CHAPTER 6

IRIS AUTHORIZATION

6.1 INTRODUCTION

The authorization process of the biometric system is used to validate or invalidate the test input. The Artificial Neural Network (ANN) approach is used for the authorization process. The function of ANN is similar to human brain. The human brain consists of huge number of neurons. The ANN also consists of neurons in the hidden layer. The connectivity of brain neurons is made through the nervous system. The ANN neurons are connected through the synaptic weights.

The human sense organs are the input for brain. The ANN consists of input layer to collect the inputs. The brain performs the decision making process based on the sense input. The ANN input is connected with the hidden layer which acts as a decision maker. The brain gives the output through the appropriate part of the human. The hidden layer of the neural network is connected with the output layer. The network gives the output through the appropriate output layer.

The authorization is performed using the FPGA based approach. The FPGA consists of Configurable Logic Blocks (CLB), inter connection switch matrix, input and output ports. The input, output and decision making process are configured in the FPGA hardware. The inputs are given to the FPGA which makes the decision process and gives the appropriate outputs.
The various biometric parameters are used to test the performance of the biometric system. The parameters are accuracy, precision, sensitivity, specificity, positive predictive value and negative predictive value.

### 6.2 TERMINOLOGIES OF ANN

The various terminologies used to design neural network are weight, bias, activation function and threshold (Jayaraman et al 2009).

#### 6.2.1 Weight

The neurons are connected to the network through the weights. The weights are adjusted during the training process. Initially the weights are fixed and during training, they are adjusted to get the optimal result. The weights are fixed after the perfect training of the network.

![Figure 6.1 Neural Network with weight parameter](image)

The network with the connected weights is shown in the Figure.6.1. The input $X_1$ is connected through the weight $W_1$. Similarly $X_2$ and $X_3$ inputs are connected through the weights $W_2$ and $W_3$. The total network input ($Y_{input}$) can be given using the Equation (6.1).

$$ Y_{input} = X_1 W_1 + X_2 W_2 + X_3 W_3 $$  (6.1)
If the total numbers of inputs are ‘n’, then network is given using the Equation (6.2).

\[ Y_{\text{Input}} = \sum_{i=1}^{n} X_i W_i \]  

(6.2)

### 6.2.2 Bias

The parameter bias is similar to the weight. The bias value is added in the neural input. The ANN with the bias function is shown in the Figure 6.2. The net input for the network is given in the Equation (6.3).

![Neural network with weight and bias function](image)

**Figure 6.2 Neural network with weight and bias function**

\[ Y_{\text{Input}} = \text{Bias} + X_1 W_1 + X_2 W_2 + X_3 W_3 \]  

(6.3)

If the total numbers of inputs are ‘n’, then network is given using the Equation (6.4).

\[ Y_{\text{Input}} = \text{Bias} + \sum_{i=1}^{n} X_i W_i \]  

(6.4)
6.2.3 Activation functions

The activation functions are used to decide the output of the neural network. The neural network model with the activation function is shown in the Figure 6.3.

![Neural network model with activation function](image)

**Figure 6.3 NN with input and output activation function**

The output of network is the function of all inputs. The various types of activation functions available are identify function, binary step function, binary sigmoid (logsig), bipolar sigmoid (tansig) and threshold function.

6.2.3.1 Identify function

The output of the identify function is equal to the input. The output \( Y_{out} \) is the function of input signal as given in the Equation (6.5). The graphical representation of identify function as shown in the Figure 6.4.

\[
Y_{out} = f(Y_{input}) = Y_{input}
\]  

(6.5)
6.2.3.2 Binary step function

The binary step function produces 2 states of outputs such as ‘0’ and ‘1’. The limit value ‘\( \theta \)’ is fixed. The output depends on the limit value. The output of binary step function is represented in the Equation (6.6). When the network input \( (Y_{\text{input}}) \) is greater than or equal to ‘\( \theta \)’ then the output \( (Y_{\text{out}}) \) is ‘1’. Otherwise the output is ‘0’. The graphical representation of step function is shown in the Figure 6.6.

\[
Y_{\text{out}} = f(Y_{\text{input}}) = \begin{cases} 
1 & \text{for } Y_{\text{input}} \geq \theta \\
0 & \text{for } Y_{\text{input}} < \theta 
\end{cases} \quad (6.6)
\]
6.2.3.3 Binary sigmoid function

The binary sigmoid function is also called as the logsigmoidal function. The output value is distributed from 0 to 1. The term ‘σ’ in the Equation (6.7) represents the steepness factor. The graphical representation of binary sigmoid function is shown in the Figure 6.6.

\[ Y_{out} = f(Y_{input}) = \frac{1}{1 + \exp(-\sigma Y_{input})} \]  

(6.7)

Figure 6.6 Illustration of binary sigmoid function

The output of binary sigmoid function is obtained using the Equation (6.7).

6.2.3.4 Bipolar sigmoid function

The bipolar sigmoid function is also called as the tansigmoidal function. The output values are distributed from -1 to +1. The output of the activation function is obtained using the Equation (6.8).

\[ Y_{out} = f(Y_{input}) = \frac{1 - \exp(-\sigma Y_{input})}{1 + \exp(-\sigma Y_{input})} \]  

(6.8)
The graphical representation of the bipolar sigmoid function is shown in the Figure 6.7.

![Figure 6.7 Illustration of tansig function](image)

6.2.3.5 Threshold Function

The threshold function is very similar to the binary step function. The output values are either ‘+1’ or ‘-1’ based on the threshold value.

\[ Y_{out} = f(Y_{input}) = \begin{cases} 
1 & \text{for } Y_{input} \geq \theta_i \\
-1 & \text{for } Y_{input} < \theta_i
\end{cases} \tag{6.9} \]

where, ‘\( \theta_i \)’ represents the threshold value. The threshold function is given using the Equation (6.9). The output of the threshold function is illustrated in the Figure 6.8.

