Chapter 2

Modified Fuzzy Neural Network

1 The results of this chapter have been published in *All India seminar on recent trends in computer communication networks (CCN 2001)*, held at I.I.T. Roorkee, in Nov. 2001.
2.1 INTRODUCTION

The FNN proposed by Kwan and Cai\textsuperscript{(10)} constructs neurons in the output layer in unsupervised environment using similarity measure and uses MAX-FNs and MIN-FNs in the second and third layer, respectively. These FNs use max and min operators to perform union and intersection, respectively. The performance of FNN can improve if Yager class of union and intersection operators is used because these functions allow the variation of strength of union and intersection operations. In this chapter, we have proposed MFNN\textsuperscript{(65)} that uses Yager class of union and intersection operators and works under the supervised environment therefore it accepts labeled patterns.

This chapter is organized as follows. In the next section, we discuss the process of image normalization and feature extraction. In section 2.3, the architecture and learning algorithm of the MFNN is described. The experimental results are tabulated in section 2.4. This is followed by the conclusions in section 2.5 and appendix, containing the discussion on the computation of Zernike features in section 2.6.

2.2 NORMALIZATION AND FEATURE EXTRACTION

The database consists of two thousand characters. Ten Devanagiri numerals from two hundred writers with multipens are scanned and stored in BMP format. Each character image is of 65 x 65 pixels. The ring and Zernike features that are extracted from the image are only rotation invariant. To achieve scale and translation invariancy, the image is first normalized with respect to these variables. The moment normalization discussed by Perantonis and Lisboa\textsuperscript{(16)} is used to normalize characters to get translation and scale invariance. The scale and translation invariancy is carried out using the regular moments of the image. The process is described below.

Given a two-dimensional $Q \times Q$ image $\{f(x,y); x,y = 0,...,Q-1\}$, the $(p+q)$th geometrical moment is defined as
\[ m_{pq} = \sum_{x=0}^{2^{p-1}} \sum_{y=0}^{2^{q-1}} x^p y^q f(x, y) \quad \text{for } p, q = 0, 1, 2, \ldots \] \tag{2.1}

Then an image function, \( f(x, y) \), can be normalized with respect to scale and translation by transforming it into \( g(x, y) \), where

\[ g(x, y) = f\left(\frac{x - \bar{x}}{a}, \frac{y - \bar{y}}{a}\right), \] \tag{2.2}

with \((\bar{x}, \bar{y})\) being centroid of \( f(x, y) \) computed as

\[ \bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}}, \] \tag{2.3}

and \( a = \left(\sigma / m_{00}\right)^{1/2} \), \( \sigma \) is a predetermined value for the number of object pixels in the image.

After normalization the rotation invariant ring-data features defined by Ueda and Nakamura and extended by Chiu and Tseng \(^{(19)}\) are extracted from normalized characters by setting the ring width to two. The extracted ring-data vector is a 16-dimensional feature vector. These feature vectors are scaled within the range \([0, 1]\), along each dimension so that the pattern space is a 16-dimensional unit cube, \( I^n \) i.e. \( n = 16 \).

Ueda and Nakamura have defined the ring – data(\( d \)) as the total number of black pixels whose distances to the centroid of a character are truncated to an integer \( d \), as shown in Fig. 2.1. Chiu and Tseng have extended this definition to allow ring width to be greater than one by

\[ \text{ring – data(\( \phi \))} = \sum_{(x, y) \in S_{\phi}} f(x, y), \quad \phi = 1, 2, \ldots, L, \] \tag{2.4}

with
\[ S_\phi = \left\{ (x, y) \mid \text{int} \left[ \frac{\sqrt{x^2 + y^2} - 1}{R_w} + 1 \right] = \phi \right\}, \]

where the \( \phi \)'s are ring numbers, \( S_\phi \) is the set of pixels in the \( \phi \)th ring, \( R_w \) is the ring width, \( L = \text{int}([(N-1)/2) - 1)/R_w + 1] \), is the largest ring number and int [ ] means taking the integer part from a real number.

Figure 2.1: The ring data with ring width \( "R_w = 1" \).

Figure 2.2: The images of character '3' with different orientations

The rotation invariant property of ring features is illustrated by an experiment. Fig. 2.2 shows 65 X 65 binary images of character '3', from top left, before and after normalization and six rotated versions of it, with rotation angles of
Table 2.1: Ring features and their corresponding statistics for character ‘3’.

