FACE DETECTION IN IMAGES USING COLOR AND APPEARANCE BASED TECHNIQUES

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

(With Specialization in Control and Guidance)

By

KIRAN BHASKAR

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2005

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in this dissertation entitled, FACE DETECTION IN IMAGES USING COLOR AND APPEARANCE BASED TECHNIQUES, in partial fulfillment of the requirement for the award of the degree of Master of Technology in Electronics and Communication Engineering with specialization in Control and Guidance, submitted in Electronics and Computer Engineering Department, Indian Institute of Technology, Roorkee, is an authentic record of my own work carried out under the supervision of Dr. M. J. Nigam, Associate Professor, Electronics & Computer Engineering Department, Indian Institute of Technology, Roorkee.

I have not submitted the matter embodied in this dissertation for the award of any other degree.

Date: 31/05/05

(KIRAN BHASKAR)

This is to certify that the above statement made by the student is correct to the best of my knowledge.

Dated : 31-5-05

(Dr. M. J. Nigam)
Associate Professor
E&CE Department
IIT Roorkee
Roorkee-247 667

ACKNOWLEDGEMENT

With great pleasure, I avail this opportunity to express my deep sense of gratitude and indebtedness Dr. M. J. Nigam, Associate Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee for his spirited guidance and inspiration in completing this dissertation.

Next, I feel indebted to all those endless researchers all over the world whose work I have used in my dissertation. Their sincerity and devotion motivates me the most.

I would also like to thank all my friends for their encouragement and help.

The blessings of my Parents and other family members contributed significantly in keeping me motivated throughout this work.

KIRAN BHASKAR
M.Tech. 2nd Yr.
Control & Guidance
E & C Deptt.

ABSTRACT

The face detection problem is to identify the presence of a human face in an image. The problem has important applications to automated security systems, lip readers, indexing and retrieval of video images, videoconferencing with improved visual sensation, and artificial intelligence.

In this dissertation, Color based Neural Network and Appearance based techniques are examined and implemented for detecting frontal view human faces in color and gray scale images respectively.

In Neural Network based systems for detecting human faces in color images two approaches are used that vary in type of input i.e. RGB and YES color space; fed into the network. It is a color-based technique combined with unsupervised learning, or training, used to set the weights of the network. The idea for the network is to learn a chroma chart from a training set. Each system is trained on the same image set using the Levenberg-Marquad method. Same training images and test images are used to compare the results obtained from two different color spaces as input.

In appearance-based technique, face detection problem is divided into two parts; the first one feature extraction and other as classification. For feature extraction, EM algorithm of PCA (principle component analysis) is used for training and K-NN (K nearest neighbor) method is used for classification purpose where K is typically taken as 1. This is the advanced pattern classification based technique but it is typically used only for gray scale images with better performance.

These two techniques are implemented in MATLAB Version 6.5 and results are presented in chapter 4. In NN based technique 20 training images were taken and they take 1 minute for training. For classification based technique 4000 training images for face and 4000 nonface training images were taken and they take only 2-3 seconds.

Comparison of these two techniques was carried out on the basis of complexity, computational cost, training time, application areas, dependence on color space etc. The study finally reveals that appearance based technique is superior for most of the applications.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

PCA	Principal Component Analysis
SVM	Support Vector Machine
K-NN	K Nearest Neighbor
EM	External Maximization
LM	Levenberg-Marquad
LDA	Linear Discriminant Analysis
ICA	Independent Component Analysis
HMM	Hidden Markov Model
FE	Feature extraction
CL	Classification

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The purpose of this dissertation is to detect faces in various images. There are many different applications in which a face detection program could be used. The querying of image databases is one possible application that would use face detection. For example, if someone wanted to search a database that consisted of many images, the user could simply instruct the program to find images with people, which would be determined by detecting faces of people in each image. Also, face detection is the first step in the process of face recognition. Many surveillance companies could make use of programs that can reliably scan a surveillance photo, and recognize certain individuals. In order to recognize a person in an image, it is first necessary to find the face of each person in that image. This type of program is especially useful in places such as airports to find criminals.

Content-based methods try to identify features in a human face. Most Content-based methods were developed for grayscale images to avoid the complexity of combining the features detected in the RGB color space. Yow and Cipolla developed a method that elongates the image in the horizontal direction and identifies thin horizontal features, such as the eyes and mouth, using second-derivative Gaussian filters [1]. A technique developed by Cootes and Taylor matches features to a model face using statistical methods [2]. Leung, Burl, and Perona presented a similar method that matches features to a model face, except they used a graph-matching algorithm to compare detected features to the model [3]. Rowley, Bluja, and Kanade developed a front-view face detection system that uses neural networks to pick out features [4]. A new content-based technique considered face detection problem as classification problem and considered face detection problem as Feature extraction and classification problem [15], [16]. Various feature extraction schemes are there like PCA [13], LDA [17], ICA[18] etc and various classification schemes exist like K-NN, SVM, HMM etc. combination of any two can be used for Face detection.

Color-based techniques [8] calculate histograms of the color values and then develop a chroma chart to identify the probability that a particular range of pixel values represent

human flesh. It has been found that the effectiveness of the method depends highly on the color space used. Chroma charts have been developed for the standard RGB color space, the YIQ color space, and the LUV space.

1.1 Statement of Problem

A survey of content-based techniques for general image retrieval can be found in [9]. Unfortunately, content-based techniques are very complex and expensive computationally. Also, if the face is rotated or partially obscured, the technique has to incorporate other techniques to solve the image registration and occlusion problems. In addition, it is often difficult to adapt the methods to color images.

The implementation of color-based techniques is fairly simple and, after the system has learned a chroma chart, the processing is very efficient. Also, the methods handle color images in a more straightforward manner than the content-based methods. However, as [7] describes, color-based techniques have several drawbacks. These disadvantages include information loss due to quantization, the strong dependence on the color space, and "erroneous retrieval in the presence of gamma nonlinearity." The most significant drawback, however, is that a technique based solely on a color histogram ignores all spatial information in the image. That is, color histograms catalog the global distribution of colors, but do not tell how the colors are arranged to form shapes and features. Despite these disadvantages, color histograms are very popular due to their simplicity and ease of calculation.

In this dissertation, method is to solve the face detection problem in a manner akin to how the human brain learns classification. Systems "learn" how to differentiate human faces from non-faces[8]. A neural network based system is presented. Color histogram in the RGB and YES color space were constructed. The idea was for the network to learn a chroma chart from a training set.

Despite of complexity in classification-based technique here a simple PCA technique [13] for feature extraction and 1-NN classification technique and training algorithm is presented for face detection to see the performance of Appearance based technique. Appearance based technique is very efficient learning algorithm stated face detection problem as classification procedure given by two examples: faces and non-faces. Classification process determines whether the given pattern is face or not [5]. To identify faces first EM algorithm of PCA is

used for feature extraction and then 1-NN classifier was trained with pictures of face and not face and for this empca toolbox is used [13]. This technique is limited up to gray scale images due to increase in complexity but it has number of advantages it can handle faces over a wide range of scales and works under different lightning conditions, even with moderately strong shadows.

1.2 Organization of Dissertation

The work presented in this dissertation has been organized into seven chapters

Chapter 1 provides an overview of the face detection techniques and formulation of the problem.

Chapter 2 provides the fundamental concepts and theory of neural network. Its application in face detection problem.

Chapter 3 provides fundamental theory of PCA technique and I-NN classifier for face detection.

Chapter 4 provides a detailed description of methodology and implementation

Chapter 5 presents results obtained from two techniques and their interpretation, and comparative performance evaluations of the two system.

Chapter 6 finally conclusion and future scope has been reported.

CHAPTER 2

FACE DETECTION TECHNIQUES

2.1 Color Based Technique Using Neural Network

Artificial neural systems, or neural networks, are massively parallel distributed processors made up of simple processing units, which can acquire, store, and utilize experiential knowledge [9]. Knowledge is acquired from its environment through a learning process. This knowledge is stored in the form of stable states or mappings, embedded in networks in terms of synaptic weights. Artificial neural systems are good at tasks such as pattern matching and classification, function approximation, optimization, vector quantization and data clustering [10].

The block diagram in figure 2.1 shows the model of a neuron, which forms the basis for designing neural networks. In mathematical terms, a neuron can describe in the following pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1a}$$

$$y_k = \varphi(u_k + b_k) \tag{2.1b}$$

The activation function may be a threshold function, Piecewise-linear function, or sigmoid function [9].

Models of the neural networks are specified by three entities: models of the neurons themselves, models of synaptic interconnections and structures (architectures), and training (learning) rules for updating the connecting weights. Basically, there are three types of

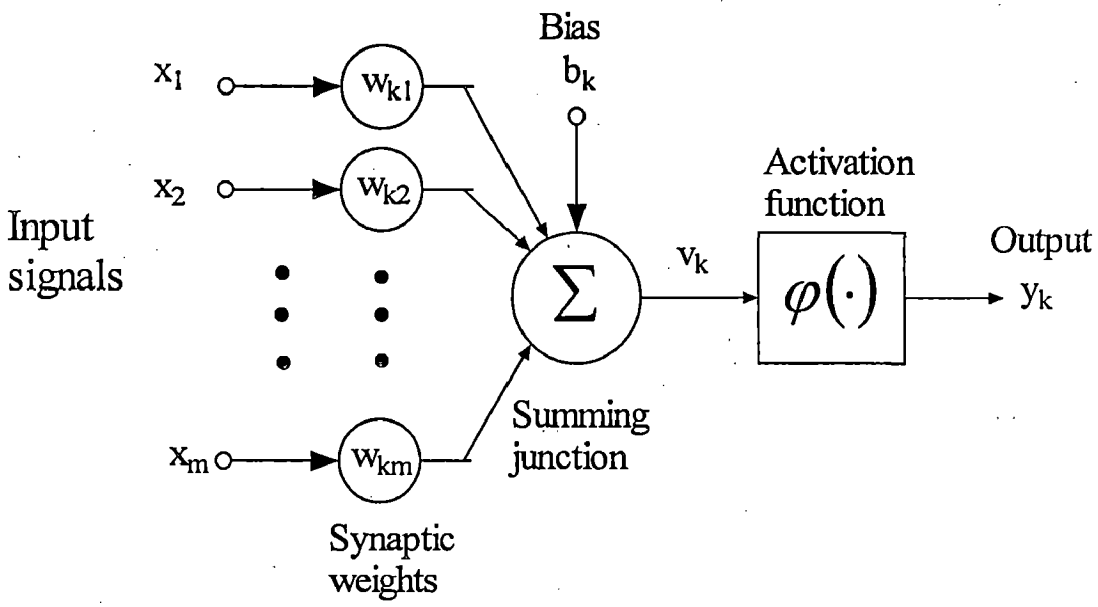


Figure 2.1: Model of a Neuron

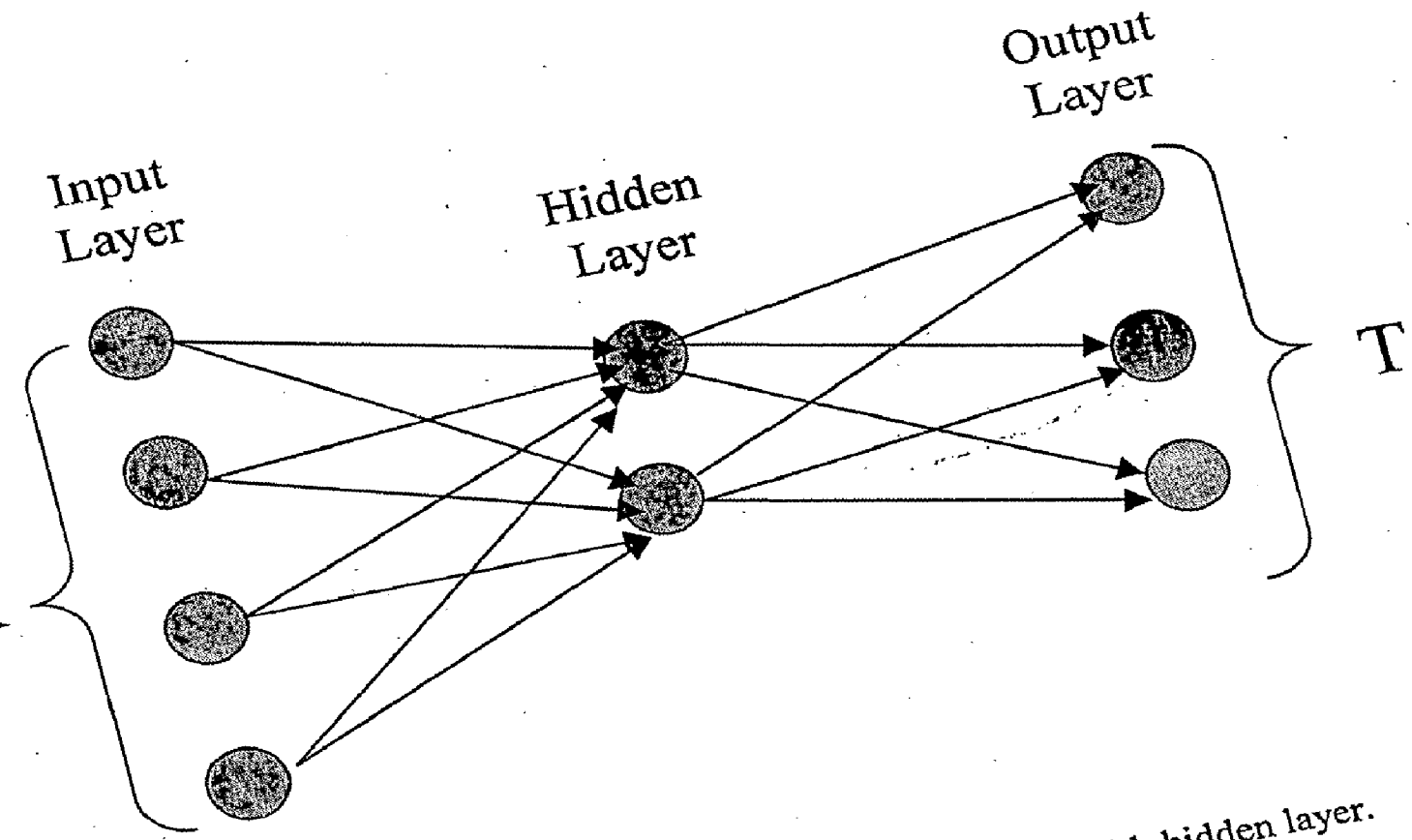


Fig. 2.2. Diagram of a generalized neural network with hidden layer.

Each vertex of the network can be thought of as a neuron, while each edge is a nerve fiber. An input vector X as shown in Fig. 2.2 is fed into a set of nodes. An input at a specific node is passed along a weighted edge, multiplied by the weight, to a neuron in the so-called hidden layer. If the passed value exceeds a threshold function, the information is then passed along another weighted edge to an output node, where it must also pass a threshold function. The sum of all such information from all input nodes gives the output vector T. Note that this network can be generalized to any number of hidden layers.

A training data set consists of a set of input vectors X with corresponding output vectors T. The actual training consists of setting the weights so that, for each input X, the output vector T_{NET} computed by the network closely matches the desired output T_{ACTUAL}. Phrased as optimization problem, we wish to find the collection of weights that minimizes $\|T_{NET} - T_{ACTUAL}\|$, where the norm is understood to be taken over all input-output pairs in the training set.

If the training set is chosen carefully to represent the entire space of possible inputs, then an input similar to one in the training set should result in a similar output. Note that training is very expensive computationally, since the determination of the weights must be done, in some sense, simultaneously for all data in the training set. However, after training is complete, the computation of an output T for a given input X is very efficient.

2.1.1 Neural Network for Face Detection

For our face detection problem, our input vector X will consist of information derived from a color image [8]. The output vector T will be a single number (node) that represents the probability that the image contains a human face. That is, if we let pattern ω be a human face and observation x a color image, then we are trying to determine $P(\omega|x)$, or P for simplicity. We should note that the interpretation of P as a probability may not hold for actual network output, since there is no guarantee that every input will give rise to an output P such that $0 \leq P \leq 1$. So we interpret the output P for a given input image X as:

$$P\begin{cases} > 0.5 \Rightarrow \text{X contains a human face} \\ < 0.5 \Rightarrow \text{X does not contain a face} \\ = 0.5 \Rightarrow \text{unclear if X contains a face} \end{cases} \qquad 2.1.1(a)$$

We chose a network with one hidden layer consisting of 20 nodes. So the number of weights to set is

$$20(|X|+1) \qquad 2.1.1(b)$$

We chose the sigmoid function $\sigma(t)$ as our threshold function:

$$\sigma(t) = \frac{1}{1+e^{-t}} \qquad 2.1.1(c)$$

The training algorithm used was the Levenberg-Marquad algorithm, which essentially minimizes the error $\|T_{NET} - T_{ACTUAL}\|$ by multi-dimensional steepest descent [14].

2.2 Appearance Based Technique for Gray Scale Images

This is a classification-based technique used for efficient face detection in gray scale images [13]. Basically, face detection algorithms consist of at least two parts: feature extraction & classification. The purpose of feature extraction is preprocessing the image data to get better representation of data, as to facilitate better classification results. Also, many classification schemes exist; each has its peculiar strength & weakness. Various Feature extraction (FE) methods are,

- Principle Component Analysis (PCA), a frenquently used statistical technique for optimal lossy compression of data under least squre sense, provide orthogonal basis vector-space to represent original data.
- Linear Discriminant Analysis (LDA), which maximize the (between-class variance)/(within-class variance)
- Independant Component Analysis (ICA), an emerging method which provide independant (but not-necessarily orthogonal) sourses to represent original data.

- Other feature extraction methods, i.e., No-negative matrix factorization(NMF), Locally Linear Embedding method (LLE), etc.

Some Classification methods are,

- K Nearest Neighbor (K-NN) method [20], and typically 1-NN while K=1
- Support Vector Machine (SVM), a novel method for classification while minimizing the "structure risk".
- Neural Network methods.
- Statistical Clustering + likelihood score, as one kind of statistical parametric classification method
- Other statistical methods, i.e., Hidden Markov Model (HMM) or Mixture Factor Analyzers (MFA), etc.

Basically, any combination of FE+Cl can be used: for example, PCA + LDA + K-NN. For easy understanding and implementation combination of PCA +K-NN method is used for face detection.

2.2.1 EM algorithm of PCA for feature extraction

The algorithm allows a few eigenvectors and eigenvalues to be extracted from large collections of high dimensional data. It is computationally very efficient in space and time. It also naturally accommodates missing information. Results on synthetic and real data showing that these EM algorithm correctly and efficiently find the leading eigenvectors of the covariance of datasets in a few iterations using up to hundreds of thousands of data points in thousands of dimensions [13].

2.2.2 Applications and Advantages

Principal component analysis (PCA) is a widely used dimensionality reduction technique in data analysis. Its popularity comes from three important properties. First, it is the optimal (in terms of mean squared error) linear scheme for compressing a set of high dimensional vectors into a set of lower dimensional vectors and then reconstructing. Second, the model

parameters can be computed directly from the data – for example by diagonalizing the sample covariance. Third, compression and decompression are easy operations to perform given the model parameters – they require only matrix multiplications. Despite these attractive features however, PCA models have several shortcomings. One is that naive methods for finding the principal component directions have trouble with high dimensional data or large numbers of datapoints. Consider attempting to diagonalize the sample covariance matrix of π n vectors in a space of p p dimensions when π n and p p are several hundred or several thousand. Difficulties can arise both in the form of computational complexity and also data scarcity. Even computing the sample covariance itself is very costly, requiring $O(\pi p^2)$ operations. In general it is best to avoid altogether computing the sample covariance explicitly. Methods such as the snap-shot algorithm [21] do this by assuming that the eigenvectors being searched for are linear combinations of the data points; their complexity is $O((\pi p^2)$. In this Thesis, presented a version of the expectation-maximization (EM) algorithm [22] for learning the principal components of a dataset. The algorithm does not require computing the sample covariance and has a complexity limited by O(knp) operations where k is the number of leading eigenvectors to be learned.

Another shortcoming of standard approaches to PCA is that it is not obvious how to deal properly with missing data. Most of the methods discussed above cannot accommodate missing values and so incomplete points must either be discarded or completed using a variety of ad-hoc interpolation methods. On the other hand, the EM algorithm for PCA enjoys all the benefits [23] of other EM algorithms in terms of estimating the maximum likelihood values for missing information directly at each iteration. Finally, the PCA model itself suffers from a critical flaw that is independent of the technique used to compute its parameters: it does not define a proper probability model in the space of inputs. This is because the density is not normalized within the principal subspace. In other words, if we perform PCA on some data and then ask how well the model fits new data, the only criterion used is the squared distance of the new data from their projections into the principal subspace. A data point far away from the training data but nonetheless near the principal subspace will be assigned a high "pseudo-likelihood" or low error. Similarly, it is not possible to generate "fantasy" data from a PCA model.

