Chapter 3

Mathematical Techniques Based Human Hidden Emotion Recognition System Model
3.0 Introduction

Chapter – 2 discussed various mathematical techniques used by the researchers for facial expression recognition. Some of them works on the 2D faces [66][54][74][111] and some works on 3D face data [70][48][53]. Many of them have tracked specific facial points which are movable during facial expression like, the region of upper eyes, nose, and especially the mouth region. A mathematical technique, Modeling and Animation of Facial Expressions Based on B-splines track these points and derive the curve like B-spline to find facial expression in 3D animation form [70]. A mathematical techniques, Hidden Markov Model [54] uses these point and measures the dens flow of them during expression and classifies the emotion on the bases of flow pattern. A technique Dynamic Model and Motion Energy [48] uses these facial points and their flow of expression and derive the motion energy pattern to recognize the emotion using Euclidean norm of distance. A technique Cauchy Naive Bayes classifier [74] has used each frame of the video of facial expression, based on some set of features computed for that time frame and classified the emotion using maximum like hood framework. Whereas the pattern recognition algorithms of Neural Network [43] are involved in the area of facial expression recognition, the mathematical models are formed in computer vision system. In which instead of specific facial points, the feature points of whole facial expressions are extracted by different mathematical techniques [17] like simple X and Y projection of facial expression images. By using different classification algorithm [53] of neural network an automatic computerized system is developed for facial expression recognition.

In this chapter we propose Mathematical Techniques Based Human Hidden Emotion Recognition System (MTBHHERS) and its process with mathematical tools and technique.

3.1 Proposed Mathematical Techniques Based Human Hidden Emotion Recognition System (MTBHHERS)

The proposed Mathematical Techniques Based Human Hidden Emotion Recognition System (MTBHHERS) shown in Fig. 3.1 contains five different levels of recognition process which are as follows.
Fig. 3.1: Mathematical Techniques Based Human Hidden Emotion Recognition System (MTBHHERS)

At the initial stage multiple expressions video is converted in to set of images of various expression and these images are entered in proposed system at Level – 1. This set are normalized at Level – 2 by applying different normalization method like, Gamma Correction (Balancing lightness and darkness) [39], Dimension reduction (RGB to Gray Scale) [39], Histogram Equalization (enhancement of contrast) [39] and Resizing Image [39]. At Level – 3 suitable feature vectors called Eigen images are extracted parallel by the methods Principle Component Analysis (PCA) [30] and independent Component Analysis (ICA) [30]. At the Level – 4 the eigen images of both PCA and ICA are classified by Neural Network algorithm especially Multilayer feed forward and Back propagation algorithm of artificial neural network [45]. At this level the images are classified according to the different emotions and it is called as training set. Here the newly entered expression image is classified on the basis of this training set for respective emotion. At Level – 5 the output (Classified Images of emotion) by neural network are combined and system matches the final output with stored image sets and will exposed the suitable emotion.

The subsequent articles of this chapter discuss each level of proposed system in details.

3.2 Video to Expression Images (Level - 1)

This article discusses the Level – 1 of proposed system with mathematical tools and techniques in which an expression video of a human is captured and it is converted into sequence of expressive images.
Facial expression or any kind of body gesture of an emotion is always expressed in continuous form so, it is necessary to capture the expression in its continuous video form. Video form captures the whole expression from its beginning to extreme intensity. The subtle changes in expression are also be captured quickly without any loss of expression data, so that the expressions in video format is most preferred. Since video is nothing but the set of images in sequential format, means the whole expression is captured in the sequential image frames. To process these expression it is necessary to derive the all expression images form its video. At this initial level all expression images are obtained from the expression video frame, which contains many frame for a single expression depends on video format. We are interested only those image frames in which the expression is varied so, we will choose only those image frames where there is a significant variation in expression and remaining frames can be skipped shown in Fig. 3.2.

![Image of Expression Video and Expression Images]

**Fig. 3.2: Video to Expression Images**

The operation can be performed by MATLAB functions;

```matlab
Video_Load = VideoReader(MyVideo); and
Image_frame = imcrop(video_Joy,[Xmin  Ymin Width Height]);
```

The whole MATLAB code is discussed in chapter – 5.

### 3.3 Image Normalization (Level -2)

This section elaborates the techniques to normalize the converted expression images in order to increase the efficiency of feature extraction and classification of emotion. To normalize the expression image the following set of mathematical techniques are adopted. (1) Gamma correction
(2) Dimension reduction - RGB to gray (3) Histogram equalization and (4) Image Resizing which are discussed in detail as follows.

### 3.3.1 Gamma Correction (on RGB Image)

Gamma correction balance darkness and lightness of an image, which improve the visibility of an image. Especially in facial feature extraction it is necessary that the prime feature of face area must be visible separately from the other part of the image.

![Gamma Curve](image)

In MATLAB gamma value varies between 0.0 and 10.0 as gamma value moves towards 0.0 the lightness will be increased, it moves towards 1.0 the darkness will be increased and while gamma value is 1.0 there will no changes in image [68].

*Gamma function:*

If X is an original value of an image then the new value of X is obtained as;

$$X_{\text{New}} = X^{\Gamma}$$  \hspace{1cm} (3.1)

Where,  \(\Gamma = \frac{\log(l_{\text{out}})}{\log(l_{\text{in}})}\), represent the slop of Gamma curve.

\(l_{\text{in}} = \text{Image input and } l_{\text{out}} = \text{Image output.}\)

Fig. 3.3 shows several gamma curves demonstrating the effect that the gamma value has on the shape of the gamma curve.

---

**Fig. 3.3: Gamma Curve**
After the implementation of gamma correction the gamma corrective image can be seen in Fig. 3.4.

![Gamma Corrected Image Preview](image.png)

**Fig. 3.4: Gamma Corrected Image Preview**

The MATLAB function for gamma correction is

```matlab
Image_Gamma = vision.GammaCorrector(Image_frame, Value)
```

### 3.3.2 Dimension Reduction: (RGB → Gray)

Basically the colored image has three dimensional array pixel value $M \times N \times 3$ of Red, Green and Blue (RGB) [39]. It is quite difficult to perform image processing technique on three dimensional matrix in MATLAB, so for convenient and smoother processing the image is converted in gray format. A gray image has two dimensional $(M \times N)$ pixel value which lies between $[0, 255]$ as per the gray value of an image pixel (See Fig. 3.5).