The rotation invariant Zernike features up to order seven defined in the paper of Khotanzad and Lu\textsuperscript{(18)} are also extracted from normalized characters and then scaled within the range [0,1] along each dimension. The pattern space with Zernike features is a 16-dimensional unit cube because the selected features start from the second order moments and last two features are not used to make number of ring and Zernike features equal. A brief discussion on computation of Zernike features is given in appendix, in section 2.6.
2.3 THE MFNN

The third layer of FNN, which is constructed during learning, uses MIN-FNs and each MIN-FN in this layer represents one learned pattern. The FNN uses unsupervised learning and the MIN-FNs are constructed using only similarity measure. If the input pattern is similar to already learned pattern then it is accommodated by the corresponding MIN-FN, otherwise a new MIN-FN is constructed for that pattern.

The MFNN also constructs the neurons in the third layer during learning, using similarity measure but works under supervised environment. If the input pattern is similar to already learned pattern of that class, then only it is accommodated by that neuron, otherwise a new neuron of that pattern class is constructed. Therefore, each neuron in this layer learns to recognize the patterns of same class. In addition to this, fuzzy neurons of MFNN use Yager class of union and intersection operators.

This section provides a description on the topology of MFNN and its learning algorithm. For the reference and comparison purposes, the notations used have been kept consistent, as far as possible, with the original paper introducing FNN.\(^{(10)}\)

2.3.1 Topology of the MFNN

The architecture of MFNN consists of four layers as shown in Fig. 2.3. The input layer is displayed in two dimension. The first layer has \(n_1 \times n_2\) nodes, assuming each input pattern has \(n_1 \times n_2\) elements.

The algorithm of \((i,j)\)th node in this layer is

\[
y_{ij}^{[1]} = x_{ij}, \quad \text{for } i = 1 \text{ to } n_1, \text{ and } j = 1 \text{ to } n_2, \quad (2.6)
\]
where $x_{ij}$ is the $(i, j)$th value of an input pattern and $y_{ij}$ is the output of $(i, j)$th neuron. Hence, the first layer nodes do not perform any processing and simply distribute the input to the next layer.

The second layer is also displayed in two-dimension and it consists of $n_1 \times n_2$ YAGER-MAX FNs instead of MAX-FNs of FNN. This layer fuzzifies the input patterns. The weight function $w[m, n]$ is used for fuzzification. The state of $(p, q)$th YAGER-MAX FN in this layer is

$$s^{[2]}_{pq} = YAGER\_MAX (w[p-i,q-j] y^{[1]}_{ij})$$

for $i = 1$ to $n_1$, $j = 1$ to $n_2$, for $p = 1$ to $n_1$ and $q = 1$ to $n_2$, \hspace{1cm} (2.7)

where $w[p-i,q-j]$ is the weight function connecting $(i, j)$th node in the first layer to $(p, q)$th YAGER-MAX FN in the second layer is defined as
\[ w[m, n] = \exp(-\beta_1^2(m^2 + n^2)) \]
for \( m = -(n_1 - 1) \) to \((n_1 - 1)\) and \( n = -(n_2 - 1) \) to \((n_2 - 1)\),

where \( \beta_1 \) controls extent of fuzzification and \( YAGER\_MAX \) represents union operator defined as

\[ YAGER\_MAX(a_1, a_2, \ldots, a_r) = \min \left\{ 1, \left( \sum_{i=1}^{r} (a_i)^{w_1} \right)^{1/w_1} \right\}, \]  

(2.9)

where, \( w_1 > 0 \), is a parameter with which this connective has a more optimistic meaning than the max operator and the strength of union operation can be varied by varying the value of \( w_1 \).

The plot of \( w[m, n] \) for \( \beta_1 = 0.3 \), focusing on first element of input pattern is shown in Fig. 2.4. Each neuron in the second layer has \( M \) different outputs equal to the number of YAGER-MIN FNs in the third layer. The computation performed by the \((p, q)\)th YAGER-MAX FN in this layer is defined as

\[ z_{p,q,m}^{[2]} = g_{p,q,m}[z_{p,q}^{[2]}] \]
for \( p = 1 \) to \( n_1 \), \( q = 1 \) to \( n_2 \), and \( m = 1 \) to \( M \)

(2.10)
where the output function $g_{pqm}$ is determined by the learning algorithm and $y^{[2]}_{pqm}$ is the $m$th output of the $(p, q)$th YAGER-MAX FN, which is to be connected to the $m$th YAGER-MIN FN in the third layer.