In summary, the methods developed in this paper provide three advantages.

- They allow simple and efficient computation of a few eigenvectors and eigenvalues when working with many data points in high dimensions. They permit this computation even in the presence of missing data.
- On a real vision problem with missing information, the 10 leading eigenvectors and eigenvalues of 21^{17} points in 2^{12} dimensions in a few hours using MATLAB on a modest workstation.
- Through a small variation, this method allow the computation not only of the principal subspace but of a complete Gaussian probabilistic model which allows one to generate data and compute true likelihood's.

2.2.3 Theory of External Maximization

Principal component analysis can be viewed as a limiting case of a particular class of linear-Gaussian models. The goal of such models is to capture the covariance structure of an observed p dimensional variable γ using fewer than the p(p+1)/2 free parameters required in a full covariance matrix. Linear-Gaussian models do this by assuming that γ as produced as a linear transformation of some k dimensional latent variable x plus additive Gaussian noise. Denoting the transformation by the pxk matrix C, and the (p dimensional) noise by v (with covariance matrix R) the generative model can be written as

$$\gamma = Cx + v \qquad x \sim N(0,I) \qquad v \sim N(0,R) \qquad 2.2.3(a)$$

The latent or cause variables x are assumed to be independent and identically distributed according to a unit variance spherical Gaussian. Since v are also independent and normal distributed (and assumed independent of x), the model reduces to a single Gaussian model for y which we can write explicitly:

$$\gamma \sim N(0,CC^T + R) \qquad 2.2.3(b)$$

In order to save parameters over the direct covariance representation in p-space, it is necessary to choose $k < p$ and also to restrict the covariance structure of the Gaussian noise v by constraining the matrix R.3 For example, if the shape of the noise distribution is restricted to be axis aligned (its covariance matrix is diagonal) the model is known as factor analysis.

2.2.4 Inference and Learning

There are two central problems of interest when working with the linear-Gaussian models described above. The first problem is that of state inference or compression which asks: given fixed model parameters C and R, what can be said about the unknown hidden states x given some observations y? Since the datapoints are independent, we are interested in the posterior probability P (x|y) over a single hidden state given the corresponding single observation. This can be easily computed by linear matrix projection and the resulting density is itself Gaussian:

$$P(x/y) = \frac{P(y/x)P(y)}{P(x)} = \frac{N(C_x, R)\big|_y N(0, I)\big|_x}{N(0, CC^T + R)\big|_y} \qquad 2.2.4(a)$$

$$P(x|y) = N(\beta y, I - \beta C)\big|_x, \qquad \beta = C^T(CC^T + R)^{-1} \qquad 2.2.4(b)$$

from which we obtain not only the expected value βy of the unknown state but also an estimate of the uncertainty in this value in the form of the covariance I -βC. Computing y from x (reconstruction) is also straightforward: P (yjx) = N (Cx;R) jy. Finally, computing the likelihood of any data point y is merely an evaluation under (2.2.4(a)). The second problem is that of learning, or parameters fitting which consists of identifying the matrices C and R that make the model assign the highest likelihood to the observed data. There are a family of EM algorithms to do this for the various cases of restrictions to R but all follow a similar structure: they use the inference formula (2.2.4(b)) above in the e-step to estimate the unknown state and then choose C and the restricted R in the m-step so as to maximize the expected joint likelihood of the estimated x and the observed y.

2.3 EM Algorithm

The key observation of this note is that even though the principal components can be computed explicitly, there is still an EM algorithm for learning them. The algorithm is:

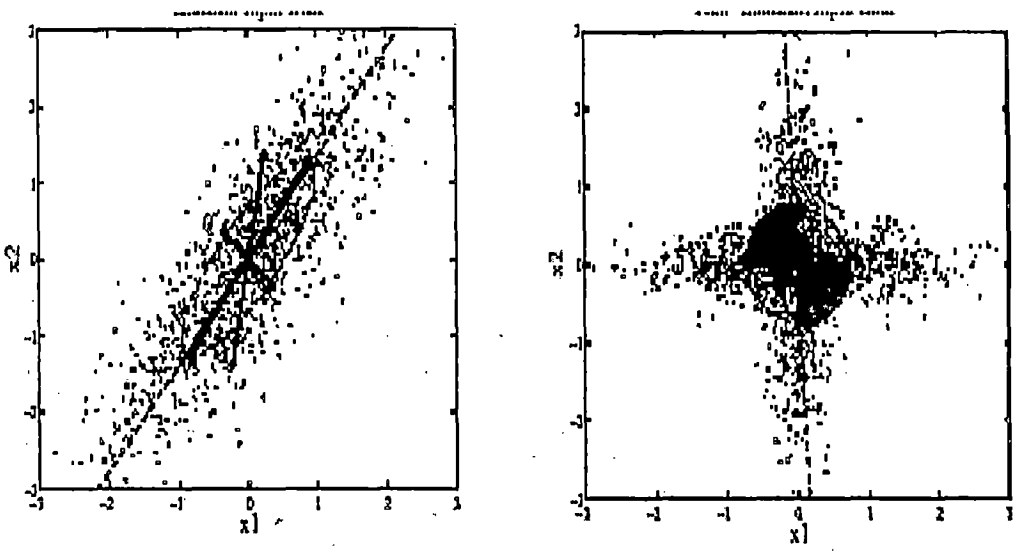
- **e-step :** $\mathbf{X} = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{Y}$

- **m-step:** $C = YX^T(XX^T)^{-1}$

whereY is a pxn matrix of all the observed data and X is a kxn matrix of the unknown states. The columns of C will span the space of the first k principal components. (To compute the corresponding eigenvectors and eigenvalues explicitly, the data can be projected into this k-dimensional subspace and an ordered orthogonal basis for the covariance in the subspace can be constructed.) Notice that the algorithm can be performed online using only a single datapoint at a time and so its storage requirements are only $O(kp) + O(k^2)$. The workings of the algorithm are illustrated graphically in figure 2.3.

The left panel shows the learning of the first principal component of data drawn from a Gaussian distribution, while the right panel shows learning on data from a non-Gaussian distribution. The dashed lines indicate the direction of the leading eigenvector of the sample covariance. The dashed ellipse is the one standard deviation contour of the sample covariance. The solid lines whose directions indicate the guess of the eigenvector and whose lengths indicate the guess of the eigenvalue at each iteration indicate the progress of the algorithm. The iterations are numbered; number 0 is the initial condition.

The intuition behind the algorithm is as follows: guess an orientation for the principal subspace. Fix the guessed subspace and project the data y into it to give the values of the hidden states x. Now fix the values of the hidden states and chooses the subspace orientation, which minimizes the squared reconstruction errors of the data points. For the simple two-dimensional example above, I can give a physical analogy. Imagine that we have a rod pinned at the origin, which is free to rotate. Pick an orientation for the rod. Holding the rod still, project every data point onto the rod, and attach each projected point to its original point with a spring. Now release the rod. Repeat. The direction of the rod represents our guess of the principal component of the dataset. The energy stored in the springs is the reconstruction error we are trying to minimize.



(a) (b)

Gaussian input data Non Gaussian input data

Figure 2.3 : Examples of iterations of the algorithm.

2.3.1 Convergence and Complexity

The EM learning algorithm for PCA amounts to an iterative procedure for finding the subspace spanned by the k leading eigenvectors without explicit computation of the sample covariance. It is attractive for small k because its complexity is limited by O(knp) per iteration and so depends only linearly on both the dimensionality of the data and the number of points. Methods that explicitly compute the sample covariance matrix have complexities limited by $O(np^2)$, while methods like the snap-shot method that form linear combinations of the data must compute and diagonalize a matrix of all possible inner products between points and thus are limited by O(n2p) complexity. The complexity scaling of the algorithm compared to these methods is shown in figure 2.4 below. For each dimensionality, a random covariance matrix _ was generated5 and then 10p points were drawn from N (0;Σ). The number of floating point operations required to find the first principal component was recorded using MATLAB's flops function. As expected, the EM algorithm scales more favorably in cases where k is small and both p and n are large. If $k \approx p \approx n$ (we want all the eigenvectors) then all methods are $O(p^3)$. The standard convergence proofs for EM [1] apply to this algorithm as well, so we can be sure that it will always reach a local maximum of likelihood. Furthermore, Tipping and Bishop have shown [8, 9] that the only stable local extremum is the global maximum at which the true principal subspace is found; so it converges to the correct result. Another possible concern is that the number of iterations required for convergence may scale with p or n. To investigate this question, the leading eigenvector for synthetic datasets (as above, with n = 10p) of varying dimension is explicitly computed and recorded the number of iterations of the EM algorithm required for the inner product of the eigendirection with the current guess of the algorithm to be 0:999 or greater. Up to 450 dimensions (4500 datapoints), the number of iterations remains roughly constant with a mean of 3:6. The ratios of the first k eigenvalues seem to be the critical parameters controlling the number of iterations until convergence.

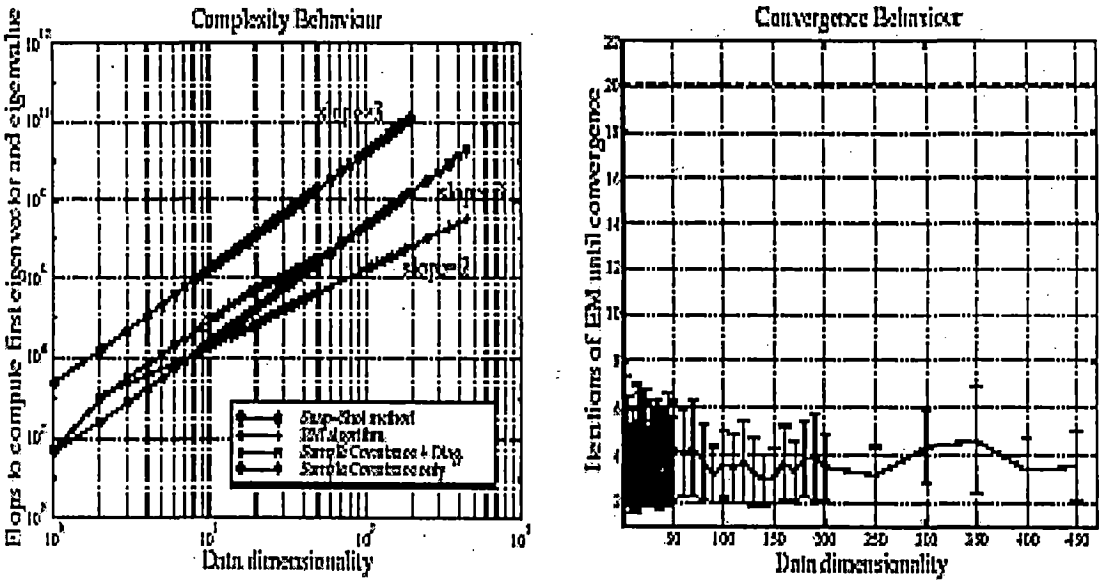


Fig. 2.4 Time complexity and convergence behavior of the algorithm.

In all cases, the number of data points n is 10 times the dimensionality p. For the left panel, the number of floating point operations to find the leading eigenvector and eigenvalue were recorded. The EM algorithm was always run for exactly 20 iterations. The cost shown for diagonalization of the sample covariance uses the MATLAB functions cov and eigs. The snapshot method is show to indicate scaling only; one would not normally use it when n > p. In the right hand panel, explicitly computing the leading eigenvector and then running the EM algorithm until the dot product of its guess investigated convergence and the true eigendirection was 0.999 or more. The error bars show ± one standard deviation across many runs. The dashed line shows the number of iterations used to produce the EM algorithm curve ('+') in the left panel.

2.3.2 Missing data

In the complete data setting, the values of the projections or hidden states x are viewed as the "missing information" for EM. During the **e-step** these values were computed by projecting the observed data into the current subspace. This minimizes the model error given the observed data and the model parameters. However, if some of the input points are missing certain coordinate values, we can easily estimate those values in the same fashion. Instead of estimating only x as the value that minimizes the squared distance between the point and its reconstruction we can generalize the **e-step** to:
Generalized e step : For each (possibly incomplete) point y find the unique pair of points x^* and y^* (such that x^* lies in the current principal subspace and y^* lies in the subspace defined by the known information about y) which minimize the norm $||Cx^* - y^*||$ Set the corresponding column of X to x^* and the corresponding column of Y to y*.

If y is complete, then $y^* = y$ and x^* is found exactly as before. If not, then x* and y^* are the solution to a least squares problem and can be found by, for example, QR factorization of a particular constraint matrix. Using this generalized e-step It is found that the leading principal components for datasets in which every point is missing some coordinates.

2.4 K-NN method for Classification

K-Nearest Neighbor (KNN) classification is a very simple, yet powerful classification method[19], [20]. The key idea behind KNN classification is that similar observations belong to similar classes. Thus, one simply has to look for the class designators of a certain number of the nearest neighbors and weigh their class numbers to assign a class number to the unknown.

The weighing scheme of the class numbers is often a majority rule, but other schemes are conceivable. The number of the nearest neighbors, k, should be odd in order to avoid ties, and it should be kept small, since a large k tends to create misclassifications unless the individual classes are well-separated. For face detection purpose to make misclassification least K is chosen as 1. So the classifier is 1-NN method.

It can be shown that the performance of a KNN classifier is always at least half of the best possible classifier for a given problem. One of the major drawbacks of KNN classifiers is that the classifier needs all available data. This may lead to considerable overhead, if the training data set is large.

CHAPTER 3

METHODOLOGY AND IMPLEMENTATION

3.1 Color Based Technique Using Neural Network

For simplicity, tried only to detect the presence of a mug shot that is a full front-view human face that is not obscured with little background noise. Our training data set consisted of 20 color images: 10 human faces and 10 non-faces as shown in Fig 3.1 and fig. 3.2. The ten faces were chosen to represent a variety of ages, genders, and skin tones. The other ten images were random objects taken from the Internet [24]. Some of the non-face images were chosen to "fool" the network. Each face would have a corresponding output $P = 1$, while images $11 - 20$ would have output $P = 0$.

Next, a group of 13 color images for a test data set are selected. An odd number of images is chosen to see if the system was above or below 50% accurate. That is, I wanted to check that the system was better than guessing. The network would not be trained on these images, so the performance on this data set would indicate the effectiveness of our system in face detection. A correct identification would result in a value $P > 0.5$ for all faces (images 1-8 of Fig.3.3) and a value $P < 0.5$ for all non-faces (images 9-13 of Fig. 3.3).

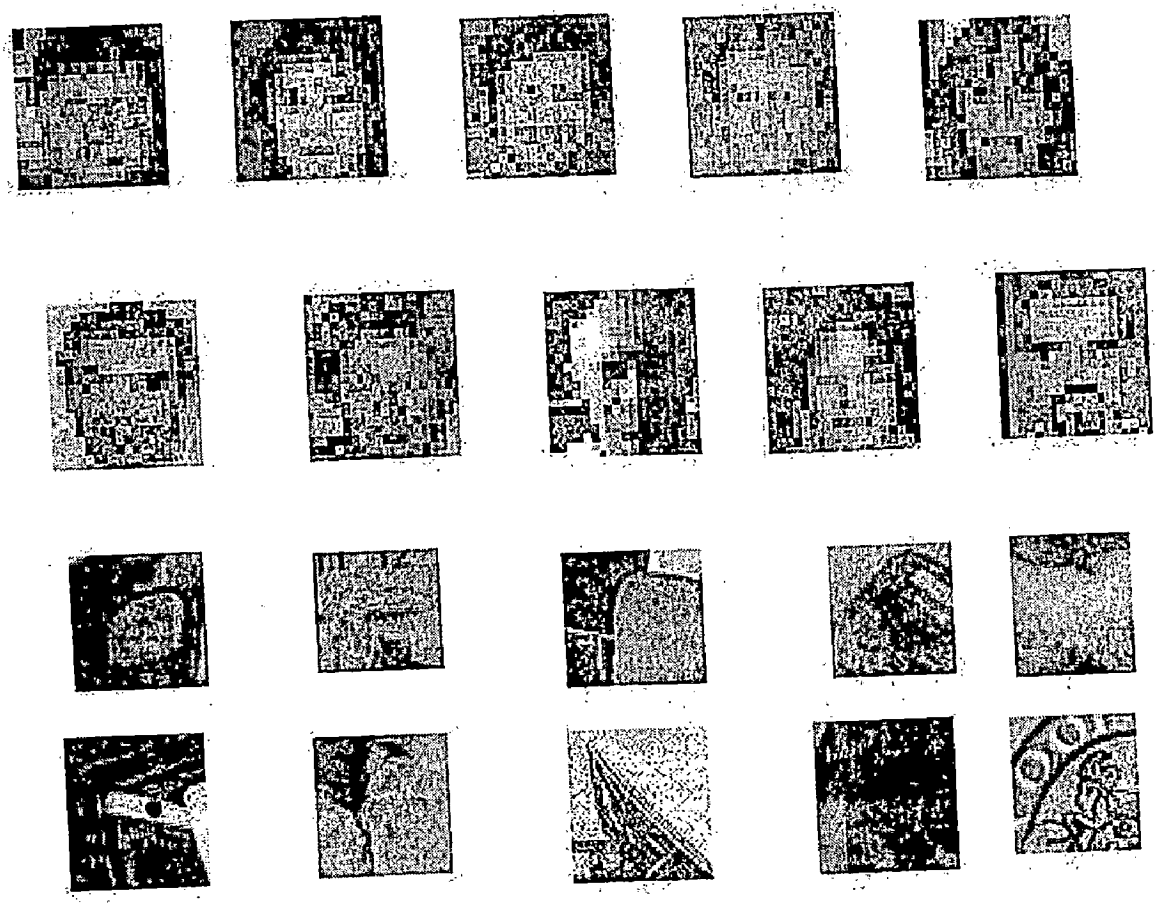


Fig 3.1 Training image set 1

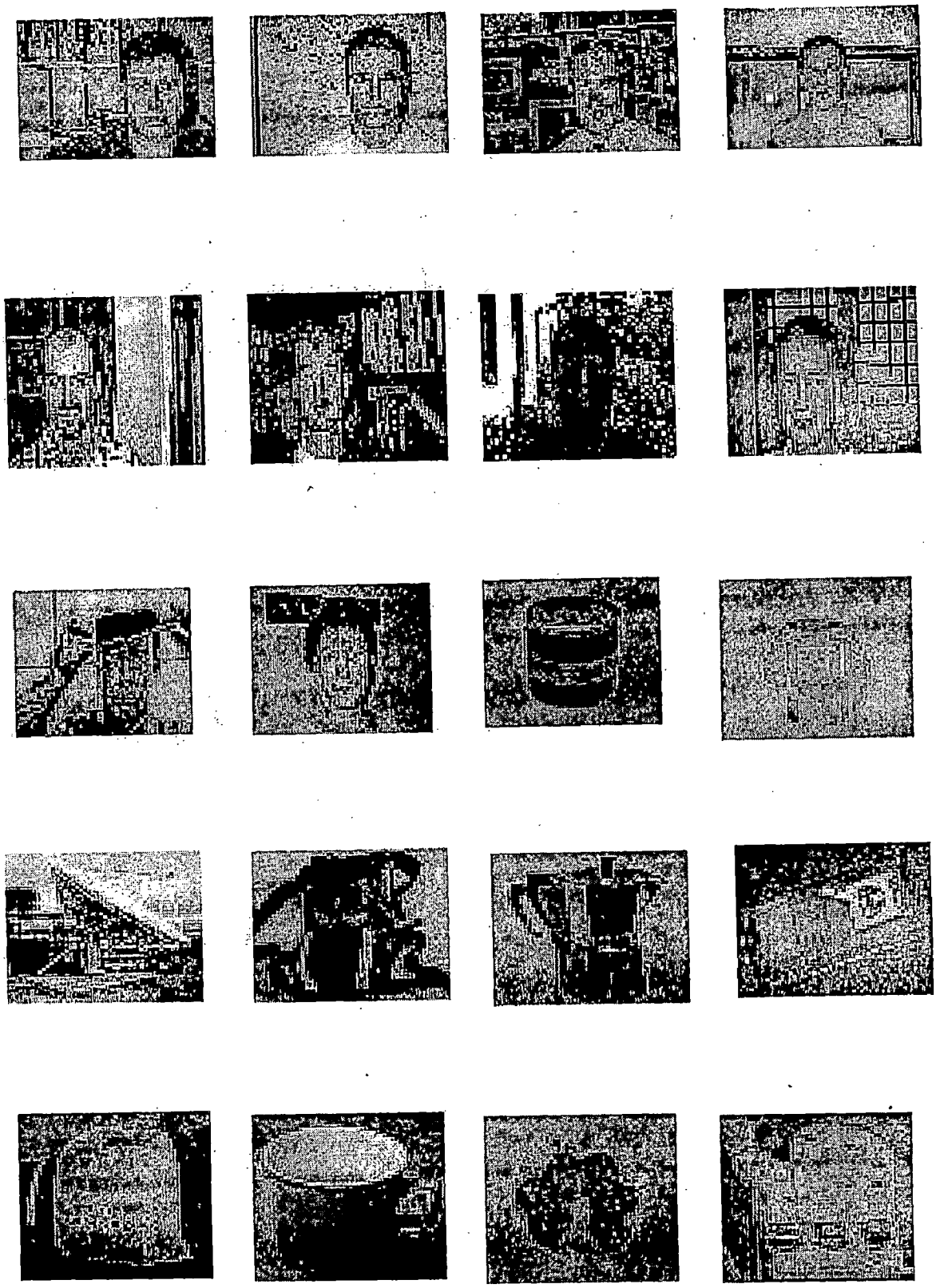
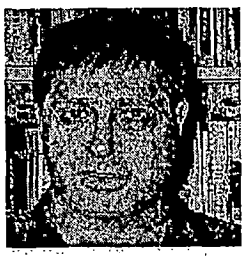
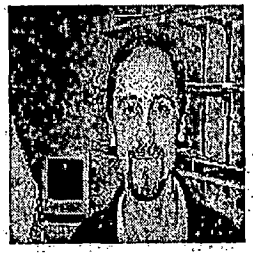


Fig 3.2 Training image set 2



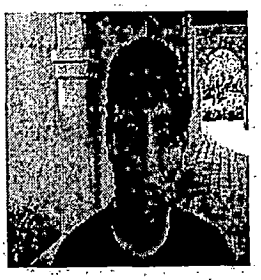
1



2

3

4



6

7

8

9

10

11

12

13

Fig. 3.3 Test image set

The only detail of the neural network that remains to be determined is the specific type of input that will be fed into the network. Since the matrix of weights for the training set will have size $20 \times 20(|X|+1) = 400(|X|+1)$, we wish to keep the input vector size $|X|$ as small as possible to keep the computational costs minimal. System was tested for two types of inputs: RGB histograms, and YES histograms. The two training and a test image sets above were used for both the systems to allow comparison.