![Figure 6.8 Illustration of threshold function](image)
6.3 ANN LEARNING METHODS

6.3.1 Supervised Learning

The supervised learning method consists of a teacher network. The teacher network is a well trained network. The input is given to both teacher network and the learning network. The outputs are compared and the error signal is generated.

6.3.2 Unsupervised Learning

The unsupervised learning method consists of input system and learning network. There is no teacher network or supervised network. The target is not defined in the network. The learning is randomly changed according to the input. The learning is very difficult and complex to implement.

6.4 ANN LEARNING RULES

The learning rules are used to train the network. It activates the weights and biasing of the network based on the generated error signals. The various learning rules are available to train the network. The Hebbian rule increases the weights between the neurons, when both the neurons are in the active high value. The hop-field learning rule strengthens the weights between the neurons, when both the neurons are in active high. Otherwise the weights are reduced.

The delta learning rule calculates the error using the least mean square error. The error signal is back propagated and weights are adjusted. It is also called as Widrow- Hoff learning rule. The gradient descent rule is very
similar to the delta learning rule. In this a constant additional learning rate is added with the existing learning rate. The Kohonen learning rule is the competitive learning rule. The maximum output element is called as the winner. The winner and its neighbour neuron weights are updated.

### 6.5 BASIC NETWORK ARCHITECTURE

The basic network architecture consists of input layer, hidden layer and the output layer as shown in the Figure 6.11.

![Figure 6.9 Basic network architecture](image)

The inputs form the input layer. The terms $X_1$, $X_2$, $X_3$, $X_4$,...,$X_n$ represent the inputs. The ‘$W$’ represents the weight matrix and is given in the Equation (6.9). The elements of weight matrix are the synaptic weight between the input layer and hidden layer.
The number of rows and columns of the weight matrix are based on the number of inputs and neurons in the hidden layer. The hidden layer consists of ‘n’ hidden neurons $N_1$, $N_2$, $N_3$, $N_4$,...,$N_n$ and are energized using the weights of input layer and the biasing elements $b_1$, $b_2$, $b_3$, $b_4$,...,$b_n$. The terms ‘m’ and ‘q’ represents the input and total number of inputs associated with the neuron $N_n$ given in the Equation (6.11).

$$N_n = \sum_{m=1}^{q} (w_{mn} \cdot x_m) + b_n$$  \hspace{1cm} (6.11)

The activation functions are applied on hidden neurons and given to the output layer. The single layer network consists of only one hidden layer. The multilayer network consists of more than one hidden layers. The outputs $z_1$, $z_2$, $z_3$, $z_4$,...,$z_n$ form the output layer. The output layer also consists of activation function (Sibai et al 2011).

### 6.6 FEED FORWARD BACK PROPAGATION NEURAL NETWORK (FFBPNN)

The simple FFBPNN consists of one or more number of hidden layers. The input is continuously proceeding in the forward direction of the network. The training process alone takes the backward path (Yogendra & Manoj 2012).

The FFBPNN architecture is the advanced version of simple FFNN. The output of the previous stage is fed back to the present stage. Every stage
of the forward path is fed back to the previous stage. The single stage FFBPNN network is shown in the Figure 6.10. The single stage FFBPNN network consists of 3 layers namely input layer, hidden layer and the output layer. The output of the hidden layer is fed back to the input layer. The output layer is fed back to the hidden layer.

![Figure 6.10 Single stage FFBPNN](image)

![Figure 6.11 Multi stage FFBPNN](image)
The multi stage FFBPNN network is shown in the Figure 6.11. It consists of 3 hidden layers. The output of first hidden layer is connected with the second hidden layer and the second layer output is connected with the 3rd hidden layer. The final hidden layer output is given to the output layer. The output layer gives out the final network output. The output is fed back to the 3rd hidden layer. The 3rd layer output is feedback to 2nd layer and 2nd layer output is fed back to the 1st hidden layer. The First layer output is feedback to the input layer.

The FFBPNN performs the operation using 3 different stages. There are feed forward stage, error back propagation and updation of weights.

6.6.1 Feed Forward Stage

The network is initialized using the weights and biasing values. The input of the hidden neuron is calculated using the Equation (6.12). The terms N_input, Bm, x and M represent the input of hidden layer, biasing element of hidden neurons, input vector, weights between the input layer and hidden layer respectively. The terms ‘i’ and ‘j’ correspond to the input and hidden layers respectively.

\[ N_{\text{input}}_j = Bm_j + \sum_{i} x_i M_{ij} \] (6.12)

The output produced by the hidden neuron is calculated using the Equation (6.13). The term ‘N_output’ represents the output of the hidden layer.

\[ N_{\text{output}}_j = f \left( N_{\text{input}}_j \right) \] (6.13)
The input to the output layer is calculated using the Equation (6.14). The terms ‘Z-input’ and ‘Bw’ represent the inputs given to the output layer and the Biasing of output layer. The term ‘k’ corresponds to the output layer.

\[ Z_{\text{input}}_k = Bw_k + \sum_{j} N_{\text{output}}_j W_{jk} \]  

(6.14)

The output produced by the output layer is calculated using the Equation (6.15). The term ‘Z_output’ represents the output of the network.

\[ Z_{\text{output}}_k = f \left( Z_{\text{input}}_k \right) \]  

(6.15)

6.6.2 Back Propagation Stage

The output produced in the output layer is compared with the target (desired) and error signal is back propagated to the hidden layer. The hidden layer error signal is back propagated to the input layer. The error correction parameter between the output and hidden layers is calculated using the Equation (6.16) which is used to adjust the weight. The change in weight between the output and hidden layer is computed using the Equation (6.17).

\[ \delta_k = (t_{\text{target}}_k - Z_{\text{output}}_k) f' \left( Z_{\text{input}}_k \right) \]  

(6.16)

\[ \Delta W_{jk} = \alpha \cdot \delta_k \cdot Z_{\text{output}}_j \]  

(6.17)

The terms ‘\( \delta \)’, ‘\( \alpha \)’, ‘\( t_{\text{target}} \)’ and ‘\( \Delta W \)’ represents the error correction parameter, learning rate, targeted output and change in weight between hidden and output layer.

