. 71, eq. (3.2)
Training Set A	Set of 45 Noiseless Images for Training	pg. 56
Training Set B	Set of 45 Noiseless + 45 SNR 8 db Images for Training	pg. 56

1. Introduction

Research into the development and application of intelligent systems in military environments is acquiring increasing significance and importance given the remarkable progress in the area of sensor technologies and in the ability to collect and process increasingly complex volumes of sensor data[41]. At the very core of the development of intelligent systems lies the task of extracting 'information' from 'data'. The distinction between these terms, while occasionally subtle, is always a fundamental and critical one in this area of research. At present, almost without exception, this task of information extraction is, largely by default, solely the domain of trained and experienced human interpreters. This situation, however, is somewhat discrepant with the observation that, for many sensor technologies such as radar imagery, sonar detection, and infrared photography, the types and characteristics of the data collected are largely alien to human experience[49,110,127]. Synthetic aperture radar (SAR) imagery offers the perfect example of this type of data. The large dynamic range of the radar returns intrinsic to this technique is a characteristic that the human visual processing system is simply incapable of perceiving. Examples of SAR images, such as those reproduced in the concluding chapter of this report, are, in fact, 'depictions' or 'representation' of the true data given the limitations of the printed medium (even if this type of imagery could be faithfully reproduced, the reader would still be unable to effectively "see" this dynamic range). In a literal sense, there is more to SAR imagery than meets the eye. In many respects, therefore, it can be argued that the task of interpreting data such as SAR imagery should ultimately be left to artificial means since only this approach may be expected to fully utilize and exploit these types of complex data. The critical step, however, towards the realization of this approach lies with the development of systems which, like the human vision system, are capable of recognizing and accurately identifying varied classes of patterns contained within sensor data. Furthermore, particularly within military contexts, artificial classification systems must be capable of accurate, robust pattern classification with high confidence limits and be able to classify quickly, ideally in real time[3,18,95,126]. Indeed, it is argued by many researchers[41,94,95], future advances in military sensor technologies may be compromised unless such artificial classification and automated target recognition systems can be successfully developed in the short to medium term[14,42,126,127].

In the particular field of SAR imagery, a principal thrust in the development of intelligent systems in this area employs some form of artificial neural network[30,49] to execute the task of identification and classification of patterns[14,42,94,126]. The work along this direction addresses two main areas of research. One is the investigation of different types of artificial neural systems and learning paradigms[4,19,23,42,43,63,118]. While the majority of work employs some form of multilayer perceptron network which is trained using the backpropagation learning algorithm (or some variation of it), many investigators are studying more general neural systems which utilize unsupervised learning models and which are capable of some

degree of self-organizing behavior in the determination of and recognition of image classes[3,63,97]. Other research examines alternate neural networks[4,23,37,51,88,124] such as higher order networks[26,39,57,75,83,91,100,102,107-109] which offer inherent invariance via the network's structure to various image transformations such as changes in scale or object orientation. The second main activity within the research on neural network based classification of SAR imagery concerns the identification of suitable features which quantitatively characterize the imagery sufficiently to permit their use as the input parameters for the neural classifier[13,14,18,32,81,105,106]. This task, somewhat surprisingly, is proving to be a difficult and elusive one insofar as determining a generic set of features or a methodology to determine such a set which proves to be both effective and general in its applicability to various types of SAR imagery. The reader is referred to the collection of articles which deal specifically with the task of classification of SAR imagery presented recently as a Special Issue[41] of the journal Neural Networks (Grossberg, Hawkins, and Waxman, eds.). In addition, the reader may find it instructive to compare the review article concerning neural network classification of SAR imagery by Rogers et al[94] in 1995 with those by Waxman et al[126] in 1993 and Roth[95] in 1990.

The work described in this report represents the successful completion of the first phase in a DND sponsored project which has as its fundamental objective the development of a neural network based classification system for SAR imagery of land targets. This initial phase of the work concentrates on characterizing the methodology of moment invariant feature vectors as the means of representing imagery to the neural classifier. To this end, a simple image database consisting of binary valued, 2-D objects is employed in these studies in order to concentrate on determining an optimum environment for image classification using this combination of moment invariant features and a neural network classifier. The essential advantage to working with simplified imagery in this phase is that the performance of the classification system can be measured against known standards and thus permit meaningful optimization of many of the classification system variables and parameters. The method decided upon by the results of this phase of the project will then form the basis for classification studies of actual SAR imagery during the second phase of the project. For this reason, considerable attention is paid to the development of the methods and algorithms created during this phase of the research to ensure that they can be easily adapted or modified to remain applicable to the more general characteristics of the SAR imagery. Indeed, at the time of preparation of this report, work is well underway in the application of these results to the classification of a recently acquired SAR image database of land targets of military significance. Other research previously reported under this project includes a study of time-delay and perceptron neural network classifiers for SAR imagery of shipping[118] and a study of the application of higher order neural networks in conjunction with boundary detection and encoding algorithms for image classification[48].

Chapter 2 describes the method of moment invariants as it applies to the extraction of feature vectors for the characterization and classification of imagery[7]. This technique is based upon the use of a series of basis functions from which are derived a sequence of global features which, loosely speaking, are directly related to the generalized Fourier coefficients of the image with respect to the basis[12,86,114,130]. By suitably transforming the image and making appropriate choices for the basis functions, features are derived which are invariant to changes in the scale, lateral position, and in-plane orientation of the image. Six different basis functions are studied in this report of which four have heretofore been unreported in the literature on this subject. Chapter 2 also includes brief expositions of the mathematical properties of these six sets of basis functions along a discussion of some of the practical consequences and limitations of working with finite resolution imagery and sampled versions of the basis functions. Chapter 3 describes the experimental procedure for the feature vector extraction and the processing steps in the preparation of this data for use by the neural network classifier. The various computer programs developed for this purpose are briefly described in chapter 3. Complete listings of the various algorithms employed in this study are included as appendices to this report to which the reader interested in the finer details of the sequence of image processing and feature vector steps is referred. Chapter 4 presents the results of several prefatory experiments which determine approximately optimum values for several of the key parameters and variables which characterize both the neural network classifier and the feature vector processing stage. The results describing the performance of the overall, neural network based classification system are then presented for each of the basis function sets employing two different training sets of feature vectors. The classifier's performance is judged by measuring the classification accuracy as a function of signal-to-noise ratio's for 23 separate test sets of images which possess random scales, lateral positions, and orientation angles. The signal-to-noise ratios (SNR) of these test sets vary from that of noiseless imagery through to pure noise. Chapter 5 offers a brief discussion of the main findings of the present work and considers some of the difficulties and questions which must be faced in the second phase of this work, namely the application of the methodology studied here to the analysis and classification of actual SAR imagery.

Chapter 2 : Moment Invariant Feature Vectors for Pattern Recognition

2.1 : Overview of Feature Vectors and Moment Invariants

There are two basic concepts which are central to the entire field of pattern recognition[7,30,37,49,65,119]. The first is that, in the preponderance of cases, complex signals such as images or speech patterns can be accurately identified and classified through the use of a relatively small number of judiciously chosen features which characterize those signals[7,73,93,98]. This concept is particularly applicable in the area of image recognition and classification where, for even fairly modest imagery, the sheer amount of image data can be problematic (e.g., a single 2048 x 2048 color image with 24 bits/pixel represents 12 megabytes of image data). By making intelligent choices for those quantitative measures to serve as image features, the task of image classification can be made tractable where it would otherwise prove pragmatically unworkable if one were forced to deal strictly with raw image data. The second concept which is also central to the problem of pattern recognition is that the classification system must possess the capability to accurately classify the signals in the presence of transformations or processes which modify them, i.e., the classifier must be invariant[1,7,12,53,61,65,114,130,132] to one or more types of signal transformations or distortions. Examples would include speaker-independent speech recognition[21] and image classification which is invariant to changes in image scale and viewing angle[7,12,130,114]. It is important to recognize a priori that the incorporation of invariance into the pattern classification task should not be regarded as an appurtenance to the classifier design and methodology. Rather, the means by which the experimenter chooses to implement invariance into the overall process will, to a large extent, determine the structure and basic design of the classification system itself. Finally, it should be noted that these underlying concepts are not entirely abstract or theoretical contrivances. A growing body of experimental evidence exists which indicates that biological pattern recognition systems such as human vision explicitly incorporate various types of feature detection and extraction along with an intrinsic, learned ability to recognize patterns independently of a broad range of variances in the patterns[30,37,49,126,127].

Image features may be divided into two distinct categories, namely local and global image features[93,52,55,119]. Local features, as their name implies, are quantitative variables which depend only upon some localized area(s) or characteristic of the image object. Examples would include a fractal dimension measured over a small, movable window and various geometric parameters such as the number of corners present in an image's boundary representation. Frequently, although not invariably, local features such as those derived from the object's geometric or shape attributes possess the extremely useful property that such features are independent of a variety of types of transformations or modifications to the image such as spatial scaling, object translation, in-plane rotation, and image intensity scale or color.

Although this intrinsic possession of invariance by such features is an attractive advantage, the use of local features to classify imagery suffers from a number of important drawbacks. Perhaps chief among these drawbacks is the difficulty the experimenter faces when required to significantly increase the dimension of the feature vector as the image database increases in image complexity and/or in the number of distinct classes which must be recognized by the image recognition system. In addition, as experimental evidence has indicated, particularly with respect to real image data such as SAR imagery of land targets[14,42,94,126,127], the classification capabilities for a set of localized features may be excellent for one image data set but deteriorate or fail completely for a different set of images (e.g., SAR imagery for which basic parameters such as pixel resolution and polarization discrimination are different). In effect, the experimenter is forced to customize the choice of feature vector components to the specific image database under consideration. One final point regarding the use of localized features which deserves comment is the fact that, in a great deal of the work in which images are classified using local invariant features, some form of boundary extraction algorithm is employed to represent the image solely on the basis of its shape[10,27,93,104,131]. For many types of real world imagery such as SAR imagery, the complex nature and wide dynamic range of the image objects effectively precludes the use of boundary detection techniques for such data.

Global features[1,7,12,40,59,86,119,130,132] of an image refer to quantities which are calculated using the entire image. The coefficients of any form of generalized Fourier transform of an image, for example, represent global image features. The principal advantages in using global features such as those based upon generalized Fourier coefficients or upon the 'projection' of the image onto some basis series of mathematical functions are as follows[114,7,12,116,70,73]:

(1) the derivation of such features is done 'blindly', i.e., the method of feature generation in no way depends upon the specific nature or type of imagery;
(2) the dimension of the feature vector can normally be increased or decreased arbitrarily according to the requirements of the image database complexity;
(3) by choosing orthogonal bases for the generalized Fourier transform or image projection, the feature vector components can be made to be independent of each other and possess no information redundancy; and
(4) global features are, in general terms, reasonably insensitive to image noise and minor occultations.

The chief drawback to the use of global features such as those derived from generalized Fourier coefficients is that they *are* dependent, in general, upon global image characteristics such as spatial scale, lateral position, in-plane rotation of the object, and image intensity scale. As noted above, this question of invariance of the pattern recognition system to such changes in image characteristics is a fundamental one for image classification systems since the observer has little if any control over the degree of image variability or degradation arising

from various types of geometric transformations (scaling, object position, viewing angle) or the presence of noise and occultation in the imaging process. The ability to recognize and classify an object independently of such variations in the image presents one of the most critical and challenging aspects in the design of an automated target recognition system[7,130,14,94,95,126].

With respect to image data, one approach to deriving global features which possess the property of invariance to one or more types of image transformation consists in actively processing and transforming the image so that it complies or conforms to predefined templates[22,7]. For example, if invariance to image rotation is required, a 'fitting' rectangle can be computed which contains the image object. The image can then be rotated in such a way that the object and its fitted rectangle are aligned with fixed, principal axes. If this alignment of the object is done for all the images contained in a database, then any global features subsequently derived from these transformed images will be independent of the object's original orientation. Until relatively recently, this type of image preprocessing and transformation was the principal method available for incorporating multiple invariances (such as invariance to image scale, position, and rotation) into the feature extraction process. Aside from the obvious disadvantage of requiring computationally extensive image preprocessing on each and every image in the image database, this task is often intractable or impractical for complex imagery (e.g., 2D imagery of 3D objects[38,99]) which may contain noise, asymmetric distortions, or occultations[7].

An alternative method to achieve invariance to geometric transformations of an image (i.e., changes to scale, position, and angle of rotation) was first proposed by Hu[53,54] in the early 1960's. Hu proposed using global features (specifically, algebraic moments) which did not possess the required invariances but for which series of algebraic expressions could be derived which did possess the required invariances. In essence, the invariances could be achieved by manipulating straightforward global features derived from the image as given rather than by manipulating the image itself, a generally much more difficult and abstruse task to perform. Although Hu's methodology for the derivation of his 'algebraic invariants' and his choice of basis functions by which the global features were calculated were subsequently replaced by better alternatives[7,12,58,114], his work quickly prompted renewed interest in this problem and led to advancements and improvements being reported by many other researchers[1,2,12,31,32,58-61,25,75,83,90,70-73,103,114,116,125,129,132]. Unquestionably, the single most important development since Hu's seminal work has been the adoption of series of orthogonal functions to serve as the basis for the feature extraction[1,5,7,12,16,72,73,86,90,92,114,130]. Several researchers have examined and experimented with a variety of basis functions, among them the Legendre polynomials, Zernike functions, and complex algebraic functions. The Zernike functions, first proposed by Teague[114], have been consistently reported by researchers[1,7,12,58-61,70-73,75,79,86,103,116,122] to outperform other bases in the task of pattern classification using

moment invariant feature vectors. To a certain extent, the (complete) standard and pseudo Zernike functions serve as standards throughout this report by which other results are measured. This report examines six types of basis functions which include the standard Zernike function (SZF), the pseudo Zernike function (PZF), the standard Zernike radial polynomial (SZRP), the pseudo Zernike radial polynomial (PZRP), the Walsh radial function (WRF), and the Haar radial function (HRF). Note that the application of SZRP, PZRP, WRF, and HRF bases described in this report represents novel results; these bases have not, heretofore, been reported in the literature.

In the remainder of this chapter, the algebraic, Zernike, Walsh, and Haar moments will be defined and brief expositions of the mathematical properties of these various basis functions will be given. The final section in this chapter will discuss the discrete (pixel) formulation of the exact, analog expressions for the moment invariants and some of the problems which arise from the finite sampling representation of both images and basis functions. The discussion given above of invariance in feature-based pattern recognition systems is, of necessity, extremely brief. The interested reader is referred to any of several review articles[7,12,65,86,114,116,130] for much more detailed discussions of this subject and invariant pattern recognition in general. More detailed descriptions of the various types of moment invariants and the mathematical bases from which they are derived may also be found in the literature[2,12,28,56,58-61,68,70,72,73,79,84,86,90,114,130,133].

2.2 : Algebraic Moments

Algebraic moments (also referred to in the literature as Hu, geometric, regular, or normal moments) of order $p+q$ are defined as the projection of the image onto the algebraic basis functions $H_{pq}(x,y) = x^p y^q$, i.e.,

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)x^p y^q \, dx\, dy \quad ; p,q \geq 0 \tag{2.1}$$

where the image density (irradiance) distribution $f(x,y)$, which is everywhere positive, is assumed to be finite in value and extent (i.e., m_{00} finite) and piecewise continuous. The moments m_{pq} represent, in effect, the coefficients of the generalized Fourier transform of the image with respect to the basis functions $H_{pq}(x,y) = x^p y^q$. However, although this set of functions is complete on the unit square, the $H_{pq}(x,y)$ with $p,q \geq 0$ are not generally orthogonal:

$$\int_{-1}^{1}\int_{-1}^{1} H_{pq}H_{st}\,dxdy = \int_{-1}^{1}\int_{-1}^{1} x^{p+s}y^{q+t}\,dxdy = \begin{cases} \dfrac{4}{(p+s+1)(q+t+1)}; & (p+s)\ and\ (q+t)\ even \\ 0 & ;(p+s)\ or\ (q+t)\ odd \end{cases} \tag{2.2}$$

Similarly, it should also be noted that the area or measure of H_{pq} is not generally equal to 0, specifically,

$$\int_{-1}^{1}\int_{-1}^{1} H_{pq}\,dxdy = \int_{-1}^{1}\int_{-1}^{1} x^{p}y^{q}\,dxdy = \begin{cases} 0 & ;p\ or\ q\ odd \\ \dfrac{4}{(p+1)(q+1)} & ;p\ and\ q\ even \end{cases} \tag{2.3}$$

The reader will note that the definition of the algebraic moments, eq. (2.1), involves integration over the entire Cartesian plane while eqs. (2.2) and (2.3) involving only the basis functions $x^p y^q$, are restricted to the region [-1,1] x [-1,1] (in effect, the functions $H_{pq}(x,y)$ are defined to vanish for $|x|$ or $|y| > 1$). In practice, the image distribution $f(x,y)$ will occupy only a finite portion of the image plane, thus allowing a unit area to be defined (through renormalization of the x, y coordinates) which encloses the image distribution.

The moments m_{pq} of eq. (2.1) have special meanings and interpretations, particularly for the lower values of $(p+q)$. Many of these properties are optical or image analogs to mechanical moments, appropriate given the strictly positive nature of the image irradiance function $f(x,y)$. The lowest order moment

$$m_{00} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,dx\,dy \tag{2.4}$$

represents the total irradiance or 'optical mass' of the image (for noiseless, binary images, m_{00} is simply the total number of 'black' pixels). There are two first order moments

$$\left.\begin{aligned} m_{10} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,x\,dx\,dy \\ m_{01} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,y\,dx\,dy \end{aligned}\right\} \tag{2.5}$$

which, taken together, define a "center of optical mass" as that point in the image about which these first order moments would vanish. To transform to a coordinate system centered at this point and to ascribe a fixed total irradiance 'β' to the image, the image is scaled and shifted and an alternate set of "irradiance normalized central algebraic moments" μ_{pq} is defined as

$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha x + \bar{x}, \alpha y + \bar{y}) x^p y^q \, dx\,dy \quad ; p,q \geq 0 \tag{2.6}$$

where

$$\left.\begin{aligned} \bar{x} &= \frac{m_{10}}{m_{00}} \\ \bar{y} &= \frac{m_{01}}{m_{00}} \\ \alpha &= \sqrt{\frac{m_{00}}{\beta}} \end{aligned}\right\} \tag{2.7}$$

Thus, for any image distribution $f(x,y)$, the lowest order, normalized central moments are

$$\mu_{00} = \beta \quad and \quad \mu_{10} = \mu_{01} = 0 \tag{2.8}$$

An alternate expression for the normalized central moments is obtained by rewriting eq. (2.6) in the form

$$\mu_{pq} = \frac{\beta}{[m_{00}]^{\gamma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)(x-\bar{x})^p (y-\bar{y})^q \, dx\,dy \quad ; p,q \geq 0 \ and \ \gamma = 1 + \frac{p+q}{2} \tag{2.9}$$

Taken together, eqs. (2.6) and (2.9), although equivalent, indicate that normalized, central moments may be viewed in two distinct ways. In the first view, the axes of the coordinate

system are scaled and its origin shifted while leaving the image distribution as given (eq. (2.9)). The second view treats the coordinate system as fixed and scales and shifts the image distribution (eq. (2.6)). This difference in point of view can become significant in practice where only a finite portion of the image plane is specified and where, for real world data, the background of the image distribution is non-uniform. For SAR imagery, for example, a potential target provided as a small segment of the overall image may require that additional background be acquired or synthesized in order to process the feature vector extraction using a uniform sized image segment which has been centered on the object's "center of optical mass" as dictated by either form given above of the μ_{pq} definition. In the present work, the computer generated images consist of black objects on white backgrounds and so this question of background "padding" or segment size simply does not arise. Note that the choice of normalization above is not unique; other normalization criteria are found in the literature[12,53,114,86,96,73,92,114,116,130]. Although not presented here, second order algebraic moments are related to moments of inertia, third order moments to image skewness, and fourth order moments to image kurtosis[7,12,73,86,114,116,130].

The irradiance normalized, central moments μ_{pq} defined above are invariant to changes in the scale and position of the image. They are not, however, invariant to the (in-plane) orientation of the image object. To achieve this invariance, Hu employed a mathematical technique referred to as the theory of algebraic invariants in conjunction with expressions relating central moments for different image rotations to derive a series of algebraic relations among the μ_{pq} which are invariant to changes in the image's angle of orientation. Such relations or, as in the case of the other bases considered in the remainder of this chapter, the moments themselves are referred to as "moment invariants". Hu derived seven such higher order expressions and then demonstrated[53,54] for a simple test case that these moment invariants could be used to classify images independently of their scale, position, and orientation. These expressions are not repeated here; the interested reader may consult the articles by Hu[53,54] or any of several other articles[7,12,73,86,114,130] which offer reviews of this approach. Other researchers[25,32,82,105,119,125,129] have also demonstrated the usefulness of Hu's moments as well as offering alternative methods for the calculation of algebraic moments[28,38,52,56,68,69,79,84,99,119,131,133].

Several researchers[116,1,121,122,7,12,86] have examined, from an information-theoretic point of view, the performance of various sets of basis functions for the extraction of moment invariants as well as their robustness (i.e., sensitivity to additive noise). The algebraic moments of Hu suffer from two fundamental shortcomings which severely limit their applicability to any but the simplest of image classification problems. The first and most limiting of these shortcomings is the fact that the moments μ_{pq}, being derived from the non-orthogonal basis functions $H_{pq}(x,y)$, are not independent of each other. Higher order moments may be viewed as composed of two parts; one is a redundant part containing

information already present in lower order moments while the other is that portion which does contain 'new' information. Unfortunately, as clearly demonstrated by Teh and Chin[116], the ratio of new to redundant information in the moments rapidly falls towards zero as the moment order increases. Thus, it would not be pragmatically feasible to work with algebraic moments in any problem requiring a relatively large feature vector dimension. The second shortcoming of the Hu algebraic moments is their acute sensitivity to the presence of image noise[116,114]. This report suggests a novel explanation for such sensitivity, namely that the measure of the basis functions, eq. (2.3), is non-zero for many of the moments. When additive noise is present, the fact that a particular basis function does not integrate to zero means that even modest amounts of random noise will severely affect the feature vector component (and, in the case of Hu moments, all higher order moments which possess information redundant with it) associated with that particular basis function. This proposal will be illustrated more clearly in Chapter 4 with the behavior of the moments for the SZRP and PZRP bases.

2.3 : Zernike Moments

2.3.1 : The Standard Zernike Function

The set of standard Zernike functions $SZF_{nm}(x,y)$ originates with the desire to derive a set of basis functions with the following characteristics[15,17,11,62]:

(1) the set of functions is complete and orthogonal on the unit circle;
(2) each function is a polynomial in the two variables $x = r\cos\theta$ and $y = r\sin\theta$;
(3) the set of functions transform into themselves under a representation of the two-dimensional rotation group (in less abstract terms, a set of functions which are invariant in form with respect to a rotation of axes about the origin); and
(4) the set of functions contains one member for each permissible combination of the indices n and m.

Together, these conditions lead to a definition of the standard Zernike function $SZF_{nm}(x,y)$ of order n and repetition m as a complex-valued polynomial in the two variables $x = r\cos\theta$ and $y = r\sin\theta$ given by

$$SZF_{nm}(x,y) = SZF_{nm}(r,\theta) = SZRP_{nm}(r)e^{im\theta} \quad ;x^2 + y^2 \le 1, |m| \le n, (n-m)\,even \qquad (2.10)$$

where the real-valued, standard Zernike radial polynomial is given by, for $0 \le m \le n$,

$$SZRP_{n,\pm m}(r) = \sum_{k=0}^{(n-m)/2} S_{nmk} r^{n-2k} \tag{2.11}$$

with

$$S_{nmk} = (-1)^k \frac{(n-k)!}{k![(n+m-2k)/2]![(n-m-2k)/2]!} \tag{2.12}$$

The standard Zernike polynomials form a complete, orthogonal basis on the unit circle with

$$\left.\begin{aligned} \iint_{x^2+y^2\le 1} SZF_{nm}(x,y)SZF_{jk}^*(x,y)dxdy &= \int_0^1 \int_0^{2\pi} SZF_{nm}(r,\theta)SZF_{jk}^*(r,\theta)\, r dr d\theta \\ &= \left\{\int_0^1 SZRP_{nm}(r)SZRP_{jk}(r)\, r dr\right\}\left\{\int_0^{2\pi} e^{im\theta}e^{-ik\theta}d\theta\right\} \\ &= \left\{\frac{1}{2(n+1)}\delta_{nj}\right\}\{2\pi\delta_{mk}\} \\ &= \frac{\pi}{n+1}\delta_{nj}\delta_{mk} \end{aligned}\right\} \tag{2.13}$$

where, since $S_{n,-m,k} = S_{nmk}$, we have $(SZF_{nm})^* = SZF_{n,-m}$. Note that the radial polynomials themselves are orthogonal and independent on the unit circle. The transformation of $SZF_{nm}(r,\theta)$ under a rotation of coordinates through an angle φ to new coordinates (x',y') given by

$$\begin{aligned} x' &= x\cos\varphi + y\sin\varphi \\ y' &= -x\sin\varphi + y\cos\varphi \end{aligned}$$

is particularly simple, taking the form of a phase change only, i.e.,

$$SZF_{nm}(x',y') = SZF_{nm}(r,\theta+\varphi) = e^{im\varphi}SZF_{nm}(x,y) \tag{2.14}$$

It is this preservation of form under rotation which makes the Zernike functions particularly suitable as a basis for rotation invariant features. Note that

$$SZRP_{nm}(r=1)=+1 \qquad \forall\, n,m \tag{2.15}$$

$$SZRP_{nm}(r=0)=(-1)^k \delta_{n,2k}\delta_{m,0} \quad ;k=0,1,2,... \tag{2.16}$$

The first few standard Zernike radial polynomials are:

$$\left.\begin{aligned}
&SZRP_{00}(r)=1\\
&SZRP_{11}(r)=r\\
&SZRP_{20}(r)=2r^2-1\\
&SZRP_{22}(r)=r^2\\
&SZRP_{31}(r)=3r^3-2r\\
&SZRP_{33}(r)=r^3\\
&SZRP_{40}(r)=6r^4-6r^2+1\\
&SZRP_{42}(r)=4r^4-3r^2\\
&SZRP_{44}(r)=r^4
\end{aligned}\right\} \tag{2.17}$$

The radial polynomials for $n = 7$ and $n = 8$ are shown in figures 2.1 and 2.2, respectively, as 2-D plots versus the variable r while figures 2.3 and 2.4 illustrate the 3-D plots of SZRP for the cases $\{n,m\} = \{7,3\}$ and $\{8,0\}$, respectively. Figures 2.5 and 2.6 show the real part of the Zernike function over the unit circle for the cases $\{n,m\} = \{7,5\}$ and $\{8,2\}$. Note that, from their definition (2.10), the real and imaginary parts of the SZF for $m \neq 0$ are simply related as

$$\mathrm{Im}\left[SZF_{nm}(r,\theta)\right]=\mathrm{Re}\left[SZF_{nm}(r,\theta-\frac{2\pi}{m})\right] \quad ;\; m\neq 0 \tag{2.18}$$

i.e., the real and imaginary parts of the SZF are identical in form, the imaginary part equal to the real part rotated through the angle $(2\pi/m)$.

A parameter of significance in the present work is the integral equation giving the measure or area of the Zernike radial polynomial over the unit circle as

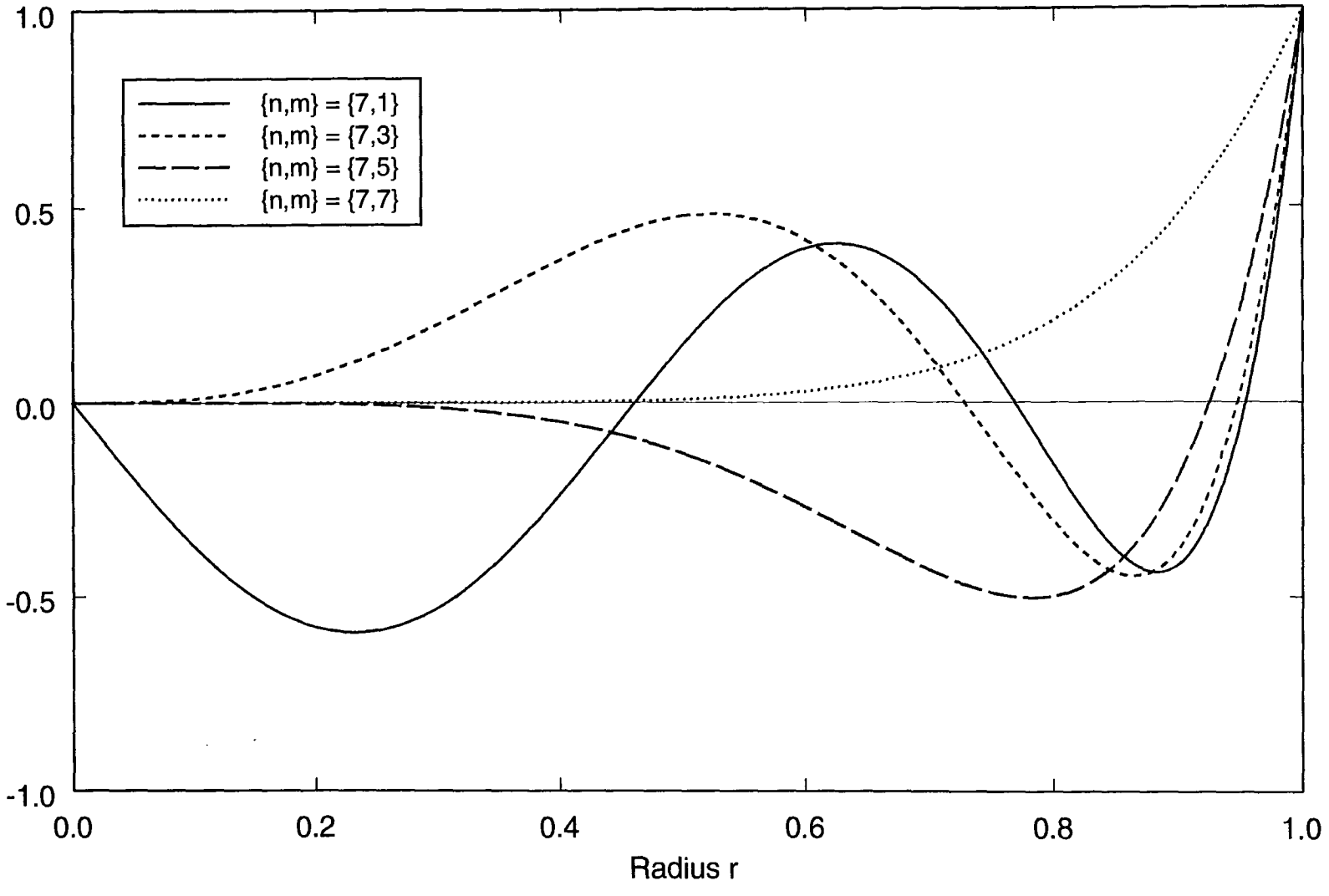


Figure 2.1

The Standard Zernike Radial Polynomial with n = 7

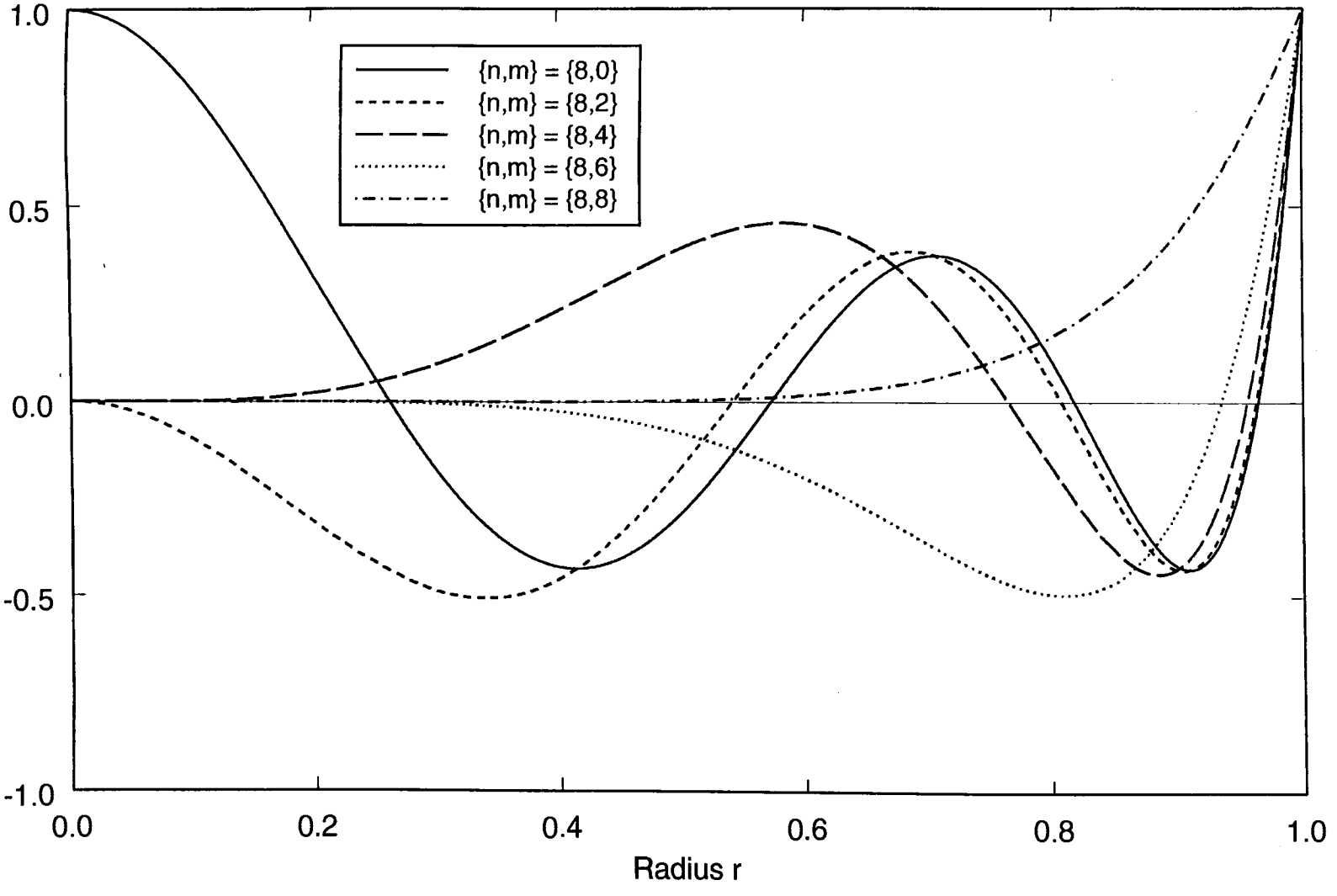


Figure 2.2

The Standard Zernike Radial Polynomial with n = 8

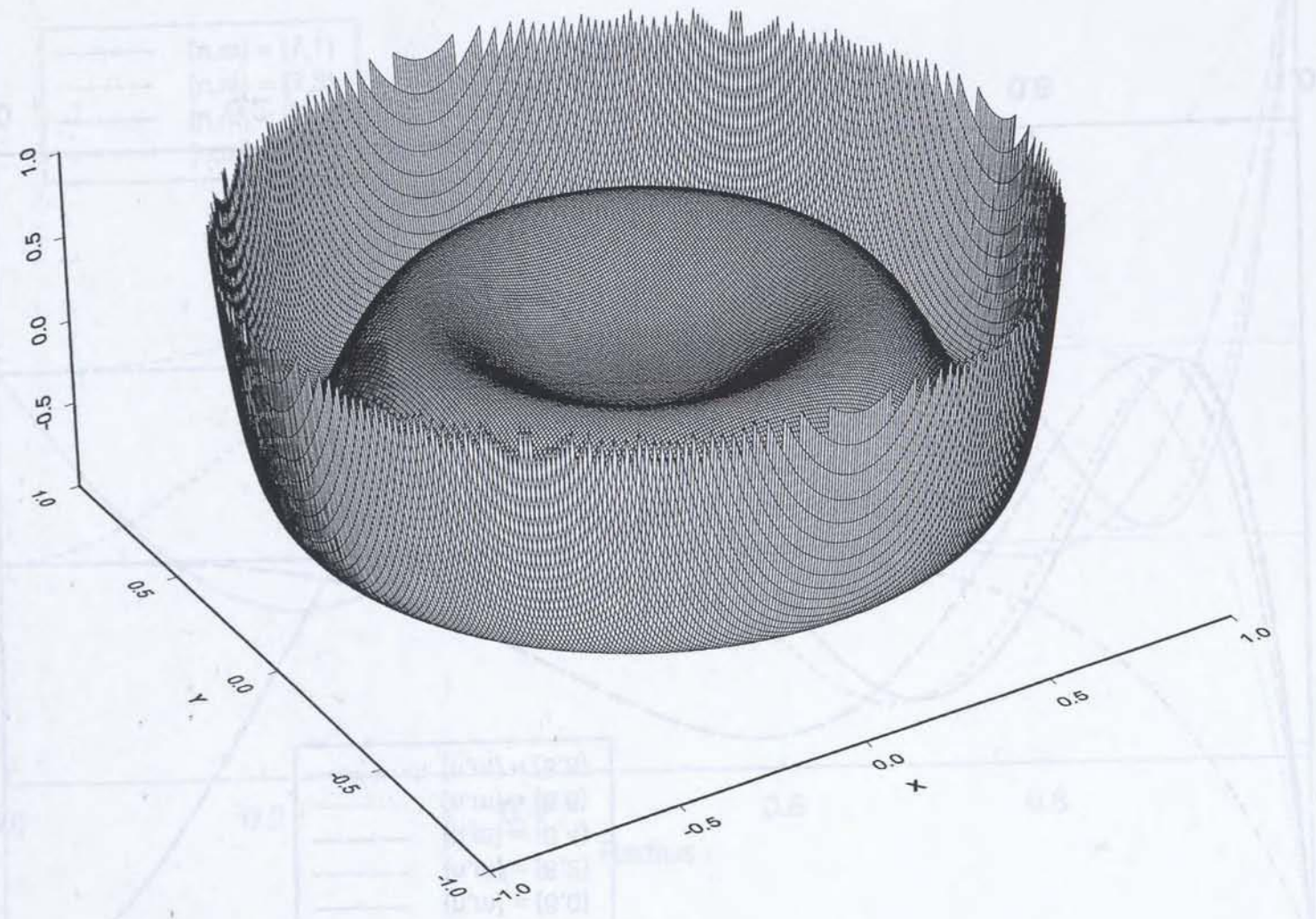


Figure 2.3

SZRP(r) with n = 7 and m = 3

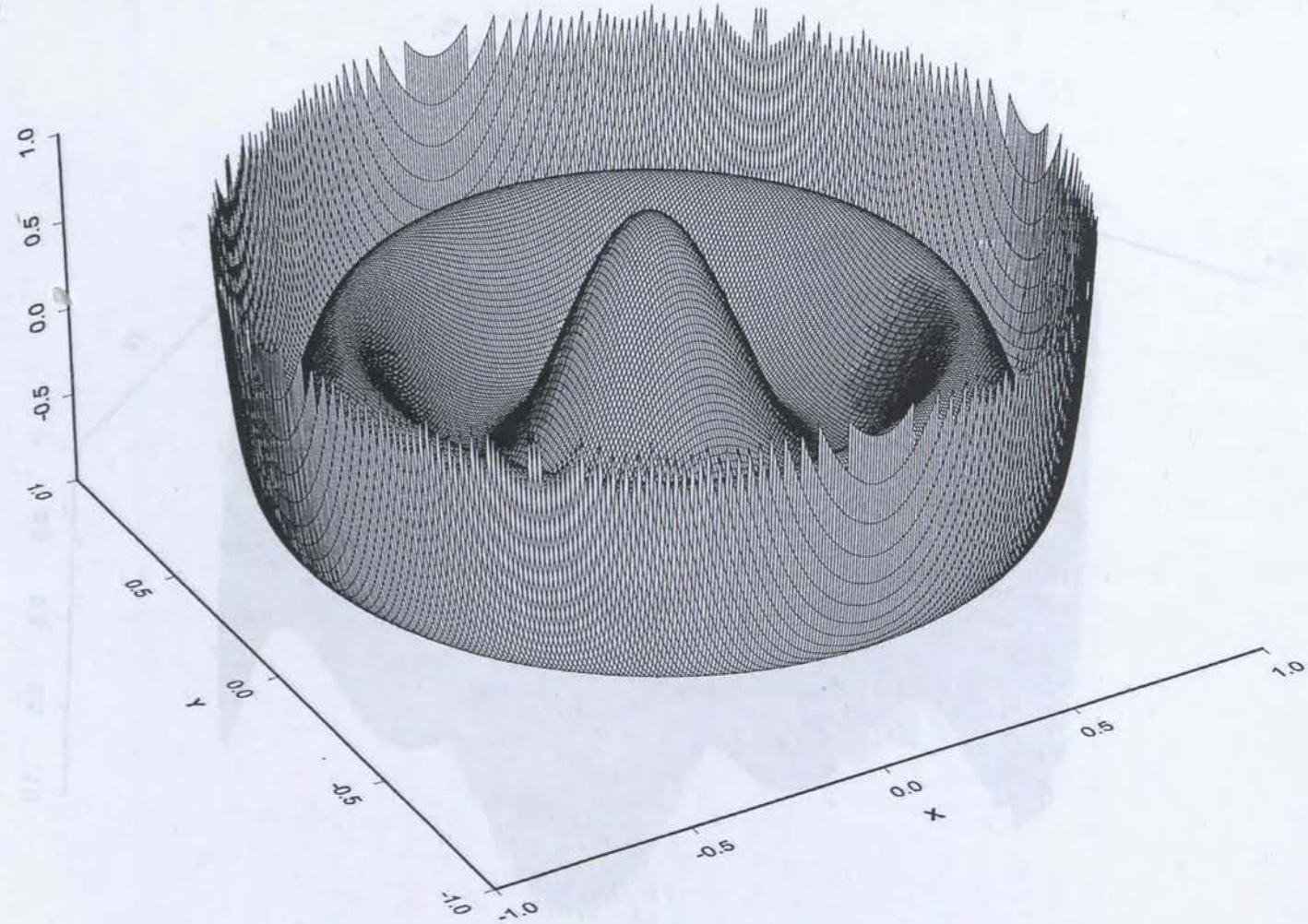


Figure 2.4

SZRP(r) with n = 8 and m = 0

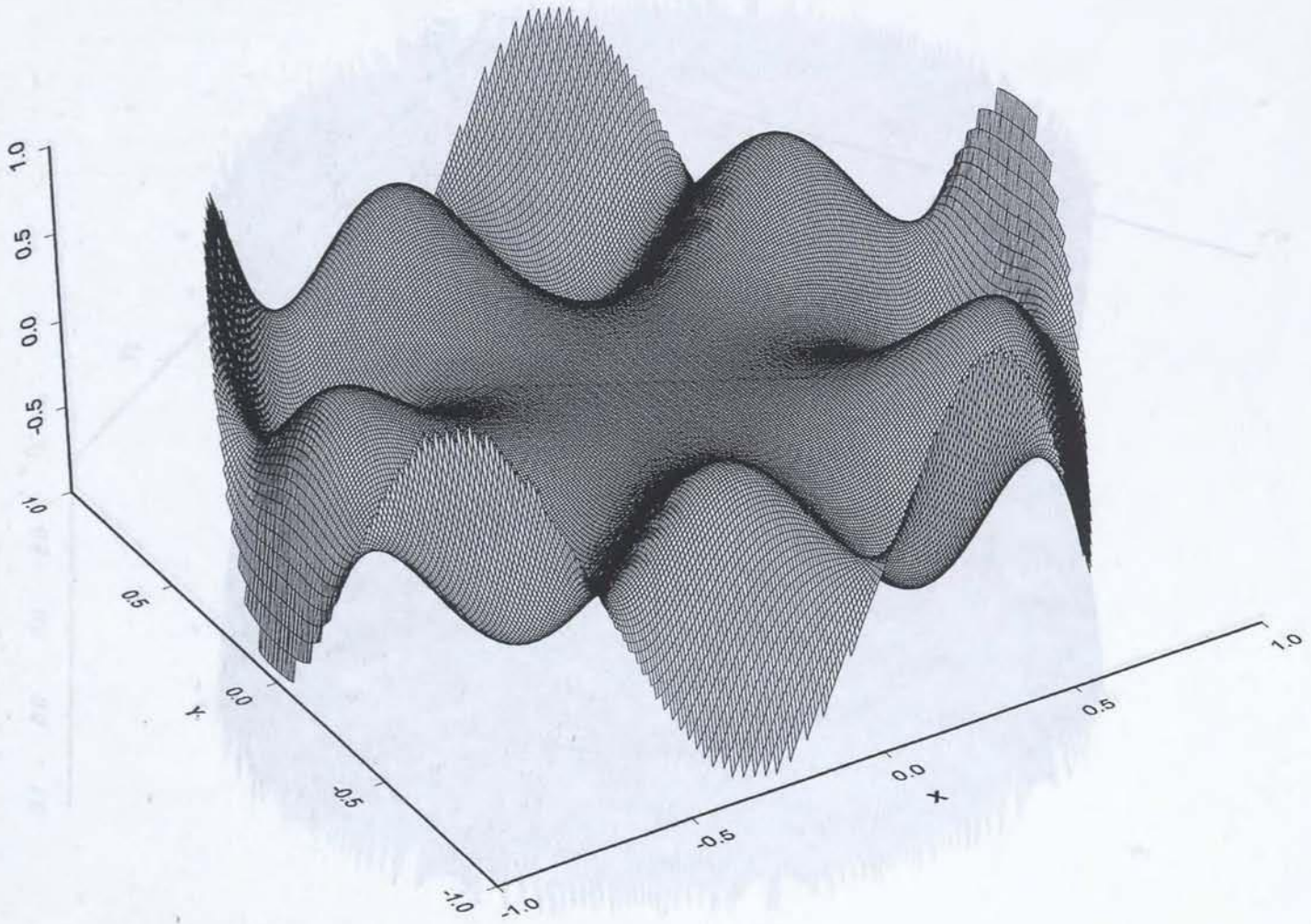


Figure 2.5

Re{SZF(x,y)} with n = 7 and m = 5

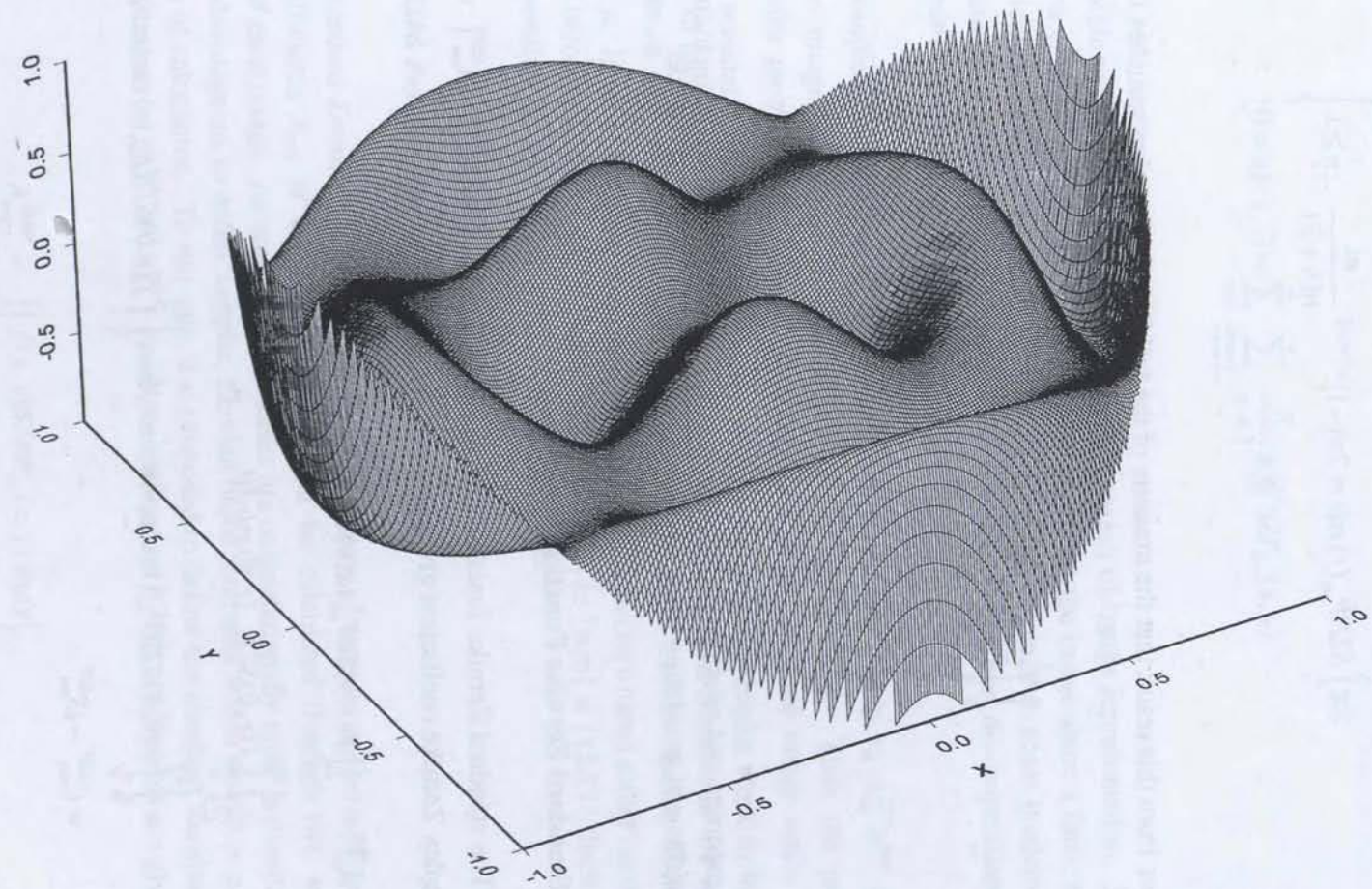


Figure 2.6

Re{SZF(x,y)} with n = 8 and m = 2

$$\left.\begin{aligned} 2\pi\int_0^1 SZRP_{nm}(r)\,rdr &= 2\pi(-1)^{(n-m)/2}\frac{m}{n(n+2)} \quad ;n\geq 1 \\ &= \pi \quad ;n=0 \end{aligned}\right\} \tag{2.19}$$

It follows from this result that the measure of the full Zernike function vanishes for $n \geq 1$, i.e.,

$$\left.\begin{aligned} \int_0^1\int_0^{2\pi} SZF_{nm}(r,\theta)rdrd\theta &= 2\pi(-1)^{(n-m)/2}\frac{m}{n(n+2)}\delta_{m0} \\ &= 0 \quad \forall n,m \; with \; n\geq 1 \\ &= \pi \quad ; n=0 \end{aligned}\right\} \tag{2.20}$$

Other properties and relationships for the standard Zernike polynomials, including recurrence relationships and generating functions, may be found in the literature[11,15,17,62].

2.3.2 : Standard Zernike Function Moments

The standard Zernike function moment is here defined as $M_{nm}^{SZF} = \left|A_{nm}^{SZF}\right|$ where A_{nm}^{SZF} is the complex Zernike coefficient given by

$$\left.\begin{aligned} A_{nm}^{SZF} &= \iint_{x^2+y^2\leq 1} f(x,y)SZF_{nm}^{*}(x,y)\,dxdy \\ &= \int_0^1\int_0^{2\pi} f(r,\theta)SZF_{nm}^{*}(r,\theta)\,rdrd\theta \\ &= \int_0^1\int_0^{2\pi} f(r,\theta)SZRP_{nm}(r)\cos(m\theta)\,rdrd\theta - i\int_0^1\int_0^{2\pi} f(r,\theta)SZRP_{nm}(r)\sin(m\theta)rdrd\theta \\ &= C_{nm}^{SZF} - iS_{nm}^{SZF} \end{aligned}\right\} \tag{2.21}$$

where, henceforth, the image distribution function $f(x,y) = f(r,\theta)$ is assumed to be the scaled and shifted version (as for eq. (2.9)) such that $\mu_{00} = \beta$ and $\mu_{10} = \mu_{01} = 0$. Since the

SZF form a complete, orthogonal basis set, the image function $f(x,y)$ may be expressed as a generalized Fourier series over the SZF, i.e.,

$$f(x,y)=\sum_{n=0}^{\infty}\sum_{\substack{m=-n \\ (n-m)\,is\,even}}^{n}\frac{\pi}{n+1}A_{nm}^{SZF}SZF_{mn}(x,y) \tag{2.22}$$

It is thus possible to use the Zernike expansion as a means of image representation. Indeed, one application of this Zernike function series is to compress the image since a finite number of the coefficients may be used in eq. (2.22) to reconstruct the image to some predetermined degree of accuracy. Examples of this type of generalized Fourier decomposition using Zernike functions have been given by different researchers[116,73,122].

The complex coefficients A_{nm}^{SZF} defined above transform simply to $A_{nm}^{SZF}e^{im\phi}$ upon a change in the image orientation (counterclockwise) by the angle ϕ. Thus the principal standard Zernike moment $M_{nm}^{SZF}=\left|A_{nm}^{SZF}\right|$ is independent of the original image scale, lateral position, and orientation and it is this quantity which is used in the present work to form the feature vector of the image. The feature vector derived from this basis consists of the series of scalar components M_{nm}^{SZF} ordered according to the primary index n with increasing values of m = 0 or 1, ..., n. In all of the work presented here, the feature vectors based either on the SZF or SZRP (see below) utilize the range $\{n,m\}=\{3,1\}$ through to $\{n,m\}=\{12,12\}$, inclusively, for a total dimension for the feature vector of 45 components.

2.3.3 : Standard Zernike Radial Polynomial Moments

The standard Zernike function moments of the previous section are derived from the complex coefficients A_{nm} of eq. (2.21) which must be calculated through two separate integrations of each image. As such, these moments are computationally quite demanding and it would be advantageous to define simpler, Zernike based moments which do not require the same amount of calculation. To this end, it is proposed to define the standard Zernike radial polynomial moments as $M_{nm}^{SZRP}=\left|A_{nm}^{SZRP}\right|$ where the real-valued SZRP coefficients are given by

$$\left.\begin{aligned}A_{nm}^{SZRP}&=\iint_{x^2+y^2\le 1}f(x,y)SZRP_{nm}(x,y)\,dxdy\\&=\int_0^1\int_0^{2\pi}f(r,\theta)SZRP_{nm}(r)\,rdrd\theta\end{aligned}\right\} \tag{2.23}$$

so that the SZRP feature vector is formed from the quantities M_{nm}^{SZRP} with the same range of $\{n,m\}$ values used for the SZF moments. The coefficient A_{nm}^{SZRP} is a real coefficient which, by itself, results in a major reduction in the amount of computation which must be done both for the basis functions and for the formation of the feature vectors. It should be noted, as shown in section 2.3.1, that the functions $SZRP_{nm}(r)$ and hence the SZRP moments M_{nm}^{SZRP} are independent.

Note that, by suppressing the (complex) angular dependence $e^{im\phi}$ to arrive at the simpler SZRP moments, it is not possible to reconstruct the image $f(x,y)$ from a knowledge of the A_{nm}^{SZRP} of eq. (2.23) alone as was the case with the SZF coefficients. However, the present work is concerned solely with the task of accurate image classification; questions of image reconstruction or image compression have no role to play in the present considerations. In short, if the SZRP moments were to prove as effective and accurate in the image classification task, they would be perfectly acceptable as moment invariant feature vectors.

2.3.4: The Pseudo Zernike Function

The pseudo Zernike functions $PZF_{nm}(r,\theta)$ are derived[15,17] from the same set of conditions as those prescribed previously for the standard Zernike functions with the one exception that condition (2) is relaxed to permit a function which is a polynomial in the three variables, x, y, and r. The resulting functions are found to take the form

$$PZF_{nm}(x,y,r) = PZF_{nm}(r,\theta) = P_{nm}(r)e^{im\theta} \quad ; x^2+y^2 \le 1, \quad |m| \le n \tag{2.24}$$

where the real valued pseudo Zernike radial polynomial is given by, with $0 \le m \le n$,

$$PZRP_{n,\pm m}(r) = \sum_{k=0}^{n-m} P_{nmk} r^{n-k} \tag{2.25}$$

with

$$P_{nmk} = (-1)^k \frac{(2n+1-k)!}{k!(n+m+1-k)!(n--m-k)!} \tag{2.26}$$

Note the lifting of the restriction that $(n\text{-}m)$ be even in the case of the pseudo Zernike functions. As a consequence, the set of pseudo Zernike functions will have $(n+1)^2$ linearly independent polynomials of order $\leq n$ in contrast to the standard Zernike functions which will have only $\frac{1}{2}(n+1)(n+2)$ linearly independent polynomials of order $\leq n$.

Like the standard Zernike polynomials, the pseudo Zernike polynomials form a complete, orthogonal basis on the unit circle with

$$\left.\begin{aligned}\iint_{x^2+y^2\leq 1} PZF_{nm}(x,y)PZF_{jk}^{*}(x,y)dxdy &= \int_0^1\int_0^{2\pi} PZF_{nm}(r,\theta)PZF_{jk}^{*}(r,\theta)\,rdrd\theta \\ &= \left\{\int_0^1 PZRP_{nm}(r)PZRP_{jk}(r)\,rdr\right\}\left\{\int_0^{2\pi} e^{im\theta}e^{-ik\theta}d\theta\right\} \\ &= \left\{\frac{1}{2(n+1)}\delta_{nj}\right\}\left\{2\pi\delta_{mk}\right\} \\ &= \frac{\pi}{n+1}\delta_{nj}\delta_{mk}\end{aligned}\right\} \quad (2.27)$$

with an equally simple rotational transformation analogous to that given in eq. (2.14), namely

$$PZF_{nm}(x',y',r) = PZF_{nm}(r,\theta+\varphi) = e^{im\varphi}PZF_{nm}(x,y,r) \quad (2.28)$$

Note that

$$PZRP_{nm}(r=1) = +1 \qquad \forall\, n,m \quad (2.29)$$

$$PZRP_{nm}(r=0) = (-1)^n(n+1)\delta_{m,0} \quad (2.30)$$

The first few pseudo Zernike radial polynomials are:

$$\left.\begin{aligned}
PZRP_{00}(r) &= 1 \\
PZRP_{10}(r) &= 3r-2 \\
PZRP_{11}(r) &= r \\
PZRP_{20}(r) &= 10r^2-12r+3 \\
PZRP_{21}(r) &= 5r^2-4r \\
PZRP_{22}(r) &= r^2 \\
PZRP_{30}(r) &= 35r^3-60r^2+30r-4 \\
PZRP_{31}(r) &= 21r^3-30r^2+10r \\
PZRP_{32}(r) &= 7r^3-6r^2 \\
PZRP_{33}(r) &= r^3
\end{aligned}\right\} \tag{2.31}$$

The pseudo Zernike radial polynomials for $n = 3$ and $n = 4$ are shown in figures 2.7 and 2.8, respectively, as 2-D plots over the variable r while the 3-D plots of the PZRP for the cases $\{n,m\} = \{3,1\}$ and $\{4,2\}$ are illustrated in figures 2.9 and 2.10, respectively. Figures 2.11 and 2.12 show the real part of $PZF_{nm}(r,\theta)$ for $\{n,\mathrm{m}\} = \{3,2\}$ and $\{4,1\}$. Note that, as with the SZF, the real and imaginary components of PZF are related simply by

$$\mathrm{Im}\left[PZF_{nm}(r,\theta)\right] = \mathrm{Re}\left[PZF_{nm}(r,\theta-\frac{2\pi}{m})\right] \;\; ; \; m \neq 0 \tag{2.32}$$

The integral equation giving the measure or area of the pseudo Zernike radial polynomial over the unit circle takes the form

$$\left.\begin{aligned}
2\pi\int_0^1 PZRP_{nm}(r)\,r\,dr &= 2\pi(-1)^{n-m}\frac{m(m+1)}{n(n+1)(n+2)} \quad ;n \geq 1 \\
&= \pi \qquad\qquad\qquad\qquad\qquad\quad ;n = 0
\end{aligned}\right\} \tag{2.33}$$

It follows from this result that the measure of the full pseudo Zernike function vanishes for $n \geq 1$

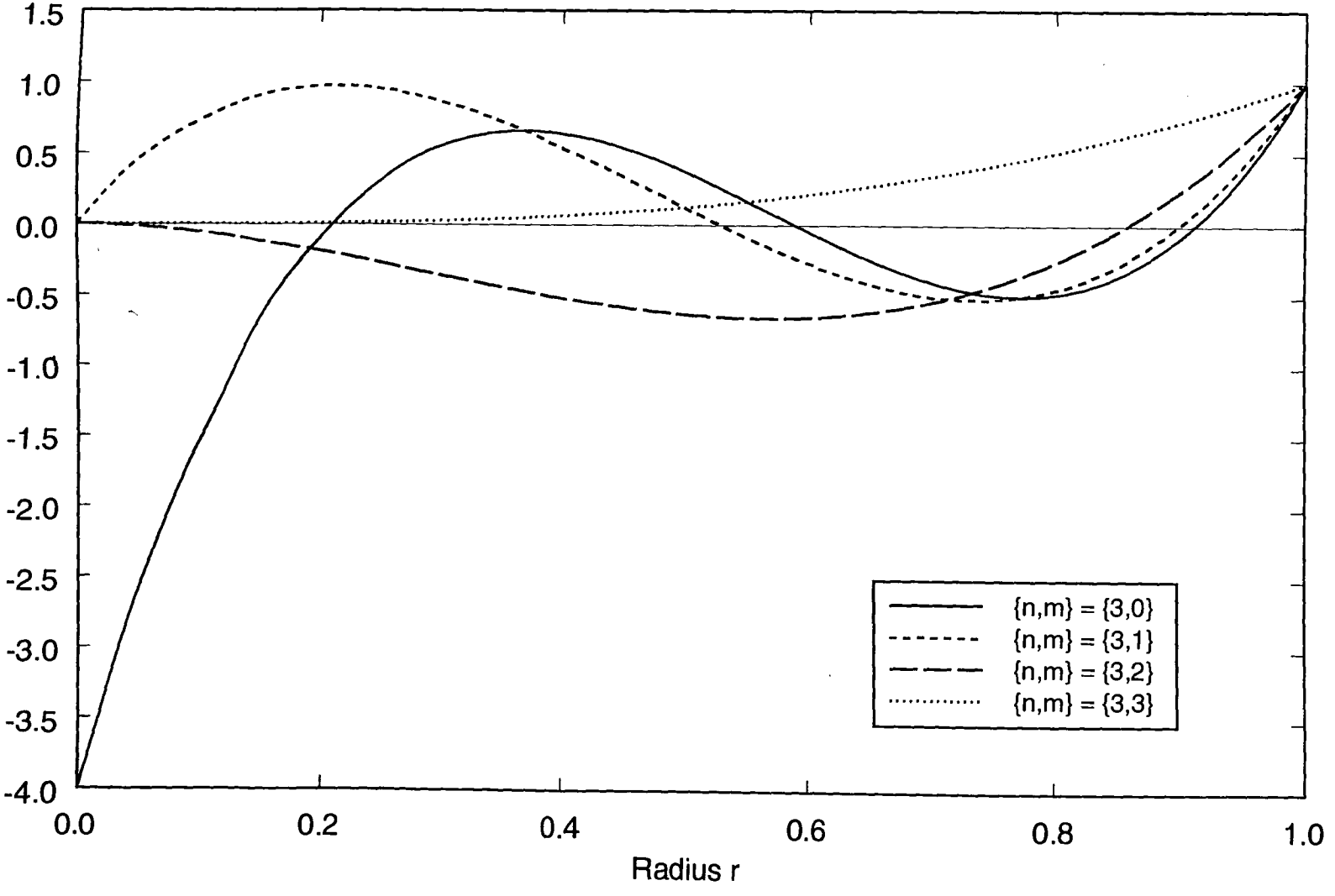


Figure 2.7

The Pseudo Zernike Radial Polynomial with n = 3

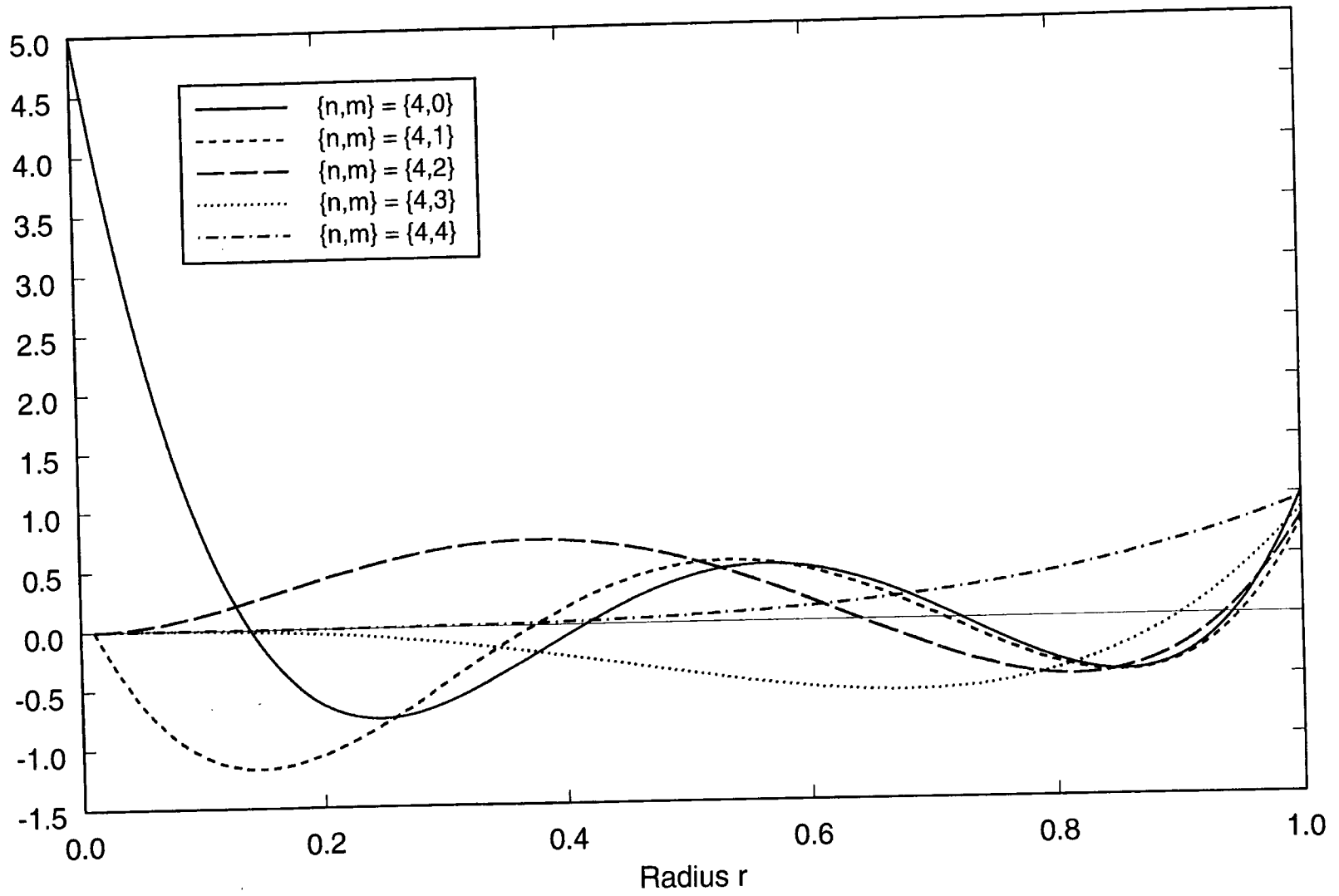


Figure 2.8

The Pseudo Zernike Radial Polynomial with n = 4

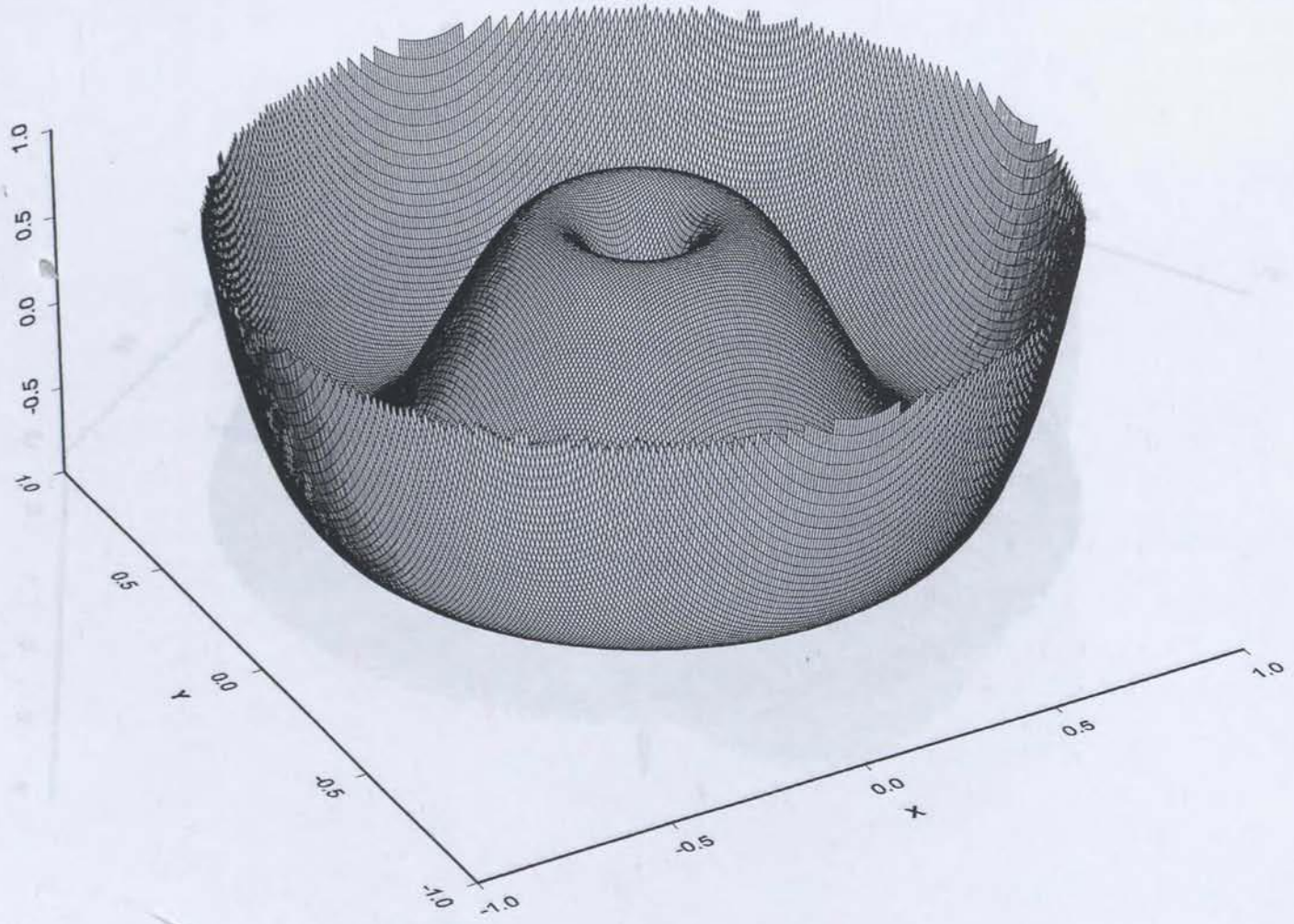


Figure 2.9

PZRP(r) with n = 3 and m = 1

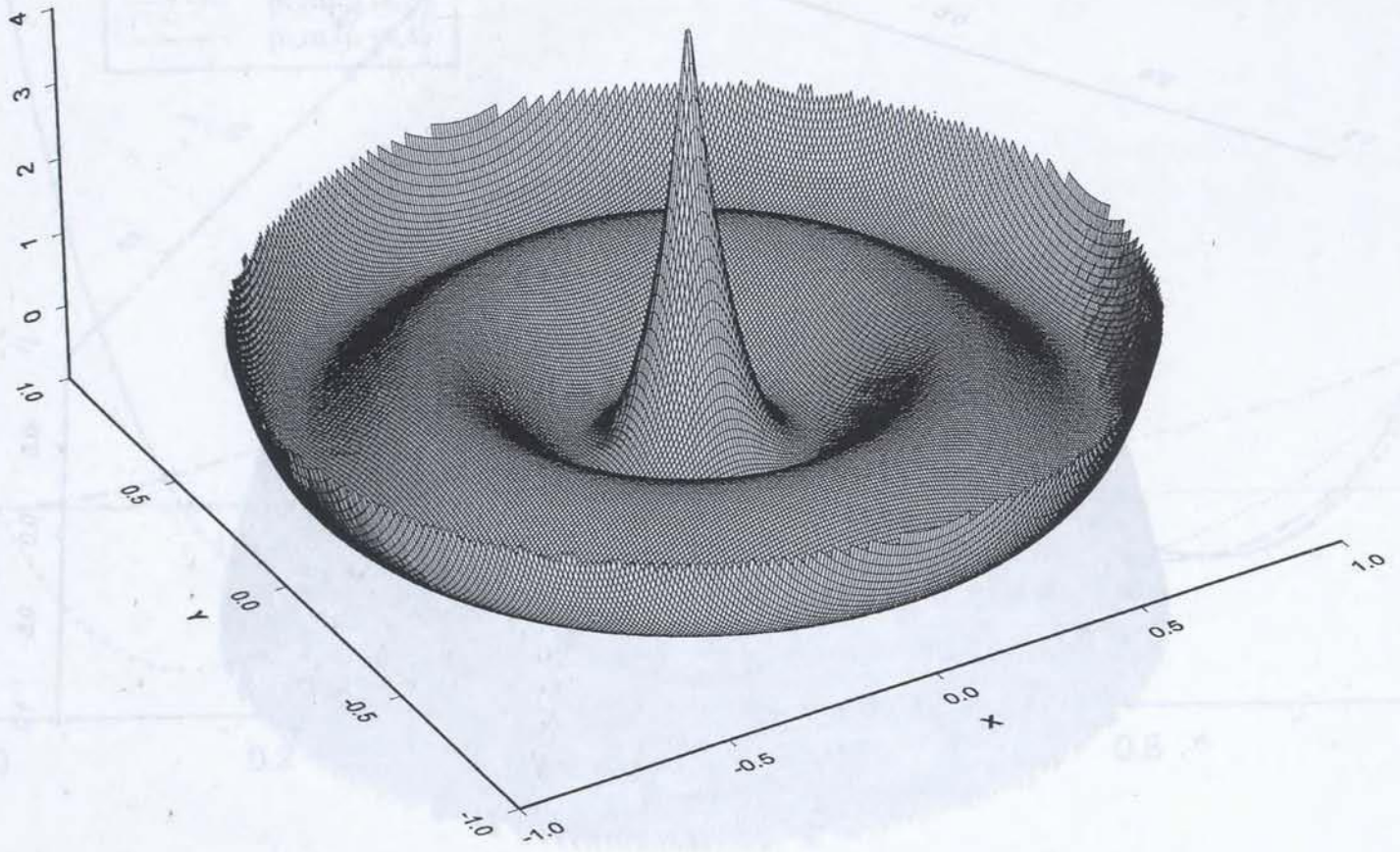


Figure 2.10

PZRP(r) with n = 4 and m = 0

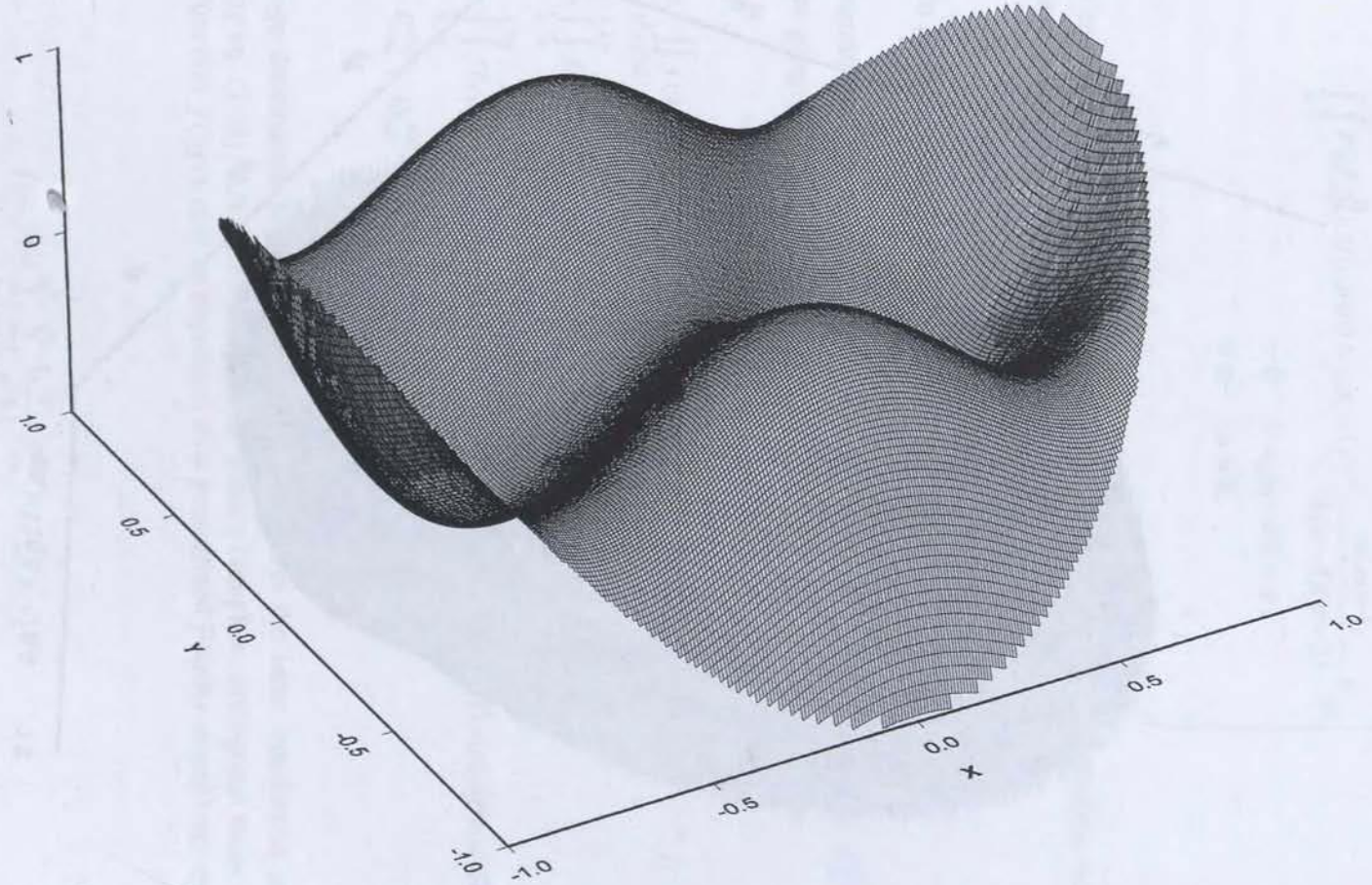


Figure 2.11

Re{PZF(x,y)} with n = 3 and m = 2

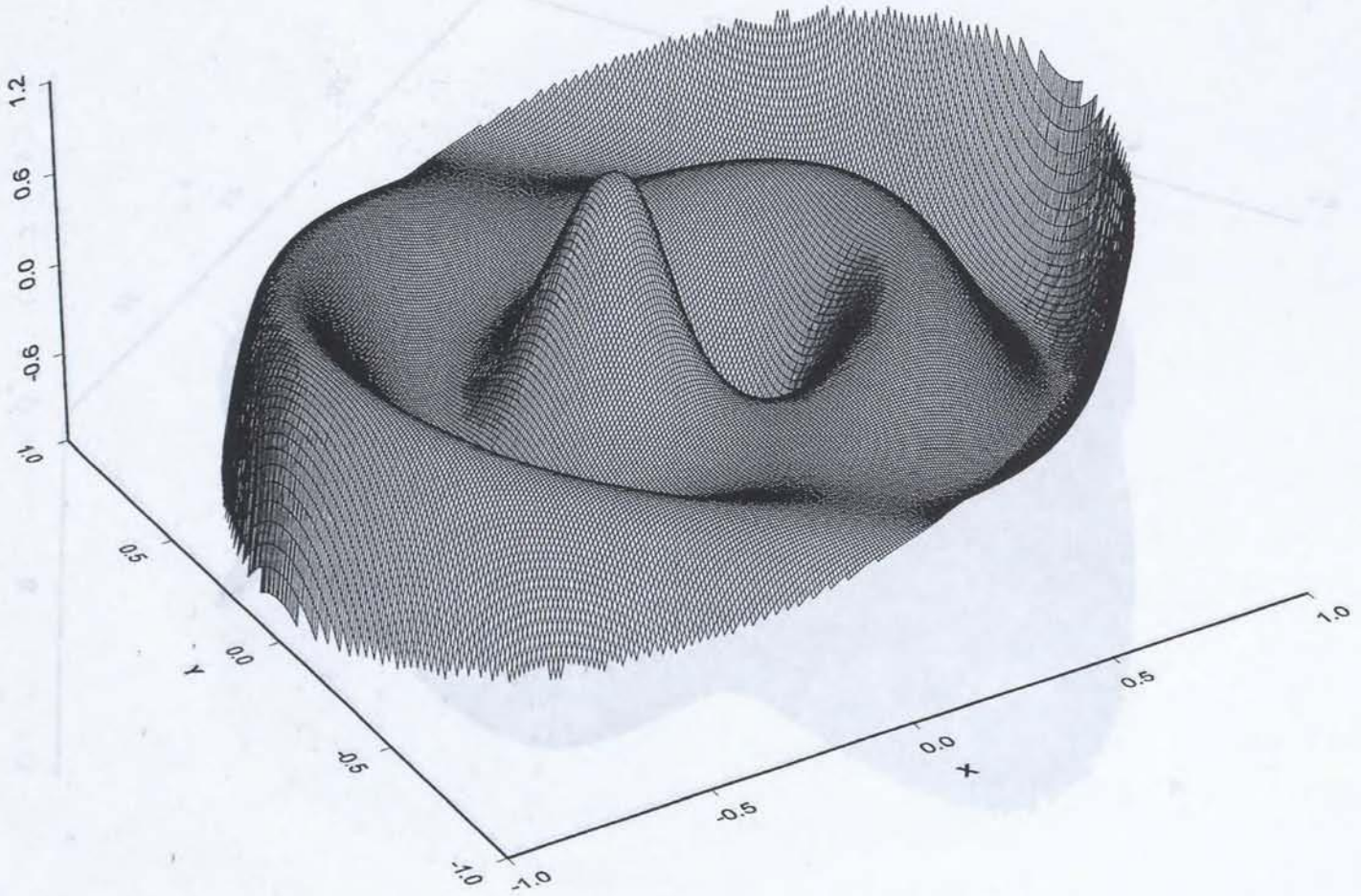


Figure 2.12

Re{PZF(x,y)} with n = 4 and m = 1

$$\left.\begin{aligned}\int_0^1\int_0^{2\pi} PZF_{nm}(r,\theta)rdrd\theta &= 2\pi(-1)^{n-m}\frac{m(m+1)}{n(n+1)(n+2)}\delta_{m0}\\ &= 0 \quad \forall\, n,m \text{ with } n\geq 1\\ &= \pi \quad ;n=0\end{aligned}\right\} \tag{2.34}$$

Other properties and relationships for the pseudo Zernike polynomials, including recurrence relationships and generating functions, may be found in the literature[11,15,17].