3.1.1 RGB Histogram Approach

It is required to choose the input vector X that would represent the image globally. RGB values of each pixel in the image is cataloged in relative frequency histograms, where each component of the color space (R,G,B) was represented by N bins. The three histograms were then appended as one vector, so the input vector would have size $|X| = 3N$. It was hoped that the neural network would "learn" which bins correspond to flesh tones and develop an internal chroma chart.

N=20 bins were chosen for each component of the RGB color space. So this input vector has size $|X| = 60$. The Levenberg-Marquad algorithm was run on the two training data set for 110 iterations. The network outputs for the training were very favorable since each image was correctly classified. That is, the images containing faces (1-10) resulted in outputs $P > 0.5$ while the non-face images gave rise to outputs $P < 0.5$.

3.1.1.1 Performance Evaluation

The results on test data set were impressive. This approach correctly classified 12 of 13 test images for training set 1, or 92.3% and for training set 2 13 out of 13, i.e. 100%. From this two things can be concluded:

- Performance is depend upon training set
- RGB approach is an efficient method for face detection

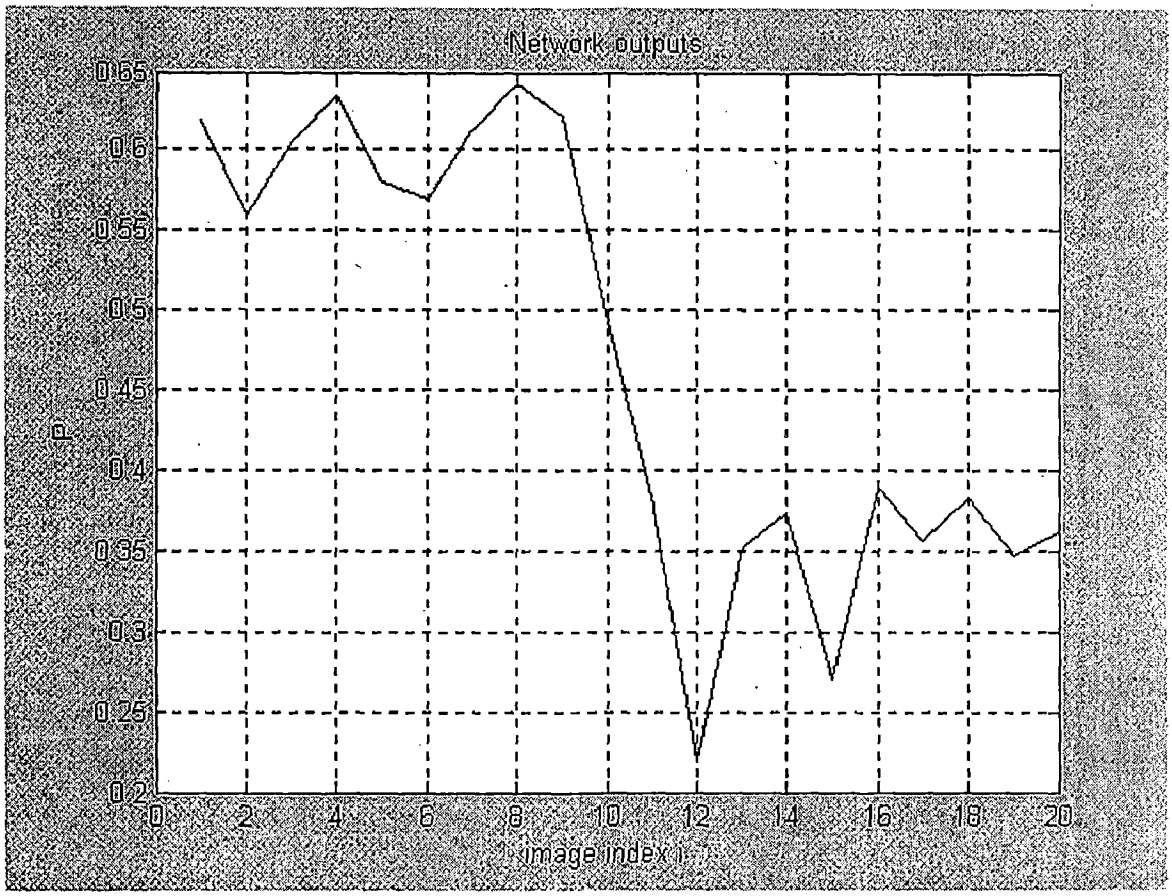


Fig. 3.4 Network outputs for Training set1 after 110 iterations of LM algorithm. The inputs were under RGB histogram approach

A common measurement of performance of classification system is the false positive and false negative rates [15]. Given by

$$\text{False Positive} = \frac{\textit{Number of incorrectly detected faces}}{\textit{Total number of actual faces}} \qquad 3.1.1.1(a)$$

$$\text{False Negative} = \frac{\textit{Number of missed faces}}{\textit{Total nuber of Actual faces}} \qquad 3.1.1.1(b)$$

For training set 1

$\text{FP} = \frac{0}{8} = 0$ (test data set) and $\frac{0}{10} = 0$ (training set)

$\text{FN} = \frac{1}{8} = 0.125$ (test dataset) and $\frac{0}{10} = 0$ (training set)

For training set 2

$\text{FP} = \frac{0}{8} = 0$ (test data set) and $\frac{1}{10} = 0.1$ (training set)

$\text{FN} = \frac{0}{8} = 0.125$ (test dataset) and $\frac{0}{10} = 0$ (training set)

Note that the errors were primarily from missed faces. This may be due to the fact that the training resulted in network outputs very close to 0.5 That is, even though the system correctly identified all 10 faces in the training set, all P values for images 1 through 10 were only slightly above 0.5 (see Fig. 3.4). Overall, taking the training and test data together, this approach correctly identified 32 of the 33 images (96.9%).

3.1.2 YES Histogram Approach

Although the RGB histogram approach yielded very good and positive results it is required tried and compare its performance with other histograms in color space. For this at the suggestion of Dr. Fadil Santosa, we implemented a color histogram approach in the YES color space. The transformation from the standard RGB space to the YES space is given by the equations below:

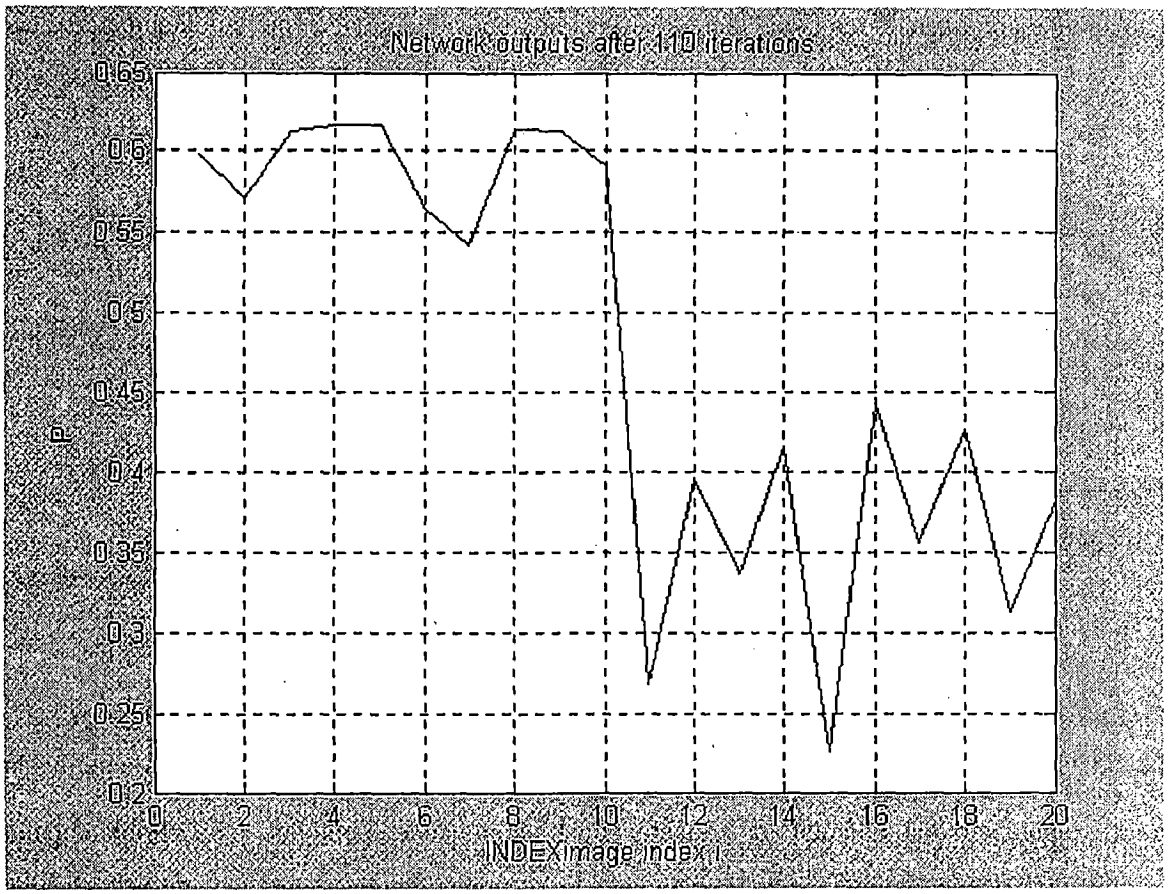


Fig. 3.5 Network outputs for Training set 2 after 110 iterations of LM algorithm. The inputs were under RGB histogram approach

$$Y = 0.253R + 0.684G + 0.063B$$
$$E = 0.5R - 0.5G \qquad 3.1.2(a)$$
$$S = 0.25R + 0.25G - 0.5B$$

In a sense, the Y matrix picks out the edges of the image while the E and S matrices encode the color intensities. The Y histogram may, in some sense, provide the neural network with spatial information, rather than just color intensities. As for the RGB histogram, we cataloged the pixel values in the image in three YES relative frequency histograms, each with N equally spaced bins. The histograms were appended to form one input vector X. Again, we chose N=20.

3.1.2.1 Performance Evaluation

The LM algorithm was run for 110 iterations with the inputs for the 20 training images determined by this YES histogram approach. After 110 iterations, the system had correctly classified 19 of the 20 training set 1 images (see Fig. 3.7). One image containing a face (image 9) had a *P* value just under 0.5. This mis-classification may have been due to the fact that images 15 and 16 contained flesh tones matching colors, while the face in image 3 had the darkest skin tone in the training set. It is probable that the neural network would be able to correctly classify the training set images if the training was run for more iteration. The detection rates for the training set were:

For training set 1

$\text{FP} = \frac{0}{8} = 0$ (test data set) and $\frac{1}{10} = 0.1$ (training set)

$\text{FN} = \frac{3}{8} = 0.375$ (test dataset) and $\frac{1}{10} = 0.1$ (training set)

For training set 2

$\text{FP} = \frac{0}{8} = 0$ (test data set) and $\frac{5}{10} = 0.2$ (training set)

$\text{FN} = \frac{2}{8} = 0.25$ (test dataset) and $\frac{0}{10} = 0$ (training set)

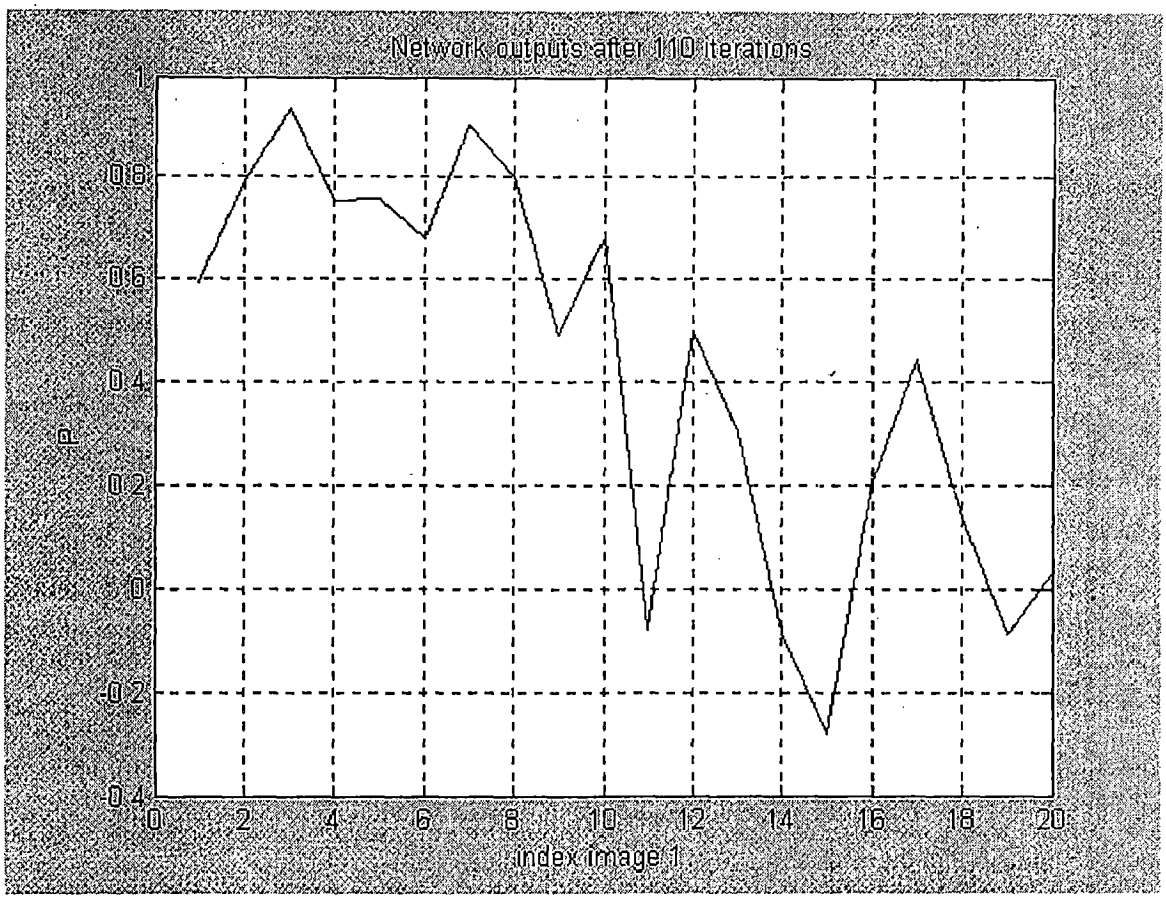


Fig 3.6 Network outputs for Training set 1 after 110 iterations of LM algorithm. The inputs were under YES histogram approach

The results for the test data set were less encouraging. The system correctly classified 10 out of 13 (76.9%) of the test images for training set 1 and 11 out of 13 (84.3%) of test images for training set 2. So overall, this system correctly classified 28 out of 33 (84.8%) of the images for training set 1 and 26 out of 33 (80.8%) of the images for training set 2. So RGB histogram approach shows the better results in comparison with YES histogram approach.

3.2 Appearance Based Technique

In Appearance based technique face detection algorithms consist of two parts:

- Feature extraction
- Classification.

The purpose of feature extraction is preprocessing the image data to get better representation of data, as to facilitate better classification results. Also, many classification schemes exist; each has its peculiar strength & weakness. Here PCA based feature extraction technique is used. Principal Component Analysis (PCA), a frequently used statistical technique for optimal lossy compression of data under least square sense; provide orthogonal basis vector space to represent original data.

Purpose of Classification is to classify the given dimensional space into two output spaces whether it is face like structure or not based on information derived from learning process. Here for Classification, K Nearest Neighbor (K-NN) method is used, and typically 1-NN while K=1.

Basically, any combination of FE+Cl can be used like in this face detection problem PCA + K-NN method is used. Based on experience & assumption it is easier to understand/implement PCA + 1-NN method.

Implementation is carried out using MATLAB 6.1 in windows environment. Various properties are:

- The codes includes basic image I/O, Also codes for simple PCA analysis and probablistic clustering algorithm are written in MATLAB and it gives out results of efficient face detector.

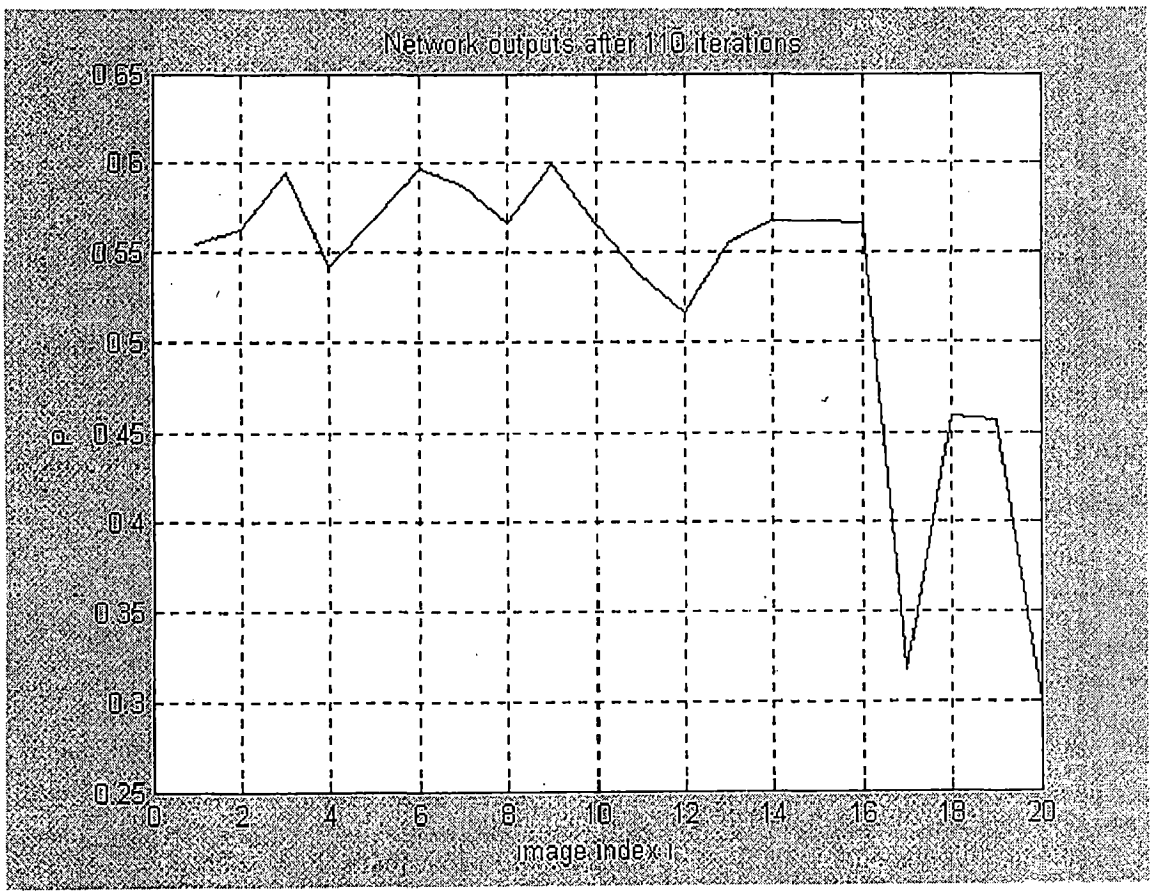


Fig. 3.7 Network outputs for Training set 2 after 110 iterations of LM algorithm. The inputs were under YES histogram approach

- The training data is composed of two parts: face data and non-face data. Program could learn offline from the training data, and you can get a feeling of its performance by running on the testing data provided.
- Due to limited amount of disk/memory space and to remove the complexity, only limited training data is used. But the training Data is for real face detection application, so it can be of very big in size.
- The training data is cropped face images with fixed size (19x19 pixels). Currently only front view is used.
- Currently only several simple test images are used for easy of comparison. To test the performance of program other type of test image set used.