The error correction parameter between hidden and input layer is calculated using the Equation (6.18). The change in weight between the
the hidden and input layer is calculated using the Equation (6.19). The term ‘\( W_{jk} \)’ represents the weights between the hidden layer \((j)\) and output layer \((k)\).

\[
\delta_j = \sum_{k} \delta_k W_{jk} \cdot f' (N_{\text{input}_j}) \quad (6.18)
\]

\[
\Delta M_{ij} = \alpha \delta_j x_i \quad (6.19)
\]

### 6.6.3 Weight and bias updation stage

The back propagation gives out the changes in weight at the consecutive stages. The weight and bias value updation between the hidden layer and the output layer is calculated using the Equation (6.20). The weight and bias updation between the input layer and the hidden layer is calculated using the Equation (6.21).

\[
W_{jk} (new) = W_{jk} (old) + \Delta W_{jk} \quad (6.20)
\]

\[
M_{ij} (new) = M_{ij} (old) + \Delta M_{ij} \quad (6.21)
\]

### 6.7 FFBPNN TRAINING METHODS

The various training methods are available to train the FFBPNN such as Levenberg-Marquardt (LM), Resilient back Propagation (RP), Conjugate Gradient Fletcher-Reeves (CGF), Scaled Conjugate Gradient (SCG), Conjugate Gradient Polak (CGP), Broyden Fletcher Goldfarb Shanno (BFGS), One Step Secant (OSS) and Gradient Decent with momentum (GDX) (Usham et al 2010) and (Demuth et al 2009).

### 6.7.1 LM Algorithm

The LM algorithm is one of the back propagation algorithms. It uses the gradient decent method for updating weights and biasing values. The
change in bias and the weight ‘dx’ is calculated using the Equation (6.22). The terms ‘J’, ‘I’, ‘μ_u’ and ‘e’ represent the jacobian matrix, identity matrix, learning rate and error signal.

\[ dx = \frac{-(JJ + I \mu_u)}{Je} \]  

(6.22)

### 6.7.2 RP Algorithm

The RP algorithm also works based on gradient descent method. The change in weight and bias value (dx) is calculated using the Equation (6.23). The current weight and bias value is ‘delta_x’ and gradient parameter ‘gx’ are the 2 variable parameters in this method. The direction of weight updation depends on the gradient direction.

\[ dx = \text{delta}_x \cdot \sin(gx) \]  

(6.23)

### 6.7.3 CGF Algorithm

It is developed by Fletcher-Reeves. The search is performed in the conjugate direction. The convergence is faster when compared with the steepest decent directions. The change in weight and bias values is calculated using the Equation (6.24). The terms ‘gx’, ‘gx_1’ and ‘dx_old’ represents the current gradient value, previous gradient and the previous weight and bias values respectively.

\[ dx = -gx + dx_{\text{old}} \cdot \left( \frac{(gx)^2}{(gx_1)^2} \right) \]  

(6.24)
6.7.4 SCG Algorithm

The scaled conjugate gradient algorithm is similar to the CGF algorithm which is developed by Moller. The line by line search is not performed. It requires more iteration to achieve the perfect training. But the computation complexity is reduced in the proceeding iteration.

6.7.5 CGP Algorithm

The algorithm is developed by Polak and Ribiere. The algorithm is similar to the CGF algorithm. The change in weight and bias values is calculated using the Equation (6.25).

\[ dx = -gx + dx_{old} \left( \frac{gx - gx_{-1}}{gx_{-1}} \right) \]  

(6.25)

6.7.6 BFGS Algorithm

The BFGS is the alternate method for the conjugate gradient decent algorithm. The change in weight and bias value is calculated using the Equation (6.26). The term ‘H’ represents the Hessian matrix.

\[ dx = -\frac{H}{gx} \]  

(6.26)

6.7.7 OSS Algorithm

It is similar to the BFGS algorithm. The computation complexity and the memory storage are very less when compared with the BFGS algorithm. The change in weight and bias value is calculated using the Equation (6.27). The parameters ‘A’ and ‘B’ are additional parameters used to adjust the weights.

\[ dx = -gx + \left( A \cdot dx_{old} \right) + \left( B \cdot gx_{-1} \right) \]  

(6.27)
6.7.8 GDX Algorithm

In the variable gradient algorithm the adaptive learning rate is used. The change in weight and bias value is calculated using the Equation (6.28). The term ‘mc’ represents the momentum coefficients. The term ‘d_{perf}’ represents the performance of the network with respect to change in weight and bias value.

\[ dx = mc \cdot dx_{old} + \left( \alpha \cdot mc \cdot d_{perf} \right) \frac{dx}{dx} \]  

(6.28)

The entire algorithms are to be trained and tested using iris feature. The suitable method is used for the proposed work.