2.3.5 : Pseudo Zernike Function Moments

The pseudo Zernike function moment is defined as $M_{nm}^{PZF} = \left|A_{nm}^{PZF}\right|$ where A_{nm}^{PZF} is the PZF coefficient given by

$$\left.\begin{aligned}A_{nm}^{PZF} &= \iint_{x^2+y^2\leq 1} f(x,y)PZF_{nm}^{*}(x,y)\,dxdy\\ &= \int_0^1\int_0^{2\pi} f(r,\theta)PZF_{nm}^{*}(r,\theta)\,rdrd\theta\\ &= \int_0^1\int_0^{2\pi} f(r,\theta)PZRP_{nm}(r)\cos(m\theta)\,rdrd\theta - i\int_0^1\int_0^{2\pi} f(r,\theta)SZRP_{nm}(r)\sin(m\theta)rdrd\theta\\ &= C_{nm}^{PZF} - iS_{nm}^{PZF}\end{aligned}\right\} \tag{2.35}$$

where the image distribution function $f(x,y) = f(r,\theta)$ obeys the same conditions as those stated following eq. (2.21). Like the SZF, the PZF form a complete, orthogonal basis set and so the image function $f(x,y)$ may be expressed as a generalized Fourier series over the PZF, i.e.,

$$f(x,y) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{\pi}{n+1}A_{nm}^{PZF}PZF_{mn}(x,y) \tag{2.36}$$

Applications of the full pseudo Zernike coefficients to represent and compress an image have been reported by different researchers[116,73,122].

The complex coefficients A_{nm}^{PZF} defined above transform simply to $A_{nm}^{PZF}e^{im\phi}$ upon a change in the image orientation (counterclockwise) by the angle ϕ. Thus the principal pseudo Zernike moment $M_{nm}^{PZF} = \left|A_{nm}^{PZF}\right|$ is independent of the original image scale, lateral position, and orientation and it is this quantity which is used in the present work to form the feature vector of the image. The feature vector derived from this basis consists of the series of scalar components M_{nm}^{PZF} ordered according to the primary index n with increasing values of m = 0, ..., n. In all of the work presented here, the feature vectors based either on the PZF or PZRP (see below) utilize the range $\{n,m\}$ = {3,0} through to $\{n,m\}$ = {9,5} inclusively for a total feature vector dimension of 45.

2.3.6 : Pseudo Zernike Radial Polynomial Moments

Exactly as was the case for the SZF moments, the pseudo Zernike function moments of the previous section are derived from the complex coefficients A_{nm}^{PZF} of eq. (2.35) which must be calculated through two separate integrations of each image. It would be computationally advantageous to define simpler, Zernike based moments which do not require the same amount of calculation. To provide such a simpler alternative, it is proposed to define the pseudo Zernike radial polynomial moments as $M_{nm}^{PZRP} = \left|A_{nm}^{PZRP}\right|$ where the real-valued PZRP coefficient is given by

$$\left.\begin{aligned} A_{nm}^{PZRP} &= \iint_{x^2+y^2\le 1} f(x,y)PZRP_{nm}(x,y)\,dxdy \\ &= \int_0^1\int_0^{2\pi} f(r,\theta)PZRP_{nm}(r)\,rdrd\theta \end{aligned}\right\} \tag{2.37}$$

so that the PZRP feature vector is formed from the quantities M_{nm}^{PZRP} with the same range of $\{n,m\}$ values used for the PZF moments. The coefficient A_{nm}^{PZRP} is a real coefficient which results in a substantive reduction in the amount of computation which must be done both for the basis functions and for the formation of the feature vectors. It should be noted, as shown in section 2.3.1, that the functions $PZRP_{nm}(r)$ and hence the PZRP moments M_{nm}^{PZRP} are independent.

Note that, by suppressing the (complex) angular dependence $e^{im\phi}$ to arrive at the simpler PZRP moments, it is not possible to reconstruct the image $f(x,y)$ from a knowledge of the A_{nm}^{PZRP} of eq. (2.37) alone as was the case with the PZF coefficients. However, the present work is concerned only with the task of accurate image classification; questions of image reconstruction or image compression play no role in the present considerations. Thus, as for the SZRP case, if the PZRP moments were to prove as effective and accurate in the image classification task, they would be completely acceptable as moment invariant feature vectors.

2.3.7: General Comments on the Zernike Moments

There are three aspects to the definition of the various Zernike feature vector components as given above which deserve further comment.

(1) The first concerns the normalization of these coefficients. Different researchers[1,2,12,25,86,96,99,114,122,144] have described renormalization schemes for the derived moments which are expected to noticeably decrease the dynamic range which would typically exist for the basic moments such as those defined above. However, such renormalization schemes have been deliberately disregarded in this work for one fundamental reason. As will be discussed in more detail in the following chapter concerning the calculation of normalized feature vectors, it is necessary when dealing with an artificial neural network to ensure that the input values (in this case, the components of the feature vector) are scaled (i.e., normalized) to fall within a preset, limited range (typically -1 to +1)[49,30,4,23,95,126]. Thus, any prior "renormalization" of the feature vector components would be, in effect, eradicated and replaced by scaling parameters which are determined by the (training) data set as a whole. Such a prior renormalization amounts, in the present work, to unnecessary computation and complication and so was not implemented. Indeed, even the usual coefficient arising from the generalized Fourier analysis (the term $(n+1)/\pi$ in eqs. (2.22) and (2.36)) was omitted from the definition of the principal moment.

(2) A second aspect of the choice for the feature vector components in this work is that only the principal components as defined by eqs. (2.21), (2.23), (2.35) and (2.37) are used to form the feature vector. There exist, however, n independent moments which can be formed from nth-order Zernike functions. The principal moment is simply the most obvious and direct one. Teague[114], Wallin and Kübler[122] and other authors[12,35,79,86,92,96,114,119] discuss methodologies for forming all of the subsidiary moments of nth order. These subsidiary components, while independent of the principal moment, play a secondary role to the nth-order principal moment from an information-theoretic view. Thus, particularly given the increased complexity of deriving the form of such subsidiary terms, only the principal

moments were employed in the present work (a custom, it appears, followed by the majority of researchers[1,2,58-61,72,73] who utilize moment invariants for image classification studies).

(3) Thirdly and lastly, it should be noted that a fourth type of invariance, namely invariance to mirror transformation(s)[114,122,1,12,65,130] of the image is obeyed by the principal moments (Teague's "true invariants"[114]) but is generally not obeyed for the subsidiary moments mentioned above (Teague's "pseudo invariants"[114]). This observation will be used in the discussion of the experimental results on the classification of the test images used in the present work where two of the images (the numerals '2' and '5') are *approximately* mirror images of one another.

2.4 : Walsh Moments

2.4.1 : Walsh Functions, Walsh Radial Functions, and Walsh Radial Function Moments

The family of Walsh functions is composed of a variety of different members which share two common characteristics. Firstly, each member of this family represents an ordered set of rectangular waveforms which, generally speaking, form a complete and orthogonal series of functions. Secondly, Walsh functions assume only a finite number of amplitude values, being piecewise continuous with a finite number of finite discontinuities at those points where the amplitude changes (for mathematical rigor, Walsh functions are defined to assume the value of 0 at the discontinuities). Some of the better known members of this family include the Walsh, Haar, slant, and Walsh-Haar functions[33,34,50,64,77,111,123]. These functions and transforms based upon them (the fast Walsh and Hadamard transforms) have been studied extensively and have found many applications in the fields of communications[45,77,46], image analysis and compression[20,77,85], and image feature representation[29,36,20,77,85]. Considerable research efforts have also been placed into designing and fabricating specialized circuitry and hardware to calculate the functions themselves or one of the variety of fast transforms based upon them. The reader is referred to the texts by Beauchamp[8,9] and Harmuth[46,47] which provide detailed expositions of the entire family of Walsh functions along with many examples of their applications. The brief exposition given in this section adopts the notational conventions found in the texts by Beauchamp[8,9].

Among the variety of ways to define and calculate the Walsh functions, perhaps the simplest and most direct is through the use of Rademacher functions which are defined as

$$RAD(n,x) = sign\{\sin(2^n \pi x)\} \tag{2.38}$$

where the *sign*(y) function is +1 for $y > 0$, -1 for $y < 0$, and equal to 0 when $y = 0$. The integer $n \geq 0$ is referred to as the order of the function. The Rademacher functions constitute an orthogonal but incomplete set of basis functions. Figure 2.13 illustrates the Rademacher functions for n = 0 through n = 6. The Walsh functions, which form an ordered set of rectangular waveforms which are *both* complete and orthogonal and which assume only the two amplitude values +1 and -1, are defined by

$$WAL(n,x) = \prod_{i=0}^{m} [RAD(i,x)]^{g_i} \tag{2.39}$$

where the order of the Walsh function, n, is expressed as a binary number

$$\left.\begin{aligned} n &= b_m 2^m + b_{m-1} 2^{m-1} + \cdots + b_1 2^1 + b_0 2^0 \\ &= \{b_m, b_{m-1}, \cdots, b_1, b_0\}_2 \end{aligned}\right\} \tag{2.40}$$

and where the g_i represent the Gray code representation of the integer n, i.e.,

$$\left.\begin{aligned} GC(n) &= \{g_m, g_{m-1}, \cdots, g_1, g_0\}_2 \\ where \quad g_i &= b_i \oplus b_{i+1} \quad ; \; i = 0, 1, \cdots m \;\; with \;\; b_{m+1} \equiv 0 \end{aligned}\right\} \tag{2.41}$$

with $\oplus$ denoting modulo-2 addition. This rather circuitous definition can be made clearer with the aid of a concrete example. To find, e.g., WAL(13,x), one would first write

$$\begin{aligned} n &= 13 = \{1,1,0,1\}_2 \\ GC(n) &= \{g_3, g_2, g_1, g_0\} = \{1,0,1,1\}_2 \end{aligned}$$

so that WAL(13,x) = RAD(4,x)•RAD(2,x)•RAD(1,x). Note that WAL^2(n,x)=1 for any order n. Figure 2.14 illustrates the 1-D Walsh functions for orders 0 through 15. The term "sequency" is commonly used in describing these functions and refers to the number of zero

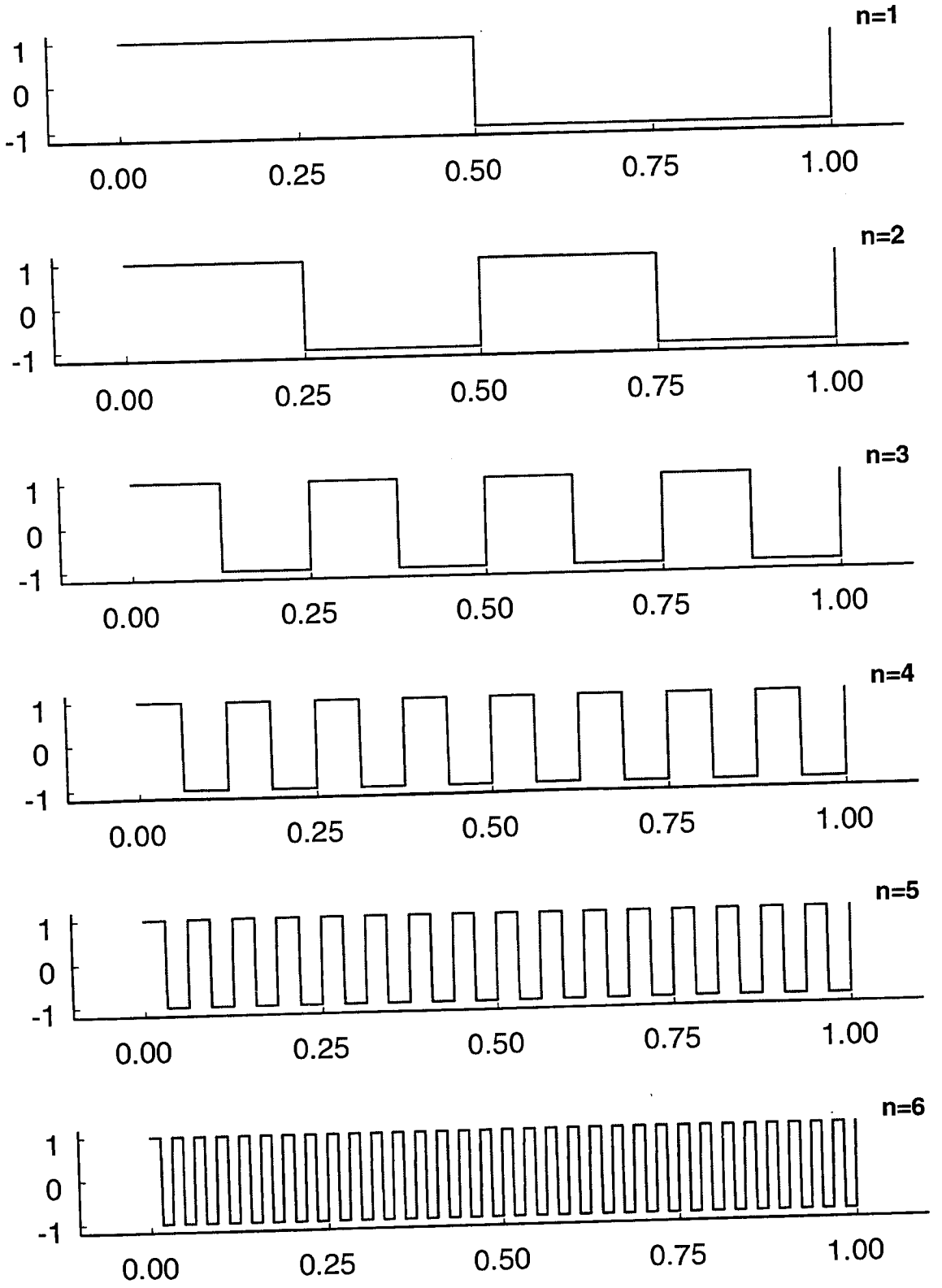


Figure 2.13

Rademacher functions for n=1 to n=6

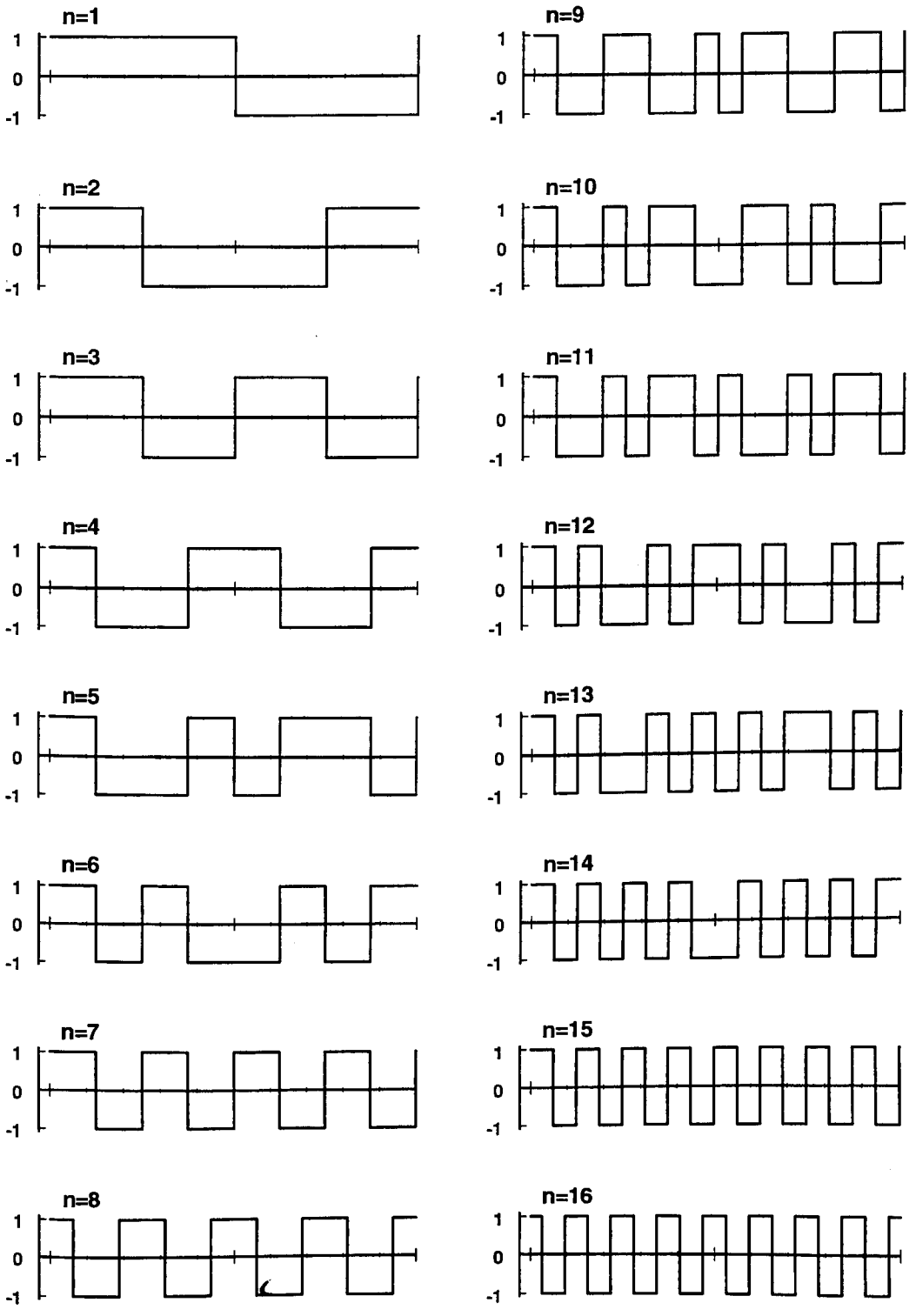


Figure 2.14

The normal Walsh Functions WAL(n,x) for n = 1 to n = 16

crossings (discontinuities) which lie in the interval (0,1). Sequency serves as an analogy to the fixed periodicity or frequency of the trigonometric series $\{\sin(nx), \cos(nx)\}$ with which the Walsh series shares many similarities. The sequence WAL(n,x) as defined by eq. (2.39) is said to be sequency ordered and positively phased (i.e., WAL(n,0^+) = +1 for all n). The reason why the Gray code conversion appears in the definition of the WAL function is precisely to ensure that the series so defined is sequency ordered. A series defined exactly as in eq. (2.39) which used the b_j directly in place of the g_j would lead to the same set of WAL functions but in dyadic order instead of sequency order[8,9].

The series WAL(n,x) is a complete, orthonormal series, i.e.,

$$\int_0^1 WAL(n,x)WAL(m,x)dx = \delta_{nm} \tag{2.42}$$

The functions, the constant WAL(0,x)=1 excepted, also possess a zero measure, i.e.,

$$\int_0^1 WAL(n,x)dx = 0 \quad ; \quad n \geq 1 \tag{2.43}$$

It is a straightforward exercise to extend the Walsh functions to the 2-D plane by forming series of products of the form WAL(n,x)WAL(m,y) defined on the unit square $x \in [0,1]$ and $y \in [0,1]$ and examples of these functions may be found in Beauchamp[8,9] and Harmuth[46,47]. However, it is also possible to define a 2-D Walsh function series on the unit circle $0 \leq r \leq 1$ and $0 \leq \theta \leq 2\pi$ whose members are functions of r and θ and which is complete and orthonormal there. Such a series is defined and illustrated in Harmuth[47] (note, however, that Harmuth uncharacteristically errs in the functional definition stated there although the illustrations of the functions are correct). In the present work, no interest is taken in the ability of the basis functions to serve as a generalized Fourier basis, i.e., one capable of image reconstruction, but only in their ability to serve as a basis which yields rotation invariant moments. Thus, in a fashion akin to that of the SZRP and PZRP bases, the Walsh radial function is defined as

$$WRF_n(r) = WAL(n, r^2) \tag{2.44}$$

The orthogonality and zero measure characteristics are preserved, i.e.,

$$\int_0^1 \int_0^{2\pi} WRF_n(r) WRF_m(r) r dr d\theta = 2\pi\delta_{nm} \tag{2.45}$$

and

$$\int_0^1 \int_0^{2\pi} WRF_n(r) r dr d\theta = 0 \quad ; \quad n \geq 1 \tag{2.46}$$

Figure 2.15 illustrates the profiles of the $WRF_n(r)$ as a function of r for n = 0 through 15. These figures should be contrasted with those for the normal Walsh functions shown in figure 2.14. The effect of the r^2 dependence for the WRF is to shift all of the zero crossings towards the r = 1 boundary by an amount which depends upon r. Figure 2.16 illustrates the $WRF_{10}(r)$ as a 3-D plot.

The WRF moments are defined directly as $M_n^{WRF} = \left|A_n^{WRF}\right|$ where the WRF coefficient is given by

$$\left.\begin{aligned} A_n^{WRF} &= \iint_{x^2+y^2 \leq 1} f(x,y) WRF_n(x,y)\, dxdy \\ &= \int_0^1 \int_0^{2\pi} f(r,\theta) WAL(n, r^2) r dr d\theta \end{aligned}\right\} \tag{2.47}$$

In the present work, the feature vectors formed from the WRF moments run from n = 3 to n = 47 and thus are composed of 45 components.

2.4.2 : Haar Functions, Haar Radial Functions, and Haar Radial Function Moments

The WAL and WRF represent global Walsh functions, i.e., functions which are zero only at a finite number of points in [0,1]. The other major class of Walsh functions includes those which are localized functions, i.e., functions which vanish over one or more intervals in [0,1]. The principal example of a localized Walsh function is the Haar function defined over [0,1] for $n \geq 1$ as

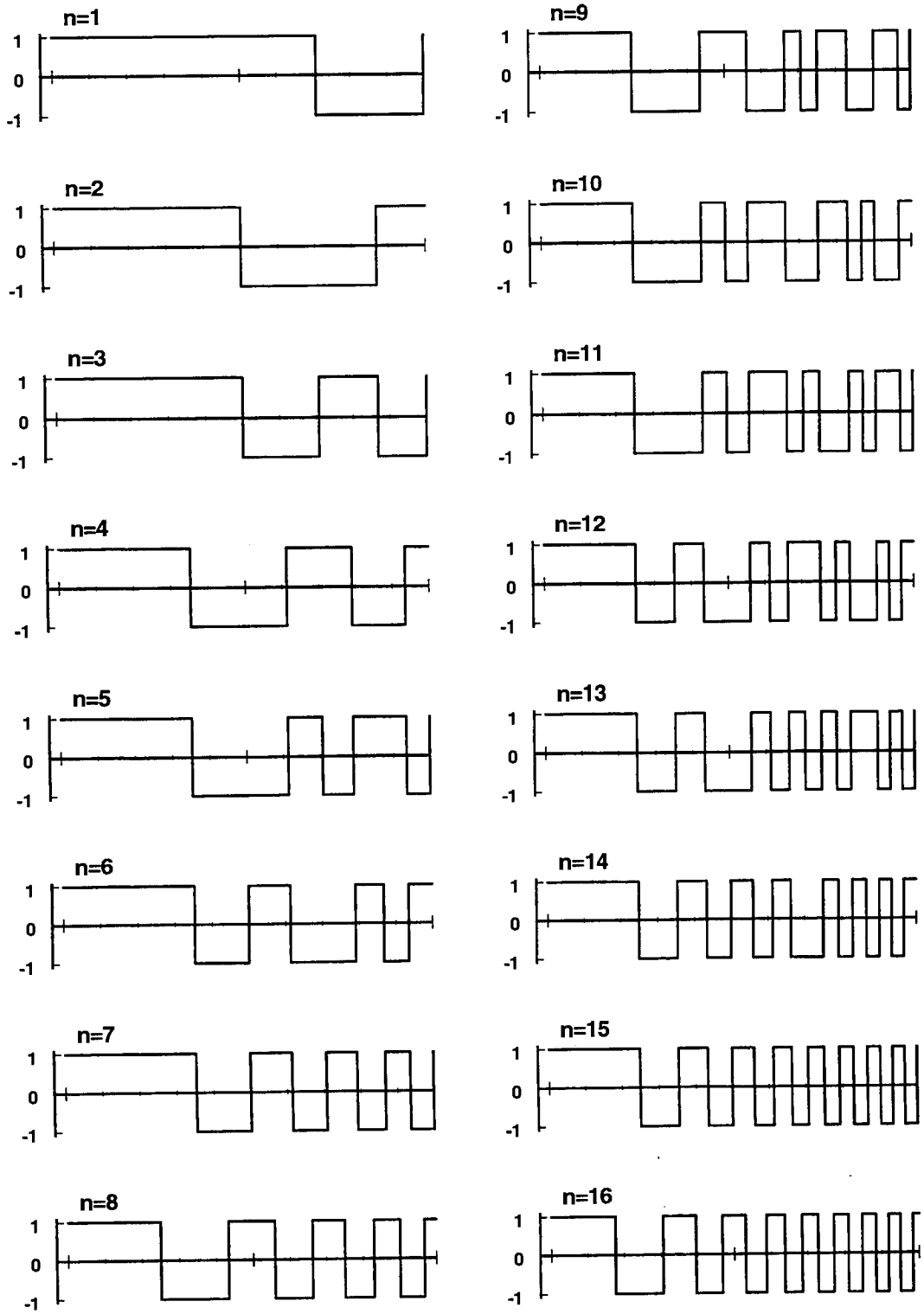


Figure 2.15

Profiles of the Walsh Radial Functions $WRF_n(r)$ for n = 1 to n = 16

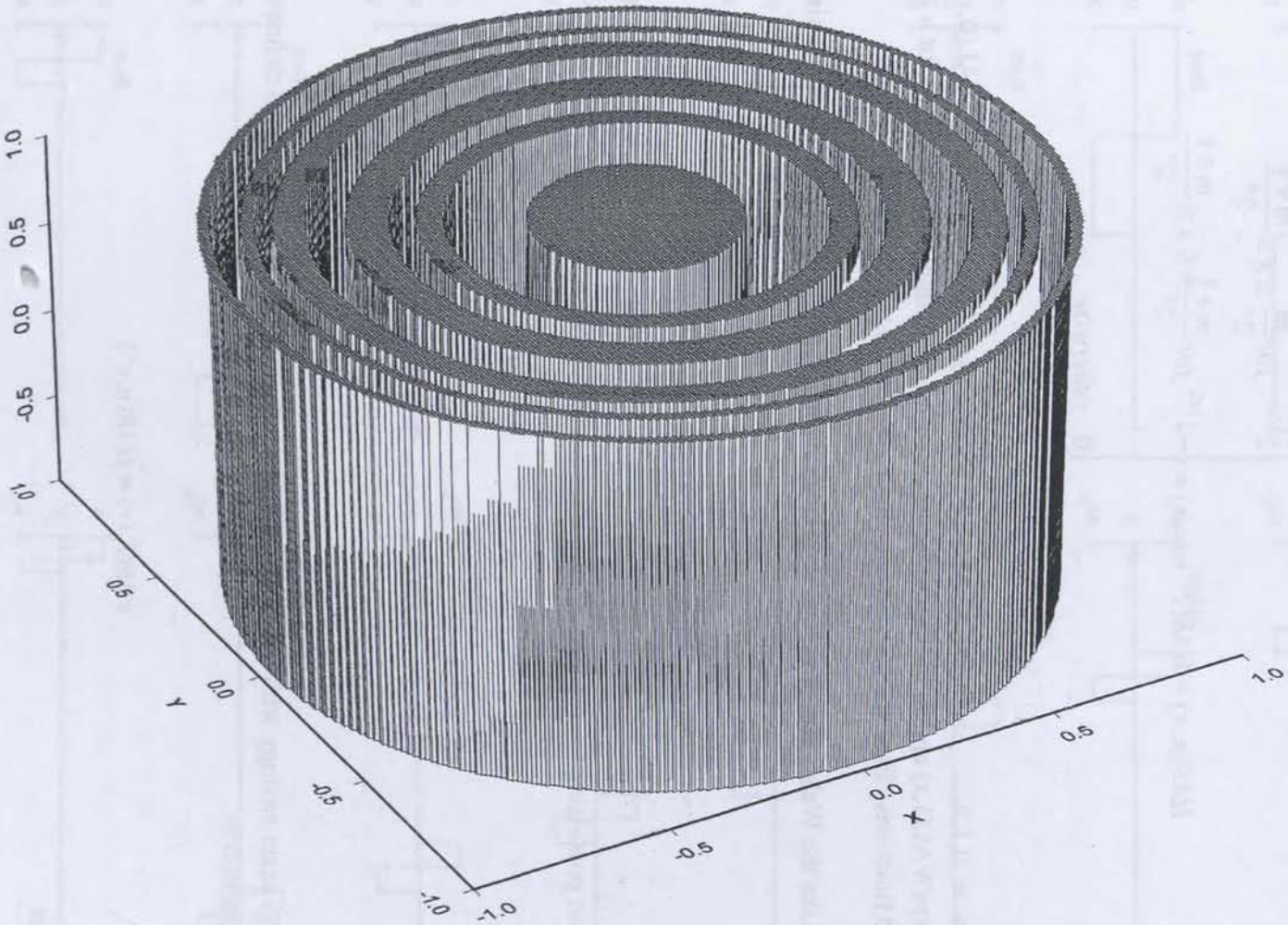


Figure 2.16

Walsh Radial Function $WRF_n(r)$ for n = 10

$$HAR(n,x) = HAR(2^p + m, x) = \begin{cases} 2^{p/2} & for \frac{m}{2^p} \le x \le \frac{m+\frac{1}{2}}{2^p} \\ -2^{p/2} & for \frac{m+\frac{1}{2}}{2^p} \le x \le \frac{m+1}{2^p} \\ 0 & otherwise \end{cases} \tag{2.48}$$

where p = 0,1,2, ... and m = 0,1, ... ,2^p-1. Note that HAR(0,x)=WAL(0,x) and HAR(1,x)=WAL(1,x) and so are global functions. The remaining HAR functions for $n \ge 2$ are localized functions. Figure 2.17 illustrates the HAR functions for n = 0 through 15.

Like the Walsh functions, the Haar functions form a complete, orthonormal basis with

$$\int_0^1 HAR(n,x)HAR(m,x)dx = \delta_{nm} \tag{2.49}$$

while also possessing the property of zero measure for $n \neq 0$, i.e.,

$$\int_0^1 HAR(n,x)dx = 0 \quad ; \quad n \ge 1 \tag{2.50}$$

In exact analogy with the WRF function defined above, it is possible to define a Haar radial function as

$$HRF_n(r) = HAR(n, r^2) \tag{2.51}$$

such that

$$\int_0^1 \int_0^{2\pi} HRF_n(r)HRF_m(r)rdrd\theta = 2\pi\delta_{nm} \tag{2.52}$$

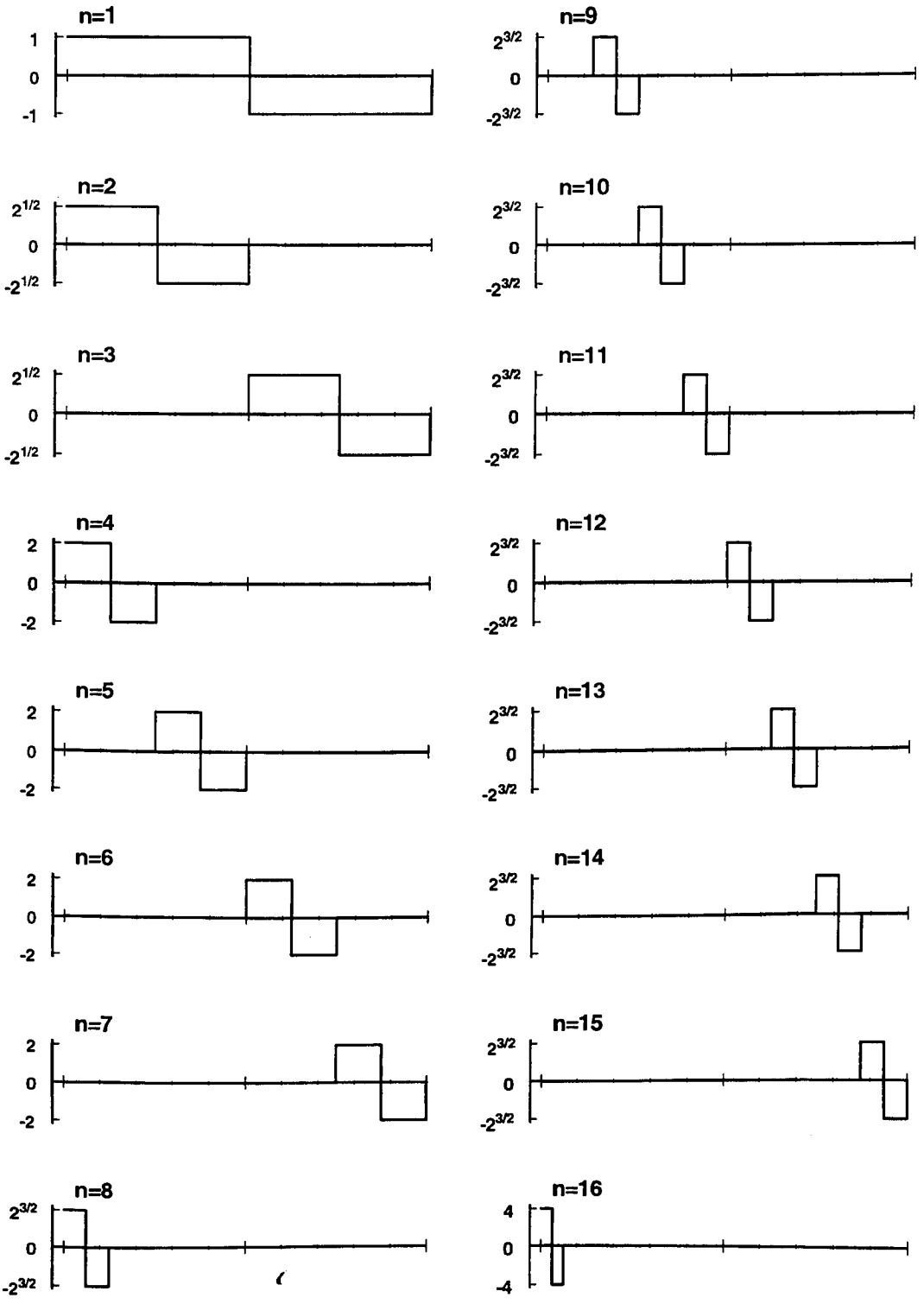


Figure 2.17

The normal Haar Functions HAR(n,x) for n = 1 to n = 16

and

$$\int_0^1 \int_0^{2\pi} HRF_n(r) r dr d\theta = 0 \quad ; \quad n \geq 1 \tag{2.53}$$

Figure 2.18 illustrates the HRF for n = 0 through 15 and should be contrasted to the case of the normal Haar dependence of figure 2.17. For the HRF, the r^2 dependence for the shifts the position of the entire localized function towards the r = 1 boundary by an amount which depends upon r. Figure 2.19 illustrates the $HRF_{10}(r)$ as a 3-D plot.

The HRF moments are defined as $M_n^{HRF} = \left| A_n^{HRF} \right|$ where the HRF coefficients A_n^{HRF} are given by

$$\left.\begin{aligned} A_n^{HRF} &= \iint_{x^2+y^2 \leq 1} f(x,y) HRF_n(x,y)\, dxdy \\ &= \int_0^1 \int_0^{2\pi} f(r,\theta) HRF(n,r)\, r dr d\theta \end{aligned}\right\} \tag{2.54}$$

The HRF-based feature vectors in the present work have dimension 45 and are composed of the HRF components from n = 3 to n = 47.

2.5 : Moment Invariants and Discrete Images

All of the results in the preceding portion of this chapter have been expressed in terms of analog functions defined in the (x,y) or (r,θ) plane. As such, the invariances so calculated to changes in the image scale, position, and orientation in the plane are exact invariances. In practice, however, both the images and the basis functions are represented by their values over a finite grid of pixels (256 x 256 in the present work). Consequently, these exact analog relationships become approximate ones in which integration becomes replaced by summation. In short, the existence of a finite pixel size forces the experimenter to deal with sampled versions of both the image and basis functions.

Throughout the present work, the relationship between the analog and discrete coordinates, illustrated by figure 2.20 (for a 16 x 16 grid), is given by

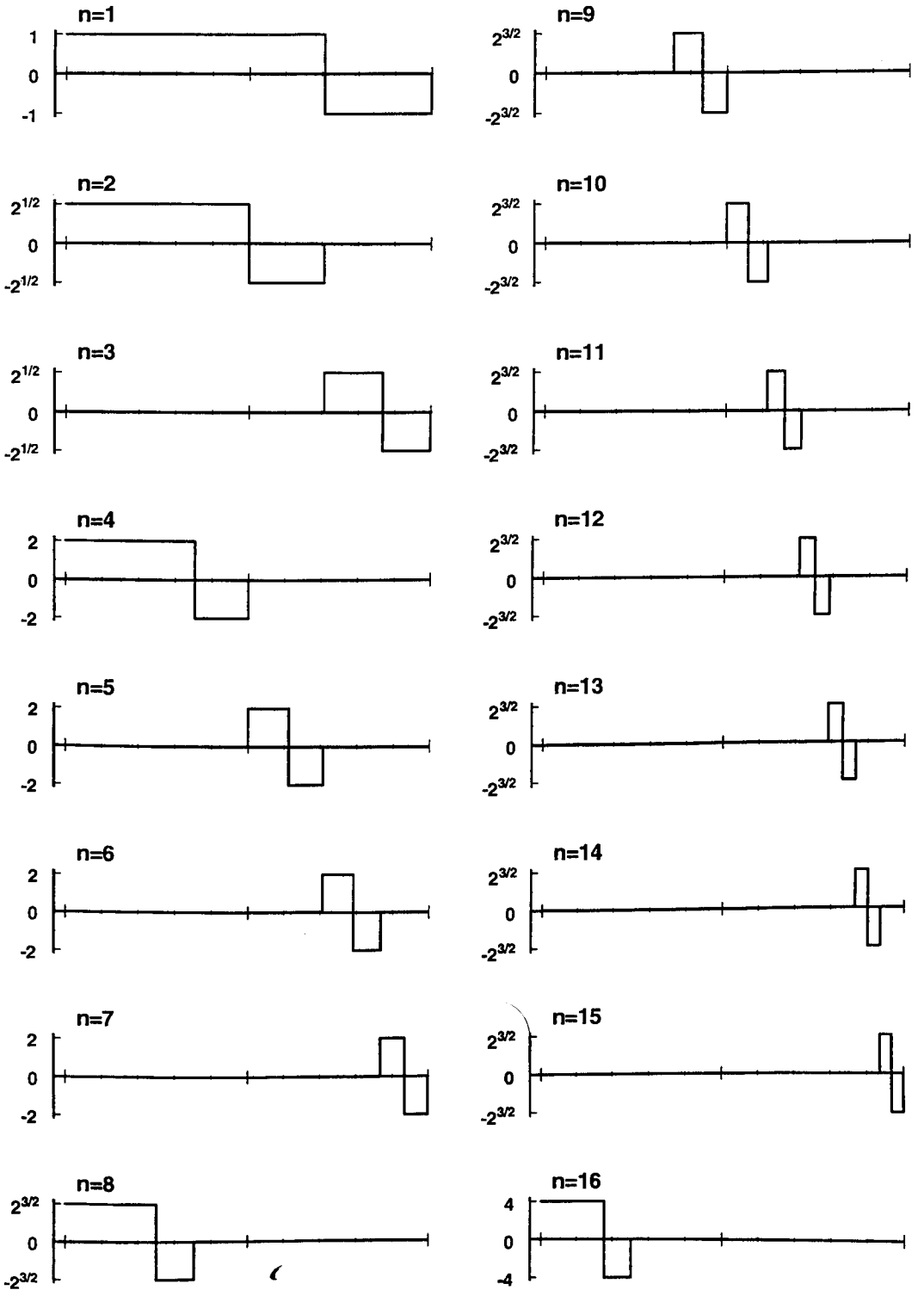


Figure 2.18

Profiles of the Haar Radial Functions $HRF_n(r)$ for n = 1 to n = 16

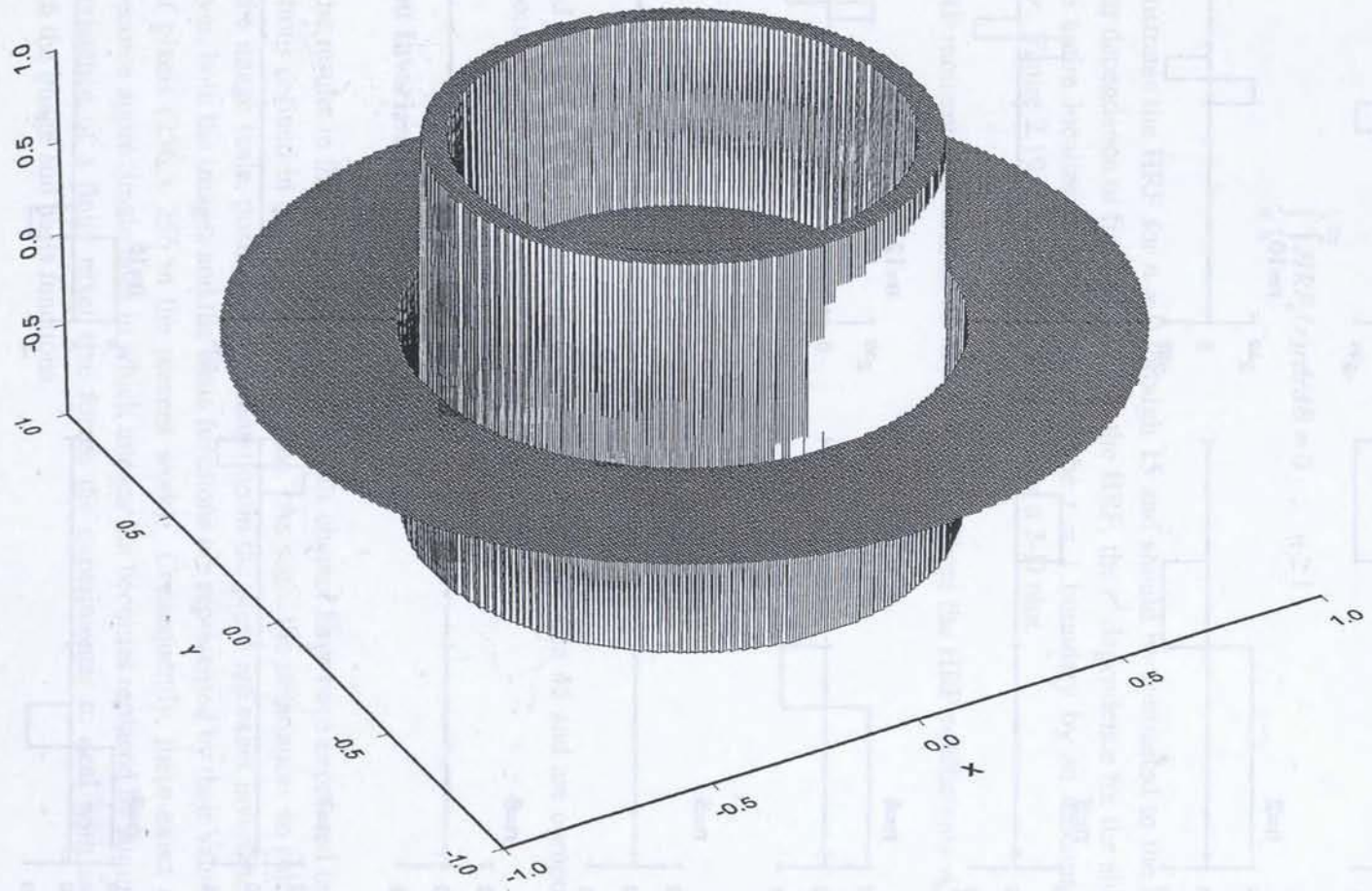


Figure 2.19

Haar Radial Function $HRF_n(r)$ for n = 10

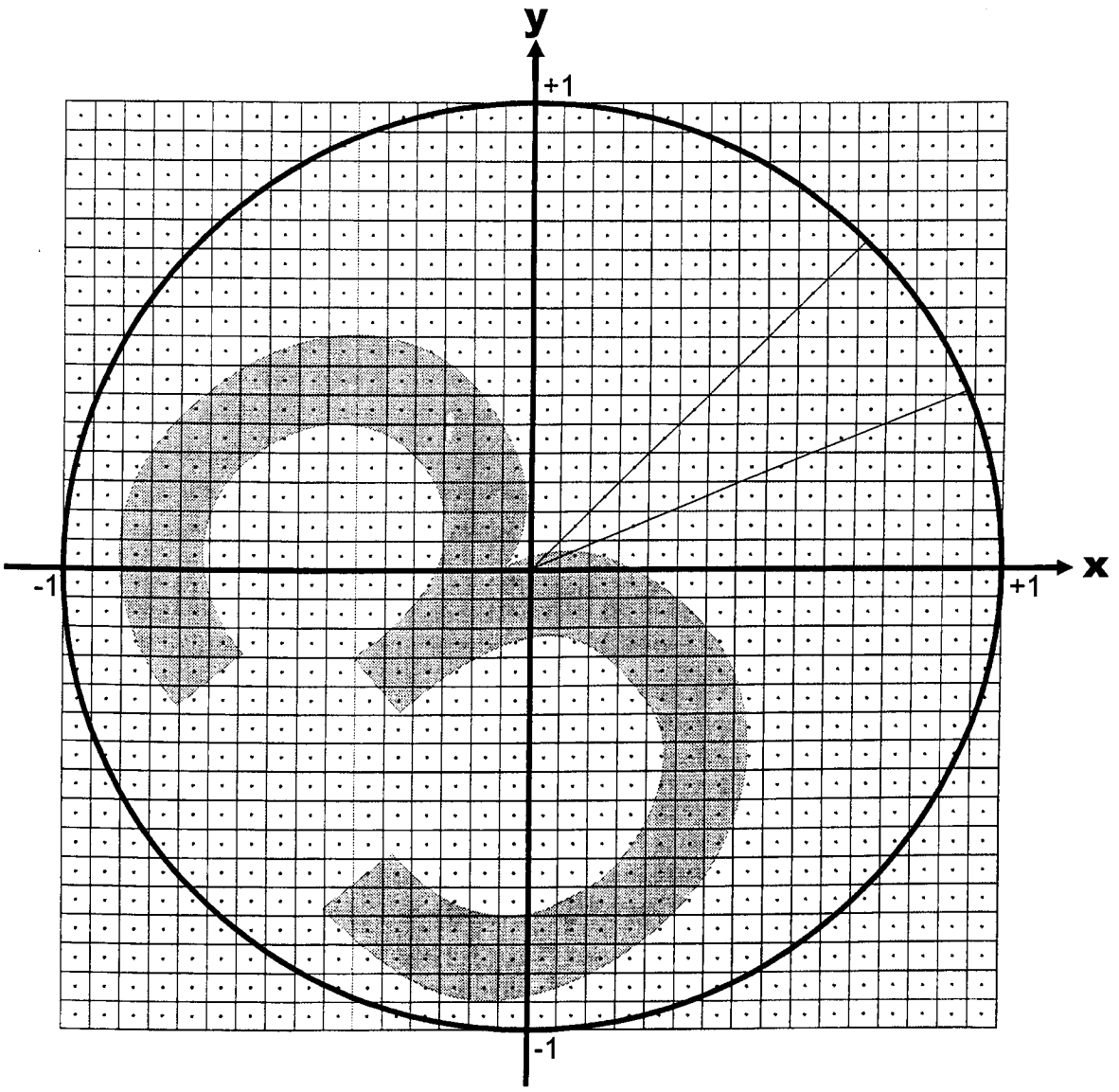


Figure 2.20

The pixel coordinate system used for both images and basis functions.
The image plane is divided into 2N x 2N pixels.
In the example shown here, N = 16 while N = 128 for all of the work
in the present report (256 x 256 grid).

$$\left.\begin{aligned} x_i &= (i-0.5)\Delta \quad ; i = (-N+1)\ to\ N \\ y_i &= (j-0.5)\Delta \quad ; j = (-N+1)\ to\ N \end{aligned}\right\} \tag{2.55}$$

where the grid dimensions are $2N$ x $2N$ pixels and $\Delta \equiv (1/N)$ is the pixel width. This particular assignment of pixel centers places the (x,y) origin at an interpixel point, i.e., there is no pixel corresponding to the values (0,0) and thus *all* of the image pixels are used in the calculations. Each of the integral relationships defined previously in this chapter can now be rewritten as a summation over the (i,j) indices. For reference, the equations defining the various coefficients A_{nm}^{x} or A_{n}^{y} (x denoting SZF, PZF, SZRP, or PZRP and y denoting WRF or HRF) are rewritten here in their discrete forms:

$$A_{nm}^{SZF} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j)\left[\mathrm{Re}(SZF_{nm}(x_i, y_j)) - i\,\mathrm{Im}(SZF_{nm}(x_i, y_j))\right] \tag{2.56}$$

$$A_{nm}^{SZRP} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j) SZRP_{nm}(x_i, y_j) \tag{2.57}$$

$$A_{nm}^{PZF} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j)\left[\mathrm{Re}(PZF_{nm}(x_i, y_j)) - i\,\mathrm{Im}(PZF_{nm}(x_i, y_j))\right] \tag{2.58}$$

$$A_{nm}^{PZRP} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j) PZRP_{nm}(x_i, y_j) \tag{2.59}$$

$$A_{n}^{WRF} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j) WRF_{n}(x_i, y_j) \tag{2.60}$$

$$A_{n}^{HRF} = \Delta^2 \sum_{i=-N+1}^{N} \sum_{j=-N+1}^{N} f(x_i, y_j) HRF_{n}(x_i, y_j) \tag{2.61}$$

In each case, the corresponding moments from which the feature vectors are formed are simply $M_{nm}^{x} = \left|A_{nm}^{x}\right|$ for the Zernike cases and $M_{n}^{y} = \left|A_{n}^{y}\right|$ for the WRF and HRF cases. For small grid dimensions (approximately 32 x 32 or less), the use of straightforward, unweighted double summations such as those above in place of the double integral would not be justifiable; instead, a more accurate numerical approximation to the double integrations such as the use of Simpson's rule would be required. In the present work with a 256 x 256 grid, the double summations are sufficiently accurate to forego the need for any type of weighted summation[115].

The finiteness of the pixel grid affects the accuracy with which the feature vectors can be calculated in two different ways. Firstly, the processes of scaling and shifting the given image distribution, as dictated in eq. (2.6), will inevitably introduce some distortion into the final image as the result of the scaling and some error in the image positioning owing to the discrete nature of the shifting process. Distortion introduced by image scaling will be most severe for large scaling factors (either up or down) and/or for relatively small grid sizes (again, small denoting approximately 32 x 32 or less). In the present work, the distortion introduced through the image rescaling is expected to be relatively minor given the 256 x 256 image plane dimensions and the fact that upper and lower limits were imposed upon the process of introducing the random scale factor into the image preparation. The shifting of the (scaled) bitmap also introduces error since, given the discrete nature of the image distribution, the minimum shift which is possible in either the x- or y-direction is one pixel. In effect, the image can be positioned to its center of optical mass to within ± 0.5 pixel width in the general case. This alignment error is expected to have lesser consequences when the image is centered against analog basis functions such as the Zernike functions while producing greater effects and errors for the discrete, stepwise continuous Walsh and Haar functions. Explicit evidence of this will be shown in chapter 4 where the variance of the feature vectors for the different basis functions is illustrated graphically.

The second way in which the discrete pixel grid affects the accuracy of the calculation of the feature vectors is that the basis functions are represented by their sampled versions on the $\{i,j\}$ grid. In principle and in practice, the computation of the basis function values at these discrete points is exact. However, the double summation, as in eqs. (2.56) to (2.61) above, uses the value of the basis function at $\{i,j\}$ as if it were constant over the pixel centered at $\{i,j\}$. For larger grid sizes, this approximation is generally excellent and the double summations may be evaluated directly as explicitly written. For smaller grid sizes (approximately 32 x 32 and smaller), it would be necessary to incorporate a more accurate means such as the trapezoidal or Simpson's rule to carry out the numerical integration. The reader is specifically referred to the papers by Teh and Chin[115,116] for a detailed description of the effect of finite pixel size on the accuracy of integration.

There is one additional manner in which the finiteness of the pixel grid affects the overall classification system and that is the fact that, because the basis functions are represented by their sampled versions, there will exist limits on the order of the basis functions which may be accurately and correctly represented by this sampling. By considering the minimum width of a variation in the SZRP or PZRP polynomial, it is possible to arrive at a "rule of thumb" estimate for the allowable maximum principal index n as a function of N (where the grid size is $2N$ x $2N$ pixels) of the form

$$\left.\begin{aligned} n_{\max}^{SZ} &\approx \frac{4N-15}{10} \quad ; \textit{for SZF or SZRP} \\ n_{\max}^{PZ} &\approx \frac{2N-5}{5} \quad ; \textit{for PZF or PZRP} \end{aligned}\right\} \tag{2.62}$$

These, in turn, leads to rules of thumb for the maximum feature vector dimension for the Zernike bases as

$$\textit{Maximum Dimension of Feature Vector} \approx \begin{cases} 0.04 \cdot N^2 & ; \textit{for SZF or SZRP} \\ 0.08 \cdot N^2 & ; \textit{for PZF or PZRP} \end{cases} \tag{2.63}$$

By way of illustration, for $N = 128$ (the 256 x 256 grid employed in the present work), these rules of thumb yield $n_{\max}^{SZ} \approx 50$ and $n_{\max}^{PZ} \approx 50$ with maximum feature vector dimensions of approximately 655 and 1310, respectively. Note that these are "generous" rules of thumb which ignore the inevitable errors incurred in the Cartesian representation of functions which are naturally defined in terms of polar coordinates (r,θ) (such considerations would reduce the estimates for $n_{\max}^{SZ}$ and $n_{\max}^{PZ}$ by a factor of $\sqrt{2}$ and, correspondingly, the estimates for the maximum feature vector dimension by a factor of 2). The essential point here is that, owing to the analog nature of the Zernike bases, the functions can be accurately represented on a finite pixel grid over a broad range of indices $\{n,m\}$ and thus permit relatively high feature vector dimensions.

The same considerations for the cases of the WRF and HRF functions, however, yield much more restrictive findings. Allowing explicitly for the r^2 dependence of these basis functions, a rule of thumb for the maximum order of either the WRF or HRF which can be accurately represented on a $2N$ x $2N$ grid takes the form

$$n_{\max}^{WRF,HRF} \approx \frac{N}{2} - 1 \tag{2.64}$$

For $N = 128$ (the 256 x 256 grid employed in this work), eq. (2.64) yields the value 63 which would limit the feature vector dimension for either the WRF or HRF to approximately 60 after excluding the $n = 0$, 1, and 2 terms. Thus the actual values employed in the present work (feature vector dimension of 45) were close to this upper limit. Indeed, the rule of thumb, eq. (2.64), is a generous one; consideration of the polar to Cartesian conversion process would introduce a further reduction into eq. (2.64) of $\sqrt{2}$ (in fact, the working upper limit would lie somewhere between these two estimates). The essential point to be taken from these numbers is that the restrictions on the order of the basis functions and hence the restrictions on the maximum allowable feature vector dimensions are much greater for the rectangular waveform Walsh and Haar bases than for the analog Zernike functions.

Chapter 3 : Experimental Design and Execution

3.1 : Overview

The primary objective of the present work is to measure the performance of a multi-layer, backpropagation-trained, neural network classifier against, (a), six different sets of basis functions used to derive the moment invariant feature vectors for the images and, (b), two different sets of training vectors, the first consisting solely of noise-free images and the second consisting of a mix of noiseless and noisy images. The results of these twelve experiments, which will be presented in chapter 4, serve to identify that set of basis functions which yields the most accurate and robust classifier performance and will also serve to quantitatively demonstrate the degree of improvement in the classifier performance which results from the incorporation of noisy data into the training process.

The overall feature vector extraction and neural network classification system is governed by several operating parameters and variables, many of which are tacit parameters for the process, e.g., the form of the activation function for the artificial neurons and the method employed for the normalization of the feature vectors. Accordingly, prior to determining the classification performance for the twelve combinations stated in the prior paragraph, a series of experiments was carried out to measure the dependence of the classification system on, (1), the number of hidden neurons for the multilayer perceptron network, (2), the epoch size used in the training stage and, (3), the method for the normalization of the feature vectors. In addition, measurements were made of the variation in the classifier performance resulting from the random nature of the network initialization and presentation of training data. The results of these experiments will also be presented in the following chapter.

In all of the measurements aimed at determining the performance of the neural network classifier, the image classification process itself from raw test image to final result consists of eight steps:

1. Creation of the scaled, translated, and rotated high resolution image.
2. Reduction of the image resolution followed by transformation (scaling and shifting) of the image to yield irradiance normalized, central moments ($\mu_{00} = \beta$ and $\mu_{10} = \mu_{01} = 0$).
3. Addition of noise and conversion of the image to a standard binary array format.
4. Calculation of the raw (unnormalized) feature vector.
5. Normalization of the feature vector (where the normalization parameters are determined by the training set).

6. Generation of the complete feature vector by concatenation of the normalized feature vector and the result vector.
7. Classification of the normalized feature vector by the neural network (where the network has been previously trained).
8. Rating of the classification result.

Figure 3.1 illustrates this sequence in a flowchart format while indicating the differences between the training and test processes. The first and last steps in this sequence were executed with the aid of commercial software packages - Corel Draw (version 5.0, run on a 80486 33MHz PC) to generate the basic images and Neuralworks Professional II+ (version 5.1, run on a Sparc 2 workstation) to simulate the artificial neural network. The intermediate steps were executed using custom programs written in Fortran 77. The core programs used to carry out the intermediate steps in the sequence above are listed in Appendix A while the various programs used to calculate and store the different basis functions employed in the derivation of the feature vectors are listed in Appendix B. The remainder of this chapter will describe in more detail the individual steps in the sequence listed immediately above. The reader may refer to the program listings in the appendices for any points of detail not covered in the following sections.

3.2 : Generation of the Training, Validation, and Test Image Sets

All of the images used in this work were derived from a fundamental set of nine images created by Corel Draw which consisted of the numerals '0' through '8' inclusive in a sans serif font style (Corel "Switzerland" equivalent to "Helvetica"). Since, for this type of font, the numeral '9' is essentially identical to a rotated '6', the numeral '9' was not included in the basis set. This basis set of images, shown in figure 3.2, consisted of the numerals unscaled, untranslated, and unrotated. To generate the images actually used to form the various data sets (i.e., training, validation, and test image sets), a series of four random numbers (each between 0 and 1) was randomly chosen from a table of random numbers. The first number was multiplied by 2 and was used to scale the image with the restriction that the scaling factor lie between 0.4 and 1.4 in order to avoid excessive distortions of the image owing to bitmap scaling. The second and third random numbers were transformed to values in [-1,1] and were used to translate the image along the x and y axes while the fourth number, when multiplied by 2π, served to introduce a random rotation of the image. The only condition applied to this overall process was that the final scaled, translated, and rotated image must lie completely within the [-1,1] x [-1,1] square in which the image was originally framed, i.e., occulted or partial images were not considered in this work. The final step in this process was to export the scaled, translated, and rotated image as an uncompressed 'TIFF' (Tagged Image File Format) bitmap file consisting of 1024 x 1024 binary-valued pixels.

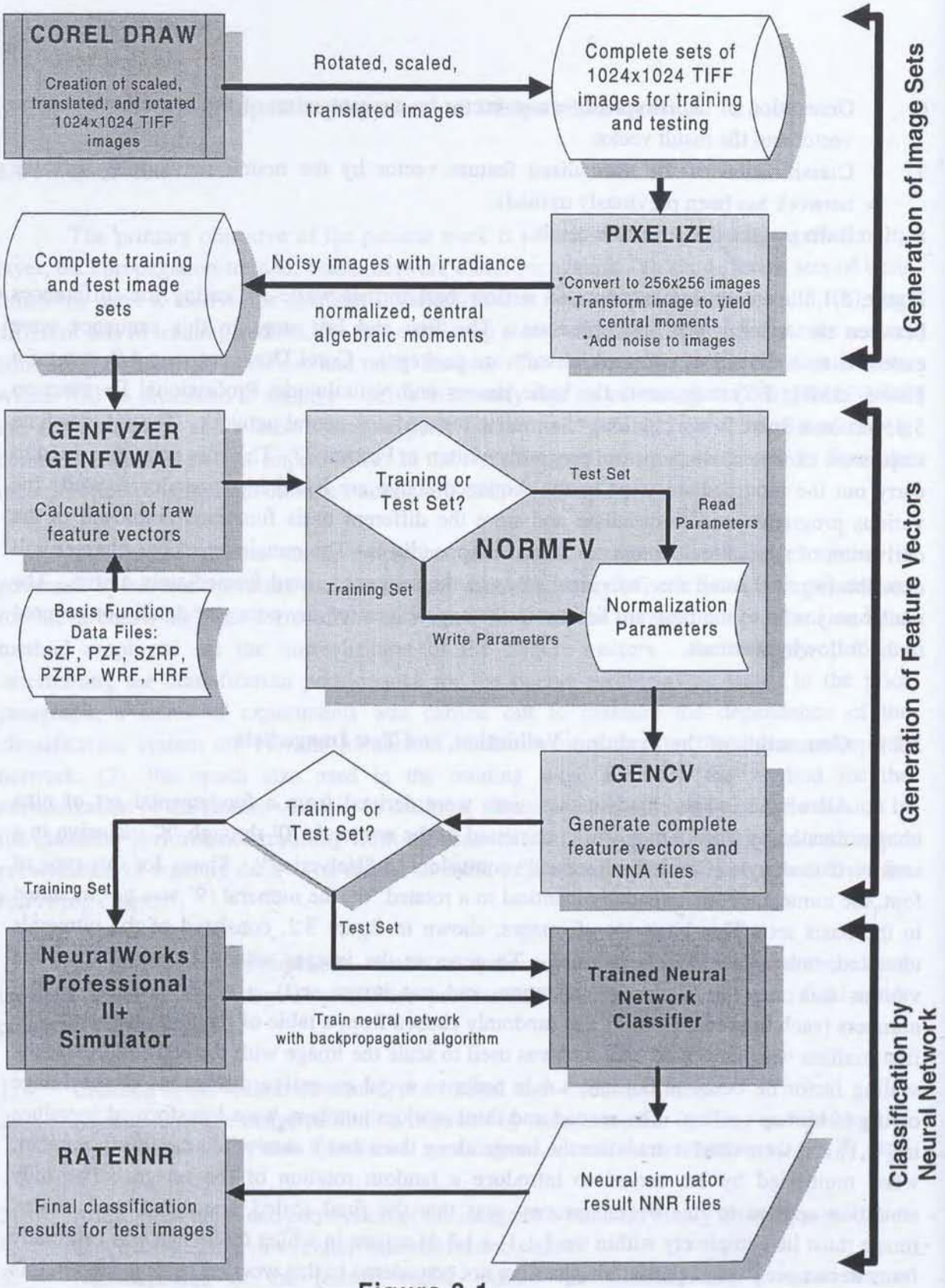


Figure 3.1

Flowchart representation of experimental design showing sequence of image and feature vector processing steps.

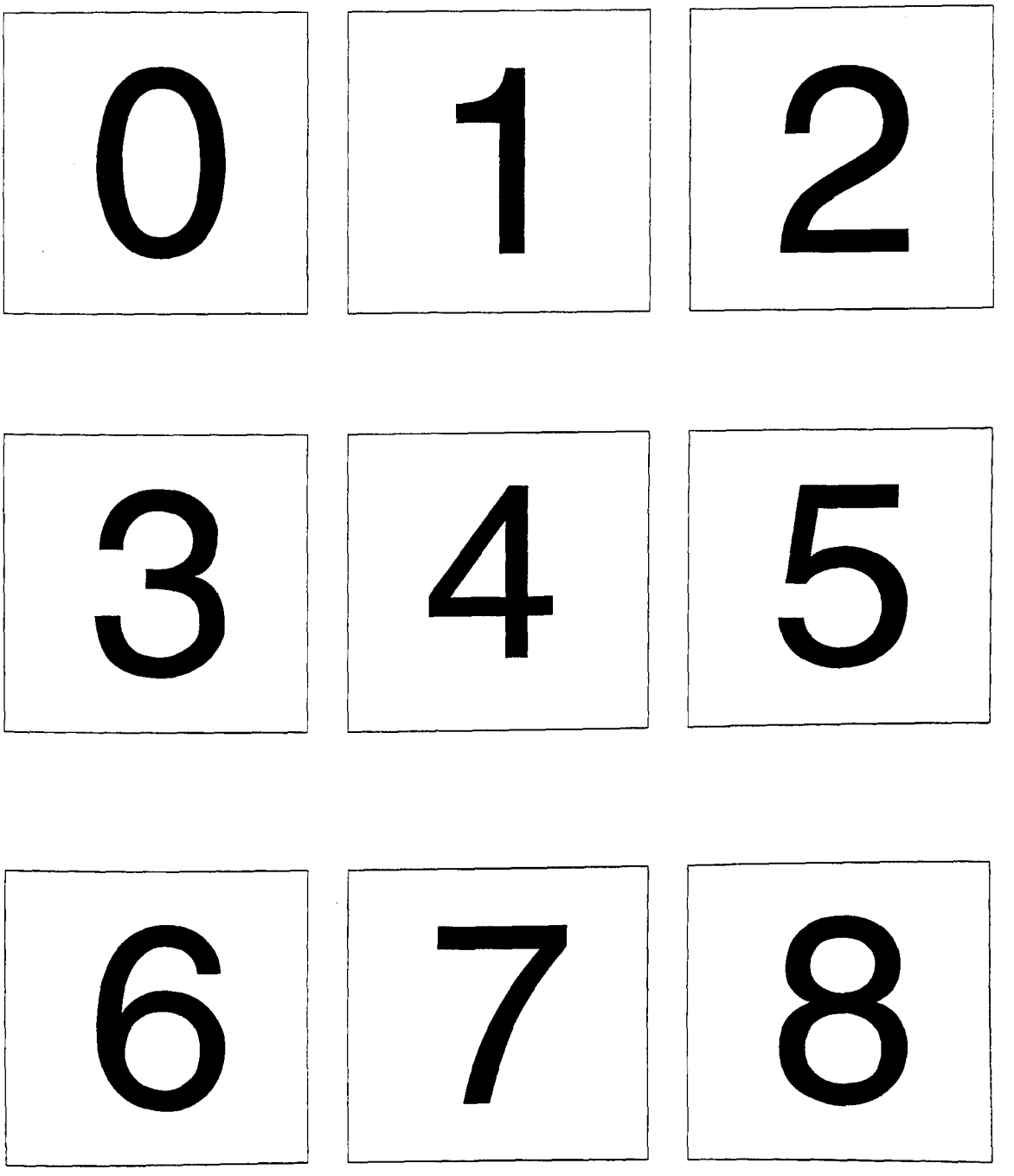


Figure 3.2

The basic set of nine images as Corel Draw graphics. All of the images in the training, validation, and test sets are derived by scaling, translating, rotating, and adding noise to these basic images.

Two training sets were generated for use in all of the classification measurements in this work:

- **Training Set A** consisted of 45 noiseless images, 5 each of the 9 numerals, with each image having randomly chosen scale, lateral position, and in-plane angle of rotation (subject only to the restrictions described above). The decision to use 5 images for each numeral was arrived at by examining the goodness of the invariance of the feature vectors for difference basis functions and by trial and error. It was found that increasing the number of examples per numeral beyond five did not result in either more rapid or more accurate training of the network.

