CHAPTER 5

NEURAL NETWORK BASED OBJECT RECOGNITION IN THE FRAMEWORK OF PICTURE DESCRIPTION LANGUAGES

5.1. The shortcomings of Traditional Pattern Recognition Systems and the need for Adaptive Processing

With the introduction of advanced software tools the imaging applications like Automatic Object Recognition (AOR) detects objects in the real world from an image or image sequence of the world using different object models which are known a-priori. Humans can perform this task of object recognition effortlessly and instantaneously. However, such a task of implementation of object recognition based on machines is performed using algorithms and has been very difficult in networks for real time imaging problems which are non-linear in nature. In some applications, the patterns or objects to be recognized are so fuzzy that they cannot be modelled with conventional tools. In order to solve these,
neural network processors can be used as the best tool. It is because that the processor can build a recognition engine from simple image annotations made by the programmer. The characteristics or feature vectors from the annotated objects are extracted and then sent to the neural network. Neural networks having adaptive processing elements are capable of performing generalization and can consequently classify situations never seen before by associating them to similar learned situations.

### 5.2. Artificial neural networks

Learning from a set of examples is an important attribute needed for most pattern recognition systems. Artificial neural network is an adaptive system that changes its structure based on external or internal information that flows through the network and are being widely used in pattern recognition systems [72]. They have a better performance in non-linear applications. Artificial neural networks are trained, so that a particular input leads to a specific target output. The network is adjusted, based on a comparison of the output and the target, until the difference between the target and the output reduces to the minimum value [30]. The neural network design part consists of two processes, training and
application. The training of the neural network continues until the mean squared error of the weights lies below a certain threshold or until the maximum number of iterations is reached. Once training is completed, the network can be applied for the actual classification of the data [38].

The classification technique used may be one of the following:

1) Supervised classification - in which the input pattern fall as a member of a predefined class.

2) Unsupervised classification - in which the pattern falls into an unknown class as there are no predefined classes.

The recognition problem here is a classification or categorization problem, where the classes are either predefined by the system designer or are learned based on the similarity of patterns. It is important to note that learning or training takes place only during the design phase of a pattern recognition system [27]. Once the results obtained are satisfactory, the system is ready to perform the task of recognition on samples drawn from the environment in which it is expected to operate.

5.2.1. Feed-Forward neural networks

Feed-forward networks are commonly used for pattern recognition. A three-layer feed forward neural network is typically composed of one input layer, one output layer and one hidden layer. In
the input layer, each neuron corresponds to a feature vector of a given input pattern while in the output layer each neuron corresponds to a predefined pattern. The best situation is that once a certain sample is input into the network, the output will be a vector with all elements as zero only except the one corresponding to the pattern that the sample belongs to. Of course, it is very complex to construct such types of neural networks.

Training a neural network model essentially includes selecting a model that minimizes the cost criterion. A commonly used cost is the mean-squared error which tries to minimize the average error between the network's output, \( f(x) \), and the target value \( y \) over all the example pairs. There are numerous algorithms available for training neural network models. Most of them can be viewed as a straightforward application of optimization theory and statistical estimation. Most of the algorithms used in training artificial neural networks are employing some form of gradient descent. This is done by simply taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient descent direction. The commonly used networks for minimizing the cost are multi-layer-feed-
forward neural networks, which uses the back-propagation learning algorithm for training neural networks [54].

5.2.2. Back-propagation (BP) Algorithm

Multi-layer feed-forward networks have been used as powerful classifiers. An effective iterative gradient-descent procedure has been developed for training these types of networks [19] [22]. The training of back-propagation algorithm involves 4 stages

1. Initialization of weights
2. Feed forward stage
3. Back propagation of errors
4. Updating of weights and biases

During the feed-forward stage each layer in the network calculates its activation value and passes it to the layers in the next level. The neurons in the output layer produce the output of the network using its activation function and compare it with the target output to determine the error. During the back-propagation stage the error is back propagated from the output layer to each layer in the previous level for the correction of the adjustable parameters weight and bias [81]. This process repeats until the error reaches a minimum threshold or until the maximum number of iterations performed.
5.3. Knowledge vector Analysis

As discussed in chapter V, the direction codes in the knowledge vector obtained by tracing the contour of an object are R, DR, D, DL, L, UL, U and UR. For better processing, let us categorize these directions into basic directions R, D, L and U and diagonal directions DR, DL, UL and UR. For example, consider the knowledge vector of a square in Figure 5.1.