3.2.1 Implementation Steps

Following steps are involved during implementation of face detection task:

1. Image Utilities

- Image vector is build from a rectangular image array.
- Result vector is build to match the image vector.
- Builds a rectangular binary mask array for face images.
- Subtracting a linear lighting plane and then rescaling the grayscale distribution histogram normalize image.
- Image set is augmented with the left right flipped versions of image.
- Rebuilding of before mask data.
- Simple I-NN classifier used to classify the face and non-face portion.
- Simple post processing to reduce the amount of detected faces.

2. Image Loading and Image Display

- A set of image according to given pattern set is loaded
- A set of image is also loaded via bootstrapping.
- Subplot of set of images in an image array.

- Pgm format images are read by different function.

3.2.2 Training

Training is a very important step in face detection problem and here PCA method is used for training. Training images are first preprocessed i.e. they are cropped (to left face pixels only) and scaled to appropriate size (19*19) and then load in the images and make up the training data matrix. Using approx. 4000 images of pgm format training is done. Most attractive feature of this training algorithm is that it takes only few seconds to train 4000 images.

CHAPTER 4

RESULTS AND INTERPRETATIONS

In this chapter Results of two techniques on various test images is presented and then results are interpreted.

4.1 Color Based Technique Using Neural Network

In Color based technique two inputs for NN is taken RGB and YES histogram and both are tested for same test images and two training sets. In the result picture histogram of image and network output is shown, if the network output is greater than 0.5 then it represents face and if it is less than 0.5 it is non face image. In face image as shown in fig 4.1 (a), network output P, is 0.600248. Similarly histogram and network outputs for all other test images are clearly shown in Fig.4.1 (a-m), Fig 4.2(a-m), Fig4.3 (a-m) and Fig 4.4(a-m). These network outputs for two training sets and various test images is also tabulated.

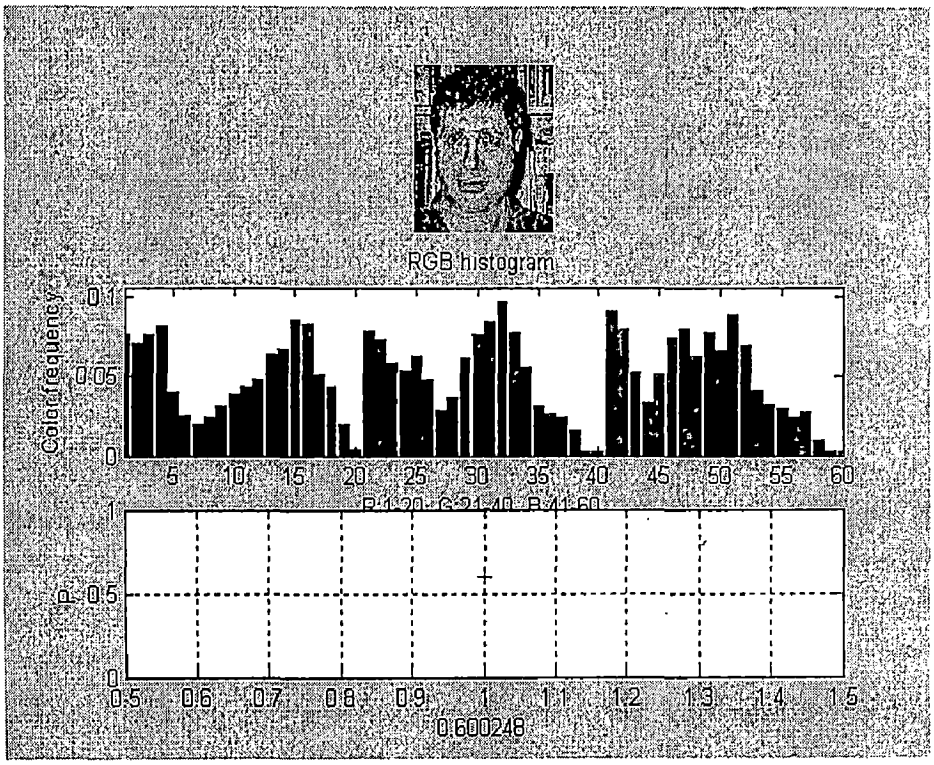
Of the two approaches RGB histogram shows the better results over YES histogram approach but they are very close to each other on performance basis. All of these results are based on rather small training and test data sets. To better judge the effectiveness of these approaches, we need to test the systems on a larger data set.

Two training data set give different network output. It is concluded that output and performance of detection process is very much depended upon:

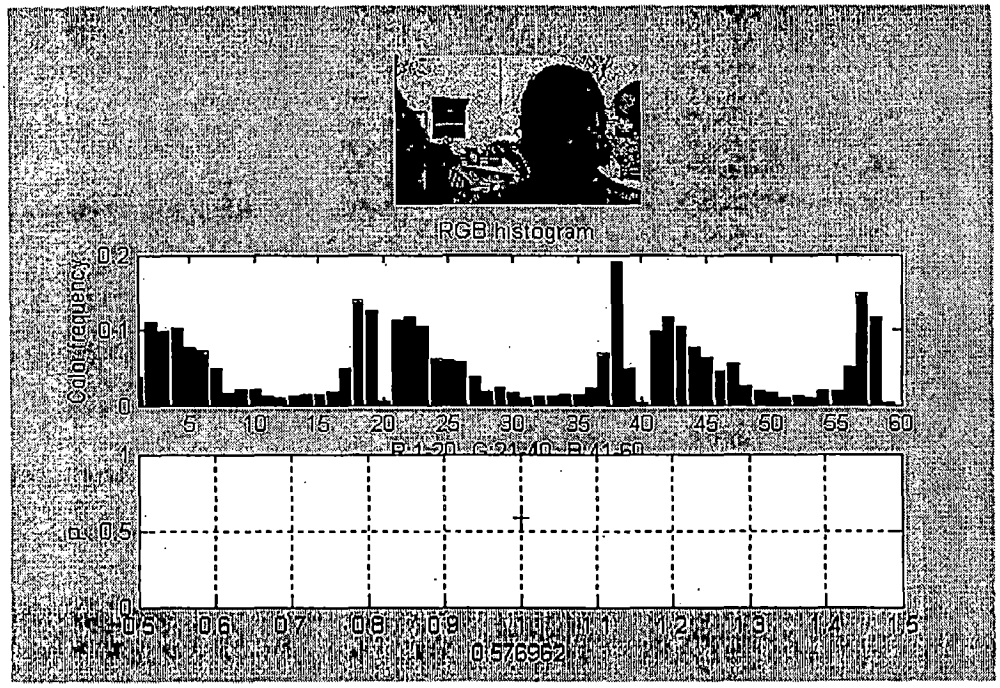
- Number of training images, current training set of 20 images is rather small increasing the size of training set while being careful to represent different possible faces, would allow the training algorithm to set the weights more accurately.
- Types of training image set, brightness, noise, and intensity level will change the network output as they provide different information to the network.

- Number of iterations of LM algorithm, the results obtained in the present system is after 110 iteration of LM algorithm. Increasing the number of iterations can increase detection efficiency.
- Type of input X fed to the NN, it contains information.
- Number of hidden nodes, increasing the number of hidden nodes might also improve the performance.
- Value of N, bins, increasing the value of N would provide the neural network with more input information.

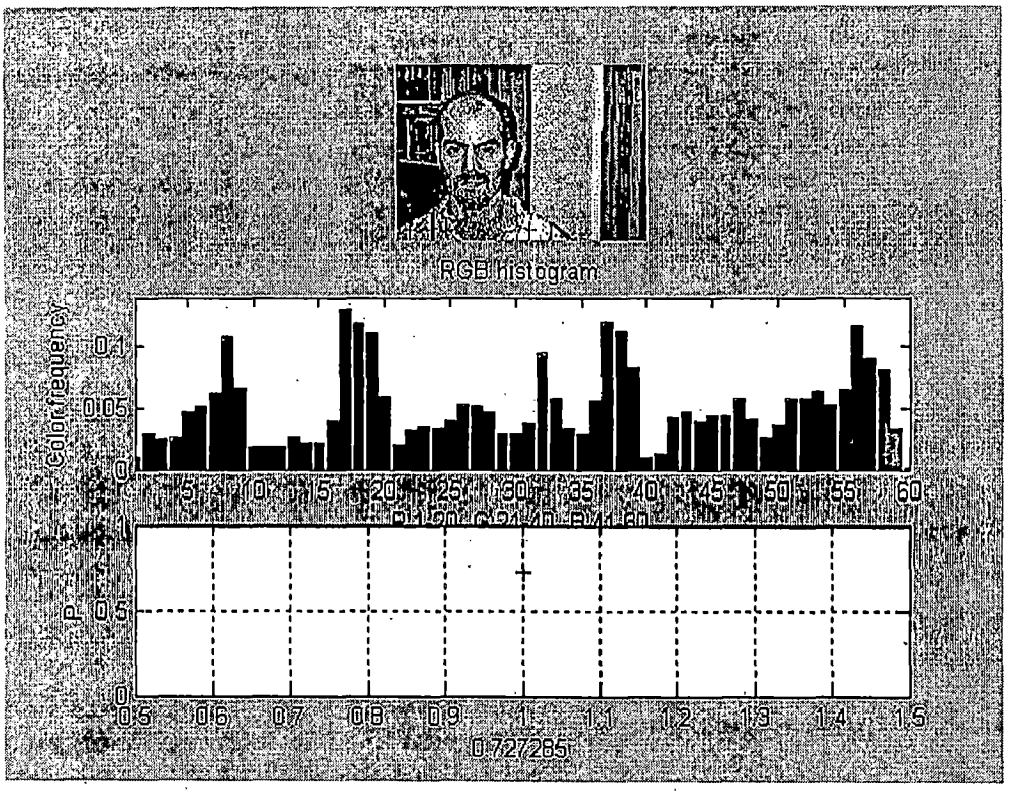
However, all these changes would increase the training costs, complexity, and training time and reduce the ease of obtaining results.



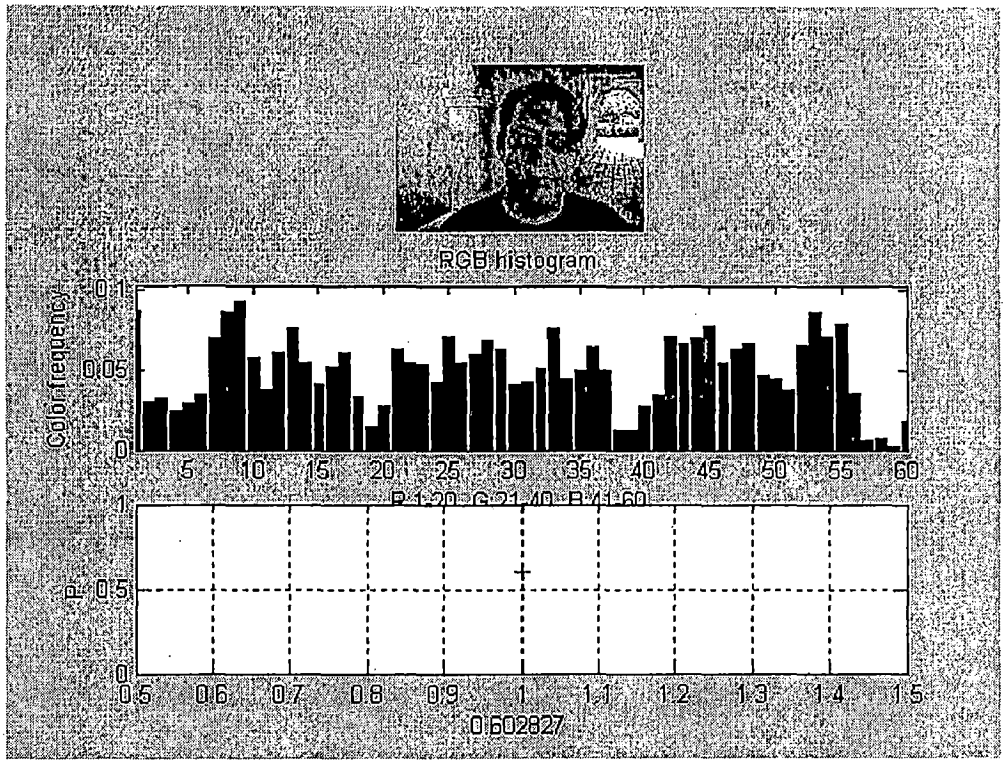
(a)


RGB histogram
Color frequency
0.2
0.1
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.576962

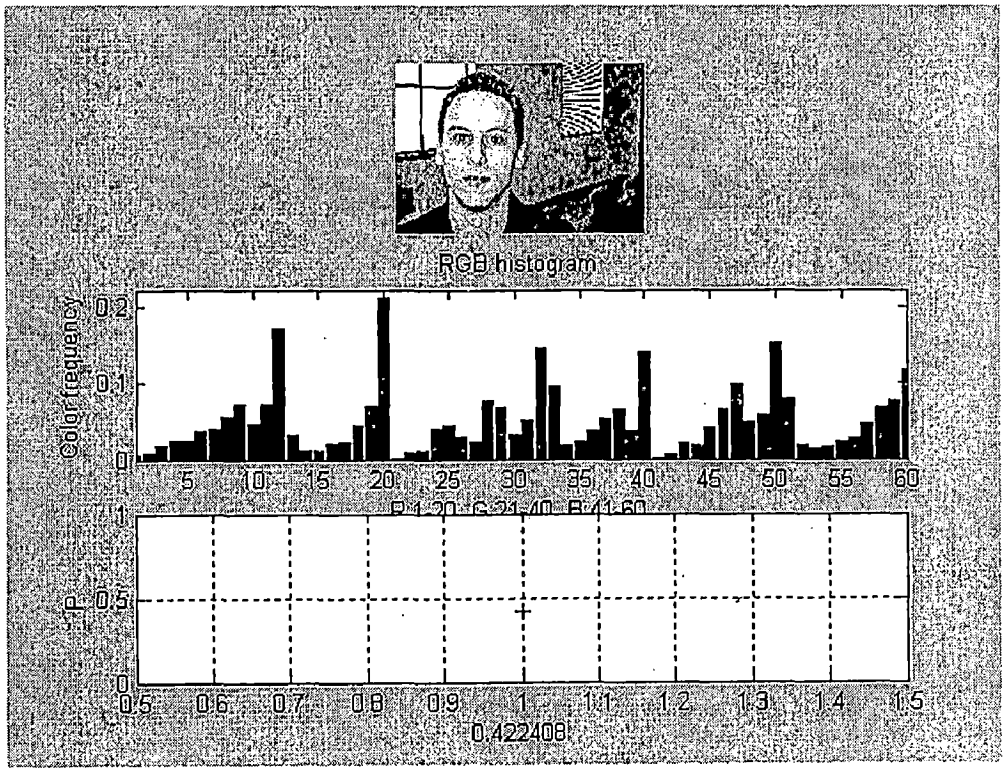
(b)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.727285

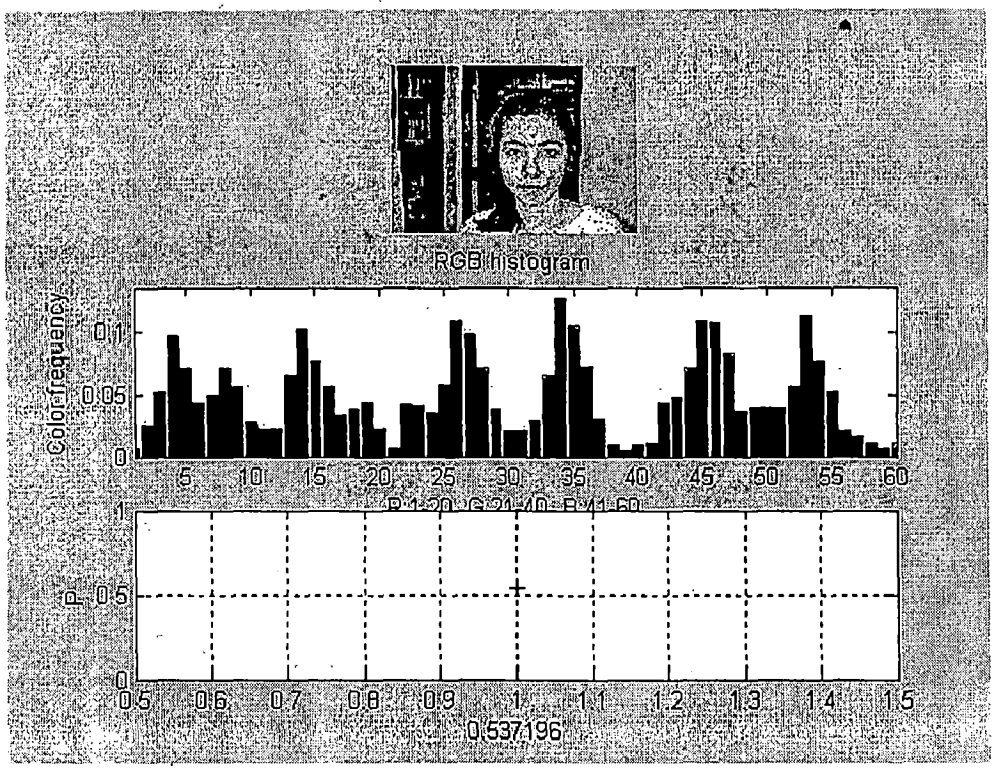
(c)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
R 1.20 G 21.40 B 11.60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.602827

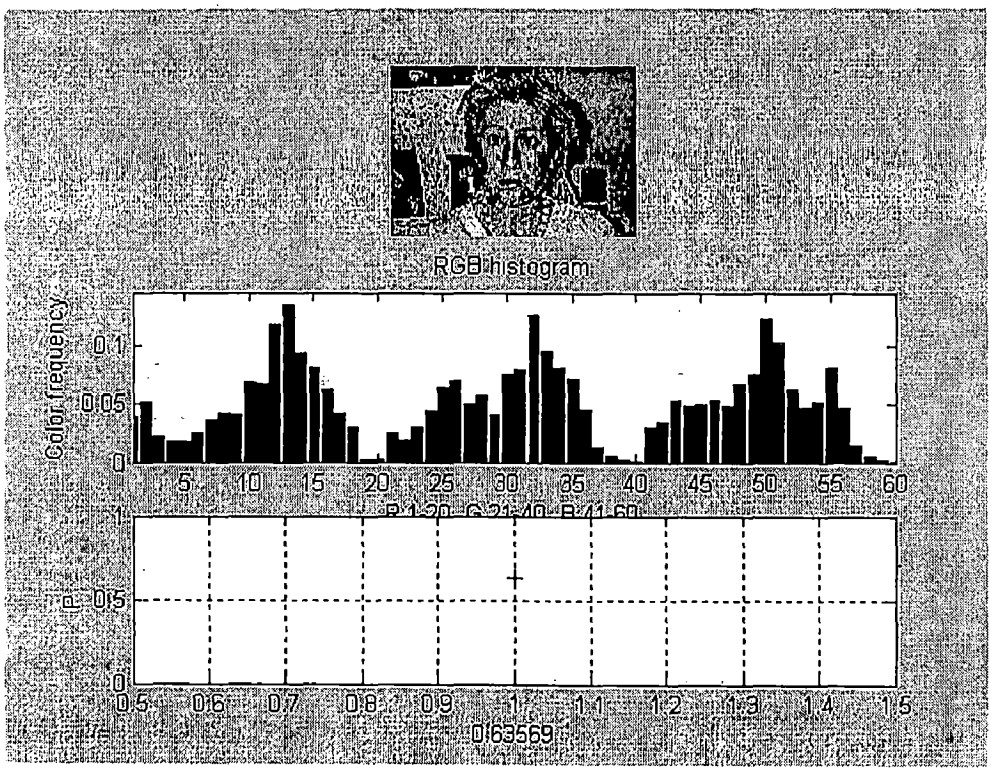
(d)