6.8 HARDWARE DESCRIPTIVE LANGUAGE (HDL)

The HDL is having the digital logic libraries. The digital logic libraries consist of logical functions, Boolean operations, relational operation, addition, subtraction, multiplication, division and comparison. The Verilog and VHDL are the two familiar HDLs. The HDL supports for concurrent as well as sequential execution. The traditional ‘C’ language supports only the sequential mode execution. The HDLs are used to model the digital logic circuit from various levels of abstraction. The VHDL and Verilog are similar languages. Both the languages have unique advantages.

6.9 VHDL

The VHDL language consists of entity declaration part and architecture body. The entity part is used to declare the input and output port of the hardware. It also supports the bi-directional input and output port. The values used in the VHDL are single bit or an array of binary values. The architecture part is used to define the function of the hardware. The execution of architecture part is either in the sequential mode or concurrent mode or
combination of both. The functional blocks are implemented using different VHDL modelling. The modelling is selected depending upon the application.

6.9.1 Structural Modelling

The structural level modelling is carried out using the pre-defined logical blocks or digital components. The complete circuit is modelled by making connections between the components. The real digital circuits are modelled using this level of abstraction. The structural modelling is operated in the concurrent mode. For example, consider the full adder circuit. The gates functions are already defined in the library. The connections between the gates are made to model the full adder circuit.

6.9.2 Data Flow Modelling

The data flow modelling is operated in the concurrent mode. The equations or data flow statements are directly implemented in the functional body. For example, full adder circuit is implemented in the data flow modelling using the logical equations.

6.9.3 Behavioural Modelling

The behavioural modelling executes the program in the sequential mode. The looping statements such as “if-else”, “for” and “switch-case” are used within the functional body. For example, full adder circuit is implemented in the behavioural modelling in sequential mode. The truth table for the full adder is coded using the “if-else” or “switch-case” statement.

6.9.4 Mixed Style of Modelling

In the mixed modelling the combination of structural, data flow and behavioural modelling are used. The modelling is operated in the combination of sequential and concurrent. The complex circuit blocks use the mixed style
of modelling. The complex blocks are divided into simple blocks. The concurrent execution is performed inside the simple blocks. The block data is dependent on each other. Therefore the sequential execution is performed between the blocks.

6.10 HARDWARE IMPLEMENTATION

The implementation of hardware is carried out using the programmable logic devices (Liu-Jimenez et al 2011). The HDL is to be used to develop the coding for hardware implementation (Hentati et al 2012). The developed codes are dumped in suitable physical blocks of the hardware. The various types of programmable devices are listed below.

- Field Programmable Logic Array (FPLA).
- Programmable Logic Array (PLA)
- Programmable Array Logic (PAL).
- Dynamic Array Logic (DLA).
- Complex Programmable Logic Device (CPLD).
- Field Programmable Gate Array (FPGA).
- Application Specific Integrated Circuits (ASIC).

The PLA is the combination of AND matrix and OR matrix. The input part consists of AND matrix and the output part consists of OR matrix. Both input and output matrices are programmable. The PAL is the combination of AND input matrix and the OR output matrix. The input AND matrix is programmable and the output matrix is fixed one. The FPLA is the combination of PLAs. The PLA’s are arranged in the form of matrix. The DLA is similar to PLA. The programming is more flexible than the PLA. The
above discussed PLDs are suitable for implementing the memory, lookup table and memory based applications.

The CPLD is the combination of PLDs. The PLDs are arranged in matrix. The interconnecting wires are used to make the connectivity between the PLD’s. The Input / Output (I/O) blocks are connected with the PLD’s through the interconnecting wires. The I/O blocks are used to carry the external inputs. The CPLDs supports the In-System-Programming (ISP) and is easily programmable. The device consists of thousands of logic blocks with coarse grain architecture and the block sizes are larger. The CPLD are based on the Electrically Erasable Programmable Read Only Memory (EEPROM). The delays produced are easily predictable in the CPLD. Thus the CPLD is suitable for implementing simple applications.

6.11 FIELD PROGRAMMABLE GATE ARRAY

The FPGA is a programmable device which consists of more than 1,00,000 equivalent logic gates where as the CPLD consists thousands of transistors. The block diagram of FPGA is shown in the Figure 6.12. The FPGA consists of Configurable Logic Block (CLB) where as the CPLD consists of PLDs. The FPGA consists of I/O block for making the input and output connections. The interconnection blocks surround the CLBs.

The connections between the CLB’s and I/O blocks are made through the interconnection blocks (Babasaheb et al 2012). The FPGA is a commercial product manufactured by the actel, altera, atmel, lucent, quick logic, vantis and Xilinx.

The block diagram of FPGA is shown in the Figure 6.14. The FPGA consists of major components such as Controllable logic block (CLB),
Input and output block (I/O block) and the switch matrix. The CLB is surrounded by the switch matrix.

6.11.1 CLB

The CLB consists of combinational functional units, multiplexers and registers. The combinational blocks are used to implement the required functions. The inputs are given through the functional units. The CLBs are located between the switch matrices. The required functional blocks are programmed in the CLBs.

6.11.2 Switch Matrix

The switch matrix is used to make the connectivity between the CLB and the I/O ports. The switch matrix lines are taken in the both horizontal and vertical directions. The switch matrices are programmable one. The connectivity is enhanced in the shortest route path.
6.11.3 I/O Ports

The external pins out connections are connected with the I/O blocks. The I/O blocks are connected with the CLB through the switch matrix. The inputs are given to the CLB blocks through the I/O blocks and similarly the outputs of the FPGA are got through the I/O ports.