Figure 5.1: The knowledge vector of a square

This shows that square consists of 4 basic directions with all lengths equal. Shapes with all the basic directions but with equal alternate length can be identified as a rectangle. Shapes with 4 sides consisting of only diagonal directions can be considered as rhombus.

The task of identifying regular shaped objects is simple. Objects in a real time environment may not be regular always. For the effective identification of those objects, the shapes of the objects can be approximated to some regular polygons to which it is closer to. This is possible by predefining the classes of regular shapes and approximating
the other shapes to one of these shapes by identifying them as a member of one of the predefined classes.

It is important to note that, the diagonal sides present in the shape of an object do not necessarily appear alone by connecting only the diagonal pixels. The amount of other pixels appear depends on the angle at which the diagonal lines appear in the image. Possibly D and R pixels appear more with DR, D and L pixels appear more with DL, U and L pixels appear with UL and U and R pixels appear with UR. For example consider a line with the direction code DR. As shown in Figure 5.2, if the line bends more towards right the more R pixels appear, in the same way if it bends less towards right the more D pixels present with DR. In this aspect when the shapes are rotated with small angle changes they can be approximated to one of these shapes.

![Figure 5.2: 3 Lines with direction code DR](image)

3 sided shapes

Listed below are the 3 sided shapes and rotated in different angles and their direction codes.
It can be observed that all the 3 sided shapes satisfy the following condition:

3 sides with 1 basic sides and 2 diagonal directions or 2 basic and 1 diagonal directions.

**4 sided shapes**

- **Square**

```
R-D-L-U          DR-DL-UL-UR
```

4 basic sides or 4 diagonal sides with all equal length.

- **Rectangle**

```
R-D-L-U          R-D-L-U          DR-DL-UL-UR
```

4 basic sides or 4 diagonal sides with alternate lengths equal.

- **Parallelogram**

```
R-DL-UR          DR-D-UL-U          R-DR-L-UL
```

2 basic sides and 2 diagonal sides DL and UR or DR and UL.
- Trapezoid

4 sides with 2 basic sides and 2 diagonal sides DL and UL or DR and UR or UL and UR or DR and DL.

In the same manner, shapes with more sides can be predefined. When the shape of an object appears it can be then classified by artificial neural networks as a member of one of these classes. The Table 5.1 shows some of the classes of regular polygons.