The output functions of YAGER-MAX FNs are isosceles triangles as shown in Fig. 2.5, with height equal to 1 and base length equal to $\alpha_1$. Hence,

$$y^{[2]}_{pqm} = g_{pqm}(s^{[2]}_{pq})$$
$$= \left[ \begin{array}{ll}
1 - 2|s^{[2]}_{pq} - \Theta_{pqm}| / \alpha_1, & \text{if } \alpha_1 / 2 \geq |s^{[2]}_{pq} - \Theta_{pqm}| \geq 0 \\
0 , & \text{otherwise}
\end{array} \right]$$

for $\alpha_1 \geq 0$, $p = 1$ to $n_1$, $q = 1$ to $n_2$ and $m = 1$ to $M$

where $\Theta_{pqm}$ is the central point of the base of function $g_{pqm}$ and the learning algorithm decides the corresponding values of $\alpha_1$ and $\Theta_{pqm}$ for every set of $p$, $q$ and $m$.

$g_{pqm}(s^{[2]}_{pq})$

Figure 2.5: Output function of a YAGER-MAX FN in the second layer

The third layer uses YAGER-MIN FNs instead of MIN-FNs of FNN. This layer is constructed during learning. Each YAGER-MIN FN in this layer represents similar learned patterns of the same class. The output of the $m$th YAGER-MIN FN in this layer is
\[ y_m^{[2]} = YAGER\_MIN (y_{pqm}^{[2]}) \]

for \( p = 1 \) to \( n_1 \), \( q = 1 \) to \( n_2 \) and \( m = 1 \) to \( M \)

where \( YAGER\_MIN \) represents intersection operator defined as

\[ YAGER\_MIN(a_1, a_2, \ldots, a_r) = 1 - \min \left\{ 1, \left( \sum_{i=1}^{r} (1 - a_i)^{w_2} \right)^{1/w_2} \right\}, \tag{2.13} \]

where, \( w_2 > 0 \), is a parameter with which this connective has a more pessimistic meaning than the min operator and the strength of intersection operation can be varied by varying the value of \( w_2 \).

The output layer gives a fuzzy decision and consists of \( N \) processing nodes i.e. equal to number of classes. The output of \( k \)th node represents the degree to which the input pattern belongs to the class \( k \). The weights assigned to the connections between YAGER-MIN and output layer are binary values. These weights are stored in the matrix \( U \) and updated during learning as

\[ u_{jk} = \begin{cases} 1 & \text{if } m_j \text{ is a YAGER\_MIN FN of the class } k \\ 0 & \text{otherwise} \end{cases} \tag{2.14} \]

for \( k = 1 \) to \( N \) and \( j = 1 \) to \( M \)

where \( m_j \) is the \( j \)th node of YAGER-MIN FN layer. Each output node performs the union of fuzzy values returned by the YAGER-MIN FNs of the same class. Suppose \( n_k \) is the \( k \)th node of output layer that represents class \( k \) then

\[ n_k = \max_{j=1}^{M} m_j u_{jk} \quad \text{for } k = 1 \text{ to } N . \tag{2.15} \]

### 2.3.2 Supervised Learning Algorithm

The MFNN also uses similarity measure like FNN but if the input pattern is similar to already learned pattern of that class then only it is accommodated by the YAGER-MIN FN, otherwise new YAGER-MIN FN of that pattern class is
constructed. Therefore, each YAGER-MIN FN learns to recognize the patterns of the same class.

The parameters \( \alpha_i, \Theta_{pqm} \) for each set of \( p, q \) and \( m \), \( \beta_i \) and \( M \) are determined by learning. Given the \( k \)th training pair \((R_k, d_k)\), where the pattern \( R_k \) is belonging to the class \( d_k, T_f^1 \) \((0 \leq T_f^1 \leq 1)\), the tolerance factor of MFNN and \( K \) as total number of training patterns, the steps in the learning algorithm are stated below.

**Step 1:** Select \( \alpha_i \geq 0 \) and \( \beta_1 \). Set \( M = 0 \) and \( k = 1 \). Create \( n_1 \times n_2 \) input nodes and \( n_1 \times n_2 \) YAGER-MAX FNs in the second layer.