- **Training Set B** was composed of two sets of 45 images, the first a noiseless set with 5 images per numeral and the second an independently derived set (with 5 images per numeral) to which was added noise to a SNR of 8 db. The process for the addition of noise to images and the definition of SNR for these images is described in detail in the following section.

In order to ensure that the training of the neural network is approximately optimum and, more to the point, is neither under- nor over-trained, an independent set of 630 images was generated to serve as a validation set. The 630 images consisted of 14 independently derived sets of 45 images each, 5 images per numeral in each set, with SNR's for the 14 sets which ranged from noiseless down to 2 db. To monitor the network training process, the neural network was trained on a given training set for a specified number of passes (one training image per pass) and training epoch (the number of passes between network weight updates). The overall classification accuracy of the trained network was then measured for the entire 630-member validation set. The network was repeatedly trained and tested in this way while varying the number of training passes. It is an essential requirement[49] in the application of artificial neural networks as pattern recognition circuits that the data sets used to train, validate, and test the network be independent of one another. In the present case, where the images are artificially generated, it is essential that each of these image sets be independently generated from the basis set described above. It would be highly facetious and inherently misleading, for example, to assess the performance of the trained network if the test image data contained or was a subset of the training data, even if the test images were modified somewhat by, e.g., the incorporation of additional noise. With 'real world' data, the underlying statistics of a given image set can vary from training to test sets or for different test sets and this variation can affect, often substantially, the performance of the neural network classifier. In the present case, it is expected that the statistical nature of the various image sets would vary little if at all given the methodology of generating the images artificially via computer software. In addition, the method of generating random noise to add to the images

was the same for all image sets. Nevertheless, these idealizations notwithstanding, it remained imperative that the training, validation, and test sets be generated in an independent manner. Under training was not a problem in the present work; in virtually all cases, the neural network converged very quickly to a trained state after only several hundred iterations through the training set. Further training then served to reduce the RMS error of the neural network. Over training the network also was not a problem in these measurements. It was found that training numbers approximately 10 times those ultimately used to train the networks were necessary to reveal any measurable degree of network overtraining and, thus, this regime was easily avoided. Undoubtedly, this insensitivity to the number of training iterations is a direct result of the fact that, even though the training, validation, and test sets were independently derived, the underlying statistics of these images sets were essentially identical given the fact that all of the images were artificially created from computer generated images and subsequently processed by similar routines.

The test set used in all cases to measure the performance of the classification system was composed of 23 subsets of images. Each subset consisted of 45 images, 5 each of the 9 image classes, with each image independently assigned random values for its scale, position, and orientation. The SNR for each subset was different, ranging from noiseless (i.e., SNR = ∞) through to a subset which represented pure noise (SNR = 0 db). The actual values for the 23 subsets are listed in table 3.1. To simplify the pragmatics of measuring the classifier's performance as the SNR of the test data decreased, these 23 test subsets were concatenated into one large test set consisting of 1035 images from which was created the ASCII files in the formats required for the neural simulation program (these were the "nna" files referred to in the data preparation routines given in Appendix A). Throughout this report, reference to 'measuring the performance of the neural network classification system' is to be interpreted as measuring the classification accuracy over this single, large test set containing image data of decreasing SNR. The details of the addition of image noise and the exact interpretation of SNR values are described in the next section.

3.3 : Conversion to Irradiance Normalized, Central Moments and the Addition of Noise

The scaled, translated, and rotated images created by the commercial graphics software Corel Draw were exported as (uncompressed) TIFF (version 5) bitmap files containing 1-bit pixels over a 1024 x 1024 grid. A Fortran program entitled "pixelize" and listed in Appendix A was written to carry out several basic image processing tasks, namely:

1. to read the uncompressed TIFF bitmap file;
2. to convert the 1024 x 1024 image to an averaged 256 x 256 pixel bitmap;

Test Subset (45 images per subset)	SNR (db)	Number of Iterations of Random Noise Generator	Exact SNR measured (db)
Test01	∞	0	∞
Test02	50	207	49.9827
Test03	40	653	39.9983
Test04	30	2,071	29.9996
Test05	25	3,701	25.0004
Test06	20	6,606	19.9997
Test07	15	11,717	14.9996
Test08	12	16,879	12.0000
Test09	10	21,665	10.0002
Test10	9	24,543	9.0002
Test11	8	28,015	8.0001
Test12	7	31,947	7.0001
Test13	6	36,446	5.9999
Test14	5	41,851	4.9998
Test15	4.5	45,053	4.4998
Test16	4	49,031	4.0003
Test17	3.5	53,544	3.5000
Test18	3	58,460	3.0004
Test19	2.5	64,970	2.5003
Test20	2	72,202	2.0000
Test21	1.5	79,910	1.4998
Test22	1	95,230	0.9999
Test23	0	282,800	0.0000

Table 3.1

The SNR parameters for the 23 test subsets used to measure the neural network classifier performance.

3. to perform the image scaling and translation;
4. to optionally add random noise to the image to a prescribed SNR; and
5. to save the 256 x 256, optically centered and scaled image, with added noise, as a standard binary "pix" file containing only the final pixel values (i.e., without headers or any other form of graphics-format-dependent structure).

The process of image scaling and translation carried out by the routine "pixelize" corresponded exactly to the scaling and shifting of the image described by eq. (2.6) so that any moments subsequently calculated were irradiance normalized, central moments. By employing a simple naming scheme for the TIFF and PIX image files (the adoption of the "pix" DOS extension for the 256 x 256 binary files being purely a notational choice), the "pixelize" program could process large numbers of images automatically over a single run by incorporating a simple "do-loop" involving certain of the filename characters.

The conversion of the high resolution 1024 x 1024 bitmapped image to the lower resolution of 256 x 256 was achieved by a simple, running average over 4 x 4 groups of pixels in the higher resolution image (with 2 or more black pixels in such a group resulting in a black pixel). Figure 3.3 shows several examples of different pixel resolutions for a specific noiseless image ranging from 1024 x 1024 down to 8 x 8. The bitmaps shown in figure 3.3 are obtained by a direct resampling of the 1024 x 1024 image using a commercial, image processing tool (Corel Photo Paint V. 7); note that some of the distortion evident in the lowest resolution bitmaps (16 x 16 and 8 x 8) originates with the technique employed by the software for image resampling or scaling. Any degree of distortion resulting from the undersampling of the bitmap image in the 1024 x 1024 to 256 x 256 down-conversion is clearly negligible. The decision to work with 256 x 256 images was arrived at by compromise. On one hand, this resolution is fine enough to avoid potential problems and errors which arise for images defined on low resolution grids (approximately 32 x 32 or less) such as those[115,116] involved in ultimately calculating the double integrals defined in Chapter 2 (see the discussion in section 2.5). On the other hand, some preliminary measurements with resolutions higher than 256 x 256 indicated that no measurable improvement in the classification performance was achieved. This meant that practical problems of greatly increased file sizes and computation times attendant to much higher resolution images could be avoided (a 1024 x 1024 image file, e.g., would be 16 times larger that the 256 x 256 image file and would, in addition, require equally larger files for the off-line storage of the basis functions).

Working with the 256 x 256 binary image, the lowest order, central algebraic moments, μ_{00}, μ_{10}, and μ_{01} were calculated. The image was then shifted in both the x and y directions to produce an image with minimal values for μ_{10} and μ_{01}. This shifted image was then scaled (up or down, as required) to yield an image with a predetermined value of μ_{00} of approximately β. Throughout this work, the images were scaled to give a value of $\beta = 8192$

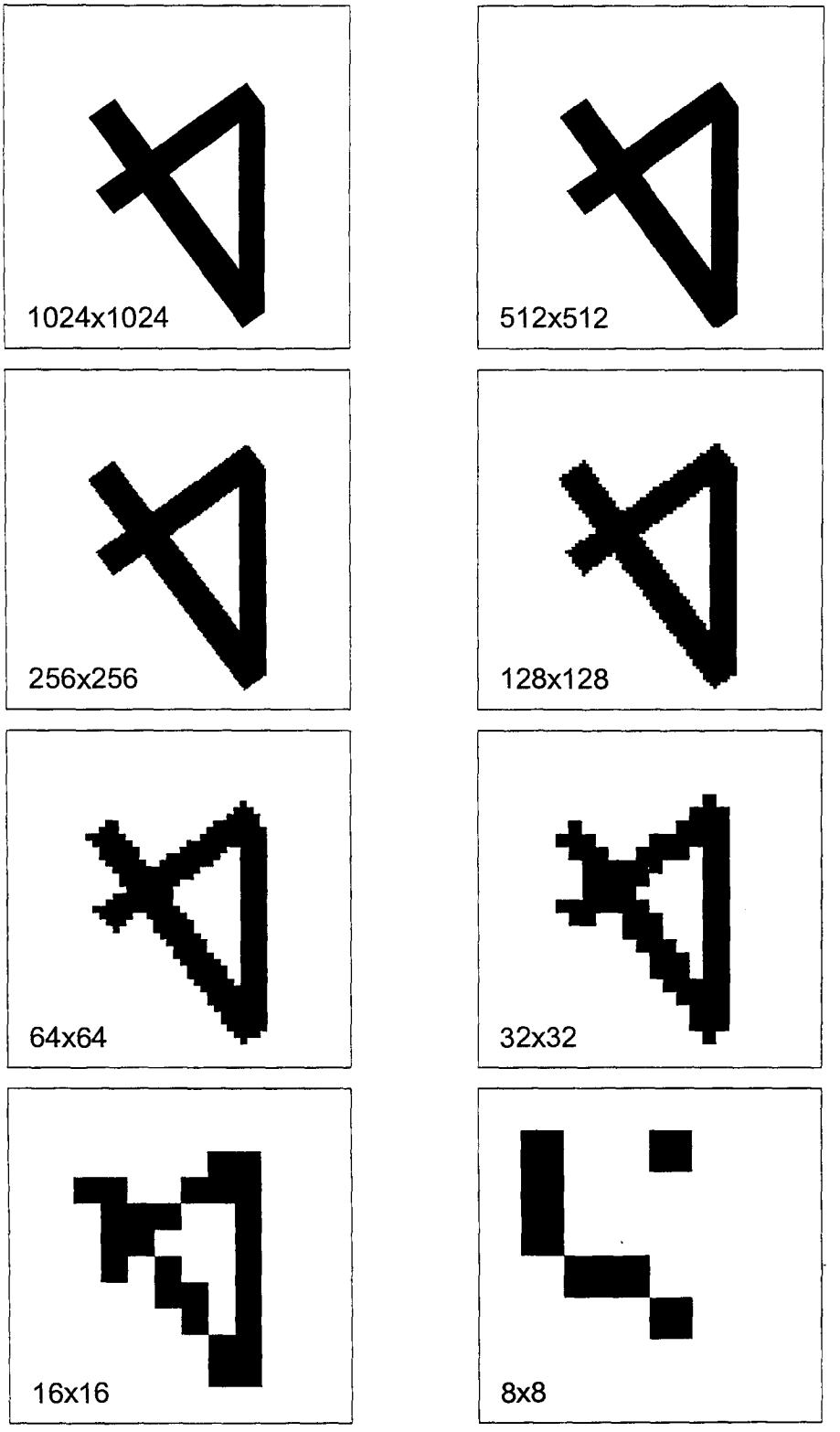


Figure 3.3

Effect of image grid dimensions on accuracy of image representation

(which is 1/8 of 256 x 256). For binary-valued, noiseless images, $\mu_{00} = \beta$ is simply the number of black pixels. Following the image scaling, the values of μ_{10} and μ_{01} were recalculated and the image was iteratively shifted to yield final μ_{10} and μ_{01} values which were minimal. As discussed in section 2.5, the discrete nature of the image representation on a 256 x 256 pixel grid meant that the final values of the normalized, central algebraic moments μ_{00}, μ_{10}, and μ_{01} were generally never exactly equal to 8192, 0, and 0, respectively. Figure 3.4 shows a sample entry from a file "pixelize.log" which was recorded by the program "pixelize" during a typical run. This file recorded several key variables and parameters used during this image preparation stage, including the final values calculated for μ_{00}, μ_{10}, and μ_{01}. Note that, although the first order moments μ_{10} and μ_{01} are not zero, the important observation regarding these values is that their ratio to $\mu_{00} = \beta$ is $<< 1$.

The user was offered the option of adding noise to this scaled, shifted image to a pre-calibrated SNR. Note that the noise was added *after* the image had been first scaled and shifted, i.e., the problems of determining the optical center and the scaling of a noisy image were not dealt with in the present work. Indeed, this problem may be a significant one in future work with real world (i.e., noisy) images where it is inherently impossible to distinguish between noise and original image for a given pixel. Accurate scaling and centering of a noisy image is possible provided the experimenter has sufficient knowledge concerning the statistical nature and level of the noise present in the image since the effect of the noise on the shifting and scaling of the image can then be predicted. In the present work, random noise was added to the image by using a known random number generator to arrive at a pair of numbers in the range of [0,1]. This pair was then trivially transformed to correspond to an $[x,y]$ coordinate in the image grid covering [-1,1] x [-1,1] for the x and y axes. The pixel value at this coordinate was then incremented by adding +1 modulo 2 (i.e., 0+0=0, 0+1=1+0=1, while 1+1=0). In other words, the existing pixel at this location would be changed to black from white or to white from black. By separately calibrating this random number routine, the image could be modified to possess any desired SNR by iteratively running the random number routine a specified number of times. Note that it is essential to account for the degeneracy of this additive noise process in that pixels which are modified an even number of time are, in effect, unchanged in value. Only those pixels which are 'hit' an odd number of times are changed. Thus, as the calibration process revealed, a rapidly increasing number of iterations was needed to produce lower and lower SNR's owing to the fact that more and more pixels were being 'hit' an even number of times when the number of iterations increased. It is also important to note that, for these computer generated images with computer generated additive noise, the SNR is an exact, *measured* parameter given by the equation

```
******************** Conversion of TIFF Data to Pixel Data ********************
                        Version 4.20 : July 22, 1994
      File : 001022.tif     Gridsize :  256 x  256   Precision : Real*4

   Tag         Tag Name              Datatype    Length    Data  Offset
   ===  ==========================  ========    ======    ====  ======

   254  New Subfile Type              Long         1         0
   256  Image Width (Pixels)          Long         1      1024
   257  Image Height (Pixels)         Long         1      1024
   258  Bits per Sample               Short        1         1
   259  Compression                   Short        1         1
   262  Photometric Interpretation    Short        1         1
   273  Strip Offsets                 Long        17               182
        Strip Offsets                 Long        17       334
   277  Samples per Pixel             Short        1         1
   278  Rows per Strip                Long         1        61
   279  Strip Byte Counts             Long        17               250
        Strip Byte Counts             Long        17      7869
   282  X Resolution (dpi)           Rational      1               318
        X Resolution (dpi)           Rational      1    117.72
   283  Y Resolution (dpi)           Rational      1               326
        Y Resolution (dpi)           Rational      1    117.72
   284  Planar Configuration          Short        1         1
   296  Resolution Units              Short        1         3

******************** Conversion of TIFF Data to Pixel Data ********************
                        Version 4.20 : July 22, 1994
      File : 001022.tif     Gridsize :  256 x  256   Precision : Real*4


       m00 =    6289.0000    m10 =    -154.9180    m01 =     850.8320

                        Scaling Factor =     .8762
                Analog shift factors :    -.0246       .1353
                Pixel shift factors  :    -3          17
                Pixel shift factors  :     0           0

       m00 =    8201.0000    m10 =      -5.6211    m01 =      22.3633


 NP = 21665 Number of hits = 15745 Percentage =  24.02 Measured db =   10.00
```

Figure 3.4

Typical result for scaling and translation of a single image by the program "pixelize" to produce irradiance normalized, central moments with nominal values of $\mu_{00} = \beta = 8192$ and $\mu_{10} = \mu_{01} = 0$.

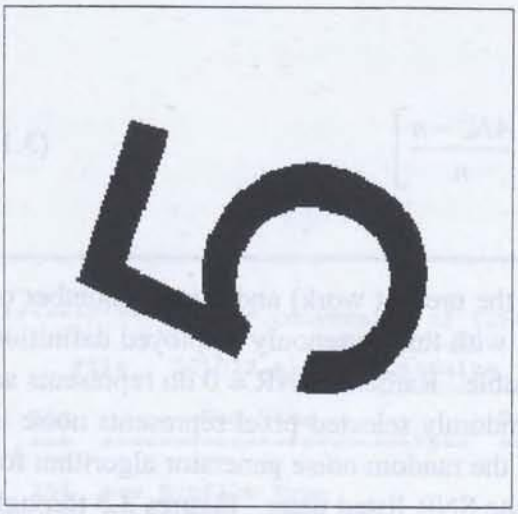
$$SNR(db) = 20 \log\left[\frac{4N^2 - n}{n}\right] \tag{3.1}$$

where the grid dimension is $2N$ x $2N$ (256 x 256 in the present work) and n is the number of pixels changed from the noiseless image. Note that, with this commonly employed definition for images, negative values for the SNR are not possible. Rather, a SNR = 0 db represents an image of "pure noise", i.e., the probability that a randomly selected pixel represents noise is 50%. Table 3.1 includes the number of iterations of the random noise generator algorithm for a fixed seed value which were required to produce the SNR listed there. Figures 3.5 through 3.8 show a complete sequence of 23 images of the same numeral to which increasing levels of noise have been added from noiseless to SNR = 0 db as described in Table 3.1. This series of images serves as an explicit illustration of the SNR levels for the classifier performance measurements presented in Chapter 4. Figure 3.9 shows a series of images for two similar numerals, the '3' and the '8', for SNR's of 3, 2, and 1 db. This figure illustrates that a SNR of 2 db represents that level below which human ability to reliably identify the image begins to fail. To underscore this point, it is left to the reader to decide which image is the '3' and which is the '8' for the 1 db SNR case in figure 3.9.

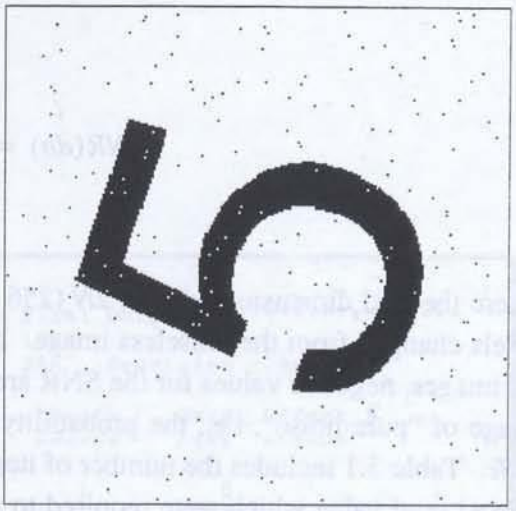
The final task performed by the "pixelize" program is simply to save the scaled and translated image with its additive noise as a binary file containing only values of the 256 x 256 pixel grid (in the present work, 2-byte integers for the binary-valued images). This native format was one readily used by the subsequent processing algorithms required to read the bitmap images.

3.4 : Calculation of the Raw Feature Vector

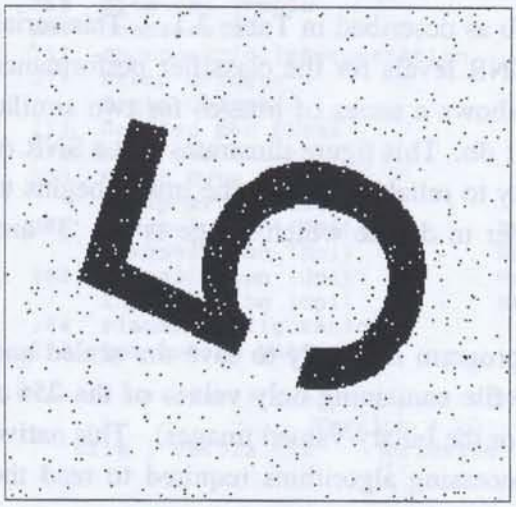
The calculation of the raw (unnormalized) feature vector is a straight forward calculation involving the double summation (integration) of the 256 x 256 'pix' file produced by the program "pixelize" multiplied by the appropriate basis function previously calculated and stored off-line. Care was exercised to ensure that no contribution was made to the calculation for any pixels lying outside of the unit circle. The programs "genfvzer" and "genfvwal", listed in Appendix A, were used to execute this step. The summation formulas used are those given by eqs. (2.56) through (2.61). Each such double summation produces one component of the feature vector in the case of the SZRP, PZRP, WRF, and HRF basis functions. For the complex-valued SZF and PZF bases, separate double summations are carried out for each image with the real and imaginary parts of the basis function and the feature vector component formed by taking the square root of the sum of these integrals squared. All of these double summations are carried out without any weighting function or



SNR = ∞ db

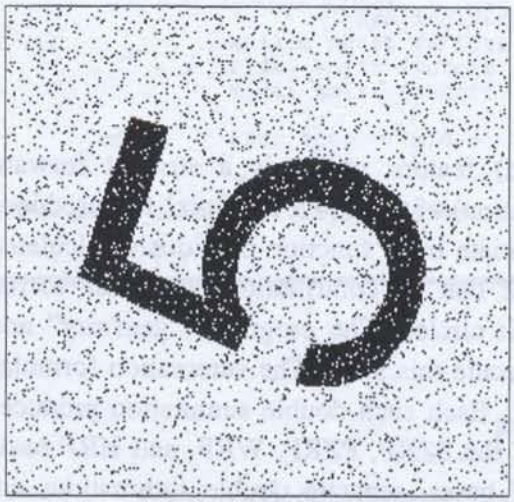


SNR = 50 db

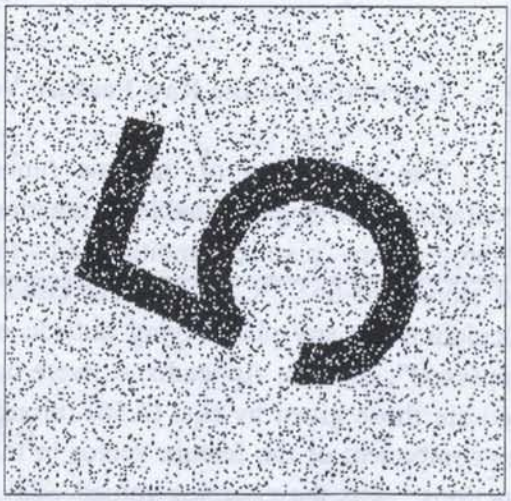


SNR = 40 db

SNR = 30 db



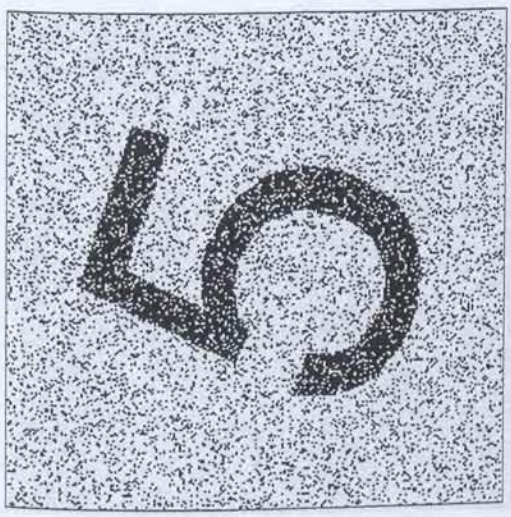
SNR = 25 db



SNR = 20 db

Figure 3.5

Illustration of the image degradation with decreasing SNR ratio from noiseless to SNR = 20 db.

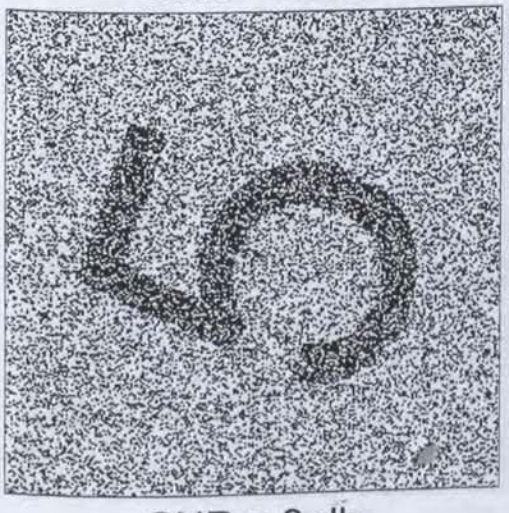


SNR = 15 db

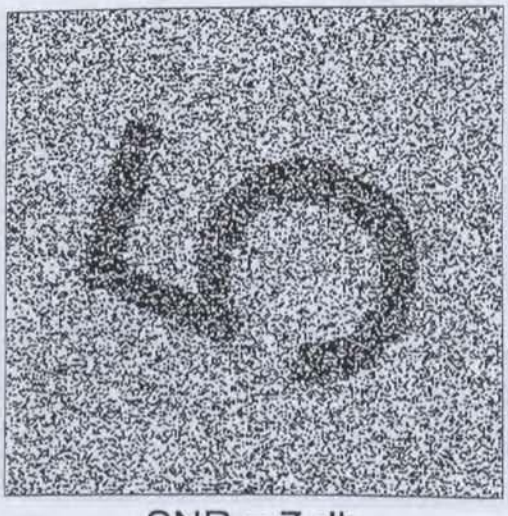
SNR = 12 db

SNR = 10 db

SNR = 9 db



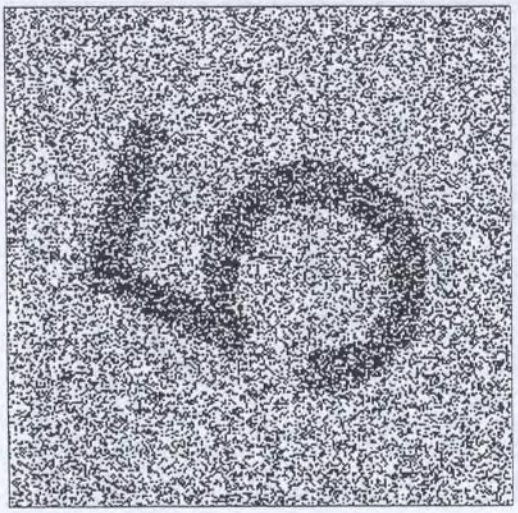
SNR = 8 db



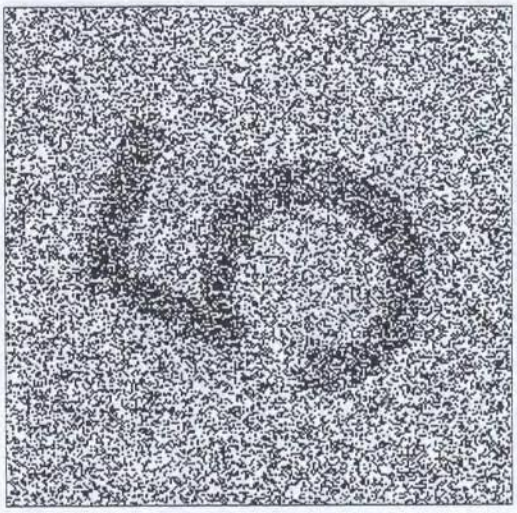
SNR = 7 db

Figure 3.6

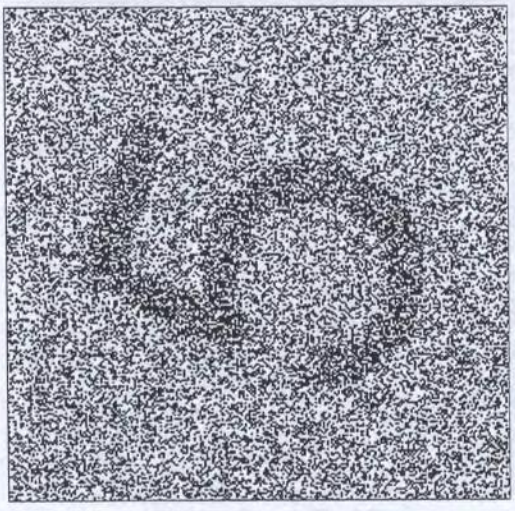
Illustration of the image degradation with decreasing SNR ratio from SNR = 15 db to SNR = 7 db.



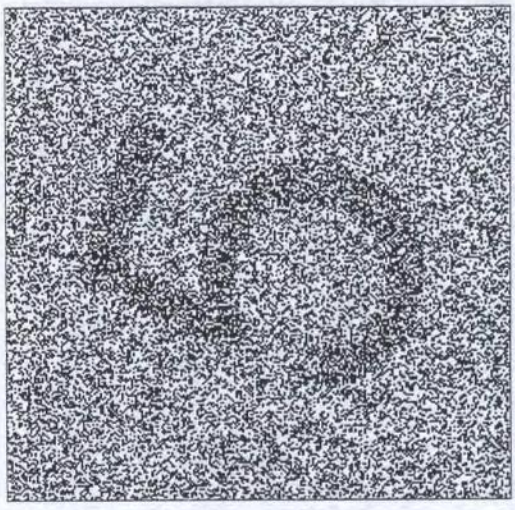
SNR = 6 db



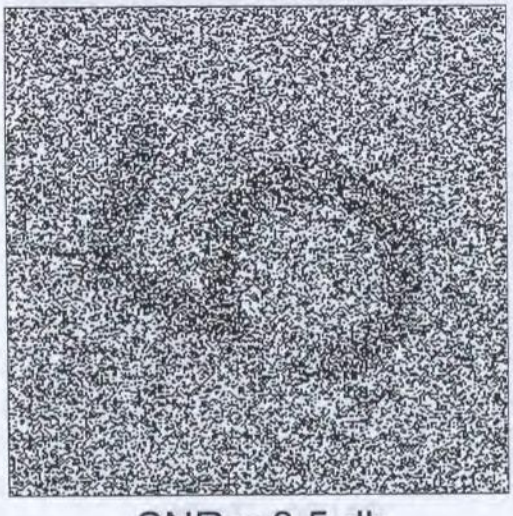
SNR = 5 db



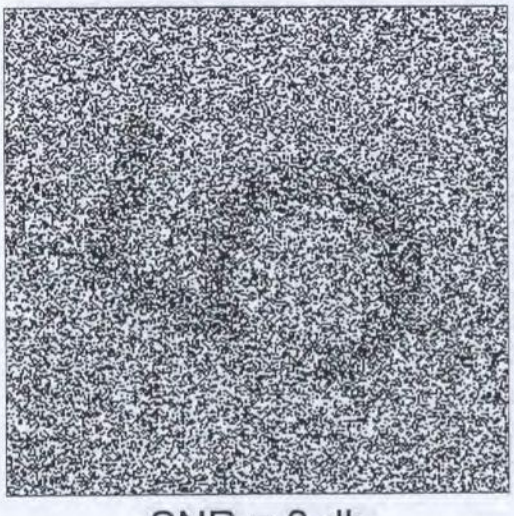
SNR = 4.5 db



SNR = 4 db



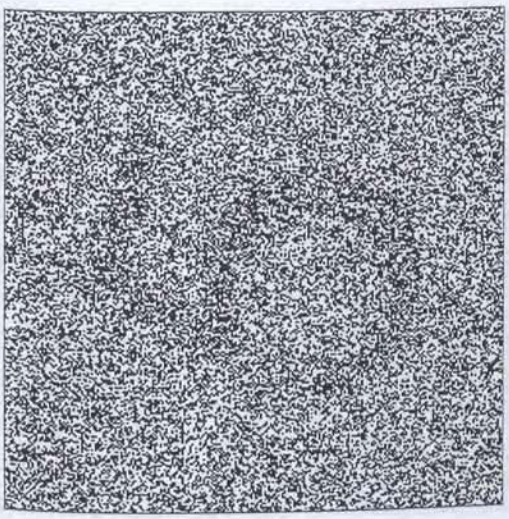
SNR = 3.5 db



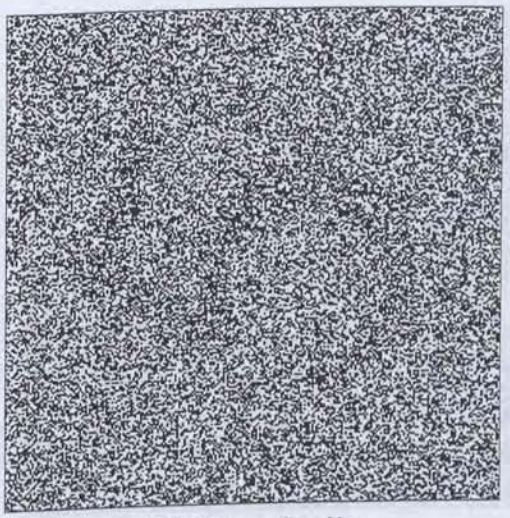
SNR = 3 db

Figure 3.7

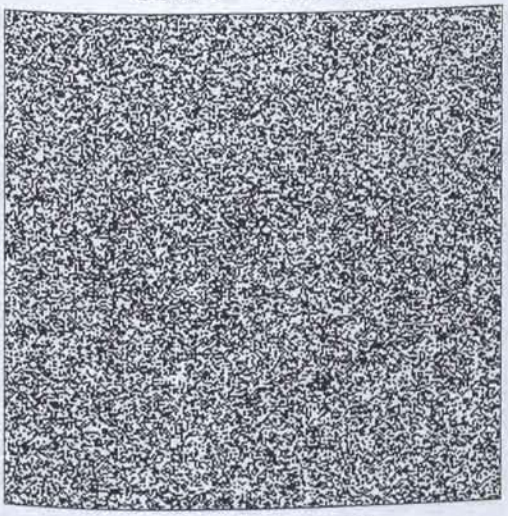
Illustration of the image degradation with decreasing SNR ratio from SNR = 6 db to SNR = 3 db.



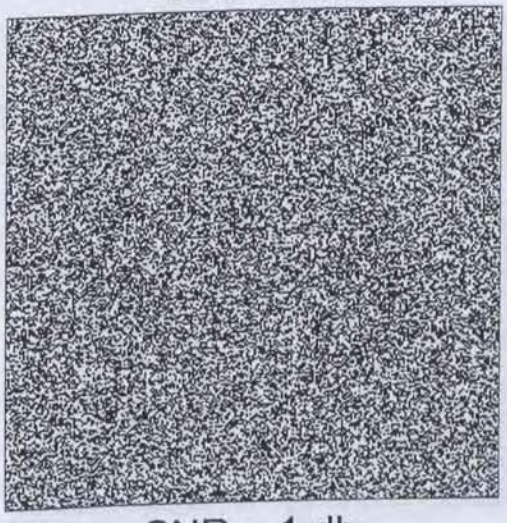
SNR = 2.5 db



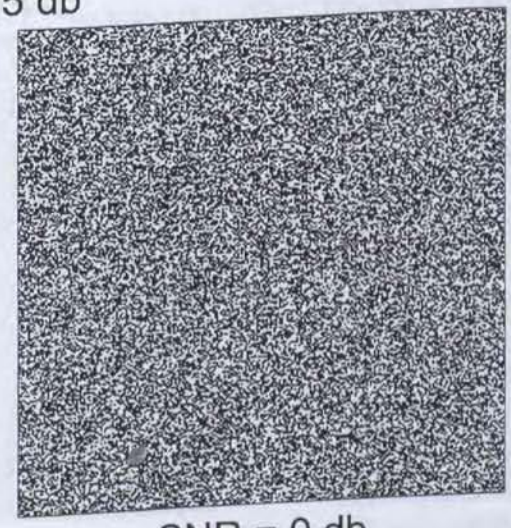
SNR = 2 db



SNR = 1.5 db



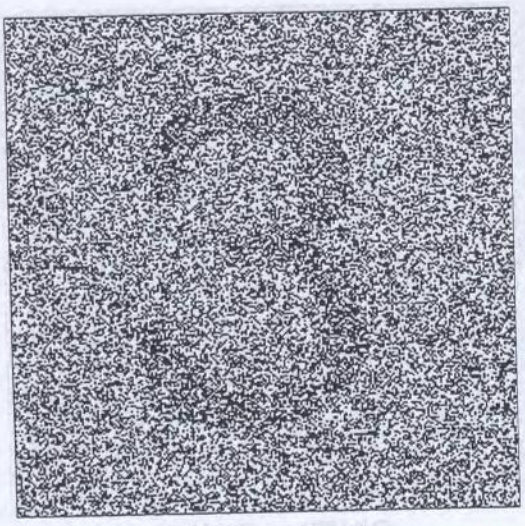
SNR = 1 db



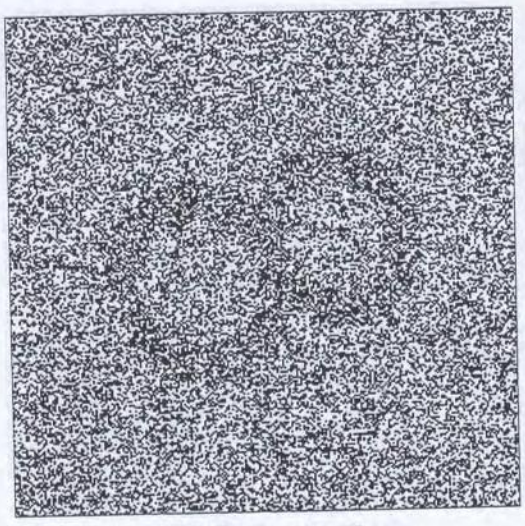
SNR = 0 db

Figure 3.8

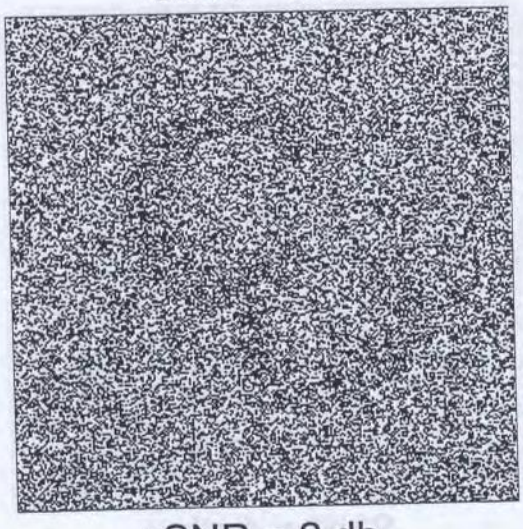
Illustration of the image degradation with decreasing SNR ratio from SNR = 2.5 db to SNR = 0 db (pure noise).



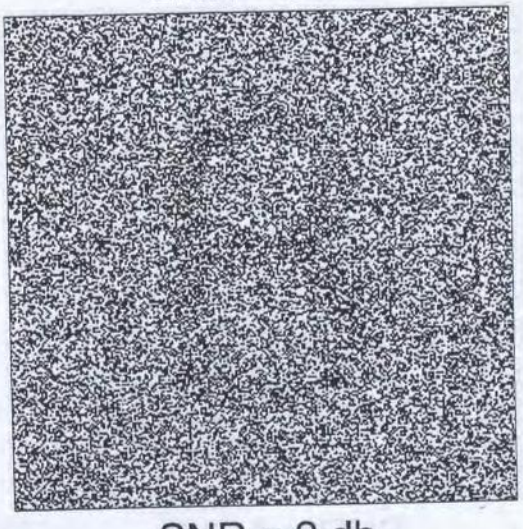
SNR = 3 db



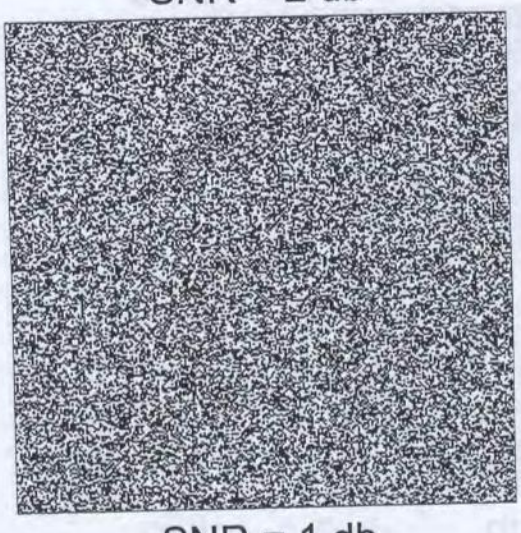
SNR = 3 db



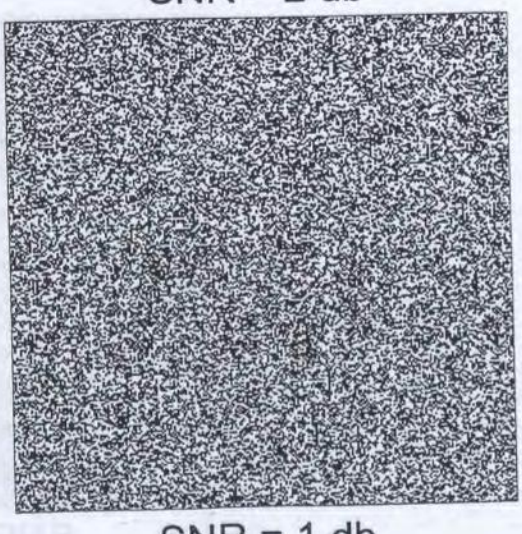
SNR = 2 db



SNR = 2 db



SNR = 1 db



SNR = 1 db

Figure 3.9

Illustration of similar images for low SNR. Human ability for accurate classification begins to sharply deteriorate for SNR's of 2 db and lower.

the use of any integration technique such as the trapezoidal rule. This simplification is a welcome consequence of using the relatively high resolution of 256 x 256 pixel images (indeed, the desire to avoid such mathematical complications was one of the principal reasons for choosing this resolution).

In all cases, the basis functions were calculated and stored offline using a series of different algorithms which permitted the calculation of the functions over a user-specified range of indices. These programs are listed in appendix B. To verify the validity of these calculations, other programs were written to explicitly test the orthonormality of the calculated basis functions over the entire range of indices employed in the calculations. Listings of these programs are included in Appendix C. In addition to these validations, many of the calculated functions were plotted using a commercial graphing package (Axum v. 4.0) as a means to provide visual confirmation of the expected behavior of the functions. The calculations of the SZRP and PZRP bases (which were also calculated as part of the SZF and PZF calculations, respectively) required explicit, algebraic expressions for these polynomials as prescribed by eqs. (2.11), (2.12), (2.25), and (2.26). These expressions were calculated 'by hand' and then checked by expanding the polynomial expressions (up to n = 40) using a commercial, symbolic mathematics program (Mathematica) in order to ensure that these expressions were error free.

The choice of 45 as the dimension of the feature vector used throughout this work was arrived at by some preliminary trial and error as well as being guided by the work reported by several other researchers working on similar problems[58-61,1,12,103]. This number represents the total number of principal moments derived from the use of the SZF and SZRP from order n=3, m=1 through to n=12, m=12, inclusively. While this exact number is therefore somewhat more arbitrary for the remaining basis functions (being n=3, m=0 through to n=9, m=5 for the PZF and PZRP functions and from n=3 to n=47 for the WRF and HRF bases), it was felt that keeping the dimension of the feature vector constant for all the different basis functions allowed for a more meaningful comparison in the final results. In addition, it was observed, by trial and error, that the final performance capability was not particularly sensitive to the value of the feature vector dimension[58,59,103]. One relatively minor difference adopted in the present work when compared to similar work reported in the literature[1,12,58-61,103] is that additional lower order moments were excluded from the feature vectors after it was observed that the very lowest moments (n=1 and n=2 for both the Zernike and Walsh functions) exhibited unusually high sensitivity to the presence of noise in the images (most researchers include the n=2 terms in their feature vectors).

The calculation of the raw feature vectors constituted one of the most time consuming, CPU intensive steps in the overall classification process. To facilitate and automate this step, the programs "genfvzer" and "genfvwal" were written to calculate the feature vectors for sets

of images in a single run, again using simple and direct filename structures to allow for automated processing of the various input and output files. In addition, the programs were written to allow for command line parameters upon invocation, i.e., these programs could be run from a command line without any need for user prompting. This greatly facilitated the task of calculating the feature vectors since these programs could be repeatedly invoked with the aid of batch files to calculate results for series of image sets using appropriate command line parameters. Typical runs using "genfvzer" and "genfvwal" took several days of continuous CPU time but could be carried out without any operator intervention.

3.5 : Normalization of the Feature Vectors

Almost without exception, the models used for the artificial neurons which make up an artificial neural network incorporate a transfer or activation function which serves as a "squashing" or limiting function for the weighted sum of the inputs to that neuron[49,30]. Two of the most common employed transfer functions are the sigmoid and tanh functions. It is an essential requirement in the preparation of data which is to serve as an input to a neural network that the range of this data be limited in its scale in order to avoid any one component dominating the behavior of those neurons to which it is applied. This scaling of network inputs is commonly referred to normalization of the data. The raw feature vectors as calculated by the "genfvzer" and "genfvwal" routines of the previous section typically exhibit a range in their component values of 4 to 5 orders of magnitude. If this data were presented to the network without normalization, the largest of the vector components would completely determine the network's behavior and the training process. The inevitable result would be a neural classifier which would exhibit little if any classification capability since it would be able to discriminate between classes only upon the basis of a few, dominant components.

So fundamental is this process of data normalization that all commercially available neural simulators as well as software designed for specific neural network hardware explicitly incorporate some mechanism to scale the network inputs for both training and test data. In the present case, the Neuralworks Professional Simulator II+ software includes, as a default, the creation of a 'minmax' table derived from the training data. This table is normally used to scale both the training data and any subsequent test data to ranges which are acceptable and meaningful to the neurons within the network. In the present work, however, this feature of the simulator was explicitly disabled in favor of directly calculating the input values according to different normalization schemes.

In all, four different methods of feature vector normalization were evaluated. In order to describe them precisely, the following notation concerning the feature vector components is first introduced:

$$\left.\begin{array}{l} rfv(i,j) = the\ j^{th}\ component\ of\ the\ i^{th}\ raw\ feature\ vector \\ fv(i,j) = the\ normalized\ feature\ vector \\ \mu(j) = mean\ of\ the\ j^{th}\ component\ of\ the\ training\ set\ \{rfv(i,j)\} \\ \left.\begin{array}{l} max1(j) = max[rfv(i,j)] - \mu(j) \\ max2(j) = \mu(j) - min[rfv(i,j)] \end{array}\right\} rfv \in training\ set \end{array}\right\} \quad (3.2)$$

where the quantities $\mu(j)$, $max1(j)$, and $max2(j)$ are determined by allowing the index "i" to vary over the entire training set (i = 1 ... 45 for training set A and i = 1 ... 90 for training set B). In the present work, j = 1, ... 45 where 45 is the dimension of the feature vectors.

1. Standard Normalization (SN).
   This is defined by

$$fv_{SN}(i,j) = \frac{rfv(i,j) - \mu(j)}{\max[\max 1(j), \max 2(j)]} \quad (3.3)$$

   Thus the characteristics of 'standard normalization' are a zero mean with a maximum value of +1 (max1≥max2) or -1 (max1<max2).

2. Positive Normalization (PN).
   This is defined by

$$fv_{PN}(i,j) = \left| (fv_{SN}(i,j) \right| \quad (3.4)$$

   The characteristics of 'positive normalization' are thus a non-zero, positive mean with a maximum value of +1.

3. Bipolar Normalization (BN).
   This form of normalization is defined by

$$fv_{BN}(i,j) = \frac{2 * rfv(i,j) - [\max 1(j) - \max 2(j) - 2 * \mu(j)]}{\max 1(j) + \max 2(j)} \quad (3.5)$$

The characteristics of 'bipolar normalization' are a non-zero mean and at least one component with a value of +1 *and* at least one with a value of -1.

4. Differential Normalization (DN).
This is a radically different type of normalization defined in the following way

$$fv_{DN}(i,j) = \begin{cases} +1; & [fv_{SN}(i,j+1) - fv_{SN}(i,j)] \geq 0 \\ -1; & [fv_{SN}(i,j+1) - fv_{SN}(i,j)] < 0 \end{cases} \tag{3.6}$$

The characteristics of 'differential normalization' are thus that, (1), each component of the feature vector is either +1 or -1 as determined by eq. (3.6) and, (2), the dimension of this feature vector will be one less than that of SN. While, in certain respects, fv_{DN} represents a fundamentally different type of feature vector, it is here regarded as one more type of normalization given its derivation directly from the standard normalized feature vector. The motivation underlying the differential normalization arose from the observation (illustrated in the next chapter) that the *form* of the normalized feature vector SN (indeed, for the PN and BN vectors as well) remained roughly constant as the SNR was decreased for several of the basis functions employed. This suggested that a feature vector based only upon the sign of the differences between adjacent components would be largely unchanged in the presence of noise and, if so, could serve as a viable feature vector.

The SN, PN, and BN feature vectors were generated by the program "Normfv" which is listed in Appendix A. This program normally processed a set of images per execution and, like most of the other core Fortran programs in Appendix A, offered the ability to be run from command line parameters so that several different sets of images could be normalized by creating simple batch files to repeatedly invoke the program. The DN vectors were created separately by the program "gencv" also listed in Appendix A. The results for the classifier performance as a function of the type of normalization are given in Chapter 4. A final note to this section concerns yet one further type of normalization found in the literature[58,12,103] which uses the standard deviation of the raw feature vector components and normalizes each component (of the training set) such that the standard deviation of the normalized feature vector components are exactly equal to one. This type of normalization was tried in the present work and found to give poor results, particularly when compared to the SN and BN schemes. It is believed that this 'standard deviation' normalization scheme fares poorly chiefly owing to the fact that the range of values of the normalized feature vectors can significantly exceed the [-1,1] range characteristic of the schemes described above.

3.6 : The Neural Network Classifier

The decision was made early in this work to employ a three-layer perceptron neural network trained using the backpropagation training algorithm (the cumulative-delta variation). Figure 3.10 shows a schematic illustration of the three-layer perceptron neural network architecture and its associated synaptic weights while figure 3.11 shows an example of the actual Neuralworks Professional Simulator II+ interface screen complete with a trained network and monitoring instruments (note that the 'three-layer' neural network of figure 3.10 would be referred to by some researchers as a "two-layer" network – there is, unfortunately, no consensus on this terminology for neural networks[49]). Measurements were carried out to determine the best choice of transfer functions for the hidden layer and output layer neurons (the input layer of neurons acts only as a distribution layer, i.e., it has a linear transfer function). Four combinations were tried, namely sigmoid-sigmoid, sigmoid-tanh, tanh-sigmoid, and tanh-tanh. The tanh-sigmoid combination was found to be noticeably superior to the others in terms of the speed of convergence and stability in the training process and so was adopted for the neural network throughout all of the present work. Note that this combination matches the fact that the inputs to the neural network are, generally, in the range [-1,+1] while the output neurons are trained to yield results in the range [0,+1].

The dimensionality of the neural network, i.e., the number of neurons in each of the three layers, was largely determined by the dimension of the feature vectors which served as the network inputs and by the number of classes of the result vectors. Thus, in all of the present work, there were 45 input neurons and 9 output neurons with each output neuron representing one of the 9 image classes (a +1 value signifying a 'hit' and a value of 0 signifying a 'miss'). A series of measurements was conducted to determine an approximate optimum for the number of hidden layer neurons. These involved measuring the classifier's performance for a specific basis function set with all other parameters (number of training iterations, initialization parameters, etc.) held constant. These results are given in the following chapter. Note that although some researchers use a Kohenen-type of neural layer on the output (such a layer iteratively cycles the output using a competitive algorithm which ultimately forces the output layer to give only one non-zero result - a "winner-take-all" behavior), the output layer in the neural classifier used in the present work produced analog values, generally in the range of 0 to +1. The values of the outputs representing the individual classes provided very useful information concerning the behavior of the classifier as its performance degenerated with decreasing SNR's in the test data. A methodology by which the outputs were 'rated' is described in the following section.

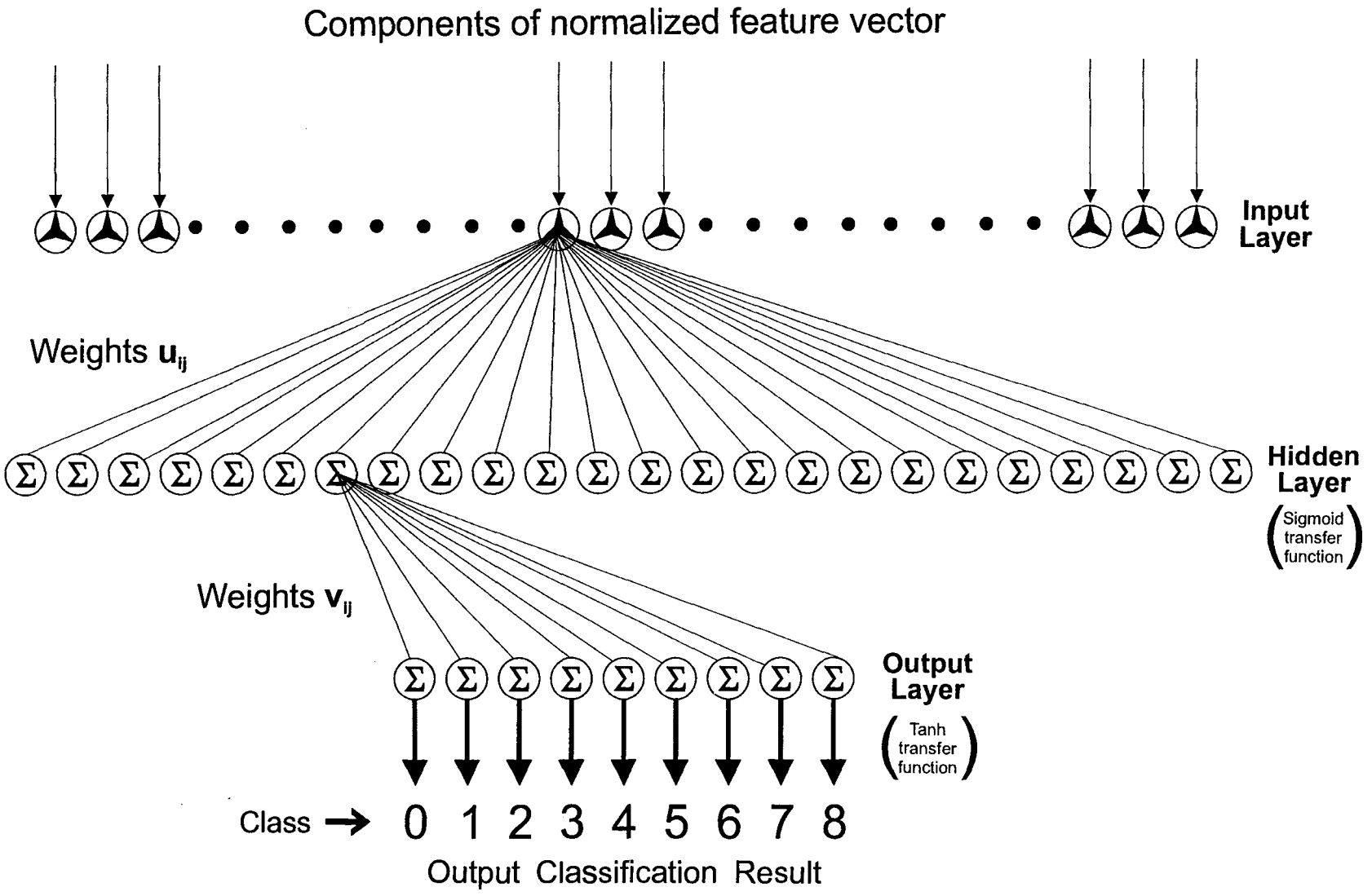


Figure 3.10

Schematic representation of the three-layer, fully interconnected neural network. The weights u_{ij} and v_{ij} are determined by the backpropagation training algorithm.

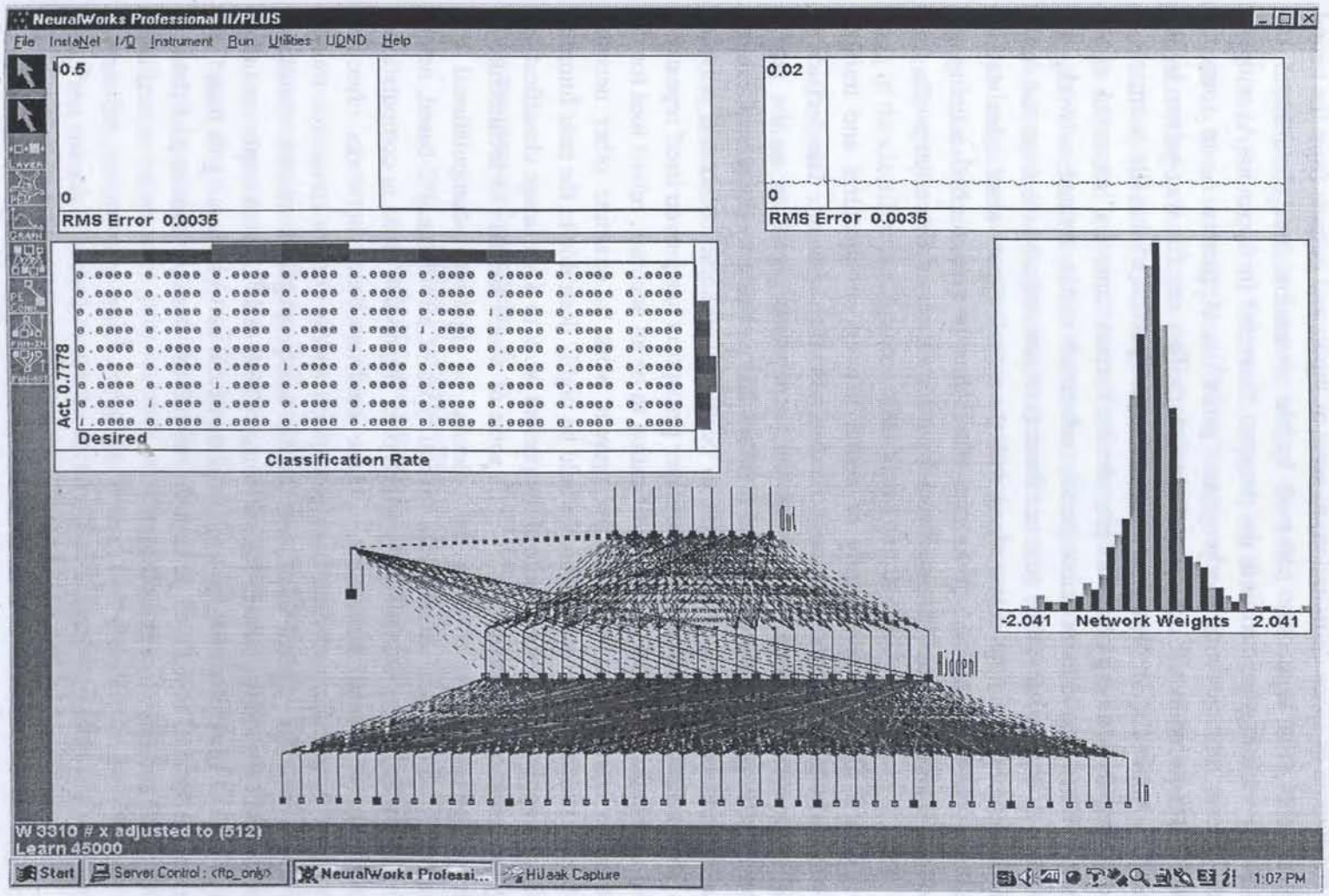


Figure 3.11

The interface screen to the commercial neural network simulator, NeuralWorks Professional II+. In addition to the schematic of the network, the monitoring instruments shown include RMS error plots, the confusion matrix, and the histogram of weight values.

The Neuralworks Professional Simulator II+ uses several different types of ASCII files (with options for comma or space delimitation) to serve as inputs and outputs from the neural network. Specifically, the input file (denoted by the extension 'nna') combined the (normalized) feature vector (generated with the program "normfv" in Appendix A) with the expected result vector (created with the program "genrv" in Appendix A) to form the complete vector file required by the commercial simulator. This nna file was written by the program "gencv" in Appendix A. Note that the result vector portion of this file is critical in that it is used during the training phase to provide the correct "answers" expected of the trained network. When test data is subsequently submitted to the trained network, the simulator produces an output ASCII file (the nnr format) which contains the expected result vector portion of the complete input vector along with the actual output values calculated by the trained network. Note that the form of the input file is thus the same for both training and test data, although the simulator uses the result vector portion for very different purposes.

There exists a richly diverse family of neural network architectures and training algorithms[49], most of which are offered through the Neuralworks software. The decision to employ a multilayer perceptron trained by the backpropagation algorithm as the neural network classifier is, in part, a reflection of the fact that this particular network and training algorithm combination has been and continues to be the "workhorse" of neural network applications. The backpropagation-trained, multilayer perceptron has proven itself repeatedly in many varied applications such as image classification to be an accurate, robust tool for the task. Although no attempt was made in the present work to examine other network architectures for the task at hand, such research would be highly desirable in the near future in connection with the application of the results of the present work to SAR image classification. Recent, newly proposed neural networks[4,23,51,88,124], some of which are akin in architecture to the multilayer perceptron appear to offer the potential of significant computational and possibly performance advantages. In addition, several types of affordable, PC-based, neural network hardware are appearing commercially which offer impressive gains in computational network throughput over neural simulators. These hardware-based networks often are designed to work with specific, non-multilayer-perceptron and/or non-backprogation-trained architectures. However, the present usefulness of the backpropagation-trained, multilayer perceptron should not be underestimated. Recent work by Nair et al[80], for example, on image classification of a series of targets using several different neural network paradigms found that the backpropagation-trained, multilayer perceptron offered the best performance of the five types of networks and training algorithms tested.

3.7 : Assessing the Performance of the Classifier

The analog result from the neural classifier consists of nine outputs, one for each image class (i.e., numeral) with each output lying in the range from 0 to +1. The network is deemed to have correctly identified an image only when the output corresponding to that particular image class is the largest of the nine outputs. The simplest method, therefore, to quantitatively measure the classifier performance is to assign the value +1 (i.e., 100%) for a correct result and a 0 otherwise. However, this clearly is a crude measuring meter. For example, a correct result for which the ratio of the (correct) largest output is only slightly larger than the second largest result represents a value with which would be associated a much smaller "confidence level" than one where the largest output is much greater than the next largest value. Conversely, a 'miss' where the output value for the correct class is only nominally smaller than the largest (but incorrect) output should contribute some value to the rating of the classifier performance other than a simple zero.

To arrive at a more faithful measure of the classifier performance over any given test subset, a sigmoidal weighting function was adopted to quantitatively rate the individual results. Specifically, the model used in the present work was to calculate a value $0 \leq f \leq 1$ for any given result according to

$$f = \frac{1}{1+e^{-4(R-1)}} \tag{3.7}$$

where the measured parameter 'R' is

$$R = \begin{cases} \text{Ratio of correct result to next largest result for a ' hit'} \\ \text{Ratio of correct result to the largest result for a ' miss'} \end{cases} \tag{3.8}$$

The parameters chosen for eq. (3.7) result in $f = 0.50$ when $R = 1$ (i.e., a 50% confidence level when the correct output is the largest result but equals the output from one other output), $f = 0.60$ when $R = 1.1$, and $f = 0.40$ when $R = 0.9$. Although these parameters are somewhat subjective in choice, trials with different parameter values revealed that the final, overall classification values were quite insensitive to these choices. The results for the measurements of classifier performance which are given in the following chapter (performance for each of the 23 test subsets of table 3.1) use the accumulated sum of the values of f as given in eq. (3.7) over one entire test subset (45 images).

One final observation which is pertinent to the method for measuring the classifier's performance is that, in virtually all of the cases studied, the trained neural network, as the SNR of the test images is decreased, fails by tending to predict the *same* result for *all* image classes which are presented to it. For example, once a given network begins to completely fail, it may predict that all test images submitted to it are the numeral '2'. Strictly speaking, for the test subsets which consist of 45 images composed of 5 samples from each class, five of these results are "correct" so that, apparently, a minimum performance for the classifier must be approximately 11%. Clearly, however, such a conclusion is facetious since, at that point, the confidence level in any given result is effectively zero. To more accurately reflect this characteristic, the classifier performance as plotted versus the individual rating for each of the test subsets of table 3.1 is renormalized according to the formula

$$Rating(\%) = 100\frac{f - f_{\min}}{|f_{\max} - f_{\min}|} \tag{3.9}$$

where, for a given test subset, f_{min} and f_{max} are the minimum and maximum values, respectively, of f over the entire 23 test subsets. In practice, this renormalization has no effect on the higher values but sets the lower bound for the measurements to 0% instead of the 11% referred to above.

Chapter 4 : Experimental Results

4.1 : The Feature Vectors

It warrants repeating that the input to the neural network classifier is *not* the image data but is instead the feature vector derived using one of the basis function sets described in Chapter 2, i.e., the neural network is employed as a classifier of feature vectors. It is, therefore, a reasonable expectation that a direct examination and comparison of the feature vectors themselves should reflect many of the characteristics and attributes of the images themselves. This section will present a several illustrations of the dependence of the feature vectors upon different parameters in order to portray graphically many of the characteristics which ultimately will be reflected in the performance of the neural network classifier.

Figures 4.1 through 4.12 show the feature vectors for the numerals '0' through '8' for each of the twelve combinations of function basis and training set with standard normalization throughout. In each case, the feature vectors shown are averages of five, single image feature vectors corresponding to the same numeral with five different, random combinations of scale, translation, and rotation (these are the 45 noiseless images which make up test01 of table 3.1). Note that, even though these figures together present the reader with a considerable number of illustrative examples, the number of feature vectors illustrated represents only a tiny percentage of the 50,000+ feature vectors generated and tested during the course of the present work.

One very general observation which follows from an inspection of these twelve figures is that the basic character of the feature vectors are quite different in form for the eight sets of Zernike-based feature vectors in comparison to the four sets of Walsh-Haar based vectors. Furthermore, the natures of the SZRP and PZRP based feature vectors are more similar to each other than to either the SZF or PZF based vectors. Note that some of the feature vectors shown in figures 4.1 to 4.12 have components for which the magnitude is > 1. It is important to remember that all of the normalization schemes for the feature vectors described in section 3.5 ensure that the *training* feature vector components have magnitudes < 1. The *test* feature vector components, however, may be > 1 in magnitude and, as will be shown shortly, the addition of noise can lead to feature vector components, in certain cases, which change by more than an order of magnitude from their values in the noiseless case. Finally, again in very general terms, note that the feature vectors for a specific basis normalized on training set A (noiseless data) are very similar to those normalized on training set B (mixed noiseless and noisy data) although significant differences are evident for the SZRP and PZRP bases.

It is anticipated that, within the set of nine numerals, there should exist greater similarities for the feature vectors derived from those numerals which are themselves

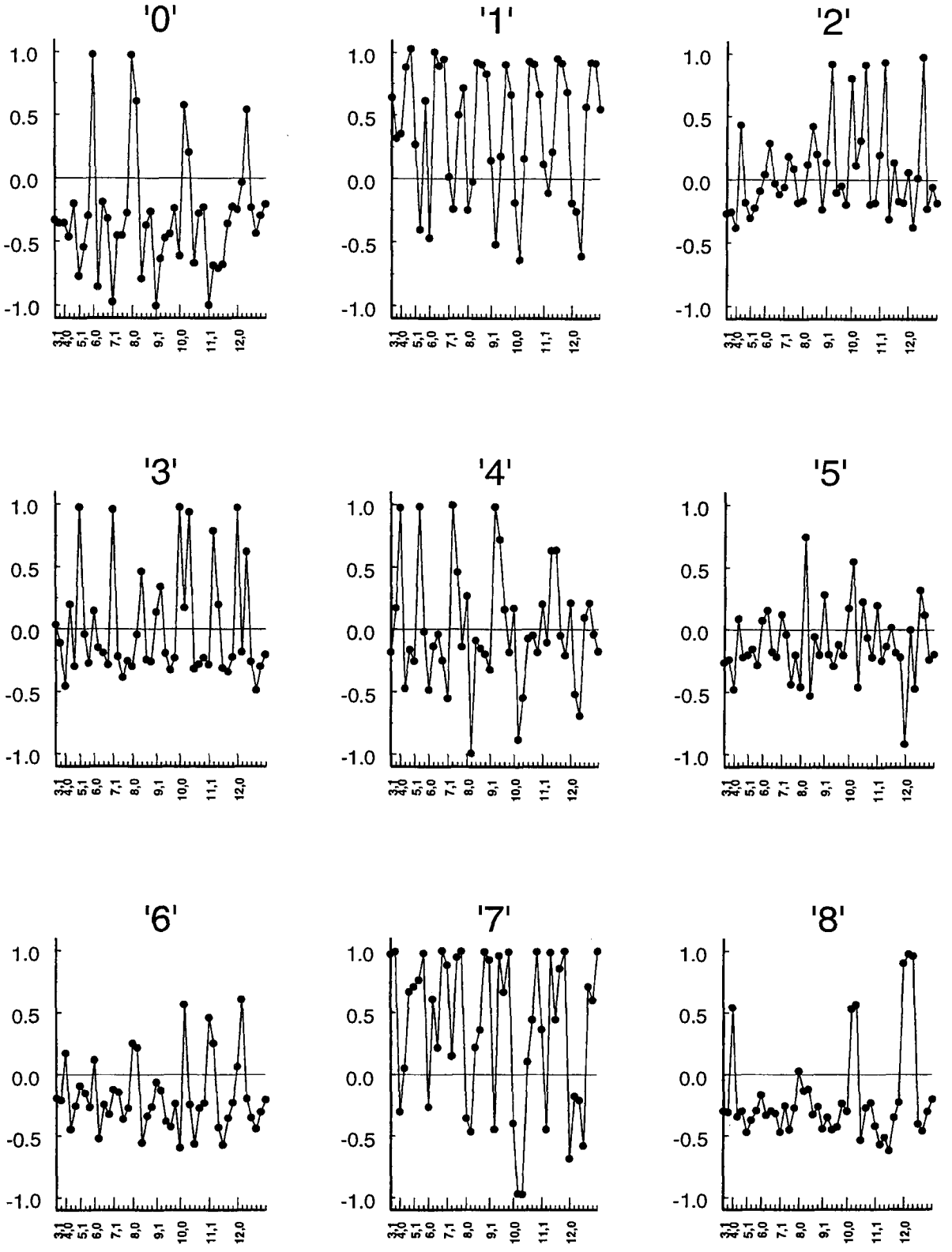


Figure 4.1

Feature vectors for the numerals '0' through '8' for the SZF basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order {n,m} of the Zernike functions.

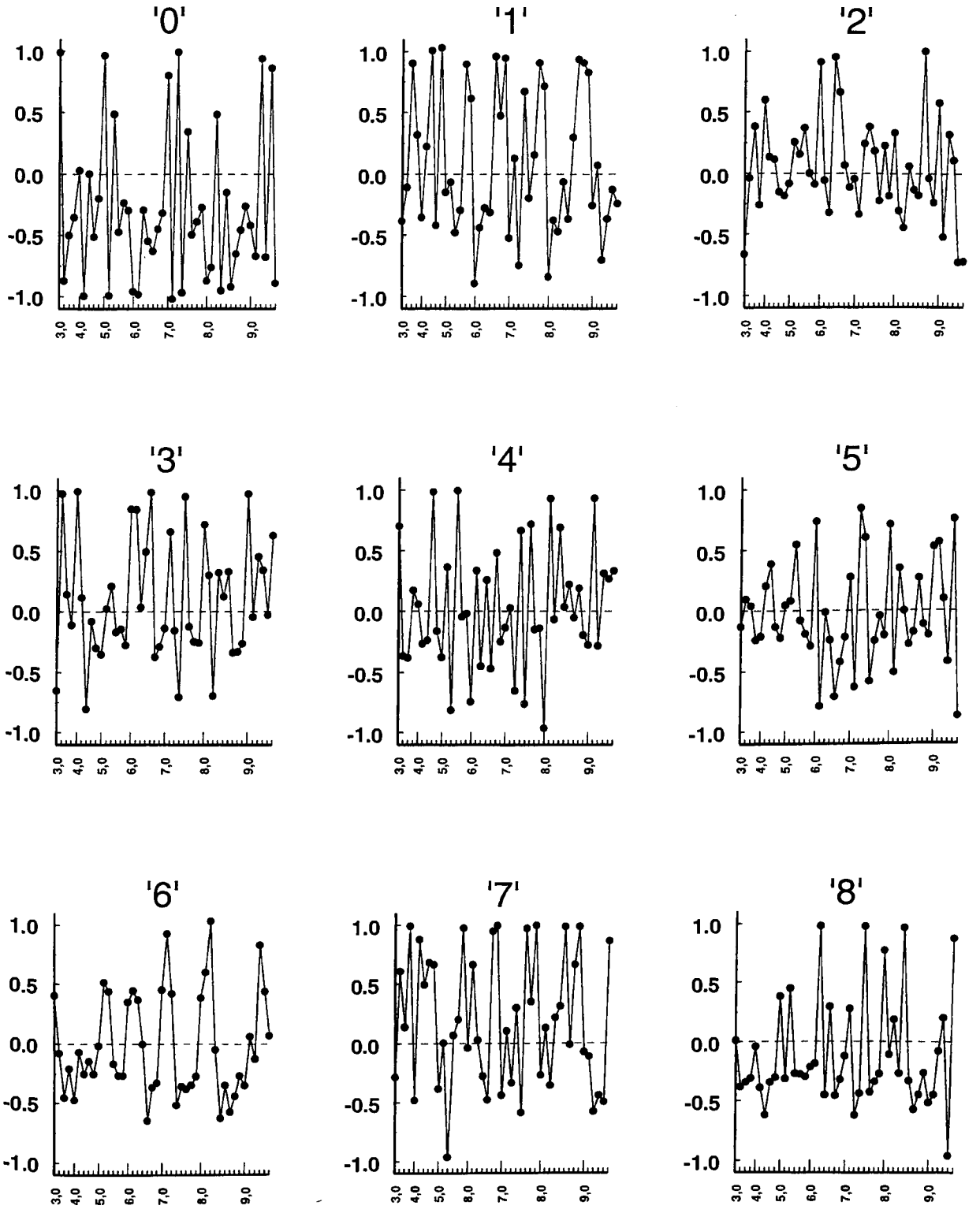


Figure 4.2

Feature vectors for the numerals '0' through '8' for the PZF basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order {n,m} of the Zernike functions.