6.12 XILINX SPATRAN-3E

The spatran-3E FPGA family consists of 1.6 million numbers of gates. The 90 nm transistor technology is used in the system. The spatran-3E FPGA is used for hardware implementation (Bethuna et al 2012). It consists of 612 numbers of CLB. Each CLB consists of 4 numbers of slices. The total number of slices present in the FPGA is around 2448.

It consists of 216 Kb of block RAM and the 38 Kb of distributed RAM. It consists of 172 I/O lines. The cost of the system is very low and also provides higher performance. The FPGA consists of Join Test Action Group (JTAG) connector which is used to configure the device. It is robust and reprogrammable. The routing elements are programmable which has controlled by the Complementary Metal Oxide Semiconductor (CMOS) configuration latches.

6.13 PROPOSED SPATRAN-3E FPGA BOARD

The proposed spatran-3E, FPGA enhancement board is manufactured by the Frontline electronics. The proposed device model is shown in the Figure.6.14. It consists of spatran3E FPGA with the device type of XC2S250e. The speed grade is 4. A Thin Quad Flat Package (TQFP) is used. The number of projected pins is 144.
The proposed FPGA enhancement mode is shown in the Figure 6.16. The device is interfaced with the I/O devices. It consists of 16 binary logic input switches, 8 Light Emitting Diode (LED) outputs, Liquid Crystal Display (LCD) system, key switch matrix, rotary encoder, serial interface, parallel interface, Universal Serial Bus (USB) interface, etc.

6.14 IMPLEMENTATION MODEL

The proposed implementation model consists of trained iris feature database, test iris data base and comparator. The data base images selected
for the proposed work is divided into trained and un-trained data base. The proposed data base consists of 500 images in the data base. The 70% of images are trained and the remaining 30% are untrained images. The test iris data base consists of both trained and un-trained features. The implementation idea is observed from Kamil & Andrzej (2011). The block diagram of the proposed implementation model on FPGA is shown in the Figure 6.15.

![Figure 6.15 Implementation block diagram](image)

### 6.14.1 Test Iris Feature Data Base

The test iris data base is like a ROM block. The floating point MPCA features are converted into binary format with the resolution of 22 bits. The 22 bit data resolution is preferred for accuracy.

The number of images involved in the data base is 500; therefore the 9 bit ($2^9 = 512$) address lines are preferred. The trained and un-trained features are dumped in the test iris feature data base block in the consecutive memory locations. In the proposed FPGA, 9 key switches are used to give the address input. The address inputs are used to select the testable feature.
6.14.2 Comparator and Trained Data Base

The trained data base is like a lookup table. The trained data base consists of 70% (350) of trained iris features. The comparator compares the test data with the trained data base. The comparator is a decision maker. The comparator input is testable feature. It has the 2 external outputs. Both the outputs are connected with the LEDs. One is authenticated and another one un-authenticated. The complete blocks are implemented in the FPGA.

The testable iris data address is selected using the key switches. The corresponding feature is given to the comparator. The comparator compares the test data with the trained one. If the test data is available in the trained data base, the authenticated LED will glow otherwise un-authenticated LED will glow.

6.15 BIOMETRIC PERFORMANCE PARAMETERS

The international biometric consortium and National Science and Technology Council have framed biometric standards [12]. It is used to analyze the performance of biometric system. The various biometric standards are available. The standards are explained in the sections from 6.15.1 to 6.15.11. The developed algorithm gives out the number of authorized and un-authorized person. The total number of authorized and un-authorized persons are computed and based on that the performance parameters are computed.

6.15.1 False Positive (FP)

The FP is used to measure the number of invalid users recognized as the valid users using the Equation (6.29).

\[ FP = \text{Number of authorized person} - \text{Number of trained images} \]  \hspace{1cm} (6.29)
6.15.2 False Negative (FN)

The FN is used to measure the number of valid users recognized as the invalid users using the Equation (6.30).

\[ FN = \text{Number of unauthorized person} - \text{Number of untrained images} \]  (6.30)

6.15.3 True Positive (TP)

The TP is used to measure the number of valid users recognized as the valid users using the Equation (6.31).

\[ TP = \text{Number of trained images} - \text{Number of unauthorized images} \]  (6.31)

6.15.4 True Negative (TN)

The TN is used to measure the number of invalid users recognized as the invalid users using the Equation (6.32).

\[ TP = \text{Number of untrained images} - \text{Number of authorized images} \]  (6.32)

6.15.5 Accuracy

The accuracy of the system defines the degree of the closeness between the system output and the target using the Equation (6.33).

\[ Accuracy = \frac{TP + TN}{\sum (TP + TN + FP + FN)} \]  (6.33)
6.15.6 Sensitivity

The sensitivity is used to measure the positive detection accuracy of the system using the Equation (6.34).

\[
Sensitivity = \frac{TP}{TP + FN}.
\]  \hspace{1cm} (6.34)

6.15.7 Specificity

The specificity is used to measure the negative detection accuracy of the system using the Equation (6.35).

\[
Specificity = \frac{TP}{TP + FP}.
\]  \hspace{1cm} (6.35)

6.15.8 False Acceptance Rate (FAR)

The FAR is used to measure the percentage of wrongly accepted inputs by the system using the Equation (6.36).

\[
FAR = \frac{FP}{\sum (TP + TN + FP + FN)}
\]  \hspace{1cm} (6.36)

6.15.9 False rejection ratio (FRR)

The FRR is used to measure the percentage of wrongly rejected inputs by the system using the Equation (6.37).

\[
FRR = \frac{FN}{\sum (TP + TN + FP + FN)}
\]  \hspace{1cm} (6.37)
6.15.10 Positive Predictive Value (PPV)

The PPV value is used to give out the measure of number of positive results from the false positive value using the Equation (6.38). The higher value indicates lesser number of positive results from false positive.

\[
PPV = \frac{TP}{TP + FP}
\]  

(6.38)

6.15.11 Negative Predictive Value (NPV)

The NNV value is used to give out the measure of number of negative results from the false negative value using the Equation (6.39). The higher value indicates lesser number of Negative results from false negative.

\[
NPV = \frac{TN}{TN + FN}
\]  

(6.39)

6.16 SIMULATION RESULTS

The neural network simulations are carried out using the MATLAB. The HDL coding and waveform simulations are performed using the ModelSim SE Version 6.2c. The program synthesis and design implementation are carried out using the Xilinx ISE project navigator version 13.2. The Xilinx Spatran-3E XC2S250e FPGA was used for the hardware implementation.