**Step 2:** Set \( M = M + 1 \). Construct the \( M \)th YAGER-MIN FN in the third layer by setting \( \Theta_{pqm} \) as

\[
\Theta_{pqM} = s_{pqM}^{[2]} = \text{YAGER-MAX} \left( \text{YAGER-MAX} \left( w[p-i,q-j]y_{ijk} \right) \right)_{i=1}^{n_1} \quad (2.16)
\]

where \( p = 1 \) to \( n_1 \), \( q = 1 \) to \( n_2 \), and \( y_{ijk} \) is the output of \((i,j)\)th input node for the \( k \)th pattern and \( \Theta_{pqM} \) is the central point of the \( M \)th output function of \((p,q)\)th YAGER-MAX FN. The class of created YAGER-MAX FN is stored in the vector \( c \) as, \( c(M) = d_k \).

**Step 3:** \( k = k + 1 \). If \( k > K \), go to the step 4, otherwise input \( k \)th pattern and compute the output of current MFNN with \( M \) YAGER-MIN FNs and calculate

\[
\sigma = 1 - \max_{j=1}^{M} (y_{jk}^{[3]}) \quad (2.17)
\]

where \( y_{jk}^{[3]} \) is the output of the \( j \)th YAGER-MIN FN for the \( k \)th pattern. If the \( m \)th YAGER-MIN FN delivers maximum output then,
if \( \sigma \leq T_f^1 \) and \( c(m) = d_k \), go to step 3
else go to step 2. \hspace{1cm} (2.18)

**Step 4:** Stop.

### 2.4 SIMULATION RESULTS

We have implemented the MFNN algorithm using MATLAB 5.1 and ran on Pentium-III, 733MHz PC. Several experiments are carried out to compare the MFNN algorithm with the FNN algorithm. Details of four data sets prepared from the database and used in the experiments are given below.

The set-1 is unrotated training set, i.e. original training set consisting of one thousand training patterns from the database, which is reused to verify the recognition. The set-2 is rotated testing set extracted from set-1, i.e. each sample of set-2 is a rotated version of sample in set-1 with an angle of 15°. The set-3 is unrotated testing set consisting of remaining one thousand patterns in the database that is used to evaluate generality. The set-4 is rotated testing set extracted from set-3, i.e. each sample of set-4 is a rotated version of sample in set-3, with an angle of 15°. The set-2 and set-4 are used to evaluate the rotation invariance power of the proposed approach.

The percentage recognition rates of MFNN algorithm are compared with the FNN algorithm using ring-data features. In each experiment recognition rate is defined as the ratio of the number of tested samples being correctly recognized to the number of all tested samples.

After performing many experiments the best results obtained from FNN with \( \beta = 0.65 \), \( \alpha = 2.5 \) and \( T_f = 0.0175 \) are listed in the last row of Table 2.2. The tolerance factor, \( T_f \), is adjusted to get 100% recognition rate for training set, set-1, which has resulted in 971 neurons in the output layer. Except the last row the results tabulated in Table 2.2 are for the MFNN algorithm.
We ran the MFNN algorithm by fixing $\alpha_i = 2.5$ and $\beta_i = 0.65$ to the same values as $\alpha$ and $\beta$ in the FNN and varying the parameters $w_1$ and $w_2$ to evaluate the performance of Yager class of union and intersection operators. The MFNN algorithm yields best recognition rates for $w_1 = 20$ and $w_2 = 10$, which are listed in the eleventh row of Table 2.2. With these setting the MFNN algorithm creates 963 YAGER-MIN FNs in the third layer. Hence, the comparison of eleventh row with last row of Table 2.2 indicates that the MFNN algorithm performs better than the FNN algorithm even with the less number of YAGER-MIN FNs in the third layer as compared to MIN-FNs of the FNN.