Table 5.1: The predefined classes of basic shapes with their vector codes

<table>
<thead>
<tr>
<th>No</th>
<th>No of sides</th>
<th>Class</th>
<th>Shapes</th>
<th>Code</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>3 Sides</td>
<td>Triangle</td>
<td><img src="image" alt="Triangle Diagram" /></td>
<td>DR-L-UR, R-DL-UL,</td>
<td>3 sides with 1 basic side and 2 diagonal sides or 3 sides with 2 basic side and 1 diagonal sides</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR-DL-U, D-UL-UR,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R-DL-UL, D-UL-UR,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR-DL-U, DR-L-UR,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR-L-U, R-DL-U,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D-L-UR, R-D-UL,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D-UL-UR, DR-L-UR,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR-DL-U, R-DL-UL</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>4 sides</td>
<td>Square</td>
<td><img src="image" alt="Square" /></td>
<td>R-D-L-U, DR-DL-UL-UR, DR-DR-L-UL-UR</td>
<td>4 sides with all lengths equal</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rectangle</td>
<td><img src="image" alt="Rectangle" /></td>
<td>R-D-L-U, R-D-L-U, DR-DL-UL-UR</td>
<td>4 sides with alternate lengths equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallelogram</td>
<td><img src="image" alt="Parallelogram" /></td>
<td>R-DL-DR-L-U, DR-L-UL-U, R-DR-L-L-U</td>
<td>4 sides with 2 basic sides and 2 diagonal sides DR and UL or DL and UR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trapezoid</td>
<td><img src="image" alt="Trapezoid" /></td>
<td>R-DL-L-U, DL-U-UR, R-DR-L-U, DR-DL-L-U</td>
<td>4 sides with 2 basic sides and 2 diagonal sides DR and UR or DL and UL or DR and DL or UL and UR.</td>
</tr>
<tr>
<td>3.</td>
<td>5 sides</td>
<td>Pentagon</td>
<td><img src="image" alt="Pentagon" /></td>
<td>DR-DL-UL-UR, DR-DL-U-UR, R-DR-DL-UL-UR, DR-DL-L-U-UR</td>
<td>5 sides with 1 basic side and 4 diagonal sides</td>
</tr>
<tr>
<td>4.</td>
<td>6 sides</td>
<td>Hexagon</td>
<td><img src="image" alt="Hexagon" /></td>
<td>R-DR-DL-L-U-UR, DR-DL-U-UR, R-DR-DL-L-U-UR, DR-DL-L-U-UR</td>
<td>6 sides with 2 basic sides and 4 diagonal sides</td>
</tr>
<tr>
<td>5.</td>
<td>7 sides</td>
<td>Heptagon</td>
<td><img src="image" alt="Heptagon" /></td>
<td>DR-DL-L-U-UR, R-DR-DL-L-U-UR, R-DR-DL-L-U-UR, R-DR-DL-L-L-U UR</td>
<td>7 sides with 3 basic sides and 4 diagonal sides</td>
</tr>
<tr>
<td>6.</td>
<td>8 sides</td>
<td>Octagon</td>
<td><img src="image" alt="Octagon" /></td>
<td>R-DR-DL-L-U-UR, R-DR-DL-L-U-UR, R-DR-DL-L-U-UR, R-DR-DL-L-L-U UR</td>
<td>8 sides with 4 basic sides and 4 diagonal sides</td>
</tr>
</tbody>
</table>
The knowledge vector can be more informative by separating the basic sides and the diagonal sides. Initially, it can be viewed as a string with every small change reflected and then limited to 8 directions. From the initial string obtained, the continuous pixels that make the straight lines (chosen only if at least 2% pixels present) can be extracted first and the remaining pixels can be approximated to diagonal directions. Let us consider an image of a triangle

![Triangle Image](a) Image ![Triangle Contour](b) Contour

Figure 5.3: Image of a Triangle and its contour

The knowledge vector obtained is

\[
<75,121>/D1*DR2*D1*DR3*R1*DR1*D1*DR3*D1*DR3*D1*DR7*D1*DR5*D1*DR2*D1*DR3*R1*DR2*D1*DR3*D1*DR7*D1*DR5*D1*DR1*D1*DR8*D1*DR4*D1*DR3*D1*DR3*R1*DR1*D1*DR3*D1*DR2*L175*UR3*U1*UR2*U1*UR2*R1*UR3*U1*UR2*U1*UR7*D1*U2*UR7*U2*UR5*R1*UR1*U1*UR2*U1*UR7*U1*UR5*U1*UR1*U1*UR4*U1*UR7*U2*UR1*R1*UR5*U1*UR2*U1*UR4*U1*UR1*R1*UR2*U1*UR2*/<77,120>#
\]

Vector limited to single occurrence of 8 directions as

\[
<75,121>/R10*DR82*D24*L175*U22*UR82*/<77,120>#
\]
And then approximated to

→ DR116*L175*UR104

Vector code normalized to 100 pixels

DR29* L44* UR26

If we observe the knowledge vector obtained, L175 (L44 in the normalized vector) appears as a vector for a straight line with no intermediate changes in directions. Such vectors can be separated as basic directions. The other directions present are DR21, D6, U6 and UR21. In this case, D appears with DR and U appears with UR. So, these lengths can be added to get the directions DR and UR. Thus we obtain the vector code: DR116*L175*UR104. This code is then normalized as DR29* L44* UR26.