$I = \text{rgb2gray}(RGB)$ converts the true color image RGB (assumed to be in the RGB color space) to the grayscale image $I$.

The theory behind the `rgb2gray` MATLAB function is to convert a true color image from the RGB color space to the YIQ color space, and then take the value of the Y component as the grayscale value. The argument is that in the YIQ color space the luminance (Y) represents the grayscale information while hue (I) and saturation (Q) represents the color information.

The transformation matrix to convert a true color image from RGB to YIQ is the inverse of the following matrix $A$:

\[
A = \begin{bmatrix}
1 & 0.956 & 0.621 \\
1 & -0.272 & -0.647 \\
1 & -1.106 & 1.703
\end{bmatrix}
\]  

such that $A^{-1} = \begin{bmatrix}
0.2989 & 0.5870 & 0.1140 \\
0.5959 & -0.2744 & -0.3216 \\
0.2115 & -0.5220 & 0.3114
\end{bmatrix}$ (3.2)
Having said the explanation above, an expression (with 4 digits of precision) to convert a true color image from the RGB color space to the YIQ color space is as follows:

\[
\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix} = \begin{bmatrix}
1 & 0.956 & 0.621 \\
1 & -0.272 & -0.647 \\
1 & -1.106 & 1.703
\end{bmatrix} \times \begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\] (3.3)

Resulting from the previous expression, the grayscale value of a pixel can be calculated from the corresponding RGB value as follows:

\[I = 0.2989 \times R + 0.5870 \times G + 0.01140 \times B \] (3.4)

![Fig. 3.5: RGB→ Gray Image Conversion](image)

The MATLAB function to perform this operation is

\[\text{Iamge\_Gray} = \text{rgb2gray(Image\_Gamma)};\]

### 3.3.3 Histogram Equalization

In a feature extraction it is necessary that the gray image must has high contrast so that the gray values of each pixel become significantly differ from each other. The better variation in range \([0,255]\) makes the identification feature better. The Histogram Equalization [39] enhances the contrast of images by transforming the values in an intensity image so that the histogram of the output image is approximately flat.

The histogram of a digital image with \(L\) total possible intensity levels in the range \([0,G]\) is defined as the discrete function

\[h(r_k) = n_k\] (3.5)

where, \(r_k\) is the \(k^{th}\) intensity level in the interval \([0,G]\) and \(n_k\) is the number of pixels in the image whose intensity level is \(r_k\). The value \(G\) is 255 for image of class uint8, 65535 for images of class uint16, and 1.0 for image of class double. Keep in mind that indices in MATLAB cannot be 0, so \(r_1\)
corresponds to intensity level 0, \( r_2 \) corresponding to intensity level 1, and so on, with \( r_L \) corresponding to level \( G \). Note also \( G = L - 1 \) for images of class uint 8 and uint 16.

Often, it is useful to work with normalized histograms, obtained simply by dividing elements of \( h(r_k) \) by the total of pixel in the image, which we denote by \( n \).

\[
p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n} \tag{3.6}
\]

Assume for a moment that the intensity levels are continuous quantities normalized to the range \([0,1]\), and let \( p_r(r) \) denote the probability density function (PDF) of the intensity level in a given image, where the subscript is used for differentiating between the PDFs of input and output images. Suppose, that we perform the following transformation on the input level to obtain output intensity levels, \( s \).

\[
s = T(r) = \int_0^r p_r(w) \, dw \tag{3.7}
\]

Where, \( w \) is a dummy variable of integration. It can be shown that the probability density function of the output levels is uniform; that is,

\[
p_s(s) = \begin{cases} 1 & \text{for } 0 \leq s \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}
\]

When dealing with discrete quantities it is necessary to work with histogram and call the proceeding technique histogram equalization.

Let, \( p_r(r_j), \ j = 1, 2, ..., L, \) denote the histogram associated with the intensity levels of a given image and recall that the values in normalized histogram are approximations to the probability of occurrence of each intensity level in the image. For discrete quantities we work with summations, and the equalization transformation becomes,
\[
s_k = T(r_k) = \sum_{j=1}^{k} p_r(r_j) = \sum_{j=1}^{k} \frac{n_j}{n} \quad (3.9)
\]

For \( k = 1, 2, \ldots, L \), where \( s_k \) is the intensity value in the output image corresponding to value \( r_k \) in the input image.

Fig. 3.7: Histogram Equalized Gray Image Diagram

Fig. 3.7 shows that after the implementation of Histogram equalization on gray image the contrast is significantly improved and the image looks far better than the previous one, which definitely increases the analytical result.

Fig. 3.8: Histogram Equalized Image Preview

The MATLAB function to perform this operation is
3.3.4 Image Resizing

Video expression is a sequence of large number of expression images, and it is not about a single expression image sequence but may be sequence of multiple expression, so there is a possibility that the different expression images are extracted in different sizes, and also for image processing in MATLAB the dimensions of each images are required to be same. To make the image process smoothly it is necessary to make all the expression images same sized, as per desired but not so small not so big and in sufficient sized that never affect the quality of result [39].

![Image Resizing](image_url)

**Fig. 3.9: Resizing Expression Image**

The MATLAB function to perform this operation is [114].

```matlab
Image_Resize = imresize(Image_Histogram, [numrows numcols]);
```

3.4 Feature Extraction (Level - 3)

Feature extraction can be categorized according to whether they focus on motion or deformation of expression. Motion extraction is more important for our research work, since it has to be applied to image sequences. Features of expression images are here considered as Eigen images of expressions that represents the whole expression from beginning to end of expression or different patterns of an expression. Different mathematical techniques are available for feature extraction, in this work researcher utilizes (I) Principle Component Analysis and (II) Independent Component Analysis approaches on same expression images.

3.4.1 Principle Component Analysis (PCA)

At Level – 1 and Level – 2 researcher has captured video of expression, converted in sequence of expressive images and normalized them to find feature at this Level – 3. At this level feature vectors of normalized images are obtained using PCA called PCA-Eigen images.
The basis of the ordinary image space is composition of all single pixel vectors. However, the image space is not an optimal space for face representation and categorization. The aim of applying PCA is to build an expression image space which better describes the expression images. The basis vectors of this expression image space are called the principal components [30]. These components will be uncorrelated and will maximize the variance accounted in the original basis. It helps to derive meaningful representation of images by mapping high dimension input space into lower dimension feature space by which the classifier runs faster and consumes less memory. To design a system with low to moderate complexity the feature vector should contain the most pertinent information about the expression to be recognized.