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>Set-1</th>
<th>Set-2</th>
<th>Set-3</th>
<th>Set-4</th>
<th>Average</th>
<th>Neurons</th>
<th>$T_f / T_f^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>50</td>
<td>100</td>
<td>80.5</td>
<td>34.2</td>
<td>30.5</td>
<td>61.3</td>
<td>970</td>
<td>0.018</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>100</td>
<td>80.4</td>
<td>34.3</td>
<td>30.1</td>
<td>61.2</td>
<td>966</td>
<td>0.0195</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>100</td>
<td>80.4</td>
<td>33.5</td>
<td>30.3</td>
<td>61.05</td>
<td>963</td>
<td>0.019</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>100</td>
<td>68.6</td>
<td>30.0</td>
<td>23.9</td>
<td>55.6</td>
<td>977</td>
<td>0.023</td>
</tr>
<tr>
<td>100</td>
<td>25</td>
<td>100</td>
<td>80.6</td>
<td>34.3</td>
<td>30.5</td>
<td>61.3</td>
<td>971</td>
<td>0.017</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>100</td>
<td>80.4</td>
<td>34.2</td>
<td>30.4</td>
<td>61.2</td>
<td>967</td>
<td>0.019</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>100</td>
<td>80.5</td>
<td>33.9</td>
<td>30.0</td>
<td>61.1</td>
<td>964</td>
<td>0.019</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>100</td>
<td>68.6</td>
<td>29.8</td>
<td>24.3</td>
<td>55.6</td>
<td>977</td>
<td>0.023</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>100</td>
<td>80.9</td>
<td>33.9</td>
<td>29.7</td>
<td>61.1</td>
<td>971</td>
<td>0.018</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>100</td>
<td>80.7</td>
<td>34.0</td>
<td>29.6</td>
<td>61.07</td>
<td>966</td>
<td>0.019</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>100</td>
<td>81.1</td>
<td>34.1</td>
<td>29.9</td>
<td>61.2</td>
<td>963</td>
<td><strong>0.0195</strong></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
<td>69</td>
<td>29.3</td>
<td>24.9</td>
<td>55.8</td>
<td>977</td>
<td>0.024</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>100</td>
<td>77.4</td>
<td>31.1</td>
<td>30.7</td>
<td>59.8</td>
<td>978</td>
<td>0.085</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>100</td>
<td>77.9</td>
<td>30.9</td>
<td>30.7</td>
<td>59.8</td>
<td>977</td>
<td>0.085</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>100</td>
<td>77.9</td>
<td>30.4</td>
<td>30.2</td>
<td>59.6</td>
<td>977</td>
<td>0.088</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>48.7</td>
<td>29.4</td>
<td>24.3</td>
<td>50.6</td>
<td>974</td>
<td>0.11</td>
</tr>
<tr>
<td><strong>FNN</strong></td>
<td>100</td>
<td>78.7</td>
<td>33.1</td>
<td>29.6</td>
<td>60.35</td>
<td>971</td>
<td><strong>0.0175</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Percentage recognition rates with ring features

The performance of these algorithms is compared for training time and recall time per pattern using ring-data features. We have trained and tested these
algorithms using set-1, with the same parameters for which best results are obtained. Timing analysis tabulated in Table 2.3 indicates that the MFNN requires more time in training and recall phase.

The performance of MFNN and FNN algorithms is also compared with Zernike features. The FNN yields best results for $\beta = 0.65$, $\alpha = 2.5$ and creates 989 MIN-FNs. By keeping the values of $\alpha$ and $\beta$ same as in the FNN algorithm, we have observed the performance of MFNN algorithm by varying $w_1$ and $w_2$. The MFNN yields best results for $w_1 = 5$, $w_2 = 5$ and creates 976 YAGER-MIN FNs. These results are listed in Table 2.4. Table 2.4 reveals that the MFNN algorithm has slight edge over the FNN algorithm with less number of YAGER-MIN FNs.

The comparison of Table 2.2 and 2.4 shows that the recognition rates obtained using FNN and MFNN algorithms are better for ring features than Zernike features.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Training time in Seconds</th>
<th>Recall time per pattern in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNN</td>
<td>584.74</td>
<td>1.1685</td>
</tr>
<tr>
<td>MFNN</td>
<td>702.61</td>
<td>1.3848</td>
</tr>
</tbody>
</table>

Table 2.3: Timing analysis using ring features

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Set-1</th>
<th>Set-2</th>
<th>Set-3</th>
<th>Set-4</th>
<th>Average</th>
<th>Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFNN</td>
<td>100</td>
<td>27.9</td>
<td>20.1</td>
<td>15.4</td>
<td>40.85</td>
<td>976</td>
</tr>
<tr>
<td>FNN</td>
<td>100</td>
<td>25.6</td>
<td>20.8</td>
<td>13.9</td>
<td>40.075</td>
<td>989</td>
</tr>
</tbody>
</table>

Table 2.4: Percentage recognition rates with Zernike features

The noise tolerance of FNN and MFNN algorithms is studied experimentally using ring features. Different sets with different noise densities are gener-
ated by adding 'salt and pepper' noise using 'imnoise' function supported by MATLAB. The sets, set-5, set-6, set-7, set-8, set-9, set-10 and set-11 are generated by adding noise in the training set, set-1, with the noise density of 0.002, 0.005, 0.01, 0.03, 0.05, 0.08 and 0.1 respectively. Fig. 2.6 shows images of sample character Devanagiri ‘five’, from top left, with noise density of 0.002, 0.005, 0.01, 0.03, 0.05, 0.08 and 0.1, respectively.