The details of database and number of image details are shown in the Table 6.1. The UBIRIS data base is selected for training and testing the proposed algorithm. The 70% of the image is taken as the training and the remaining 30% is used for testing the algorithm. The sorted feature matrix of MPCA is used to train and test the neural network.
Table 6.1 Details of iris data set

<table>
<thead>
<tr>
<th>S.No</th>
<th>Data base</th>
<th>Number of Persons</th>
<th>Number of samples / person</th>
<th>Total Number of Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UBIRIS</td>
<td>100</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Trained images 70%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Un trained images 30%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150</td>
</tr>
</tbody>
</table>

6.16.1 Results of BFGS

The FFBPNN with the BFGS training result is shown in the Figure 6.16. The output is taken at the 54th epoch. The trained and test output lines lie on the fit line. But many number of trained and test outputs do not lie on the fit line. Therefore the BFG training algorithm is not preferred for the proposed algorithm.

Figure 6.16 FFBPNN Training and Testing using BFGS algorithm
6.16.2 Results of CGF

The FFBPNN with the CGF training result is shown in the Figure 6.17 and output is taken at the 36th epoch. The trained and test data output lines are slightly deviated from the fit line. The deviation of test output is more than the trained data output. More number of trained and test data outputs do not lie on the fit line. Therefore the CGF training algorithm is not preferred for the proposed algorithm.

Figure 6.17 FFBPNN Training and Testing using CGF algorithm

6.16.3 Results of CGP

The FFBPNN with the CGP training result is shown in the Figure 6.18. The shown output is obtained at the 21st epoch. The trained data output line almost lies on the fit line. The test data output is having more deviation from the fit line. More number of trained and test data outputs do not lie on the fit line. Therefore the CGP training algorithm is not preferred for the proposed algorithm.
6.16.4 Results of SCG

The FFBPNN with the SCG training result is shown in the Figure 6.19. The shown output is obtained at the 52\textsuperscript{nd} epoch. The trained data output line almost lies on the fit line. The test data output line is having more deviation with the fit line. Few of the trained and test data outputs do not lie on the fit line. Therefore the SCG training algorithm is not preferred for the proposed algorithm.
6.16.5 Results of RP

The FFBPNN with the RP training algorithm result is shown in the Figure 6.20. The shown output is obtained at the 74th epoch. The trained data output line almost lies on the fit line. The test data output line is having more deviation from the fit line. The trained outputs are near to the fit line. But more number of trained and test data outputs do not lie on the fit line. Therefore the RP training algorithm is not preferred for the proposed algorithm.

![Figure 6.20 FFBPNN Training and Testing using RP algorithm](image)

(a)  
(b)

6.16.6 Results of OSS

The FFBPNN with the OSS training algorithm result is shown in the Figure 6.21. The shown output is obtained at the 38th epoch. The trained and test data output lines do not lie on the fit line. Many number of trained and test data outputs are not lie on the fit line. Therefore the OSS training algorithm is not preferred for the proposed algorithm.
6.16.7 Results of GDX

The FFBPNN with the GDX training algorithm result is shown in the Figure 6.22. The shown output is obtained at the 17th epoch. The trained and test data output lines are having more deviation from the fit line. More number of trained and test outputs are not lies on the fit line. Therefore the GDX training algorithm is not preferred for the proposed algorithm.
6.16.8 Results of LM

The FFBPNN with the LM training algorithm at 7th epoch result is shown in the Figure 6.23. The trained and test data output lines almost lies on the fit line. Very few of trained and test outputs are not lies on the fit. The further iteration makes the perfect fit of outputs with the target.

![Figure 6.23 FFBPNN with LM algorithm at 7th epoch](a) ![Figure 6.23 FFBPNN with LM algorithm at 7th epoch](b)

Figure 6.23 FFBPNN with LM algorithm at 7th epoch

![Figure 6.24 FFBPNN with LM algorithm at 9th epoch](a) ![Figure 6.24 FFBPNN with LM algorithm at 9th epoch](b)

Figure 6.24 FFBPNN with LM algorithm at 9th epoch
The FFBPNN with the LM training algorithm at 9th epoch result is shown in the Figure 6.24. The trained and test data output lines are slightly deviated from the fit line. The trained outputs are almost near and lie on the fit line. Very few test outputs do not lie on the fit line.

![Figure 6.24 FFBPNN with LM algorithm at 9th epoch](image)

The FFBPNN with the LM training algorithm at 20th epoch result is shown in the Figure 6.25. The trained data outputs line perfectly lies on the fit line. The test data outputs line is having more deviation with the fit line. The trained outputs are lies on the fit line. The test data outputs almost lie on the fit line.