By concatenating each row of the image by row, an expression image can be transformed to a column vector. Assume the width and height of the image are n pixels and m pixels respectively, the size of the transformed vector of this image will be \( n \times m \) by 1. Given M expression images as training data, convert these images to corresponding column image vectors \( \alpha_i \), where, \( i = 1, 2, ..., M \). Compute the mean of training data

\[
\mu = \frac{1}{M} \sum_{n=1}^{M} \alpha_n \tag{3.10}
\]

and let the normalized vectors be

\[
\sigma_i = \alpha_i - \mu. \tag{3.11}
\]

We want to seek a set of \( M \) orthonormal vectors \( \delta_n \), that best represents the distribution of the data. The \( k^{th} \) vector \( \delta_k \), is chosen such that;

\[
\lambda_k = \frac{1}{M} \sum_{n=1}^{M} (\delta_k^T \sigma_n)^2 \tag{3.12}
\]

is maximum, where \( \delta_l^T \delta_k = \begin{cases} 1 & \text{if } l=k \\ 0 & \text{otherwise} \end{cases} \).

Extend Eq. (3.12) and multiply \( \delta_k^T \) to both sides of the equation, we obtain

\[
\lambda_k \delta_k^T = \delta_k^T \frac{1}{M} \sum_{n=1}^{M} (\sigma_n \sigma_n^T) \delta_k \delta_k^T \tag{3.13}
\]

Assume the covariance matrix \( C = AA^T \), where \( A = [\sigma_1, \sigma_2, \sigma_3, ..., \sigma_M] \), we obtain

\[
\lambda_k \delta_k^T = \delta_k^T C \tag{3.14}
\]

Transpose both the side of equation (3.14), we have

\[
\lambda_k (\delta_k^T)^T = C^T (\delta_k^T)^T = C^T \delta_k \quad \text{and} \quad C^T = AA^T = C.
\]

Then we can conclude that \( \lambda_k \delta_k = C\delta_k \).
Thus the vectors $\delta_k$ and scalars $\lambda_k$ are the eigenvectors and eigenvalues, respectively, of the covariance matrix $C = AA^T$.

Because the size of covariance matrix $C$ is $nm \times nm$, it is time-consuming to determine $nm$ eigenvalues and eigenvectors. It is necessary to reduce the complexity of the computation. Define a matrix

$$C' = A^T A \quad (3.15)$$

Then, the size of $C'$ is $M \times M$ and let $V_i$ denote the eigenvectors of the matrix $C'$, we obtain $A^T AV_i = \eta_i V_i$. Multiply $A$ to both sides, we have

$$AA^T AV_i = \eta_i AV_i \quad (3.16)$$

We can observe from Eq. (3.16) that $AV_i$ is the eigenvector of $C = AA^T$ [40]. So we can now reduce the method by derive $p$ ($p < nm$) eigenvectors ($V_i$) of the matrix $C'$ and derive the $p$ eigenvectors of covariance matrix $C$ by multiplying $A$ to $V_i$.

The expression images can be represented in the way by projecting the data in the expression image space onto the expression space (also called Eigen Expression images). The dimension of the projected data in the feature space is much smaller than that in the original image space. After the Eigen expression images are obtained from the training set, transform expression image into feature space by a simple projection operation

$$x_i = \delta_i^T (\alpha - \mu), \quad i = 1, 2, 3, \ldots, p \quad (3.17)$$

where $x_i$ denotes the projection of the expression image $\alpha$ projected onto the $i^{th}$ eigen expression image component $\delta_i$, where $\mu$ is the average face image of the training set. The projections constitute a vector $X^T = [x_1, x_2, \ldots, x_p]$ called projection vector. We will treat the projection vector $X$ as the features for further training process.

Many eigen images (vectors) for input expression images are obtained by PCA, but the significantly vary images are more useful for classification of emotion and the similar eigen images may consume more memory in image processing with negligible effect on classification. So, the obtained eigen images (vectors) are arranged in the decreasing order of their respective eigen values and choose only those eigen vectors for further process that are significantly vary. The number of eigen vectors are chosen by plotting the decreased order eigen values on graph and the number of eigen values up to which curve is vary (See Fig. 3.10).
Fig. 3.10: Graph of Decrease Ordered Eigen Value Based Respective Eigen Vectors (Images)

From the Fig. 3.10 it is seen that out of 60 eigen vectors the curve is significantly vary up to 10 to 15 eigen vectors and then it becomes constant, in this case it is necessary to chose the number eigen images between 10 to 15 for effective process and result.

The feature vectors of original expression images referred as eigen images obtained by PCA can be seen in Fig. 3.11.

Fig. 3.11: Eigen Images by Principle Component Analysis
3.4.2 Independent Component Analysis (ICA)

There are number of algorithms to perform ICA. Here Researcher has chosen the info-max algorithm proposed by Bell and Sejnowski [1], which was derived from the principle of optimal information transfer in neurons with sigmoidal transfer functions [93][94]. The algorithm works as follows:

**Info-Max algorithm for ICA.**

Let $X$ be an $n$-dimensional (n-D) random vector representing a distribution of inputs in the environment. Let $W$ be an $n \times n$ invertible matrix, $U = WX$ and $Y = f(U)$ an n-D random variable representing the outputs of n-neurons. Each component of $f = (f_1, f_2, f_3 \ldots f_n)$ is an invertible squashing function, mapping real numbers into the $[0,1]$ interval. Typically, the logistic function is used.

$$f_i(u) = \frac{1}{1 + e^{-u}}$$  \hspace{1cm} (3.18)