Let us now consider the triangle that is rotated by 90° to the left,

![Image of a rotated triangle](image)

(a) Image  (b) Contour

Figure 5.4: Image of a rotated triangle and its contour

The Knowledge vector obtained is


Vector limited to 8 directions as

<34,170>/R15*D184*UL84*U15*UR84*/<35,169>#
Approximated to

\[ \rightarrow D184*UL107*UR107 \]

and, the normalized vector \[ - D47*UL27*UR27 \]

It is observed that these 2 vector codes has 1 basic direction and 2 diagonal directions and can be classified as triangle.

In the same way the normalized vector of an image of apple can be found by,

![Image of an apple and its contour](image)

Figure 5.5: Image of an apple and its contour
5.4. Neural Network Architecture

The neural network architecture with back propagation algorithm is shown in Figure 5.6.

![Figure 5.6: Architecture of a two-layer feed forward network with back propagation.](image)

The weights are initialized to some random values. During the feed forward stage each neuron in the input unit $x_i$ receives input signal and transmits it to the neurons in the hidden unit $z_1$ to $z_p$. Each hidden layer unit calculates the activation function and sends its signal $z_j$ to the neurons in the output unit. The output unit calculates the activation function $y_k$ to produce the response of the net for the given input pattern. During the back propagation of errors the neurons in each output unit compares its computed activation $y_k$ with its target value $t_k$ to determine
the error for that pattern. Based on the error the factor $\delta_k$ is computed and is used to distribute the error at output unit $y_k$ back to all units in the previous layer. Similarly the factor $\delta_j$ is computed for each hidden unit $z_j$. Updating of weights is done by computing the weight correction term $\Delta w_{ij}$ and $\Delta w_{jk}$ and adding it to the old weight parameter. The algorithm is given in steps below:

Parameters used:

- $x = (x_1, x_2, \ldots, x_{n})$ - Input training vector
- $t = (t_1, t_2, \ldots, t_k, \ldots, t_m)$ - Output target vector
- $\delta_k$ = error at output unit $y_k$
- $\delta_j$ = error at hidden unit $z_j$
- $\alpha$ = learning rate
- $v_{ij}$ = weights of input layer
- $v_{oj}$ = bias on hidden unit $j$
- $z_j$ = activation of hidden unit $j$
- $w_{jk}$ = weights of hidden layer
- $w_{ok}$ = bias on output unit $k$
- $y_k$ = activation of output unit $k$

1. Initialize the weights to small random values
2. While the stopping condition is false, do steps 3 to 10
3. For each training pair do steps 4 to 10

4. Each input receives the input signal $x_i$ (i=1, $\cdots$, n) and transmits it to all units in the hidden layer $z_j$ (j=1, $\cdots$, p).

5. Each hidden unit $z_j$ sums its weighted input signals

$$ = v + \sum \ldots \ldots \quad (1)$$

and applies its activation function

$$ = (\quad \ldots \ldots \quad (2)$$

and sends this signal to all units in the output layer $y_k$ (k=1, $\cdots$, m).

6. Each output unit $y_k$ (k=1, $\cdots$, m) sums its weighted input signals

$$ = w + \sum \ldots \ldots \quad (3)$$

and applies its activation function to calculate the output signals

$$ = (\quad \ldots \ldots \quad (4)$$

7. Each output unit $y_k$ receives a target pattern $t_k$ corresponding to an input pattern and calculates the error term as

$$ = - '(\quad \ldots \ldots \quad (5)$$

8. Each hidden unit $z_j$ sums its delta inputs from units in the layer above. The error information term is calculated as,

$$ = \sum '\(\quad \ldots \ldots \quad (6)$$

9. Each output unit $y_k$ updates its bias and weights.
The change in weight is given by

\[ \Delta = \ldots \ldots \quad (7) \]

and the bias correction term is given by

\[ \Delta = \ldots \ldots \quad (8) \]

due to

\[ = + \Delta \quad \ldots \ldots \quad (9) \]

and,

\[ = + \Delta \quad \ldots \ldots \quad (10) \]