RGB histogram
Color frequency
0.2
0.1
0
5 10 15 20 25 30 35 40 45 50 55 60
R 1.20 G 21.40 B 11.60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.422408

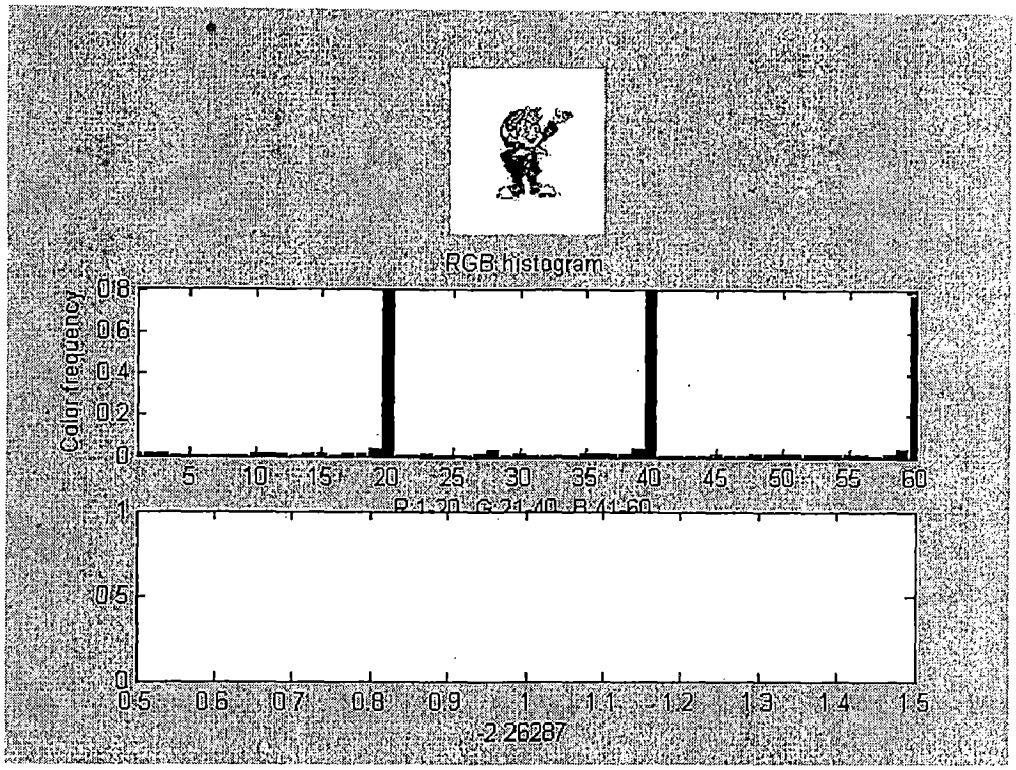
(e)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.537196

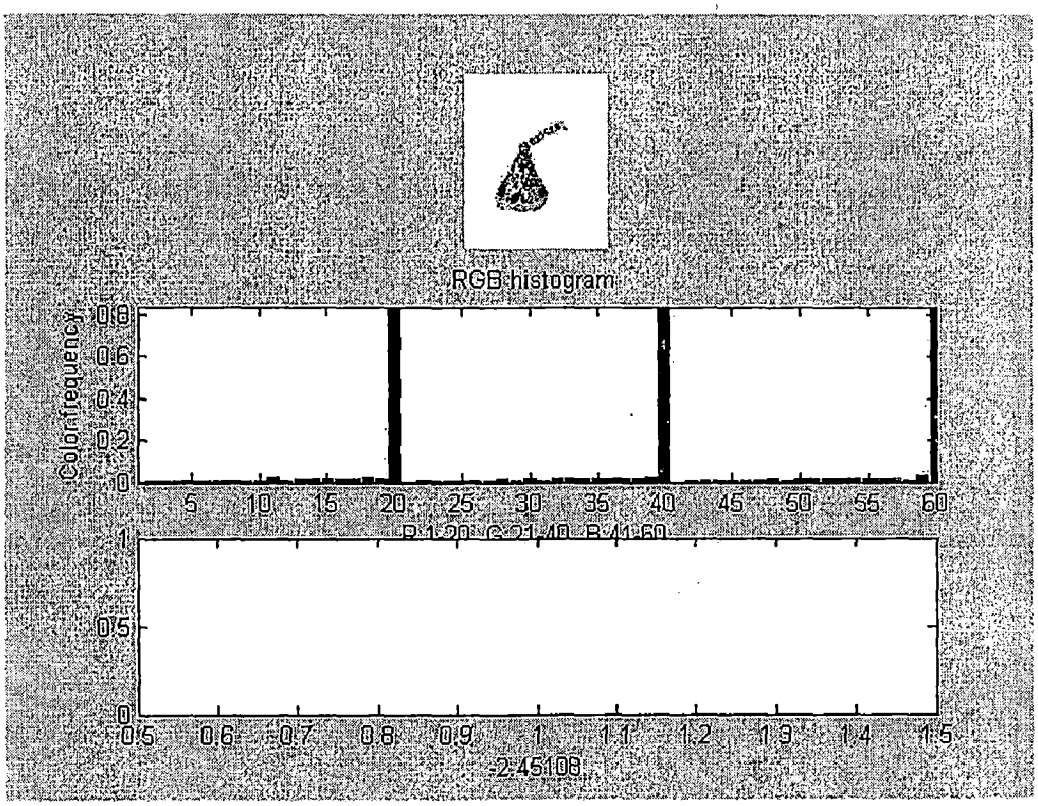
(f)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.63569

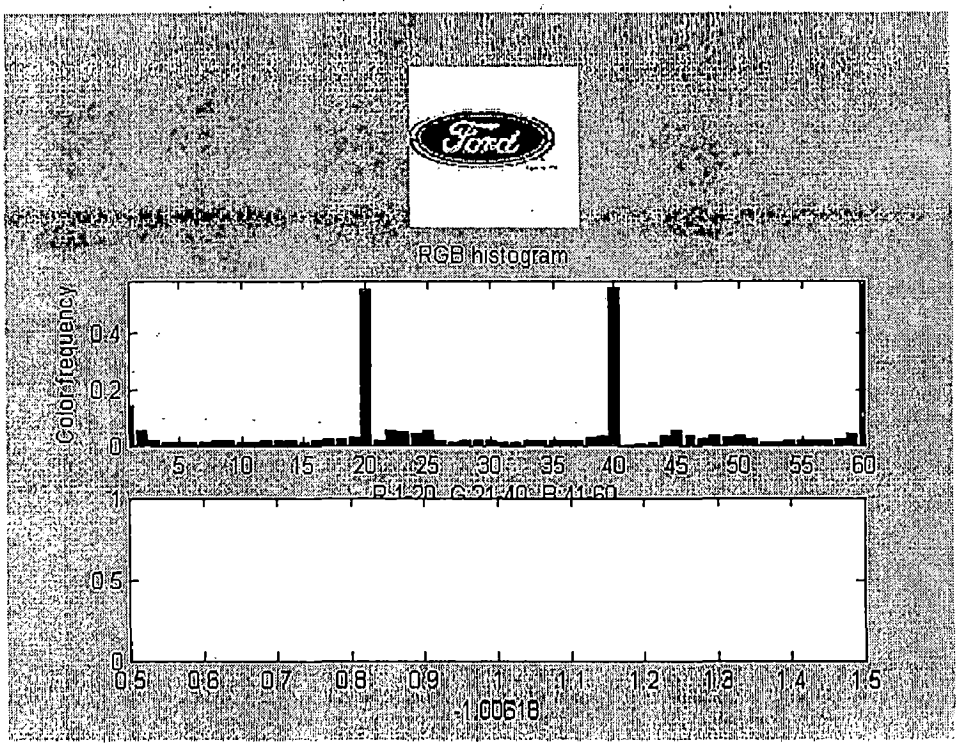
(g)


RGB histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
-2.26287

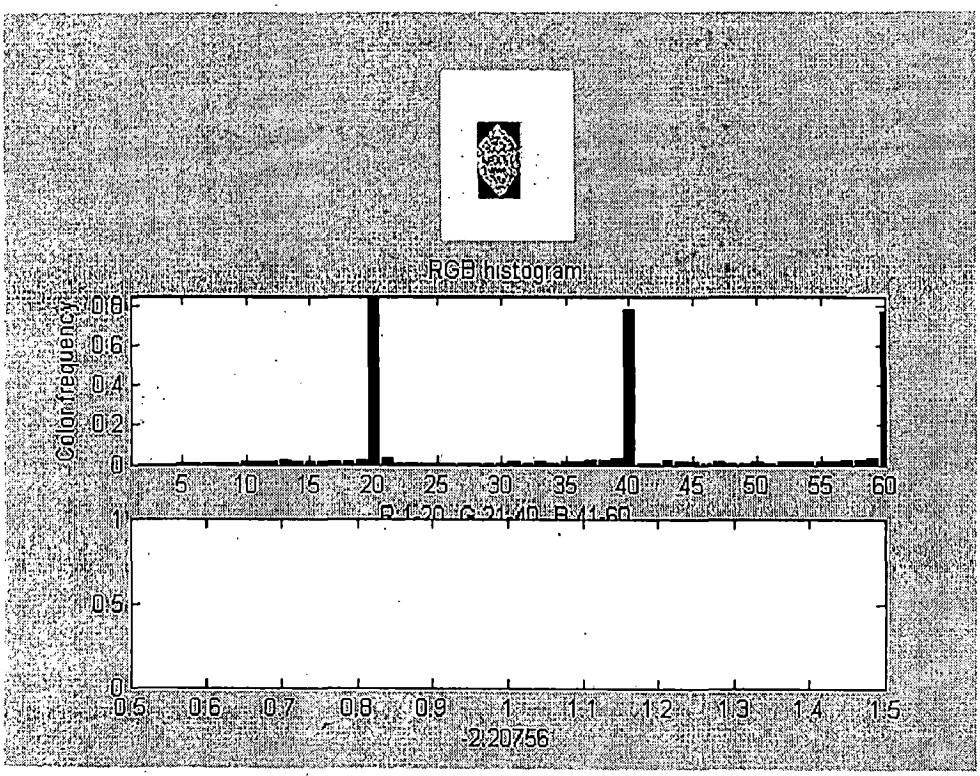
(h)


RGB histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
-2.45108

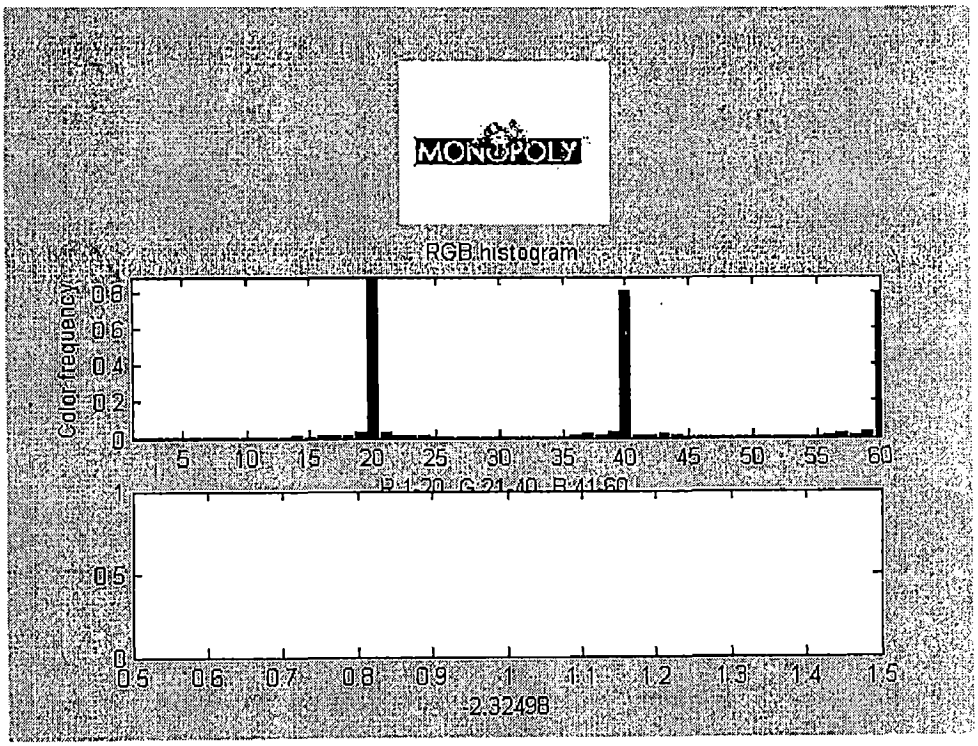
(i)


RGB histogram
Color frequency
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
1.00618

(j)

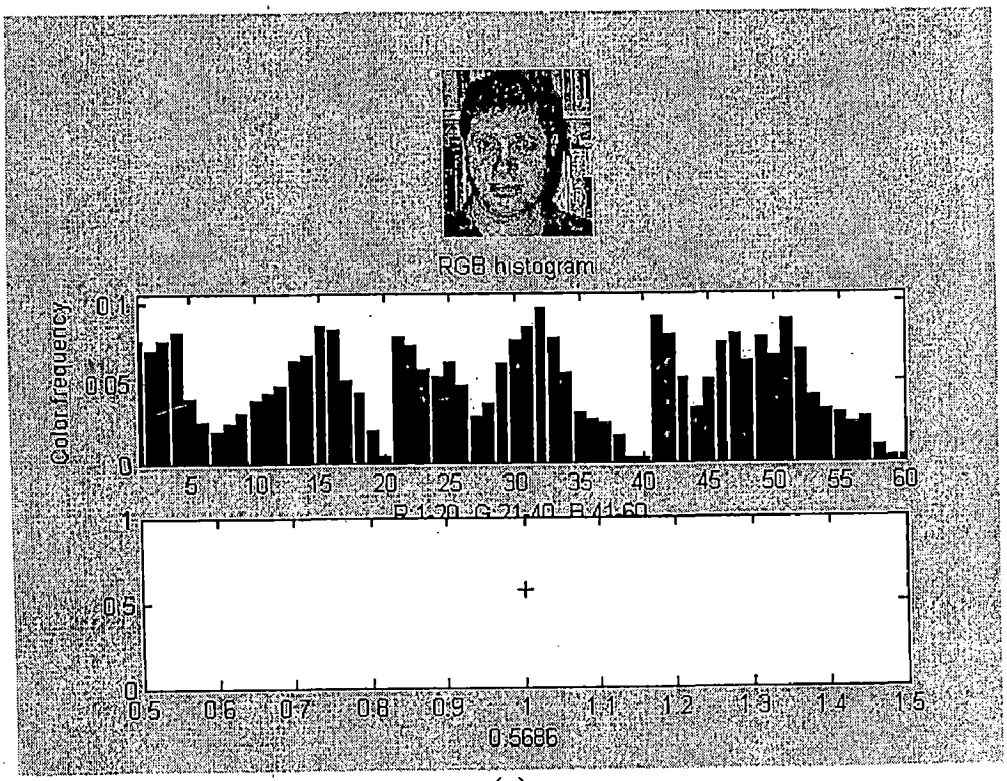

RGB histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
2.20756

(k)

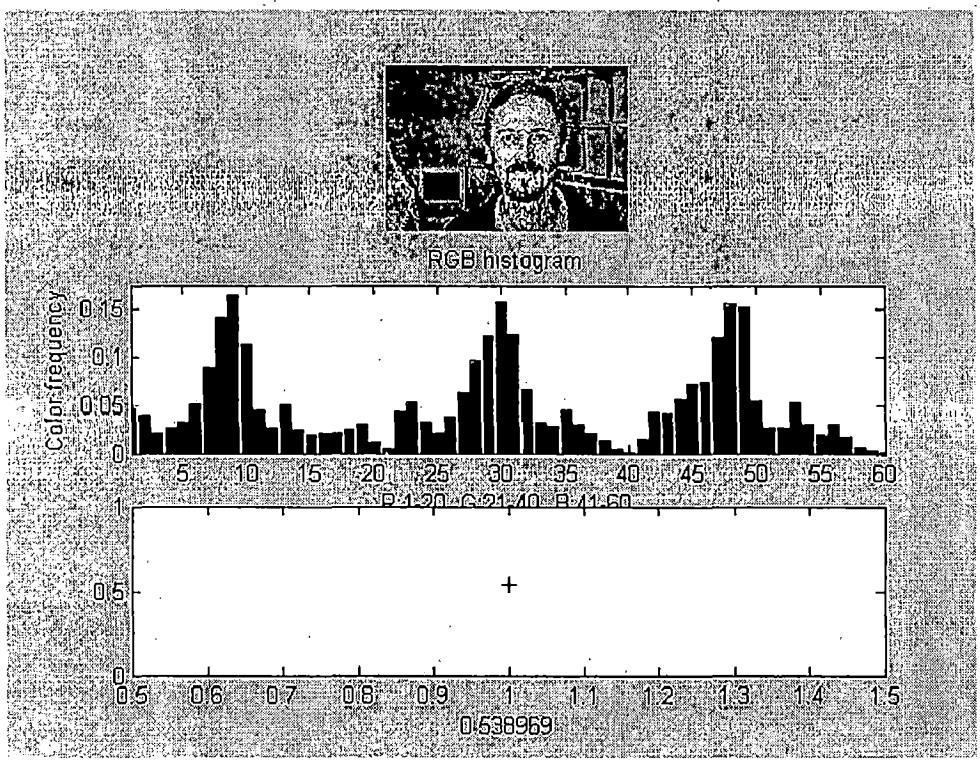


(l)

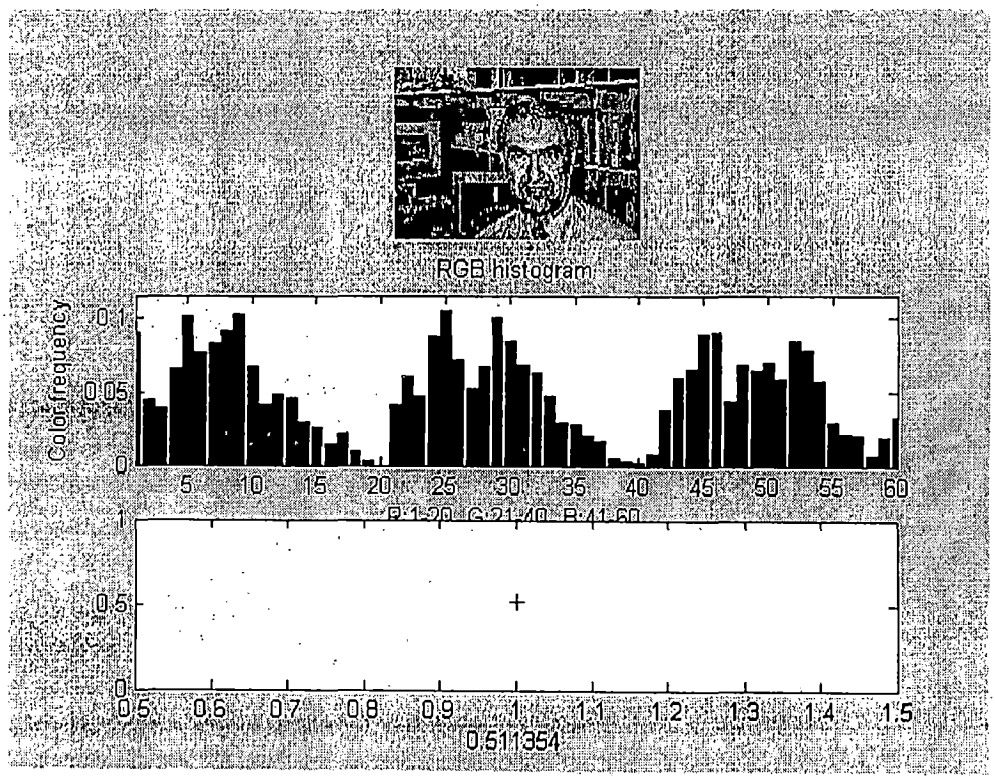
Fig 4.1 RGB histogram with N=20 for training set 1