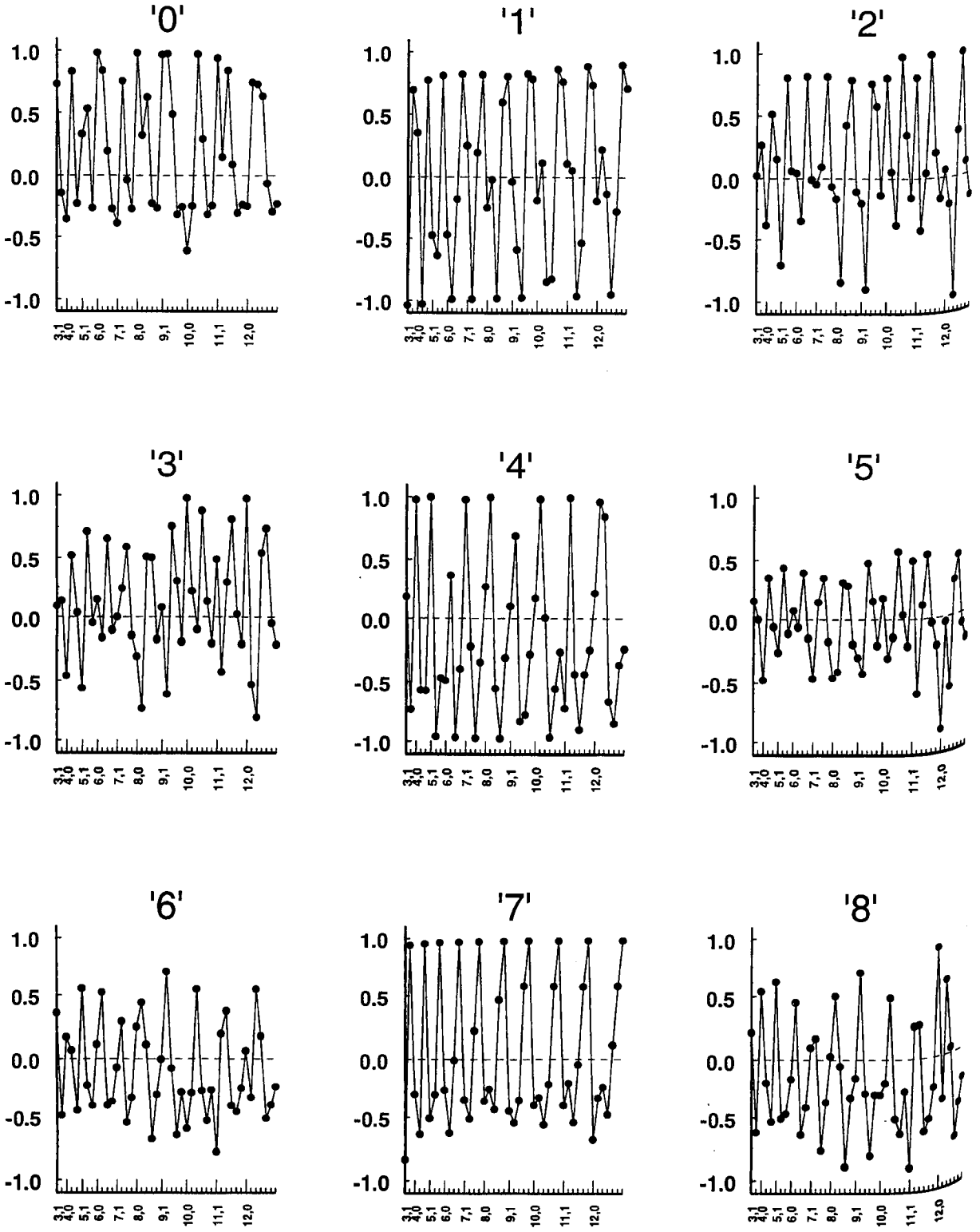


Figure 4.3

Feature vectors for the numerals '0' through '8' for the SZRP basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order {n,m} of the Zernike functions.

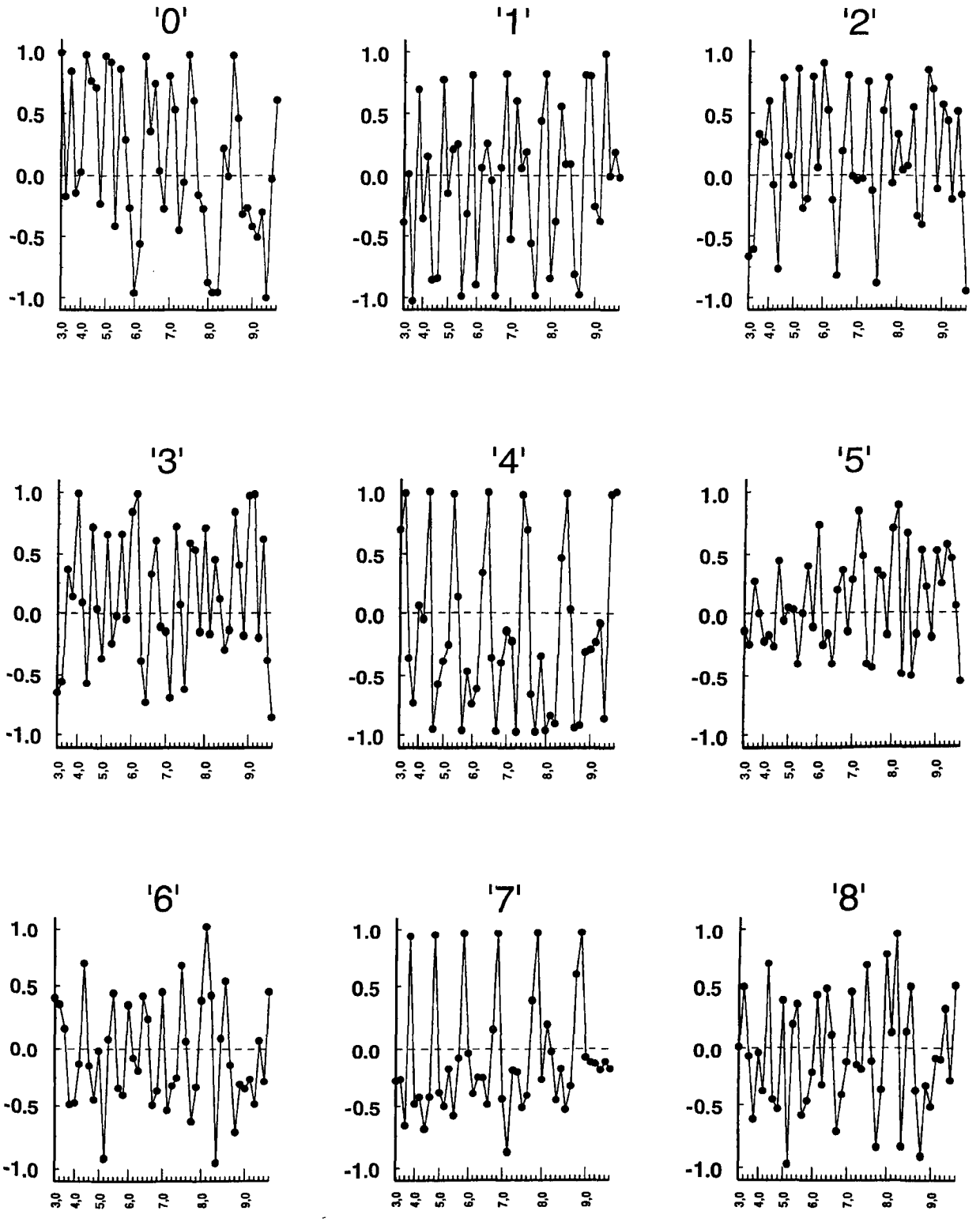


Figure 4.4

Feature vectors for the numerals '0' through '8' for the PZRP basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order {n,m} of the Zernike functions.

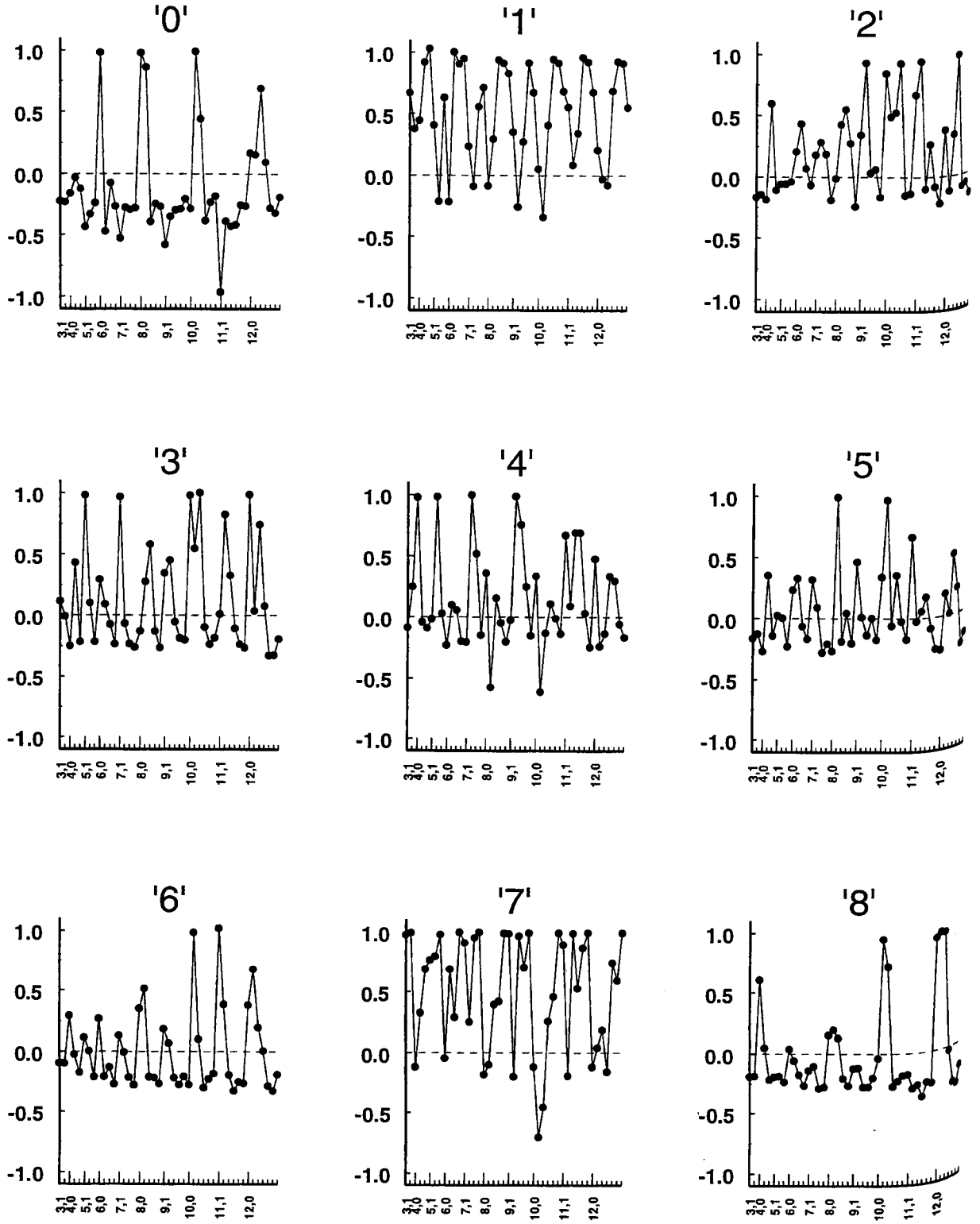


Figure 4.5

Feature vectors for the numerals '0' through '8' for the SZF basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order {n,m} of the Zernike functions.

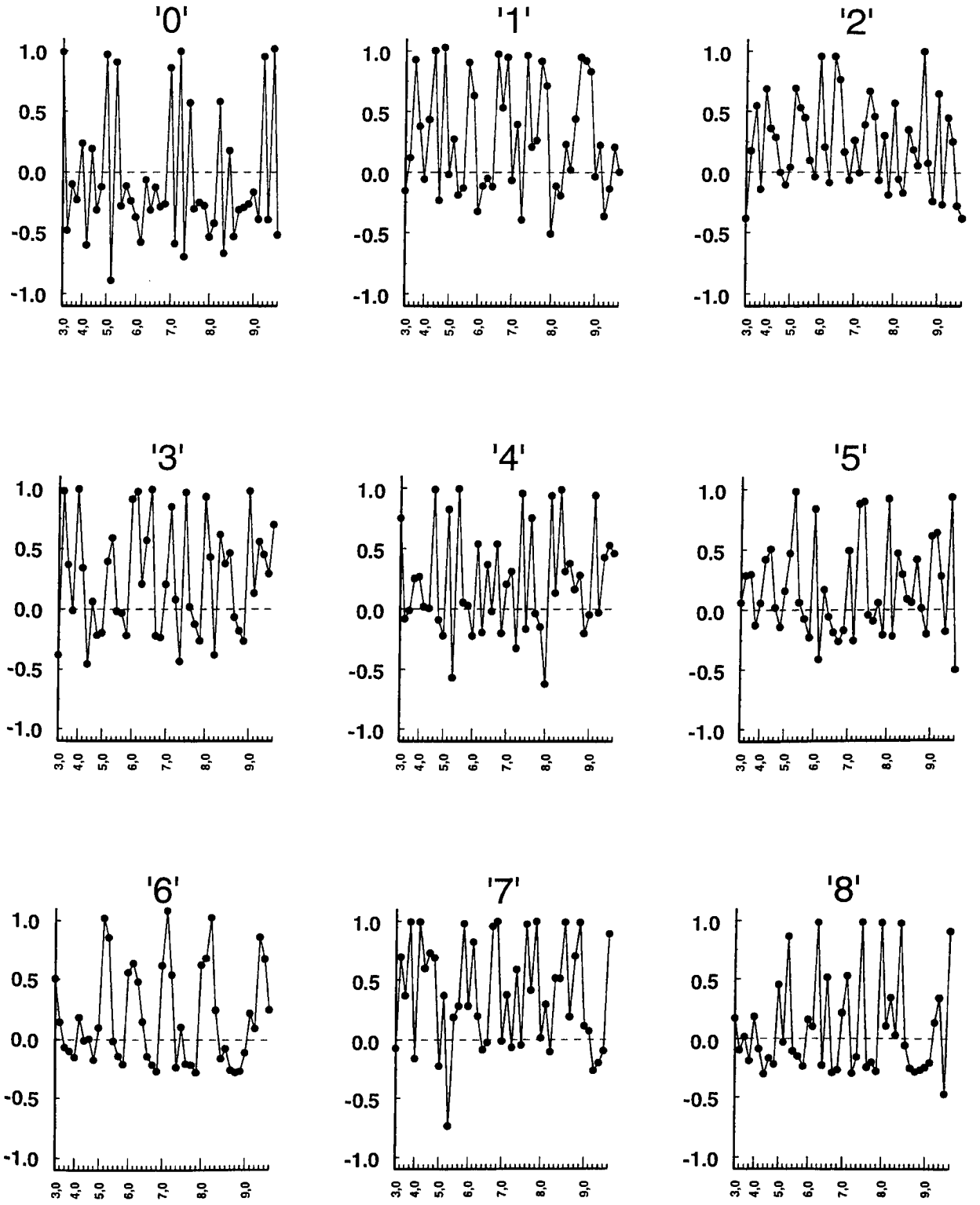


Figure 4.6

Feature vectors for the numerals '0' through '8' for the PZF basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order {n,m} of the Zernike functions.

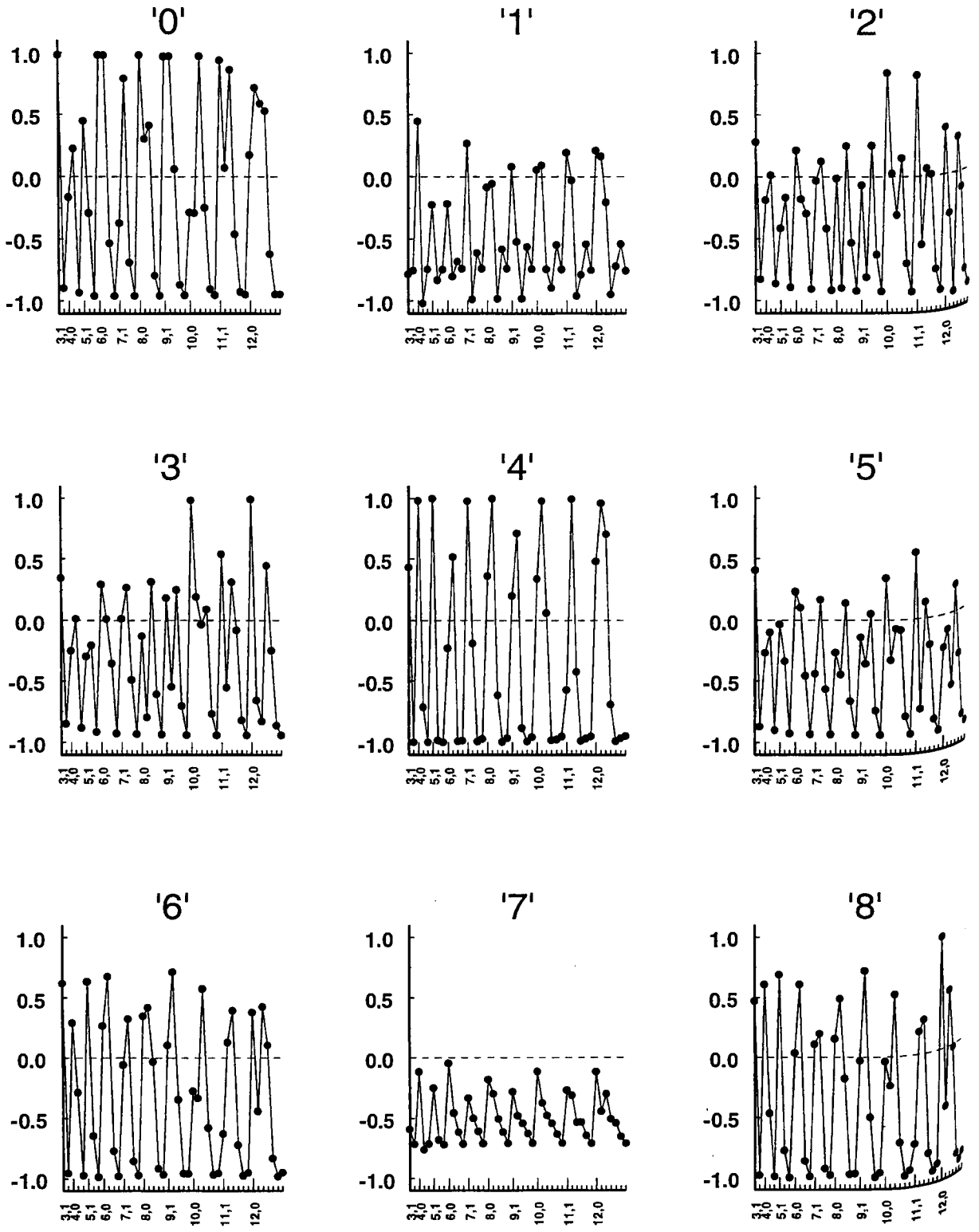


Figure 4.7

Feature vectors for the numerals '0' through '8' for the SZRP basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order {n,m} of the Zernike functions.

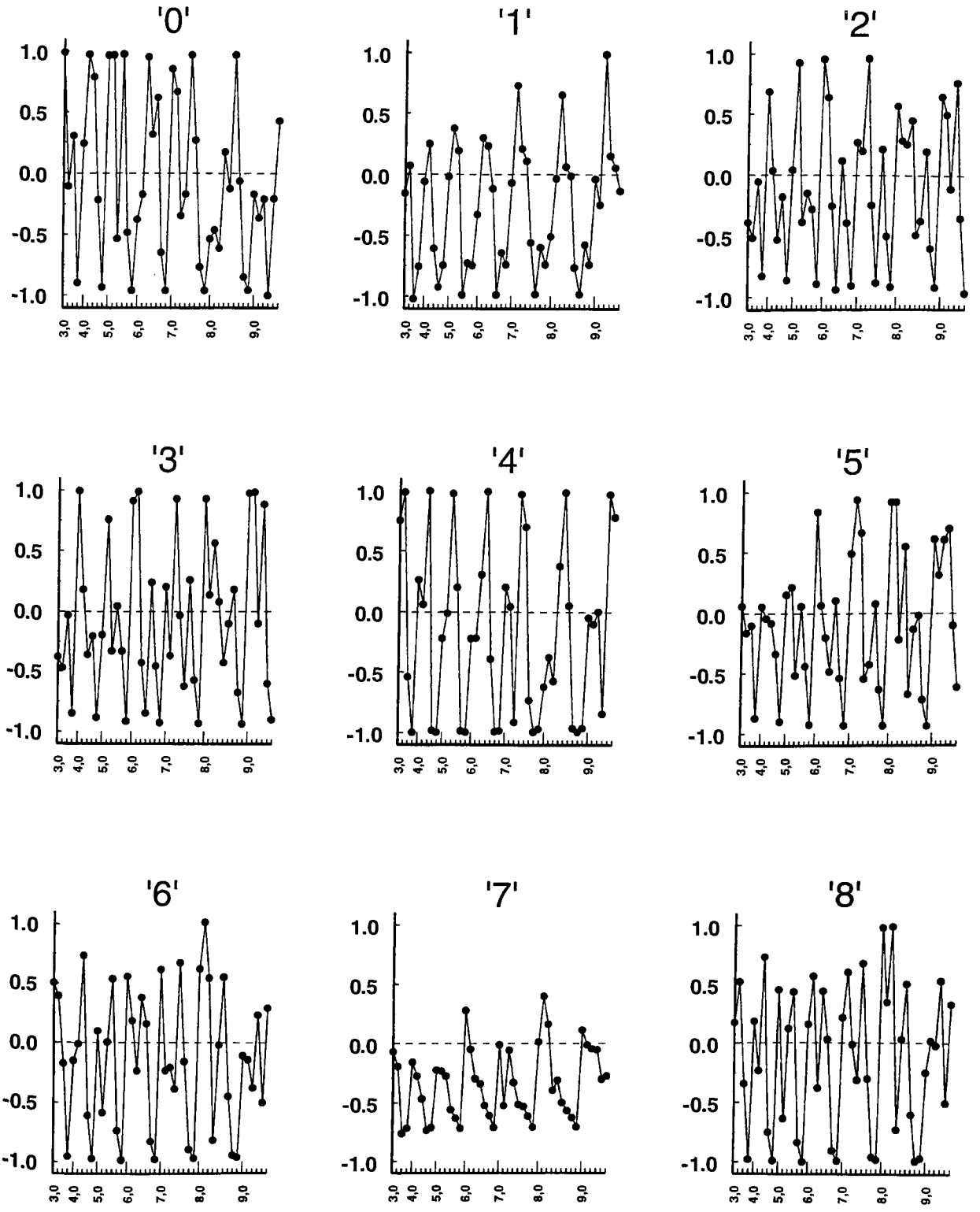


Figure 4.8

Feature vectors for the numerals '0' through '8' for the PZRP basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order {n,m} of the Zernike functions.

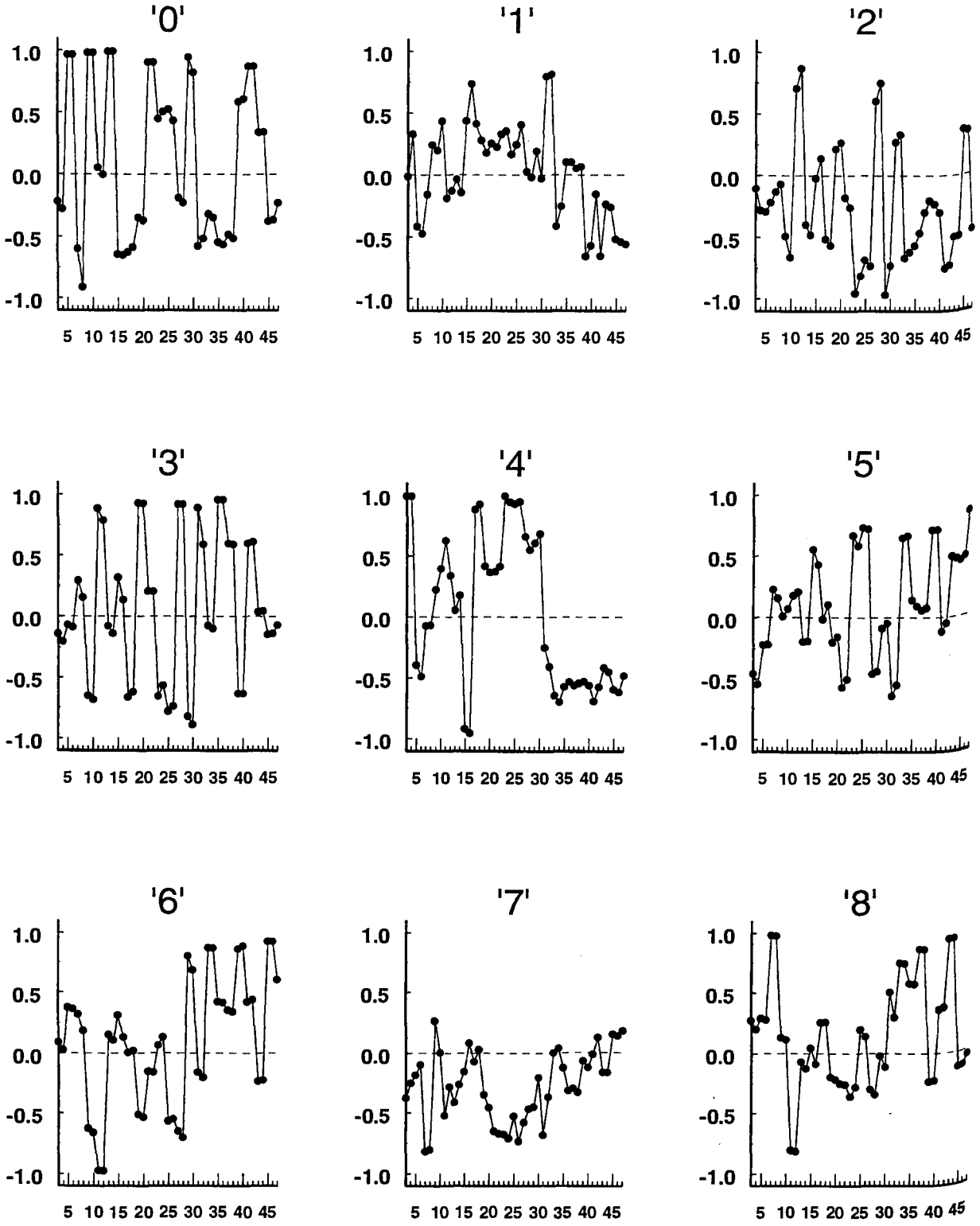


Figure 4.9

Feature vectors for the numerals '0' through '8' for the WRF basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order n of the WRF functions.

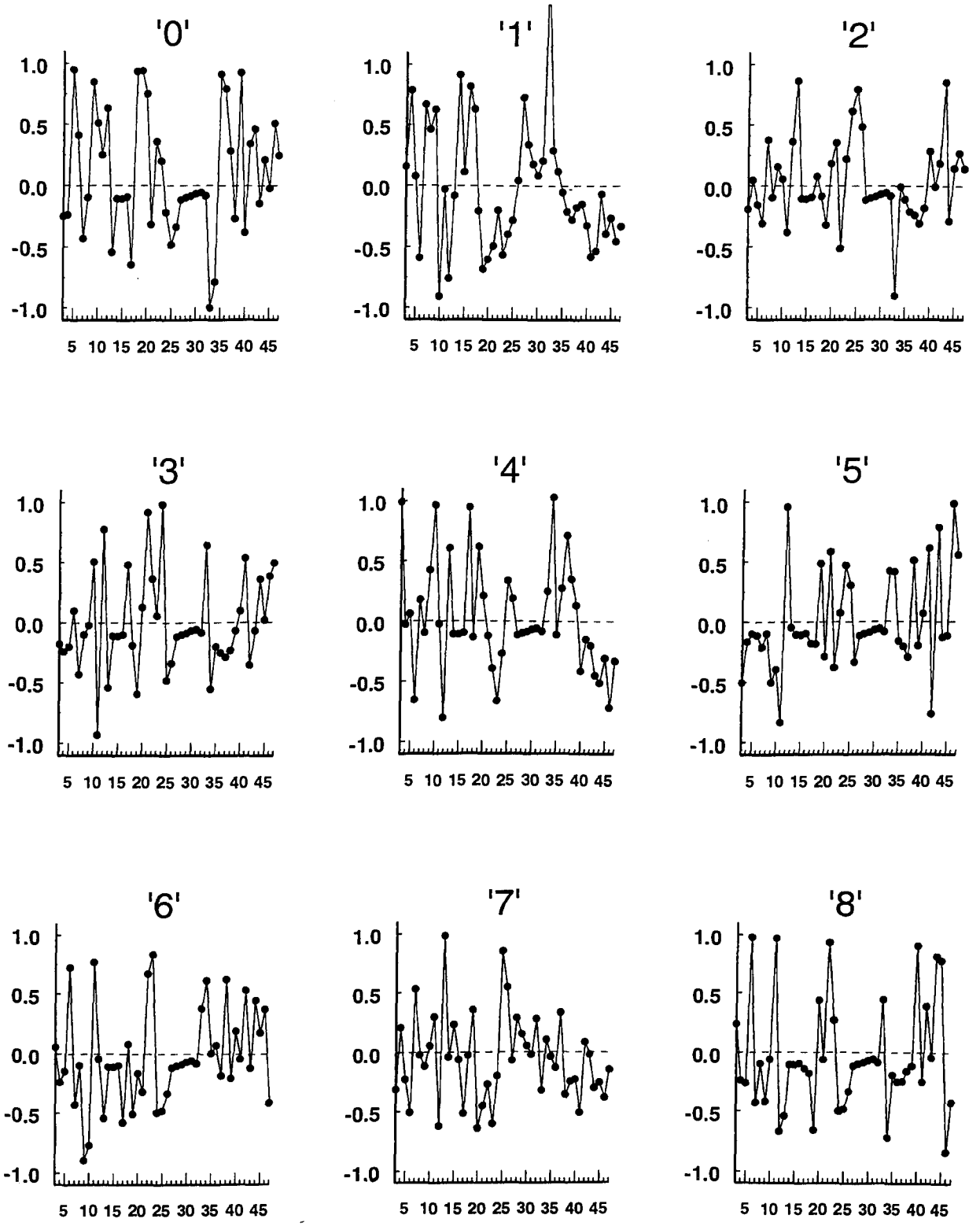


Figure 4.10

Feature vectors for the numerals '0' through '8' for the HRF basis, noiseless images, and standard normalization using Training Set A (noiseless). The horizontal axes show the order n of the HRF functions.

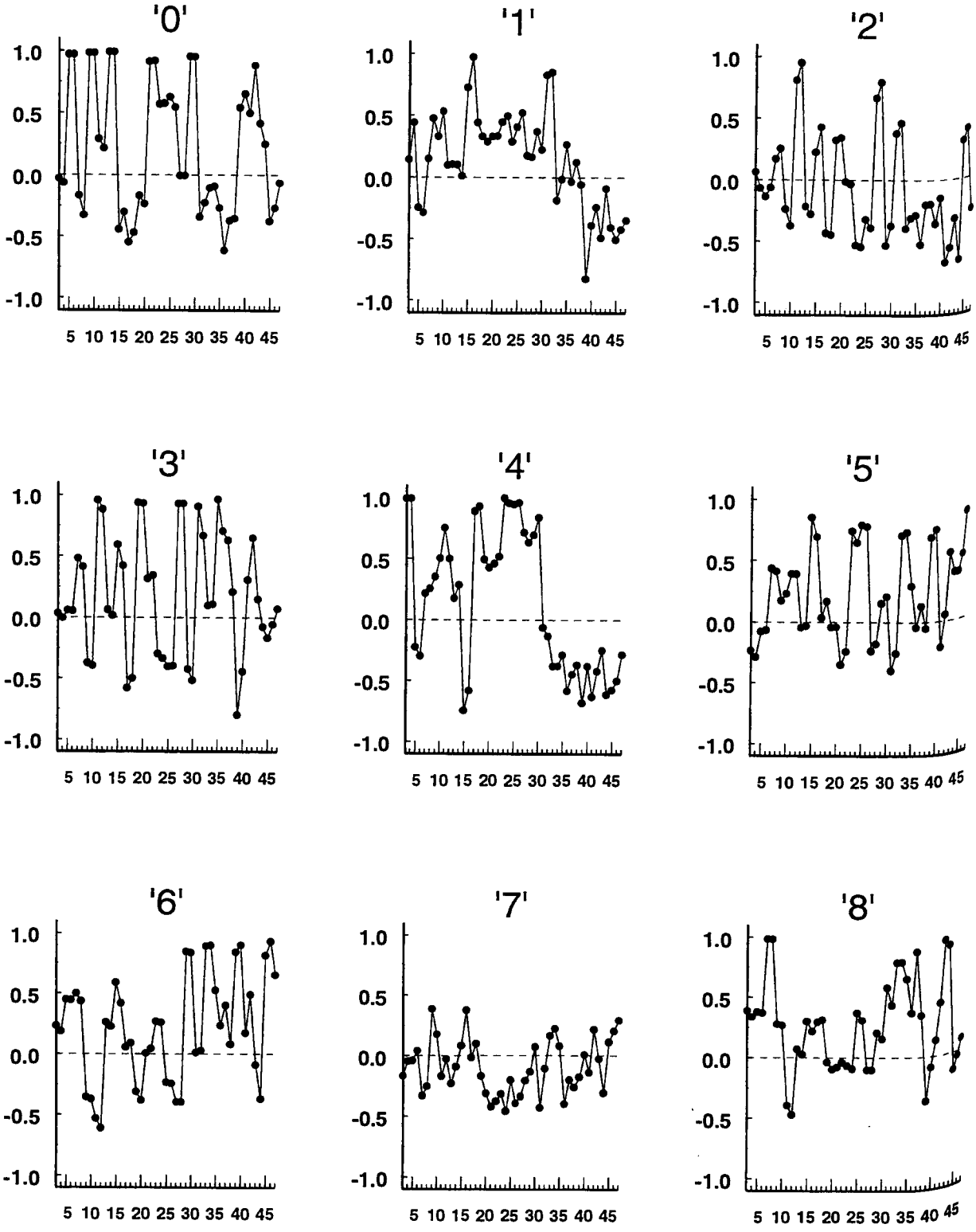


Figure 4.11

Feature vectors for the numerals '0' through '8' for the WRF basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order n of the WRF functions.

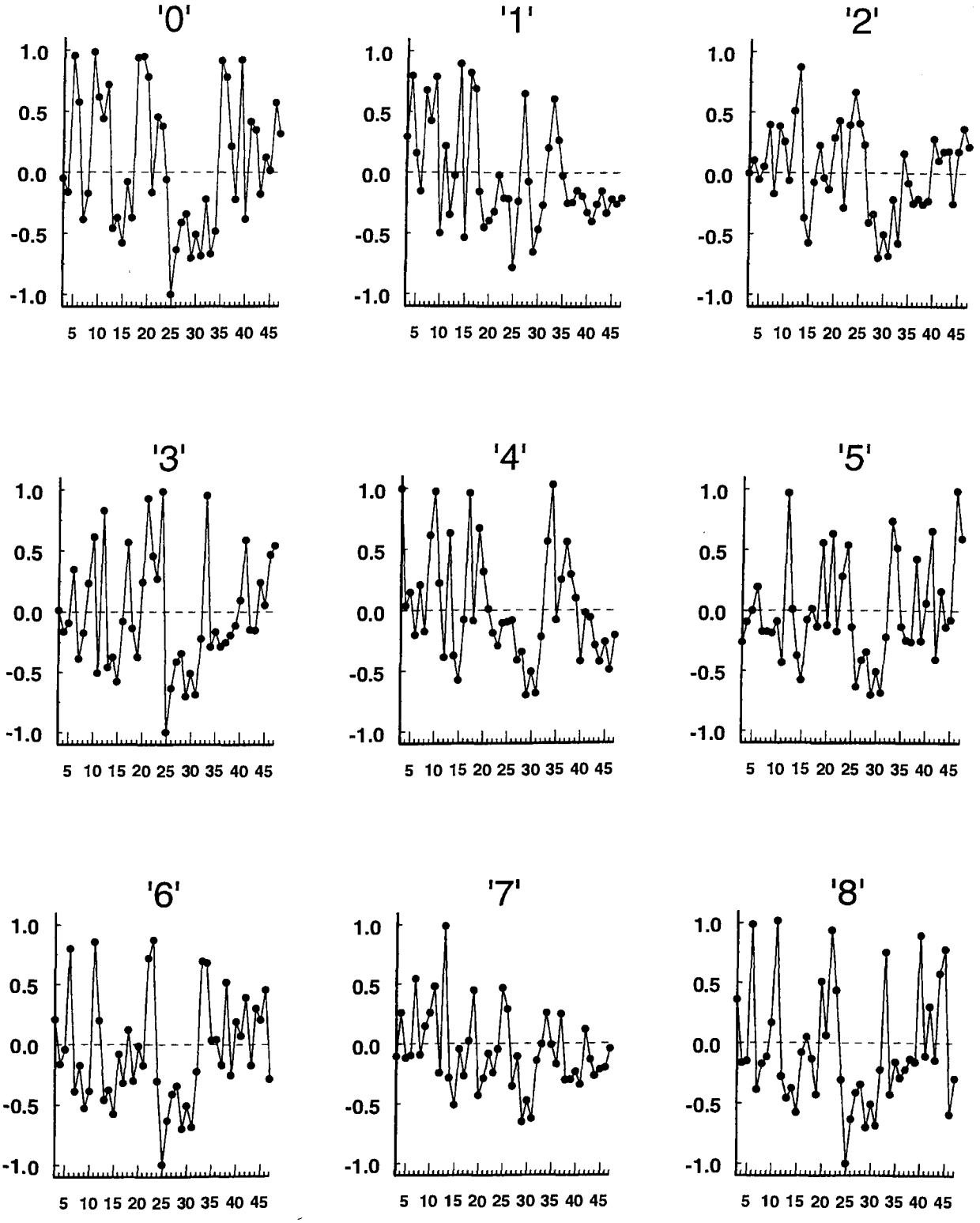


Figure 4.12

Feature vectors for the numerals '0' through '8' for the HRF basis, noiseless images, and standard normalization using Training Set B. The horizontal axes show the order n of the HRF functions.

somewhat similar in appearance. Thus, for example, the feature vectors for the numerals '0', '3', and '8' should be more "correlated" than, for example, the feature vectors for the numerals '0' and '1'. This expectation is illustrated in figure 4.13 which shows the difference between the feature vectors for '0' and '8' in part (a) and the difference between the feature vectors for '0' and '1' in part (b) (all for the SZF basis with standard normalization against the training set A). Clearly, as the number of terms which lie close to zero reveals, the feature vectors for the numerals '0' and '8' are much more closely related than those for the numerals '0' and '1'. In effect, the neural classifier must distinguish between a '0' and an '8' using a reduced number of feature vector components. This means that when the neural classifier's performance is measured against the full, 1035 image test set, it should be expected that the classifier will 'fail' sooner on the '0' and '8' feature vectors than on the '0' and '1' or '1' and '8' combinations. It was noted in Chapter 2 that the feature vectors employed in this work, being composed of purely principal or 'true' invariants, are also invariant to mirror transformations of the basic images[114]. The numerals '2' and '5' are approximately mirror images of one another and figure 4.14, which shows the difference between their feature vectors, clearly illustrates this "mirror invariance" by the number of near zero differences between the two image vectors.

The question of how accurately invariance is obeyed for the actual calculated feature vectors is addressed in figures 4.15 through 4.20 for the six different basis functions using the numeral '3' in each case as a typical representative of the test image set. Each of these six figures shows five plots of the feature vector for the numeral '3' for the case of noiseless test images (test01 of table 3.1), each of the five vectors representing a different scale, translation, and orientation of the numeral. Clearly, the approximation to exact invariance for the case of the SZF, PZF, SZRP, and PZRP bases is excellent with only relatively minor fluctuations from the average. Furthermore, of these four Zernike bases, the SZRP and PZRP bases come closest to perfect invariance in the calculated feature vectors. This is in sharp contrast to the case for the Walsh functions, figures 4.19 and 4.20. Evidently, the Walsh and Haar bases produce feature vectors with considerably more variance in their values, particularly for the higher sequency components of the feature vectors. The principal cause of this increased departure from exact invariance is believed to be chiefly the result of the finite pixel size and the fact that the image cannot be *exactly* scaled and translated to its center of optical mass as was discussed in sections 2.5 and 3.3. Although the alignment of the scaled and shifted image with the basis function can only be done to an accuracy of approximately one-half pixel width, such "misalignment" introduces very little error into the double integral calculation of the feature vector component in the cases involving the smoothly varying, analog Zernike bases. For the Walsh functions, however, this "misalignment" obviously produces greater errors given the step-like nature of these basis functions. This misalignment is expected to have its greatest effect the finer the step sizes for the Walsh function and this expectation is borne out

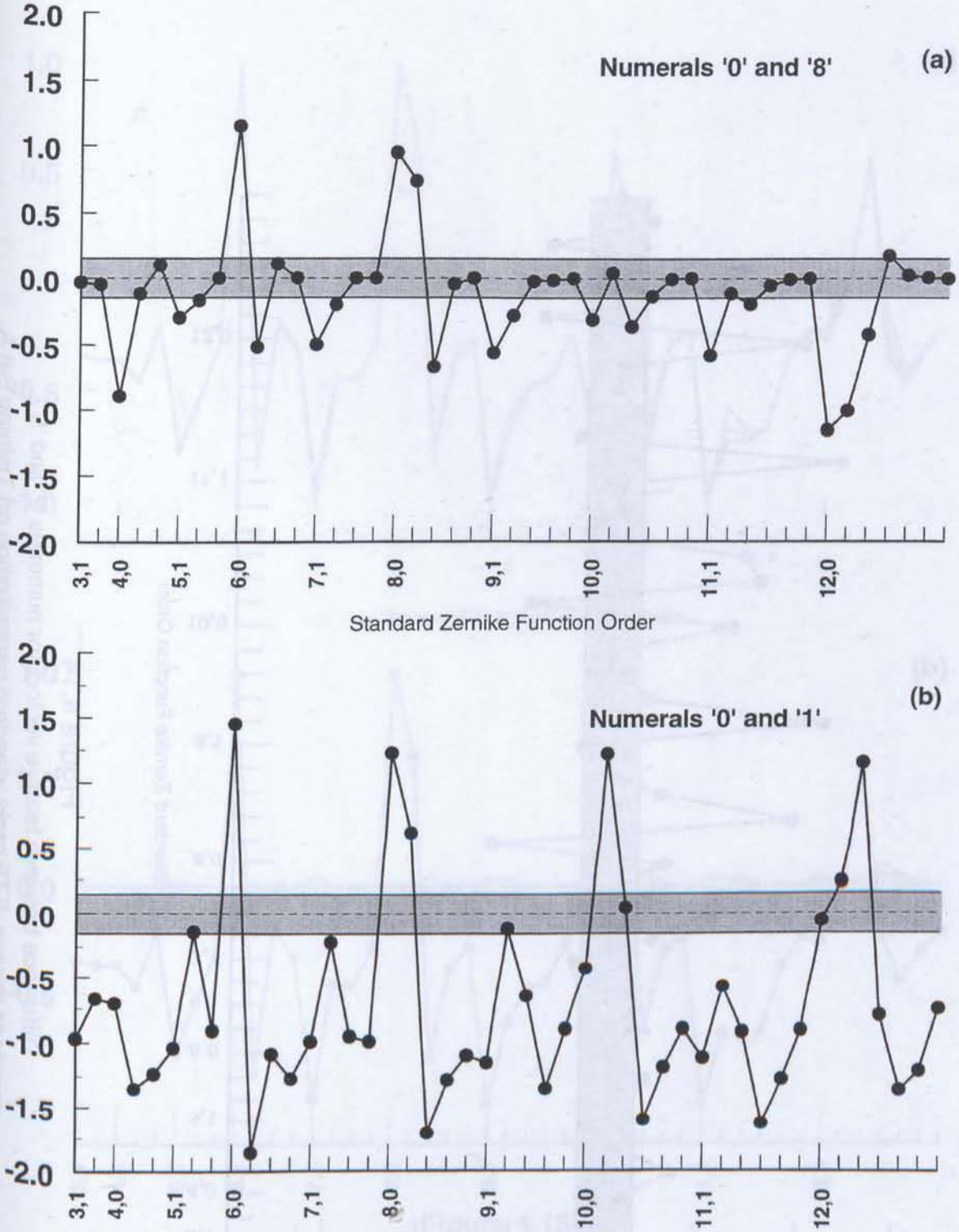


Figure 4.13

Difference between feature vectors for SZF basis, noiseless images, standard normalization on Training Set A.
(a) Feature vector difference for '0' and '8'
(b) Feature vector difference for '0' and '1'

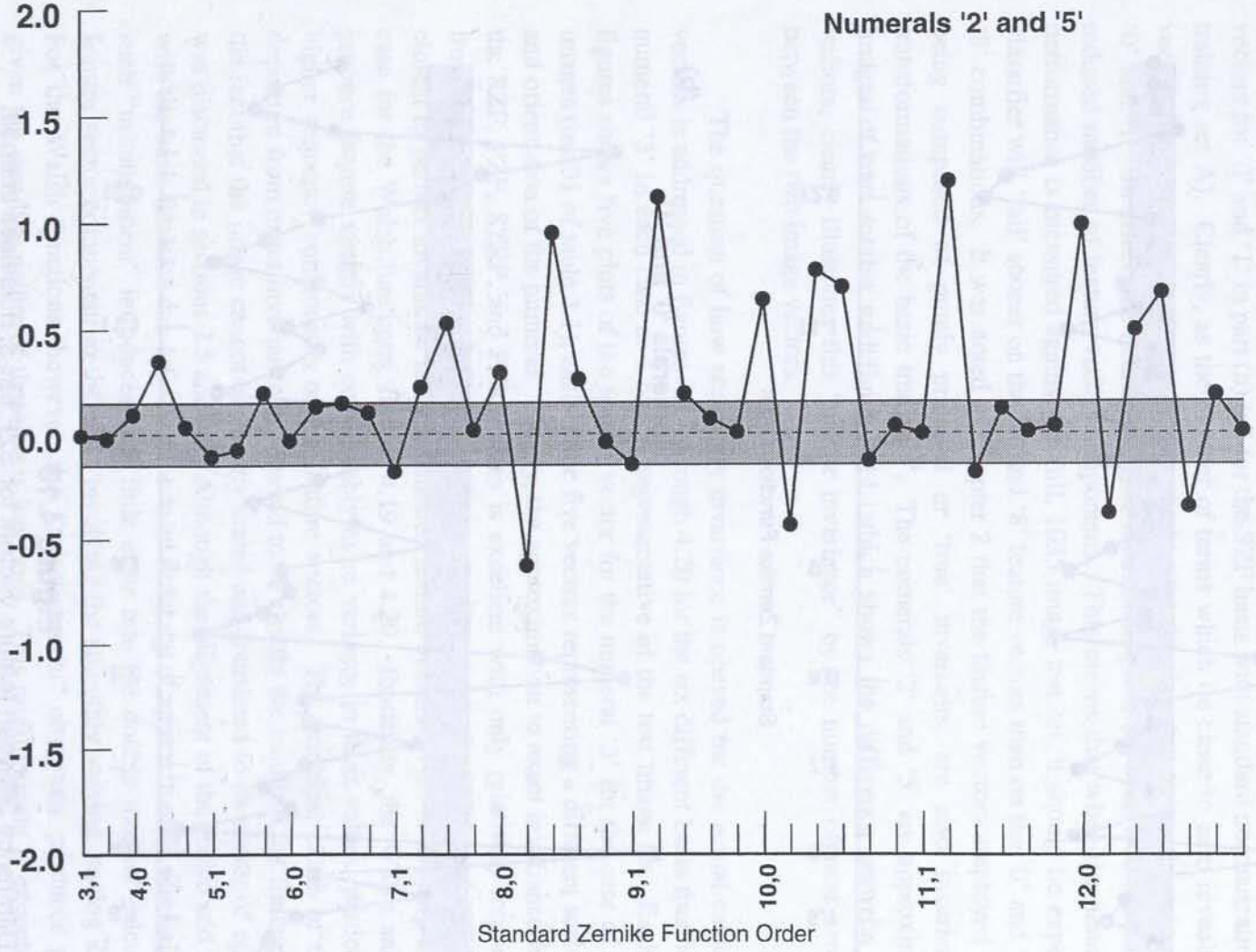


Figure 4.14

Difference between feature vectors for numerals '2' and '5', noiseless images, SZF basis, standard normalization on Training Set A.

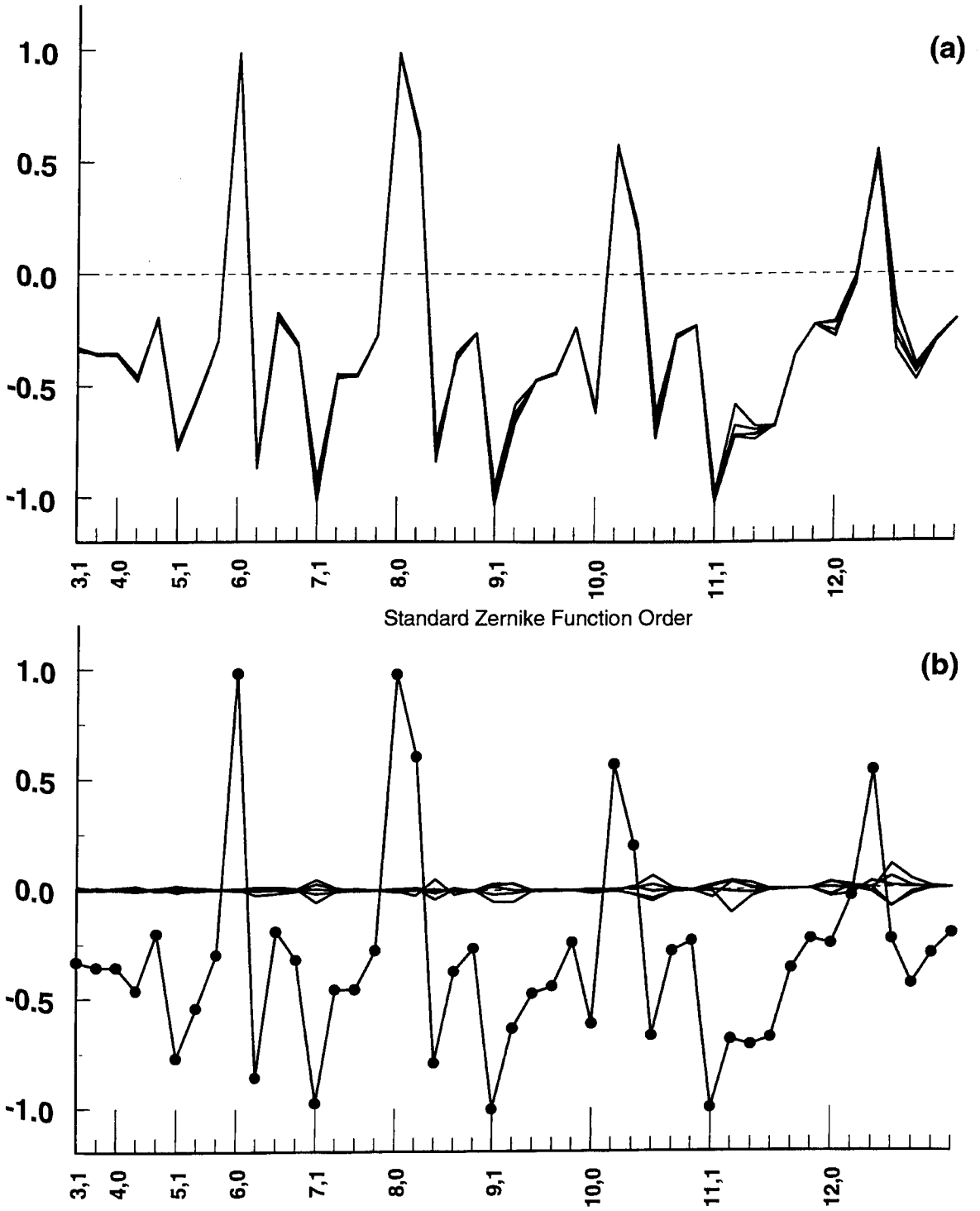


Figure 4.15

Variance of the feature vectors for the SZF basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

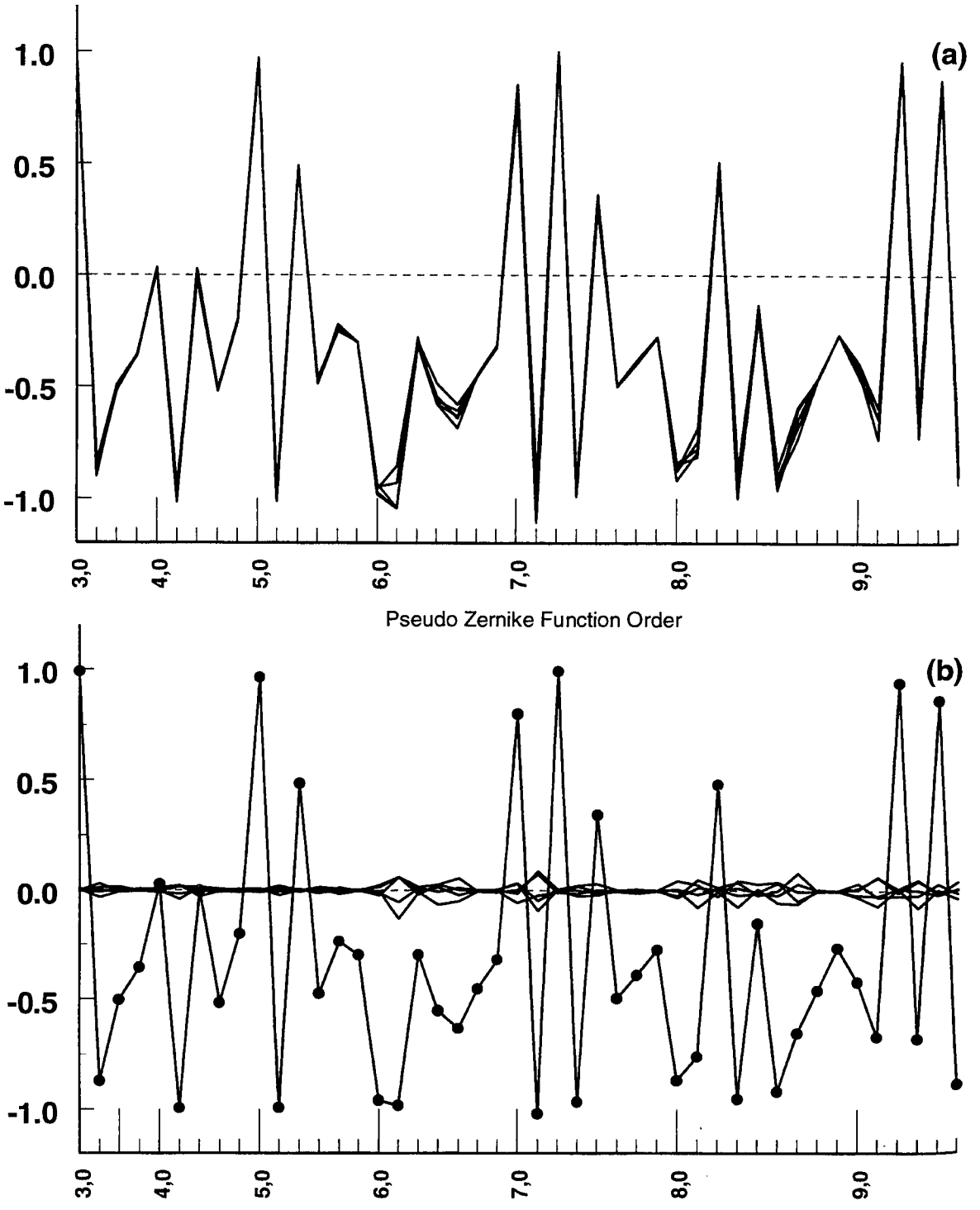


Figure 4.16

Variance of the feature vectors for the PZF basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

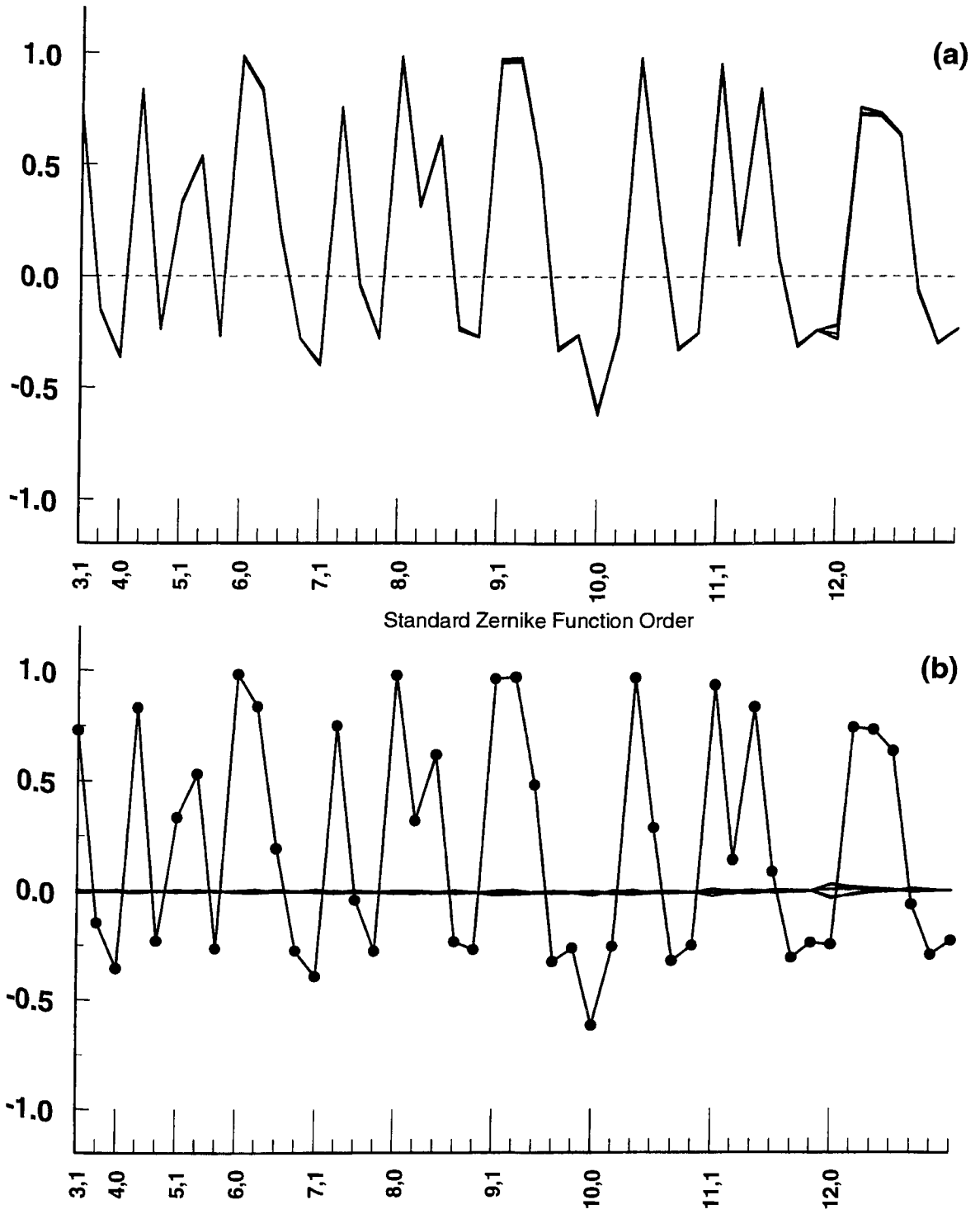


Figure 4.17

Variance of the feature vectors for the SZRP basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

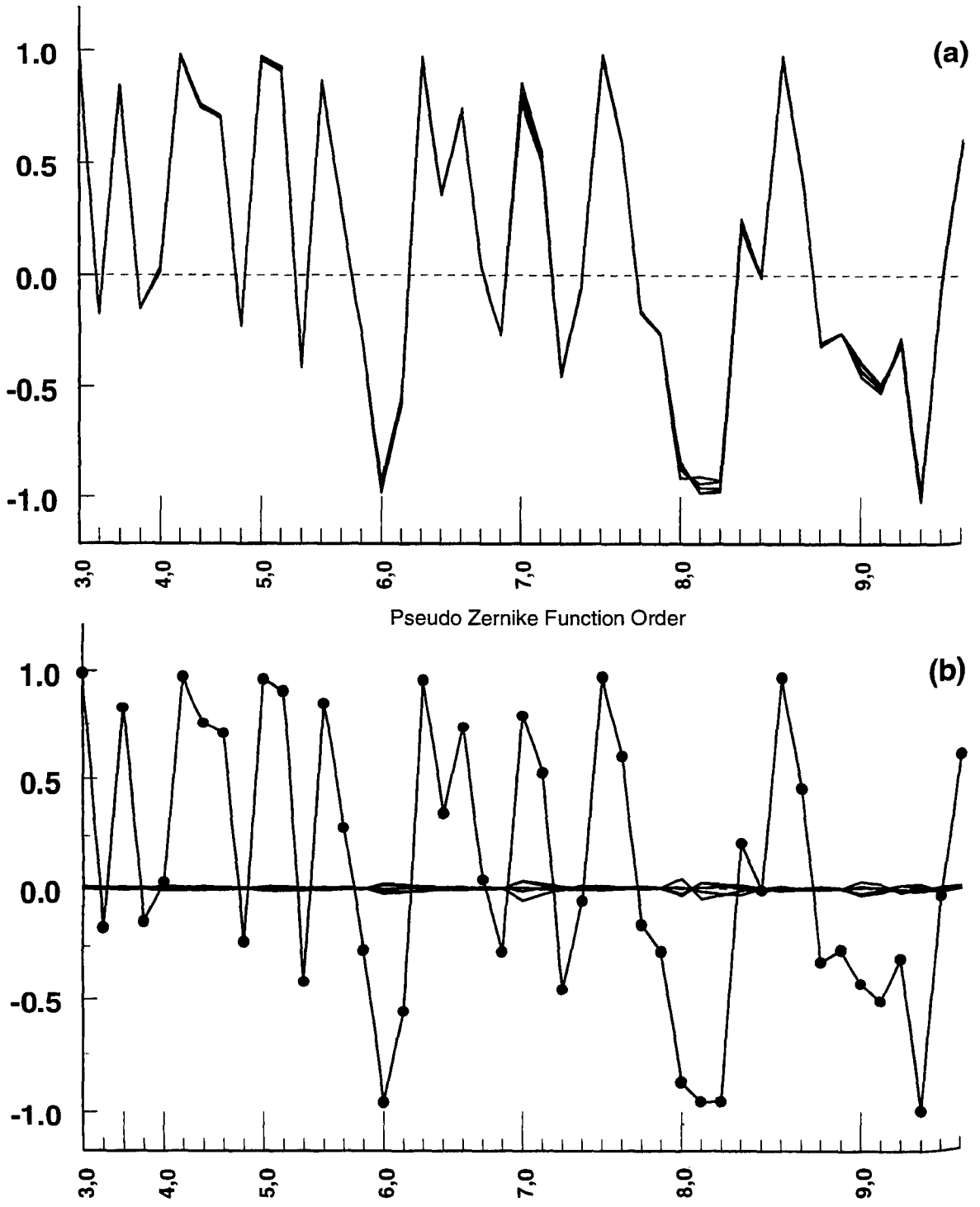


Figure 4.18

Variance of the feature vectors for the PZRP basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

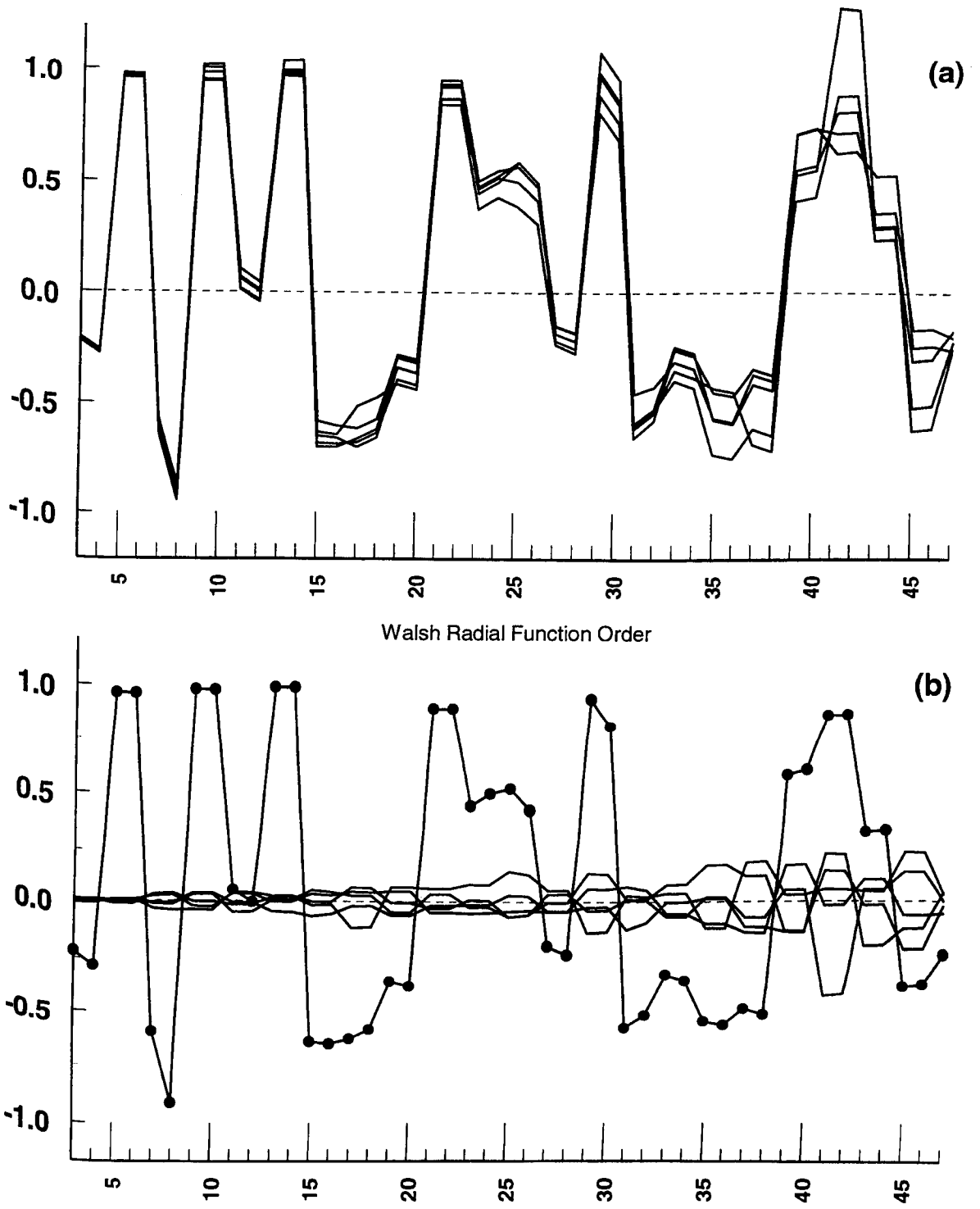


Figure 4.19

Variance of the feature vectors for the WRF basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

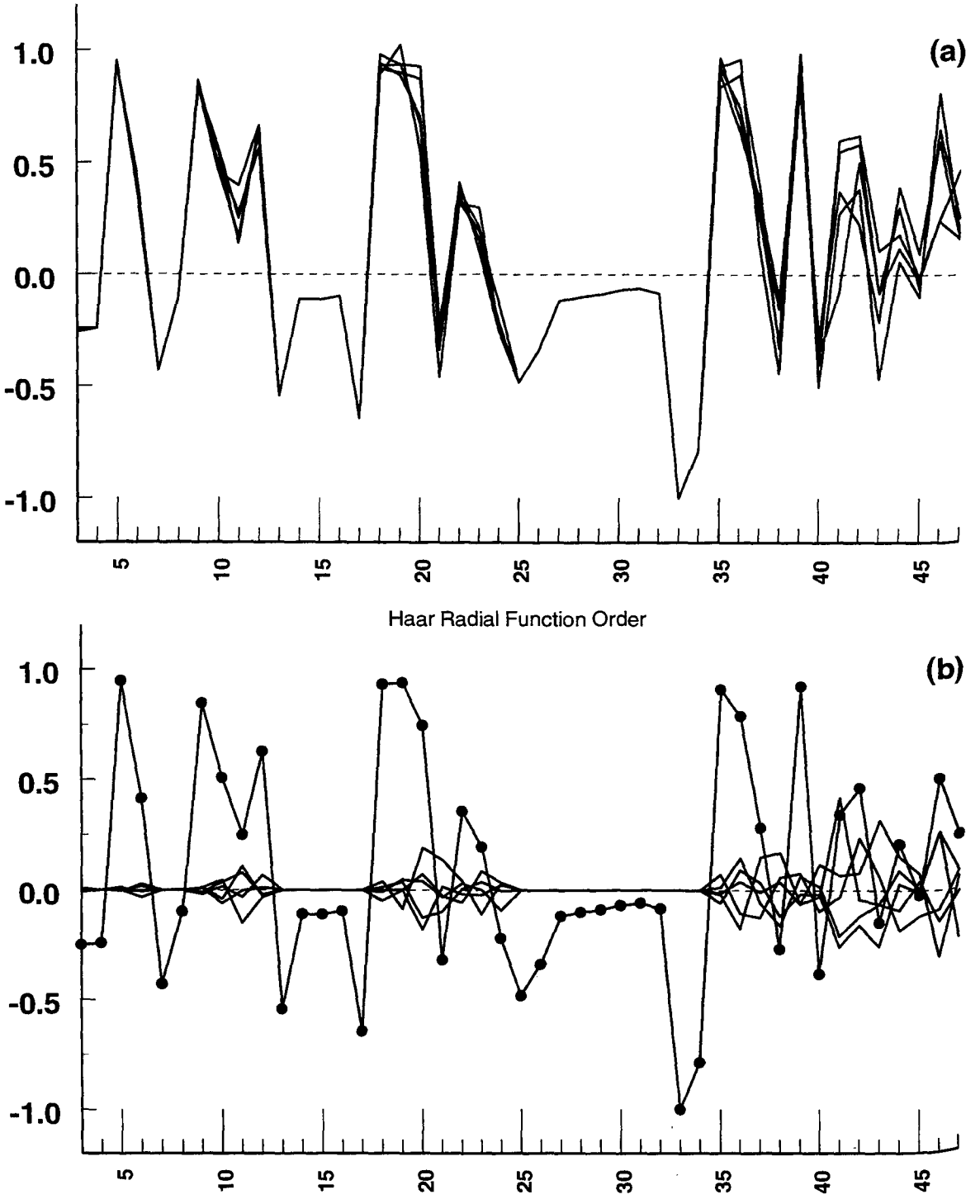


Figure 4.20

Variance of the feature vectors for the HRF basis, noiseless images, for standard normalization on Training Set A.

(a) Plot of 5 feature vectors for 5 different examples of the numeral '3'

(b) The mean of the 5 feature vectors of (a) plus their deviations from the mean

by the observation that the variation in the feature vectors for both types of Walsh functions is considerably greater at higher orders than at lower orders.

In Chapter 3, four different normalization schemes were described, namely standard normalization (SN), positive normalization (PN), bipolar normalization (BN), and differential normalization (DN). These schemes produce very different looking feature vectors as illustrated in figure 4.21 for the case of noiseless test images, the PZF basis, and, in each case, using the noiseless training set A for the normalization parameters. The question of which normalization scheme resulted in the best neural network performance will be dealt with in section 4.4 of this chapter. Note that, in figure 4.21, the DN feature vector is shown over the 45 PZF components even though its dimension is only 44 and, strictly speaking, the x-axis for DN vectors would be a simple running index and not the PZF $\{n,m\}$ indices. To facilitate the comparison to the SN vector, the DN example simply 'adds' a zero first value for purposes of plotting so that the actual 44 values for DN run from $\{3,1\}$ to $\{9,5\}$ in figure 4.21.

The feature vectors, as expected, change as the SNR of the images is decreased. The severity of change, however, is strongly dependent upon the basis functions used in the extraction of the feature vector. Figures 4.22 through 4.27 illustrate the degradation of the feature vector with decreasing SNR for the SZF, PZF, SZRP, PZRP, WRF, and HRF bases, respectively, for the case of standard normalization derived from the noiseless training set. In each case, the feature vectors corresponding to SNR's of ∞, 25, 10, and 5 db are shown. Quite clearly, the degradation of the feature vectors is the least drastic for the SZF and PZF bases. It is considerably worsened for the WRF base but clearly the worst of all for both the SZRP and PZRP cases. In the latter cases, the addition of even very moderate amounts of noise to the image produces relatively major distortions in the feature vectors. Figure 4.28 shows the difference between the normalized feature vectors for the SZRP basis for the numeral '1' and the cases of noiseless and SNR = 8 db images. This figure explicitly reveals that comparatively little distortion of the feature vector with noise occurs for those components for which the secondary index m is small but rises quickly as m increases towards $m = n$. This correlates exactly with eq. (2.19) (and eq. (2.33) for PZRP) giving the measure of the SZRP basis function as a function of m and thus reinforces the previously stated conclusion that image noise will most severely affect those feature vectors which are derived from basis functions with nonzero measure. In the present example, the measure of the SZRP (and PZRP) basis functions is small (or zero) only for $m \approx 0$ (or $m = 0$) and increases to a maximum value when $m = n$.

Figure 4.27 for the HRF basis reveals a behavior quite different from that for the other five basis functions employed insofar as the degradation of the feature vector with added noise is radically different for different components of the HRF vector. Examination of the feature vector degradation with noise for the HRF basis for other numerals reveals a consistent

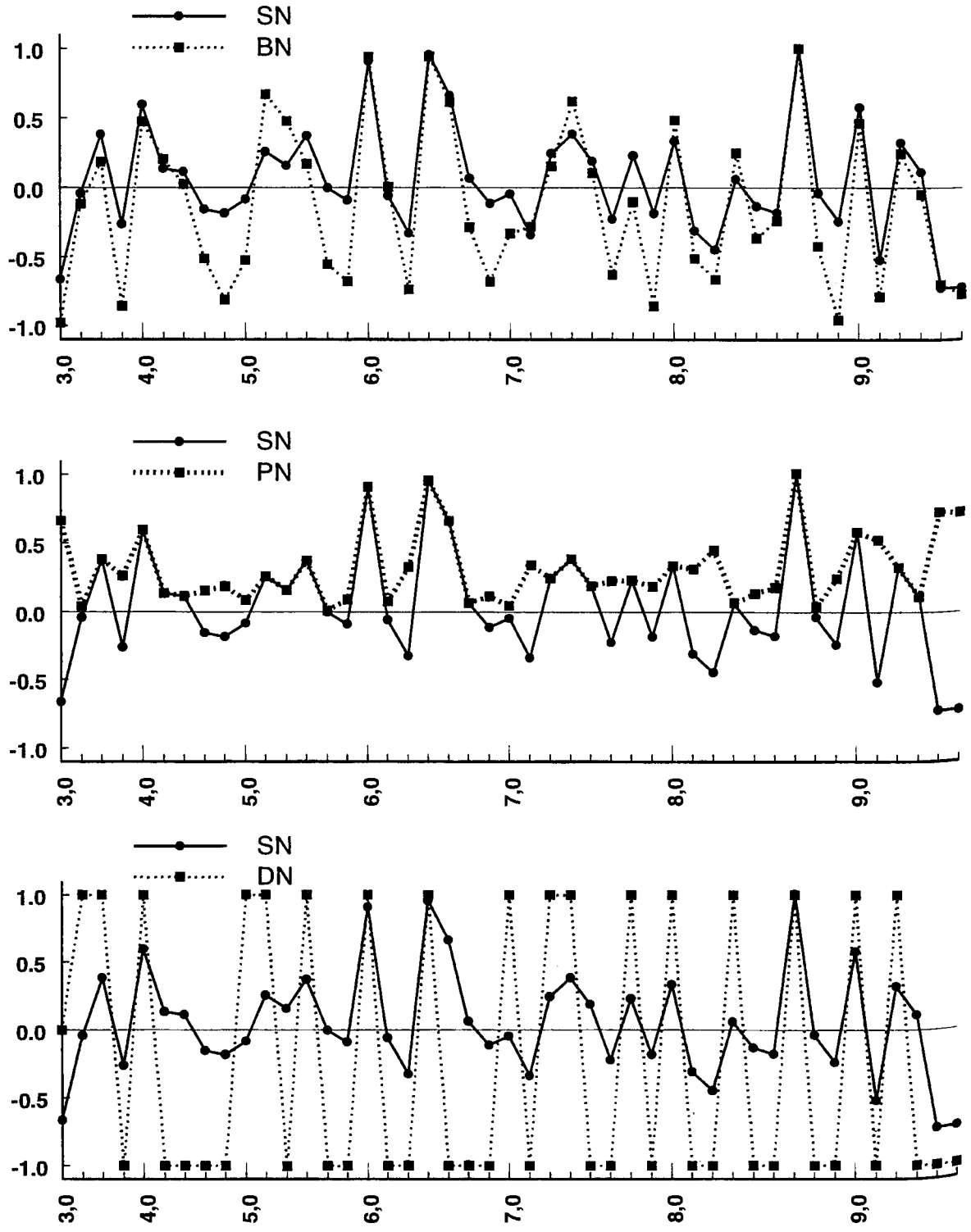


Figure 4.21

Illustration of the four types of feature vector normalization for the noiseless numeral '2', PZF basis, normalized on Training Set A.