The $U_1, U_2 \ldots U_n$ variables are linear combinations of inputs and can be interpreted as presynaptic activations of n-neurons. The $Y_1, Y_2 \ldots Y_n$ variables can be interpreted as postsynaptic activation rates and are bounded by the interval $[0, 1]$. The goal in Bell and Sejnowski’s algorithm [1] is to maximize the mutual information between the environment $X$ and the output of the neural network $Y$. This is achieved by performing gradient ascent on the entropy of the output with respect to the weight matrix $W$. The gradient update rule for the weight matrix $W$, is as follows:

$$\Delta W \propto \nabla W \ln H(Y) = (W^T)^{-1} + E(Y'X^T)$$  \hspace{1cm} (3.19)

where, $Y_i' = f_i''(U_i)/f_i'(U_i)$, the ratio between the second and first partial derivatives of the activation function, $T$ stands for transpose, $E$ for expected value, $H(Y)$ is the entropy of the random vector $Y$, and $\nabla W \ln H(Y)$ is the gradient of the entropy in matrix form, i.e., the cell in row $i$, column $j$ of this matrix is the derivative of $H(Y)$ with respect to $W_{ij}$. Computation of the matrix inverse can be avoided by employing the natural gradient [93][94], which amounts to multiplying the absolute gradient by $W^TW$, resulting in the following learning rule [2]:

$$\Delta W \propto \nabla W \ln H(Y) W^T W = (I + Y'U^TW)W$$  \hspace{1cm} (3.20)

where $I$ is the identity matrix. The logistic transfer function eq. (3.20) gives, $Y_i' = (1 - 2Y_i)$.

When there are multiple inputs and outputs, maximizing the joint entropy of the output $Y$ encourages the individual outputs to move toward statistical independence. When the form of the nonlinear transfer function $f$ is the same as the cumulative density functions of the underlying ICs (up to scaling and translation) it can be shown that maximizing the joint entropy of the outputs $Y$ in also minimizes...
the mutual information between the individual outputs in $U$ [2][52]. In practice, the logistic transfer function has been found sufficient to separate mixtures of natural signals with sparse distributions including sound sources [1].

The algorithm is speeded up by including a “Sphering” step prior to learning [30]. The row means of $X$ are subtracted from each row, and then $X$ is passed through the whitening matrix $W_z$, which is twice the inverse square root of the covariance matrix.

$$W_z = 2 \times (\text{Cov}(X))^{-1/2}$$

(3.21)

This removes the first and the second-order statistics of the data; both the mean and covariance are set to zero and the variances are equalized. When the inputs to ICA are the “sphered” data, the full transform matrix $W_f$ is the product of the sphering matrix and the matrix learned by ICA.

$$W_f = W W_z$$

(3.22)

MacKay [26] and Pearlmutter [13] showed that the ICA algorithm converges to the maximum likelihood estimate of $W^{-1}$ for the following generative model of the data:

$$X = W^{-1} S$$

(3.23)

where $S = (S_1, \ldots, S_n)'$ is a vector of independent random variables, called the sources, with cumulative distributions equal to $f_l$, in other words, using logistic activation functions corresponds to assuming logistic random sources and using the standard cumulative Gaussian distribution as activation functions corresponds to assuming Gaussian random sources. Thus, $W^{-1}$ the inverse of the weight matrix in Bell and Sejnowski’s algorithm [1], can be interpreted as the source mixing matrix and the $U = WX$ variables can be interpreted as the maximum likelihood (ML) estimates of the sources that generated the data.

**Statistically Independent Basis Images**

As described earlier, the goal in this approach is to find a set of statistically independent basis images. We organize the data matrix $X$ so that the images are in rows and the pixels are in columns, and each image has zero mean.

$$= b_1 * \begin{pmatrix} \text{u}_1 \end{pmatrix} + b_2 * \begin{pmatrix} \text{u}_2 \end{pmatrix} + \ldots + b_n * \begin{pmatrix} \text{u}_n \end{pmatrix}$$

ICA representation $= (b_1, b_2, \ldots, b_n)$

**Fig. 3.12: The Independent Basis Image Representation Consisted of the Coefficients, b for the Linear Combination of Independent Basis Images u that Comprised each Face Image**
In this approach, ICA finds a matrix $W$ such that the rows of $U = WX$ are as statistically independent as possible. The source images estimated by the rows of $U$ are then used as basis images to represent faces. Face image representations consist of the coordinates of these images with respect to the image basis defined by the rows of $U$, as shown in Fig. 3.12. These coordinates are contained in the mixing matrix $A = W^{-1}$.

The number of ICs found by the ICA algorithm corresponds to the dimensionality of the input. In order to have control over the number of ICs extracted by the algorithm, instead of performing ICA on the $n_r$ original images, we performed ICA on a set of $m$ linear combinations of those images, where $m < n_r$. Recall that the image synthesis model assumes that the images in $X$ are a linear combination of a set of unknown statistically independent sources. The image synthesis model is unaffected by replacing the original images with some other linear combination of the images.

**Adaption if Info – max algorithm for ICA.**

Adopting a method that has been applied to ICA of fMRI data [62], researcher choses a linear combinations the first $m$, PC eigenvectors of the image set. PCA on the image set in which the pixel locations are treated as observations and each face image a measure, gives the linear combination of the parameters (images) that accounts for the maximum variability in the observations (pixels). The use of PCA vectors in the input did not throw away the high-order relationships. These relationships still existed in the data but were not separated.

Let $P_m$ denote the matrix containing the first $m$, PC axes in its columns. We performed ICA on $P_m^T$, producing a matrix of m independent source images in the rows of $U$. In this implementation, the coefficients $b$ for the linear combination of basis images in $U$ that comprised the face images in $X$ was determined as follows.

The PC representation of the set of zero-mean images in $X$ based on $P_m$ is defined as $R_m^T = XP_m$. A minimum squared error approximation of $X$ is obtained by $\hat{X} = R_mP_m^T$.

The ICA algorithm produced a matrix $W_i = WW_z$, such that $W_iP_m^T = U$, where $W$ is weighted matrix.

$$P_m^T = W_i^{-1}U$$  \hspace{1cm} (3.24)

Therefore, $\hat{X} = R_mP_m^T$ becomes

$$\hat{X} = R_mW_i^{-1}U$$  \hspace{1cm} (3.25)

where $W_z$ was the sphering matrix defined in eq. (3.22).
Hence $R_m W_i^{-1}$, the rows of contained the coefficients for the linear combination of statistically independent sources $U$ that comprised $\hat{X}$, where $\hat{X}$ was a minimum squared error approximation of $X$, just as in PCA. The IC representation of the expression images based on the set of $m$ statistically independent feature images, $U$ was, therefore, given by the rows of the matrix

$$B = R_m W_i^{-1}$$

(3.26)

A representation for test images was obtained by using the PC representation based on the training images to obtain $R_{test} = X_{test} P_m$, and then computing

$$B_{test} = R_{test} W_i^{-1}$$

(3.27)

The feature vectors of original expression images called as eigen images obtained by ICA can be seen in Fig. 3.13.