![Figure 6.25 FFBPNN with LM algorithm at 20th epoch](image)

The FFBPNN with the LM training algorithm at 24th epoch result is shown in the Figure 6.26. The trained data output line perfectly lies on the fit line. The test data output line deviates with the fit line. The trained data outputs lies on fit lie. The test data outputs are almost near and lie on the fit line.
Figure 6.26 FFBPNN with LM algorithm at 24\textsuperscript{th} epoch

Figure 6.27 FFBPNN with LM algorithm at 26\textsuperscript{th} epoch

The FFBPNN with the LM training algorithm at 26\textsuperscript{th} epoch result was shown in the Figure 6.27. The trained and test data output lines lies on the fit line. The trained outputs are perfectly lies on the fit line. More number of test output lies on the fit line and the countable test outputs do not lie on the fit line. It shows the network is well trained at this epoch.
The FFBPNN with the LM training algorithm at 29th and 35th epoch results are shown in the Figures 6.28 and 6.29. The trained output lines lie on the fit line. The test data output lines are having more deviation with the fit line. It shows that further iteration on the network is not suitable for training and testing the network. Therefore the LM algorithm with the 26th epoch (iteration) is preferred for the proposed method.
6.16.9 **Results of Authentication**

The FFBPNN is trained using the LM algorithm with the epoch value of 26. The trained network is having only the classified trained images mentioned in the Table 6.1. The testing is carried out using both the trained and un-trained iris features. The test data input is compared with the trained network. Based on the test input, the neural comparator gives out the outputs.

If the test input is available in the trained network data base, then the output is positive and the display output for positive result is shown in the Figure 6.30. The menu appears with the title of “IRIS MATCHED” and shows the message as “AUTHORIZED PERSON”.

![Figure 6.30 Matched iris output](image)

![Figure 6.31 Un-matched iris output](image)

If the test input is not available in the trained network data base then the output is negative and display output for negative result is shown in
the Figure 6.31. The menu appears with the title of “IRIS NOT MATCHED” and shows the message as “UN-AUTHORIZED PERSON”.

6.17 BIOMETRIC PERFORMANCE ANALYSIS

The system gives the total number of authorized person and unauthorized person. The various performance parameters are calculated and shown in the Table 6.2. The system achieves an accuracy of 98.8%. The sensitivity is 99.14%, which infers the authorization accuracy of the system. The specificity is 98.67%, which infers the Un-authorization accuracy of the system.

Table 6.2 Biometric performance parameter

<table>
<thead>
<tr>
<th>S.NO</th>
<th>PARAMETER</th>
<th>VALUE IN %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accuracy</td>
<td>98.80</td>
</tr>
<tr>
<td>2</td>
<td>Sensitivity</td>
<td>99.14</td>
</tr>
<tr>
<td>3</td>
<td>Specificity</td>
<td>98.67</td>
</tr>
<tr>
<td>4</td>
<td>Positive Predictive Value (PPV)</td>
<td>99.43</td>
</tr>
<tr>
<td>5</td>
<td>Negative Predictive Value (NPV)</td>
<td>98.01</td>
</tr>
<tr>
<td>6</td>
<td>False acceptance rate (FAR)</td>
<td>0.4</td>
</tr>
<tr>
<td>7</td>
<td>False rejection rate (FRR)</td>
<td>0.6</td>
</tr>
</tbody>
</table>

The positive predictive and negative predictive values are 99.43% and 98.01%, which are the high values. It shows that the system has a higher recognition capability. The FAR is 0.4% and FRR is 0.6% which shows the false acceptance and false rejection percentage.
Table 6.3 Comparison of proposed method

<table>
<thead>
<tr>
<th>S.NO</th>
<th>METHOD ADOPTED</th>
<th>ACCURACY (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCA</td>
<td>77.8</td>
</tr>
<tr>
<td>2</td>
<td>DWT + PCA + Empirical decomposition</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>DWT + PCA + neural network</td>
<td>93.3</td>
</tr>
<tr>
<td>4</td>
<td>DWT + ICA + Fuzzy classification</td>
<td>97</td>
</tr>
<tr>
<td>5</td>
<td>Sparse Radon Transform</td>
<td>97.5</td>
</tr>
<tr>
<td>6.</td>
<td>Proposed method (2DCDWT + MPCA + neural network)</td>
<td>98.9</td>
</tr>
</tbody>
</table>

The proposed method is compared with the various existing techniques (Naresh & Vaidehi 2011) and (anis et al 2013). The comparative analysis is shown in the Table 6.4. The proposed method is compared with the existing DWT, PCA, Empirical decomposition and fuzzy based classification. The accuracy level is comparatively higher than the existing method.

6.18 VHDL SIMULATION RESULTS

The VHDL waveform simulation result is shown in the Figure 6.32. The input ports are clock (clk), ce (chip enable), test_image_addr (testable iris feature address). The internal signal is testable_image (iris feature value). The output ports are authenticated and un-authenticated. The scalar value of MPCA is used for the hardware implementation.

The authenticated is used to represent the positive result. The un-authenticated is used to represent the negative result. The proposed algorithm used 500 memory locations. The test_image_addr is the 9 bit port used to
represent the 512 memory locations. The iris feature is represented in 22 bit format.

Figure 6.32 VHDL waveform simulation
The testable iris feature address is placed at the instant of 0 ns and the waveform is shown in the Figure 6.34. The test_image_addr value is “000001011”. The corresponding test image feature is available in the testable_image port. The testable_image value is “00000010000111111100”. The authenticated output port becomes ‘1’ and un-authenticated port value is ‘0’ which represents the positive result.

At the instant of 200 ns another testable image address is placed at the test_image_addr. The test_image_addr value is “001011110”. The testable_image is “00000100101010001110”. The authenticated output port becomes ‘0’ and the un-authenticated port value is ‘1’ which represents the negative result.