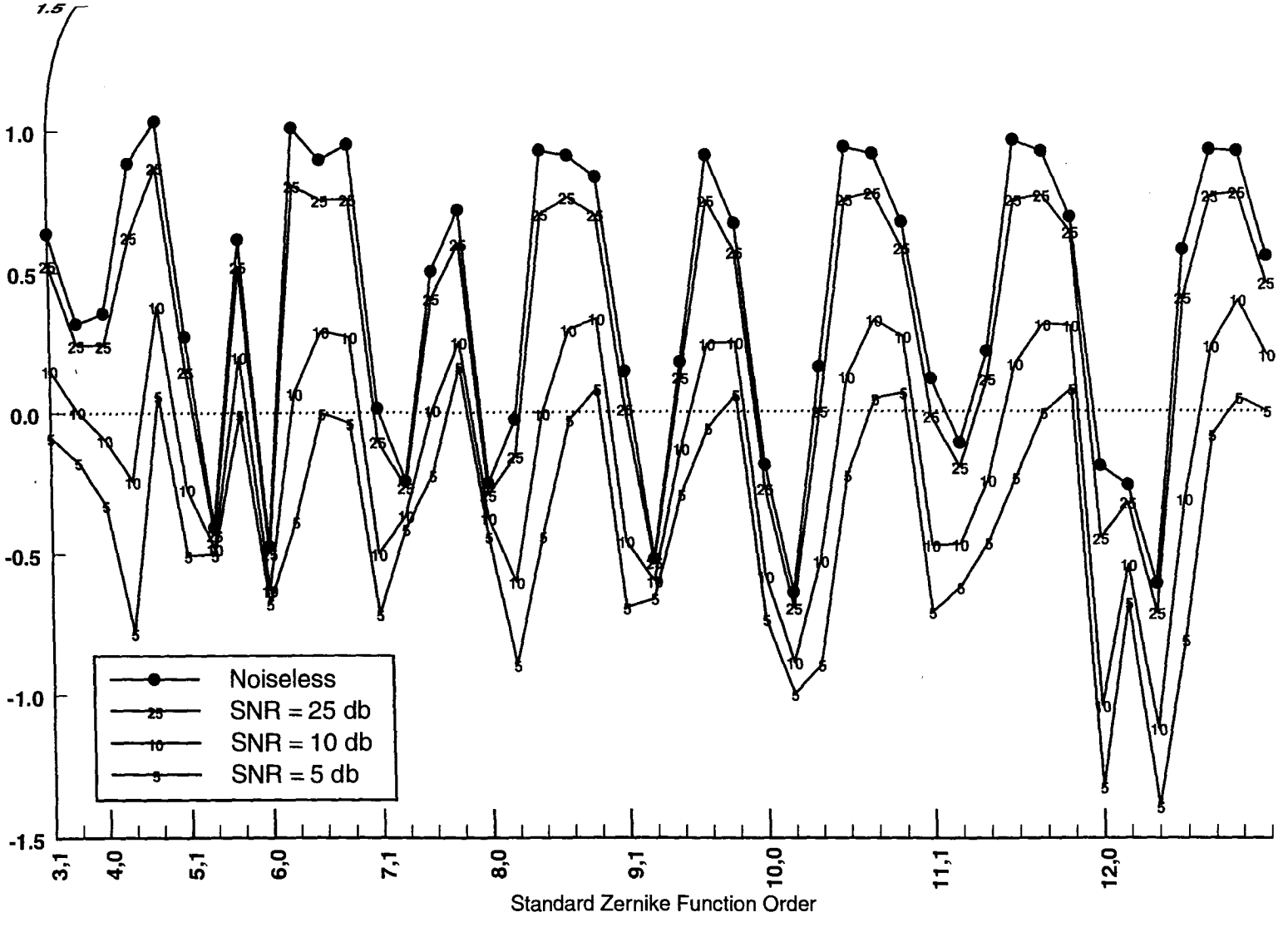


Figure 4.22

Degradation of feature vector with decreasing SNR for SZF basis, numeral '1', standard normalization on Training Set A.

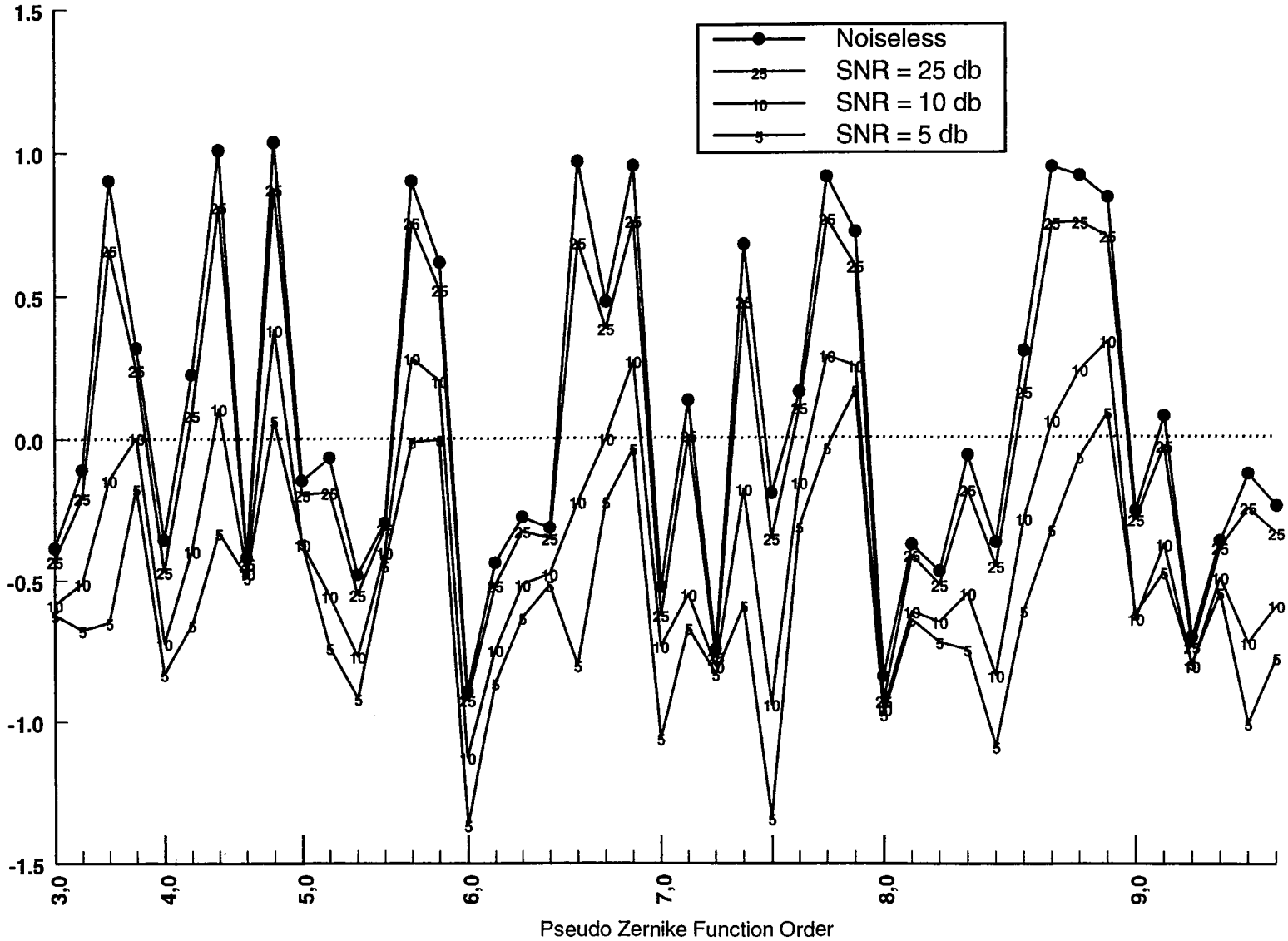


Figure 4.23

Degradation of feature vector with decreasing SNR for PZF basis, numeral '1', standard normalization on Training Set A.

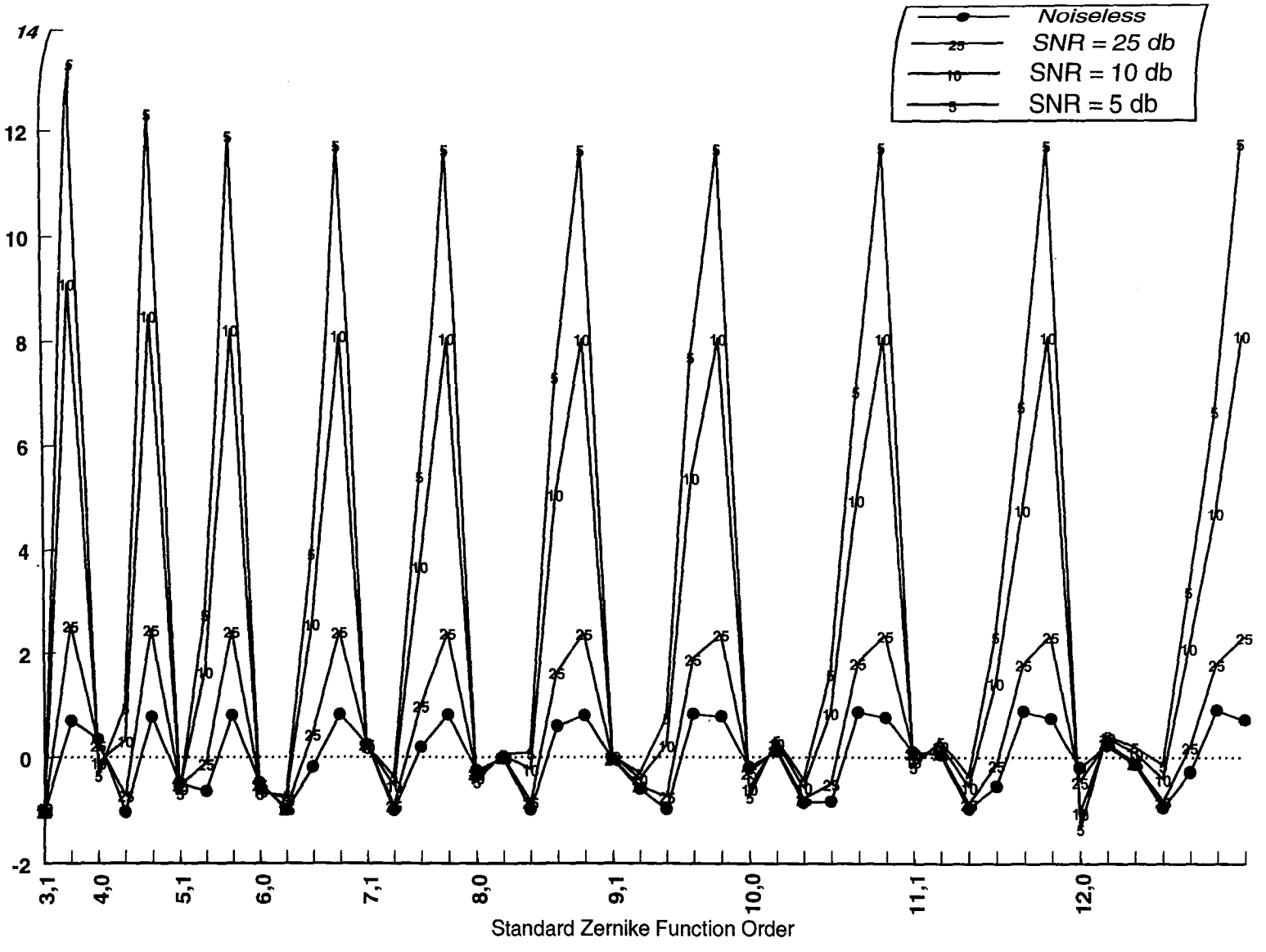


Figure 4.24

Degradation of feature vector with decreasing SNR for SZRP basis, numeral '1', standard normalization on Training Set A.

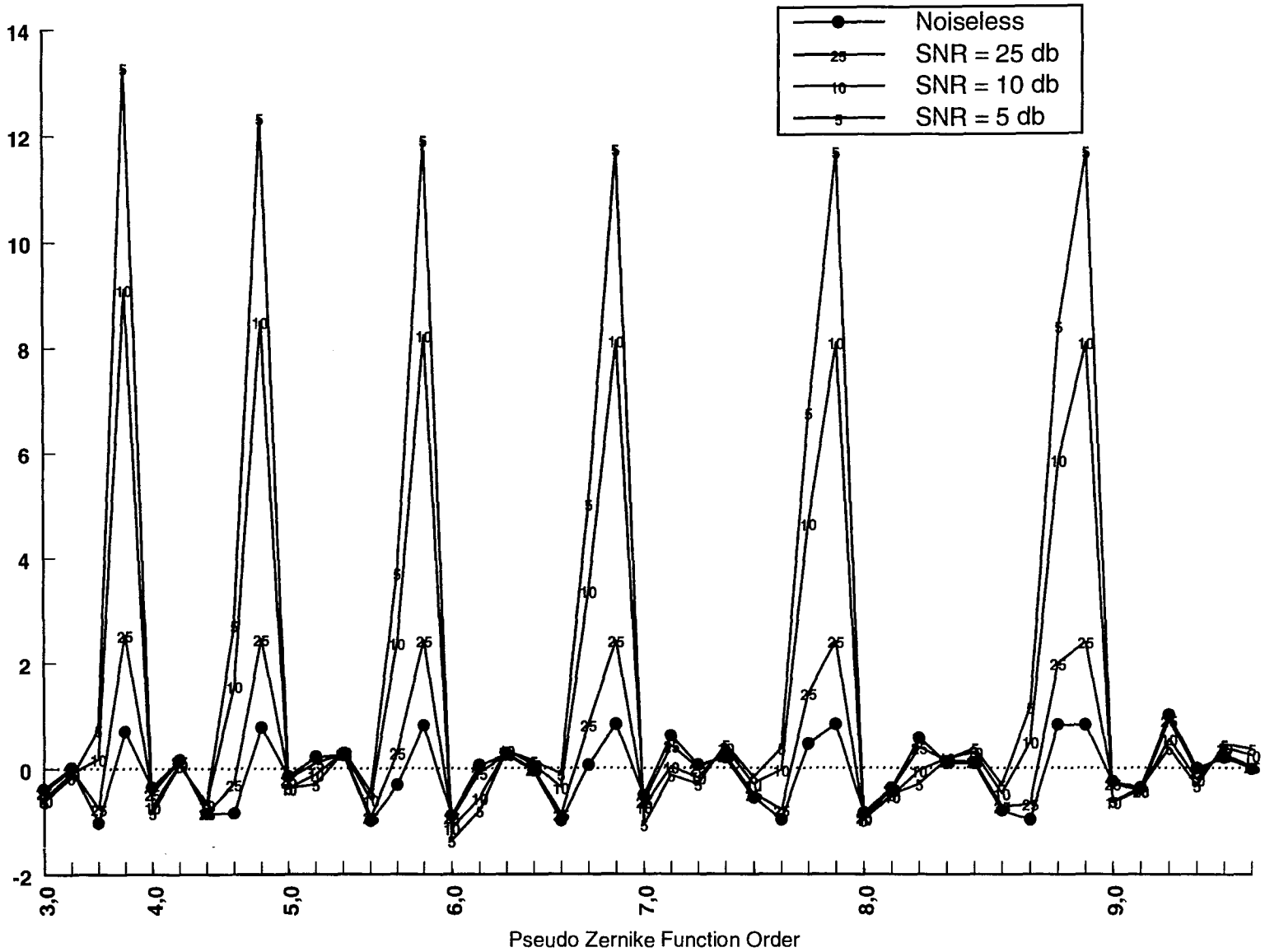


Figure 4.25

Degradation of feature vector with decreasing SNR for PZRP basis, numeral '1', standard normalization on Training Set A.

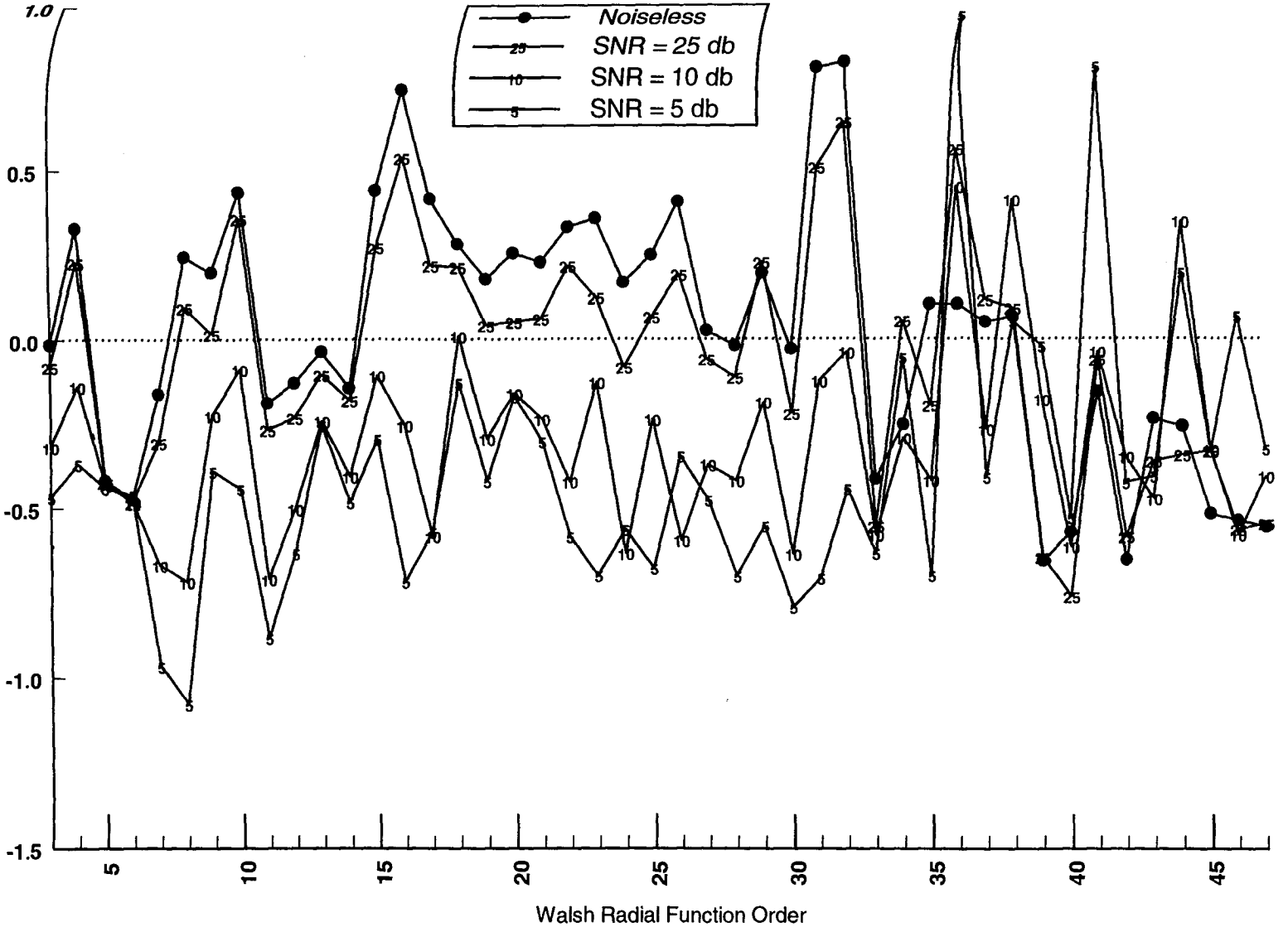


Figure 4.26

Degradation of feature vector with decreasing SNR for WRF basis, numeral '1', standard normalization on Training Set A.

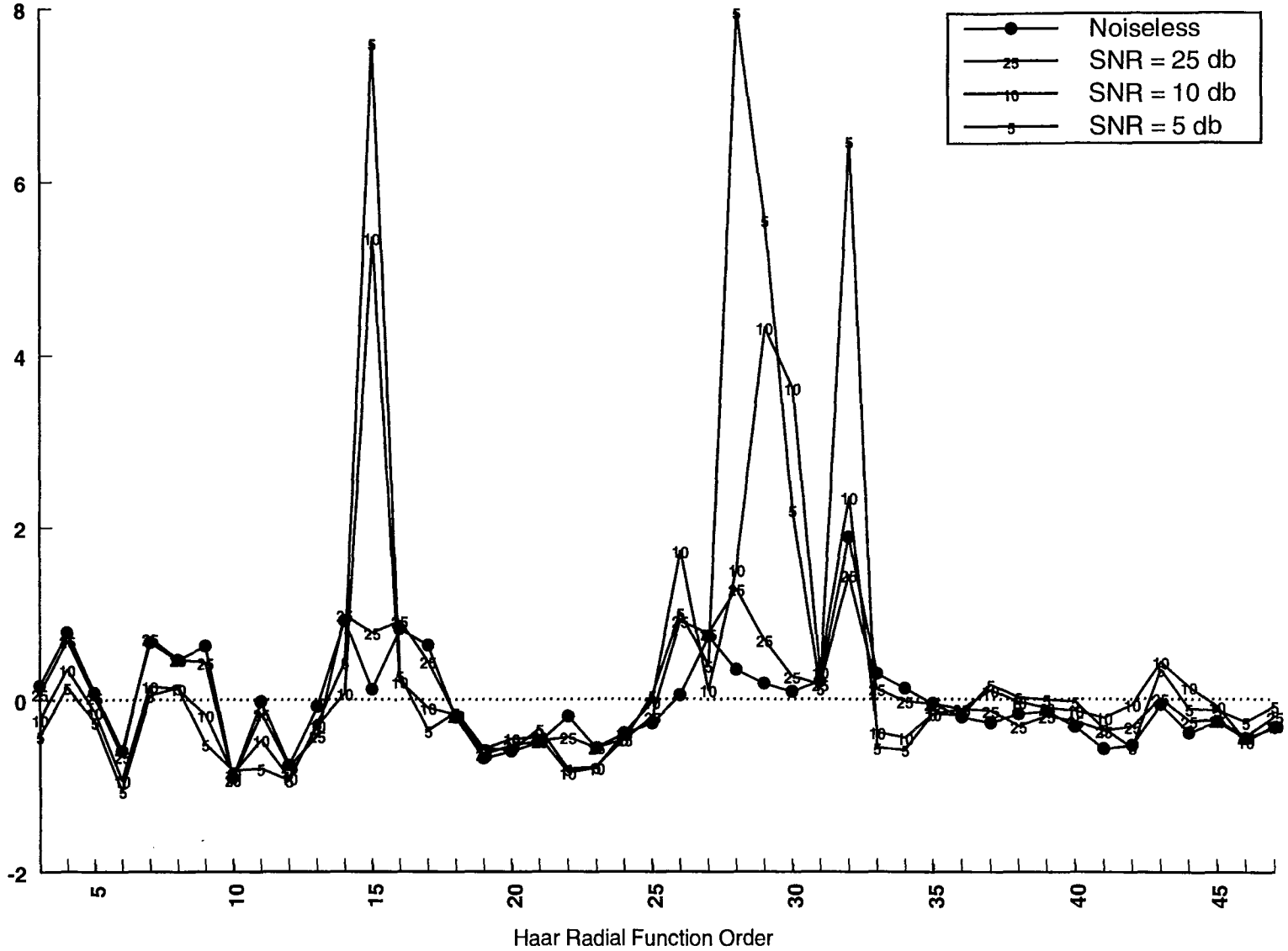


Figure 4.27

Degradation of feature vector with decreasing SNR for HRF basis, numeral '1', standard normalization on Training Set A.

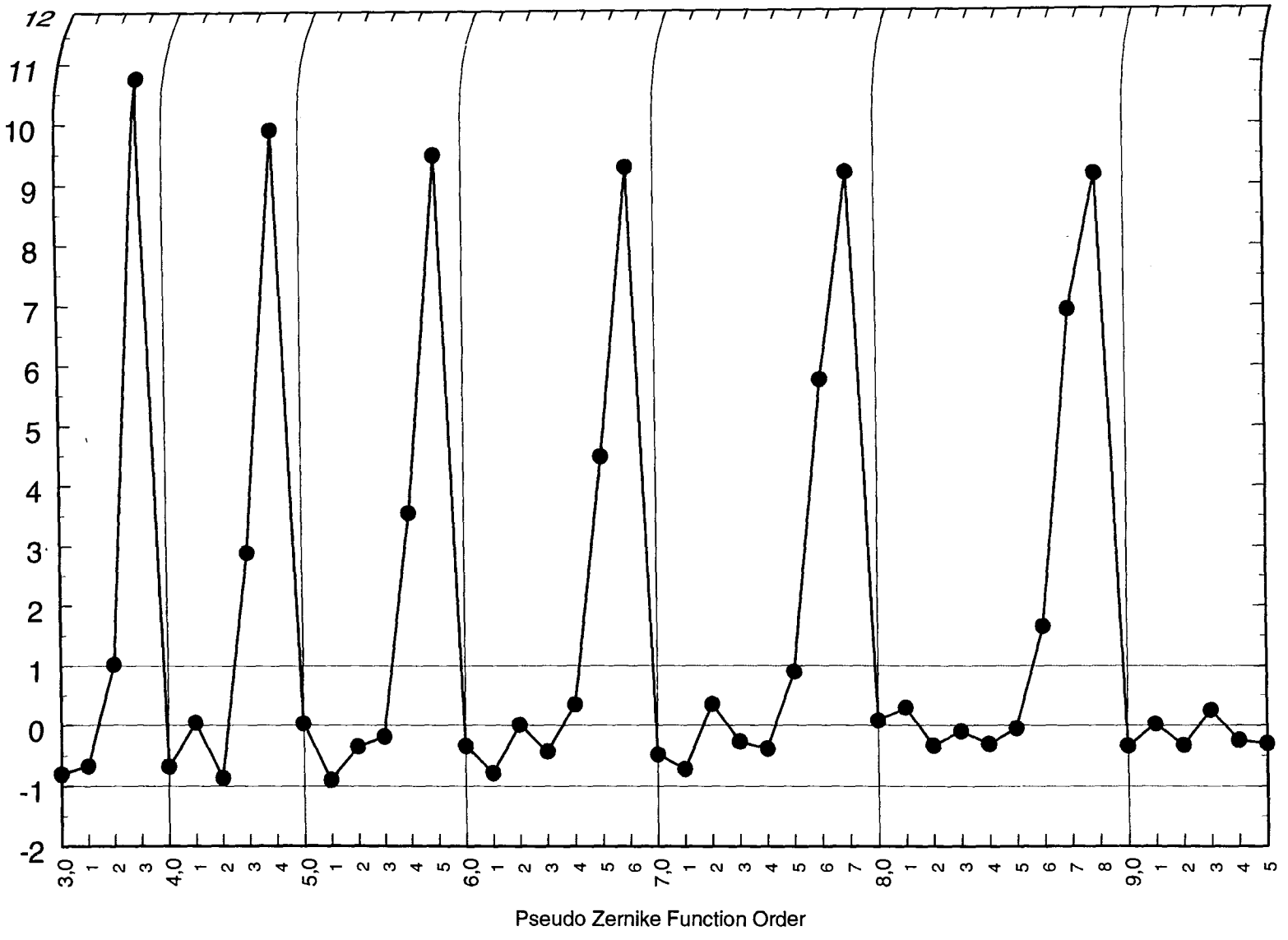


Figure 4.28

Difference between feature vectors of noiseless and SNR 8 db images for numeral '4', PZRP basis, standard normalization on Training Set A

behavior essentially identical to that exhibited in figure 4.27. Note that certain of the HRF-based vector components are almost exactly constant as the SNR is decreased while a small subset of the feature vector components exhibit large changes. This somewhat anomalous behavior is directly traceable to the localized nature of the HRF functions (all of the other bases are global functions). For example, the component for index $n = 31$ is one of those HRF-based components which is essentially unchanged by the addition of noise. Referring back to the illustrations and definition of the HRF functions in chapter 2, it may be seen that the $n = 31$ HRF has a very narrow, non-zero portion which is located at $r \approx 1$ where either very few or no image object pixels can be found. Thus, since this very localized function integrates to zero (eq. (2.53)), the corresponding feature vector component can be expected to remain approximately constant as the SNR is decreased. This behavior should be contrasted with the behavior for the adjoining component at $n = 32$. The HRF for $n = 32$ consists of a much broader (but still localized) portion which is located at $r \approx 0$ and it should be expected that the corresponding feature vector component will exhibit greater change as the SNR is decreased. Although the marked changes for $n = 29$ and $n = 30$ are less obvious, the behavior at those index values may also be traced to the specific form of the HRF function for those indices.

Each of the graphs shown thus far depicts the "normal" view, i.e., the 45 scalar components of a feature vector are shown for a single image as a function of the order of the basis function from which the components are derived. However, during the course of examining the correlation between feature vector components, it was discovered that there existed strong degrees of correlation between certain components of the WRF and HRF feature vectors. This follows from the piecewise continuous, rectangular waveform nature of these functions. The difference between any two WRF (and, of course, the HRF) will be a localized function. For certain pairs of WRF (and HRF) this difference will be non-zero only for values of $r \approx 1$ (such pairs can be seen by simply examining the plots such as those of figure 2.15). Since the images are irradiance normalized to $\beta = 8192$, i.e., normalized to a constant 'area', it follows that some of the images will cover the range $0 \leq r \leq 1$ more than others (the '1', e.g., more than the '0'). In general, all of the images (in the noiseless case) will have some "white space" in the image plane for $r \approx 1$. The feature vector components corresponding to these pairs of WRF (or HRF) components may thus be expected to be approximately equal for *all* images. This characteristic is illustrated by figure 4.29(a) which shows the $n = 5$ and $n = 6$ components of the WRF based features over the entire set of 207 averaged feature vectors (each vector an average of 5 images). Clearly, as figure 4.29(b) shows, these two components of the WRF based feature vector are highly correlated throughout the entire test set, i.e., these components are essentially redundant. Note that even though the WRF are independent and orthogonal, the feature components formed from them *can* be correlated and hence, to some degree, redundant. This means that, in the cases of the WRF and HRF bases, the "effective dimension" of the feature vectors employed will be less

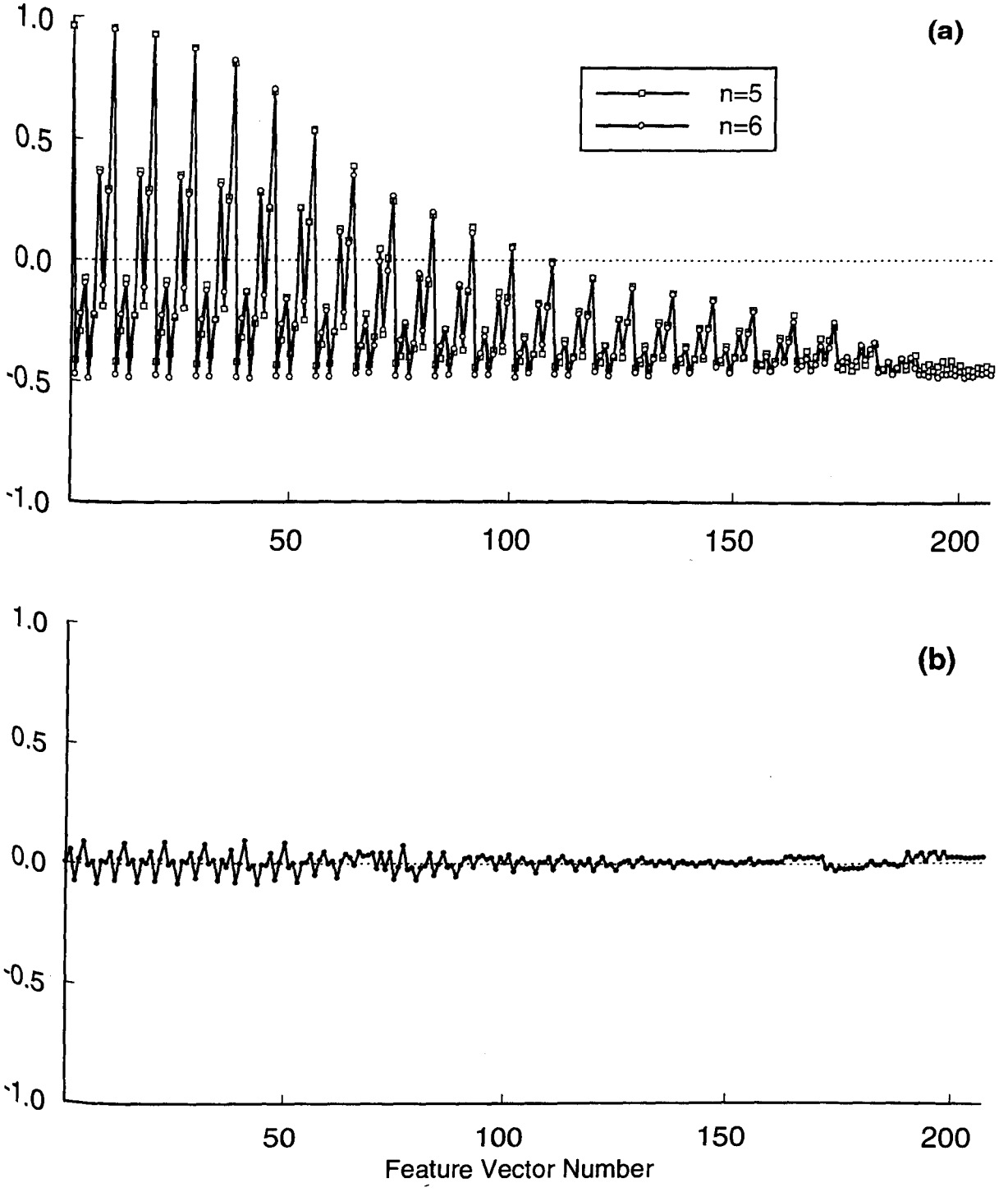


Figure 4.29

Correlation between WRF feature vector components, noiseless images, standard normalization on Training Set A.
(a) Feature vector components for WRF with n = 5 and n = 6
(b) Difference of feature vectors given in (a)

than the nominal 45 as stated. Indeed, in the present work, it is estimated that this "correlation effect" among the WRF and HRF components reduces the effective feature vector dimension by approximately 30%. This compounds further the difficulties with Walsh-type bases discussed in section 2.5 where it was shown that the practical limits on the dimensions of their feature vectors for a given grid size were severe. Note that no such behavior or correlation was observed for any of the (analog-valued) Zernike functions.

A final note to this section on feature vectors concerns the ability of a neural classifier to learn distinguishing features which may not be readily observable by a mere inspection of the feature vectors themselves along the lines of the figures presented thus far in this chapter. A properly trained, multilayer perceptron, neural network classifier, trained with the backpropagation algorithm, is expected to learn not only the gross characteristics, examples of which have been presented above, but to also learn other, much more subtle differences between the feature vectors which should enable the classifier to provide much better performance with decreasing SNR than the foregoing, simple observations alone would imply. However, it is important that the reader keep in mind that, even though a neural network classifier may come impressively close to matching human performance in this one specific task (which is the case in the present work), the classification system described in this report is an extremely myopic one, limited strictly to an ability to classify *only* those images presented to it during the training process. Human capabilities are far more extensive in that general scenes which may contain a plethora of distinct objects are easily recognized and accurately classified in a manner invariant to a broad range of changes or transformations to such objects.

4.2 : Classifier Performance and the Number of Hidden Neurons

The three layer perceptron used throughout this work had the dimension of its input layer determined by the feature vector dimension (45 in all cases, 44 for the cases of differential normalization DN) and the dimension of its output layer set by the number of image classes (9 for all cases). The dimension of the important, hidden layer of the neural network, however, was left to be determined experimentally. While there does exist several "rules of thumb" to estimate this number, these estimates yielded a wide range of possible values.

To determine an optimum value for the number of hidden neurons, the classifier performance was measured for hidden neuron sizes of 6, 12, 18, 24, 30, 36, and 42 for the twelve cases of the PZF and SZF bases, training set A, 45,000 training iterations, and an epoch size of 16. The results, shown in figure 4.30, indicate a rapid improvement in performance up to 24, a subsequent decrease, and a second increase for the largest hidden

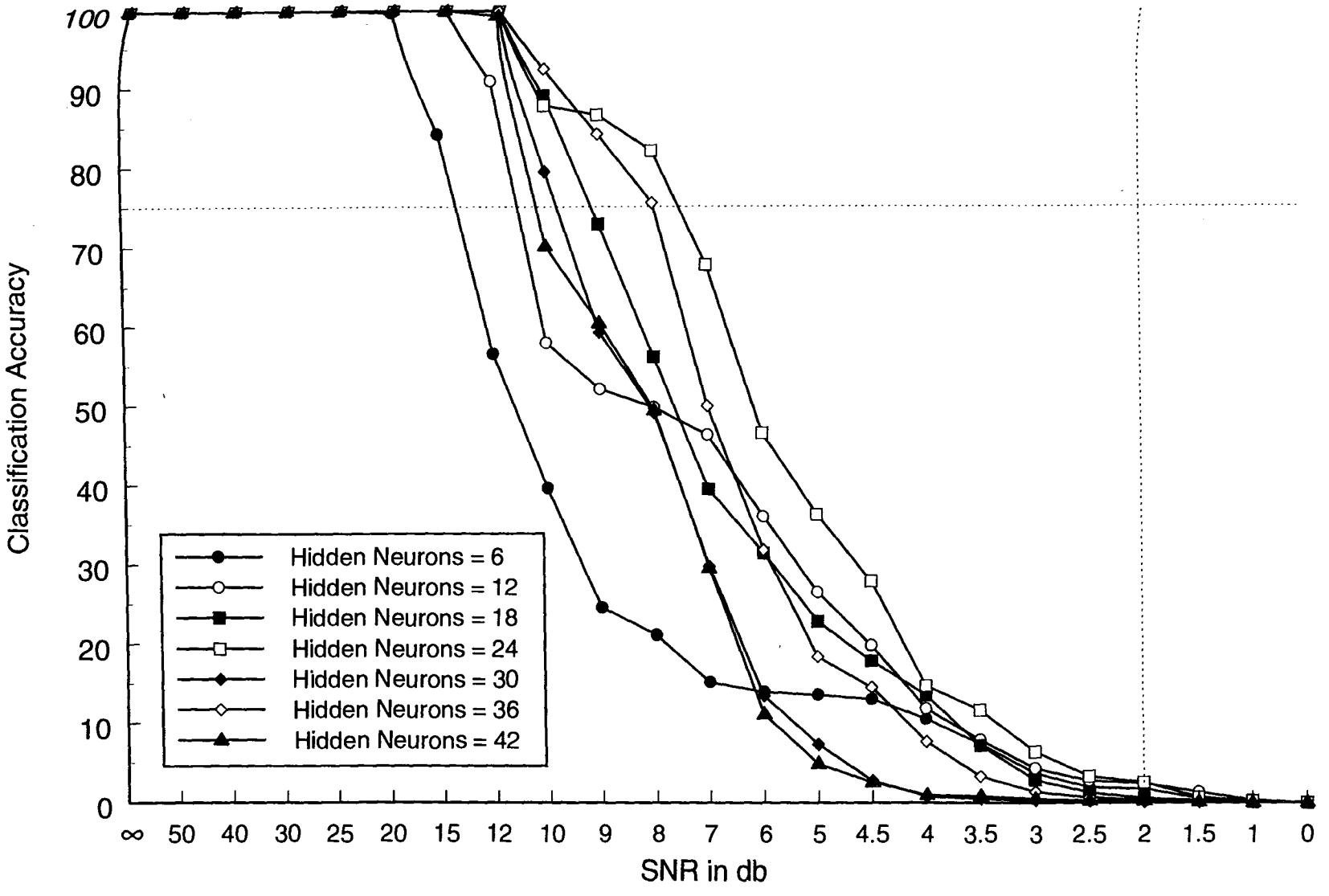


Figure 4.30

Classifier performance as a function of the number of hidden neurons.

neuron numbers. However, an excessive number of hidden neurons produces a behavior not dissimilar to the "overtraining phenomena" discussed in Chapter 3. For this reason, it was determined that a hidden neuron layer dimension of 24 represented the best compromise and most accurately trained classifier. This value was subsequently used for all of the performance measurements.

In the remainder of this chapter, several results such as those illustrated in figure 4.30 will be presented which measure the performance of the neural-network-based classification system. It is, therefore, imperative that it be made clear exactly what a "classification performance measurement" represents. In each case, the classification accuracy of the neural network is measured over a test set consisting of 1035 images which is made up from the 23 individual test subsets (of 45 image each) as described in section 3.2. Each of the subsets has a definite SNR ranging from '∞' (noiseless) to 0 db (pure noise) as given in table 3.1. The classification accuracy for each subset is measured separately and plotted versus its SNR, i.e., the performance plots as in figure 4.30 and in the remainder of this chapter plot these 23 points as the measurement of classifier performance as a function of SNR of the test data. The x-axis is somewhat arbitrarily divided into 23 equal intervals corresponding to the SNR's of the test subsets of table 3.1. This scale is essentially immaterial in the sense that the crucial characteristic which is illustrated is that region of SNR over which the classification falls from 100% to values less than 50%, i.e., it is the relative position of this failing of the classifier among the various results which is of importance. The actual values for the classification accuracy as shown in these figures are arrived at using the "rating" system described in section 3.7 in which the 'hit' or 'miss' ratios are weighted by the sigmoid function of eq. (3.7) and the classification curve is normalized according to eq. (3.9). Note that in each of the "classifier performance" graphs presented in this chapter, lines corresponding to a SNR of 2 db and a classification accuracy of 75% are indicated. These lines are intended to explicitly indicate that SNR ratio below which human capability falls off rapidly and to indicate a classification accuracy level which divides the performance ability as "good" (above 75%) and "poor" (below 75%).

4.3 : Classifier Performance and the Training Epoch Size

The term "epoch", unfortunately, is used with widely differing meanings by the neural network community and, understandably, can easily lead to confusion. It is used in this work to mean the number of training samples presented to the neural network before the network weights are actually updated within the backpropagation routing. An epoch size of 1 results in a dramatically increased training time and considerably more fluctuation in the network error as the training process progresses. An overly large epoch value, on the other hand, while leading to much reduced training times, leads also to trained networks with appreciably larger

errors than those obtained for smaller values of the epoch for the same number of training samples. Clearly, some approximate optimum for this parameter exists and must be determined by trial and error.

To determine this value, the classifier performance was measured for epoch values of 2, 4, 8, 16, 32, and 45 for the case of the PZF basis, training set A, 45,000 training iterations, and 24 hidden neurons. The results are shown in figure 4.31. From this result along with a knowledge of the network's RMS training error upon successful training completion, it was determined that an epoch value of 16 represented the best compromise and most accurately trained classifier. This value was subsequently used for all of the performance measurements.

4.4 : Comparison of Classifier Performance with Normalization Schemes

Measurements of classifier performance were carried out for each of the normalization schemes over each of the twelve combinations of basis function and training sets and were critically compared in all cases. Representative examples are shown in figures 4.32 and 4.33 for the case of the SZF basis using training set A and the PZRP basis using training set B, respectively. Clearly, the performance of the PN scheme is the worst of the lot (this was a consistent finding). Arguing that the use of the sigmoidal transfer function on the hidden layer neurons was inappropriate for the PN scheme, several runs were conducted using a modified neural network which used the tanh transfer function for *both* neural layers. Only a very slight improvement was observed for the PN and the observation that it was the worst of the four remained unchanged. The DN scheme performed better than the PN in all cases but, as figures 4.32 and 4.33 indicate, performed noticeable poorer than either the SN or BN cases.

In truth, the BN normalization scheme typically performed close to and often equal to that measured for the SN scheme for several of the runs. However, an examination of *all* the performance measurements for these two schemes clearly indicated that the SN scheme was consistently better than that of the BN scheme. Thus, in all of the work to follow concerning the measurements of classifier performance versus choice of basis function, the Standard Normalization scheme described in section 3.5 was used exclusively.

4.5 : Variability in Classifier Performance with Random Initialization in the Training Process

The overall classification system comprising the feature vector extraction algorithms and the neural network classifier presents the experimenter with an abundance of variables and parameters which can be tuned to optimize the overall performance of the classifier. In

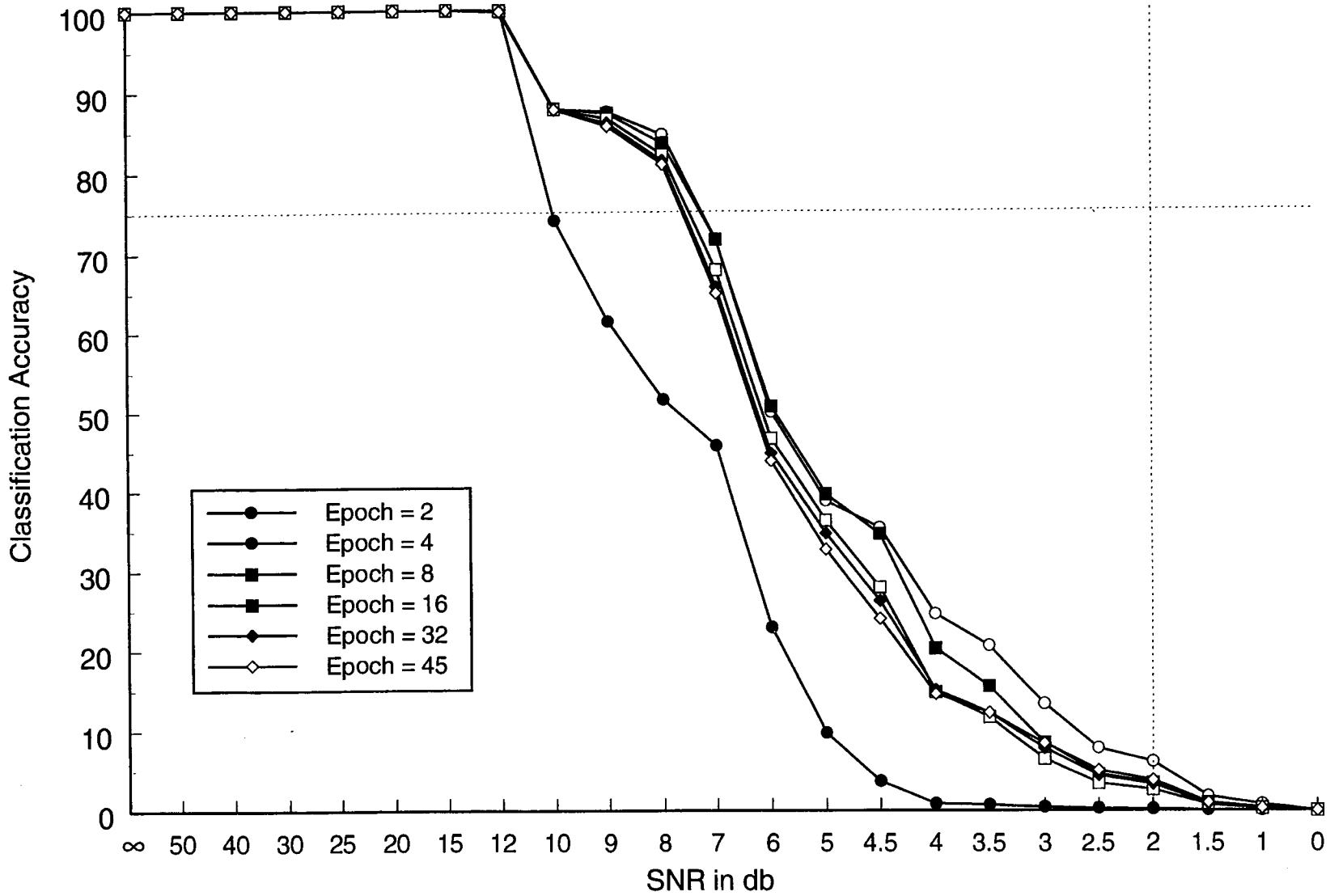


Figure 4.31

Classifier performance as a function of the epoch size

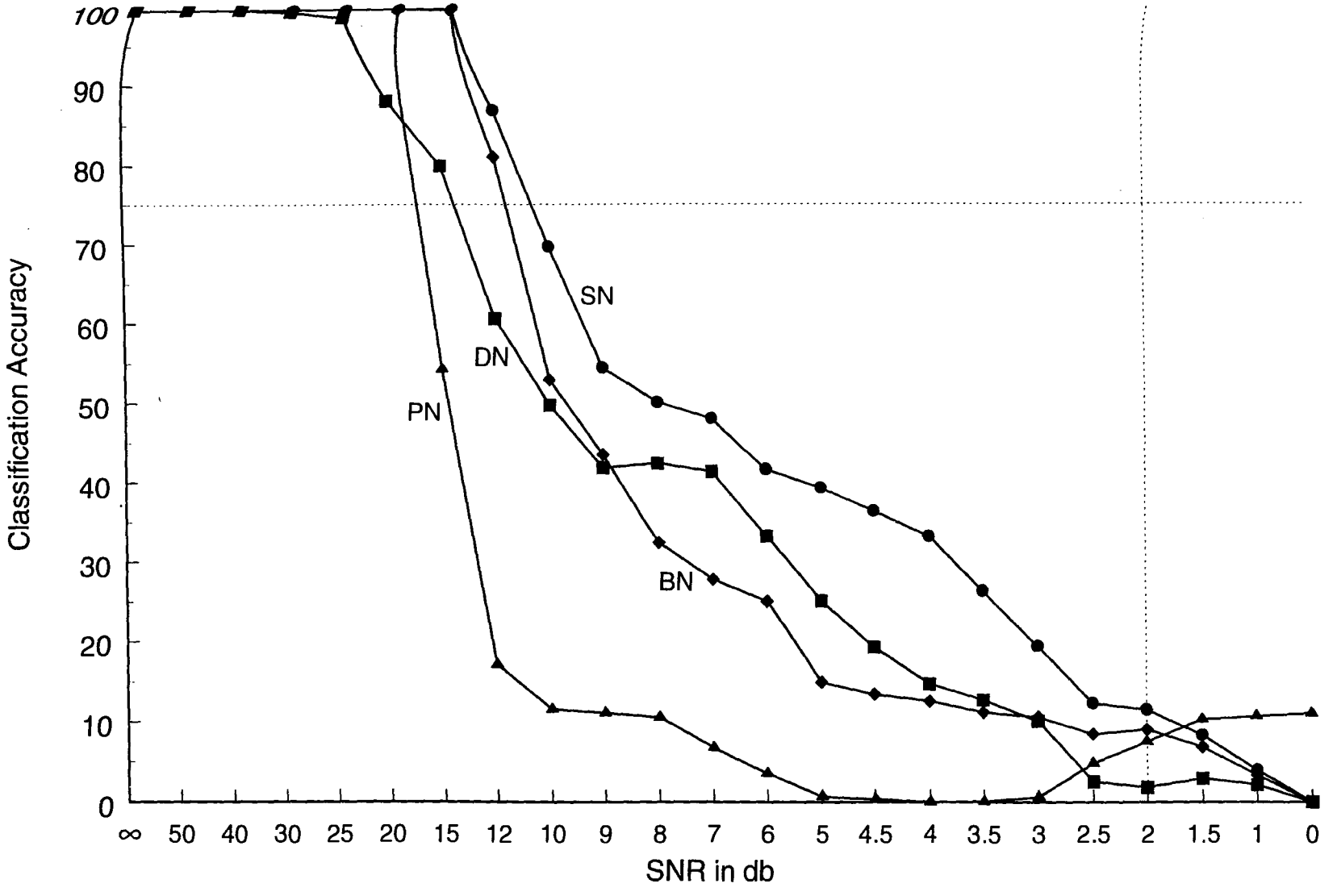


Figure 4.32

Classifier performance for different normalization schemes, SZF, Training Set A.

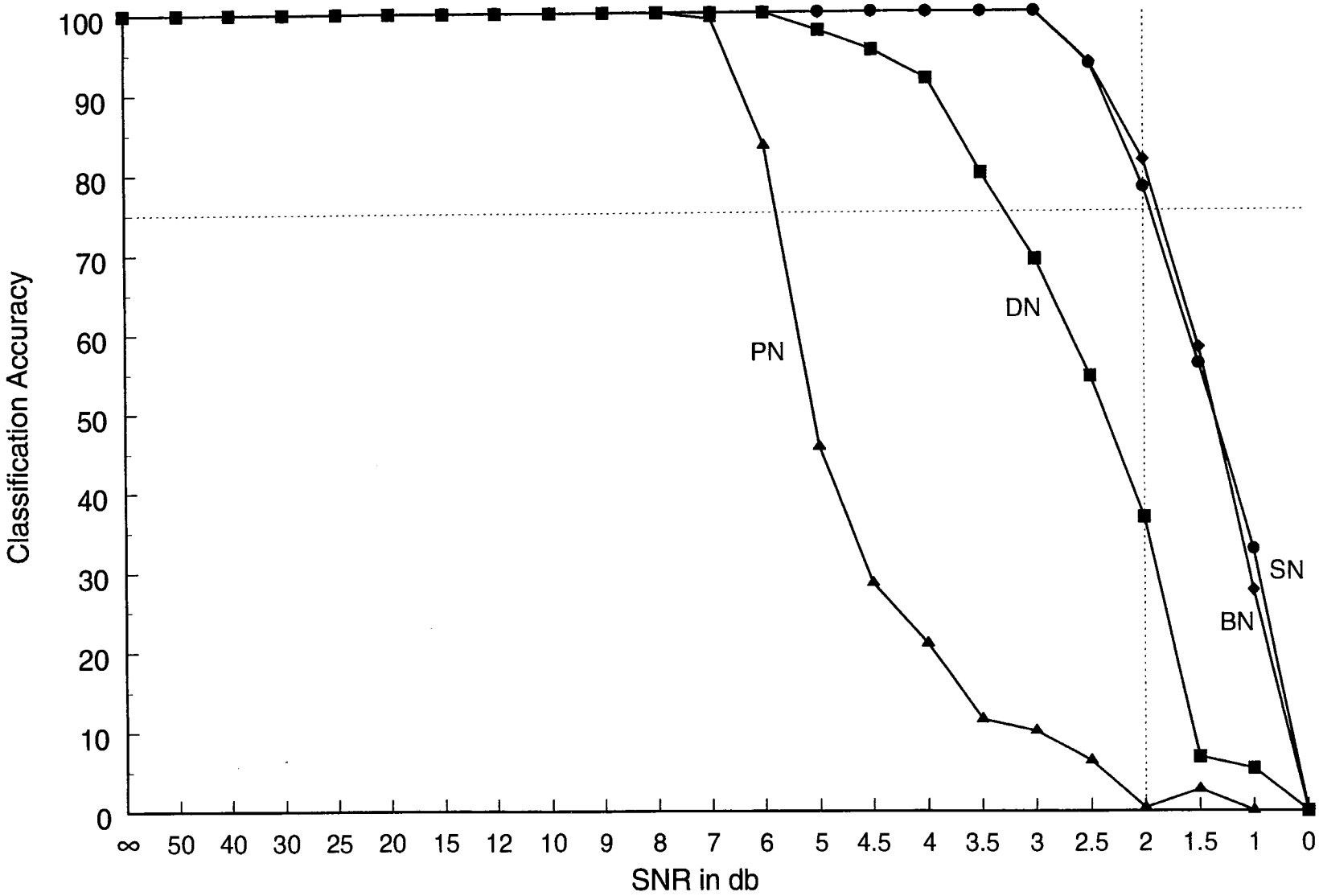


Figure 4.33

Classifier performance for different normalization schemes, PZF, Training Set B.

many respects, a neural network based classification system could be termed a "tweaker's delight". However, it is fundamentally important to always keep in mind that the performance of the neural network is somewhat dependent upon underlying statistical factors which include inherent statistical variations in the composition of the training and test image sets, the nature by which the untrained network is initialized to random weights, and the randomness in the presentation of the training vectors to the network during the backpropagation training phase (keeping in mind that, in the present work, the training vector set is presented 1000 times to the network during the course of training). The latter two factors, in particular, mean that, for a fixed set of training and test data bases, different runs which use different random initializations and/or "shuffle and deal" routines to randomize training vector presentation order will produce differences in the final measurement of the classifier's performance. This "fuzziness" in the performance of the final network is illustrated in figure 4.34 which shows performance measurements for the PZF basis using training set A conducted for eight different choices of the "random seed variable" used by the NeuralWare simulator to both initialize the untrained network weights and to run the "shuffle and deal" algorithm for the presentation order of the training vectors to the network. Clearly, appreciable differences occur in the final measurements and these are due solely to statistical differences in the training process - the training and test sets are identical in each of the results shown in figure 4.34. Note, however, the constancy of the "shoulder" position of these curves; for all eight plots, the classification accuracy begins to fall quickly beyond the 12 db SNR. This result is intended to remind both the experimenter and reader alike that attempts to precisely optimize the neural network classifier represent a somewhat facetious activity since such inherent, underlying statistical dependencies in the training process ensure that only approximate optimization of the overall classification process is truly meaningful.

4.6 : Dependence of Classifier Performance upon Basis Function and Training Sets

This section presents what constitute the chief results of the present work, namely the measurements of the classification performance for each of the six basis functions using both training set A (noiseless) and training set B (mixed noiseless and noisy imagery). In all the results presented here, the environment employed was:

- trilayer perceptron with 45 input neurons, 24 hidden neurons, 9 output neurons, trained using backpropagation, sigmoid transfer function (hidden layer neurons) and tanh transfer function (output layer neurons), network initialization seed value of 257;

- training on 45,000 iterations for training set A (45 vectors) and 90,000 iterations for training set B (90 vectors) using an epoch value of 16 in all cases;

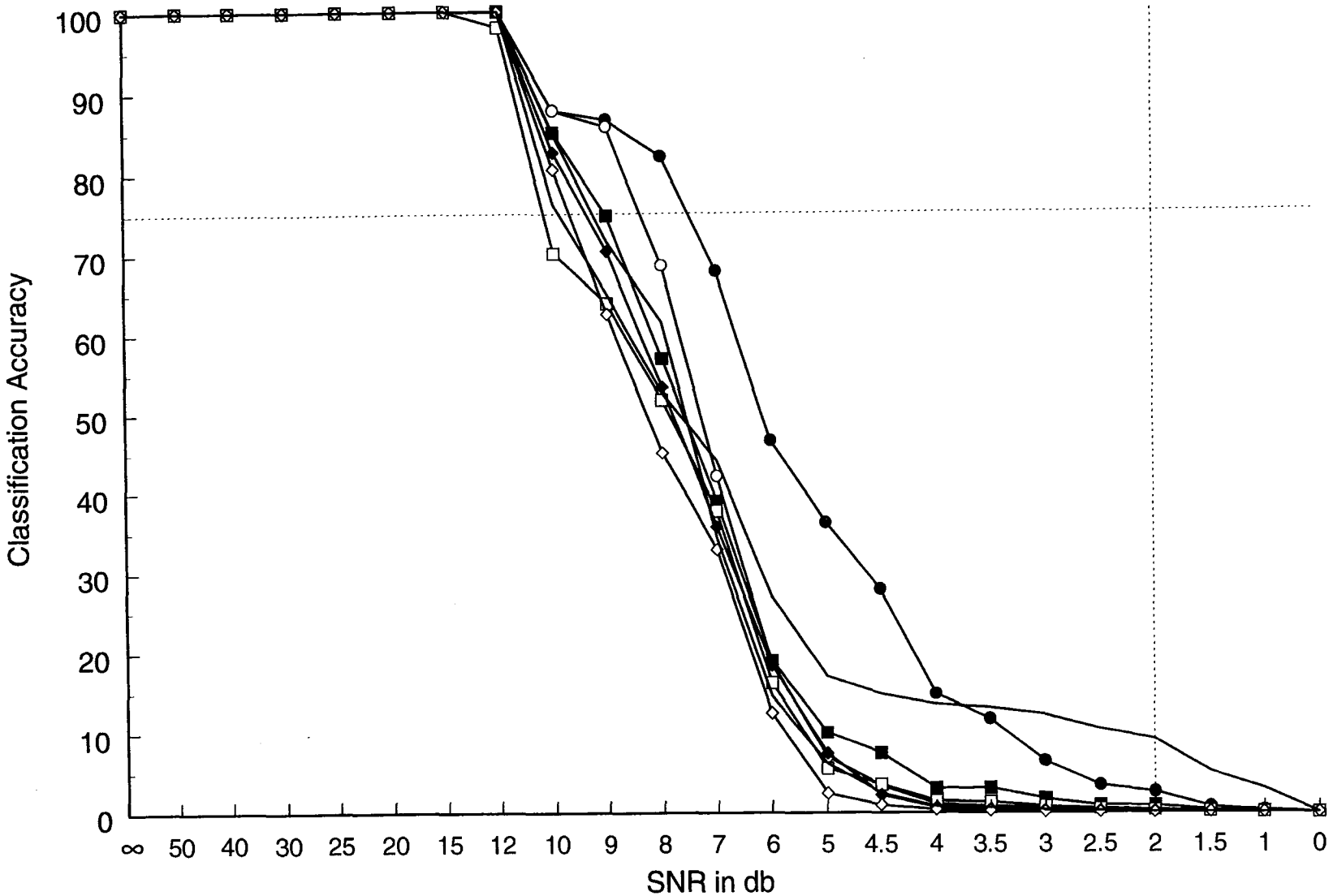


Figure 4.34

Variation in classifier performance with initialization of network weights

- standard normalization of the training and test feature vectors.

The results for the SZF, PZF, SZRP, PZRP, WRF, and HRF bases are shown in figures 4.35 through 4.37. Each figure shows the classifier performance for similar basis pairs (i.e., SZF with PZF, SZRP with PZRP, and WRF with HRF) for both training sets and it is obvious in each case that the incorporation of noisy data into the training process greatly improves the classification accuracy.

The results shown for SZF and PZF cases clearly show the best performance for both training sets. Indeed, these results provide the standard by which the remaining basis function results are measured. The performance shown in figure 4.35 for the case of the training set B and the basis PZF approaches that of a human interpreter of the same images, i.e., classification accuracies of 75% or better are measured for a SNR as low as 2 db. The reader is referred back to figures 3.5 and 3.4 for illustrations of 2 db SNR images.

The results illustrated for the SZRP and PZRP bases in figure 4.36 clearly fall far short of those for the SZF and PSF results. Nevertheless, the SZRP and PZRP measurements are among the most interesting and revealing (keeping in mind the maxim that more is learned from failures than from successes). The most surprising and striking characteristic of the SZRP and PZRP results is that, for the case of the training set A, the classifier catastrophically fails in each case when only very small amounts of noise have been added to the images (SNR of approximately 30 db - see figure 3.4). The reason for this is directly traceable to the fact that, since the SZRP and PZRP bases have non-zero measures (and only these bases have this), the feature vectors are dramatically changed by the addition of even very small amounts of noise, i.e., figures 4.24 and 4.25. The use of training set B, which includes examples of noisy feature vectors, dramatically improves the classification performance of the SZRP and PSRP; indeed, the degree of improvement is such that they approach that measured for the SZF and PSF bases. In fact, when compared to figure 4.37, it may be seen that the SZRP and PZRP cases improve from being the worst cases (i.e., fifth and sixth places) for training set A to being "runners-up" (i.e., third and fourth places) when training set B is used.

Given the relatively poor invariance for the WRF and HRF feature vectors as illustrated in figures 4.19 and 4.20, respectively, the lack luster performance of the neural classifier when these bases are used, figure 4.37, should come as no great surprise. However, even with the high variability in the feature vectors for the WRF and HRF and the extremely low variation in those for the SZRP and PZRP bases, figures 4.17 and 4.18, respectively, it is interesting that the performance for the WRF and HRF betters that for the SZRP and PSRP for training set A although it falls short of matching the superior performance for the SZF and

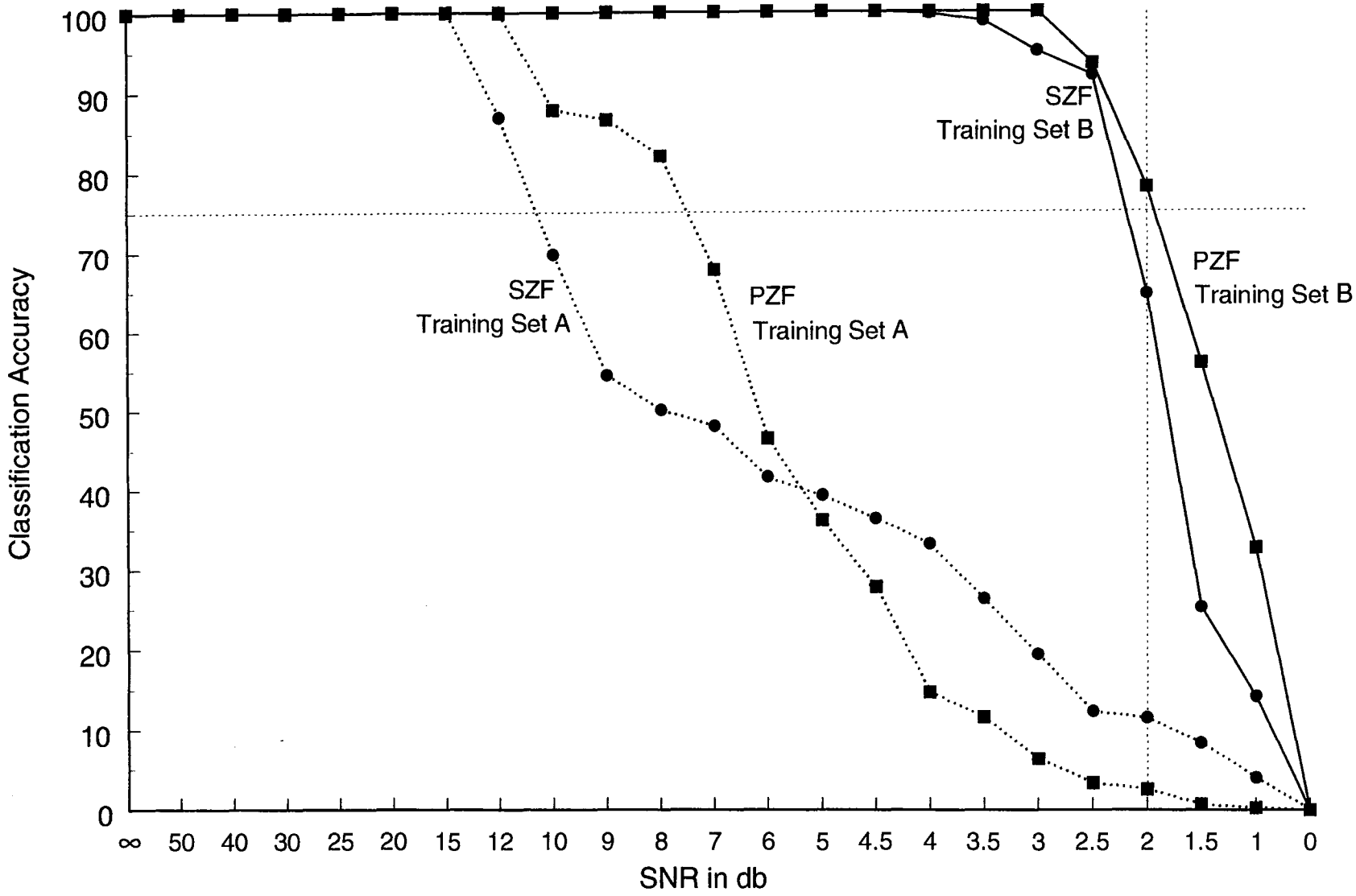


Figure 4.35

Classifier performance for the SZF and PZF bases, standard normalization.

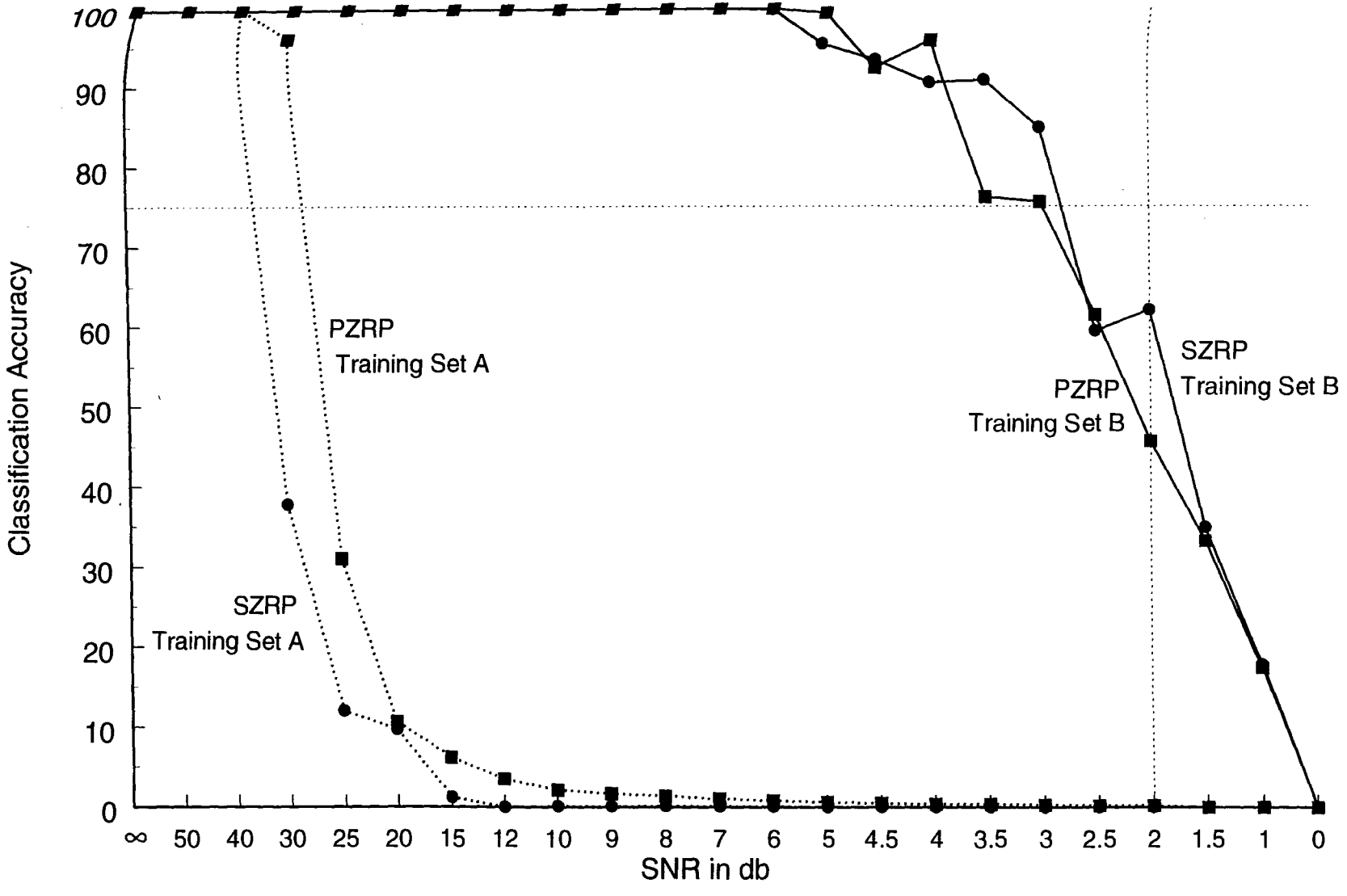


Figure 4.36

Classifier performance for the SZRP and PZRP bases, standard normalization.

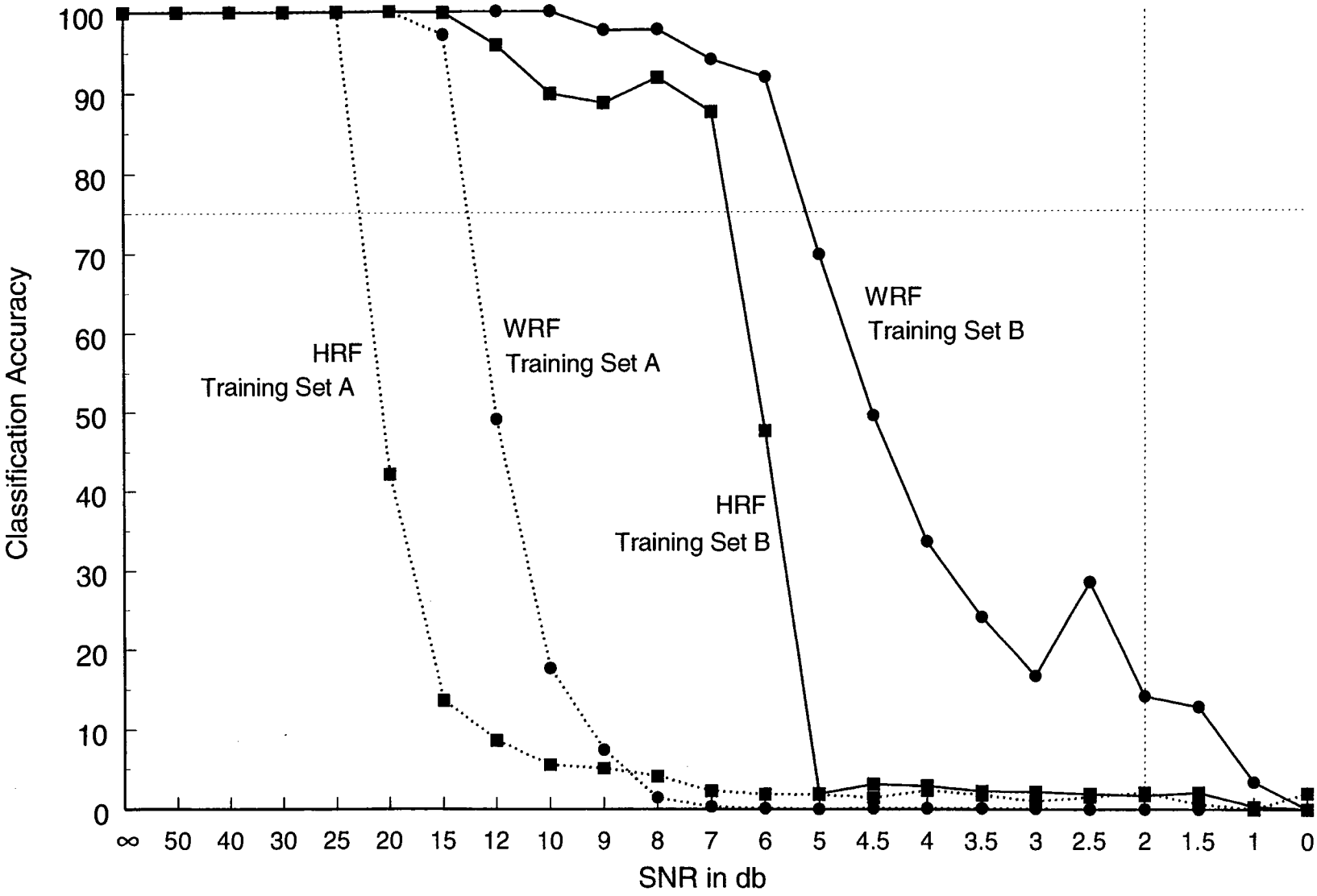


Figure 4.37

Classifier performance for the WRF and HRF bases, standard normalization.