![Fig. 3.13: Eigen Images by Independent Component Analysis](image)

These images can be interpreted as follows: Each row of the mixing matrix $W$ found by ICA represents a cluster of pixels that have similar behavior across images. Each row of the $U$ matrix tells us how close each pixel is to the cluster $i$, identified by ICA. Since we use a sparse independent source model, these basis images are expected to be sparse and independent. Sparseness in this case means that the basis images will have a large number of pixels close to zero and a few pixels with large positive or negative values.

3.5 Classification (Level - 4)

[Multi-layer Perceptron Neural Networks (Includes feed-forward & Back Propagation algorithm)]

At Level – 3 eigen features of normalized expression images have been obtained by PCA and ICA which are classified for emotion recognition at this level using neural network (NN) individually. At Level – 4 neural network is trained to classify PCA and ICA eigen images by, using Multi-layer Perceptron (MLP) with Back Propagation classification algorithm. In this level the NN parameters
like, Error surface, number of hidden layers, learning rate, momentum, input standardization, weight initialization, training and stopping criteria, generalization of NN is decided.

The general terminology of neural network in pattern recognition is described in following subsequent sub articles. After this discussion researcher has described whole process in context of proposed system.

**General Terminology of Neural Network**

Neural Networks are widely used in pattern recognition because of their ability to generalize and to respond well to novel patterns. The general concept is the following: “During training neurons are taught to recognize specific (training data) patterns. If a novel pattern is received (without an associated output) each neuron selects the output that corresponds to the training pattern that is least different from the input”.

According to Haykin [99], a Neural Network is a massively parallel distributed processor that has a natural prosperity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Inter-neuron connection strengths known as synaptic weights are used to store the knowledge.

The required model and parameters to train neural network are as follows

### 3.5.1 Multilayer Perceptrons (MLPs)

MLP is one of the most popular neural network models for solving pattern classification and image classification problems. Because of their ability to learn complex decision boundaries, MLPs are used in many practical computer vision applications involving classification (or supervised segmentation). Once the connection weights in a MLP have been learnt, the network can be used repeatedly for classification of new input patterns.

![Fig. 3.14: Sigmoid Function is Ideal for MLPs](image-url)
One of the main tasks of the research is to become experienced with Multilayer Perceptron Neural Networks, which are feed-forward and use the Back-propagation algorithm. From now on, when referring to MLPs we imply feed-forward networks and Back-propagation algorithm (plus full connectivity). A typical topology of a fully connected feed-forward network is shown in Fig. 3.15. Back-propagation algorithm is a variation of Delta rule. While inputs are fed to the ANN forwardly, the ‘Back’ in Back-propagation algorithm refers to the direction to which the error is transmitted.

Algorithm discussed in Table 3.1 gives the basic steps for the stochastic gradient descent version of BACKPROPAGATION algorithm [103]. Here a factor $\delta_h$ is introduced! But what are the target values for the outputs in each hidden layer? Since only target values for the output units have been provided, in that case the error term, instead of being $(t_k - o_k)$, is calculated by summing the errors $\delta_k$ for each output unit connected with the specified hidden unit ‘$h$’. As long as, we have a fully connected feed-forward network the total number of the latter ‘errors’ are the same as the number of output units (one for each output). To put it straight, each weight in

$$\delta_h = o_h(1 - o_h) \sum_k W_{kh} \delta_k$$

gives the degree to which hidden unit ‘$h$’ is responsible for the error in output ‘$k$’.

---

**Fig. 3.15: Fully Connected, Feed-Forward MLP network**

(Discussion of Fig. 3.15) A training sample, $X = (x_1, x_2, \ldots, x_i)$, is fed to input layer. Weighted connections exits between each layer, where $W_{ij}$ denotes the weight form a unit $j$ in one layer to a unit $i$ in the previous layer. We have an input layer ($i$) consisting of input nodes and an output layer ($k$) consisting of output nodes. The input nodes are connected to the output nodes via one or more hidden layers ($j$). The nodes in the network are connected together, and each of the links has a weight associated with it. The output value from a node is a weighted sum of all the input values to the node. By changing the different weights of the input values we can adjust the influence from different input nodes. For face recognition the input nodes will typically correspond to image pixel
values from the face image. The output layer will correspond to classes or individuals in the database. Each unit in the output layer can be trained to respond with +1 for a matching class and 0 for all others. In practice real outputs are not exactly +1 or 0, but vary in the range between these values. Classification is done by finding the output neuron with the maximal value. Then a threshold algorithm can be applied to reject or confirm the decision.)

The algorithm described in Table 3.1 can be converted to the standard gradient descent version of BACKPROPAGATION if the gradient becomes:

$$\delta_k = \sum_{pattern} O_{n,k} (1 - O_{n,k})(t_{n,k} - O_{n,k})$$

(3.28)

Where \( n \) is the number of one of the training patterns and \( k \) is the number of the output unit. Usually \( \delta_k \) is divided by the total number of training patterns in order to constrain the weight update to the mean of the updates caused by each training pattern.

<table>
<thead>
<tr>
<th>Table 3.1: Stochastic Gradient Descent version of BACKPROPOGATION Algorithm [103]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Initialize all network weights to small random numbers</td>
</tr>
<tr>
<td>- Until the termination condition (it will be discussed later) do:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>For each training example do:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Propagate the input forward to the network and compute the observed outputs.</td>
</tr>
<tr>
<td>Propagate the errors backward as follows:</td>
</tr>
<tr>
<td>For each network output unit ( k ) calculate its error term ( \delta_k = O_k(1 - O_k)(t_k - O_k) )</td>
</tr>
<tr>
<td>For each hidden unit calculate its error term ( \delta_h = a_h(1 - a_h) \sum_{outputs} W_{kh}\delta_k )</td>
</tr>
<tr>
<td>Finally, update each weight ( w_{ji} = w_{ji} + \Delta w_{ji} ), where ( \Delta w_{ji} = -\eta.\delta_j x_{ji} )</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(Here, ‘( ji )’ means from unit ‘( i )’ to ‘( j )’)</td>
</tr>
</tbody>
</table>

**3.5.2 Error Surface**

If we wanted to be more explanatory on the BACKPROPAGATION algorithm of Table 3.1 we should refer to the term as the gradient of the error function for the output ‘\( k \)’. The total error, which in standard gradient descent version of BACKPROPAGATION is the SSE (Sum of Squared Errors):
\[
E(\mathbf{w}) = \sum_{n \text{ pattern}} \sum_{k \text{ outputs}} (t_{n,k} - o_{n,k})^2
\]  \hspace{1cm} (3.29)

Based on the fact that this \( E \) is actually a function of the network’s weight vector, we conclude that \( E \) is actually a multidimensional (depending on the number of weights) parabola. Gradient descent starts with an arbitrary weight vector and tries to minimize \( E \) at each step. In order to go deeper in these multidimensional surface weights must be updated in the direction of the negative of the gradient \( \frac{dE(\mathbf{w})}{d(\mathbf{w})} \).