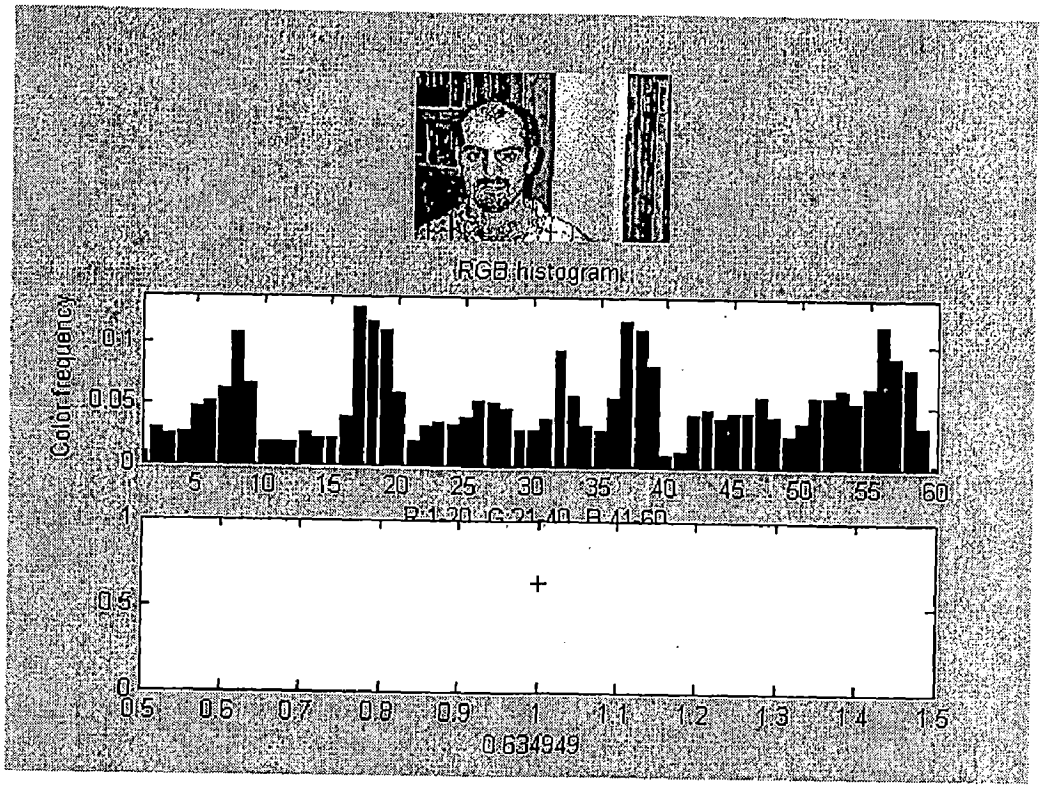
(a)


RGB histogram
Color frequency
0.15
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.538969

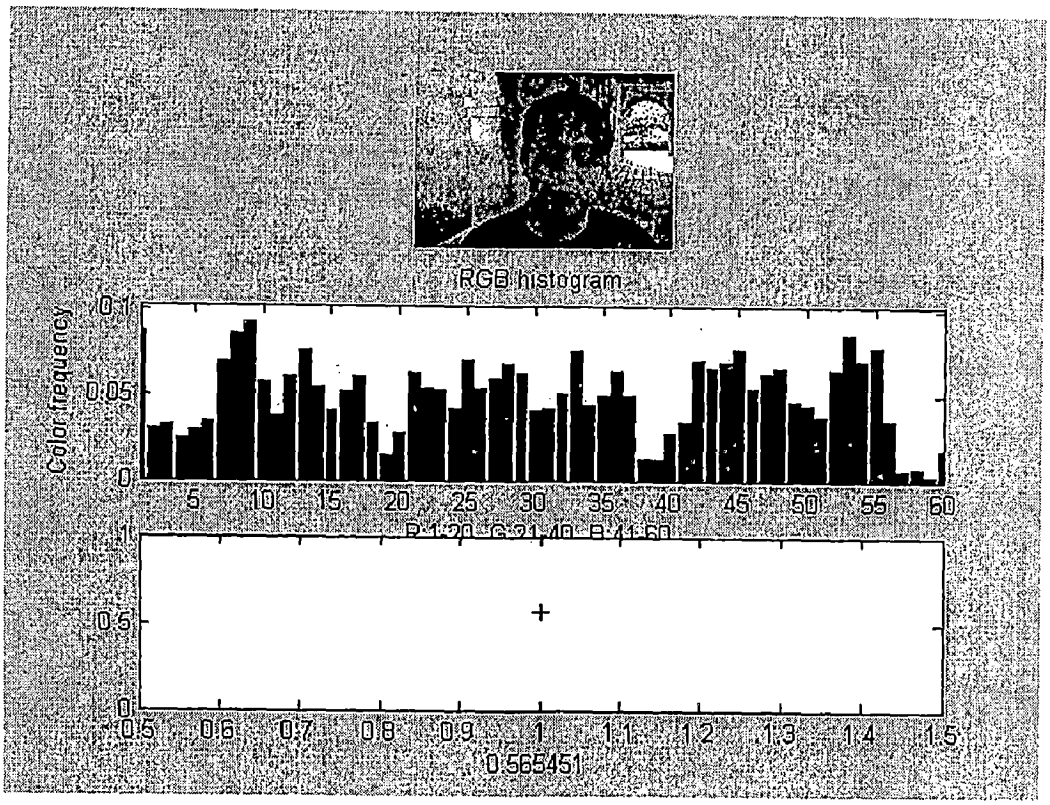
(b)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.511354

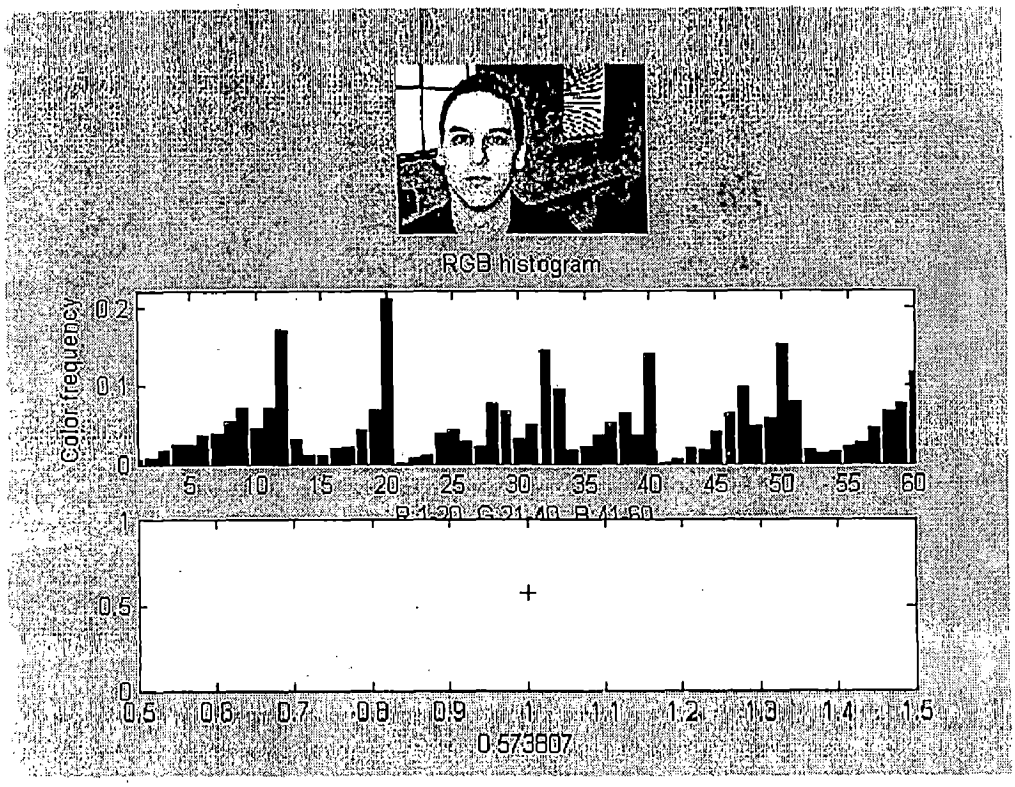
(c)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.634949

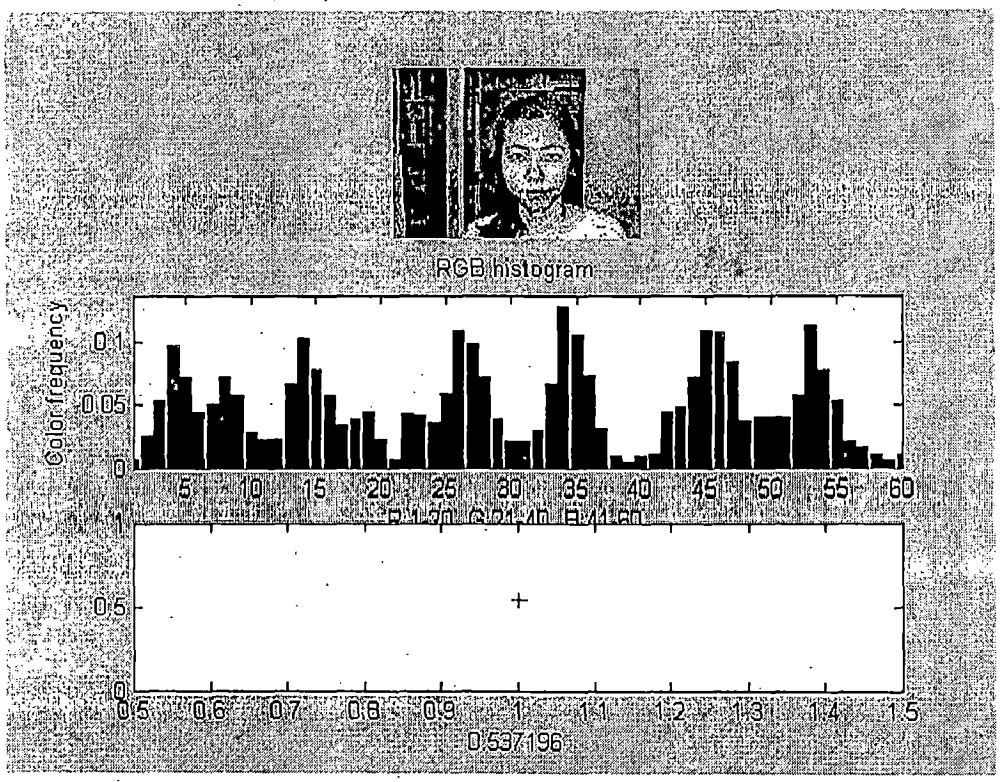
(d)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.565451

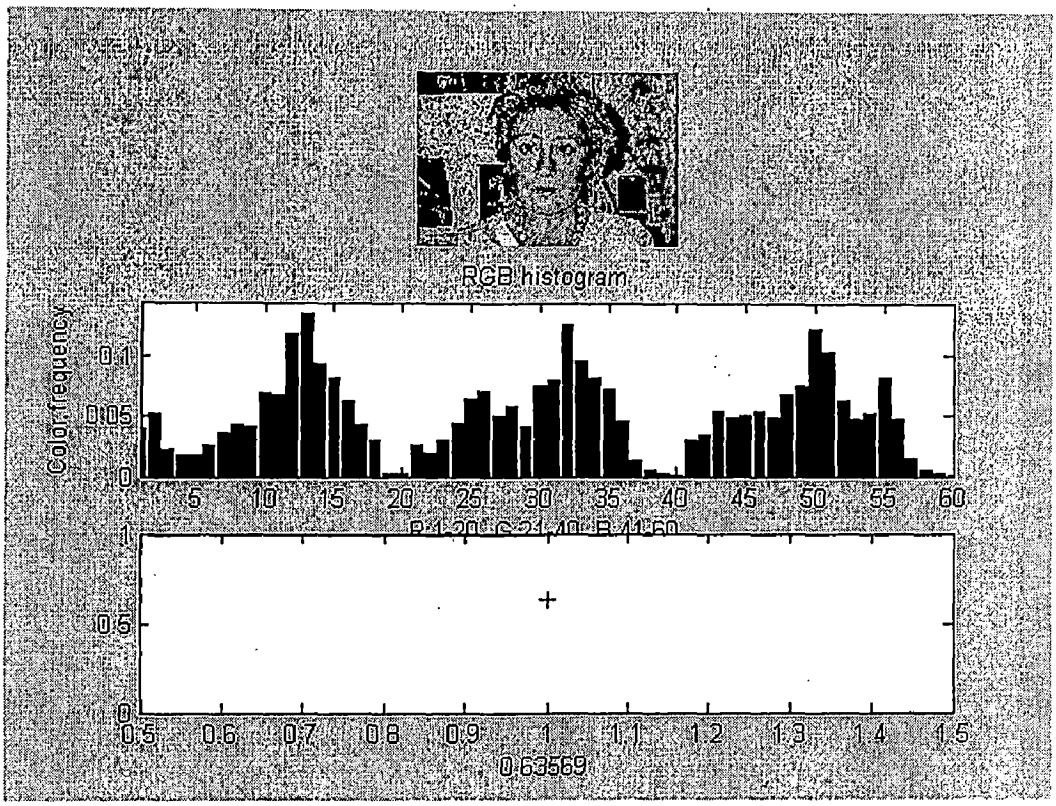
(e)


RGB histogram
Color frequency
0.2
0.1
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.573807

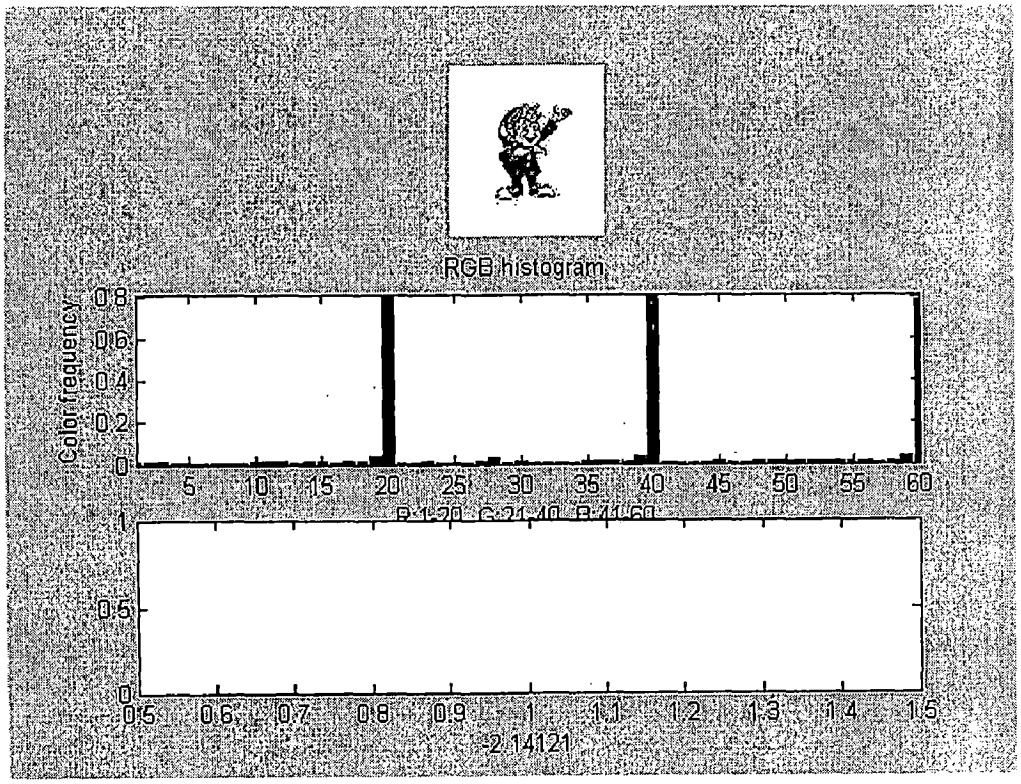
(f)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.537196

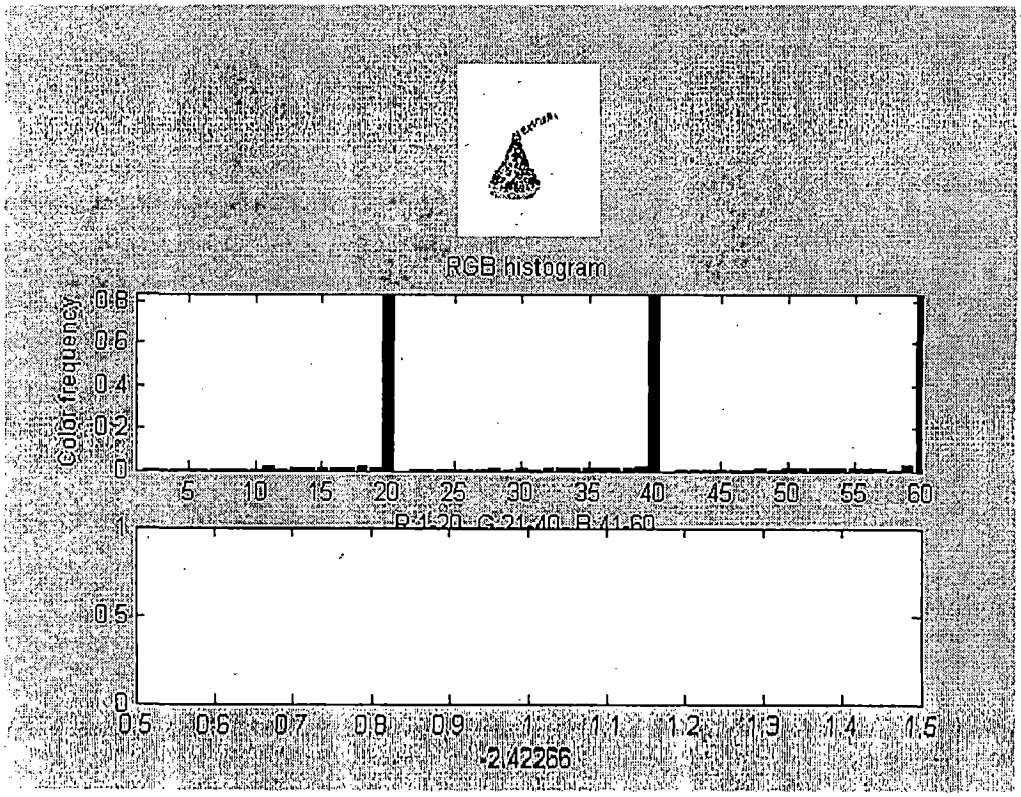
(g)


RGB histogram
Color frequency
0.1
0.05
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.63569

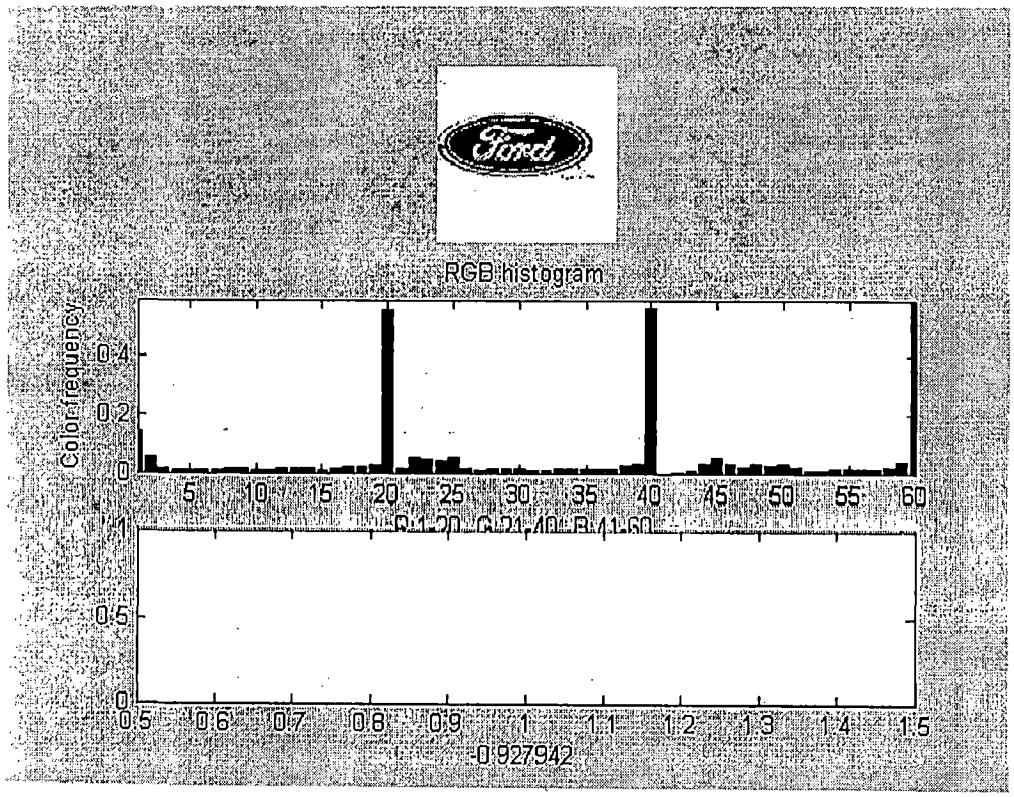
(h)