PZF cases for either training set. As well, the improvement for the WRF and HRF bases from training set A to training set B is the least of all the improvement observed for all six bases with the consequence that, as noted above, the SZRP and PZRP bases outperform the WRF and HRF bases when training set B is employed. Clearly, the importance of adopting zero measure basis functions outweighs the shortcomings in calculated invariance in determining the performance of the classifier. In addition, as discussed in section 4.1, several of the components in the WRF and HRF feature vectors are redundant so that, in effect, the measurements made for these basis functions are with a feature vector having an effective dimension considerably less than the stated value of 45. This is a compensating factor when judging the relatively poor performance of these bases in comparison to the full SZF and PZF. However, this "allowance" may amount to choosing between "a rock and a hard place" since it would not be possible, given the limitations on the sampling capabilities of the pixel grid for the Walsh functions, to significantly increase the dimension of either the WRF or HRF bases in order to compensate for this redundancy in the feature vector components. In particular, these difficulties would only be expected to become significantly worse for grid dimensions less than 256 x 256.

For ease of comparison, figure 4.38 shows the performance measurements for the six basis functions using training set A while figure 4.39 shows the results for the six bases when using the training set B. Overall, the PZF basis consistently outperforms all other bases although the SZF basis runs a close second and deserves, particularly in light of figure 4.34, an "honorable mention".

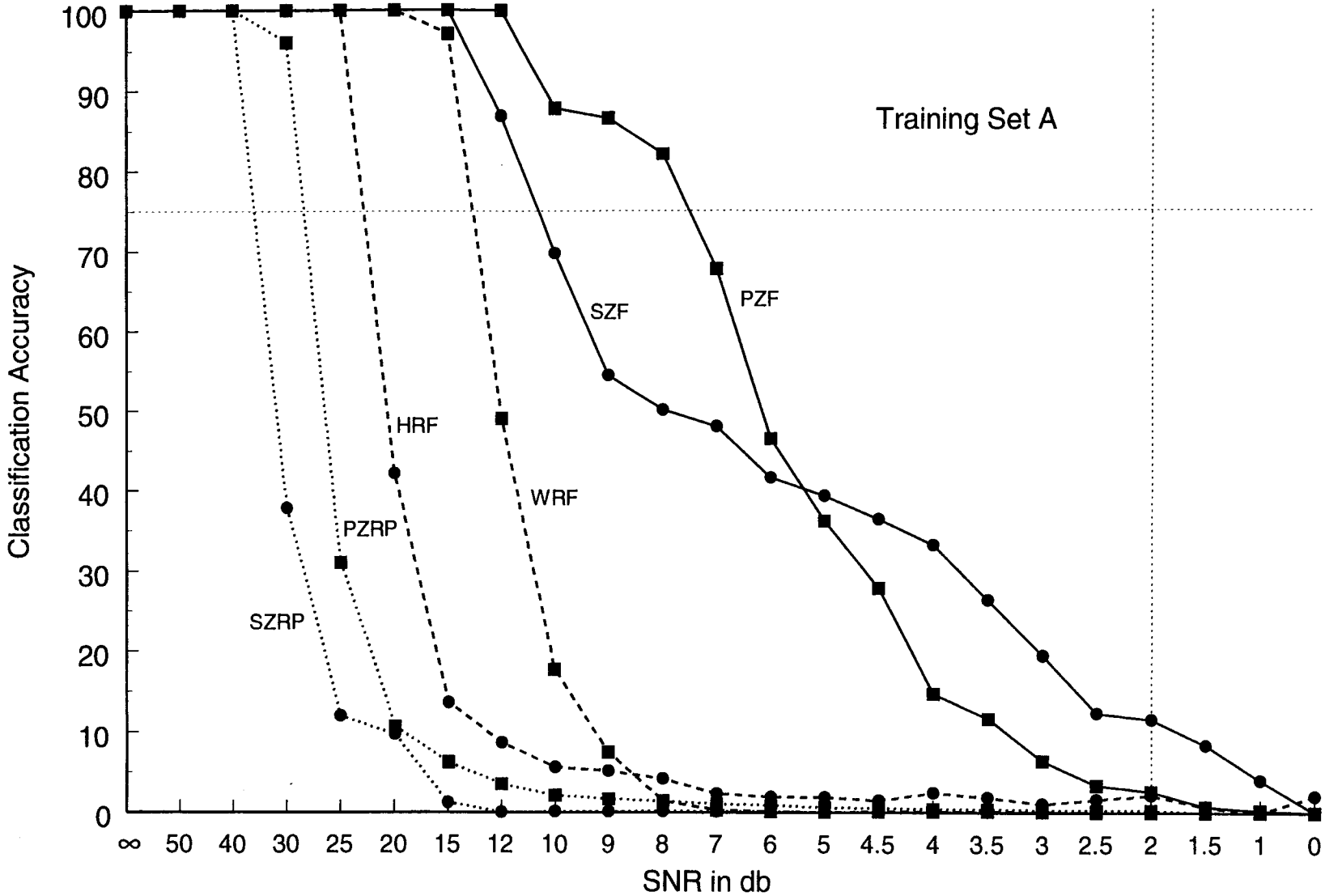


Figure 4.38

Classifier performance for all bases, standard normalization on Training Set A

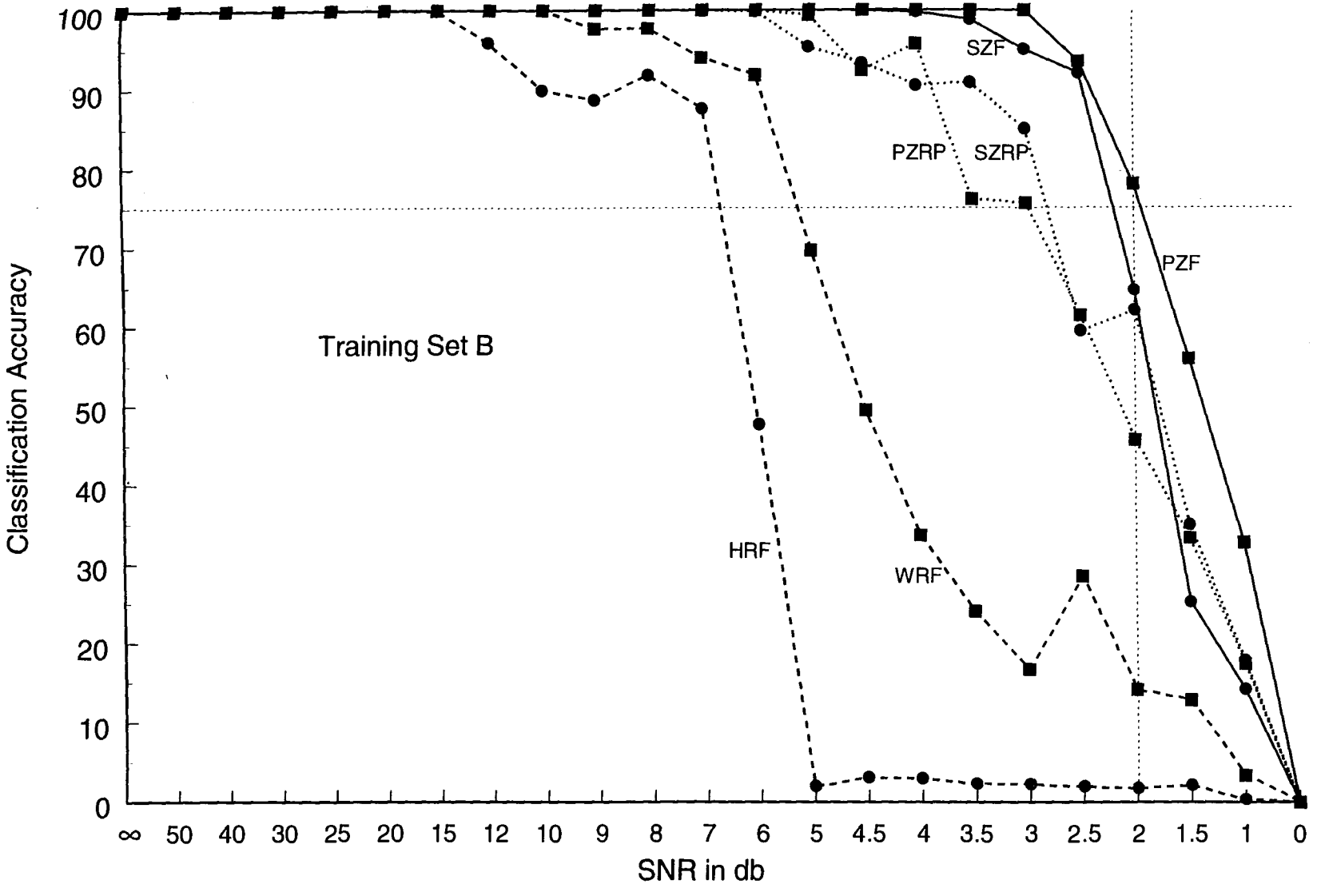


Figure 4.39

Classifier performance for all bases, standard normalization on Training Set B

Chapter 5 : Discussion and Conclusions

The work described in this report has examined the performance capabilities of a multi-layer perceptron neural network trained with the backpropagation algorithm as a classifier of elementary, 2-D imagery. Six different mathematical bases were studied for two types of training sets, one containing only noiseless imagery and the second containing a mixture of noiseless and noisy imagery. In addition, four different schemes were investigated for the normalization of the feature vectors. Other prefatory experiments were described which measured the neural network's dependence upon epoch size, the number of hidden neurons, and upon the network initialization. A series of graphs of feature vectors was presented which illustrated the measured invariance of the feature vectors, their degradation with decreasing SNR, similarities and differences between feature vectors for similar and dissimilar numerals, and the dependence of the feature vectors upon the normalization method.

The principal findings of this report may be summarized as follows:

- The neural classifier used was a three-layer perceptron network with 24 hidden neurons, 45 input neurons (the feature vector dimension), and 9 output neurons (the number of image classes). The network was trained with the backpropagation algorithm using an epoch value of 16.

- Of the six sets of basis functions tested, the basis which provided the best performance was the PZF (closely followed by the SZF basis). Indeed, as figure 4.39 indicates, the neural network classifier trained on training set B and using the PZF basis for the feature vector extraction exhibited an accuracy comparable to human capability to classify the same set of imagery (the neural classifier maintained a classification accuracy above 75% for SNR's down to 2 db inclusively).

- All of the bases studied showed marked improvement in the classifier's performance when the mixed noiseless plus noisy training set B was used in place of the noiseless training set A.

- The SZRP and PZRP bases exhibited "catastrophic" failure for training set A but showed the greatest improvement in performance when training set B was employed.

- The Walsh bases, WRF and HRF, performed poorly for both training sets, showing the least improvement of all the bases for training set B over training set A.

- Based upon the results with the SZRP and PZRP bases which were the only basis functions which possessed non-zero measures, it was concluded that a necessary condition for the basis functions to be insensitive to the presence of image noise was that they possess zero measure.

- Of the four normalization schemes investigated, the normalization method which consistently resulted in the best performance was the 'standard normalization' (section 3.5). This normalization is characterized by a zero mean for each of the feature vector components over the training set and training feature vectors which have *either* a maximum value of +1 *or* a minimum value of –1.

Retrospectively, the decision to study the characteristics of the *methodology* of moment invariant feature vectors and a neural network classifier using the simplified image database of this report was both a sound and necessary one. The principal findings from this phase of the work were greatly facilitated and the conclusions drawn made much clearer by the use of this simplified image database possessing known, well understood image and statistical characteristics. The more subtle inferences such as that concerning the dependence of noise sensitivity upon the measure of the basis functions being zero or non-zero or the highly component-dependent invariance of the HRF basis would almost certainly have gone undetected had a more complex image database been employed.

With the results of this study in hand along with the extensive library of computer algorithms developed, the stage is set to proceed with the application of neural network classification using PZF-based moment invariant feature vectors to identify and classify SAR imagery. Four examples drawn from a database of SAR imagery which is intended for use in this next phase of the project are illustrated in figure 5.1. Three distinct object classes are shown in this figure with figures 5.1(a) and 5.1(d) representing different images of the exact same object. Clearly, as these images are intended to illustrate, the classification of such imagery, except to the very experienced interpreter, is neither intuitive nor elementary by nature and the intended objective of accurate classification by artificial means represents a challenging one. An essential, arguably crucial, requirement for the application of the experimental classification system presented in this report to this type of SAR imagery is that the feature vectors computed for images such as those typified in figure 5.1 must possess, to some acceptable degree of approximation, invariance to the scale, lateral position, and angle of view for images belonging to the same class of objects. The finite resolution of the SAR imagery system, the existence of measurable returns from the background of the object, and the presence of noise and speckle in the image are examples of characteristics which are expected to detract from the invariance of the calculated feature vectors. However, the two

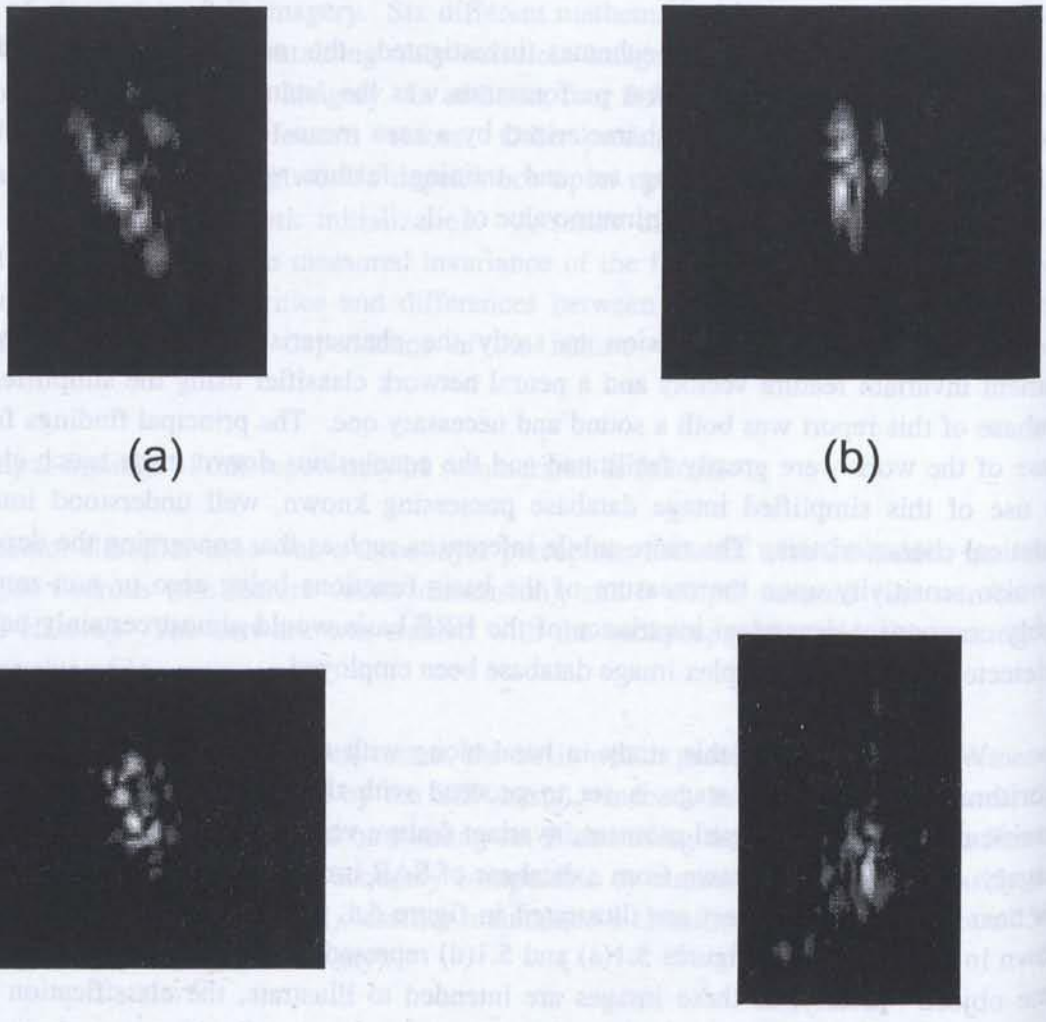


Figure 5.1

Examples of airborne SAR imagery of isolated ground targets. Figures (a), (b), and (c) represent different object classes. Figure (d) is from the same class as figure (a).

factors which are anticipated to most seriously affect the accuracy and invariance of the calculated features are the following:

1. SAR images of land targets represent 2-D projections of complex, 3-D objects. Thus, there are *two* angles which determine the profile of the object as seen by the imaging system and not simply the "in-plane" rotation angle considered in the present study of 2-D images. Even under ideal circumstances, the neural network classifier must, in the case of SAR imagery, be expected to learn in the training phase some measure of invariance beyond the in-plane invariance incorporated into the calculation of the feature vector. The practical limitation to how well the neural network can learn this more general invariance may be ultimately determined by the number of images which constitute the training database. Moreover, as with any form of 3-D to 2-D projection[38,99], there will inevitably exist some degree of degeneracy with angle of view, i.e., similar objects will be indistinguishable when viewed from certain perspectives.

2. The fundamental nature of the radar imaging process underlying SAR technology dictates that the intensity of both the absolute and relative radar returns from the different portions of the object will change dramatically with small changes in the angle of view. This, in turn, means that the feature vectors may be expected to exhibit, even after feature vector normalization, a high degree of sensitivity to angle of view. In most respects, it is this variability of intensity distribution over the 2-D projection which is expected to present the most serious difficulty in the classification process. Again, as with the previous factor, limits to overcoming this difficulty may be largely determined by the experimenter's ability to present a sufficiently large number of examples to the neural network during the training phase.

Clearly, concerns such as those discussed above must be explicitly addressed by the experimenter in applying the method of moment invariant feature vectors for neural network classification of SAR imagery.

In conclusion, the methodology of using a neural network classifier trained on moment invariant feature vectors as described in this report offers a very promising and encouraging approach to the particular pattern recognition problem posed by SAR image classification. The robustness and superior performance levels of the PZF-based classifier as presented in this report strongly suggest that it should be capable of classifying SAR imagery with an accuracy which would make it a viable and practical tool for SAR image classification.

REFERENCES

1. Abu-Mostafa Y. S. and D. Psaltis, "Recognitive aspects of moment invariants", IEEE Trans. Pattern Anal. Machine Intell. **6**, no. 6, pp. 698-706, Nov. 1984.

2. Abu-Mostafa Y. S. and D. Psaltis, "Image normalization by complex moments", IEEE Trans. Pattern Anal. Machine Intell. **7**, no. 1, pp. 46-55, Jan. 1985.

3. Bachmann C. M., S. A. Musman, D. Luong, and A. Schultz, "Unsupervised BCM projection pursuit algorithms for classification of simulated radar presentations", Neural Networks **7**, no. 4, pp. 709-728, 1994.

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

5. Bailey R. R. and M. Srinath, "Orthogonal moment features for use with parametric and non-parametric classifiers", IEEE Trans. Pattern Anal. Machine Intell. **18**, no. 4, pp. 389-398, Apr. 1996.

6. Baldi P. and S. S. Venkatesh, "Random interactions in higher order neural networks", IEEE Trans. Information Theory **39**, pp. 274-283, Jan. 1993.

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

8. Beauchamp K. G., **Walsh Functions and their Applications**, Academic Press, London, U.K., 1975.

9. Beauchamp K. G., **Applications of Walsh and Related Functions - with an Introduction to Sequency Theory**, Academic Press, London, U.K., 1984

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

11. Beckmann P., **Orthogonal Polynomials for Engineers and Physicists**, Golem Press, Boulder CO, 1973, pp. 150-156.

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

13. Berenyi H. M., V. F. Leavers, an R. E. Burge, "Automatic detection of targets against cluttered backgrounds using a fractal-orientated statistical analysis and Radon transform", Pattern Recognition Lett. **13**, pp. 869-877, Dec. 1992.

14. Bernardon A. M. and J. E. Carrick, "A neural system for automatic target learning and recognition applied to bare and camouflaged SAR targets", Neural Networks **8**, no. 7/8, pp. 1103-1108, 1995.

15. Bhatia A. B. and E. Wolf, "On the circle polynomials of Zernike and related othogonal sets", Proc. Camb. Phil. Soc. **50**, pp. 40-48, 1954.

16. Blumenkrans A., "Two-dimensional object recognition using a two-dimensional polar transform", Pattern Recognition **24**, no. 9, pp. 879-890, 1991.

17. Born M. and E. Wolf, **Principles of Optics**, 4th ed., 1970, pp. 464-468 and 767-772.

18. Botha E. C., E. Barnard, and C. J. Barnard, "Feature-based classification of Aerospace radar targets using neural networks", Neural Networks **9**, no. 1, pp. 129-142, 1996.

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

20. Burns T. J., K. H. Fielding, S. K. Rogers, S. D. Pinski, and D. W. Ruck, "Optical Haar wavelet transform", Opt. Eng. **31**, pp. 1852-1857, Sept. 1992.

21. Burr D. J., "Experiments on neural net recognition of spoken and written text", IEEE Trans. Acoustics, Speech, and Signal Processing **36**, no. 7, pp. 1162-1168, July 1988.

22. Caelli T. M. and Z. Q. Liu, "On the minimum number of templates required for shift, rotation and size invariant pattern recognition", Pattern Recognition **21**, no. 3, pp. 205-216, 1988.

23. Casasent D. and S. Natarajan, "A classifier neural net with complex-valued weights and square-law nonlinearities", Neural Networks **8**, no. 6, pp. 989-998, 1995.

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

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

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

27. Chen C. C., "Improved moment invariants for shape discrimination", Pattern Recognition **26**, no. 5, pp. 683-686, 1993.

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

29. Chen P. and W. W. Seemuller, "Application of Walsh transforms for topographic feature extraction using a sensor array system", IEEE Trans. Instru. Meas. **29**, pp. 52-57, March 1980.

30. Dayhoff J., **Neural Network Architectures - An Introduction**, Van Nostrand Reinhold, New York, 1990.

31. Delopoulos A., A. Tirakis, and S. Kollias, "Invariant image recognition using triple-correlations", Proc. Sixth European Signal Processing Conf. EUSIPCO-92, Brussels, Belgium, Aug. 1992, Signal Proc. VI : Theories and Applications, Vol. I, pp.571-574.

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

33. Fine N. J., "On the Walsh functions", Trans. Am. Math. Soc. **65**, pp. 372-414, 1949.

34. Fine N. J., "The generalized Walsh functions", Trans. Am. Math. Soc. **69**, pp. 66-77, 1950.

35. Flusser J. and T. Suk, "Pattern recognition by affine moment invariants", Pattern Recognition **26**, no. 1, pp. 167-174, 1993

36. Fu C., J. Yen, and S. Chang, "Calculation of moment invariants via Hadamard transform", Pattern Recognition **26**, no. 2, pp. 287-293, 1993.

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

38. Galvez J. M. and M. Canton, "Normalization and shape recognition of three-dimensional objects by 3D moments", Pattern Recognition **26**, no. 5, pp. 667-681, 1993.

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

40. Grace A. E. and M. Spann, "A comparison between Fourier-Mellin descriptors and moment based features for invariant object recognition using neural networks", Pattern Recognition Lett. **12**, pp. 635-643, Oct. 1991.

41. Grossberg S., H. Hawkins, and A. Waxman (eds.), **Automatic Target Recognition**, 1995 Special Issue of Neural Networks **8**, no. 7/8, 1995.

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

43. Grossberg S., E. Mingolla, and J. Williamson, "A multiple scale neural system for boundary and surface representation of SAR data", Proc. 1995 IEEE Workshop "Neural Networks for Signal Processing V", IEEE Press, 1995, pp. 313-322.

44. Gullichsen E. and E. Chang, "Pattern classification by neural network: An experimental system for icon recognition", IEEE 1st Int. Conf. Neural Networks, San Diego, CA, June 1987, vol. IV, pp. 727-732.

45. Hammond Jr. J. L. and R. S. Johnson, "A review of orthogonal square-wave functions and their applications to linear networks", J. Franklin Institute **273**, pp. 211-225, 1962.

46. Harmuth H. F., **Transmission of Information by Orthogonal Functions**, Springer, New York, 2nd ed.,1972.

47. Harmuth H. F., **Sequency Theory - Foundations and Applications**, Academic Press, New York, 1977 (2-D and 3-D Walsh functions are discussed in section 1.1.7).

48. Hatzivasiliou F. and K. L. Sala, "A novel approach to the application of higher order neural networks to image classification", Communications Research Center Technical Report CRC 96-005, May 1996.

49. Haykin S., **Neural Networks - A Comprehensive Foundation**, Macmillan College Publ. Co., New York, 1994.

50. Henderson K. W., "Some notes on the Walsh functions", IEEE Trans. Elec. Computers **13**, no. 1, pp. 50-52, Feb. 1964.

51. Hirose A., "Proposal of fully complex-valued neural networks", Int. Joint Conf. Neural Networks, Baltimore, MD, June 1992, vol. IV, pp. 152-157.

52. Hong Z. Q., "Algebraic feature extraction of image for recognition", Pattern Recognition **24**, no. 3, pp. 211-219, 1991.

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

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

55. Jia X. and M. S. Nixon, "Extending the feature vector for automatic face recognition", IEEE Trans. Pattern Anal. Machine Intell. **17**, no. 12, pp. 1167-1176, Dec. 1995.

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

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

58. Khotanzad A. and Y. H. Hong, "Invariant image recognition by Zernike moments", IEEE Trans. Pattern Anal. Machine Intell. **12**, no. 5, pp. 489-497, May 1990.

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

60. Khotanzad A. and J. H. Lu, "Distortion invariant character recognition by a multi-layer perceptron and back-propagation learning", IEEE Conf. Neural Networks, San Diego, CA, July 1988, vol. I, pp. 625-632.

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

62. Kintner E. C., "On the mathematical properties of the Zernike polynomials", Optica Acta **23**, no. 8, pp. 679-680, 1976.

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

64. Lackey R. B. and D. Meltzer, "A simplified definition of Walsh functions", IEEE Trans. Computers **20**, no. 2, pp. 211-213, Feb. 1971.

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

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

67. Li B., "The moment calculation of polyhedra", Pattern Recognition **26**, no. 8, pp. 1229-1233, 1993.

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

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

70. Li S. Z., "Matching: Invariant to translations, rotations and scale changes", Pattern Recognition **25**, no. 6, pp. 583-594, 1992.

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

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

73. Liao S. X. and M. Pawlak, "On image analysis by moments", IEEE Trans. Pattern Anal. Machine Intell. **18**, no. 3, pp. 254-266, March 1996.

74. Lin W. G. and S. S. Wang, "A new neural model for invariant pattern recognition", Neural Networks **9**, no. 5, pp. 899-913, 1996.

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

76. Menon M. M., E. R. Boudreau, and P. J. Dolodzy, "An automatic ship classification system for ISAR imagery", Lincoln Laboratory Journal **6**, no. 2, pp. 289-307, 1993.

77. Morgenthaler G. W., "On Walsh-Fourier series", Trans. Am. Math. Soc. 84, pp. 472-507, 1957.

78. Mukundan R., N. K. Malik, and K. R. Ramakrishnan, "Attitude estimation using moment invariants", Pattern Recognition Lett. **14**, pp. 199-205, Mar. 1993.

79. Mukundan R. and K. R. Ramakrishnan, "Fast computation of Legendre and Zernike moments", Pattern Recognition **28**, no. 9, pp. 1433-1442, 1995.

80. Nair D., A. Mitiche, and J. K. Aggarwal, "On comparing the performance of object recognition systems", Proc. IEEE Int. Conf. Image Processing ICIP-95, Washington D. C., Oct. 1995, vol. II, pp. 631-634.

81. Paquet E., M. Rioux, and H. H. Arsenault, "Invariant pattern recognition for range images using the phase Fourier transform and a neural network", Optical Eng. **34**, no. 4, pp. 1178-1183, Apr. 1995.

82. Pawlicki T., "Recognizing image invariants in a neural network architecture", IEEE Conf. Neural Networks, San Diego, CA, July 1988, vol. II, pp. 135-142.

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

84. Philips W., "A new fast algorithm for moment computation", Pattern Recognition **26**, no. 11, pp. 1619-1621, 1993.

85. Pratt W. K., J. Kane, and H. C. Andrews, "Hadamard transform image coding", Proc. IEEE **57**, no. 1, pp. 58-68, Jan. 1969.

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

87. Raghu P. P., R. Poongodi, and B. Yegnanarayana, "A combined neural network approach for texture classification", Neural Networks **8**, no. 6, pp. 975-987, 1995.

88. Ranganath H. S., D. E. Kerstetter, and S. R. F. Sims, "Self partitioning neural networks for target recognition", Neural Networks **8**, no. 9, pp. 1475-1486, 1995.

89. Ravichandran A. and B. Yegnanarayana, "Studies on object recognition from degraded images using neural networks", Neural Networks **8**, no. 3, pp. 481-488, 1995.

90. Reddi S. S., "Radial and angular moment invariants for image identification", IEEE Trans. Pattern Anal. Machine Intell. **3**, no. 2, pp. 240-242, March 1981.

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

92. Reiss T. H., "The revised fundamental theorem of moment invariants", IEEE Trans. Pattern Anal. Machine Intell. **13**, no. 8, pp. 830-834. Aug. 1991.

93. Rivlin E. and I. Weiss, "Local invariants for recognition", IEEE Trans. Pattern Anal. Machine Intell. **17**, no. 3, pp. 226-238, Mar. 1995.

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

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

96. Rothe I., H. Süsse, and K. Voss, " The method of normalization to determine invariants", IEEE Trans. Pattern Anal. Machine Intell. **18**, no. 4, pp. 366-375, 1996.

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

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

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

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

101. Shepherd T. S., W. Uttal, S. Dayanand, and R. Lovell, "A method for shift, rotation, and scale invariant pattern recognition using the form and arrangement of pattern-specific features", Pattern Recognition **25**, no. 4, pp. 343-356, 1992.

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

103. Shirali-Shahreza M. H., K. Faez, and A. Khotanzad, "Recognition of handwritten Farsi numerals by Zernike moments features and a set of class-specific neural network classifiers", Int. Conf. Signal Processing Applications and Technology ICSPAT 94, Dallas, Texas, Oct. 1994, pp. 998-1003.

104. Singer M. H., "A general approach to moment calculation for polygons and line segments", Pattern Recognition **26**, no. 7, pp. 1019-1028, 1993.

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

106. Soumekh M., S. Nugroho, S. Jones, R. Rysdyk, and S. Tran, "ISAR imaging of an airborne DC-9", IEEE Int. Conf. Acoustics, Speech, and Signal Processing ICASSP-93, Minneapolis, Minnesota, Apr. 1993, vol. V, pp. 465-468.

107. Spirkovska L., "Three-dimensional object recognition using similar triangles and decision trees", Pattern Recognition **26**, no. 5, pp. 727-732, 1993.

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

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

110. Svärdström A., "Neural network feature vectors for sonar targets classifications", J. Acoust. Soc. Am. **93**, no. 5, pp. 2656-2665, May 1993.

111. Swick D. A., "Walsh function generation", IEEE Trans. Information Theory **15**, no. 1, pg. 167, Jan. 1969.

112. Szu H. H., B. Telfer, and S. Kadambe, "Neural network adaptive wavelets for signal representation and classification", Opt. Eng. 31, pp. 1907-1916, Sept. 1992.

113. Szu H. H., B. Telfer, and A. Lohmann, "Causal analytical wavelet transform", Opt. Eng. 31, pp. 1825-1829, Sept. 1992.

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

115. Teh C. and R. T. Chin, "On digital approximation of moment invariants", Computer Vision, Graphics, and Image Processing **33**, pp. 318-326, 1986.

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

117. Telfer B. and H. H. Szu, "New wavelet transform normalization to remove frequency bias", Opt. Eng. **31**, pp. 1830-1834, Sept. 1992.

118. Treurniet W. C., K. L. Sala, and R. Klepko, "Neural network models for classifying synthetic aperture radar images of ships – Phase 1", Communications Research Center Technical Report CRC 93-001, Oct. 1993.

119. Trier O. D. , A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition - A survey", Pattern Recognition **29**, no. 4, pp. 641-662, 1996.

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

121. Tsirikolias K. and B. G. Mertzios, "Statistical pattern recognition using efficient two-dimensional moments with applications to character recognition", Pattern Recognition **26**, no. 6, pp. 877-882, 1993.

122. Wallin A. and O. Kübler, "Complete sets of complex Zernike moment invariants and the role of the pseudoinvariants", IEEE Trans. Pattern Anal. Machine Intell. **17**, no. 11, pp. 1106-1110, Nov. 1995.

123. Walsh J. L., "A closed set of orthogonal functions", Am. J. Math. **55**, pp. 5-24, 1923.

124. Wang S. S. and W. G. Lin, "A new self-organizing neural model for invariant pattern recognition", Pattern Recognition **29**, no. 4, pp. 677-687, 1996.

125. Wang P. P. and R. C. Shiau, "Machine recognition of printed Chinese characters via transformation algorithms", Pattern Recognition **5**, no. 4, pp. 303-321, 1973.

126. Waxman A. M., M. Seibert, A. M. Bernardon, and D. A. Fay, "Neural systems for automatic target learning and recognition", Lincoln Laboratory Journal **6**, no. 1, pp. 77-115, 1993

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

128. Widrow B., M. Lehr, F. Beaufays, E. Wan, and M. Bilello, "Learning algorithms for adaptive signal processing and control", IEEE Conf. Neural Networks, San Francisco, CA, March/Apr. 1993, vol. I, pp. 1-8.

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

130. Wood J., "Invariant pattern recognition: A review", Pattern Recognition **29**, no. 1, pp. 1-17, 1996.

131. Yang L. and F. Albregtsen, "Fast and exact computation of Cartesian geometric moments using discrete Green's theorem", Pattern Recognition **29**, no. 7, pp. 1061-1073, 1996.

132. Yüceer C. and K. Oflazer, "A rotation, scaling, and translation invariant pattern classification system", Pattern Recognition **26**, no. 5, pp. 687-710, 1993.

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

Appendix A
Fortran Source Code for Core Programs

This appendix contains the source code listings for the following programs:

1. PIXELIZE.FOR
   Reads TIFF bitmap images, reduces the image resolution from 1024 x 1024 to 256 x 256, scales and shifts image to produce irradiance normalized central moments, adds noise to a user-provided SNR level, and saves final result as native binary format file.

2. GENFVZER.FOR
   Generates the raw feature vector file for an image or set of images using any of the SZF, PZF, SZRP, or PZRP bases.

3. GENFVWAL.FOR
   Generates the raw feature vector file for an image or set of images using either of the WRF or HRF bases.

4. NORMFV.FOR
   Normalizes the raw feature vectors. For training data, stores the normalization parameters derived from the training set. For test data, reads normalization parameters. Three different normalization schemes SN, PN, and BN are available as a user option.

5. GENRV.FOR
   Generates the result vector (generally, a 9 x 9 matrix with +1 for 'true' and 0 for 'false') for use by gencv.for.

6. GENCV.FOR
   Generates the complete vector in the nna format required as input for the NeuralWare simulator software by combining the normalized feature vector with the result vector from genrv.for. Also generates the DN feature vectors.