Gradient shows the direction of the steepest increase in the surface; hence the negative is computed, since we need the steepest decrease. Therefore, updating weights by \( \Delta W_{ji} = -\eta \delta_j x_{ji} \) we simply go deeper in the error surface. This is clearly straightforward when only two weights are present and the error surface looks like in following figure.

Fig. 3.16: 3D Parabola of SSE with respect to Weight 1&2

Fig. 3.16 shows both the total gradient and the partial gradients of errors in red colour with respect to weight 1 and weight 2 axis, respectively. At each step these vectors shows the direction of weights update upon the error surface. The magnitude of this update is affected both from the gradient and the factor ‘\( \eta \)’ (see Table 3.1), which is called learning rate. However, in MLPs the error parabola is multidimensional and often there are more than one minimum. In that case the learning might get stuck in a local minimum than in the desired global minimum.

### 3.5.3 Size of Hidden Layer – Under fitting, Over fitting

In most situations, there is no way to determine the best number of hidden units without training several networks and estimate the generalization error (performance on novel patterns) of each. If you have too few units, you will get high training error and high generalization error due to under
fitting and high statistical bias. On the other hand, the training error can be made as small as desired by adding more neurons, but generally each additional unit will produce less and less benefit. We should take into account the cost in processing time and storage requirements for each extra unit. Beside a relatively large number of neurons in the hidden layer can give high generalization error due to over fitting and high variance.

(i) **Under fitting:** means that the model is not flexible enough to capture the underlying process (the process we try to teach). And we say that this happens due to large bias. Bias in regression problems (function approximation) is the inability to fit the correct result in average, while in classification problems it can be observed of the fact that our model favors only some classes.

(ii) **Over fitting** means that the model is too flexible for the limited training data set we are using. In that case, ANNs adopt the idiosyncrasies of the training data and do not generalize well on novel patterns. Generally training data and test patterns usually have some large-scale similarities in their features. In the first steps of training network fit those large-scale similarities and generalizes well. As training evolves the network fit the small-scaled features (otherwise idiosyncrasies) of the training data and generalizes poorly on the test patterns.

3.5.4 **BACKPROPAGATION Parameters**

Two of the most important parameters in BACKPROPAGATION are the learning rate and the momentum term. The learning rate, $\Delta w_{ji}$ introduced in Table 3.1 as a scaling factor of the gradient of the error function. The momentum term is an extra factor, which is added to the term and makes it more or less (depending on the momentum term) dependent to the weight update of the previous step in the algorithm. Adjusting learning rate to correct values and adding a reasonable momentum term can improve a neural network classifier’s performance dramatically.

(i) **Learning Rate:**

In Table 3.1 weights are updated by

$$\Delta w_{ji} = -\eta \cdot \delta_j x_{ji}$$

(3.30)

This is called learning rate of the BACKPROPAGATION algorithm. With standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm may oscillate and become unstable. If the learning rate is too small, the algorithm will take too long to converge. It is not practical to determine the optimal setting for the learning rate
before training. This is most likely obtained by trial and error, just like the method we used to find the number of hidden neurons (see 3.5.3).

Generally it is not possible to calculate a best learning rate a priori, but typically it ranges from 0 to 1. It is also called step size, apparently because it affects the step of gradient descent algorithm towards the error surface minimum. It is common knowledge that a very small learning rate will cause very long training times; hence larger rates are usually used. However, when large learning rates are used we might come faster to a region of convergence (slide faster down the slope and approach the global minimum of the error surface) but we might “jump” and miss the global minimum. That is why it is necessary take care of what learning rate is chosen. Ideally, when adjust the learning rate “on the fly”. This means, it might want big steps in the beginning and thus use big learning rate. But as training progresses and the training system approximates convergence, gradually reduce the learning rate to zero to allow the system to settle to the minimum.

There are some factors affecting the choice of an appropriate learning rate. For example when the training set is large and representative of the pattern population, then it might be wise to use large learning rate for fast convergence. Additionally, when the error surface is complex, (and this is so when treating multidimensional data) and consists of hills, valleys, ridges etc., the gradient term \( \frac{dE(\vec{W})}{d(\vec{W})} \) changes dramatically as \( \vec{W} \) changes. This means large learning rates can be used to move along a flat area quickly, but smaller values are needed to avoid stability in “hilly” areas. Another factor is the use of a momentum term, which is discussed in detail next.

In general, at very small learning rates, training times are high simply because each weight change is so small. However, beyond a point the training time and the generalization error increase sharply. The detail discussion is given in chapter-5 with algorithm and MATLAB code.

(ii) Momentum Term:
A common modification, of the basic weight update rule is the addition of a momentum term. By adding this term to the formula of the final step in Table 3.1, we obtain the following update rule:

\[
\Delta W_{ji} = \eta \delta_j x_{ji} + \alpha \Delta W_{ji}(n-1)
\]

(3.31)

Therefore the update in \( n^{th} \) iteration is affected by the update in \( n^{th} \) iteration multiplied by a factor ‘\( \alpha \)’, called momentum. Momentum takes values in the range \( 0 \leq \alpha < 1 \). Empirical evidence shows that the use of a momentum in the BACKPROPAGATION algorithm can be helpful in speeding
the convergence and avoiding local minima in the error surface. The idea about using a momentum is to stabilize the weight change by making non-radical revisions using a combination of the gradient decreasing term with a fraction of the previous weight change. Substantially, the addition of the momentum gives the system a certain amount of inertia since the weight vector will tend to continue moving in the same direction unless opposed by the gradient term.