10. The hidden unit \( z_j \) updates its bias and weights. The weight correction term is given by

\[ \Delta = \ldots \ldots \quad (11) \]

and the bias correction term is

\[ \Delta = \ldots \ldots \quad (12) \]

due to

\[ = + \Delta \quad \ldots \ldots \quad (13) \]

and

\[ = + \Delta \quad \ldots \ldots \quad (14) \]

11. Test the stopping condition
5.4.1. Levenberg-Marquardt Algorithm

The back-propagation algorithm known as Levenberg-Marquardt back propagation (trainlm) present in matlab toolbox is used to train the network. This algorithm appears to be the fastest method for training moderate-sized feed forward neural networks. \textit{trainlm} is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization.

The network \textit{trainlm} can train any network as long as its weight, net input, and transfer functions have derivative functions. Back propagation is used to calculate the Jacobian $jX$ with respect to the weight and bias variables $X$. Each variable is adjusted according to Levenberg-Marquardt equation,

\begin{align*}
jj &= jX \ast jX \\
je &= jX \ast E \\
dX &= -(jj + I*mu) \backslash je
\end{align*}

where $E$ is all errors and $I$ is the identity matrix. The adaptive value $mu$ is increased until the change results in a reduced performance value. When $mu$ is zero, this is just Newton's method, using the approximate Hessian matrix. When $mu$ is large, this becomes gradient descent with a small step size. As Newton's method is faster and more accurate near an
error minimum the aim here is to shift towards Newton's method as quickly as possible. Thus, $\mu$ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way the performance function will always be reduced at each iteration of the algorithm.

Training stops when any of these conditions occur:

1. The maximum number of iterations is reached.
2. The maximum amount of time has been exceeded.
3. Performance has been minimized to the goal.
4. The performance gradient falls below minimum.
5. $\mu$ exceeds maximum.
6. Validation performance has increased more than max times since the last time it decreased.

5.5. Results

Neural network with 8 neurons in the input layer and 9 neurons in the output layer each representing 9 different classes of shapes is trained with a set of input patterns. Figure 5.7 shows the neural network created for the problem defined.
5.5.1. Network Predictive Ability and General Performance

Mean Squared Error (MSE) is the performance metric that determines the network performance and regressions (R) is used to measure the correlation between outputs and targets. MSE is the average squared difference between outputs and targets. Lower values of MSE shows the better performance of the network as zero indicate no error while R value of 1 indicates closed relationship and 0 indicates a random relationship.

Figure 5.8 shows the performance of the neural network.

Figure 5.8: Neural network performance
A linear regression is performed between the network outputs and the targets to check the quality of the network training. A linear regression between each element of the network response and the corresponding target takes the array of network outputs and the array of targets as inputs and returns the slope of the linear regression computing the R-value. Figure 5.9 shows the fitting curve between the network response and the target and the MSE and Regression is tabulated in Table 5.2.

![Regression](image)

Figure 5.9: Regression

Table 5.2: The MSE and R of the network.

<table>
<thead>
<tr>
<th></th>
<th>MSE</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>0.0015</td>
<td>0.9926</td>
</tr>
<tr>
<td>Validation</td>
<td>0.2397</td>
<td>0.8918</td>
</tr>
<tr>
<td>Testing</td>
<td>0.2655</td>
<td>0.4024</td>
</tr>
<tr>
<td>All</td>
<td>0.0240</td>
<td>0.8261</td>
</tr>
</tbody>
</table>
From the results shown above, it is proven that the network trained with Levenberg-Marquardt back propagation algorithm has good ability to identify the objects with high regression correlation and with less error for both training and testing indicating good accuracy. Performance of the network has shown a promising result. Further investigation with larger sample and different network properties will definitely improve predictive ability and general performance of ANN.