7. RATENNR.FOR
   Rates the classification results obtained from the network as given by the nnr file generated by the NeuralWare simulator software using a sigmoidal weighting function to measure the effective value of the individual results.

```
C=====================================================================
C     Program : PIXELIZE.FOR (PIXELIZE the TIFF images)
C     Version : 4.2  July 22, 1994
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        PIXELIZE reads TIFF bitmap files (generated by the COREL Draw
C        software) and converts them to "pixel" image files (see below).
C
C        Four major operations are carried out by this program:
C
C        (1)   The resolution of the TIFF file is reduced by an integral
C              factor (1,2,3,4,..) with the reduced-size pixel derived
C              as the average value of the square it represents (e.g.,
C              a 1024x1024 bitmap image can be reduced to a 256x256
C              pixel image with each pixel the average of 16 of the
C              original TIFF pixels);
C
C        (2)   The 1-bit (B&W), 2-bit (4 level grayscale), 4-bit
C              (16-bit), or 8-bit (256 level grayscale) TIFF values
C              are converted to (possibly averaged) pixel values with
C              one byte per pixel (integer for B&W and floating point
C              for true grayscales) with white=0 and black=1;
C
C        (3)   The image moments m00, m01, and m10 are calculated for
C              the image.  The image is first shifted so that its
C              "optical center of mass" is approximately at the
C              coordinate origins and then scaled to produce an image
C              with a pre-determined value for m00.  The scaled image
C              is then iteratively shifted to produce minimized values
C              for the first moments m01 and m10 (ideally = 0), i.e.,
C              the scaled image is "optically" centered on the x,y
C              grid.
C
C        (4)   Random noise is superimposed upon the pixel image (a
C              user option).  The user, if electing to add noise to
C              the image, is prompted for the number of iterations to
C              be used for the random noise routine (these numbers having
C              been precalibrated to correspond to known SNR ratios for
C              the final image).  The program then uses a "Numerical
C              Recipes" function RAN2 to generate the noise
C              corresponding to the requested SNR.  For binary images,
C              the modification of the pixels is carried out by adding
C              1 modulo 2 to the pixel.
C
C        The pixel data is calculated for a square grid of dimension
C        2N x 2N and then saved as FASCII files with 'automatic'
C        assignment of filenames.
C
C
C     PROGRAMMING NOTES:
C
```

```
c     1. This program is NOT, repeat NOT, a general TIFF reader.  It
c        does read and decipher the TIFF version 5.0 files generated by
c        Corel Draw software for B&W/grayscale bitmaps.
c
c     2. The gridsize is NOT a variable for this program.  However,
c        care has been taken to allow the user to change this
c        parameter relatively easily.  Only the dimensioning
c        assignments, initialization values, and some format
c        statements would have to be altered to allow for a different
c        gridsize.
c
c     3. The TIFF file/tag parameters used in this program are those
c        recommended for the TIFF 5.0 specification (HP/Aldus).
c        Specifically, four arrays TAGNAME, TAGID, TAGVALUE, and
c        TAGTYPE (dimensions of 37, 36, 51, and 5 respectively) "define"
c        the various characteristics of the TIFF data.  All of these
c        arrays are predefined by program 'data' statements with the
c        values of TAGVALUE representing default values (where
c        possible).  Only TAGVALUE is altered during the execution of
c        this program.  The arrays are:
c
c ARRAY   TAG              TAGNAME                    DATA    DEFAULT      ERROR
c   #     ID#                                         TYPE                 VALUES
c
c    1   254(0FEh)  New Subfile Type                  4          0          .ne.0
c    2   256(100h)  Image Width (Pixels)             3 or 4     0   .ge.0 & .le.2N
c    3   257(101h)  Image Height (Pixels)            3 or 4     0   .ge.0 & .le.2N
c    4   258(102h)  Bits per Sample                  3          1       .ne.1,4,8
c    5   259(103h)  Compression                       3          1          .ne.1
c    6   262(106h)  Photometric Interpretation       3          0          .ne.0,1
c    7   263(107h)  Thresholding                     3          1
c    8   269(10Dh)  Document Name                    2          0
c    9   270(10Eh)  Image Description               2          0
c   10   271(10Fh)  Make                            2          0
c   11   272(110h)  Model                           2          0
c   12   273(111h)  Strip Offsets                   3 or 4     0
c   13   277(115h)  Samples per Pixel               3          1          .ne.1
c   14   278(116h)  Rows per Strip                  3 or 4     1
c   15   279(117h)  Strip Byte Counts              3 or 4     1
c   16   282(11Ah)  X Resolution (dpi)             5         38 (offset)
c   17   283(11Bh)  Y Resolution (dpi)             5         40 (offset)
c   18   284(11Ch)  Planar Configuration           3          1
c   19   285(11Dh)  Page Name                       2          0
c   20   286(11Eh)  X Position                      5         42 (offset)
c   21   287(11Fh)  Y Position                      5         44 (offset)
c   22   290(122h)  Gray Response Unit             3          0
c   23   291(123h)  Gray Response Curve            3          0
c   24   292(124h)  Group 3 Options                4          0
c   25   293(125h)  Group 4 Options                4          0
c   26   296(128h)  Resolution Units              3          2
c   27   297(129h)  Page Number                    3          0
c   28   305(131h)  Software                        2          0
c   29   306(132h)  Date & Time                     2          0
c   30   315(13Bh)  Artist                          2          0
c   31   316(13Ch)  Host Computer                   2          0
c   32   317(13Dh)  Predictor                       3          1
c   33   318(13Eh)  White Point                     5         46 (offset)
c   34   319(13Fh)  Primary Chromaticities         5         48 (offset)
```

```
c  35  320(140h)  Color Map                    3        0
c  36  321(141h)  Highlight Shadow             3        0
c  37             UNRECOGNIZED TAG NUMBER               50 (offset)
c
c        Data Types are :  1 = BYTE
c                          2 = ASCII
c                          3 = SHORT (integer*2)
c                          4 = LONG  (integer*4)
c                          5 = RATIONAL (integer*4 / integer*4)
c
c        Note: the values stored as defaults for the RATIONAL tagvalues
c              are 'pointers' to the array location in TAGVALUE where
c              the two integer*4 values are located, i.e., the first
c              integer*4 is put in TAGVALUE(TAGVALUE(ID)) and the second
c              in TAGVALUE(TAGVALUE(ID)+1).
c
c     4. Once read from the TIFF file, certain of the tagvalues are
c        checked to ensure that they are compatible with the present
c        program.  If they are not, or if 2 or more "unknown tags"
c        have been read, the program is aborted and an appropriate
c        error message is displayed.  The critical tagvalues are:
c
c      1  254(0FEh)  New Subfile Type            MUST equal 0
c      2  256(100h)  Image Width (Pixels)        MUST not be .le. 0
c      3  257(101h)  Image Height (Pixels)       MUST not be .le. 0
c      4  258(102h)  Bits per Sample             MUST equal 1, 4, or 8
c      5  259(103h)  Compression                 MUST equal 1
c      6  262(106h)  Photometric Interpretation  MUST equal 0 or 1
c     13  277(115h)  Samples per Pixel           MUST equal 1
c
c=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      integer*1 idots(4,1048),dummy(4096)
      integer*1 black,cr,lf,space
      integer*2 iver,numtags,tagnum,datatype,isum,int2
      integer*2 image(4,1048),datalength(5),tagid(36)
      integer*2 idirect(-127:128,-127:128),inorm(-127:128,-127:128)
      integer*2 ihits(-127:128,-127:128)
      integer*4 ioffset,length,tagdata,tagvalue(51),int4a,int4b
      integer*4 nhits,np,kk
      real*4 m00,m10,m01,ranval
      character*1 answer,chipid1,chipid2,infinity
      character*8 tagtype(5)
      character*26 tagname(37)
      character*54 imagefile,pixelfile,trainlog
c
c
c
      data tagtype/
     +'  Byte  ','  ASCII ','  Short ','  Long  ','Rational'/
      data datalength/1,1,2,4,5/
      data tagid/254,256,257,258,259,262,263,269,270,271,272,
     +273,277,278,279,282,283,284,285,286,287,290,291,
     +292,293,296,297,305,306,315,316,317,318,319,320,321/
      data tagname/'New Subfile Type','Image Width (Pixels)',
     +'Image Height (Pixels)','Bits per Sample','Compression',
```

```
     +'Photometric Interpretation','Thresholding','Document Name',
     +'Image Description','Make','Model','Strip Offsets',
     +'Samples per Pixel','Rows per Strip','Strip Byte Counts',
     +'X Resolution (dpi)','Y Resolution (dpi)',
     +'Planar Configuration','Page Name','X Position','Y Position',
     +'Gray Response Unit','Gray Response Curve','Group 3 Options',
     +'Group 4 Options','Resolution Units','Page Number',
     +'Software','Date & Time','Artist','Host Computer','Predictor',
     +'White Point','Primary Chromaticities','Color Map',
     +'Highlight Shadow','UNRECOGNIZED TAG NUMBER'/
      data tagvalue/0,0,0,1,1,0,1,0,0,0,0,0,1,1,1,38,40,1,0,42,44,0,0,
     +0,0,2,0,0,0,0,0,1,46,48,0,0,50,300,1,300,1,0,0,0,0,0,0,0,0,0,0/
      data cr/13/,lf/10/,space/32/,infinity/236/
c=====================================================================
c
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=====================================================================
    1 format(1h+,20(1h*),' Conversion of TIFF Data to Pixel Data ',
     +20(1h*))
    2 format(1h ,26x,'Version 4.20 : July 22, 1994')
    3 format(1h ,7x,'File : ',a12,'   Gridsize : ',
     +i4,' x ',i4,'   Precision : Real*',i1)
    4 format(1h0,
     +'Enter filename (c/w path & extension) of first TIFF file : ')
    5 format(a1)
    6 format(a54)
    7 format(1h0,18X,
     +'Enter number of image files (default=1) : '\)
    8 format(i3)
    9 format(1h0,
     +'Enter filename (c/w path & extension) of first PIX  file : ')
   11 format(1h0,4x,
     +'Tag',9x,'Tag Name',13x,'Datatype',4x,
     +'Length',4x,'Data  Offset')
   12 format(1h,4x,
     +'===',2x,'==========================',
     +2x,'========',4x,'======',4x,'====  ======')
   13 format(1h ,4x,i3,2x,a26,2x,a8,4x,i4,5x,i5)
   14 format(1h ,4x,i3,2x,a26,2x,a8,4x,i4,12x,i5)
   15 format(1h ,9x,a26,2x,a8,4x,i4,5x,i5)
   16 format(1h ,9x,a26,2x,a8,4x,i4,4x,F6.2)
   20 format(1h ,16X,
     +'Superimpose noise on pixel image (default=no) ? '\)
   21 format(1h ,17X,
     +'Enter NP - no. of random write loops: '\)
   30 format(1h ,2x,'File : ',a12,'   Gridsize : ',
     +i4,' x ',i4,'   Precision : Real*',i1,'   SNR : ',a1)
   51 format(/20(1h*),' Conversion of TIFF Data to Pixel Data ',
     +20(1h*))
   52 format(24x,'Version 4.20 : July 22, 1994')
   53 format(7x,'File : ',a12,'   Gridsize : ',
     +i4,' x ',i4,'   Precision : Real*',i1/)
   54 format(2x,'File : ',a12,'   Gridsize : ',
     +i4,' x ',i4,'   Precision : Real*',i1,'   SNR : ',a1/)
   61 format(4x,
     +'Tag',9x,'Tag Name',13x,'Datatype',4x,
     +'Length',4x,'Data  Offset')
   62 format(4x,
     +'===',2x,'==========================',
```

```
     +2x,'========',4x,'======',4x,'====  ======'/)
    63 format(4x,i3,2x,a26,2x,a8,4x,i4,5x,i5)
    64 format(4x,i3,2x,a26,2x,a8,4x,i4,12x,i5)
    65 format(9x,a26,2x,a8,4x,i4,5x,i5)
    66 format(9x,a26,2x,a8,4x,i4,4x,F6.2)
    70 format(18x,
     +'Pixel shift factors  : ',i5,7x,i5)
    71 format(18X,
     +'Analog shift factors : ',F9.4,3x,F9.4)
    72 format(27X
     +'Scaling Factor = ',F9.4)
    73 format(/9X,
     +'m00 = ',F12.4,'    m10 = ',F12.4,'    m01 = ',F12.4/)
    80 format(1h ,18x,
     +'Pixel shift factors  : ',i5,7x,i5)
    81 format(1h ,18X,
     +'Analog shift factors : ',F9.4,3x,F9.4)
    82 format(1h0,27X
     +'Scaling Factor = ',F9.4)
    83 format(1h0,9X,
     +'m00 = ',F12.4,'    m10 = ',F12.4,'    m01 = ',F12.4)
    85 format(1h0,2X,'NP =',i6,' Number of hits =',i6,
     +' Percentage =',F7.2,' Measured db =',F8.2)
    86 format(/2X,'NP =',i6,' Number of hits =',i6,
     +' Percentage =',F7.2,' Measured db =',F8.2)
    87 format(i7)
    90 format(1h0,13x,
     +'Now have ',i1,' unrecognized tag numbers.  Program aborted.')
    92 format(1h0,14X,
     +'Imagefile name incorrect.  Must have .tif extension.')
    93 format(1h0,25X,
     +'Hit any key to restart program.')
    94 format(1h ,8X,'Fatal error in one or more ',
     +'tagvalues above (1,2,3,4,5,6, or 13).')
    95 format(1h ,11X,'Program aborted.  Correct image tagvalue ',
     +'and rerun program.')
    96 format(1h0,10X,'WARNING!  A total of ',i2,
     +' unknown tagnumbers were encountered.')
    98 format(1h0,21X,
     +'A total of ',I3,' file(s) were processed.')
    99 format(1h0,21X,
     +'Program completed.  Normal termination.')
c=====================================================================
c
c     Initialize parameters and then prompt user for filenames and
c     number of files.
c
c=====================================================================
      ifile=0
      ierr=0
      igridsize=256
      igsp2=igridsize+2
      N=128
      del=1./float(N)
      beta=float((igridsize*igridsize)/8)
      iprecision=1
      iunknown=0
c
c
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
```

```
      write(0,4)
      read(0,6) imagefile
      write(0,9)
      read(0,6) pixelfile
      do 102 i=1,54
      trainlog(i:i)=pixelfile(i:i)
  102 continue
      i=0
  104 i=i+1
      if(imagefile(i:i).eq.'.') go to 106
      if(i.eq.51) go to 910
      go to 104
  106 mark1=i
      i=0
  108 i=i+1
      if(pixelfile(i:i).eq.'.') go to 110
      if(i.eq.51) go to 910
      go to 108
  110 mark2=i
      init1=1
      do 112 i=1,mark1+3
      if(imagefile(i:i).eq.'\') init1=i+1
  112 continue
      init2=1
      do 114 i=1,mark2+3
      if(pixelfile(i:i).eq.'\') init2=i+1
  114 continue
      trainlog(init2:init2+11)='pixelize.log'
      open(3,file=trainlog,status='unknown',form='formatted')
      write(0,7)
      read(0,8) number
      if(number.le.0) number=1
      noise=0
      write(0,20)
      read(0,5) answer
      if(answer.eq.'y'.or.answer.eq.'Y') noise=1
      if(noise.eq.0) go to 125
      write(0,21)
      read(0,87) np
  125 istart=100*(ichar(imagefile(mark1-3:mark1-3))-48)
      istart=istart+10*(ichar(imagefile(mark1-2:mark1-2))-48)
      istart=istart+(ichar(imagefile(mark1-1:mark1-1))-48)
      ifinish=istart+number-1
      istartp=100*(ichar(pixelfile(mark2-3:mark2-3))-48)
      istartp=istartp+10*(ichar(pixelfile(mark2-2:mark2-2))-48)
      istartp=istartp+(ichar(pixelfile(mark2-1:mark2-1))-48)
      ifinishp=istartp+number-1
      index=istart-1
  130 index=index+1
      if(index.gt.ifinish) go to 990
      nhuns=index/100
      ntens=(index-100*nhuns)/10
      nunits=index-100*nhuns-10*ntens
      imagefile(mark1-3:mark1-3)=char(nhuns+48)
      imagefile(mark1-2:mark1-2)=char(ntens+48)
      imagefile(mark1-1:mark1-1)=char(nunits+48)
      indexp=index+istartp-1
      nhunsp=indexp/100
      ntensp=(indexp-100*nhunsp)/10
      nunitsp=indexp-100*nhunsp-10*ntensp
      pixelfile(mark2-3:mark2-3)=char(nhunsp+48)
```

```
      pixelfile(mark2-2:mark2-2)=char(ntensp+48)
      pixelfile(mark2-1:mark2-1)=char(nunitsp+48)
      call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      if(noise.eq.0) write(0,30) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision),infinity
      if(noise.eq.1) write(0,3) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision)
      write(3,51)
      write(3,52)
      if(noise.eq.0) write(3,54) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision),infinity
      if(noise.eq.1) write(3,53) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision)
      open(2,file=imagefile,status='unknown',form='binary')
      read(2) chipid1,chipid2,iver,ioffset
      rewind 2
      read(2) (dummy(i),i=1,ioffset)
      read(2) numtags
c=======================================================================
c
c     At this point, have the fixed image header data plus the number
c     of tags in the first IFD.  File has been rewound and then
c     positioned at the first tag number given by the value 'ioffset'.
c     Proceed to read and interpret the tags.
c
c=======================================================================
      write(0,11)
      write(0,12)
      write(3,61)
      write(3,62)
      do 200 i=1,numtags
      read(2) tagnum,datatype,length,tagdata
      id=37
      do 135 j=1,36
      if(tagnum.eq.tagid(j)) id=j
  135 continue
      if(id.eq.37) iunknown=iunknown+1
      numbytes=length*datalength(datatype)
      if(numbytes.gt.4) go to 140
      write(0,13) tagnum,tagname(id),tagtype(datatype),length,tagdata
      write(3,63) tagnum,tagname(id),tagtype(datatype),length,tagdata
      tagvalue(id)=tagdata
      go to 200
  140 write(0,14) tagnum,tagname(id),tagtype(datatype),length,tagdata
      write(3,64) tagnum,tagname(id),tagtype(datatype),length,tagdata
      rewind 2
      read(2) (dummy(j),j=1,tagdata)
      if(datatype.le.2) go to 170
      if(datatype.gt.3) go to 150
      read(2) int2
      tagdata=int2
      tagvalue(id)=tagdata
      write(0,15) tagname(id),tagtype(datatype),length,tagdata
      write(3,65) tagname(id),tagtype(datatype),length,tagdata
      go to 170
  150 if(datatype.gt.4) go to 160
      read(2) int4a
      tagdata=int4a
      tagvalue(id)=tagdata
```

```
      write(0,15) tagname(id),tagtype(datatype),length,tagdata
      write(3,65) tagname(id),tagtype(datatype),length,tagdata
      go to 170
  160 read(2) int4a,int4b
      fdata=float(int4a)/float(int4b)
      tagvalue(tagvalue(id))=int4a
      tagvalue(tagvalue(id)+1)=int4b
      write(0,16) tagname(id),tagtype(datatype),length,fdata
      write(3,66) tagname(id),tagtype(datatype),length,fdata
  170 rewind 2
      read(2) (dummy(j),j=1,(ioffset+2+12*i))
  200 continue
      rewind 2
      read(2) (dummy(j),j=1,tagvalue(12))
      black=2**(tagvalue(4)-1)
      ipack=8/tagvalue(4)
      nbytes=tagvalue(2)/ipack
      if((tagvalue(2)-ipack*nbytes).gt.0) nbytes=nbytes+1
      isample=tagvalue(2)/igridsize
      if(isample.lt.1) isample=1
      ndots=isample*igridsize
      ierr=4
      if(tagvalue(4).eq.1.or.tagvalue(4).eq.2) ierr=0
      if(tagvalue(4).eq.4.or.tagvalue(4).eq.8) ierr=0
      if(tagvalue(1).ne.0) ierr=1
      if(tagvalue(2).le.0) ierr=2
      if(tagvalue(3).le.0) ierr=3
      if(tagvalue(5).ne.1) ierr=5
      if(tagvalue(6).ne.0.and.tagvalue(6).ne.1) ierr=6
      if(tagvalue(13).ne.1) ierr=13
      if(ierr.gt.0) go to 920
C=====================================================================
C
C
C     All the tags have been read and deciphered and the TIFF file has
C     been rewound and repositioned to point at the beginning of the
C     raster image data.
C     Begin reading the TIFF data, converting it to byte values with
C     white=0, and 'averaging' the image(i,j) over isample x isample,
C     saving as idirect(i,j).
C
C     NOTE: The formulas relating the counting integers i,j with x,y are:
C
C     x = i*del - 0.5*del  ; i = -N+1 to N
C     y = j*del - 0.5*del  ; j = -N+1 to N
C
C=====================================================================
      nn=0
  250 nn=nn+1
      if(nn.gt.igridsize) go to 500
      do 400 i=1,isample
      read(2) (idots(i,j),j=1,nbytes)
      if(tagvalue(6).eq.0) go to 310
      do 300 j=1,nbytes
      idots(i,j)=NOT(idots(i,j))
  300 continue
  310 if(tagvalue(4).eq.1) go to 350
      if(tagvalue(4).eq.4) go to 330
      do 320 j=1,ndots,ipack
      image(i,j)=idots(i,j)
      if(idots(i,j).lt.0) image(i,j)=idots(i,j)+256
  320 continue
```

```
      go to 400
  330 do 340 j=1,ndots,ipack
      jj=1+(j/ipack)
      image(i,j)=ishl((iand(idots(i,jj),2#11110000)),-4)
      image(i,j+1)=iand(idots(i,jj),2#00001111)
  340 continue
      go to 400
  350 do 360 j=1,ndots,ipack
      jj=1+(j/ipack)
      image(i,j)=ishl((iand(idots(i,jj),2#10000000)),-7)
      image(i,j+1)=ishl((iand(idots(i,jj),2#01000000)),-6)
      image(i,j+2)=ishl((iand(idots(i,jj),2#00100000)),-5)
      image(i,j+3)=ishl((iand(idots(i,jj),2#00010000)),-4)
      image(i,j+4)=ishl((iand(idots(i,jj),2#00001000)),-3)
      image(i,j+5)=ishl((iand(idots(i,jj),2#00000100)),-2)
      image(i,j+6)=ishl((iand(idots(i,jj),2#00000010)),-1)
      image(i,j+7)=iand(idots(i,jj),2#00000001)
  360 continue
  400 continue
      mm=0
  420 mm=mm+1
      if(mm.gt.igridsize) go to 250
      isum=0
      do 460 i=1,isample
      do 440 j=1,isample
      isum=isum+image(i,j+isample*(mm-1))
  440 continue
  460 continue
      iarea=isample*isample
      ibase=isum/iarea
      irem=isum-iarea*ibase
      idiv=iarea/2
      if(iarea-2*(iarea/2).eq.1) idiv=idiv+1
      idirect(mm-N,-nn+N+1)=ibase+(irem/idiv)
      go to 420
  500 close(2)
c=======================================================================
c
c     Calculate the m00, m10, and m01 values for the original image.
c
c=======================================================================
      m00=0.
      m10=0.
      m01=0.
      do 540 i=-N+1,N
      do 520 j=-N+1,N
      fdir=float(idirect(i,j))
      m00=m00+fdir
      m10=m10+del*(float(i)-.5)*fdir
      m01=m01+del*(float(j)-.5)*fdir
  520 continue
  540 continue
      call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      if(noise.eq.0) write(0,30) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision),infinity
      if(noise.eq.1) write(0,3) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision)
      write(3,51)
      write(3,52)
```

```
      if(noise.eq.0) write(3,54) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision),infinity
      if(noise.eq.1) write(3,53) imagefile(init1:init1+11),
     +igridsize,igridsize,(4*iprecision)
      write(0,83) m00,m10,m01
      write(3,73) m00,m10,m01
c=======================================================================
c
c     Calculate the scaling factor 'a' and the first-order shiftx and
c     shifty values.  Do the shifting first and then scale the image
c     to produce a value of m00 = beta.
c
c=======================================================================
  600 a=sqrt(m00/beta)
      bx=m10/m00
      cx=(bx/del)
      ixshift=nint(cx)
      by=m01/m00
      cy=(by/del)
      iyshift=nint(cy)
      do 620 i=-N+1,N
      do 610 j=-N+1,N
      inorm(i,j)=0.
  610 continue
  620 continue
      do 640 i=-N+1,N
      ii=i+ixshift
      if(ii.lt.-127.or.ii.gt.128) go to 640
      do 630 j=-N+1,N
      jj=j+iyshift
      if(jj.lt.-127.or.jj.gt.128) go to 630
      inorm(i,j)=idirect(ii,jj)
  630 continue
  640 continue
      do 646 i=-N+1,N
      do 644 j=-N+1,N
      ihits(i,j)=0
      idirect(i,j)=inorm(i,j)
  644 continue
  646 continue
      do 660 i=-N+1,N
      ii=NINT(a*(float(i)-0.5)+0.5)
      if(ii.lt.-127.or.ii.gt.128) go to 660
      do 650 j=-N+1,N
      jj=NINT(a*(float(j)-0.5)+0.5)
      if(jj.lt.-127.or.jj.gt.128) go to 650
      inorm(i,j)=idirect(ii,jj)
  650 continue
  660 continue
c=======================================================================
c
c
c     Original image has now been shifted (once) and scaled.
c     Recalculate the required shift values and perform the image shift,
c     repeating until m10 and m01 are minimized (ideally, both = 0).
c
c=======================================================================
      m00=0.
      m10=0.
      m01=0.
      do 692 i=-N+1,N
      do 690 j=-N+1,N
```

```
      fnor=float(inorm(i,j))
      m00=m00+fnor
      m10=m10+del*(float(i)-.5)*fnor
      m01=m01+del*(float(j)-.5)*fnor
  690 continue
  692 continue
      ixshift2=nint(m10/(m00*del))
      iyshift2=nint(m01/(m00*del))
      write(0,82) (1./a)
      write(0,81) cx,cy
      write(0,80) ixshift,iyshift
      write(0,80) ixshift2,iyshift2
      write(3,72) a
      write(3,71) bx,by
      write(3,70) ixshift,iyshift
      write(3,70) ixshift2,iyshift2
      if(ixshift2.eq.0.and.iyshift2.eq.0) go to 800
  700 do 720 i=-N+1,N
      do 710 j=-N+1,N
      idirect(i,j)=inorm(i,j)
  710 continue
  720 continue
      do 740 i=-N+1,N
      ii=i+ixshift2
      if(ii.lt.-127.or.ii.gt.128) go to 740
      do 730 j=-N+1,N
      jj=j+iyshift2
      if(jj.lt.-127.or.jj.gt.128) go to 730
      inorm(i,j)=idirect(ii,jj)
  730 continue
  740 continue
      m00=0.
      m10=0.
      m01=0.
      do 792 i=-N+1,N
      do 790 j=-N+1,N
      fnor=float(inorm(i,j))
      m00=m00+fnor
      m10=m10+del*(float(i)-.5)*fnor
      m01=m01+del*(float(j)-.5)*fnor
  790 continue
  792 continue
      ixshift2=nint(m10/(m00*del))
      iyshift2=nint(m01/(m00*del))
      write(0,80) ixshift2,iyshift2
      write(0,83) m00,m10,m01
      write(3,70) ixshift2,iyshift2
      write(3,73) m00,m10,m01
      if(ixshift2.eq.0.and.iyshift2.eq.0) go to 802
      go to 700
c=======================================================================
c
c     Write the final values for m00, m10, and m01, and then, if elected
c     as an option, calculate the noise pixels using the random number
c     generator RAN2(IDUM) (from "Numerical Recipes").  Then write the
c     pixelfile to be used in the feature vector calculations along with
c     a 'text' version of the same (the latter very useful for quick
c     printing/viewing by a word processor).
c
c     Consistent with all the other programs comprising this "chain",
c     the pixelfile is written from left to right, beginning with the
```

```
c     upper left corner of the image, i.e., the image is written as
c     rows x=-1 to x=+1 (i=-N+1 to N) from y=+1 (j=N) down to y=-1
c     (j=-N+1).
c
c=======================================================================
  800 write(0,83) m00,m10,m01
      write(3,73) m00,m10,m01
  802 if(noise.eq.0) go to 875
      idum=-7
      do 850 kk=1,np
      ranval=2*ran2(idum)-1.
      i=int(float(N)*ranval+.5)
      if(i.lt.(-N+1)) i=-N+1
      if(i.gt.N) i=N
      ranval=2*ran2(idum)-1.
      j=int(float(N)*ranval+.5)
      if(j.lt.(-N+1)) j=-N+1
      if(j.gt.N) j=N
      ihits(i,j)=ihits(i,j)+1
  850 continue
      nhits=0
      do 860 i=-N+1,N
      do 855 j=-N+1,N
      k=ihits(i,j)
      imod2=k-2*(k/2)
      if(imod2.eq.0) go to 855
      nhits=nhits+1
      if(inorm(i,j).eq.0) ichange=1
      if(inorm(i,j).eq.1) ichange=0
      inorm(i,j)=ichange
  855 continue
  860 continue
      percent=100.*float(nhits)/float(4*N*N)
      db=20.*alog10(-1+float(4*N*N)/float(nhits))
      write(0,85) np,nhits,percent,db
      write(3,86) np,nhits,percent,db
c=======================================================================
c
c     Write the 'pixel' file.
c
c=======================================================================
  875 open(2,file=pixelfile,status='unknown',form='binary')
      do 880 j=N,-N+1,-1
      write(2) (inorm(i,j),i=-N+1,N)
  880 continue
      ifile=ifile+1
      endfile 2
  882 if(N.ge.0) go to 899
c=======================================================================
c
c     The code immediately below is used to write an ASCII file which
c     can be read, displayed, and printed by WordPerfect, i.e., a QAD
c     way to verify the program's operation.  The statement 882 above
c     should be commented out if these ASCII files are to be written.
c     Although QAD, the images displayed in this way, when suitable
c     graphics characters are used, are indeed accurate representations
c     of the images.  The code below offers two possibilities: (1), put
c     the numerals 0 through 9 for black pixels to represent their
c     grayscale values, or, (2), put a graphics symbol (extended ASCII)
c     which corresponds to a printed black pixel.
c
```

```
c=====================================================================
      pixelfile(mark2+1:mark2+3)='txt'
      open(2,file=pixelfile,status='unknown',form='binary')
      idiv=1
      if(ipack.eq.2) idiv=4
      if(ipack.eq.1) idiv=32
      do 883 i=1,igsp2
      dummy(i)=-6
  883 continue
      write(2) (dummy(i),i=1,igsp2)
      write(2) cr,lf
      iblackpixel=48
c     iblackpixel=218
      do 890 j=N,-N+1,-1
      do 885 i=-N+1,N,+1
      if(iblackpixel.eq.48) dummy(i+N+1)=iblackpixel+(inorm(i,j)/idiv)
      if(iblackpixel.eq.218) dummy(i+N+1)=iblackpixel
      if(inorm(i,j).eq.0) dummy(i+N+1)=space
  885 continue
      write(2) (dummy(i),i=1,igsp2)
      write(2) cr,lf
  890 continue
      do 892 i=1,igsp2
      dummy(i)=-6
  892 continue
      write(2) (dummy(i),i=1,igsp2)
      write(2) cr,lf
      endfile 2
      pixelfile(mark2+1:mark2+3)='pix'
  899 go to 130
c=====================================================================
c
c     Error messages.  Program execution will be aborted if an unknown
c     tag number is encountered.
c
c=====================================================================
  900 write(0,90) iunknown
      go to 1000
  910 write(0,92)
      write(0,93)
      read(0,5) answer
      go to 100
  920 write(0,94)
      write(0,95)
      read(0,5) answer
      go to 1000
  990 if(iunknown.gt.0) write(0,96) iunknown
      write(0,98) ifile
      write(0,99)
c
c
 1000 continue
      endfile 3
      end
c=====================================================================
c
c     Random number generator from section 7.1 of "Numerial Recipes :
c     Fortran".
c
c=====================================================================
      FUNCTION RAN2(IDUM)
```

```
      PARAMETER (M=714025,IA=1366,IC=150889,RM=1.4005112E-6)
      DIMENSION IR(97)
      DATA IFF /0/
      IF(IDUM.LT.0.OR.IFF.EQ.0)THEN
        IFF=1
        IDUM=MOD(IC-IDUM,M)
        DO 11 J=1,97
          IDUM=MOD(IA*IDUM+IC,M)
          IR(J)=IDUM
11      CONTINUE
        IDUM=MOD(IA*IDUM+IC,M)
        IY=IDUM
      ENDIF
      J=1+(97*IY)/M
      IF(J.GT.97.OR.J.LT.1) PAUSE
      IY=IR(J)
      RAN2=IY*RM
      IDUM=MOD(IA*IDUM+IC,M)
      IR(J)=IDUM
      RETURN
      END
C=======================================================================
```

```
C=====================================================================
C     Program : GENFVZER.FOR (GENerate Feature Vector for ZERnike basis)
C     Version : 3.4  June 21, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        GENFVZER (GENerate Feature Vectors) calculates the
C        feature vectors for a series of .pix files w.r.t. the
C        Zernike functions (previously calculated and stored as files).
C        The features are calculated on a square grid of dimension
C        2N x 2N in either single or double precision.
C        The resulting file is saved as a FASCII file in the format
C        correct for use by NeuralWare Professional II (a .nna file).
C
C
C     PROGRAMMING NOTES:
C
C     1. The formats for the various data files are specified
C        separately in those programs which generate them (ZERNIKE,
C        PIXELIZE, ...).
C
C     2. The gridsize is NOT a variable for this program.  However,
C        some care has been taken to allow the user to change this
C        parameter relatively easily.  Only the dimensioning
C        assignments, initialization values, and some format
C        statements would have to be altered to allow for a different
C        gridsize.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 basis(-127:128,-127:128),fv(1100,100)
      integer*2 pixel(-127:128,-127:128),kchar(3)
      character*1 answer,cr,lf,drive
      character*3 bufint
      character*54 pixelfile,basisfile,fvfile,buffer
      data cr/13/,lf/10/
C=====================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,17(1h*),
     +' Rotation Invariant Feature Vector Generation ',17(1h*)\)
    2 format(1h ,25x,'Version 3.40 : June 21, 1995')
    3 format(1h ,27x,'Grid Size : ',i4,' x ',i4)
    4 format(1h0,'Enter filename ',
     +'(c/w path & extension) of starting .pix  file : ')
   44 format(1h0,'Enter filename ',
     +'(c/w path & extension) of feature vector file : ')
```

```
    5 format(a1)
    6 format(a54)
    7 format(1h0,18X,
     +'Enter number of pixel files (default=1) : '\)
    8 format(i3)
    9 format(1h0,18x,'Initial Principal Index Value (NO default) = '\)
   10 format(1h ,18x,' Final  Principal Index Value (NO default) = '\)
   11 format(1h0,'Drive letter location of basis functions
     + (ONE character) : ')
   20 format(1h0,26x,
     +'Choice of basis functions :')
   21 format(1h ,12X,
     +'Pseudo Zernike Polynomials                              (1)')
   22 format(1h ,12X,
     +'Standard Zernike Polynomials                            (2)')
   23 format(1h ,12X,
     +'Pseudo Zernike Polynomials   - Radial Portion Only      (3)')
   24 format(1h ,12X,
     +'Standard Zernike Polynomials - Radial Portion Only      (4)')
   29 format(1h0,16X,
     +'Indicate basis function choice (NO default) : '\)
   31 format(1h+,14(1h*),
     +' Feature Vector using ',
     +'the Pseudo Zernike Polynomials ',13(1h*)\)
   32 format(1h+,13(1h*),
     +' Feature Vector using ',
     +'the Standard Zernike Polynomials ',12(1h*)\)
   40 format(1h ,11x,
     +'Now integrating ',a12,' with ',a12,' [n=',i2,',m=',i2,']')
   42 format(1h0,24x,'Program  completed  successfully.')
   43 format(1h ,20x,'Total of ',i4,' pixel files were processed.')
   50 format(F12.5,',',\)
   51 format(2a1\)
   90 format(1h0,
     +'The index value is negative.')
   91 format(1h0,
     +'The final index value is less than the initial value.')
   92 format(1h0,
     +'The secondary index value exceeds the principal index value.')
   93 format(1h0,
     +'The two indices must be the same parity.')
   94 format(1h0,
     +'Secondary index position is outside permissible range.')
   99 format(1h0,'Hit <RET> to restart program. '\)
C=====================================================================
C
C      Begin by prompting user for the initial pixel image filename (c/w
C      path if necessary), the number of image files, the choice of basis
C      functions, and the range of basis function parameters (in effect,
C      the dimension of the feature vector).
C
C=====================================================================
      igridsize=256
      N=128
      itype=1
      irco=0
      del=1.D0/dfloat(N)
      zero=0.D0
      half=0.5D0
      one=1.D0
  100 call clearscreen ($GCLEARSCREEN)
```

```
      mall=1
      ifile=0
      write(0,1)
      write(0,2)
      write(0,3) igridsize,igridsize
c=======================================================================
c
c       Check at this point to see if the "prompts" have been passed
c       to this program as command line parameters.  If so, skip the
c       prompts and proceed to the calculations.
c
c       Order and specification of the command line prompts are:
c
c       arg1 = pixelfile (a54 c/w path if needed)
c       arg2 = fvfile (a54 c/w path if needed)
c       arg3 = drive letter for basis functions (a1)
c       arg4 = itype (integer = 1,2,3,4)
c       arg5 = number (number of pixel files)
c       arg6 = nstart (starting index value)
c       arg7 = nfinish (ending index value)
c
c=======================================================================
      numargs=nargs()
      if(numargs.gt.1) ibatch=1
      if(ibatch.eq.0) go to 120
      call getarg(1,pixelfile,istatus)
      call getarg(2,fvfile,istatus)
      call getarg(3,buffer,istatus)
      drive=buffer(1:1)
      bufint='    '
      call getarg(4,bufint,istatus)
      nsf=3
      do 102 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 102
      kchar(i)=0
      nsf=nsf-1
  102 continue
      itype=0
      do 104 i=1,nsf
      itype=10*itype+kchar(4-i)
  104 continue
      bufint='    '
      call getarg(5,bufint,istatus)
      nsf=3
      do 106 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 106
      kchar(i)=0
      nsf=nsf-1
  106 continue
      number=0
      do 108 i=1,nsf
      number=10*number+kchar(4-i)
  108 continue
      bufint='    '
      call getarg(6,bufint,istatus)
      nsf=3
      do 110 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 110
```

```
      kchar(i)=0
      nsf=nsf-1
  110 continue
      nstart=0
      do 112 i=1,nsf
      nstart=nstart*10+kchar(4-i)
  112 continue
      bufint='    '
      call getarg(7,bufint,istatus)
      nsf=3
      do 114 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 114
      kchar(i)=0
      nsf=nsf-1
  114 continue
      nfinish=0
      do 116 i=1,nsf
      nfinish=nfinish*10+kchar(4-i)
  116 continue
      go to 130
C=======================================================================
C
C       We come here only if ibatch=0, i.e., no command line
C       parameters have been entered and the user must be prompted
C       for each of the filenames and variables.
C
C=======================================================================
  120 write(0,4)
      read(0,6) pixelfile
      write(0,44)
      read(0,6) fvfile
      write(0,11)
      read(0,5) drive
      write(0,7)
      read(0,8) number
      if(number.le.0) number=1
      write(0,20)
      write(0,21)
      write(0,22)
      write(0,23)
      write(0,24)
      write(0,29)
      read(0,8) itype
      write(0,9)
      read(0,8) nstart
      write(0,10)
      read(0,8) nfinish
      if(nstart.lt.0) go to 900
C
C
  130 if(itype.gt.2) irco=1
      if(itype.gt.2) itype=itype-2
      basisfile(1:54)='X:\PZZ\PZ_nnmm.bin'
      if(itype.eq.2) basisfile(8:8)='S'
      if(itype.eq.2) basisfile(4:4)='S'
      if(irco.eq.1) basisfile(10:10)='C'
      if(irco.eq.1) basisfile(6:6)='C'
      if(irco.eq.0) basisfile(6:6)='F'
      basisfile(1:1)=drive
      i=0
```

```
  134 i=i+1
      if(pixelfile(i:i).eq.'.') go to 136
      if(i.eq.51) go to 910
      go to 134
  136 markpix=i
      initpix=1
      do 140 i=1,markpix+3
      if(pixelfile(i:i).eq.'\') initpix=i+1
  140 continue
      i=0
  144 i=i+1
      if(fvfile(i:i).eq.'.') go to 146
      if(i.eq.51) go to 910
      go to 144
  146 markfv=i
      initfv=1
      do 150 i=1,markfv+3
      if(fvfile(i:i).eq.'\') initfv=i+1
  150 continue
      ipbeg=nstart-2*(nstart/2)
      ipfin=nfinish-2*(nfinish/2)
      if((nfinish-nstart).1t.0) go to 910
      if(itype.eq.2.or.itype.eq.4) go to 170
      isub=(nstart*(nstart+1))/2
      idim=(((nfinish+1)*(nfinish+2))/2)-isub
      if(nfinish.eq.9) idim=45
      go to 180
  170 L=(nfinish+1)/2
      LSUB=nstart/2
      if(ipbeg.eq.0) isub=LSUB*(LSUB+1)
      if(ipbeg.eq.1) isub=(LSUB+1)*(LSUB+1)
      if(ipfin.eq.0) idim=(L+1)*(L+1)-isub
      if(ipfin.eq.1) idim=L*(L+1)-isub
  180 istart=100*(ichar(pixelfile(markpix-3:markpix-3))-48)
      istart=istart+10*(ichar(pixelfile(markpix-2:markpix-2))-48)
      istart=istart+(ichar(pixelfile(markpix-1:markpix-1))-48)
      ifinish=istart+number-1
      index=istart-1
      irow=0
  190 index=index+1
      irow=irow+1
      icol=0
      if(index.gt.ifinish) go to 1200
      mstart=0
      if(itype.eq.2.or.itype.eq.4) mstart=ipbeg
      mfinish=nstart
      nhuns=index/100
      ntens=(index-100*nhuns)/10
      nunits=index-100*nhuns-10*ntens
      pixelfile(markpix-3:markpix-3)=char(nhuns+48)
      pixelfile(markpix-2:markpix-2)=char(ntens+48)
      pixelfile(markpix-1:markpix-1)=char(nunits+48)
      open(2,file=pixelfile,status='unknown',form='binary')
      do 196 j=N,-N+1,-1
      read(2) (pixel(i,j),i=-N+1,N)
  196 continue
      ifile=ifile+1
      endfile 2
c=======================================================================
c
c     Have read pixel file into array pixel(i,j).  Now read the basis
```

```
c     function files and perform the 'integration'.
c
c=====================================================================
      call clearscreen ($GCLEARSCREEN)
      if(itype.eq.1) write(0,31)
      if(itype.eq.2) write(0,32)
      write(0,2)
      write(0,3) igridsize,igridsize
c=====================================================================
c
c     We now have specific starting values for n and m.
c
c=====================================================================
  200 nindex=nstart-1
      mindex=mstart-itype
  210 nindex=nindex+1
      if(nindex.eq.9.and.itype.eq.1) mfinish=5
      if(nindex.gt.nfinish) go to 190
  220 mindex=mindex+itype
      if(mindex.gt.mfinish) go to 210
      icol=icol+1
c=====================================================================
c
c     We are now ready to proceed with the actual calculations of the
c     feature vector components.
c
c=====================================================================
      ntens=nindex/10
      nunit=nindex-10*ntens
      mtens=mindex/10
      munit=mindex-10*mtens
      basisfile(11:11)=char(ntens+48)
      basisfile(12:12)=char(nunit+48)
      basisfile(13:13)=char(mtens+48)
      basisfile(14:14)=char(munit+48)
      basisfile(10:10)='R'
      if(irco.eq.1) basisfile(10:10)='C'
      write(0,40) pixelfile(initpix:initpix+11),basisfile(8:19),
     +nindex,mindex
      open(2,file=basisfile,status='unknown',form='binary')
      do 300 j=N,-N+1,-1
      read(2) (basis(i,j),i=-N+1,N)
  300 continue
      endfile 2
  310 sumr=0.0
      do 330 j=-N+1,N
      do 320 i=-N+1,N
      if(basis(i,j).eq.0.0.or.pixel(i,j).eq.0) go to 320
      sumr=sumr+basis(i,j)*float(pixel(i,j))
  320 continue
  330 continue
c
c

      if(mindex.eq.0.or.irco.eq.1) go to 600
      basisfile(10:10)='I'
      write(0,40) pixelfile(initpix:initpix+11),basisfile(8:19),
     +nindex,mindex
      open(2,file=basisfile,status='unknown',form='binary')
      do 350 j=N,-N+1,-1
      read(2) (basis(i,j),i=-N+1,N)
  350 continue
```

```
      endfile 2
  360 sumi=0.0
      do 380 j=-N+1,N
      do 370 i=-N+1,N
      if(basis(i,j).eq.0.0.or.pixel(i,j).eq.0) go to 370
      sumi=sumi+basis(i,j)*float(pixel(i,j))
  370 continue
  380 continue
c
c
  600 if(mindex.eq.0.or.irco.eq.1) fv(irow,icol)=abs(sumr)
      if(mindex.gt.0.and.irco.eq.0) fv(irow,icol)=sqrt(sumr**2+sumi**2)
      if(mindex.lt.mfinish) go to 220
      mindex=0
      ip=nindex+1-2*((nindex+1)/2)
      if(itype.eq.2) mindex=ip
      mfinish=nindex+1
      mindex=mindex-itype
      go to 210
c=======================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=======================================================================
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
  920 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
  930 write(0,93)
      write(0,99)
      read(0,5) answer
      go to 100
  940 write(0,94)
      write(0,99)
      read(0,5) answer
      go to 100
c
c
 1200 open(2,file=fvfile,status='unknown',form='formatted')
      do 1260 i=1,number
      do 1250 j=1,idim
      write(2,50) fv(i,j)
 1250 continue
      write(2,51) cr,lf
 1260 continue
      endfile 2
      write(0,42)
      write(0,43) ifile
      end
c=======================================================================
```

```
C=====================================================================
C
C     Program : GENFVWAL.FOR (GENerate Feature Vectors for WALsh basis)
C     Version : 1.1  June 9, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        GENFVWAL (GENerate Feature Vectors) calculates the
C        feature vectors for a series of .pix files w.r.t. a set of
C        basis functions (previously calculated and stored as files).
C        A choice of either Walsh or Haar functions is offered as
C        the basis functions.
C        The features are calculated on a square grid of dimension
C        2N x 2N in either single or double precision.
C        The resulting file is saved as a FASCII file in the format
C        correct for use by NeuralWare Professional II (a .nna file).
C
C
C     PROGRAMMING NOTES:
C
C     1. The formats for the various data files are specified
C        separately in those programs which generate them (WALSH,
C        PIXELWAL, ...).
C
C     2. The gridsize is NOT a variable for this program.  However,
C        some care has been taken to allow the user to change this
C        parameter relatively easily.  Only the dimensioning
C        assignments, initialization values, and some format
C        statements would have to be altered to allow for a different
C        gridsize.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 fv(1100,100)
      integer*1 basis(-127:128,-127:128)
      integer*2 pixel(-127:128,-127:128),kchar(3)
      character*1 answer,cr,lf,drive1,null
      character*3 bufint
      character*54 pixelfile,basisfile,fvfile,buffer
      data cr/13/,lf/10/,null/0/
C=====================================================================
C
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,17(1h*),
     +' Rotation Invariant Feature Vector Generation ',17(1h*)\)
    2 format(1h ,25x,'Version 1.10 : June 9, 1995')
    3 format(1h ,28x,'Grid Size : ',i4,' x ',i4)
    4 format(1h0,'Enter filename ',
```

```
     +'(c/w path & extension) of   starting  .pix   file : ')
    44 format(1h0,'Enter filename ',
     +'(c/w path & extension) of raw feature vector file : ')
     5 format(a1)
     6 format(a54)
     7 format(1h0,
     +'Enter number of pixel files (default=1) : '\)
     8 format(i3)
     9 format(1h0,'Initial Index Value (NO default) = '\)
    10 format(1h ,' Final  Index Value (NO default) = '\)
    20 format(1h0,
     +'Choice of basis functions :')
    21 format(1h ,
     +'Radial Walsh Functions                              (1)')
    22 format(1h ,
     +'Radial Haar  Functions                              (2)')
    29 format(1h0,
     +'Indicate basis function choice (NO default) : '\)
    23 format(1h0,'Drive letter location of basis functions
     + (ONE character) : ')
    31 format(1h+,16(1h*),
     +' Feature Vector using ',
     +'the Radial Walsh Functions ',15(1h*)\)
    32 format(1h+,16(1h*),
     +' Feature Vector using ',
     +'the Radial Haar Functions ',16(1h*)\)
    40 format(1h ,15x,
     +'Now integrating ',a12,' with ',a10)
    42 format(1h0,24x,'Program  completed  successfully.')
    43 format(1h ,20x,'Total of ',i4,' pixel files were processed.')
    50 format(F12.5,',','\)
    51 format(2a1\)
    90 format(1h0,
     +'The index value is negative.')
    91 format(1h0,
     +'The final index value is less than the initial value.')
    99 format(1h0,'Hit <RET> to restart program. '\)
c=======================================================================
c
c     Begin by prompting user for the initial pixel image filename (c/w
c     path if necessary), the number of image files, the choice of basis
c     functions, and the range of basis function parameters (in effect,
c     the dimension of the feature vector).
c
c=======================================================================
      igridsize=256
      N=128
      itype=1
      del=1.D0/dfloat(N)
      ibatch=0
  100 call clearscreen ($GCLEARSCREEN)
      ifile=0
      write(0,1)
      write(0,2)
      write(0,3) igridsize,igridsize
c=======================================================================
c
c       Check at this point to see if the "prompts" have been passed
c       to this program as command line parameters.  If so, skip the
c       prompts and proceed to the calculations.
c
```

```
c       Order and specification of the command line prompts are:
c       arg1 = pixelfile (a54 c/w path if needed)
c       arg2 = fvfile (a54 c/w path if needed)
c       arg3 = drive letter for basis functions (a1)
c       arg4 = itype (integer, 1 for Walsh, 2 for Haar)
c       arg5 = number of pixel files (integer)
c       arg6 = starting index value (integer)
c       arg7 = ending index value (integer)
c=======================================================================
      numargs=nargs()
      if(numargs.gt.1) ibatch=1
      if(ibatch.eq.0) go to 120
      call getarg(1,pixelfile,istatus)
      call getarg(2,fvfile,istatus)
      call getarg(3,buffer,istatus)
      drive1=buffer(1:1)
      bufint='    '
      call getarg(4,bufint,istatus)
      nsf=3
      do 102 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 102
      kchar(i)=0
      nsf=nsf-1
  102 continue
      itype=0
      do 104 i=1,nsf
      itype=10*itype+kchar(4-i)
  104 continue
      bufint='    '
      call getarg(5,bufint,istatus)
      nsf=3
      do 106 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 106
      kchar(i)=0
      nsf=nsf-1
  106 continue
      number=0
      do 108 i=1,nsf
      number=10*number+kchar(4-i)
  108 continue
      bufint='    '
      call getarg(6,bufint,istatus)
      nsf=3
      do 110 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 110
      kchar(i)=0
      nsf=nsf-1
  110 continue
      nbegin=0
      do 112 i=1,nsf
      nbegin=nbegin*10+kchar(4-i)
  112 continue
      bufint='    '
      call getarg(7,bufint,istatus)
      nsf=3
      do 114 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
```

```
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 114
      kchar(i)=0
      nsf=nsf-1
  114 continue
      nfinish=0
      do 116 i=1,nsf
      nfinish=nfinish*10+kchar(4-i)
  116 continue
      go to 130
c=====================================================================
c
c       We come here only if ibatch=0, i.e., no command line
c       parameters have been entered and the user must be prompted
c       for each of the filenames and variables.
c
c=====================================================================
  120 write(0,4)
      read(0,6) pixelfile
      write(0,44)
      read(0,6) fvfile
      write(0,20)
      write(0,21)
      write(0,22)
      write(0,29)
      read(0,8) itype
      if(itype.ne.2) itype=1
      write(0,23)
      read(0,5) drive1
      write(0,7)
      read(0,8) number
      if(number.le.0) number=1
      write(0,9)
      read(0,8) nbegin
      write(0,10)
      read(0,8) nfinish
  130 if(itype.eq.1) basisfile(1:54)='x:\walsh\walnnn.bin'
      if(itype.eq.2) basisfile(1:54)='x:\haar\harnnn.bin'
      basisfile(1:1)=drive1
      i=0
  132 i=i+1
      if(pixelfile(i:i).eq.'.') go to 134
      if(i.eq.51) go to 910
      go to 132
  134 markpix=i
      initpix=1
      do 136 i=1,markpix+3
      if(pixelfile(i:i).eq.'\') initpix=i+1
  136 continue
      i=0
  140 i=i+1
      if(fvfile(i:i).eq.'.') go to 142
      if(i.eq.51) go to 910
      go to 140
  142 markfv=i
      initfv=1
      do 144 i=1,markfv+3
      if(fvfile(i:i).eq.'\') initfv=i+1
  144 continue
      idim=nfinish-nbegin+1
      if(nbegin.lt.0) go to 900
      if((nfinish-nbegin).lt.0) go to 910
```

```
      istart=100*(ichar(pixelfile(markpix-3:markpix-3))-48)
      istart=istart+10*(ichar(pixelfile(markpix-2:markpix-2))-48)
      istart=istart+(ichar(pixelfile(markpix-1:markpix-1))-48)
      ifinish=istart+number-1
      index=istart-1
      irow=0
  180 index=index+1
      irow=irow+1
      icol=0
      if(index.gt.ifinish) go to 1200
      nhuns=index/100
      ntens=(index-100*nhuns)/10
      nunits=index-100*nhuns-10*ntens
      pixelfile(markpix-3:markpix-3)=char(nhuns+48)
      pixelfile(markpix-2:markpix-2)=char(ntens+48)
      pixelfile(markpix-1:markpix-1)=char(nunits+48)
      open(2,file=pixelfile,status='unknown',form='binary')
      do 190 j=N,-N+1,-1
      read(2) (pixel(i,j),i=-N+1,N)
  190 continue
      ifile=ifile+1
      endfile 2
C=====================================================================
C
C
C     Have read pixel file into array pixel(i,j).  Now read the basis
C     function files and perform the 'integration'.
C
C=====================================================================
      call clearscreen ($GCLEARSCREEN)
      if(itype.eq.1) write(0,31)
      if(itype.eq.2) write(0,32)
      write(0,2)
C     write(0,3) igridsize,igridsize
C
C
  200 nindex=nbegin-1
  210 nindex=nindex+1
      if(nindex.gt.nfinish) go to 180
      icol=icol+1
C=====================================================================
C
C
C     We are now ready to proceed with the actual calculations of the
C     feature vector components.
C
C=====================================================================
      nhuns=nindex/100
      ntens=(nindex-100*nhuns)/10
      nunit=nindex-100*nhuns-10*ntens
      kk=13
      if(itype.eq.2) kk=12
      basisfile(kk:kk)=char(nhuns+48)
      basisfile(kk+1:kk+1)=char(ntens+48)
      basisfile(kk+2:kk+2)=char(nunit+48)
      write(0,40) pixelfile(initpix:initpix+11),basisfile(kk-3:kk+6)
      open(2,file=basisfile,status='unknown',form='binary')
      do 300 j=N,-N+1,-1
      read(2) (basis(i,j),i=-N+1,N)
  300 continue
      endfile 2
  310 sum=0.0
      do 330 j=-N+1,N
```

```
      do 320 i=-N+1,N
      if(basis(i,j).eq.0.0.or.pixel(i,j).eq.0) go to 320
      sum=sum+float(basis(i,j)*pixel(i,j))
  320 continue
  330 continue
c
c
  600 fv(irow,icol)=abs(sum)
      go to 210
c=====================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=====================================================================
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
c
c
 1200 open(2,file=fvfile,status='unknown',form='formatted')
      do 1260 i=1,number
      do 1250 j=1,idim
      write(2,50) fv(i,j)
 1250 continue
      write(2,51) cr,lf
 1260 continue
      endfile 2
      write(0,42)
      write(0,43) ifile
      end
c=====================================================================
```

```
C=====================================================================
C
C     Program : NORMFV.FOR (NORMalize the Feature Vectors)
C     Version : 4.0  July 08, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C       Normalizes either a training raw feature vector (assumed to be
C       with a filename as  'FVnn.RAW') or a test raw feature vector
C       file (assumed to be with a filename as 'FVnnmm.RAW').  The
C       former case writes the 'statfile' containing the mean, maximum,
C       minimum, and sigma information while the latter reads the
C       statfile and uses these parameters in order to normalize the
C       test set. The maximum and minimum referred to for the statfile
C       are equal to (fvmax-mean) and (mean-fvmin) respectively.  Thus,
C       all four elements of the statfile are .ge. 0.
C
C       This program writes 3 types of normalized feature vector files.
C
C       The first (code letter 'v') is:
C       fvnorm1 = (fv - mean)/max(big1,big2)
C       and is characterized by a zero mean, with a maximum value of
C       +1 (big1.gt.big2) OR a minimum value of -1 (big1.lt.big2)
C
C       The second (code letter 'p') is:
C       fvnorm2 = abs(fvnorm1)
C       and is characterized by a non-zero mean, components which are
C       always .ge. 0, and a maximum value of +1.
C
C       The third (code letter 'b') is:
C       fvnorm3 = [2*fv - (big1-big2-2*mean)]/(big1+big2)
C       and is characterized by always having at least one value of
C       +1 AND one value of -1.
C
C     Note 1:
C       An earlier version of this program produced a .sig file
C       representing 'sigma normalized' vectors, i.e.,
C       fvnorm = (fv - mean)/sigma
C       characterized by a zero mean and unit sigma value.  However,
C       these "normalized" fv often resulted in vectors (for both
C       training and test sets) with components substantially greater
C       than +1 or less than -1.  This meant either "renormalizing" or
C       explicitly setting up "minmax" tables for the neural simulator.
C       Either way, the end result was that a form of fvnorm1 or
C       fvnorm3 was used as input.  Actual tests using the .sig and
C       .max files (fvnorm1) with NO minmax tables revealed that the
C       networks trained with the .max files performed noticably
C       better than those trained with the .sig files.
C
C     Note 2:
C       All the filenames are a12 - this program MUST be run from
C       within the directory containing all the necessary feature
C       vector and stat files.
```

```
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 fv(1100,100),fvnorm1(1100,100),fvnorm2(1100,100)
      real*4 fvnorm3(1100,100)
      real*4 mean(100),sigma(100),big1(100),big2(100)
      integer*2 kchar(3)
      character*1 answer,cr,lf,comma,dummy1,dummy2
      character*3 bufint
      character*12 fvfile,normfile,statfile
      data cr/13/,lf/10/
C=====================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,21(1h*),
     +' Normalization of Feature Vector File ',21(1h*)\)
    2 format(1h ,25x,'Version 4.00 : July 08, 1995')
    3 format(1h0,'Is this a training or a test/recall set ?')
    4 format(1h0,'Enter filename ',
     +'(path & extension) for reading raw feature file       : ')
    7 format(1h0,15X,
     +'Enter number of vectors            (NO default) : '\)
    8 format(1h0,15X,
     +'Enter dimension of feature vectors (NO default) : '\)
    9 format(1h0,'Enter filename ',
     +'(path & extension) for writing mean/maxim/sigma file  : ')
   10 format(1h0,24x,'Program  completed  successfully.')
   11 format(F12.5,a1\)
   12 format(2a1)
   13 format(a1)
   14 format(a12)
   15 format(i3)
   16 format(F12.5,',',\)
   17 format(2a1\)
   18 format(i2)
   19 format(1h0,'Enter filename ',
     +'(path & extension) for reading mean/maxim/sigma file  : ')
   20 format(1h ,'Training Set    - Answer 1 :')
   21 format(1h ,'Test/Recall Set - Answer 2 :')
   22 format(1h0,'Answer (no default) : '\)
   23 format(1h ,'Writing feature vector file ',a12)
   90 format(1h0,
     +'The number of vectors must be a positive integer .gt. 1.')
   91 format(1h0,
     +'The feature vector dimension must be a positive integer.')
   99 format(1h0,'Hit <RET> to restart program. '\)
C
C
C
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
C=====================================================================
C
C       Check at this point to see if the "prompts" have been passed
C       to this program as command line parameters.  If so, skip the
```

```
c       prompts and proceed to the calculations.
c
c       Order and specification of the command line prompts are:
c       arg1 = fvfile (a12)
c       arg2 = statfile (a12)
c       arg3 = itypefv (integer, 1 for train, 2 for test)
c       arg4 = nv (number of feature vectors)
c       arg5 = ndim (dimension of feature vector)
c
c=======================================================================
      ibatch=0
      numargs=nargs()
      if(numargs.gt.1) ibatch=1
      if(ibatch.eq.0) go to 120
      call getarg(1,fvfile,istatus)
      call getarg(2,statfile,istatus)
      bufint='    '
      call getarg(3,bufint,istatus)
      nsf=3
      do 102 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 102
      kchar(i)=0
      nsf=nsf-1
  102 continue
      itypefv=0
      do 104 i=1,nsf
      itypefv=10*itypefv+kchar(4-i)
  104 continue
      bufint='    '
      call getarg(4,bufint,istatus)
      nsf=3
      do 106 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 106
      kchar(i)=0
      nsf=nsf-1
  106 continue
      nv=0
      do 108 i=1,nsf
      nv=10*nv+kchar(4-i)
  108 continue
      bufint='    '
      call getarg(5,bufint,istatus)
      nsf=3
      do 110 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 110
      kchar(i)=0
      nsf=nsf-1
  110 continue
      ndim=0
      do 112 i=1,nsf
      ndim=ndim*10+kchar(4-i)
  112 continue
      go to 200
c=======================================================================
c
c       We come here only if ibatch=0, i.e., no command line
c       parameters have been entered and the user must be prompted
c       for each of the filenames and variables.
```

```
c
c=======================================================================
  120 write(0,3)
      write(0,20)
      write(0,21)
      read(0,18) itypefv
      write(0,4)
      read(0,14) fvfile
      if(itypefv.eq.1) write(0,9)
      if(itypefv.eq.2) write(0,19)
      read(0,14) statfile
      write(0,7)
      read(0,15) nv
      if(nv.le.0) go to 900
      write(0,8)
      read(0,15) ndim
      if(ndim.le.0) go to 910
c
c
  200 normfile(1:12)=fvfile(1:12)
      mark=6
      if(itypefv.eq.2) mark=8
      normfile(mark:mark+2)='max'
      normfile(2:2)='v'
c=======================================================================
c
c       Now read the 'raw' feature vector file noting that both
c       programs GENFVZER and GENFVWAL produce raw feature vector
c       files for which fv(i,j).GE.0 for all values of i,j.
c
c=======================================================================
      open(2,file=fvfile,status='unknown',form='formatted')
      do 220 i=1,nv
      do 210 j=1,ndim
      read(2,11) fv(i,j),comma
  210 continue
      read(2,12) dummy1,dummy2
  220 continue
      close(2)
      if(itypefv.eq.1) go to 300
c
c
  250 open(2,file=statfile,status='unknown',form='formatted')
      do 260 j=1,ndim
      read(2,11) mean(j),comma
  260 continue
      read(2,12) dummy1,dummy2
      do 270 j=1,ndim
      read(2,11) big1(j),comma
  270 continue
      read(2,12) dummy1,dummy2
      do 275 j=1,ndim
      read(2,11) big2(j),comma
  275 continue
      read(2,12) dummy1,dummy2
      do 280 j=1,ndim
      read(2,11) sigma(j),comma
  280 continue
      read(2,12) dummy1,dummy2
      close(2)
      do 298 j=1,ndim
```

```
      do 290 i=1,nv
      big=big1(j)
      if(big2(j).gt.big1(j)) big=big2(j)
      fvnorm1(i,j)=(fv(i,j)-mean(j))/big
      fvnorm2(i,j)=abs(fvnorm1(i,j))
      a=2./(big1(j)+big2(j))
      b=-(big1(j)-big2(j)+2.*mean(j))/(big1(j)+big2(j))
      fvnorm3(i,j)=a*fv(i,j)+b
  290 continue
  298 continue
      go to 400
c
c
  300 do 340 j=1,ndim
      sum=0.
      do 310 i=1,nv
      sum=sum+fv(i,j)
  310 continue
      mean(j)=sum/float(nv)
      sum=0.
      fmax=mean(j)
      fmin=mean(j)
      do 320 i=1,nv
      sum=sum+(fv(i,j)-mean(j))**2
      if(fv(i,j).gt.fmax) fmax=fv(i,j)
      if(fv(i,j).lt.fmin) fmin=fv(i,j)
  320 continue
      sigma(j)=sqrt(sum/float(nv))
      big1(j)=fmax-mean(j)
      big2(j)=mean(j)-fmin
      if(big1(j).le.0.001) big1(j)=1.
      if(big2(j).le.0.001) big2(j)=1.
      big=big1(j)
      if(big2(j).gt.big1(j)) big=big2(j)
      do 330 i=1,nv
      fvnorm1(i,j)=(fv(i,j)-mean(j))/big
      fvnorm2(i,j)=abs(fvnorm1(i,j))
      a=2./(big1(j)+big2(j))
      b=-(big1(j)-big2(j)+2.*mean(j))/(big1(j)+big2(j))
      fvnorm3(i,j)=a*fv(i,j)+b
  330 continue
  340 continue
c
c
  400 normfile(2:2)='v'
      normfile(mark:mark+2)='max'
      open(2,file=normfile,status='unknown',form='formatted')
      do 460 i=1,nv
      do 450 j=1,ndim
      write(2,16) fvnorm1(i,j)
  450 continue
      write(2,17) cr,lf
  460 continue
      endfile 2
c
c
  500 normfile(2:2)='p'
      open(2,file=normfile,status='unknown',form='formatted')
      do 520 i=1,nv
      do 510 j=1,ndim
      write(2,16) fvnorm2(i,j)
```

```
  510 continue
      write(2,17) cr,lf
  520 continue
      endfile 2
c
c
  540 normfile(2:2)='b'
      open(2,file=normfile,status='unknown',form='formatted')
      do 560 i=1,nv
      do 550 j=1,ndim
      write(2,16) fvnorm3(i,j)
  550 continue
      write(2,17) cr,lf
  560 continue
      endfile 2
      if(itypefv.eq.2) go to 1000
c
c
  600 open(2,file=statfile,status='unknown',form='formatted')
      do 620 j=1,ndim
      write(2,16) mean(j)
  620 continue
      write(2,17) cr,lf
      do 640 j=1,ndim
      write(2,16) big1(j)
  640 continue
      write(2,17) cr,lf
      do 650 j=1,ndim
      write(2,16) big2(j)
  650 continue
      write(2,17) cr,lf
      do 660 j=1,ndim
      write(2,16) sigma(j)
  660 continue
      write(2,17) cr,lf
      endfile 2
      go to 1000
c
c
  900 write(0,90)
      write(0,99)
      read(0,13) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,13) answer
      go to 100
 1000 write(0,10)
      end
c=======================================================================
```

```
C=====================================================================
C     Program : GENRV.FOR (GENerate Result Vectors)
C     Version : 1.0  March 11, 1993
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        GENRV (GENerate Result Vectors) writes a set of
C        result vectors corresponding to a series of .pix files
C        generated by GENFV.
C        This program prompts the user to supply the number of
C        vectors NV (rows) and the vector dimension NDIM (columns).
C
C        If "AUTO" mode is selected, a simple matrix is then written
C        consisting of NGROUP groups of (NV/NDIM) rows each, with each
C        group having a single value = 1. (its 'class') and the
C        remainder of the columns = 0.
C
C        If "PROMPT" mode is selected, then the user is prompted to
C        give the position of the vector component = 1 for each feature
C        vector in turn (1 to NV).
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 rv(1100,100)
      character*1 answer,cr,lf
      character*54 rvfile
      data cr/13/,lf/10/
C=====================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,22(1h*),
     +' Generation of a Result Vector File ',22(1h*)\)
    2 format(1h ,25x,'Version 1.00 : March 11, 1993')
    3 format(1h ,8X)
    4 format(1h0,8X,'Enter filename ',
     +'(c/w path & extension) for writing result file : ')
    5 format(a1)
    6 format(a54)
    7 format(1h0,15X,
     +'Enter number of result vectors/cycle (NO default) : '\)
    8 format(1h0,15X,
     +'Enter number of classes of vectors (NO default)   : '\)
    9 format(i3)
   10 format(1h0,24x,'Program  completed  successfully.')
   11 format(F12.5,',',\)
   12 format(2a1\)
   13 format(1h0,15X,
     +'Run prompt or automated mode  (default=Auto)     ? '\)
```

```
   14 format(1h0,15X,
     +'Enter result vector dimension  (NO default)      : '\)
   15 format(1h ,24X,
     +'RV(i,j) = 1 with i =',i3,' and j = '\)
   16 format(1h0,12X,
     +'Enter number of cycles for RV generation (default=1) : '\)
   90 format(1h0,
     +'The number of vectors must be a positive integer.')
   91 format(1h0,
     +'The number of classes must be a positive integer.')
   92 format(1h0,
     +'The number of vectors must be an integral multiple '
     +'of the number of classes.')
   99 format(1h0,'Hit <RET> to restart program. '\)
c=====================================================================
c
c     Begin by prompting user for the initial pixel image filename (c/w
c     path if necessary), the number of image files, the choice of basis
c     functions, and the range of basis function parameters (in effect,
c     the dimension of the feature vector).
c
c=====================================================================
      iprecision=1
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      write(0,4)
      write(0,3)
      read(0,6) rvfile
      write(0,7)
      read(0,9) nv
      if(nv.le.0) go to 900
      write(0,14)
      read(0,9) ndim
      write(0,13)
      do 210 i=1,nv
      do 200 j=1,ndim
      rv(i,j)=0.
  200 continue
  210 continue
      read(0,5) answer
      if(answer.eq.'p'.or.answer.eq.'P') go to 400
      ngroup=nv/ndim
      itest=nv-ndim*ngroup
      if(itest.ne.0) go to 920
      write(0,16)
      read(0,9) ncycle
      if(ncycle.le.0) ncycle=1
      do 320 j=1,ncycle
      do 310 k=1,ndim
      do 300 i=1,ngroup
      ii=(j-1)*nv+(k-1)*ngroup+i
      rv(ii,k)=1.
  300 continue
  310 continue
  320 continue
      go to 1000
c=====================================================================
c
c      Come to line 400 if user wishes to write the result vector by
c      'prompt' mode, i.e., program will write each vector sequentially
```

```
c       prompting the user for the position of the '1' component for
c       each.
c
c=====================================================================
  400 do 440 i=1,nv
      write(0,15) i
      read(0,9) j
      rv(i,j)=1.
  440 continue
      go to 1000
c=====================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=====================================================================
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
  920 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
c
c
 1000 open(2,file=rvfile,status='unknown',form='formatted')
      do 1070 k=1,ncycle
      do 1060 i=1,nv
      do 1050 j=1,ndim
      ii=(k-1)*nv+i
      write(2,11) rv(ii,j)
 1050 continue
      write(2,12) cr,lf
 1060 continue
 1070 continue
      endfile 2
      write(0,10)
      end
c=====================================================================
```

```
c=====================================================================
c     Program : GENCV.FOR (GENerate Complete Vectors)
c     Version : 4.2  July 9, 1995
c     Author  : Kenneth L. Sala
c               Communications Research Center
c               Ottawa, Ontario, Canada
c               (613) 998-2823
c               e-mail  sala@digame.dgcd.doc.ca
c
c
c     Summary:
c
c        GENCV (Combined/Complete Feature Vector) combines a
c        normalized feature vector file (from NORMFV) with its
c        result vector file (from GENRV) to form the complete
c        .nna training or test data file to serve as the input
c        for NeuralWare Professional II simulator. GENCV will
c        generate CXnnmm.NNA files where X = v, p, or b with
c        the corresponding FXnnmm.MAX file as input.  Note, the
c        "differential" file CDnnmm.NNA is generated by using
c        FVnnmm.MAX with CDnnmm.NNA as input file names.
c
c        This program prompts the user to supply the number of
c        vectors NV (rows), the dimension NFVDIM of the feature vector,
c        and the dimension NRVDIM of the result vector, i.e., the
c        feature vector file is dimension NVxNFVDIM while that of
c        the result vector is NVxNRVDIM.
c
c        These two files are then combined to form the single data
c        set of dimension NVx(NFVDIM+NRVDIM), written as an ASCII file
c        with the elements delimited by commas and 'end-of-records'
c        (vectors) by CR,LF.
c
c=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      real*4 fv(1100,200)
      integer*2 kchar(3)
      character*1 answer,cr,lf,comma,dummy1,dummy2
      character*3 bufint
      character*12 fvfile,rvfile,cvfile
      data cr/13/,lf/10/
c=====================================================================
c
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=====================================================================
    1 format(1h+,22(1h*),
     +' Generation of a Result Vector File ',22(1h*)\)
    2 format(1h ,25x,'Version 4.20 : July 09, 1995')
    4 format(1h0,'Enter filename ',
     +'(c/w path & extension) for reading feature file : ')
    5 format(1h0,'Enter filename ',
     +'(c/w path & extension) for reading result file  : ')
    6 format(1h0,'Enter filename ',
     +'(c/w path & extension) for writing the cv file  : ')
```

```
    7 format(1h0,15X,
     +'Enter number of vectors            (NO default) : '\)
    8 format(1h0,15X,
     +'Enter dimension of feature vectors (NO default) : '\)
    9 format(1h0,15X,
     +'Enter dimension of result vectors  (NO default) : '\)
   10 format(1h0,24x,'Program  completed  successfully.')
   11 format(F12.5,a1\)
   12 format(2a1)
   13 format(a1)
   14 format(a12)
   15 format(i3)
   16 format(F12.5,',',\)
   17 format(2a1\)
   18 format(i2)
   90 format(1h0,
     +'The number of vectors must be a positive integer.')
   91 format(1h0,
     +'The feature vector dimension must be a positive integer.')
   92 format(1h0,
     +'The result vector dimension must be a positive integer.')
   99 format(1h0,'Hit <RET> to restart program. '\)
C=====================================================================
C
C     Begin by prompting user for the initial pixel image filename (c/w
C     path if necessary), the number of image files, the choice of basis
C     functions, and the range of basis function parameters (in effect,
C     the dimension of the feature vector).
C
C=====================================================================
      iprecision=1
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
C=====================================================================
C
C       Check at this point to see if the "prompts" have been passed
C       to this program as command line parameters.  If so, skip the
C       prompts and proceed to the calculations.
C
C       Order and specification of the command line prompts are:
C       arg1 = fvfile (a12)
C       arg2 = rvfile (a12)
C       arg3 = cvfile (a12)
C       arg4 = nv (number of feature vectors)
C       arg5 = nfvdim (dimension of feature vector)
C       arg6 = nrvdim (dimension of result vector)
C
C=====================================================================
      numargs=nargs()
      if(numargs.gt.1) ibatch=1
      if(ibatch.eq.0) go to 120
      call getarg(1,fvfile,istatus)
      call getarg(2,rvfile,istatus)
      call getarg(3,cvfile,istatus)
      bufint='   '
      call getarg(4,bufint,istatus)
      nsf=3
      do 102 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 102
```

```
      kchar(i)=0
      nsf=nsf-1
  102 continue
      nv=0
      do 104 i=1,nsf
      nv=10*nv+kchar(4-i)
  104 continue
      bufint='    '
      call getarg(5,bufint,istatus)
      nsf=3
      do 106 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 106
      kchar(i)=0
      nsf=nsf-1
  106 continue
      nfvdim=0
      do 108 i=1,nsf
      nfvdim=10*nfvdim+kchar(4-i)
  108 continue
      bufint='    '
      call getarg(6,bufint,istatus)
      nsf=3
      do 110 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 110
      kchar(i)=0
      nsf=nsf-1
  110 continue
      nrvdim=0
      do 112 i=1,nsf
      nrvdim=nrvdim*10+kchar(4-i)
  112 continue
      go to 200
c=======================================================================
c
c       We come here only if ibatch=0, i.e., no command line
c       parameters have been entered and the user must be prompted
c       for each of the filenames and variables.
c
c=======================================================================
  120 write(0,4)
      read(0,14) fvfile
      write(0,5)
      read(0,14) rvfile
      write(0,6)
      read(0,14) cvfile
      write(0,7)
      read(0,15) nv
      if(nv.le.0) go to 900
      write(0,8)
      read(0,15) nfvdim
      if(nfvdim.le.0) go to 910
      write(0,9)
      read(0,15) nrvdim
      if(nrvdim.le.0) go to 920
c
c
  200 open(2,file=fvfile,status='unknown',form='formatted')
      do 220 i=1,nv
      do 210 j=1,nfvdim
```

```
      read(2,11) fv(i,j),comma
  210 continue
      read(2,12) dummy1,dummy2
  220 continue
      close(2)
c
c
  300 open(2,file=rvfile,status='unknown',form='formatted')
      do 320 i=1,nv
      do 310 j=1,nrvdim
      k=j+nfvdim
      read(2,11) fv(i,k),comma
  310 continue
      read(2,12) dummy1,dummy2
  320 continue
      close(2)
c
c
  400 ndim=nfvdim+nrvdim
      if(cvfile(2:2).eq.'d'.or.cvfile(2:2).eq.'D') go to 500
      open(2,file=cvfile,status='unknown',form='formatted')
      do 420 i=1,nv
      do 410 j=1,ndim
      write(2,16) fv(i,j)
  410 continue
      write(2,17) cr,lf
  420 continue
      endfile 2
      go to 1000
c=====================================================================
c
c       Special case for writing the file CDnnmm.NNA.
c       We form the "differential" by comparing fv(i,j+1) with fv(i,j),
c       setting the result to +1 if fv(i,j+1).ge.fv(i,j) and to -1
c       otherwise.  Note that the dimension of CDnnmm.NNA will be
c       (nfvdim-1+nrvdim).
c
c=====================================================================
  500 do 520 i=1,nv
      do 510 j=nfvdim,2,-1
      delta=fv(i,j)-fv(i,j-1)
      fv(i,j)=1.
      if(delta.lt.0.) fv(i,j)=-1
  510 continue
  520 continue
  600 open(2,file=cvfile,status='unknown',form='formatted')
      do 620 i=1,nv
      do 610 j=2,ndim
      write(2,16) fv(i,j)
  610 continue
      write(2,17) cr,lf
  620 continue
      endfile 2
      close(2)
      go to 1000
c=====================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=====================================================================
```

```
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
  920 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
c
c
 1000 write(0,10)
      end
c=======================================================================
```

```
C=====================================================================
C     Program : RATENNR.FOR (RATE the Neural Network Result vectors)
C     Version : 5.6  July 5, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        RATENNR provides a convenient method of "rating" the result
C        file (*.nnr) obtained from the NeuralWare simulator.
C        It examines the .nnr file and determines whether or not a
C        given vector result is correct (correct interpreted simply
C        as the largest network output agreeing in position with the
C        result vector).  A ratio is calculated as, (a), if correct,
C        the value of the correct result divided by the second largest
C        output or, (b), if incorrect, the value of the output of the
C        "correct" position divided by the largest output.  A summary
C        is given of the number of result vectors in the file along
C        with the overall, weighted classification accuracy.
C
C        In order to account for some measure of "confidence" in the
C        classification results, a weighting factor is calculated
C        using the ratio defined above as:
C
C        factor = 1/[1 + exp(-4*(ratio-1))]
C
C        and it is this factor which is measured as a 'hit'.  Thus a
C        ratio of 1 produces a factor of 0.5, a ratio of 1.5 yields
C        a factor of .88, while a ratio of 0.5 gives a factor = .12.
C        Although this is a purely ad hoc manner of weighting the
C        performance, it gives more meaningful results than those
C        obtained by simply counting 'hits' (i.e., ratio.gt.1) as
C        +1 and 'misses' (i.e., ratio.le.1) as 0.
C
C        The user is presented with options to print either summary
C        information or a complete listing of the ratio results on a
C        vector by vector basis.  An "information" file (*.inf) is
C        also written which contains the detailed listing of the
C        classification results.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 rv(1536,50),info(1536,4),sumup(100,100)
      character*1 cr,lf,tab,ff,dummy1,dummy2,answer
      character*54 rvfile,infofile,sumupfile
      data cr/13/,lf/10/,tab/09/,ff/12/
C=====================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
```

```
    1 format(1h+,21(1h*),
     +' Result Rating of an .nnr Vector File ',21(1h*)\)
    2 format(1h ,25x,'Version 5.60 : July 05, 1995')
    3 format(1h0,
     +'Enter number of nnr files to process (NO default) : '\)
    4 format(1h ,'Total hits : ',i4,' of ',
     +i4,' (',f7.3,'%)  Sigmoid Value :',F9.4)
    5 format(1h0,'Enter filename ',
     +'(c/w path & extension) for reading  .nnr  file    : ')
    6 format(1h0,'Enter filename ',
     +'(c/w path & extension) for writing the info file  : ')
    7 format(1h0,
     +'Enter total no. of vectors in NNR file (NO default) : '\)
    8 format(1h0,
     +'Enter the number of classes for this result (NO default) : '\)
    9 format(1h0,'Subfile ',i3,' of nnr file : ',a54\)
   10 format(1h0,24x,'Program  completed  successfully.')
   11 format(F7.6\)
   12 format(F8.6\)
   13 format(a1)
   14 format(a54)
   15 format(i4)
   16 format(F12.5,',',\)
   17 format(2a1\)
   18 format(i2)
   20 format(1h0,'Ensure that printer is online and set to ',
     +'non-ps mode before proceeding. '\)
   21 format(a1\)
   22 format(1h ,'Sum of Ratios = ',F11.3,' : Average = ',F9.4)
   23 format(1h ,15X,'Sum of Ratios = ',F11.3,' : Average = ',F9.4//)
   24 format(1h ,'Do you wish to print the results (LPT1:)? '\)
   25 format(1h ,'Details or Summary (default = details) ?')
   26 format(1h ,'Answer d,D or s,S : '\)
   27 format(1h ,'NNR files are concatenated (default=no) ? '\)
   28 format(1h ,'No. of vectors per subfile = '\)
   29 format(1h ,'No. of subfiles = '\)
   30 format(1h ,14X,4F12.5)
   32 format(1h0/)
   33 format(1h ,9X,'Total hits : ',i4,' of ',
     +i4,' (',f7.3,'%)   Sigmoid Value :',F9.4)
   34 format(1h+,16X,'Result for subfile ',i3,' of nnr file : ',a54)
   35 format(1h ,a1)
   36 format(1h ,5a1)
   37 format(1h ,2a1)
   38 format(1h ,23X,'SUMMARY FOR NNR FILE : ',a54)
   39 format(1h ,
     +'Do you wish to write a summary info file (default=no) ? '\)
   40 format(1h ,'Enter filename ',
     +'for writing the summary info file  : '\)
   90 format(1h0,
     +'The number of vectors must be a positive integer.')
   91 format(1h0,
     +'The feature vector dimension must be a positive integer.')
   92 format(1h0,
     +'The result vector dimension must be a positive integer.')
   99 format(1h0,'Hit <RET> to restart program. '\)
c=====================================================================
c
c     Begin by prompting user for the number of nnr files to process,
c     filenames, feature vector dimensions, and choices for printing
c     calculated results in detail or in summary form.
```

```
c
c=====================================================================
      iprecision=1
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      write(0,24)
      read(0,13) answer
      iprint=0
      if(answer.eq.'y'.or.answer.eq.'Y') iprint=2
      if(iprint.eq.0) go to 101
      write(0,25)
      write(0,26)
      read(0,13) answer
      if(answer.eq.'s'.or.answer.eq.'S') iprint=1
      write(0,20)
      read(0,13) answer
  101 write(0,5)
      read(0,14) rvfile
      write(0,3)
      read(0,15) number
      write(0,39)
      read(0,13) answer
      isumup=0
      if(answer.eq.'y'.or.answer.eq.'Y') isumup=1
      if(isumup.eq.0) go to 102
      write(0,40)
      read(0,14) sumupfile
c
c
  102 do 103 i=1,54
      infofile(i:i)=rvfile(i:i)
  103 continue
      i=0
  104 i=i+1
      if(rvfile(i:i).eq.'.') go to 106
      go to 104
  106 mark=i
      infofile(mark+1:mark+3)='inf'
      if(number.eq.1) go to 108
      itens=ichar(rvfile(mark-2:mark-2))-48
      iunits=ichar(rvfile(mark-1:mark-1))-48
      istart=10*itens+iunits
      ifinish=istart+number-1
      go to 110
  108 istart=1
      ifinish=1
c
c
  110 nf=1
      write(0,27)
      icat=0
      read(0,13) answer
      if(answer.eq.'Y'.or.answer.eq.'y') icat=1
      if(icat.eq.0) go to 120
      write(0,28)
      read(0,15) nvf
      write(0,29)
      read(0,15) nf
      go to 130
  120 write(0,7)
```

```
      read(0,15) nvf
  130 write(0,8)
      read(0,15) nc
      ncp1=nc+1
      nc2=2*nc
      nv=nvf*nf
      if(iprint.eq.0) go to 150
      open(3,file='PRN')
c=====================================================================
c
c     Begin execution of large, outside DO-loop which processes the
c     individual result (nnr) files written by the NeuralWare Simulator.
c
c=====================================================================
  150 do 600 k=istart,ifinish
      if(number.eq.1) go to 200
      dummy1=char((k/10)+48)
      dummy2=char(k-10*(k/10)+48)
      rvfile(mark-2:mark-2)=dummy1
      rvfile(mark-1:mark-1)=dummy2
      infofile(mark-2:mark-2)=dummy1
      infofile(mark-1:mark-1)=dummy2
c
c
  200 open(2,file=rvfile)
      do 220 i=1,nv
      read(2,*) (rv(i,j),j=1,nc2)
  220 continue
      close(2)
c
c
      do 260 i=1,nv
      do 250 j=1,nc2
      rv(i,j)=abs(rv(i,j))
  250 continue
  260 continue
c=====================================================================
c
c     Initialize the sums to 0 and then begin the large inner DO-loop
c     to process the 'subfiles' contained within each nnr file (this
c     allows the program to treat concantenated test sets).
c
c=====================================================================
      bigsum=0.
      nthits=0
      bigsumsig=0.
      do 550 ifile=1,nf
      nhits=0
      sigfunc=0.
      lines=0
      do 380 i=1,nvf
      ii=nvf*(ifile-1)+i
      fpmax=0.
      frmax=0.
      do 340 j=1,nc
      if(rv(ii,j).lt.fpmax) go to 340
      fpmax=rv(ii,j)
      markp=j
  340 continue
      do 360 j=ncp1,nc2
      if(rv(ii,j).lt.frmax) go to 360
```

```
      frmax=rv(ii,j)
      markr=j
  360 continue
      temp=rv(ii,markr)
      rv(ii,markr)=0.
      frmax=0.
      do 370 j=ncp1,nc2
      if(rv(ii,j).lt.frmax) go to 370
      frmax=rv(ii,j)
      markr2=j
  370 continue
c=====================================================================
c
c     Having determined the two largest outputs from the network,
c     calculate the appropriate ratio according to whether it is a
c     'hit' or a 'miss'.  Then compute the sigmoidal "factor" which
c     will be used as the measure of the 'hit'-'miss'.
c     Note that a ratio of 1 yields a factor of 0.5, i.e., a 50%
c     "confidence" level.
c
c=====================================================================
      rv(ii,markr)=temp
      if((markr-markp).eq.nc) ratio=rv(ii,markr)/rv(ii,markr2)
      if((markr-markp).ne.nc) ratio=rv(ii,markp+nc)/rv(ii,markr)
      if((markr-markp).eq.nc) nhits=nhits+1
      factor=1./(1.+EXP(-4.*(ratio-1.)))
      sigfunc=sigfunc+factor
      info(i+1,1)=float(markp-1)
      info(i+1,2)=float(markr-nc-1)
      info(i+1,3)=ratio
      info(i+1,4)=factor
  380 continue
c
c

      percent=100.*float(nhits)/float(nvf)
      info(1,1)=float(nhits)
      info(1,2)=float(nvf)
      info(1,3)=percent
      info(1,4)=sigfunc
      sum=0.
      sumsig=0.
      do 390 kk=2,(nvf+1)
      sum=sum+info(kk,3)
      sumsig=sumsig+info(kk,4)
  390 continue
      avg=sum/float(nvf)
      bigsum=bigsum+sum
      nthits=nthits+nhits
      avgsig=sumsig/float(nvf)
      bigsumsig=bigsumsig+sumsig
c=====================================================================
c
c
c     Write the various measurements derived from the nnr file into
c     the 'infofile' with one file per nnr file.
c
c=====================================================================
  400 open(2,file=infofile,status='unknown',form='formatted')
      do 420 i=1,(nvf+1)
      do 410 j=1,4
      write(2,16) info(i,j)
  410 continue
```

```
      write(2,17) cr,lf
  420 continue
c
c
      write(0,9) ifile,rvfile
      write(0,4) nhits,nvf,percent,sigfunc
      write(0,22) sum,avg
      if(iprint.eq.0) go to 500
      write(3,34) ifile,rvfile
      write(3,33) nhits,nvf,percent,sumsig
      write(3,23) sum,avg
      lines=lines+4
      if(lines.le.52) go to 440
      write(3,35) ff
      lines=0
  440 if(iprint.eq.1) go to 500
      do 460 i=1,(nvf+1)
      write(3,30) (info(i,j),j=1,4)
      lines=lines+1
      if(lines.le.55) go to 460
      write(3,35) ff
      lines=0
  460 continue
      write(3,35) ff
  500 continue
      jcol=4*(k-1)+1
      sumup(ifile,jcol)=info(1,1)
      sumup(ifile,jcol+1)=info(1,4)
  550 continue
c
c
      bigpercent=100.*float(nthits)/float(nv)
      bigavg=bigsum/float(nv)
      write(2,16) float(nthits)
      write(2,16) float(nv)
      write(2,16) bigpercent
      write(2,17) cr,lf
      close(2)
      write(0,32)
      write(0,4) nthits,nv,bigpercent,sumsig
      write(0,22) bigsum,bigavg
      if(iprint.eq.0) go to 590
      write(3,32)
      write(3,38) rvfile
      write(3,33) nthits,nv,bigpercent,sigfunc
      write(3,23) bigsum,bigavg
      write(3,35) ff
  590 continue
  600 continue
c=======================================================================
c
c     Finish the program by writing the final summary information.
c
c=======================================================================
      if(isumup.eq.0) go to 900
      do 700 k=1,number
      j=4*(k-1)+1
      smin=1000.
      do 650 i=1,nf
      if(sumup(i,j+1).le.smin) smin=sumup(i,j+1)
  650 continue
```

```
      do 660 i=1,nf
      sumup(i,j+2)=float(nvf)*(sumup(i,j+1)-smin)/(float(nvf)-smin)
      sumup(i,j+3)=100.*sumup(i,j+2)/float(nvf)
  660 continue
  700 continue
      open(2,file=sumupfile,status='unknown',form='formatted')
     do 880 i=1,nf
      do 860 k=1,number
      jcol=4*(k-1)
      do 840 j=1,4
      write(2,16) sumup(i,jcol+j)
  840 continue
  860 continue
      write(2,17) cr,lf
  880 continue
      endfile 2
      close(2)
  900 write(0,10)
      end
C=====================================================================
```

Appendix B
Fortran Source Code for Programs to Calculate the Basis Functions

This appendix contains the source code listings for the following programs:

1. ZERNIKE.FOR
   Calculates the SZF or PZF bases in either of 3 modes (single n single m, single n many m, or many n all m) and stores the results as separate real and imaginary parts. Double precision variables are used in the calculation of the polynomial coefficients to permit calculation up to approximately $n = 40$.

2. WALSH.FOR
   Calculates the 2-D Walsh radial function WRF for a user specified range of orders.

3. HARR.FOR
   Calculates the 2-D Haar radial function HRF for a user specified range of orders.

4. ZERNRAD.FOR
   Calculates the SZRP or PZRP bases in either of 3 modes (single n single m, single n many m, or many n all m). Double precision variables are used in the calculation of the polynomial coefficients to permit calculation up to approximately $n = 40$.

5. WALSH1D.FOR
   Calculates the one-dimensional Walsh function for a user-specified range of orders. Can calculate either the 'normal' Walsh function (x dependence) or the 'radial' Walsh function ($x*x$ dependence) by a trivial modification of the code.