![Fig. 3.17: Effect of the Momentum Term in Training Procedure](image)

Large learning rates usually give rise to oscillations, when moving near the convergence region. A system with momentum stabilizes those oscillations and makes training smoother [91]. Without momentum learning takes longer to settle down to a minimum. The effect of momentum term is shown in Fig. 3.17, also the detail discussion is given in chapter-5 with algorithm and MATLAB code.

### 3.5.5 Input Standardization and Weights Initialization

Weights initialization follows nearly the same path as input standardization. The main emphasis in the NN literature on initial values has been on the avoidance of saturation, hence the desire to use small random values. Symmetry breaking in the weight space is needed in order to make neurons compute different functions. If all nodes have identical weights then they would respond identically. Therefore the gradient, which updates the weights, would be the same for each neuron. This way the weights would remain identical even after the update and this means no learning. A special case is to initialize all weights of every neuron to 0. Then in every neuron the gradient of a zero function would be zero and thus weights would remain zero until training is terminated. Small weights (as well as small inputs) are needed to avoid immediate saturation because large weights could amplify a moderate input to produce an extremely large weighted sum at the inputs of the next layer. This would put the nodes into the flat regions of their nonlinearities (see Fig. 3.18 for sigmoid saturation) and learning would be very slow because of the very small derivatives. The detail discussion is given in chapter-5 with algorithm and MATLAB code.
3.5.6 Training Stopping Criteria

When we train Multilayer Perceptron it is unlikely to know a priori when to stop. Since various “learning rate – momentum - # hidden neurons”-schemes are being tested it becomes obvious that each time we have to adapt our stopping criteria to each case if we want efficient learning. Four basic termination conditions when training an ANN:

- **Fixed number of iterations.** Iterations, also called epochs, refer to the number of times the total training set is being presented in the Neural Network (see Table 3.1).

- **Use threshold for the error.** Empirically estimate a certain value for the error, which considered being acceptable.

- **Use threshold for the error gradient.** Usually we have to restrict training to steps which error gradient is larger than a fixed value. Small changes in error gradient mean that training reached a minimum (local or global) and it would be wise to stop without delay.

- **Early stopping.** Divide the available data into training and validation sets. Commonly use a large number of hidden units and very small initial values. Compute the validation error rate periodically during training. Finally, stop training when the validation errors rate “start to go up”. However, it is important to stress that the validation error is not a good estimate of the generalization error. The most common method for getting an unbiased estimate of the generalization error is to run the ANN on a third set of data that is not used at all during the training process.
We can combine stopping criteria when constructing Neural Networks. For example it would wiser to use a good (quite small) threshold for the error function and a large number for iterations, when we train/test various model schemes with varying number of neurons in hidden layers. This way we get how many of the model schemes converge, when they converge (time in seconds), how many did not converge and the generalization errors of convergent and non-convergent networks in order to obtain the best NN structure for our expression recognition systems.

3.5.7 Generalization

Generalization is the ability of capturing the underlying function [99][103][91][45][98], during the training phase, and hence producing correct outputs in response to novel patterns (patterns that has not seen before). A system then is said to generalize well. If performance in new patterns is poor then poor is the generalization as well. From a statistic perspective the generalization error can be considered as the summation of a variance and a bias term:

\[ E_{gen} = Variance + Bias^2 \] (3.32)

Minimizing the generalization error is not equivalent to selecting a model where the bias is zero. This is because the model variance penalty may be too high. This is called the bias/variance trade-off. Variance and bias are well-understood issues when it comes to regression problems (function approximation using Neural Networks). However, in classification there is a correspondence but it is surely more complex subject. An attempt was made in (3.5.3) to give some definitions of variance and bias with respect to the issues of under fitting and over fitting.

There are a few conditions that are typically necessary – although not sufficient - for good generalization:

- In order to generalize well, a system needs to be sufficiently powerful to approximate the target function. If it is too simple to fit even the training data then generalization to new data is also likely to be poor.
- The inputs contain sufficient information pertaining to the target, so that really exists a concept (unknown and complex mathematical function) that relates inputs with correct outputs. We cannot expect a network to learn a nonexistent function or a non-existed classification rule.
- In general, the training set must be a representative subset of the theoretical population. A poor set of training data may contain misleading regularities not found in the underlying function/classifier.
Adaption of Terminology of NN in MTBHHERS

Now we use the general terminology of neural network discussed above in classification of eigen images of expressions for emotion in the proposed system MTBHHERS the whole process of NN for classification using general terminology is described as follows.

(i) Multilayer Perceptrons (MLPs) for MTBHHERS

To train NN for emotion recognition from expression images using MTBHHERS researchers has used Multilayer Perceptrons of NN includes one hidden layer as in pattern recognition it is universal approximation [91] and seven output layers as basic seven emotion Neutral, Joy, Anger, Sad, Disgust, Fear and Surprise are considered. Log-sigmoid transfer function is used at both the layers hidden as well as output layer to restrict the output in [0, 1]. Since researcher has decided the outputs range of 0.1 to 0.9 (0.9 for confident, 0.1 for not confident) log-sigmoid is the ideal for system.

(ii) Weight initialization for MTBHHERS

In the training of NN a problem is how to initialize weights at hidden and output layer? Researcher had three ways for that (1) hidden-zero, output-zero, then no update occurs, which means that weights remain zero and performance constant so it has not been chosen (2) hidden – random, output-random, then the output of expression images are noisy and the expression can’t be visible can be seen in Fig. 3.19 (3) hidden – zero, output – random, then the output as clear appearance of expression images can be seen in Fig. 3.20.

Fig. 3.19 Hidden – random, Output - random

Fig. 3.20 Hidden – zero, Output - random
Thus, in the training of NN the third option that hidden layer is initialized with zero weight and output layer initialized with random weight is chosen.

(iii) Error in Classification for MTBHHERS

In the classification of emotion the two processes are performed by NN in MTBHHERS on data set, training and testing. During these procedure NN finds the percentage of accuracy for classification of emotion from eigen images in training and testing process and finds the related error in both the process in the term of Mean Square Error (MSE). The MSE shows the mean of squared error in the different between output and target, smaller MSE means the output of the system is closer to the target function to be minimized.

(iv) Number of neurons in hidden layer (size of hidden layer) for MTBHHERS

To choose optimum number of neurons in hidden layer prior training or testing is still an issue of research, it is totally based on experiment and on the choice of other parameters of NN. In the implementation of system MTBHHERS for emotion recognition from expressions of a person researcher has done many experiment and on the trial and error base he has chosen 16 neurons in hidden layer, at which the accuracy of classification of emotion is higher and the test MSE is lower, it will be discussed with details in chapter – 5.