6.19 EXPERIMENTAL SETUP

The experimental setup is shown in the Figure 6.33. The developed program is downloaded on the FPGA-ISP using JTAG and the USB interface.
Table 6.4 Hardware pin out details

<table>
<thead>
<tr>
<th>S.NO</th>
<th>INPUT FEATURE</th>
<th>FPGA ADDRESS LINE</th>
<th>FPGA PHYSICAL PIN NUMBER</th>
<th>ENHANCEMENT BOARD PIN NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A₀</td>
<td>P129</td>
<td></td>
<td>S₄- Switch</td>
</tr>
<tr>
<td>2.</td>
<td>A₁</td>
<td>P78</td>
<td></td>
<td>S₅- Switch</td>
</tr>
<tr>
<td>3.</td>
<td>A₂</td>
<td>P84</td>
<td></td>
<td>S₆- Switch</td>
</tr>
<tr>
<td>4.</td>
<td>A₃</td>
<td>P89</td>
<td></td>
<td>S₇- Switch</td>
</tr>
<tr>
<td>5.</td>
<td>A₄</td>
<td>P95</td>
<td></td>
<td>S₈- Switch</td>
</tr>
<tr>
<td>6.</td>
<td>A₅</td>
<td>P101</td>
<td></td>
<td>S₉- Switch</td>
</tr>
<tr>
<td>7.</td>
<td>A₆</td>
<td>P107</td>
<td></td>
<td>S₁₀- Switch</td>
</tr>
<tr>
<td>8.</td>
<td>A₇</td>
<td>P38</td>
<td></td>
<td>S₁₁- Switch</td>
</tr>
<tr>
<td>9.</td>
<td>A₈</td>
<td>P41</td>
<td></td>
<td>S₁₂- Switch</td>
</tr>
<tr>
<td>10.</td>
<td>CE</td>
<td>P69</td>
<td></td>
<td>S₁- Switch</td>
</tr>
<tr>
<td>11.</td>
<td>CLK</td>
<td>T120</td>
<td></td>
<td>Internal clock</td>
</tr>
<tr>
<td>12.</td>
<td>Authenticated</td>
<td>P124</td>
<td></td>
<td>D₁-LED</td>
</tr>
<tr>
<td>13.</td>
<td>Un-authenticated</td>
<td>P132</td>
<td></td>
<td>D₂-LED</td>
</tr>
</tbody>
</table>

The inputs are given through the key switches and the outputs are verified using the LED’s. The input switch and the output switch details are given in the Table 6.4.
Table 6.5  Hardware setting input and output details

<table>
<thead>
<tr>
<th>S.No</th>
<th>Testable image address</th>
<th>Switch inputs</th>
<th>LED outputs</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S12,S11,S10,S9,S8,S7,S6,S5,S4</td>
<td></td>
<td>D1  D0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0010000000</td>
<td>0010000000</td>
<td>0  1</td>
<td>Authenticated</td>
</tr>
<tr>
<td>2</td>
<td>001011001</td>
<td>001011001</td>
<td>1  0</td>
<td>Unauthenticated</td>
</tr>
</tbody>
</table>

The sample of testable image address, switch position and the corresponding LED outputs are given in the Table 6.5. The first value is applied on the hardware which is shown in the Figure 6.34. The input image is trained image. Therefore the D0 LED glows and it shows the authentication.

![Figure 6.34 Hardware settings for trained image](image)

Figure 6.34 Hardware settings for trained image

The second value of Table 7.5 is applied on the hardware which is shown in the Figure 6.35. The input image is un-trained image. Therefore the D1 LED glows and it shows the un-authenticated image.
Figure 6.35 Hardware settings for untrained image

Table 6.6 Device utilization summary

<table>
<thead>
<tr>
<th>S.No</th>
<th>Description</th>
<th>Usage</th>
<th>Available resource</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Slices</td>
<td>268</td>
<td>2448</td>
<td>10.9%</td>
</tr>
<tr>
<td>2</td>
<td>Number of slice Flip Flops</td>
<td>280</td>
<td>4896</td>
<td>5.72%</td>
</tr>
<tr>
<td>3</td>
<td>Number of 4 I/P LUT</td>
<td>484</td>
<td>4896</td>
<td>9.89%</td>
</tr>
<tr>
<td>4</td>
<td>Number of I/O</td>
<td>13</td>
<td>144</td>
<td>9%</td>
</tr>
<tr>
<td>5</td>
<td>GCLK</td>
<td>1</td>
<td>24</td>
<td>4.1%</td>
</tr>
</tbody>
</table>

The device utilization summary in the hardware is shown in the Table 6.6. The device utilization shows that the implantation requires the very low resources. The hardware is further able to hold approximately 4500 iris feature. The logic blocks make the time delay of 6.591 ns. The routing time delay is 1.042 ns. The overall time required for verifying the one input is 7.561 ns.
6.20 SUMMARY

The MPCA features were trained and tested using the neural network technique. The iris features were divided into trained and test iris features. The training was carried out using the FFBPN network. The network was trained by various training algorithms such as LM, BFG, RP, SCG, CGB, CGF, OSS and GDX. The LM algorithm was preferred for training the proposed network. The LM algorithm with 26th iteration was applied to the network. The network was simulated. The number of authorization and un-authorization were computed. The biometric parameters were evaluated using ANN outputs. The system achieved the accuracy of 98.8%, FAR of 0.4% and the FRR of 0.6%. The accuracy was higher than the exiting methods. The neural network comparison process was implemented in the FPGA. The HDL coding was developed using the VHDL. The design was implemented in the Xilinx spartan-3E XC2S250e FPGA. The design occupied 10% of hardware resources. The overall time required for FPGA for verifying one person was 7.561ns.