6. HAAR1D.FOR
   Calculates the one-dimensional Haar function for a user-specified range of orders. Can calculate either the 'normal' Haar function (x dependence) or the 'radial' Haar function ($x*x$ dependence) by a trivial modification of the code.

```
C=====================================================================
C     Program : ZERNIKE.FOR
C     Version : 6.0  June 19, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        ZERNIKE calculates the real and imaginary parts of either the
C        Standard (itype=2) or Pseudo (itype=1) Zernike polynomial
C        V   (x,y) for x,y in the unit circle and for positive values
C         nm
C        of the integer index n .LE. 40 (with m.LE.n, m.GE.0, and, for
C        itype=2, (n-m) even).
C
C        The data is calculated on a square grid of dimension
C        2N x 2N and then saved as FASCII files with 'automatic'
C        assignment of filenames.
C
C
C     PROGRAMMING NOTES:
C
C     1. This program offers three 'modes' of calculating the Zernike
C        polynomials:
C
C             imode = 1 corresponds to the calculation for one specific
C                       n,m pair.
C             imode = 2 corresponds to the calculation for a single value
C                       of n over a range of allowable values for m.
C             imode = 3 allows for the calculation over a range of the
C                       principal index n.  In this case, the calculations
C                       are carried out for all possible values of the
C                       secondary index m allowed for each value of n.
C
C     2. Double precision variables are used to calculate the
C        coefficients c(k) of the "radial" function part of V(x,y)
C        and subsequently for the calculation of the function itself.
C        This is necessary since the c(k) requires the calculation of
C        ratios of factorials (the function 'fact' listed at the end
C        of this program) and the numerators and denominators get very
C        large for even modest values of n.  The range for double
C        precision means that this calculation is limited to values of
C        n .LE. 40.  To extend this program beyond this value, it would be
C        necessary to calculate and store the c(k) values separately
C        using an infinite precision tool such as Mathematica and to
C        evaluate the polynomial terms with explicit retention of as
C        many significant figures such as occur in the largest c(k)
C        coefficient.
C
C     3. The saved data files have names in the form TZKNNMM.TYP where:
C
C        T  = S (Standard Zernike) or P (Pseudo Zernike)
C        X  = R (real part) or I (imaginary part)
C        NN = value of principal index (0,1,2,3,.....)
```

```
c        MM = value of secondary index (0,1,2,3,...) such that
c             (NN-MM) is even and .GE. 0
c        TYP= bin (raw data stored as 4-byte floating point)
c           = fsc (FASCII file used for 'import data' for graphing
c                  software such as AXUM,...)
c
c     4. The gridsize is NOT a variable for this program.  However,
c        care has been taken to allow the user to change this
c        parameter relatively easily.  Only the dimensioning
c        assignments, initialization values, and some format
c        statements would have to be altered to allow for a different
c        gridsize.
c
c=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      real*8 c(86),top,bot,fact,x,y,a,b,r,rho,theta,del,power
      real*8 zero,half,one,delsq
      real*4 vr(-127:128,-127:128),vi(-127:128,-127:128)
      real*4 theory(200),areas(200),diff(200)
      character*1 answer,response,cr,lf
      character*12 fullname,measures,names(200)
      data cr/13/,lf/10/
c=======================================================================
c
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=======================================================================
    1 format(1h+,19(1h*),
     +' Calculation of the Zernike Polynomials ',20(1h*))
    2 format(1h ,25x,'Version 6.00 : June 19, 1995')
    3 format(1h ,27x,'Grid Size : ',i4,' x ',i4)
    4 format(1h0,6x,'Do you wish to calculate for a range',
     +' of n,m values (default=no)? '\)
    5 format(a1)
    6 format(1h0,18x,'Principal Index Value  (NO default)     = '\)
    7 format(i2)
    8 format(1h ,18x,'Secondary Index Value  (NO default)     = '\)
    9 format(1h0,18x,'Initial Principal Index Value (NO default) = '\)
   10 format(1h ,18x,' Final  Principal Index Value (NO default) = '\)
   11 format(1h0,18x,'Initial Secondary Index Value (NO default) = '\)
   12 format(1h ,18x,' Final  Secondary Index Value (NO default) = '\)
   13 format(1h0,6x,'Do you wish to calculate for all possible',
     +' m values (default=yes)? '\)
   23 format(1h0,6x,
     +'Calculate Pseudo or Standard Zernike',
     +' polynomials (default=P) ? '\)
   24 format(1h+,16(1h*),
     +' Calculation of the Pseudo Zernike Polynomials ',16(1h*))
   25 format(1h+,15(1h*),
     +' Calculation of the Standard Zernike Polynomials ',15(1h*))
   26 format(1h0,6x,
     +'Suppress the angular dependence',
     +' (default=n) ? '\)
   27 format(1h+,3(1h*),' Calculation of the Pseudo Zernike,'
     +' Polynomials - Radial Dependence Only ',4(1h*))
   28 format(1h+,2(1h*),' Calculation of the Standard Zernike,'
```

```
     +' Polynomials - Radial Dependence Only ',2(1h*))
   30 format(65536(F10.5)\)
   31 format(256(F10.5)\)
   32 format(128(F10.5)\)
   33 format(85(F10.5)\)
   34 format(64(F10.5)\)
   35 format(a12,3(F18.6)\)
   36 format(2a1\)
   40 format(1h ,24x,'Now writing datafile ',a12)
   41 format(1h )
   42 format(1h0,24x,'Program  completed  successfully.')
   43 format(1h ,24x,'Total of ',i4,' files were written.')
   50 format(a10\)
   51 format(F10.5\)
   90 format(1h0,
     +'The index value is negative.')
   91 format(1h0,
     +'The final index value is less than the initial value.')
   92 format(1h0,
     +'The secondary index value exceeds the principal index value.')
   93 format(1h0,
     +'The two indices must be the same parity.')
   94 format(1h0,
     +'Secondary index position is outside permissible range.')
   99 format(1h0,25x,'Hit <RET> to restart program.'\)
c=======================================================================
c
c     Begin by prompting user for the index values after first
c     offering the option to calculate over a range of index values or
c     for a specific n,m combination.
c
c     The values and the combinations they represent are:
c
c       imode=1     Calculate for one specific n,m pair
c       imode=2     Calculate for all allowable values of m for one
c                   specific value of n
c       imode=3     Calculate over a range of n values supplied
c                   by user via a prompt (and for all allowable values
c                   of m)
c
c       itype=1     The Pseudo Zernike polynomials
c       itype=2     The Standard Zernike polynomials
c       inangle=0   Suppress angular dependence ("Circular" functions)
c       inangle=1   Include the angular dependence (the complete
c                   standard or pseudo functions with real and
c                   imaginary parts)
c
c=======================================================================
      igridsize=256
      N=128
      itype=1
      fullname(1:12)='PZ_nnmm.bin'
      measures='measures.___'
      del=1.D0/dfloat(N)
      delsq=del*del
      pi=3.14159265
      zero=0.D0
      half=0.5D0
      one=1.D0
  100 call clearscreen ($GCLEARSCREEN)
      mall=0
```

```
      ifilebin=0
      ifilefsc=0
      write(0,1)
      write(0,2)
      write(0,3) igridsize,igridsize
      write(0,23)
      read(0,5) response
      if(response.eq.'s'.or.response.eq.'S') itype=2
      if(itype.eq.2) fullname(1:1)='S'
      write(0,26)
      read(0,5) answer
      inangle=1
      if(answer.eq.'y'.or.answer.eq.'Y') inangle=0
      if(inangle.eq.0) fullname(3:3)='C'
      write(0,4)
      read(0,5) answer
      call clearscreen ($GCLEARSCREEN)
      if(itype.eq.1.and.inangle.eq.1) write(0,24)
      if(itype.eq.2.and.inangle.eq.1) write(0,25)
      if(itype.eq.1.and.inangle.eq.0) write(0,27)
      if(itype.eq.2.and.inangle.eq.0) write(0,28)
      write(0,2)
      write(0,3) igridsize,igridsize
      if(answer.eq.'y'.or.answer.eq.'Y') go to 175
  150 imode=1
      write(0,6)
      read(0,7) nstart
      if(nstart.lt.0) go to 900
      nfinish=nstart
      write(0,8)
      read(0,7) mstart
      if(mstart.lt.0) go to 900
      if(mstart.gt.nstart) go to 920
      mfinish=mstart
      if(itype.eq.1) go to 200
      if((nstart-mstart-2*((nstart-mstart)/2)).ne.0) go to 930
      go to 200
  175 write(0,9)
      read(0,7) nstart
      if(nstart.lt.0) go to 900
      iparity=nstart-2*(nstart/2)
      write(0,10)
      read(0,7) nfinish
      if((nfinish-nstart).lt.0) go to 910
      if((nfinish-nstart).gt.0) go to 180
      imode=2
      write(0,11)
      read(0,7) mstart
      if(mstart.lt.0) go to 900
      if(mstart.gt.nstart) go to 920
      if(itype.eq.1) go to 177
      if((nstart-mstart-2*((nstart-mstart)/2)).ne.0) go to 930
  177 write(0,12)
      read(0,7) mfinish
      if((mfinish-mstart).lt.0) go to 910
      if(mfinish.gt.nstart) go to 920
      if(itype.eq.1) go to 200
      if((nstart-mfinish-2*((nstart-mfinish)/2)).ne.0) go to 930
      go to 200
  180 imode=3
      mall=1
```

```
      mstart=0
      if(itype.eq.2) mstart=iparity
      mfinish=nstart
      go to 200
c=====================================================================
c
c     We now have specific starting values for n and m.  Begin by
c     calculating the coefficients for the radial portion of V  (x,y)
c                                                             nm
c     after initializing the nindex and mindex variables for the
c     n-loop (label 210 - applicable only when imode = 3) and the
c     m-loop (label 220 - applicable when imode = 2 or 3).
c
c=====================================================================
  200 write(0,41)
      nindex=nstart-1
      mindex=mstart-itype
  210 nindex=nindex+1
      iparity=nindex-2*(nindex/2)
      if(nindex.gt.nfinish) go to 1000
  220 mindex=mindex+itype
      if(mindex.gt.mfinish) go to 210
      if(itype.eq.1) iparity=mindex-2*(mindex/2)
      nterms=nindex-mindex+1
      if(itype.eq.2) nterms=1+((nindex-mindex)/2)
      n1=nindex-mindex+1
      n2=nindex+mindex+2
      if(itype.eq.2) n1=1+((nindex-mindex)/2)
      if(itype.eq.2) n2=1+((nindex+mindex)/2)
      do 240 k=1,nterms
      top=fact((3-itype)*nindex+3-itype-k)
      bot=fact(k-1)*fact(n2-k)*fact(n1-k)
      c(k)=dfloat((-1)**(k+1))*top/bot
  240 continue
c=====================================================================
c
c     We are now ready to proceed with the actual calculations of the
c     real and imaginary parts of V over the grid 2N x 2N.
c     We set up two loops with the y variable on the outside and the x
c     variable on the inside.  The formulas relating the counting
c     integers i,j with x,y are:
c
c     x = i*del - 0.5*del  ; i = -N+1 to N
c     y = j*del - 0.5*del  ; j = -N+1 to N
c
c     Because of the symmetry of the Zernike functions, each calculation
c     results in the value for 8 points in the x,y plane (4 points if
c     y = +/- x).  Accordingly, we calculate the function for each value
c     of j from 1 to N and with i running from i=j to i=N and then
c     assign the other 7 (or 3) values from the symmetry relations.
c     Each point is first checked to see whether or not it lies within
c     the unit circle - if not, the values for vr and vi are set
c     to zero.  Since the imaginary part of the Zernike polynomial
c     vanishes when m = 0, no file is written in this case.
c
c=====================================================================
      do 700 j=1,N
      y=del*(dfloat(j)-half)
      do 600 i=j,N
      x=del*(dfloat(i)-half)
      r=dsqrt(x**2+y**2)
```

```
      if(r.gt.one) go to 410
      theta=datan2(y,x)
      rho=c(1)
      if(nterms.le.1) go to 380
      if(itype.eq.2) go to 340
      do 330 k=2,nterms
      rho=rho*r+c(k)
  330 continue
      go to 380
  340 do 350 k=2,nterms
      rho=rho*(r**2)+c(k)
  350 continue
  380 power=r**mindex
      if(mindex.eq.0) power=one
      rho=rho*power
      if(inangle.eq.0) go to 405
      a=rho*dcos(dfloat(mindex)*theta)
      b=rho*dsin(dfloat(mindex)*theta)
      go to 420
  405 a=rho
      b=0.
      sign=1.
      go to 432
  410 a=zero
      b=zero
  420 if(iparity.eq.1) go to 450
  430 k=mindex/2
      sign=float((-1)**k)
  432 vr(i,j)=a
      vr(-j+1,i)=sign*a
      vr(-i+1,-j+1)=a
      vr(j,-i+1)=sign*a
      vi(i,j)=b
      vi(-j+1,i)=sign*b
      vi(-i+1,-j+1)=b
      vi(j,-i+1)=sign*b
      if(i.eq.j) go to 500
      vr(j,i)=sign*a
      vr(-i+1,j)=a
      vr(-j+1,-i+1)=sign*a
      vr(i,-j+1)=a
      vi(j,i)=-1.*sign*b
      vi(-i+1,j)=-1.*b
      vi(-j+1,-i+1)=-1.*sign*b
      vi(i,-j+1)=-1.*b
      go to 500
  450 k=(mindex+1)/2
      sign=float((-1)**k)
      vr(i,j)=a
      vr(-j+1,i)=sign*b
      vr(-i+1,-j+1)=-1.*a
      vr(j,-i+1)=-1.*sign*b
      vi(i,j)=b
      vi(-j+1,i)=-1.*sign*a
      vi(-i+1,-j+1)=-1.*b
      vi(j,-i+1)=sign*a
      if(i.eq.j) go to 500
      vr(j,i)=-1.*sign*b
      vr(-i+1,j)=-1.*a
      vr(-j+1,-i+1)=sign*b
      vr(i,-j+1)=a
```

```
      vi(j,i)=-1.*sign*a
      vi(-i+1,j)=b
      vi(-j+1,-i+1)=sign*a
      vi(i,-j+1)=-1.*b
  500 continue
  600 continue
  700 continue
c=====================================================================
c
c     Now save the results as data files, assigning the filenames as
c     described in the comments at the start of this listing.
c
c=====================================================================
      ntens=nindex/10
      nunit=nindex-10*ntens
      mtens=mindex/10
      munit=mindex-10*mtens
      fullname(4:4)=char(ntens+48)
      fullname(5:5)=char(nunit+48)
      fullname(6:6)=char(mtens+48)
      fullname(7:7)=char(munit+48)
      fullname(3:3)='R'
      if(inangle.eq.0) fullname(3:3)='C'
      fullname(9:11)='bin'
      write(0,40) fullname
  720 open(2,file=fullname,status='unknown',form='binary')
      do 722 j=N,-N+1,-1
      write(2) (vr(i,j),i=-N+1,N)
  722 continue
      ifilebin=ifilebin+1
      endfile 2
      sumr=0.0
      do 740 j=-N+1,N
      do 730 i=-N+1,N
      sumr=sumr+vr(i,j)
  730 continue
  740 continue
      names(ifilebin)=fullname
      areas(ifilebin)=sumr*delsq
      if(inangle.eq.0) go to 742
      theory(ifilebin)=0.0
      if(nindex.eq.0) theory(ifilebin)=pi
      diff(ifilebin)=theory(ifilebin)-areas(ifilebin)
      go to 748
  742 if(nindex.gt.0) go to 744
      theory(ifilebin)=pi
      diff(ifilebin)=pi-areas(ifilebin)
      go to 748
  744 if(itype.eq.1) go to 746
      signt1=float((-1)**((3*nindex+mindex)/2))
      top=2.*pi*signt1*float(mindex)
      bot=float(nindex*(nindex+2))
      theory(ifilebin)=top/bot
      diff(ifilebin)=theory(ifilebin)-areas(ifilebin)
      go to 748
  746 signt2=float((-1)**(nindex+mindex))
      top=2.*pi*signt2*float(mindex*(mindex+1))
      bot=float(nindex*(nindex+1)*(nindex+2))
      theory(ifilebin)=top/bot
      diff(ifilebin)=theory(ifilebin)-areas(ifilebin)
  748 fullname(9:11)='fsc'
```

```
      write(0,40) fullname
      open(2,file=fullname,status='unknown',form='formatted')
      do 760 j=N,-N+1,-1
      do 750 i=-N+1,N
      rsq=delsq*((float(i)-0.5)**2+(float(j)-0.5)**2)
      if(rsq.gt.1.0) then
      write(2,50) 'missingval'
      else
      write(2,51) vr(i,j)
      end if
  750 continue
  760 continue
      ifilefsc=ifilefsc+1
      endfile 2
c
c
      if(mindex.eq.0.or.inangle.eq.0) go to 800
      fullname(3:3)='I'
      fullname(9:11)='bin'
      write(0,40) fullname
      open(2,file=fullname,status='unknown',form='binary')
      do 762 j=N,-N+1,-1
      write(2) (vi(i,j),i=-N+1,N)
  762 continue
      ifilebin=ifilebin+1
      endfile 2
      sumi=0.0
      do 780 j=-N+1,N
      do 770 i=-N+1,N
      sumi=sumi+vi(i,j)
  770 continue
  780 continue
      names(ifilebin)=fullname
      areas(ifilebin)=sumi*delsq
      theory(ifilebin)=0.0
      diff(ifilebin)=-areas(ifilebin)
  788 fullname(9:11)='fsc'
      write(0,40) fullname
      open(2,file=fullname,status='unknown',form='formatted')
      do 792 j=N,-N+1,-1
      do 790 i=-N+1,N
      rsq=delsq*((float(i)-0.5)**2+(float(j)-0.5)**2)
      if(rsq.gt.1.0) then
      write(2,50) 'missingval'
      else
      write(2,51) vi(i,j)
      end if
  790 continue
  792 continue
      ifilefsc=ifilefsc+1
      endfile 2
c=====================================================================
c
c     For imode = 2 or 3, have to reassign the values of mstart and
c     mfinish and then return to either the m-loop (label 220) or the
c     n-loop (label 210).  If imode = 1, quit.
c
c=====================================================================
  800 if(imode.eq.1) go to 1000
      if(imode.eq.2) go to 220
      if(mindex.lt.mfinish) go to 220
```

```
      ip=(nindex+1)-2*((nindex+1)/2)
      mstart=0
      if(itype.eq.2) mstart=ip
      mfinish=nindex+1
      mindex=mstart-itype
      go to 210
c=======================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=======================================================================
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
  920 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
  930 write(0,93)
      write(0,99)
      read(0,5) answer
      go to 100
  940 write(0,94)
      write(0,99)
      read(0,5) answer
      go to 100
 1000 if(itype.eq.2.and.inangle.eq.1) measures(10:12)='SZF'
      if(itype.eq.1.and.inangle.eq.1) measures(10:12)='PZF'
      if(itype.eq.2.and.inangle.eq.0) measures(10:12)='SZC'
      if(itype.eq.1.and.inangle.eq.0) measures(10:12)='PZC'
      write(0,41)
      write(0,40) measures
      open(2,file=measures,status='unknown',form='formatted')
      do 1200 i=1,ifilebin
      write(2,35) names(i),theory(i),areas(i),diff(i)
      write(2,36) cr,lf
 1200 continue
      endfile 2
      write(0,42)
      write(0,43) (ifilebin+ifilefsc+1)
      end
c=======================================================================
c
c     This function calculates the factorial of integer 'ii' using
c     double precision floating point values.  As such, the limit for
c     for ii is ii.LE.170.
c     The return of a value  fact(ii).lt.0  should be interpreted by
c     the calling program as an error.
c
c=======================================================================
      real*8 function fact(ii)
      integer*4 ij
      if(ii.eq.0.or.ii.eq.1) fact=1.
      if(ii.lt.0) fact=-1.
      if(ii.le.1) go to 100
```

```
      if(ii.gt.12) go to 80
      ij=1
      do 50 jj=2,ii
      ij=ij*int4(jj)
   50 continue
      fact=dfloat(ij)
      go to 100
   80 fact=479001600.D0
      do 90 jj=13,ii
      fact=fact*dfloat(jj)
   90 continue
  100 continue
      end
C=======================================================================
```

```
c=====================================================================
c     Program : WALSH.FOR (calculate the radial WALSH functions
c               with r**2 dependence on the unit circle)
c     Version : 2.0  June 12, 1995
c     Author  : Kenneth L. Sala
c               Communications Research Center
c               Ottawa, Ontario, Canada
c               (613) 998-2823
c               e-mail  sala@digame.dgcd.doc.ca
c
c
c     Summary:
c
c        WALSH calculates the radial Walsh function WAL(nwal,r**2) over
c        the rectangular grid 2Nx2N with WAL(nwal,r**2)=0 for r.gt.1.
c        The functions are calculated via the Rademacher functions
c        and the graycode representation of the integer nwal.
c
c        Three files are written for each value of nwal:
c        - a 'bin' file contains the function as an integer*1 binary
c          file
c        - an 'fsc' file contains the function stored as a FASCII file
c          for direct import into AXUM
c        - a 'WP' file is an ASCII represention of the function which
c          can easily be imported into WordPerfect and printed
c
c        In addition, a file "measures" is written which contains
c        the values of the integral of WAL(nwal,r**2) over the
c        unit circle.
c
c  NOTE: The gridsize is NOT a variable for this program.  However,
c        some care has been taken to allow the user to change this
c        parameter relatively easily.  Only the dimensioning
c        assignments, initialization values, and some format
c        statements would have to be altered to allow for a different
c        gridsize.
c
c=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      integer*1 itemp,b(8),g(8),walsh(-127:128,-127:128),rad,order(256)
      real*4 measure(256)
      character*54 walshfile,axumfile,wpfile,measures
      character*1 answer,cr,lf,space,black,plus,minus,zero
      data cr/13/,lf/10/,space/32/,black/88/,plus/43/,minus/45/,zero/48/
c=====================================================================
c
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=====================================================================
    1 format(1h+,'Calculation of Walsh Funtion - ',
     +'Sequency Ordering')
    2 format(1h0,'Enter starting integer value : '\)
    7 format(1h0,'Enter  ending  integer value : '\)
    3 format(1h0,'Enter filename ',
     +'(c/w path & extension) for writing Walsh file : ')
```

```
    4 format(1h ,'Enter filename ',
     +'(c/w path & extension) for writing Axum  file : ')
    5 format(1h ,'Enter filename ',
     +'(c/w path & extension) for writing  WP   file : ')
    6 format(1h ,'Area calculation: nwal = ',I3,'   Area = ',F9.6)
   20 format(1h0,24x,'Program  completed  successfully.')
   10 format(1h0,'nwal = ',i3,' ; graycode = ',
     +i3,' ; maxbit = ',i3)
   11 format(1h ,'nwal = ',8i1)
    9 format(1h ,'gray = ',8i1)
   12 format(1h0,'Repeat ? '\)
   13 format(a1)
   14 format(a54)
   15 format(i3)
   16 format(F12.5,','\)
   17 format(2a1\)
   18 format(i4)
   19 format(256(i2)\)
   21 format(256a1\)
   22 format(i4,F10.6\)
   23 format(i2\)
C=======================================================================
C
C       Get the starting and ending values for nwal.
C
C=======================================================================
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      read(0,18) nwalbeg
      write(0,7)
      read(0,18) nwalend
      walshfile='wal___.bin'
      axumfile='wal___.fsc'
      wpfile='wal___.wp'
      measures='measures'
      N=128
      del=1.0/float(N)
      delsq=del*del
C=======================================================================
C
C       Begin the calculations
C
C=======================================================================
      do 2000 nwal=nwalbeg,nwalend
      nhuns=nwal/100
      ntens=(nwal-100*nhuns)/10
      nunits=nwal-100*nhuns-10*ntens
      walshfile(4:4)=char(nhuns+48)
      walshfile(5:5)=char(ntens+48)
      walshfile(6:6)=char(nunits+48)
      axumfile(4:4)=char(nhuns+48)
      axumfile(5:5)=char(ntens+48)
      axumfile(6:6)=char(nunits+48)
      wpfile(4:4)=char(nhuns+48)
      wpfile(5:5)=char(ntens+48)
      wpfile(6:6)=char(nunits+48)
C=======================================================================
C
C
C       Calculate the graycode value for nwal.
        Use the g(i) plus the rad function to calculate Walsh
```

```
c       for r.le.1.
c
c=======================================================================
      b(1)=iand(nwal,2#00000001)
      b(2)=ishl((iand(nwal,2#00000010)),-1)
      b(3)=ishl((iand(nwal,2#00000100)),-2)
      b(4)=ishl((iand(nwal,2#00001000)),-3)
      b(5)=ishl((iand(nwal,2#00010000)),-4)
      b(6)=ishl((iand(nwal,2#00100000)),-5)
      b(7)=ishl((iand(nwal,2#01000000)),-6)
      b(8)=ishl((iand(nwal,2#10000000)),-7)
      g(1)=iabs(b(1)-b(2))
      g(2)=iabs(b(2)-b(3))
      g(3)=iabs(b(3)-b(4))
      g(4)=iabs(b(4)-b(5))
      g(5)=iabs(b(5)-b(6))
      g(6)=iabs(b(6)-b(7))
      g(7)=iabs(b(7)-b(8))
      g(8)=b(8)
      M=0
      do 200 i=8,1,-1
      M=2*M+g(i)
  200 continue
      maxbit=1
      do 220 i=1,8
      if(g(i).eq.1) maxbit=i
  220 continue
      write(0,10) nwal,M,maxbit
      write(0,11) (b(i),i=8,1,-1)
      write(0,9) (g(i),i=8,1,-1)
      do 700 j=1,N
      do 600 i=j,N
      rsq=delsq*float(i*(i-1)+j*(j-1)+0.5)
      ival=1
      do 300 k=1,maxbit
      if(g(k).eq.1) ival=ival*rad(k,rsq)
  300 continue
      if(rsq.gt.1.) ival=0
c=======================================================================
c
c       Exploit the radial symmetry of the function to assign one
c       calculated value to 7 other values (4 if x=y) on the grid.
c       In this way, the calculations need only be done on
c       (2Nx2N)/8 points.
c
c=======================================================================
      walsh(i,j)=ival
      walsh(i,-j+1)=ival
      walsh(-i+1,j)=ival
      walsh(-i+1,-j+1)=ival
      if(i.eq.j) go to 600
      walsh(j,i)=ival
      walsh(j,-i+1)=ival
      walsh(-j+1,i)=ival
      walsh(-j+1,-i+1)=ival
  600 continue
  700 continue
c=======================================================================
c
c       Now calculate the 'measure', i.e., the integral of the function
c       over the unit circle.  This is used to check on the algorithm's
```

```
c       validity.  Note, however, that the measure will still run to
c       zero even when the function is under sampled, i.e., nwal exceeds
c       the limit possible for the given gridsize.  NWAL max is given
c       by ( (N/2) - 1 ) - a "soft" limit.
c
c=======================================================================
      sum=0.
      do 900 j=-N+1,N
      do 800 i=-N+1,N
      sum=sum+float(walsh(i,j))
  800 continue
  900 continue
      sum=delsq*sum
      write(0,6) nwal,sum
      order(nwal)=nwal
      measure(nwal)=sum
 1000 open(2,file=walshfile,status='unknown',form='binary')
      do 1010 j=N,-N+1,-1
      write(2) (walsh(i,j),i=-N+1,N)
 1010 continue
      endfile 2
c=======================================================================
c
c       Write an AXUM data file as a FASCII file.
c
c=======================================================================
 1100 open(2,file=axumfile,status='unknown',form='formatted')
      do 1120 j=N,-N+1,-1
      do 1110 i=-N+1,N
      rsq=delsq*((float(i)-0.5)**2+(float(j)-0.5)**2)
      if(rsq.gt.1.0) then
      write(2,17) 'ms'
      else
      write(2,23) walsh(i,j)
      end if
 1110 continue
 1120 continue
c=======================================================================
c
c       Note: If "missing values" are not required, then replace
c       the 2 loops above with:
c       do 1120 j=N,-N+1,-1
c       write(2,19) (walsh(i,j),i=-N+1,N)
c 1120 continue
c=======================================================================
      endfile 2
c     close(2)
c
c
 1200 open(2,file=wpfile,status='unknown',form='formatted')
      do 1400 j=N,-N+1,-1
      do 1300 i=-N+1,N
      itemp=walsh(i,j)
      walsh(i,j)=space
      if(itemp.eq.1) walsh(i,j)=plus
      if(itemp.eq.-1) walsh(i,j)=minus
 1300 continue
 1400 continue
      do 1500 j=N,-N+1,-1
      write(2,21) (walsh(i,j),i=-N+1,N)
```

```
      write(2,17) cr,lf
 1500 continue
      close(2)
c
c
 2000 continue
      open(2,file=measures,status='unknown',form='formatted')
      do 2200 i=nwalbeg,nwalend
      write(2,22) order(i),measure(i)
      write(2,17) cr,lf
 2200 continue

      write(0,12)
      read(0,13) answer
      if(answer.eq.'y'.or.answer.eq.'Y') go to 100
      end
c=======================================================================
c
c       The Rademacher function.
c       rad(k,x) = sign[sin{(2**k)*pi*x}]
c
c=======================================================================
      integer*1 function rad(k,x)
      temp=x
      x=x-aint(x)
      if(x.lt.0.) x=1.+x
      ir=NINT(AINT((x*float(2**k))))
      ir=ir-2*(ir/2)
      rad=1
      if(ir.ne.0) rad=-1
      x=temp
      return
      end
c=======================================================================
```

```
C=====================================================================
C      Program : HAAR.FOR (calculate the radial HAAR functions
C                with r**2 dependence on the unit circle)
C      Version : 2.0  June 12, 1995
C      Author  : Kenneth L. Sala
C                Communications Research Center
C                Ottawa, Ontario, Canada
C                (613) 998-2823
C                e-mail  sala@digame.dgcd.doc.ca
C
C
C      Summary:
C
C         HAAR calculates the radial Haar function HAR(nhaar,r**2) over
C         the rectangular grid 2Nx2N with HAR(nhaar,r**2)=0 for r.gt.1.
C         The functions are calculated directly by the definition
C         of the Haar functions as local functions on specific
C         subintervals of the range [0,1].
C
C         Three files are written for each value of nhaar:
C         - a 'bin' file contains the function as an integer*1 binary
C           file
C         - an 'fsc' file contains the function stored as a FASCII file
C           for direct import into AXUM
C         - a 'WP' file is an ASCII represention of the function which
C           can easily be imported into WordPerfect and printed
C
C         In addition, a file "measures" is written which contains
C         the values of the integral of HAR(nhaar,r**2) over the
C         unit circle.
C
C   NOTE: The gridsize is NOT a variable for this program.  However,
C         some care has been taken to allow the user to change this
C         parameter relatively easily.  Only the dimensioning
C         assignments, initialization values, and some format
C         statements would have to be altered to allow for a different
C         gridsize.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      integer*1 itemp,haar(-127:128,-127:128),order(256)
      real*4 measure(256)
      character*54 haarfile,axumfile,wpfile,measures
      character*1 answer,cr,lf,space,black,plus,minus,zero
      data cr/13/,lf/10/,space/32/,black/88/,plus/43/,minus/45/,zero/48/
C=====================================================================
C
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,'Calculation of Haar Function - ',
     +'Sequency Ordering')
    2 format(1h0,'Enter starting integer value : '\)
    7 format(1h0,'Enter  ending  integer value : '\)
    3 format(1h0,'Enter filename ',
```

```
     +'(c/w path & extension) for writing Haar  file : ')
    4 format(1h ,'Enter filename ',
     +'(c/w path & extension) for writing Axum  file : ')
    5 format(1h ,'Enter filename ',
     +'(c/w path & extension) for writing  WP   file : ')
    6 format(1h ,'Area calculation: nhaar =',I3,'   Area = ',F9.6)
   20 format(1h0,24x,'Program  completed  successfully.')
   10 format(1h0,'nhaar = ',i3,'  = 2**(',i2,') + ',i3)
   11 format(1h ,'Nonzero region : ',f9.6,' to ',
     +f9.6,' to ',f9.6)
   12 format(1h0,'Repeat ? '\)
   13 format(a1)
   14 format(a54)
   15 format(i3)
   16 format(F12.5,','\)
   17 format(2a1\)
   18 format(i4)
   19 format(256(i2)\)
   21 format(256a1\)
   22 format(i4,F10.6\)
   23 format(i2\)
c=======================================================================
c
c       Begin by prompting user for the start and end values of nhaar.
c
c=======================================================================
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      read(0,18) nhaarbeg
      write(0,7)
      read(0,18) nhaarend
      haarfile='har___.bin'
      axumfile='har___.fsc'
      wpfile='har___.wp'
      measures='measures'
      N=128
      del=1.0/float(N)
      delsq=del*del
c=======================================================================
c
c       Begin the calculations of the Haar functions.
c
c=======================================================================
      do 2000 nhaar=nhaarbeg,nhaarend
      ip=0
      m=nhaar
  200 m=m/2
      if(m.eq.0) go to 250
      ip=ip+1
      go to 200
  250 m=nhaar-2**ip
      if(nhaar.eq.0) m=0
c=======================================================================
c
c       Now have nhaar defined as = 2**ip + m.
c       Calculate the three "boundries" a1, a2, a3 as below.
c       Haar is +1 between a1,a2 and -1 between a2,a3 (zero elsewhere).
c
c=======================================================================
      a1=float(m)/float(2**ip)
```

```
      a2=(float(m)+0.5)/float(2**ip)
      a3=(float(m)+1.0)/float(2**ip)
      if(nhaar.eq.0) a1=0.
      if(nhaar.eq.0) a2=1.
      if(nhaar.eq.0) a3=1.
      write(0,10) nhaar,ip,m
      write(0,11) a1,a2,a3
      nhuns=nhaar/100
      ntens=(nhaar-100*nhuns)/10
      nunits=nhaar-100*nhuns-10*ntens
      haarfile(4:4)=char(nhuns+48)
      haarfile(5:5)=char(ntens+48)
      haarfile(6:6)=char(nunits+48)
      axumfile(4:4)=char(nhuns+48)
      axumfile(5:5)=char(ntens+48)
      axumfile(6:6)=char(nunits+48)
      wpfile(4:4)=char(nhuns+48)
      wpfile(5:5)=char(ntens+48)
      wpfile(6:6)=char(nunits+48)
      do 700 j=1,N
      do 600 i=j,N
      rsq=delsq*float(i*(i-1)+j*(j-1)+0.5)
      ival=0
      if(rsq.gt.1) go to 300
      if(rsq.lt.a1) go to 300
      ival=1
      if(rsq.lt.a2) go to 300
      ival=-1
      if(rsq.lt.a3) go to 300
      ival=0
c=====================================================================
c
c       Exploit the symmetry of these functions to assign 7 values
c       to each one calculated (4 if x=y).  Thus it is necessary to
c       calculate only (2Nx2N)/8 points to complete the 2Nx2N grid.
c
c=====================================================================
  300 haar(i,j)=ival
      haar(i,-j+1)=ival
      haar(-i+1,j)=ival
      haar(-i+1,-j+1)=ival
      if(i.eq.j) go to 600
      haar(j,i)=ival
      haar(j,-i+1)=ival
      haar(-j+1,i)=ival
      haar(-j+1,-i+1)=ival
  600 continue
  700 continue
c=====================================================================
c
c       Calculate the "measure" of the Haar function, i.e., its
c       integral over the unit circle.  Note that this value will
c       still tend to zero even in the undersampled case.  A limit
c       for nhaar max is (N/2)*(2**0.5) or about 80 for N=128 (this
c       is a "soft" limit).
c
c=====================================================================
      sum=0.
      do 900 j=-N+1,N
      do 800 i=-N+1,N
      sum=sum+float(haar(i,j))
```

```
  800 continue
  900 continue
      sum=delsq*sum
      write(0,6) nhaar,sum
      order(nhaar)=nhaar
      measure(nhaar)=sum
c=====================================================================
c
c       Now write the binary, FASCII, and WordPerfect files.
c
c=====================================================================
 1000 open(2,file=haarfile,status='unknown',form='binary')
      do 1010 j=N,-N+1,-1
      write(2) (haar(i,j),i=-N+1,N)
 1010 continue
      endfile 2
c
c
 1100 open(2,file=axumfile,status='unknown',form='formatted')
      do 1120 j=N,-N+1,-1
      do 1110 i=-N+1,N
      rsq=delsq*((float(i)-0.5)**2+(float(j)-0.5)**2)
      if(rsq.gt.1.0) then
      write(2,17) 'ms'
      else
      write(2,23) haar(i,j)
      end if
 1110 continue
 1120 continue
c=====================================================================
c
c       Note: If "missing values" are not required, then replace
c       the 2 loops above with:
c       do 1120 j=N,-N+1,-1
c       write(2,19) (haar(i,j),i=-N+1,N)
c  1120 continue
c
c=====================================================================
      endfile 2
      close(2)
c
c
 1200 open(2,file=wpfile,status='unknown',form='formatted')
      do 1400 j=N,-N+1,-1
      do 1300 i=-N+1,N
      itemp=haar(i,j)
      haar(i,j)=space
      if(itemp.eq.1) haar(i,j)=plus
      if(itemp.eq.-1) haar(i,j)=minus
 1300 continue
 1400 continue
      do 1500 j=N,-N+1,-1
      write(2,21) (haar(i,j),i=-N+1,N)
      write(2,17) cr,lf
 1500 continue
      close(2)
c=====================================================================
c
c       Write the "measures" file and then offer user option to rerun.
c
c=====================================================================
```

```
 2000 continue
      open(2,file=measures,status='unknown',form='formatted')
      do 2200 i=nhaarbeg,nhaarend
      write(2,22) order(i),measure(i)
      write(2,17) cr,lf
 2200 continue

      write(0,12)
      read(0,13) answer
      if(answer.eq.'y'.or.answer.eq.'Y') go to 100
      end
c=====================================================================
c
c       The Rademacher function.
c       rad(k,x) = sign[sin{(2**k)*pi*x}]
c
c=====================================================================
      integer*1 function rad(k,x)
      temp=x
      x=x-aint(x)
      if(x.lt.0.) x=1.+x
      ir=NINT(AINT((x*float(2**k))))
      ir=ir-2*(ir/2)
      rad=1
      if(ir.ne.0) rad=-1
      x=temp
      return
      end
c=====================================================================
```

```
C=====================================================================
C     Program : ZERNRAD.FOR
C     Version : 5.0  November 17, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        ZERNRAD calculates either the Standard (itype=2) or Pseudo
C        (itype=1) Zernike polynomial V  (r) along the real line
C                                      nm
C        segment r in [0,1] for n positive and n.LE.40 (with m.LE.n,
C        m.GE.0, and, for itype=2, (n-m) even).
C
C
C     PROGRAMMING NOTES:
C
C     1. This program offers three 'modes' of calculating the Zernike
C        polynomials:
C
C             imode = 1 corresponds to the calculation for one specific
C                       n,m pair.
C             imode = 2 corresponds to the calculation for a single value
C                       of n over a range of allowable values for m.
C             imode = 3 allows for the calculation over a range of the
C                       principal index n.  In this case, the calculations
C                       are carried out for all possible values of the
C                       secondary index m allowed for each value of n.
C
C     2. Double precision variables are used to calculate the
C        coefficients c(k) of the "radial" function part of V(x,y)
C        and subsequently for the calculation of the function itself.
C        This is necessary since the c(k) requires the calculation of
C        ratios of factorials (the function 'fact' listed at the end
C        of this program) and the numerators and denominators get very
C        large for even modest values of n.  The range for double
C        precision means that this calculation is limited to values of
C        n .LE. 40.  To extend this program beyond this value, it would be
C        necessary to calculate and store the c(k) values separately
C        using an infinite precision tool such as Mathematica and to
C        evaluate the polynomial terms with explicit retention of as
C        many significant figures such as occur in the largest c(k)
C        coefficient.
C
C     3. The saved data files have names in the form TRADnnmm.ASC where:
C
C        T  = S (Standard Zernike) or P (Pseudo Zernike)
C        nn = value of principal index (0,1,2,3,.....)
C        mm = value of secondary index (0,1,2,3,...) such that
C             (NN-MM) is even and .GE. 0
C        (Note that, in the case where more than one function is
C        calculated, imode=2 or 3, then nn and mm will be the values
C        corresponding to the last function calculated).
C
```

```
C=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*8 c(86),top,bot,fact,r,rho,del,power
      real*4 poly(1001,100),theory(100),areas(100),diff(100)
      character*1 answer,response,cr,lf
      character*12 fullname,measures,names(100)
      data cr/13/,lf/10/
C=======================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=======================================================================
    1 format(1h+,19(1h*),
     +' Calculation of the Zernike Polynomials ',20(1h*))
    2 format(1h ,23x,'Version 5.00 : November 17, 1995')
    3 format(1h ,29x,'Segment Size : ',i5)
    4 format(1h0,6x,'Do you wish to calculate for a range',
     +' of n,m values (default=no)? '\)
    5 format(a1)
    6 format(1h0,18x,'Principal Index Value  (NO default)     = '\)
    7 format(i2)
    8 format(1h ,18x,'Secondary Index Value  (NO default)     = '\)
    9 format(1h0,18x,'Initial Principal Index Value (NO default) = '\)
   10 format(1h ,18x,' Final  Principal Index Value (NO default) = '\)
   11 format(1h0,18x,'Initial Secondary Index Value (NO default) = '\)
   12 format(1h ,18x,' Final  Secondary Index Value (NO default) = '\)
   13 format(1h0,6x,'Do you wish to calculate for all possible',
     +' m values (default=yes)? '\)
   23 format(1h0,6x,
     +'Calculate Pseudo or Standard Zernike',
     +' polynomials (default=P) ? '\)
   27 format(1h+,3(1h*),' Calculation of the Pseudo Zernike,'
     +' Polynomials - Radial Dependence Only ',4(1h*))
   28 format(1h+,2(1h*),' Calculation of the Standard Zernike,'
     +' Polynomials - Radial Dependence Only ',2(1h*))
   30 format(65536(F10.5)\)
   31 format(256(F10.5)\)
   32 format(128(F10.5)\)
   33 format(85(F10.5)\)
   34 format(64(F10.5)\)
   35 format(a12,3(F18.6)\)
   36 format(2a1\)
   40 format(1h ,24x,'Now writing datafile ',a12)
   41 format(1h )
   42 format(1h0,24x,'Program  completed  successfully.')
   43 format(1h ,20x,'Total of ',i4,' functions were calculated.')
   50 format(a10\)
   51 format(2001(F10.5)\)
   52 format(F10.5,','\)
   53 format(2a1\)
   90 format(1h0,
     +'The index value is negative.')
   91 format(1h0,
     +'The final index value is less than the initial value.')
   92 format(1h0,
     +'The secondary index value exceeds the principal index value.')
   93 format(1h0,
```

```
     +'The two indices must be the same parity.')
   94 format(1h0,
     +'Secondary index position is outside permissible range.')
   99 format(1h0,25x,'Hit <RET> to restart program.'\)
c=======================================================================
c
c     Begin by prompting user for the index values after first
c     offering the option to calculate over a range of index values or
c     for a specific n,m combination.
c
c     The values and the combinations they represent are:
c
c       imode=1      Calculate for one specific n,m pair
c       imode=2      Calculate for all allowable values of m for one
c                    specific value of n
c       imode=3      Calculate over a range of n values supplied
c                    by user via a prompt (and for all allowable values
c                    of m)
c
c       itype=1      The Pseudo Zernike polynomials
c       itype=2      The Standard Zernike polynomials
c
c=======================================================================
      N=1001
      itype=1
      fullname(1:12)='PRADnnmm.ASC'
      measures='measures.___'
      del=1.D0/dfloat(N-1)
      pi=3.14159265
  100 call clearscreen ($GCLEARSCREEN)
      mall=0
      write(0,1)
      write(0,2)
      write(0,3) N
      write(0,23)
      read(0,5) response
      if(response.eq.'s'.or.response.eq.'S') itype=2
      if(itype.eq.2) fullname(1:1)='S'
      inangle=1
      write(0,4)
      read(0,5) answer
      call clearscreen ($GCLEARSCREEN)
      if(itype.eq.1) write(0,27)
      if(itype.eq.2) write(0,28)
      write(0,2)
      write(0,3) N
      if(answer.eq.'y'.or.answer.eq.'Y') go to 175
  150 imode=1
      write(0,6)
      read(0,7) nstart
      if(nstart.lt.0) go to 900
      nfinish=nstart
      write(0,8)
      read(0,7) mstart
      if(mstart.lt.0) go to 900
      if(mstart.gt.nstart) go to 920
      mfinish=mstart
      if(itype.eq.1) go to 200
      if((nstart-mstart-2*((nstart-mstart)/2)).ne.0) go to 930
      go to 200
  175 write(0,9)
```

```
      read(0,7) nstart
      if(nstart.lt.0) go to 900
      iparity=nstart-2*(nstart/2)
      write(0,10)
      read(0,7) nfinish
      if((nfinish-nstart).lt.0) go to 910
      if((nfinish-nstart).gt.0) go to 180
      imode=2
      write(0,11)
      read(0,7) mstart
      if(mstart.lt.0) go to 900
      if(mstart.gt.nstart) go to 920
      if(itype.eq.1) go to 177
      if((nstart-mstart-2*((nstart-mstart)/2)).ne.0) go to 930
  177 write(0,12)
      read(0,7) mfinish
      if((mfinish-mstart).lt.0) go to 910
      if(mfinish.gt.nstart) go to 920
      if(itype.eq.1) go to 200
      if((nstart-mfinish-2*((nstart-mfinish)/2)).ne.0) go to 930
      go to 200
  180 imode=3
      mall=1
      mstart=0
      if(itype.eq.2) mstart=iparity
      mfinish=nstart
      go to 200
c=======================================================================
c
c     We now have specific starting values for n and m.  Begin by
c     calculating the coefficients for the radial polynomial V  (r)
c                                                             nm
c     after initializing the nindex and mindex variables for the
c     n-loop (label 210 - applicable only when imode = 3) and the
c     m-loop (label 220 - applicable when imode = 2 or 3).
c
c=======================================================================
  200 kk=0
      write(0,41)
      nindex=nstart-1
      mindex=mstart-itype
  210 nindex=nindex+1
      iparity=nindex-2*(nindex/2)
      if(nindex.gt.nfinish) go to 1000
  220 mindex=mindex+itype
      if(mindex.gt.mfinish) go to 210
      kk=kk+1
      if(itype.eq.1) iparity=mindex-2*(mindex/2)
      nterms=nindex-mindex+1
      if(itype.eq.2) nterms=1+((nindex-mindex)/2)
      n1=nindex-mindex+1
      n2=nindex+mindex+2
      if(itype.eq.2) n1=1+((nindex-mindex)/2)
      if(itype.eq.2) n2=1+((nindex+mindex)/2)
      do 240 k=1,nterms
      top=fact((3-itype)*nindex+3-itype-k)
      bot=fact(k-1)*fact(n2-k)*fact(n1-k)
      c(k)=dfloat((-1)**(k+1))*top/bot
  240 continue
c=======================================================================
c
```

```
c     We are now ready to proceed with the actual calculations of
c     V(r) over the line segment.
c
c     r = (i-1)/del ; i = 1 to N
c
c=======================================================================
      do 600 i=1,N
      r=float(i-1)*del
      rho=c(1)
      if(nterms.le.1) go to 380
      if(itype.eq.2) go to 340
      do 330 k=2,nterms
      rho=rho*r+c(k)
  330 continue
      go to 380
  340 do 350 k=2,nterms
      rho=rho*(r**2)+c(k)
  350 continue
  380 power=r**mindex
      if(mindex.eq.0) power=1.
      rho=rho*power
      poly(i,kk)=rho
  600 continue
c
c
      ntens=nindex/10
      nunit=nindex-10*ntens
      mtens=mindex/10
      munit=mindex-10*mtens
      fullname(5:5)=char(ntens+48)
      fullname(6:6)=char(nunit+48)
      fullname(7:7)=char(mtens+48)
      fullname(8:8)=char(munit+48)
  725 sumr=0.0
      do 730 i=1,N
      sumr=sumr+poly(i,kk)*float(i-1)*del
  730 continue
      names(kk)=fullname
      areas(kk)=sumr*del
  742 if(nindex.gt.0) go to 744
      theory(kk)=1.
      diff(kk)=1.-areas(kk)
      go to 748
  744 if(itype.eq.1) go to 746
      signt2=float((-1)**((nindex-mindex)/2))
      top=signt2*float(mindex)
      bot=float(nindex*(nindex+2))
      theory(kk)=top/bot
      diff(kk)=theory(kk)-areas(kk)
      go to 748
  746 signt1=float((-1)**(nindex-mindex))
      top=signt1*float(mindex*(mindex+1))
      bot=float(nindex*(nindex+1)*(nindex+2))
      theory(kk)=top/bot
      diff(kk)=theory(kk)-areas(kk)
  748 continue

c=======================================================================
c
c     For imode = 2 or 3, have to reassign the values of mstart and
c     mfinish and then return to either the m-loop (label 220) or the
```

```
c     n-loop (label 210).  If imode = 1, quit.
c
c=============================================================================
  800 if(imode.eq.1) go to 1000
      if(imode.eq.2) go to 220
      if(mindex.lt.mfinish) go to 220
      ip=(nindex+1)-2*((nindex+1)/2)
      mstart=0
      if(itype.eq.2) mstart=ip
      mfinish=nindex+1
      mindex=mstart-itype
      go to 210
c=============================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=============================================================================
  900 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
  910 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
  920 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
  930 write(0,93)
      write(0,99)
      read(0,5) answer
      go to 100
  940 write(0,94)
      write(0,99)
      read(0,5) answer
      go to 100
c=============================================================================
c
c     Now save the results as data files, assigning the filenames as
c     described in the comments at the start of this listing.
c
c=============================================================================
 1000 write(0,40) fullname
      open(2,file=fullname,status='unknown',form='formatted')
      do 1040 i=1,N
      do 1020 j=1,kk
      write(2,52) poly(i,j)
 1020 continue
      write(2,53) cr,lf
 1040 continue
      endfile 2
      if(itype.eq.2) measures(10:12)='SZC'
      if(itype.eq.1) measures(10:12)='PZC'
      write(0,41)
      write(0,40) measures
      open(2,file=measures,status='unknown',form='formatted')
      do 1200 i=1,kk
      write(2,35) names(i),theory(i),areas(i),diff(i)
      write(2,36) cr,lf
```

```
 1200 continue
      endfile 2
      write(0,42)
      write(0,43) kk
      end
c=======================================================================
c
c     This function calculates the factorial of integer 'ii' using
c     double precision floating point values.  As such, the limit for
c     for ii is ii.LE.170.
c     The return of a value  fact(ii).lt.0  should be interpreted by
c     the calling program as an error.
c
c=======================================================================
      real*8 function fact(ii)
      integer*4 ij
      if(ii.eq.0.or.ii.eq.1) fact=1.
      if(ii.lt.0) fact=-1.
      if(ii.le.1) go to 100
      if(ii.gt.12) go to 80
      ij=1
      do 50 jj=2,ii
      ij=ij*int4(jj)
   50 continue
      fact=dfloat(ij)
      go to 100
   80 fact=479001600.D0
      do 90 jj=13,ii
      fact=fact*dfloat(jj)
   90 continue
  100 continue
      end
c=======================================================================
```

```
c=====================================================================
c     Program : WALSH1D.FOR (calculate the 1-D WALSH functions with
c               either a r**2 dependence along the 'normal' r dependence
c               along the line segment [0,1]
c     Version : 3.0  November 18, 1995
c     Author  : Kenneth L. Sala
c               Communications Research Center
c               Ottawa, Ontario, Canada
c               (613) 998-2823
c               e-mail  sala@digame.dgcd.doc.ca
c
c
c     Summary:
c
c        WALSH1D calculates the radial Walsh function WAL(nwal,r**2) or
c        WAL(nwal,r) along the line interval [0,1] in "high resolution"
c        (2001 points).  These functions are meant to serve as detailed
c        references for the 2-D radial Walsh functions calculated
c        on the unit circle by WALSH.
c        The functions are calculated via the Rademacher functions
c        and the graycode representation of the integer nwal.
c
c       NOTE:
c        This program is easily modified and recompiled to give
c        either one which calculates the "normal" Walsh functions,
c        i.e., WAL(nwal,r) on [0,1] or the 'radial' Walsh functions
c        WAL(nwal,r**2) by simply setting imode=0 for the 'normal'
c        case and imode=1 for the 'radial' case - statement 101.
c
c=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      integer*1 b(8),g(8),walsh(2001,128),rad
      character*54 walshfile
      character*1 cr,lf
      data cr/13/,lf/10/
c=====================================================================
c
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=====================================================================
    1 format(1h+,'Calculation of Walsh Funtion - ',
     +'Sequency Ordering')
    2 format(1h0,'Enter starting integer value : '\)
    3 format(1h ,'Enter  ending  integer value : '\)
   10 format(1h0,'nwal = ',i3,' ; graycode = ',i3,' ; maxbit = ',i3)
   11 format(1h ,'nwal = ',8i1)
   12 format(1h ,'gray = ',8i1)
   16 format(i2,','\)
   17 format(2a1\)
   18 format(i4)
   20 format(1h0,24x,'Program  completed  successfully.')
c
c       Begin by prompting user for values of nwalbeg and nwalend.
c
  100 call clearscreen ($GCLEARSCREEN)
  101 imode=1
```

```
      write(0,1)
      write(0,2)
      read(0,18) nwalbeg
      write(0,3)
      read(0,18) nwalend
      ntotal=nwalend-nwalbeg+1
      if(imode.eq.0) walshfile='wal1n___.asc'
      if(imode.eq.1) walshfile='wal1d___.asc'
      N=2001
      del=1./float(N-1)
      delsq=del*del
c
c       Begin calculations of Wal(nwal,r**2)
c
      do 2000 nwal=nwalbeg,nwalend
      index=nwal-nwalbeg+1
c
c       Convert nwal to graycode g(i) and then use g(i) plus the
c       Rademacher functions rad to calculate walsh(i,j)
c
      b(1)=iand(nwal,2#00000001)
      b(2)=ishl((iand(nwal,2#00000010)),-1)
      b(3)=ishl((iand(nwal,2#00000100)),-2)
      b(4)=ishl((iand(nwal,2#00001000)),-3)
      b(5)=ishl((iand(nwal,2#00010000)),-4)
      b(6)=ishl((iand(nwal,2#00100000)),-5)
      b(7)=ishl((iand(nwal,2#01000000)),-6)
      b(8)=ishl((iand(nwal,2#10000000)),-7)
      g(1)=iabs(b(1)-b(2))
      g(2)=iabs(b(2)-b(3))
      g(3)=iabs(b(3)-b(4))
      g(4)=iabs(b(4)-b(5))
      g(5)=iabs(b(5)-b(6))
      g(6)=iabs(b(6)-b(7))
      g(7)=iabs(b(7)-b(8))
      g(8)=b(8)
      M=0
      do 200 i=8,1,-1
      M=2*M+g(i)
  200 continue
      maxbit=1
      do 220 i=1,8
      if(g(i).eq.1) maxbit=i
  220 continue
      write(0,10) nwal,M,maxbit
      write(0,11) (b(i),i=8,1,-1)
      write(0,12) (g(i),i=8,1,-1)
      do 600 i=1,N
      r=del*float(i-1)
      if(imode.eq.0) var=r
      if(imode.eq.1) var=r*r
      walsh(i,index)=1.
      do 300 j=1,maxbit
 1000 if(g(j).eq.1) walsh(i,index)=walsh(i,index)*rad(j,var)
  300 continue
  600 continue
 2000 continue
c
c       Write the ASCII (comma delimited) formatted files.
c
 3000 nwal=nwal-1
```

```
      nhuns=nwal/100
      ntens=(nwal-100*nhuns)/10
      nunits=nwal-100*nhuns-10*ntens
      walshfile(6:6)=char(nhuns+48)
      walshfile(7:7)=char(ntens+48)
      walshfile(8:8)=char(nunits+48)
      open(2,file=walshfile,status='unknown',form='formatted')
      do 3200 i=1,N
      do 3100 j=1,ntotal
      write(2,16) walsh(i,j)
 3100 continue
      write(2,17) cr,lf
 3200 continue
      endfile 2
      write(0,20)
      end
c
c       The Rademacher function.
c       rad(k,x) = sign[sin{(2**k)*pi*x}]
c
      integer*1 function rad(k,x)
      temp=x
      x=x-aint(x)
      if(x.lt.0.) x=1.+x
      ir=NINT(AINT((x*float(2**k))))
      ir=ir-2*(ir/2)
      rad=1
      if(ir.ne.0) rad=-1
      x=temp
      return
      end
c=======================================================================
```

```
C=======================================================================
C     Program : HAAR1D.FOR (calculate the 1-D HAAR functions with either
C               a r**2 dependence or the 'normal' r dependence along the
C               line segment [0,1]
C     Version : 3.0  November 18, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        HAAR1D calculates the 1-D Haar function HAR(nwal,r**2) or
C        HAR(nwal,r) along the line interval [0,1] in "high
C        resolution" (2001 points).
C        These functions are meant to serve as detailed refernences
C        for the 2-D radial Haar functions calculated on the unit
C        circle by HAAR.
C        The functions are calculated directly by the definition
C        of the Haar functions as local functions on specific
C        subintervals of the range [0,1].
C
C
C       NOTE:
C        This program is easily modified and recompiled to give
C        either one which calculates the "normal" Haar functions,
C        i.e., HAR(nwal,r) on [0,1] or the 'radial' Haar functions
C        HAR(nwal,r**2) by simply setting imode=0 for the 'normal'
C        case and imode=1 for the 'radial' case - statement 101.
C
C=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      integer*1 haar(2001,128)
      character*54 haarfile
      character*1 cr,lf
      data cr/13/,lf/10/
C=======================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=======================================================================
    1 format(1h+,'Calculation of Haar Funtion - ',
     +'Sequency Ordering')
    2 format(1h0,'Enter starting integer value : '\)
    3 format(1h ,'Enter  ending  integer value : '\)
    4 format(1h0,24x,'Program  completed  successfully.')
   10 format(1h0,'nhaar = ',i3,'  = 2**(',i2,') + ',i3)
   11 format(1h ,'Nonzero region : ',f9.6,' to ',
     +f9.6,' to ',f9.6)
   15 format(i4)
   16 format(i2,','\)
   17 format(2a1\)
C=======================================================================
```

```
c
c       Begin by prompting user for start & end values of nhar
c
c=======================================================================
  100 call clearscreen ($GCLEARSCREEN)
  101 imode=1
      write(0,1)
      write(0,2)
      read(0,15) nharbeg
      write(0,3)
      read(0,15) nharend
      ntotal=nharend-nharbeg+1
      if(imode.eq.0) haarfile='har1n___.asc'
      if(imode.eq.1) haarfile='har1d___.asc'
      N=2001
      del=1./float(N-1)
      delsq=del*del
c
c
      do 2000 nhar=nharbeg,nharend
      index=nhar-nharbeg+1
      ip=0
      m=nhar
  200 m=m/2
      if(m.eq.0) go to 250
      ip=ip+1
      go to 200
  250 m=nhar-2**ip
      if(nhar.eq.0) m=0
c=======================================================================
c
c       Now have nhar defined as = 2**ip + m.
c       Calculate the three "boundries" a1, a2, a3 as below.
c       Haar is +1 between a1,a2 and -1 between a2,a3 (zero elsewhere).
c
c=======================================================================
      a1=float(m)/float(2**ip)
      a2=(float(m)+0.5)/float(2**ip)
      a3=(float(m)+1.0)/float(2**ip)
      if(nhar.eq.0) a1=0.
      if(nhar.eq.0) a2=1.
      if(nhar.eq.0) a3=1.
      write(0,10) nhar,ip,m
      write(0,11) a1,a2,a3
      do 600 i=1,N
      r=del*float(i-1)
      if(imode.eq.0) var=r
      if(imode.eq.1) var=r*r
      ival=0
 1000 if(var.gt.1) go to 300
 1001 if(var.lt.a1) go to 300
      ival=1
 1002 if(var.lt.a2) go to 300
      ival=-1
 1003 if(var.lt.a3) go to 300
      ival=0
  300 haar(i,index)=ival
  600 continue
 2000 continue
c=======================================================================
c
```

```
c       Write the ASCII (comma delimited) file.
c
c
c=======================================================================
 3000 nhar=nhar-1
      nhuns=nhar/100
      ntens=(nhar-100*nhuns)/10
      nunits=nhar-100*nhuns-10*ntens
      haarfile(6:6)=char(nhuns+48)
      haarfile(7:7)=char(ntens+48)
      haarfile(8:8)=char(nunits+48)
 3001 open(2,file=haarfile,status='unknown',form='formatted')
      do 3200 i=1,N
      do 3100 j=1,ntotal
      write(2,16) haar(i,j)
 3100 continue
      write(2,17) cr,lf
 3200 continue
      close(2)
c
c
      write(0,4)
      end
c=======================================================================
```

Appendix C
Fortran Source Code for Utility Programs

This appendix contains the source code listings for the following programs:

1. CALNOISE.FOR
   Calibrates the SNR for the Numerical Recipes routine "RAN2" for a given seed variable and number of iterations. This calibration was the basis for the preparation of the test subsets used to measure classifier performance.

2. FVASC.FOR
   Calculates a series of feature vector files in various transposed versions and a fitted version for convenience of graphical display. The files produced by this routine are extremely useful for the display/plotting of individual and groups of feature vectors since they can be imported directly as FASCII files in different orderings by a number of different graphical software packages.

3. TORTHZER.FOR
   Tests the orthogonality and normality for functions from the SZF, SZRP, PZF, or PZRP bases.

4. TORTHWAL.FOR
   Test the orthogonality and normality for the Walsh functions.

5. TORTHHAR.FOR
   Tests the orthogonality and normality for the Haar functions.

```
c=====================================================================
c     Program : CALNOISE.FOR
c     Version : 2.2  July 22, 1993
c     Author  : Kenneth L. Sala
c               Communications Research Center
c               Ottawa, Ontario, Canada
c               (613) 998-2823
c               e-mail  sala@digame.dgcd.doc.ca
c
c
c     Summary: The purpose of this little program is to provide a
c              calibration of the random number subroutine "RAN2(IDUM)"
c              from section 7.1 of "Numerical Recipes (Fotran)".
c              This program asks for :
c
c              NP = Number of iterations for the CALL RAN2 loop
c              Seed Integer = initial value of IDUM (first call of RAN2)
c
c              The program then runs the loop (NP times) and "measures"
c              the final number of changed pixels (keeping in mind that
c              the addition is done modula 2 for binary images) in order
c              to calculate the actual SNR generated by NP and ISEED.
c              The noise runs for the Zernike NN tests used ISEED = -7
c              in all cases.
c              The program, in addition to giving the measure SNR, gives
c              various other statistics such as percentage of changed
c              pixels, number of double, triple, ..... "hits", etc.
c
c              Note that the size of the image grid is NOT a variable
c              but can be changed readily by altering the one line
c              defining "N=  " below and changing the dimensioning of
c              array "ihits".
c
c=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
c
c
      integer*2 ihits(-127:128,-127:128)
      integer*4 nhits(0:20),np,k,nchange
      character*1 answer
c
c
    1 format(1h0,'NP = '\)
    2 format(i7)
    3 format(1h0,'Seed integer (.LT.0) = '\)
    4 format(i3)
    5 format(1h0,'Rerun program (default=yes) ? '\)
    6 format(a1)
   85 format(1h0,2X,'NP =',i7,' No. of changes =',i7,
     +' Percentage =',F7.2,' Measured db =',F9.4)
   86 format(/2X,'NP =',i7,' No. of changes =',i7,
     +' Percentage =',F7.2,' Measured db =',F9.4)
   87 format(1h ,'Ratio nchange/NP = ',F7.4)
   88 format(1h ,7i8)
c
c
      N=128
  100 call clearscreen ($GCLEARSCREEN)
```

```
      do 646 i=-N+1,N
      do 644 j=-N+1,N
      ihits(i,j)=0
  644 continue
  646 continue
      write(0,1)
      read(0,2) np
c      write(0,3)
c      read(0,4) iseed
      idum=-7
c      idum=iseed
      do 850 k=1,np
      ranval=2*ran2(idum)-1.
      i=int(float(N)*ranval+.5)
      if(i.lt.(-N+1)) i=-N+1
      if(i.gt.N) i=N
      ranval=2*ran2(idum)-1.
      j=int(float(N)*ranval+.5)
      if(j.lt.(-N+1)) j=-N+1
      if(j.gt.N) j=N
      ihits(i,j)=ihits(i,j)+1
  850 continue
      do 860 i=0,20
      nhits(i)=0
  860 continue
      icount=0
      nchange=0
      do 890 i=-N+1,N
      do 880 j=-N+1,N
      k=ihits(i,j)
      if(k.gt.20) go to 870
      nhits(k)=nhits(k)+1
  870 icount=k-2*(k/2)
      nchange=nchange+icount
  880 continue
  890 continue
      ratio=float(nchange)/float(np)
      percent=100.*float(nchange)/float(4*N*N)
      db=20.*alog10(-1+float(4*N*N)/float(nchange))
      write(0,85) np,nchange,percent,db
      write(0,87) ratio
      write(0,88) (nhits(i),i=0,6)
      write(0,88) (nhits(i),i=7,13)
      write(0,88) (nhits(i),i=14,20)
      write(0,5)
      read(0,6) answer
      if(answer.eq.'n'.or.answer.eq.'N') go to 1000
      go to 100
 1000 end
c=======================================================================
c
c     Random number generator from section 7.1 of "Numerial Recipes :
c     Fortran".
c
c=======================================================================
      FUNCTION RAN2(IDUM)
      PARAMETER (M=714025,IA=1366,IC=150889,RM=1.4005112E-6)
      DIMENSION IR(97)
      DATA IFF /0/
      IF(IDUM.LT.0.OR.IFF.EQ.0)THEN
        IFF=1
```

```
          IDUM=MOD(IC-IDUM,M)
          DO 11 J=1,97
            IDUM=MOD(IA*IDUM+IC,M)
            IR(J)=IDUM
11        CONTINUE
          IDUM=MOD(IA*IDUM+IC,M)
          IY=IDUM
        ENDIF
        J=1+(97*IY)/M
        IF(J.GT.97.OR.J.LT.1) PAUSE
        IY=IR(J)
        RAN2=IY*RM
        IDUM=MOD(IA*IDUM+IC,M)
        IR(J)=IDUM
        RETURN
        END
C=======================================================================
```

```
C=====================================================================
C      Program : FVASC.FOR
C      Version : 6.0  November 3, 1995
C      Author  : Kenneth L. Sala
C                Communications Research Center
C                Ottawa, Ontario, Canada
C                (613) 998-2823
C                e-mail  sala@digame.dgcd.doc.ca
C
C
C      Summary:
C
C         FVASC (Feature Vector ASCII files) reads a CXkkllmm.NNA
C         (test) file or a CXkk.NNA (train) file and generates a
C         series of other ASCII files as below: (X = S, P, or B)
C
C         CXkkllmm.NNA = original file
C         AXkkllmm.ASC = transposed averaged version of CXkkllmm.NNA
C                        arranged as testsets 01, 02, ....
C         RXkkllmm.ASC = transposed averaged version of AXkkllmm.ASC
C                        arranged as classes '0', '1', ..., '8'
C         FXkkllmm.ASC = Fitted version of RXkkllmm.ASC
C
C         The averaged versions are formed by replacing the fv's for
C         any given image by its average (e.g., a set of 23 sets
C         each consisting of 5 fv's for 9 classes will be reduced to
C         a collection of 23 sets consisting of 9 averaged fv's).
C
C         The "fitted" version using a linear transformation on each
C         of the averaged fv's representing a noisy image to give a
C         LMS best fit to its noiseless counterpart.
C
C         The .ASC files are written for convenience as column vectors
C         for direct import into graphing software such as AXUM.
C         In addition, the files RX... and FX... are written as 9
C         groups of 23 vectors, each group representing ONE image class
C         (contrast to CV... CP... CB... where the vectors are
C         arranged as 23 groups of 45 vectors and AX... which is
C         written as 23 groups of 9 (averaged) vectors).
C
C         If the original CX file represents a training set, then a
C         shortened filename as CXkk.NNA is assumed.
C
C         NOTE: All filenames are a12.  This program must be run from
C         within the directory containing the original CX... file.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 fv(1200,200),cv(1200,200)
      integer*2 kchar(3)
      character*1 answer,cr,lf,comma,dummy1,dummy2
      character*3 bufint
      character*12 cvfile,fvfile
      data cr/13/,lf/10/
C=====================================================================
C
```

```
c     All format statements and only format statements have labels
c     in the range 1 - 99.
c
c=====================================================================
    1 format(1h+,14(1h*),
     +' Generation of Extended Feature Result Vector Files ',14(1h*)\)
    2 format(1h ,24x,'Version 6.00 : November 03, 1995')
    4 format(1h0,
     +'Enter  filename  for  reading  cv  file              : '\)
    5 format(1h0,
     +'Training or test set (1=train, 2=test)  (NO default) : '\)
    6 format(1h0,
     +'Enter number of subfiles in cv file (NO default)     : '\)
    7 format(1h0,
     +'Enter number of vectors in each subfile (NO default) : '\)
    8 format(1h0,
     +'Enter dimension of feature vectors (NO default)      : '\)
    9 format(1h0,
     +'Enter dimension of result vectors  (NO default)      : '\)
   10 format(1h0,24x,'Program  completed  successfully.')
   11 format(F12.5,a1\)
   12 format(2a1)
   13 format(a1)
   14 format(a12)
   15 format(i4)
   16 format(F12.5,',',\)
   17 format(2a1\)
   18 format(i2)
   90 format(1h0,
     +'The number of vectors must be a positive integer.')
   91 format(1h0,
     +'The feature vector dimension must be a positive integer.')
   92 format(1h0,
     +'The result vector dimension must be a positive integer.')
   99 format(1h0,'Hit <RET> to restart program. '\)
c=====================================================================
c
c     Begin by prompting or by reading the original cv filename, train
c     or test set status, fv dimension, rv dimension (no. of classes),
c     no. of subfiles, and feature vectors per subfile.
c
c=====================================================================
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2
c=====================================================================
c
c       Check at this point to see if the "prompts" have been passed
c       to this program as command line parameters.  If so, skip the
c       prompts and proceed to the calculations.
c
c       Order and specification of the command line prompts are:
c       arg1 = cvfile (a12)
c       arg2 = itype (1=train, 2=test)
c       arg3 = nfvdim (dimension of feature vectors)
c       arg4 = nrvdim (dimension of result vector (no. of classes))
c       arg5 = nsub (no. of subfiles in cvfile)
c       arg6 = nfvsub (no. of fv's per subfile)
c
c       Note:
c       Total no. of feature vectors in cvfile = nsub*nfvsub
```

```
c       No. of vectors per class per subfile   = nfvsub/nrvdim
c
c=======================================================================
      ibatch=0
      numargs=nargs()
      if(numargs.gt.1) ibatch=1
      if(ibatch.eq.0) go to 160
      call getarg(1,cvfile,istatus)
      bufint='    '
      call getarg(2,bufint,istatus)
      nsf=3
      do 102 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 102
      kchar(i)=0
      nsf=nsf-1
  102 continue
      itype=0
      do 104 i=1,nsf
      itype=10*itype+kchar(4-i)
  104 continue
      bufint='    '
      call getarg(3,bufint,istatus)
      nsf=3
      do 106 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 106
      kchar(i)=0
      nsf=nsf-1
  106 continue
      nfvdim=0
      do 108 i=1,nsf
      nfvdim=10*nfvdim+kchar(4-i)
  108 continue
      bufint='    '
      call getarg(4,bufint,istatus)
      nsf=3
      do 110 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 110
      kchar(i)=0
      nsf=nsf-1
  110 continue
      nrvdim=0
      do 112 i=1,nsf
      nrvdim=nrvdim*10+kchar(4-i)
  112 continue
      bufint='    '
      call getarg(5,bufint,istatus)
      nsf=3
      do 114 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 114
      kchar(i)=0
      nsf=nsf-1
  114 continue
      nsub=0
      do 116 i=1,nsf
      nsub=nsub*10+kchar(4-i)
  116 continue
      bufint='    '
```

```
      call getarg(6,bufint,istatus)
      nsf=3
      do 118 i=1,3
      kchar(i)=ichar(bufint(4-i:4-i))-48
      if(kchar(i).ge.0.and.kchar(i).le.9) go to 118
      kchar(i)=0
      nsf=nsf-1
  118 continue
      nfvsub=0
      do 120 i=1,nsf
      nfvsub=nfvsub*10+kchar(4-i)
  120 continue
      go to 180
c=======================================================================
c
c       We come here only if ibatch=0, i.e., no command line
c       parameters have been entered and the user must be prompted
c       for each of the filenames and variables.
c
c=======================================================================
  160 write(0,4)
      read(0,14) cvfile
      write(0,5)
      read(0,15) itype
      write(0,8)
      read(0,15) nfvdim
      write(0,9)
      read(0,15) nrvdim
      write(0,6)
      read(0,15) nsub
      write(0,7)
      read(0,15) nfvsub
c
c
  180 if(nfvdim.le.0) go to 1010
      if(nrvdim.le.0) go to 1020
      nfv=nsub*nfvsub
      nfvpi=nfvsub/nrvdim
      ncvdim=nfvdim+nrvdim
      base=float(nfvpi)
c=======================================================================
c
c       Read the CXkkllmm.NNA file where X = S, P, or B.
c       Size check for this file:
c       Size = nfv*[13*(nfvdim+nrvdim)+2]
c
c       The CXkkllmm.NNA file contains nfv rows by (nfvdim+nrvdim)
c       columns.  In the processes below, we will transpose this
c       (after 'discarding' the rv portion of the cx... file) to
c       *.ASC files which contain nfvdim rows by nstep (= nfv/nfvpi)
c       averaged columns.
c
c=======================================================================
  200 open(2,file=cvfile,status='unknown',form='formatted')
      do 220 i=1,nfv
      do 210 j=1,nfvdim
      read(2,11) cv(i,j),comma
  210 continue
      read(2,12) dummy1,dummy2
  220 continue
      close(2)
```

```
c=====================================================================
c
c     Now replace each nfvpi vectors (rows of cx...) with its average
c     value, reducing the total number of vectors from nfv to nstep =
c     nfv/nfvpi.  Then reset the remaining cx... vectors (vectors from
c     [nstep+1] to nfv) to 0.
c
c=====================================================================
      nstep=nfv/nfvpi
      do 360 j=1,nfvdim
      do 340 i=1,nstep
      k=5*(i-1)+1
      cv(i,j)=(cv(k,j)+cv(k+1,j)+cv(k+2,j)+cv(k+3,j)+cv(k+4,j))/base
  340 continue
  360 continue
      do 390 i=(nstep+1),nfv
      do 380 j=1,nfvdim
      cv(i,j)=0.
  380 continue
  390 continue
c=====================================================================
c
c       Write the AXkk11mm.ASC file.
c       Size check for this file:
c       Size = nfvdim*[13*(nstep)+2]
c
c=====================================================================
      fvfile=cvfile
      fvfile(1:1)='a'
      if(itype.eq.1) fvfile(6:8)='asc'
      if(itype.eq.2) fvfile(10:12)='asc'
  300 open(2,file=fvfile,status='unknown',form='formatted')
      do 320 j=1,nfvdim
      do 310 i=1,nstep
      write(2,16) cv(i,j)
  310 continue
      write(2,17) cr,lf
  320 continue
      endfile 2
      close(2)
c=====================================================================
c
c     The AXkk11mm.ASC file above is written as vector no. running
c     along the columns and vector component running along the rows.
c     The vectors are grouped as for CXkk11mm.NNA, i.e., in groups
c     corresponding to the testsets, each group containing the image
c     classes in sequence (45 = 9x5 for CX... and 9 = 9x1 for AX...).
c     Now want to rearrange the vectors into nrvdim (always=9) groups,
c     with each group containing nsub (usually = 23) vectors.  Each
c     group of nsub vectors correspond to ONE image class '0', '1',
c     .... '8'.
c
c=====================================================================
  400 do 420 i=1,nstep
      ndiv=(i-1)/nrvdim
      nclass=i-nrvdim*ndiv
      nset=1+ndiv
      k=nsub*(nclass-1)+nset
      do 410 j=1,nfvdim
      fv(k,j)=cv(i,j)
  410 continue
```

```
  420 continue
c=======================================================================
c
c       Write the RXkkllmm.NNA file.
c       Size check for this file (same as AXkkllmm.ASC):
c       Size = nfvdim*[13*(nstep)+2]
c
c=======================================================================
      fvfile(1:1)='r'
  500 open(2,file=fvfile,status='unknown',form='formatted')
      do 520 j=1,nfvdim
      do 510 i=1,nstep
      write(2,16) fv(i,j)
  510 continue
      write(2,17) cr,lf
  520 continue
      endfile 2
      close(2)
c=======================================================================
c
c       Now calculate and write the "fitted" FXkkllmm.ASC file.
c       Size check for this file:
c       Size = nfvdim*[13*(nstep)+2]
c
c=======================================================================
  600 do 700 i=1,nrvdim
      sumx=0.0
      do 620 j=1,nfvdim
      sumx=sumx+cv(i,j)
  620 continue
      do 680 isub=2,nsub
      k=9*(isub-1)+i
      sumy=0.0
      sumxy=0.0
      sumysq=0.0
      do 640 j=1,nfvdim
      sumy=sumy+cv(k,j)
      sumxy=sumxy+cv(i,j)*cv(k,j)
      sumysq=sumysq+(cv(k,j)*cv(k,j))
  640 continue
      denom=(sumy*sumy)-sumysq*float(nfvdim)
      a=(sumx*sumy-sumxy*float(nfvdim))/denom
      b=(sumy*sumxy-sumx*sumysq)/denom
      do 660 j=1,nfvdim
      cv(k,j)=a*cv(k,j)+b
  660 continue
  680 continue
  700 continue
      do 720 i=1,nstep
      ndiv=(i-1)/nrvdim
      nclass=i-nrvdim*ndiv
      nset=1+ndiv
      k=nsub*(nclass-1)+nset
      do 710 j=1,nfvdim
      fv(k,j)=cv(i,j)
  710 continue
  720 continue
      fvfile(1:1)='f'
  800 open(2,file=fvfile,status='unknown',form='formatted')
      do 820 j=1,nfvdim
      do 810 i=1,nstep
```

```
      write(2,16) fv(i,j)
  810 continue
      write(2,17) cr,lf
  820 continue
      endfile 2
      close(2)
      go to 2000
c=============================================================================
c
c     Error messages.  The program restarts (from line 100) after each
c     message is displayed.
c
c=============================================================================
 1000 write(0,90)
      write(0,99)
      read(0,5) answer
      go to 100
 1010 write(0,91)
      write(0,99)
      read(0,5) answer
      go to 100
 1020 write(0,92)
      write(0,99)
      read(0,5) answer
      go to 100
c
c
 2000 write(0,10)
      end
c=============================================================================
```

```
C=======================================================================
C     Program : TORTHZER.FOR
C     Version : 5.2  March 10, 1993
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        TORTHZER calculates the 'orthogonality product' for two
C        Standard or Pseudo Zernike functions V  (x,y) for x,y
C                                              nm
C        in the unit circle and for positive values of integer indices
C        n and m (with m .LE. n, m .GE. 0, and, for the Standard case,
C        (n-m) even).
C
C
C     PROGRAMMING NOTES:
C
C     1. The data files have names in the form TZXNNMM.BIN where:
C
C        T  = P (Pseudo Zernike) or S (Standard Zernike)
C        X  = R (real part) or I (imaginary part)
C        NN = value of principal index (0,1,2,3,.....)
C        MM = value of secondary index (0,1,2,3,...) such that
C             (NN-MM) is even and .GE. 0
C
C     2. The gridsize is NOT a variable for this program.  However,
C        some care has been taken to allow the user to change this
C        parameter relatively easily.  Only the dimensioning
C        assignments, initialization values, and some format
C        statements would have to be altered to allow for a different
C        gridsize.
C
C=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      real*4 v1(256,256),v2(256,256),sum(4)
      character*1 answer
      character*28 fullname1,fullname2
C=======================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=======================================================================
    1 format(1h+,15(1h*),
     +' Test of Orthogonality of the'
     +' Zernike Polynomials ',15(1h*)\)
    2 format(1h ,25x,'Version 5.20 : March 10, 1993')
    3 format(1h ,19x,'Grid Size : ',i4,' x ',i4,
     +'   Precision : Real*',i1)
    4 format(1h0,6x,
     +'Calculate for Pseudo or Standard Zernike',
```

```
     +' polynomials (default=P) ? '\)
    5 format(a1)
    7 format(i4)
    8 format(1h+,11(1h*),
     +' Test of Orthogonality of the Pseudo'
     +' Zernike Polynomials ',12(1h*)\)
    9 format(1h+,10(1h*),
     +' Test of Orthogonality of the Standard'
     +' Zernike Polynomials ',11(1h*)\)
   10 format(1h0,22x,
     +'Enter n value for first file.  n1 = '\)
   11 format(1h ,22x,
     +'Enter m value for first file.  m1 = '\)
   12 format(1h ,22x,
     +'Enter n value for second file. n2 = '\)
   13 format(1h ,22x,
     +'Enter m value for second file. m2 = '\)
   31 format(256(F8.5)\)
   40 format(1h )
   41 format(1h0)
   42 format(1h0,17x,
     +'Theoretical result = (',F9.6,') + i ( 0.000000)')
   43 format(1h ,17x,
     +'Calculated  result = (',F9.6,') + i (',F9.6,')')
   44 format(1h0,12x,'Sum(i) : ',4(E12.6))
   45 format(1h ,25x,'R1-R2',7x,'I1-I2',7x,'R2-I1',7x,'R1-I2')
   60 format(1h0,24x,'Rerun program (default=yes)? : '\)
c=====================================================================
c
c     Begin by prompting user for the index values after first
c     offering the option to calculate over a range of index values or
c     for a specific n,m combination
c
c=====================================================================
      igridsize=256
      N=128
      ndata=256
      pixelarea=1./(float(N*N))
      ipower=8
      iprecision=1
      pi=3.141592654
      fullname1(1:28)='e:\zernike\PZ_nnmm.___'
      fullname2(1:28)='e:\zernike\PZ_nnmm.___'
      fullname1(20:20)=char(iprecision+48)
      fullname2(20:20)=char(iprecision+48)
      fullname1(21:22)='08'
      fullname2(21:22)='08'
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      write(0,2)
      write(0,3) igridsize,igridsize,(4*iprecision)
      itype=1
      write(0,4)
      read(0,5) answer
      if(answer.eq.'s'.or.answer.eq.'S') itype=2
      if(itype.eq.1) fullname1(12:12)='P'
      if(itype.eq.1) fullname2(12:12)='P'
      if(itype.eq.2) fullname1(12:12)='S'
      if(itype.eq.2) fullname2(12:12)='S'
      iauto=0
      theory=0.
```

```
      amplitude=1.
      do 110 i=1,4
      sum(i)=0.
  110 continue
      call clearscreen ($GCLEARSCREEN)
      if(itype.eq.1) write(0,8)
      if(itype.eq.2) write(0,9)
      write(0,2)
      write(0,3) igridsize,igridsize,(4*iprecision)
      write(0,10)
      read(0,7) n1
      write(0,11)
      read(0,7) m1
      write(0,12)
      read(0,7) n2
      write(0,13)
      read(0,7) m2
      if(n1.eq.n2.and.m1.eq.m2) iauto=1
      if(iauto.eq.1) theory=1.
      if(iauto.eq.1) amplitude=(1.+float(n1))/pi
c
c
c
      ntens=n1/10
      nunit=n1-10*ntens
      mtens=m1/10
      munit=m1-10*mtens
      fullname1(15:15)=char(ntens+48)
      fullname1(16:16)=char(nunit+48)
      fullname1(17:17)=char(mtens+48)
      fullname1(18:18)=char(munit+48)
      ntens=n2/10
      nunit=n2-10*ntens
      mtens=m2/10
      munit=m2-10*mtens
      fullname2(15:15)=char(ntens+48)
      fullname2(16:16)=char(nunit+48)
      fullname2(17:17)=char(mtens+48)
      fullname2(18:18)=char(munit+48)
      fullname1(14:14)='R'
      fullname2(14:14)='R'
  250 open(2,file=fullname1,status='unknown',form='binary')
      do 254 j=1,ndata
      read(2) (v1(i,j),i=1,ndata)
  254 continue
      endfile 2
      rewind 2
      if(iauto.eq.0) go to 260
      do 258 i=1,ndata
      do 256 j=1,ndata
      sum(1)=sum(1)+v1(i,j)*v1(i,j)
  256 continue
  258 continue
      if(m1.eq.0) go to 600
      go to 300
  260 open(2,file=fullname2,status='unknown',form='binary')
      do 264 j=1,ndata
      read(2) (v2(i,j),i=1,ndata)
  264 continue
      endfile 2
      rewind 2
```

```
      do 268 i=1,ndata
      do 266 j=1,ndata
      sum(1)=sum(1)+v1(i,j)*v2(i,j)
  266 continue
  268 continue
c
c
  300 if(m2.eq.0) go to 400
      fullname2(14:14)='I'
  350 open(2,file=fullname2,status='unknown',form='binary')
      do 354 j=1,ndata
      read(2) (v2(i,j),i=1,ndata)
  354 continue
      endfile 2
      rewind 2
      if(iauto.eq.1) go to 400
      do 358 i=1,ndata
      do 356 j=1,ndata
      sum(4)=sum(4)+v1(i,j)*v2(i,j)
  356 continue
  358 continue
c
c
  400 if(m1.eq.0) go to 500
      if(iauto.eq.0) go to 440
      do 430 i=1,ndata
      do 420 j=1,ndata
      sum(2)=sum(2)+v2(i,j)*v2(i,j)
  420 continue
  430 continue
      go to 500
  440 fullname1(14:14)='I'
  450 open(2,file=fullname1,status='unknown',form='binary')
      do 454 j=1,ndata
      read(2) (v1(i,j),i=1,ndata)
  454 continue
      endfile 2
      rewind 2
      if(m2.eq.0) go to 555
      do 458 i=1,ndata
      do 456 j=1,ndata
      sum(2)=sum(2)+v1(i,j)*v2(i,j)
  456 continue
  458 continue
c
c
  500 if(m1.eq.0.or.iauto.eq.1) go to 600
      fullname2(14:14)='R'
  550 open(2,file=fullname2,status='unknown',form='binary')
      do 554 j=1,ndata
      read(2) (v2(i,j),i=1,ndata)
  554 continue
      endfile 2
      rewind 2
  555 do 558 i=1,ndata
      do 556 j=1,ndata
      sum(3)=sum(3)+v1(i,j)*v2(i,j)
  556 continue
  558 continue
c
c
```

```
  600 continue
      sumr=(sum(1)+sum(2))*pixelarea*amplitude
      sumi=(sum(3)-sum(4))*pixelarea*amplitude
      write(0,40)
      write(0,44) sum(1),sum(2),sum(3),sum(4)
      write(0,45)
      write(0,40)
      write(0,42) theory
      write(0,43) sumr,sumi
      write(0,60)
      read(0,5) answer
      if(answer.eq.'n'.or.answer.eq.'N') go to 1000
      go to 100
 1000 continue
      end
c=====================================================================
```

```
C=====================================================================
C     Program : TORTHWAL.FOR
C     Version : 1.0  May 25, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        TORTHWAL calculates the 'orthonormality product' for two
C        radial Walsh functions defined on the unit circle.
C
C     PROGRAMMING NOTES:
C
C     1. The data files have names in the form WALnnn.bin and MUST
C        reside in the same path as this program.
C
C     2. The gridsize is NOT a variable for this program.  However,
C        some care has been taken to allow the user to change this
C        parameter relatively easily.  Only the dimensioning
C        assignments, initialization values, and some format
C        statements would have to be altered to allow for a different
C        gridsize.
C
C=====================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      integer*1 wal1(-127:128,-127:128),wal2(-127:128,-127:128)
      character*1 answer
      character*28 fullname1,fullname2
C=====================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=====================================================================
    1 format(1h+,12(1h*),
     +' Test of Orthonormality of the'
     +' Radial Walsh Functions ',12(1h*))
    5 format(a1)
    7 format(i4)
   10 format(1h0,
     +'Enter n value for first  file.  n1 = '\)
   11 format(1h ,
     +'Enter n value for second file.  n2 = '\)
   40 format(1h )
   42 format(1h ,
     +'Theoretical result = ',F9.6)
   43 format(1h0,
     +'Calculated  result = ',F9.6)
   60 format(1h0,24x,'Rerun program (default=yes)? : '\)
C=====================================================================
C
C     Begin by prompting user for the index values after first
C     offering the option to calculate over a range of index values or
```

```
c     for a specific n,m combination
c
c=========================================================================
      igridsize=256
      N=128
      ndata=256
      del=1./float(N)
      pixelarea=1./(float(N*N))
      fullname1(1:28)='wal___.bin'
      fullname2(1:28)='wal___.bin'
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      iauto=0
      theory=0.
      write(0,10)
      read(0,7) n1
      write(0,11)
      read(0,7) n2
      if(n1.eq.n2) iauto=1
      if(iauto.eq.1) theory=3.1415927
c
c
c
      n1huns=n1/100
      n1tens=(n1-100*n1huns)/10
      n1unit=n1-100*n1huns-10*n1tens
      n2huns=n2/100
      n2tens=(n2-100*n2huns)/10
      n2unit=n2-100*n2huns-10*n2tens
      fullname1(4:4)=char(n1huns+48)
      fullname1(5:5)=char(n1tens+48)
      fullname1(6:6)=char(n1unit+48)
      fullname2(4:4)=char(n2huns+48)
      fullname2(5:5)=char(n2tens+48)
      fullname2(6:6)=char(n2unit+48)
  250 open(2,file=fullname1,status='unknown',form='binary')
      do 254 j=-N+1,N
      read(2) (wal1(i,j),i=-N+1,N)
  254 continue
      endfile 2
      rewind 2
      if(iauto.eq.0) go to 260
      sum=0.
      do 258 j=-N+1,N
      do 256 i=-N+1,N
      sum=sum+float(wal1(i,j)*wal1(i,j))
  256 continue
  258 continue
      go to 600
  260 open(2,file=fullname2,status='unknown',form='binary')
      do 264 j=-N+1,N
      read(2) (wal2(i,j),i=-N+1,N)
  264 continue
      endfile 2
      rewind2
      sum=0.
      do 268 j=-N+1,N
      do 266 i=-N+1,N
      sum=sum+float(wal1(i,j)*wal2(i,j))
  266 continue
  268 continue
```

```
c
c
  600 continue
      sum=sum*pixelarea
      write(0,40)
      write(0,43) sum
      write(0,42) theory
      write(0,60)
      read(0,5) answer
      if(answer.eq.'n'.or.answer.eq.'N') go to 1000
      go to 100
 1000 continue
      end
c=======================================================================
```

```
C=======================================================================
C     Program : TORTHHAR.FOR
C     Version : 1.0  June 2, 1995
C     Author  : Kenneth L. Sala
C               Communications Research Center
C               Ottawa, Ontario, Canada
C               (613) 998-2823
C               e-mail  sala@digame.dgcd.doc.ca
C
C
C     Summary:
C
C        TORTHHAR calculates the 'orthonormality product' for two
C        radial Haar functions defined on the unit circle.
C
C     PROGRAMMING NOTES:
C
C     1. The data files have names in the form HARnnn.BIN and MUST
C        reside in the same path as this program.
C
C     2. The gridsize is NOT a variable for this program.  However,
C        some care has been taken to allow the user to change this
C        parameter relatively easily.  Only the dimensioning
C        assignments, initialization values, and some format
C        statements would have to be altered to allow for a different
C        gridsize.
C
C=======================================================================
      include 'fgraph.fi'
      include 'fgraph.fd'
C
C
      integer*1 haar1(-127:128,-127:128),haar2(-127:128,-127:128)
      character*1 answer
      character*28 fullname1,fullname2
      data root2/1.41421356/
C=======================================================================
C
C     All format statements and only format statements have labels
C     in the range 1 - 99.
C
C=======================================================================
    1 format(1h+,12(1h*),
     +' Test of Orthonormality of the'
     +' Radial Haar Functions ',12(1h*))
    5 format(a1)
    7 format(i4)
   10 format(1h0,
     +'Enter n value for first  file.  n1 = '\)
   11 format(1h ,
     +'Enter n value for second file.  n2 = '\)
   40 format(1h )
   42 format(1h ,
     +'Theoretical result = ',F9.6)
   43 format(1h0,
     +'Calculated  result = ',F9.6)
   60 format(1h0,24x,'Rerun program (default=yes)? : '\)
C=======================================================================
C
C     Begin by prompting user for the index values after first
```

```
c     offering the option to calculate over a range of index values or
c     for a specific n,m combination
c
c=======================================================================
      igridsize=256
      N=128
      ndata=256
      del=1./float(N)
      pixelarea=1./(float(N*N))
      fullname1(1:28)='har___.bin'
      fullname2(1:28)='har___.bin'
  100 call clearscreen ($GCLEARSCREEN)
      write(0,1)
      iauto=0
      theory=0.
      write(0,10)
      read(0,7) n1
      write(0,11)
      read(0,7) n2
      if(n1.eq.n2) iauto=1
      if(iauto.eq.1) theory=3.1415927
      ip1=0
      m=n1
  140 m=m/2
      if(m.eq.0) go to 150
      ip1=ip1+1
      go to 140
  150 m1=n1-2**ip1
      amp1=root2**ip1
      if(n1.le.1) amp1=1.
      ip2=0
      m=n2
  160 m=m/2
      if(m.eq.0) go to 170
      ip2=ip2+1
      go to 160
  170 m2=n2-2**ip2
      amp2=root2**ip2
      if(n2.le.1) amp2=1.
c
c
      n1huns=n1/100
      n1tens=(n1-100*n1huns)/10
      n1unit=n1-100*n1huns-10*n1tens
      n2huns=n2/100
      n2tens=(n2-100*n2huns)/10
      n2unit=n2-100*n2huns-10*n2tens
      fullname1(4:4)=char(n1huns+48)
      fullname1(5:5)=char(n1tens+48)
      fullname1(6:6)=char(n1unit+48)
      fullname2(4:4)=char(n2huns+48)
      fullname2(5:5)=char(n2tens+48)
      fullname2(6:6)=char(n2unit+48)
  250 open(2,file=fullname1,status='unknown',form='binary')
      do 254 j=-N+1,N
      read(2) (haar1(i,j),i=-N+1,N)
  254 continue
      endfile 2
      rewind 2
      if(iauto.eq.0) go to 260
      sum=0.
```

```
      do 258 j=-N+1,N
      do 256 i=-N+1,N
      sum=sum+float(haar1(i,j)*haar1(i,j))
  256 continue
  258 continue
      go to 600
  260 open(2,file=fullname2,status='unknown',form='binary')
      do 264 j=-N+1,N
      read(2) (haar2(i,j),i=-N+1,N)
  264 continue
      endfile 2
      rewind2
      sum=0.
      do 268 j=-N+1,N
      do 266 i=-N+1,N
      sum=sum+float(haar1(i,j)*haar2(i,j))
  266 continue
  268 continue
c
c
  600 continue
      sum=sum*pixelarea*amp1*amp2
      write(0,40)
      write(0,43) sum
      write(0,42) theory
      write(0,60)
      read(0,5) answer
      if(answer.eq.'n'.or.answer.eq.'N') go to 1000
      go to 100
 1000 continue
      end
c=============================================================================
```

DOCUMENT CONTROL DATA

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

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

COMMUNICATIONS RESEARCH CENTRE
3701 CARLING AVENUE, P.O. BOX 11490, STN H
OTTAWA, ONTARIO, CANADA K2H 8S2

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

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

IMAGE CLASSIFICATION BY NEURAL NETWORKS USING MOMENT INVARIANT FEATURE VECTORS (U)

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

KENNETH L. SALA

5. DATE OF PUBLICATION (month and year of publication of document)

FEBRUARY 1997

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

~~256~~ 259

6b. NO. OF REFS (total cited in document)

133

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

TECHNICAL REPORT

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
3701 CARLING AVENUE
OTTAWA, ONTARIO

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

1410 FE 233

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

CRC-RP-97-002

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

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

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

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

UNLIMITED

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

AN IMAGE CLASSIFICATION SYSTEM BASED UPON THE EXTRACTION OF MOMENT INVARIANT FEATURE VECTORS AND AN ARTIFICIAL NEURAL NETWORK CLASSIFIER IS DESCRIBED. THE MOMENT INVARIANT FEATURE VECTORS ARE DERIVED FROM THE TEST IMAGES USING SERIES OF ORTHOGONAL BASIS FUNCTIONS. SIX DIFFERENT BASIS FUNCTIONS ARE STUDIED WHICH INCLUDE FOUR TYPES OF ZERNIKE FUNCTIONS AND TWO TYPES OF WALSH FUNCTIONS. FOUR DIFFERENT SCHEMES FOR THE NORMALIZATION OF THE FEATURE VECTORS ARE ALSO INVESTIGATED. THE IMAGES USED IN THE STUDY POSSESS RANDOM SCALES, LATERAL POSITIONS, AND ANGLES OF ORIENTATION IN THE IMAGE PLANE. IN ADDITION, RANDOM NOISE WITH DIFFERENT SIGNAL-TO-NOISE RATIOS IS SUPERIMPOSED UPON THE IMAGES. THE FEATURE VECTOR EXTRACTION TECHNIQUE EMPLOYS THE CONCEPT OF MOMENT INVARIANTS SO THAT THE FEATURE VECTOR COMPONENTS ARE INDEPENDENT OF THE IMAGE'S SCALE, LATERAL POSITION, AND ORIENTATION. THE NEURAL NETWORK EMPLOYED FOR THE CLASSIFICATION TASK IS A MULTILAYER PERCEPTION NETWORK WHICH IS TRAINED WITH THE BACKPROPAGATION ALGORITHM. THE PERFORMANCE OF THE OVERALL CLASSIFICATION SYSTEM IS DETERMINED BY MEASURING THE CLASSIFICATION ACCURACY AS A FUNCTION OF THE SIGNAL-TO-NOISE RATIO OF THE TEST IMAGERY. THE WORK AND THE RESULTS PRESENTED IN THIS STUDY FORM THE BASIS FOR A NEURAL NETWORK BASED, IMAGE RECOGNITION SYSTEM WHICH WILL BE EMPLOYED IN THE CLASSIFICATION OF MILITARY, SYNTHETIC APERTURE RADAR (SAR) IMAGERY OF LAND TARGETS.

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

NEURAL NETWORKS, MOMENT INVARIANT FEATURE VECTORS, IMAGE CLASSIFICATION, SAR IMAGERY