(v) Learning rate – Back propagation parameter for MTBHHERS

In the training of NN for classification of emotion from facial and body expression the value of learning rate decides how quick output converges to target. Experiment shows that the larger value of learning rate increase MSE in classification [Fig 5.14], while at low learning rate NN training process long time as because update in weight too small. Thus the moderate learning rate between 0 to 1 is most preferable, here in emotion recognition from various expression researcher has chosen 0.9 learning rate.

(vi) Momentum – Back propagation parameter for MTBHHERS

Momentum makes the tanning process smoothly by speeding the convergence and avoiding local minima in the error surface (See Fig. 3.17) but it only works when learning rate is quite small. The very small and very large value of momentum increase MSE in training [Fig 5.14], thus by experimental results researcher has chosen 0.6 a moderate value of momentum for NN classification.
(vii) **Input Standardization for MTBHHERS**

The contribution of an input will depend heavily on its variability relative to other inputs. If for example one of the inputs has range of 0 to 1 and another has a range of 0 to 1000, then the contribution of the first input will be swamped by the second input. So it is essential to rescale the inputs such that their variability reflects their importance. Here, in the training of NN for MTBHHERS researcher is using PCA or ICA feature vector as input, here PCA is centralized with zero mean and 1 standard deviation before feeding in NN while ICA is already rescaled in sphering the image data. So both the inputs are rescaled before trained in NN.

Considering all above parameters to train neural network the optimum topology of NN for proposed system MTBHHERS can be obtained as Table 3.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Standardization</td>
<td>PCA / ICA</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>Hidden Layer = 0 , Output Layer = random</td>
</tr>
<tr>
<td>Training Algorithm</td>
<td>Gradient Descent with Momentum</td>
</tr>
<tr>
<td>Transfer Function</td>
<td>Both layer use log-sigmoid</td>
</tr>
<tr>
<td>Neurons in Hidden Layer</td>
<td>16 neurons</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.9</td>
</tr>
<tr>
<td>Momentum term</td>
<td>0.6</td>
</tr>
</tbody>
</table>

### 3.6 Fusion of PCA-NN and ICA-NN

After Level – 4 the PCA-NN and ICA-NN classified expression images are fused in this Level – 5 using score based strategy discussed as under.

If the score functions are directly comparable or if there exists at least an acceptable transformation scheme to make the involved classifiers comparable, score based strategies are good ways for decision process. In this work, NN is used as a classifier for both systems (PCA & ICA), so naturally outputs of both systems are in same format, so we select score based strategy (SBS) as combiner [18]. The algorithm in Table 3.3.
Table 3.3: Algorithm of Score Based Strategy (SBS)

<table>
<thead>
<tr>
<th>Step 1: Assemble Label, PCA-NN and ICA-NN Score(S) of emotion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2: Set threshold values (TV) for PCA-NN and ICA-NN</td>
</tr>
<tr>
<td>Step 3: If PCA-NN(S) &gt; PCA-NN(TV) and ICA-NN(S) &gt; ICA-NN(TV)</td>
</tr>
<tr>
<td>If PCA-NN(S) &gt; ICA-NN(S)</td>
</tr>
<tr>
<td>PCA-NN(S)</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>ICA-NN(S)</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>Else Go to Step 4</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>Step 4: If PCA-NN(S) &gt; PCA-NN(TV) and ICA-NN(S) &lt; ICA-NN(TV)</td>
</tr>
<tr>
<td>PCA-NN(S)</td>
</tr>
<tr>
<td>Else if PCA-NN(S) &lt; PCA-NN(TV) and ICA-NN(S) &gt; ICA-NN(TV)</td>
</tr>
<tr>
<td>ICA-NN(S)</td>
</tr>
<tr>
<td>Else PCA-NN(S) &lt; PCA-NN(TV) and ICA-NN(S) &lt; ICA-NN(TV)</td>
</tr>
<tr>
<td>Access Denied</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

[Where, TV = Threshold Value, S = Score]

Neural Network gives the individual PCA and ICA accuracy score of classification of an emotion in percentage which are used as PCA-NN and ICA-NN scores at Step – 1 of algorithm. The threshold value is consider as expected accuracy we want from the NN classification as per our expectation that can be set manually, for both PCA-NN and ICA-NN classifier at Step – 2 which can be same or differ. In Step – 3 conditional approach is used for both PCA-NN and ICA-NN, if both classification scores crossed their limit of threshold value then the higher classified score is accepted otherwise in failure situation it indicates not to reach the expected level of accuracy. In winning situation if one of the classifier crosses the threshold limit at Step – 4 then its accuracy score will be considered as
final matching score. The detail discussion of SBS algorithm and its MATLAB code is given in chapter – 5.

3.7 Emotion Recognition (Level-5)

After fusion of PCA-NN and ICA-NN, to identify an emotion system needs the training set of an individual emotion of individual person and the mapping between the expression and related emotion. The detail discussion about mapping of expression to emotion is discussed in chapter-4 while the database of training set is sown in chapter-5.

![Diagram showing emotion identification process](image)

**Fig. 3.21: Emotion Identification**

At this level-5 we consider expression images according to the emotion class \( E_1, E_2, E_3, \ldots, E_k \) of an individual classified by NN with same feature extraction techniques (PCA & ICA) is called as “Training Set” which have been already stored in our database. The more customized expression
images makes the database stronger for highly accurate classification, so the update in training set is much essential for emotion recognition.

When the new expression image or sequence of images are entered as test image, at that time proposed system (MTBHHERS) performs all processes of Levels 1 to 4 on it and obtained an output(emotion) which is matched with all the stored emotion class($E_1, E_2, E_3, \ldots, E_k$) of an individual. At this level system finds the association between output emotion and stored classes of categorized emotions then identifies the hidden emotion which are useful to identify human hidden psychology.

The proposed system is applied to identify human psychology through facial expression and body language and fusion of both. The required database, algorithm and MATLAB code are discussed in subsequent chapters – 5, 6, and 7.

### 3.8 Conclusion
In this chapter we have proposed Mathematical Techniques Based Human Hidden Emotion Recognition System model which will be useful to recognize human hidden emotion and psychology.