![Sample character images with different noise densities](image)

**Figure 2.6: The images of sample character with different noise densities.**

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Set-5</th>
<th>Set-6</th>
<th>Set-7</th>
<th>Set-8</th>
<th>Set-9</th>
<th>Set-10</th>
<th>Set-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNN</td>
<td>79.1</td>
<td>65.0</td>
<td>58.7</td>
<td>45.7</td>
<td>33.4</td>
<td>25.1</td>
<td>22.0</td>
</tr>
<tr>
<td>MFNN</td>
<td>80.7</td>
<td>67.7</td>
<td>63.0</td>
<td>48.6</td>
<td>35.8</td>
<td>26.9</td>
<td>23.1</td>
</tr>
</tbody>
</table>

**Table 2.5: Percentage recognition rates with different noise densities**

The FNN and MFNN are trained using set-1 and the performance is tested with sets, set-5 to 11. These results are listed in Table 2.5. This table shows that the performance of these classifiers degrades relatively as the noise density increases.
2.5 CONCLUSIONS

Unlike FNN, the MFNN algorithm can accept labeled patterns and uses new YAGER-MIN and YAGER-MAX FNs. The performance of MFNN algorithm is compared with the FNN algorithm using ring and Zemike features. The MFNN algorithm yields better percentage recognition rates as compared to the FNN algorithm at the cost of insignificant increase in training time and recall time per pattern.

The FNN and MFNN algorithm gives better recognition rates for ring features as compared to Zernike features; hence discrimination power of ring features is superior as compared to Zernike features.

The performance of both the classifiers degrades relatively as the noise density increases.

2.6 APPENDIX

Zernike features: The basis set \( x^p y^q \), used to compute regular geometrical moments is not orthogonal. Therefore, the features defined on these moments lack useful properties that might result from using orthogonal basis functions.

Zernike introduced a set of complex polynomials, which form a complete orthogonal set over the interior of the unit circle, i.e. \( x^2 + y^2 = 1 \). Let the set of these polynomials be denoted by \( \{V_{nm}(x,y)\} \). The form of these polynomials is

\[
V_{nm}(x,y) = V_{nm}(\rho, \psi) = R_{nm}(\rho) \exp(jm\psi)
\]  
(A2.1)

where \( n \) is a positive integer or zero, \( m \) can be positive or negative integer subject to constraints \( n - |m| \) even, \( |m| \leq n \), \( \rho \) is a length of vector from origin to \((x,y)\) pixel, \( \psi \) is a angle between vector \( \rho \) and \( x \)-axis in counter-clockwise direction and \( R_{nm}(\rho) \) is a radial polynomial defined as
Note that $R_{nm}(\rho) = R_{nm}(\rho)$ and these polynomials are orthogonal.

Zernike moments are the projections of the image function onto these orthogonal basis functions. The Zernike moments of order $n$ with repetition $m$ for a digital image, $f(x, y)$, that vanishes outside the unit circle is

$$A_{nm} = \frac{n + 1}{\pi} \sum_x \sum_y f(x, y) V_{nm}^*(\rho, \psi), \text{ where } x^2 + y^2 \leq 1.$$ \hspace{1cm} (A2.3)

To compute Zernike moments of a given image, the center of the image is taken as the origin and pixel coordinates are mapped to the range of the unit circle and pixels falling outside the unit circle are not used in the computation. Also note that $A_{*nm} = A_{n-m}$.

The features defined on Zernike moments are derived by using rotational properties of these moments. Consider a rotation of the image through an angle $\theta$. The relationship between $A'_{nm}$ and $A_{nm}$, Zernike moments of the rotated image and the unrotated one is

$$A'_{nm} = A_{nm} \exp(-jm\theta).$$ \hspace{1cm} (A2.4)

This relation shows that Zernike moments have rotational transformation properties; each Zernike moment merely acquires a phase shift on rotation. This property indicates that the magnitudes of Zernike moments of a rotated image remain identical to those before rotation. Thus $|A_{nm}|$, the magnitude of Zernike moment, can be taken as a rotation invariant feature. Since $|A_{00}| = \omega / \pi$ and $|A_{11}| = 0$, for all the normalized images, these features are not included in a feature vector.