RGB histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
-2.14121

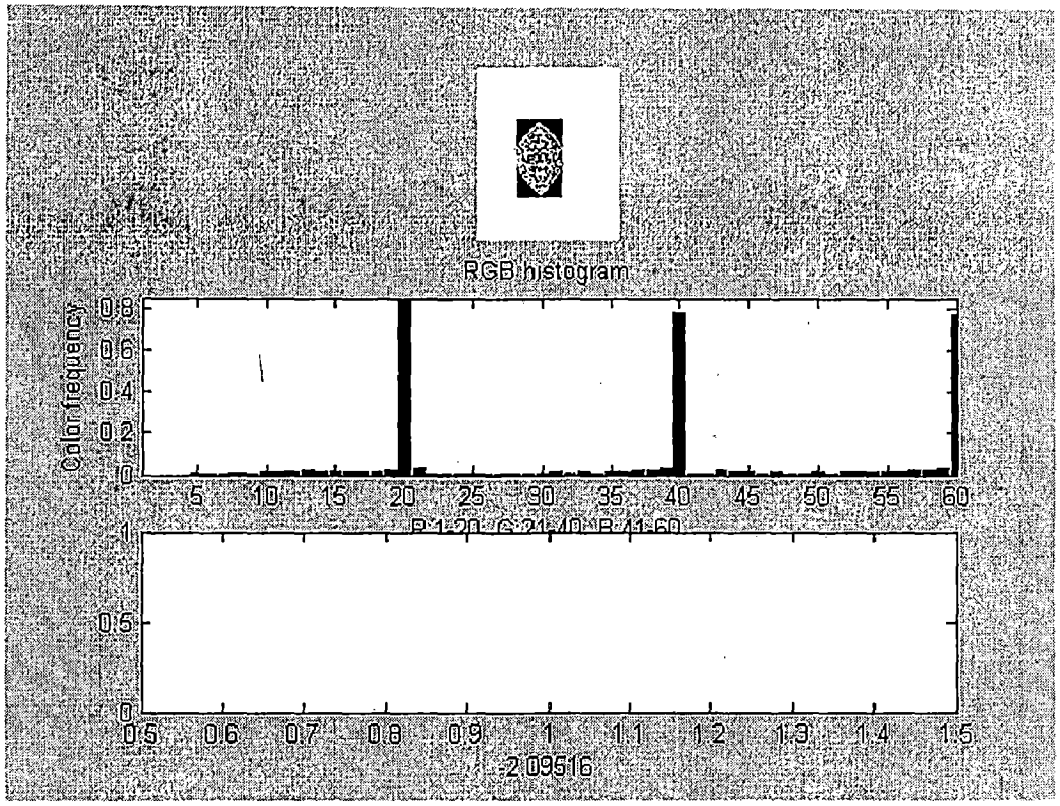
(i)


RGB histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
2.42266

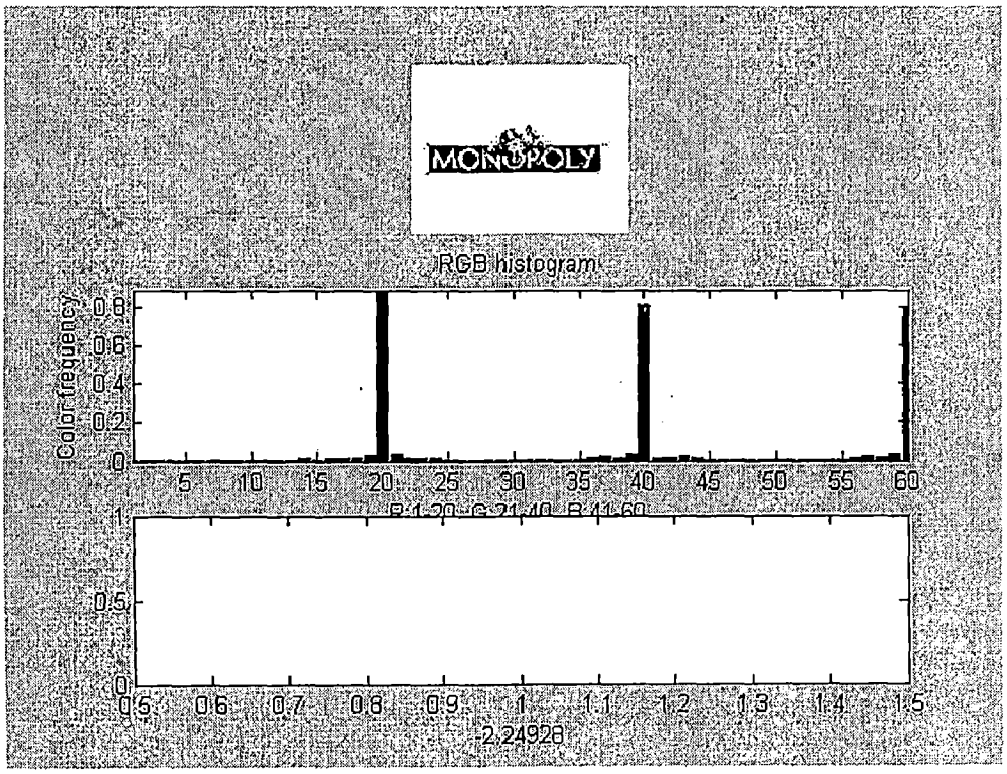
(j)


RGB histogram
Color frequency
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
R:1-20 G:21-40 B:41-60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.927942

(k)

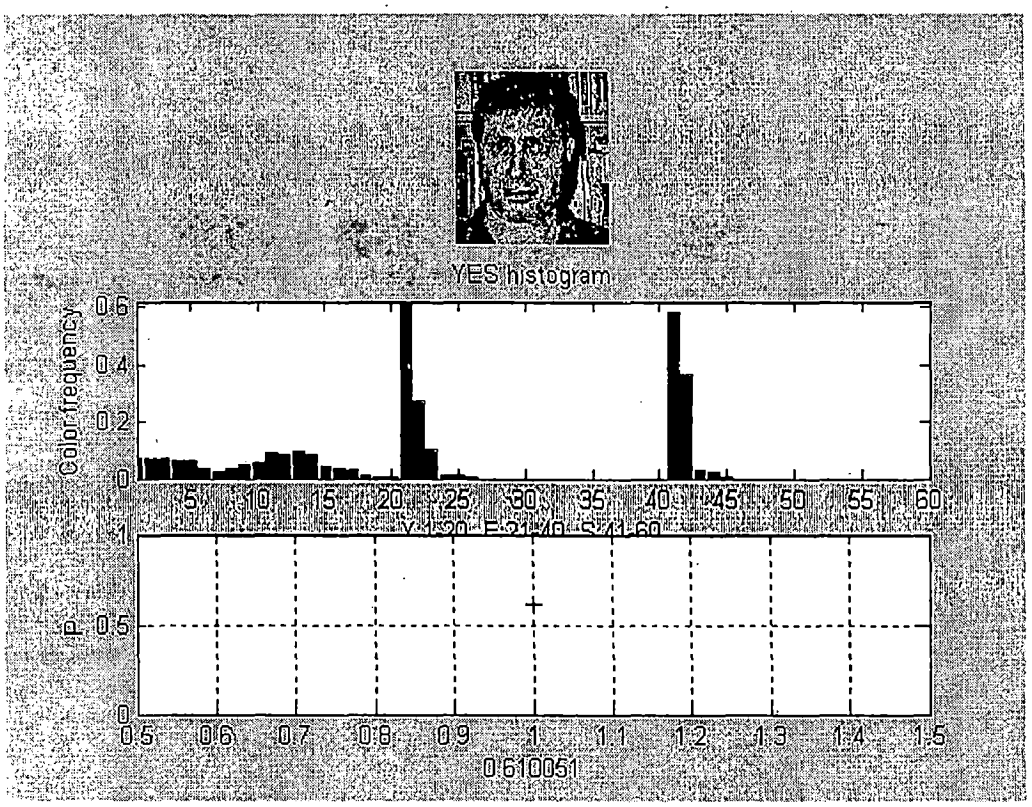


(l)

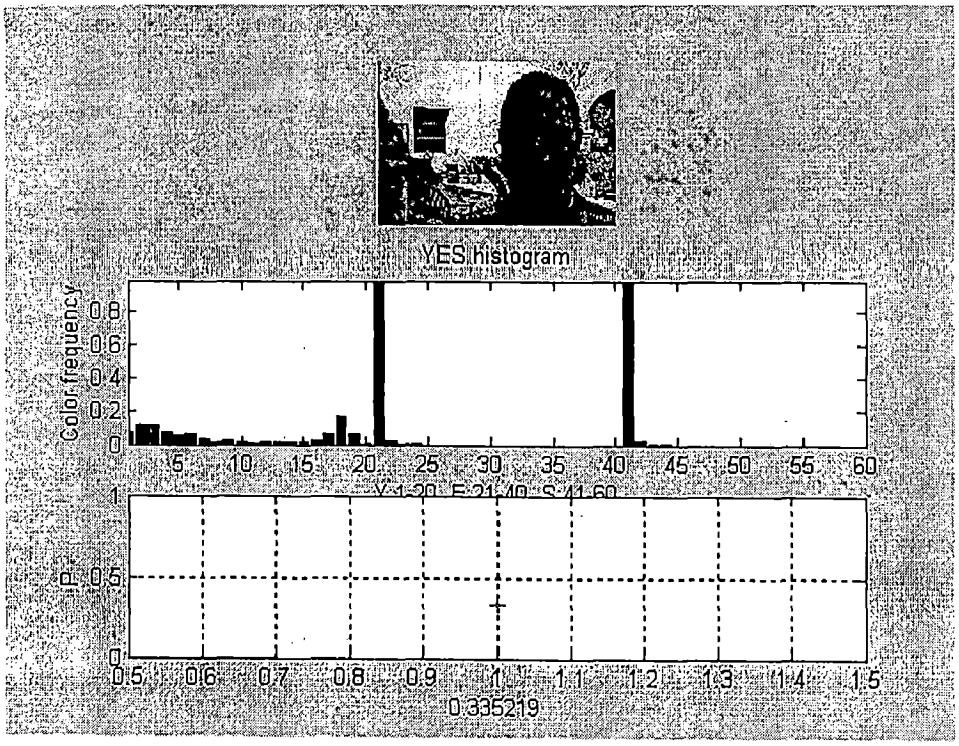


(m)

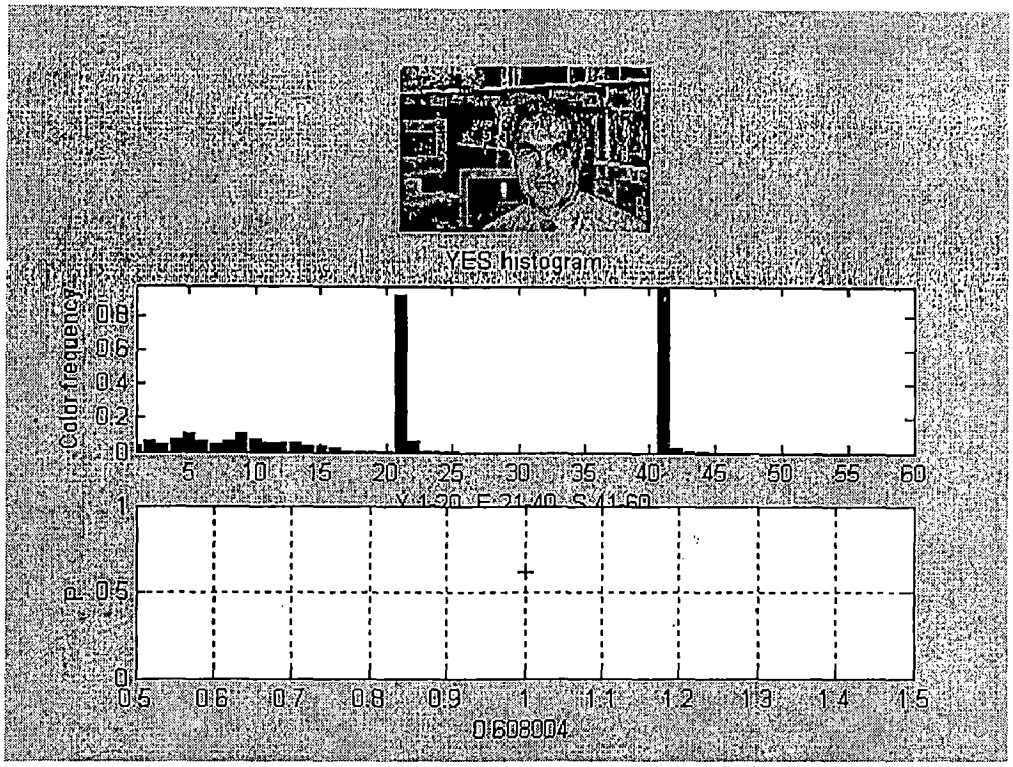
Fig 4.2 RGB histogram with N=20 for training set 2


YES histogram
Color frequency
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
P
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.610051

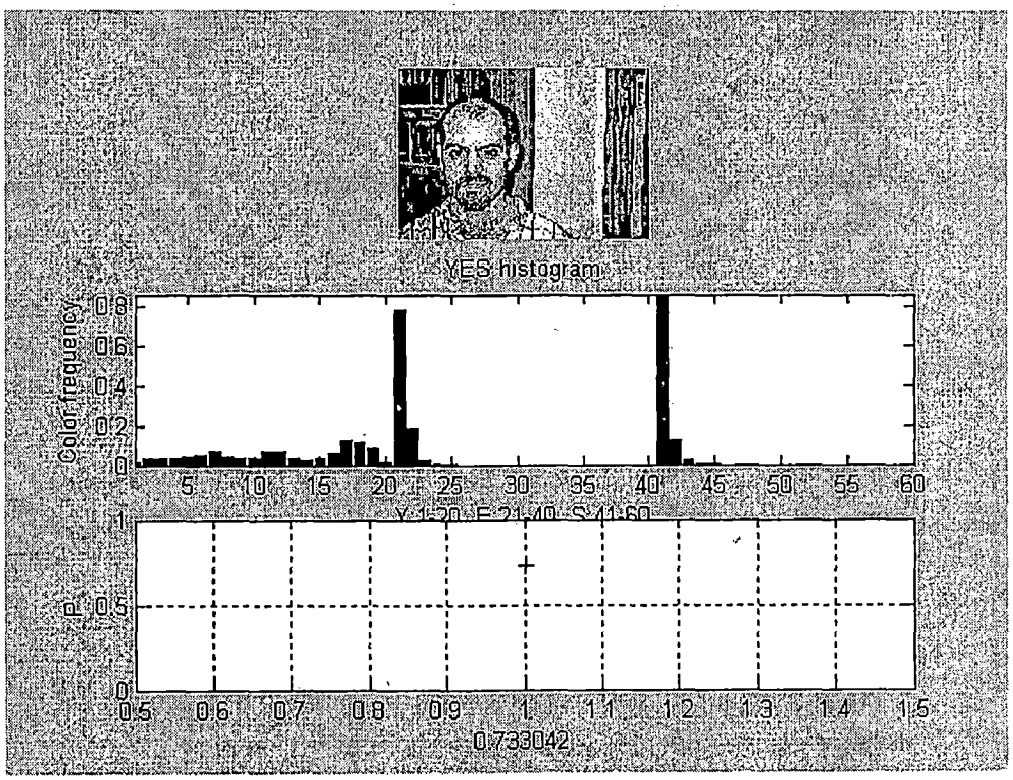
(a)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
P
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.335219

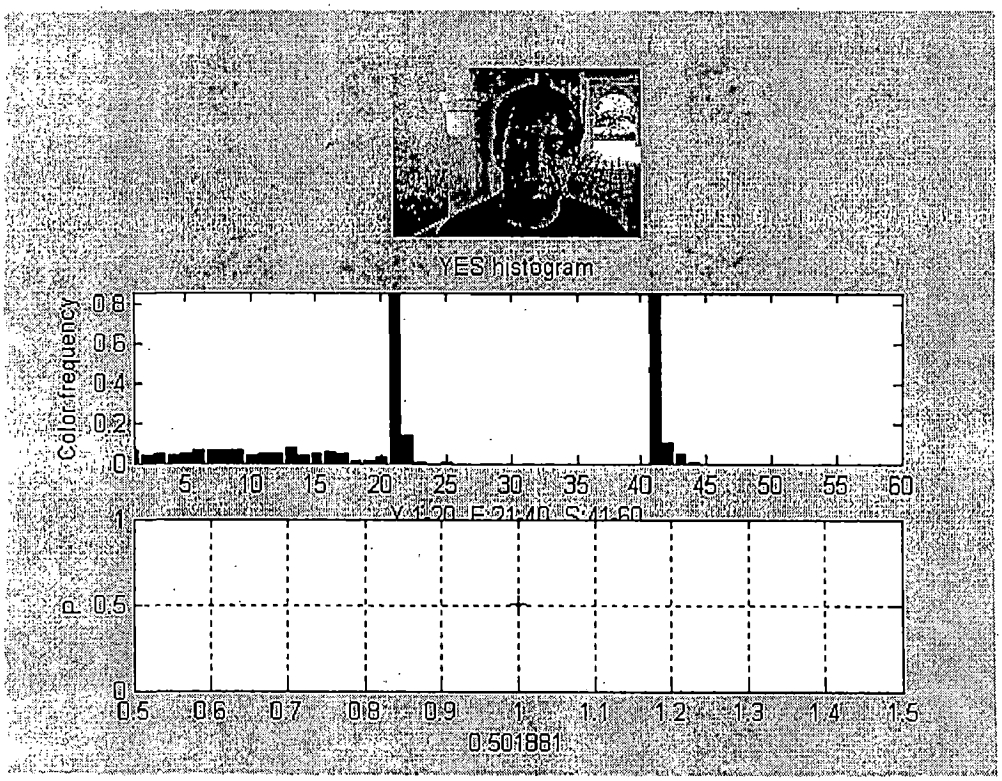
(b)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.608004

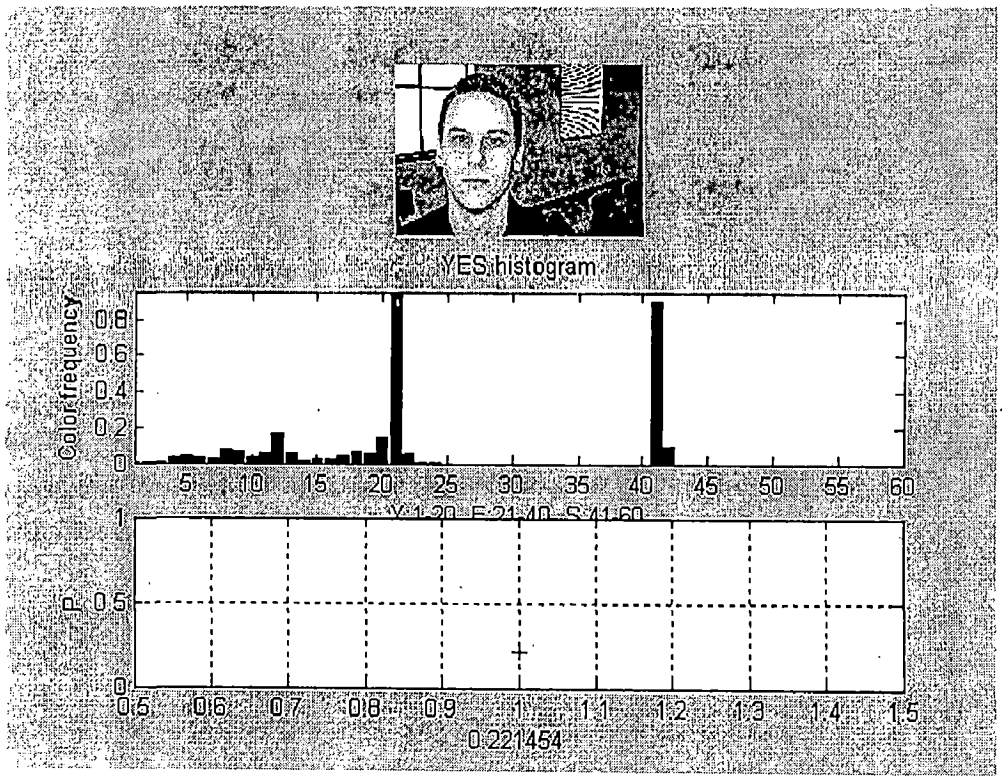
(c)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.733042

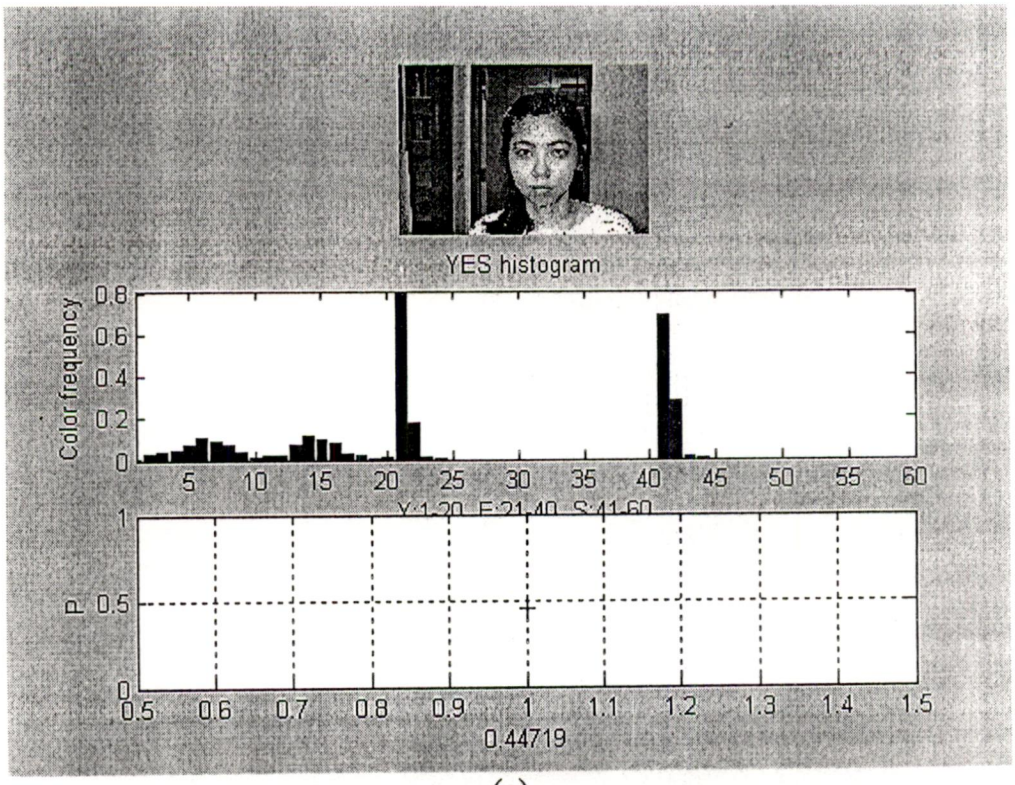
(d)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
P 0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.501881

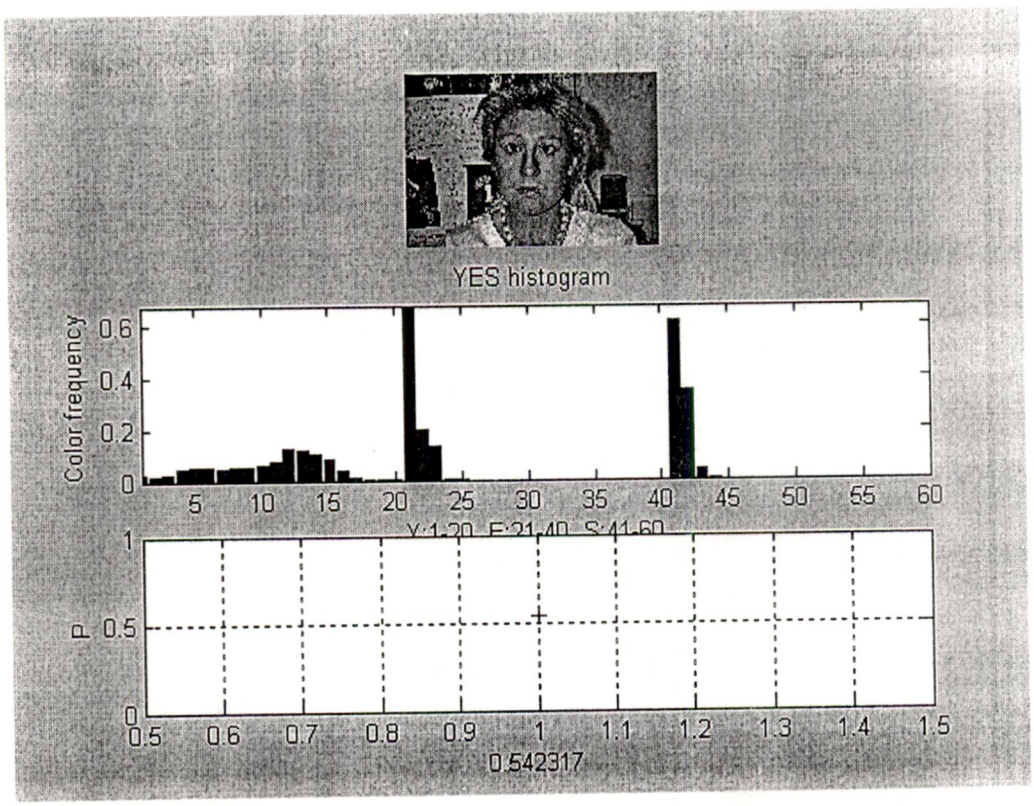
(e)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
P 0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.221464

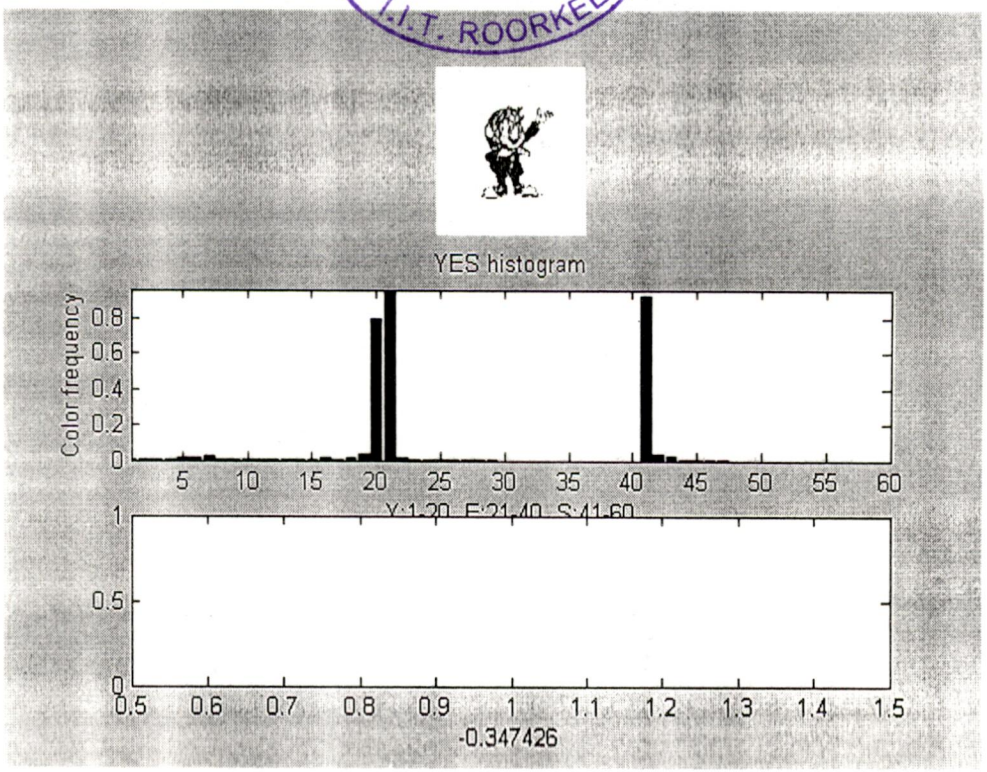
(f)



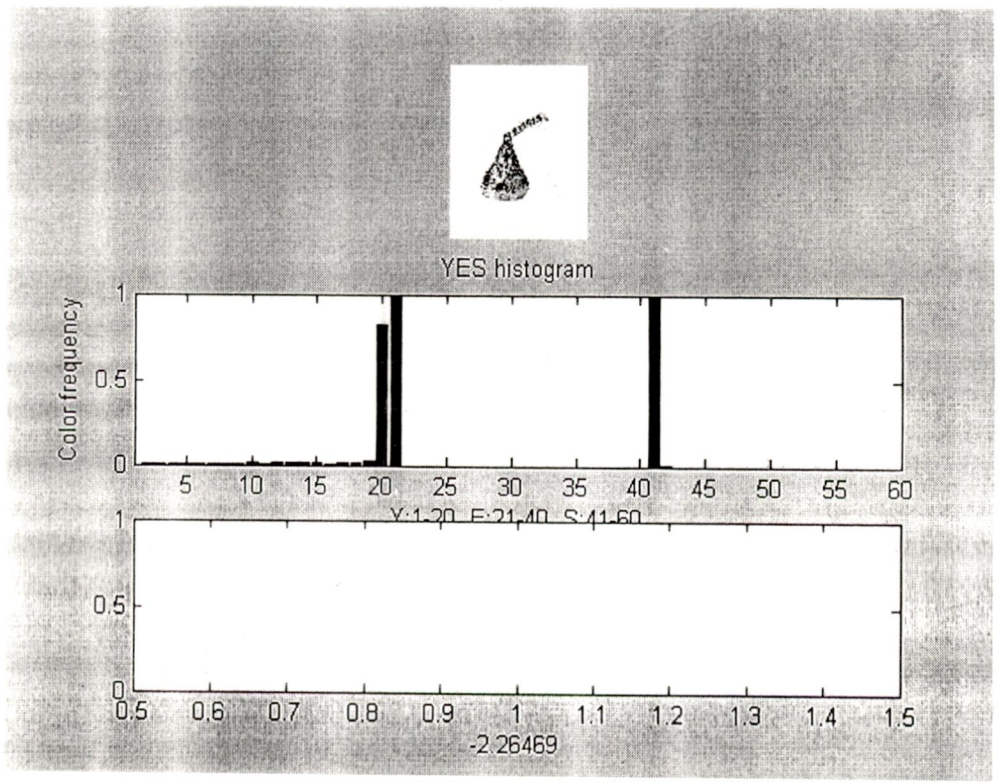
(g)



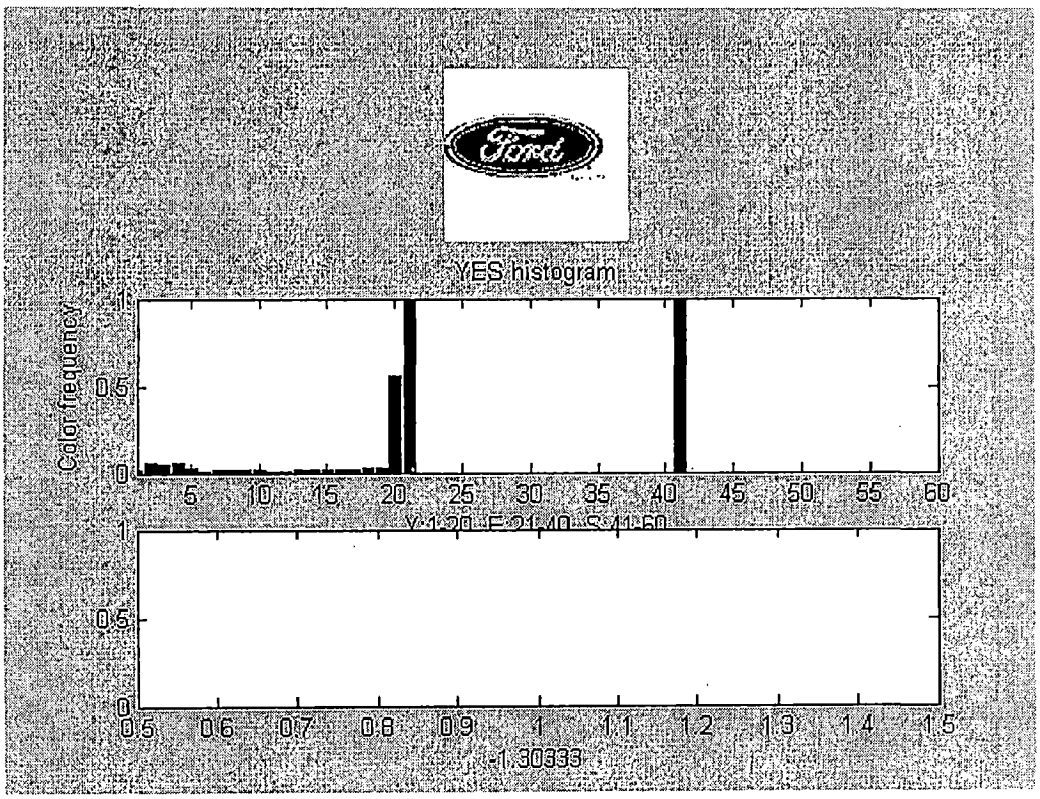
(h)



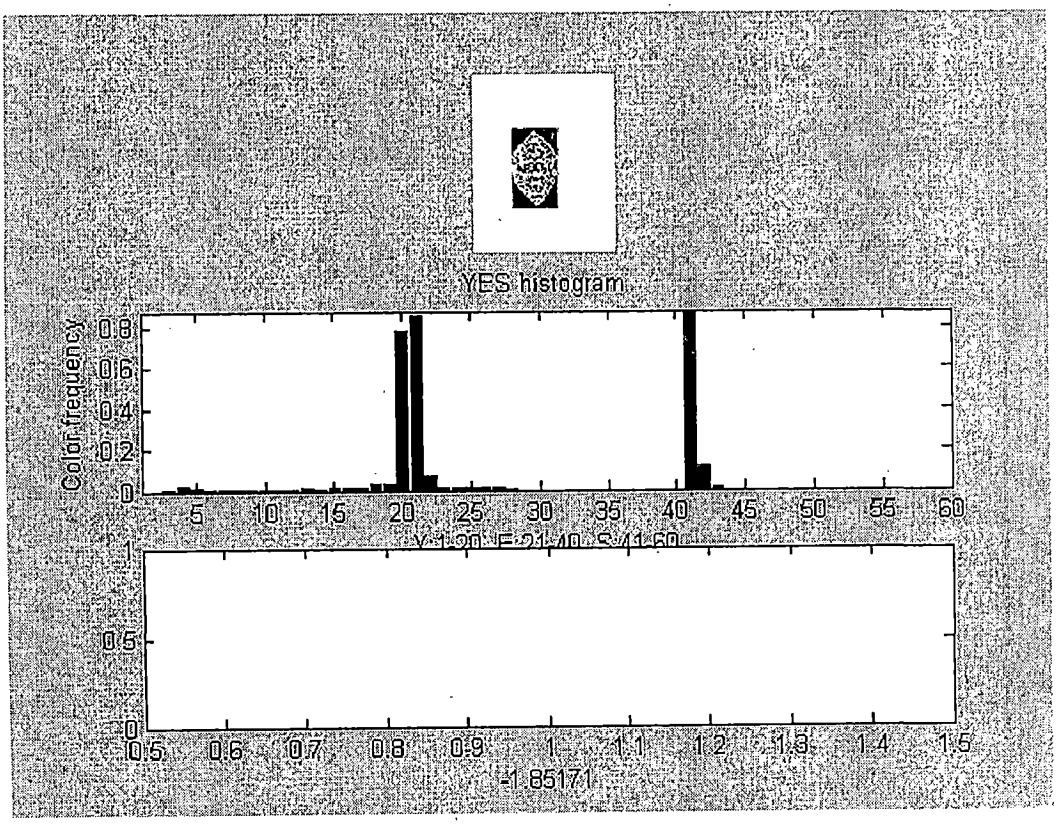
(i)



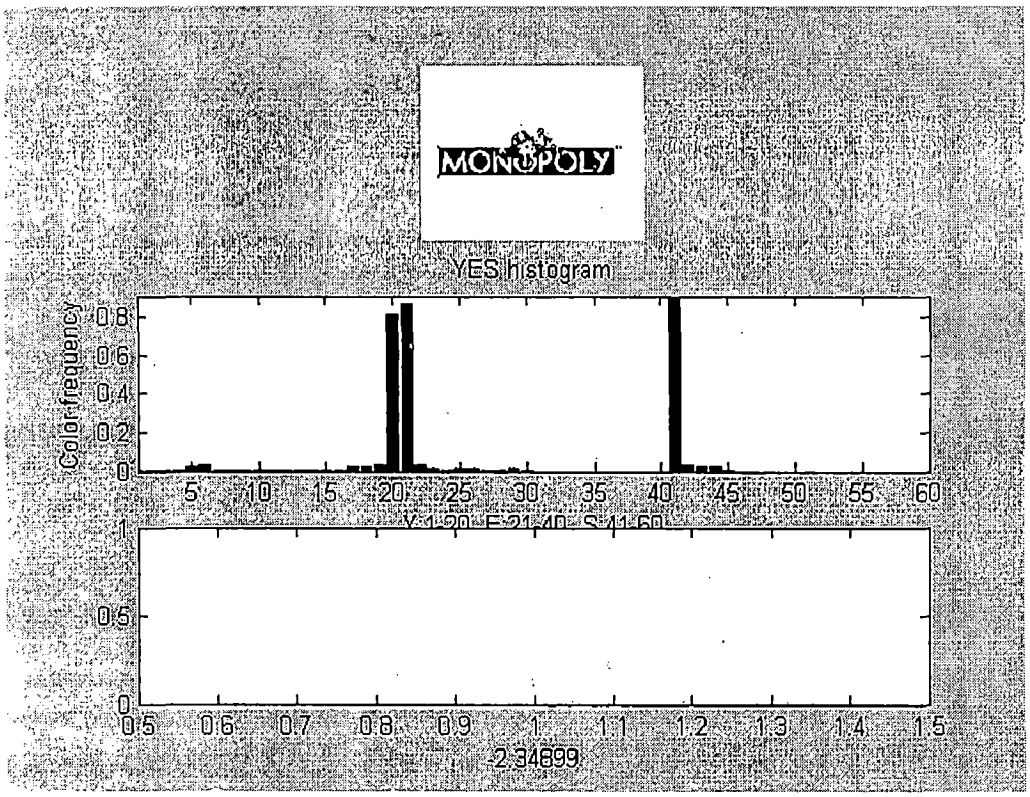
(j)


YES histogram
Color frequency
0
0.5
1
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
1.30333

(k)

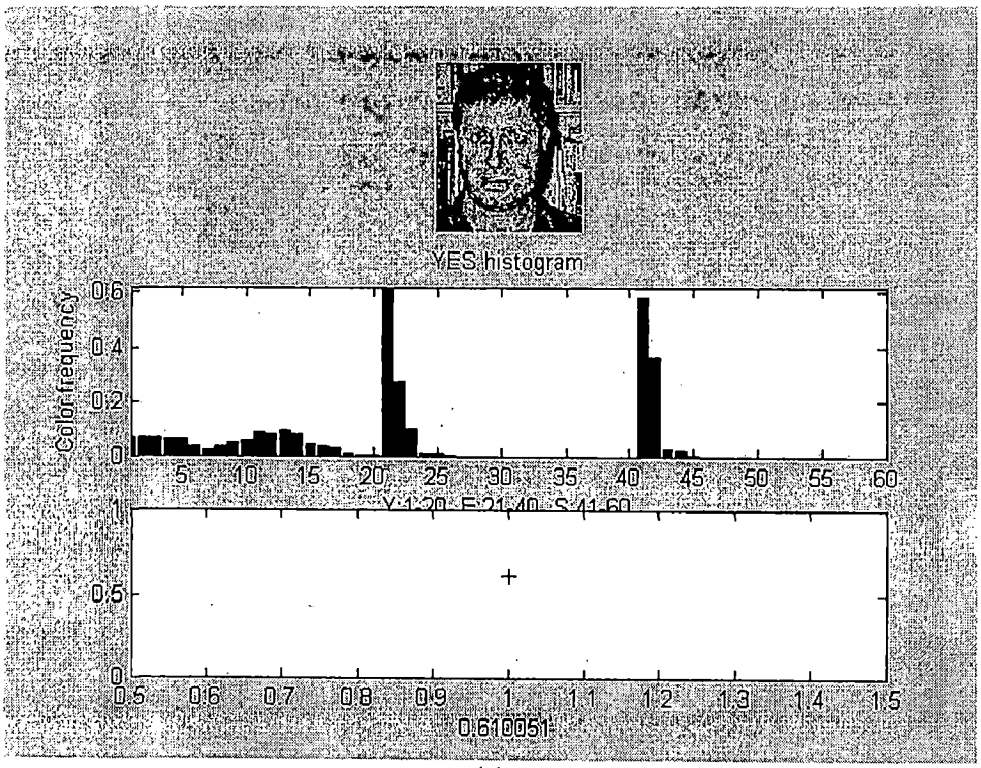

YES histogram
Color frequency
0
0.2
0.4
0.6
0.8
1
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
1.85171

(l)

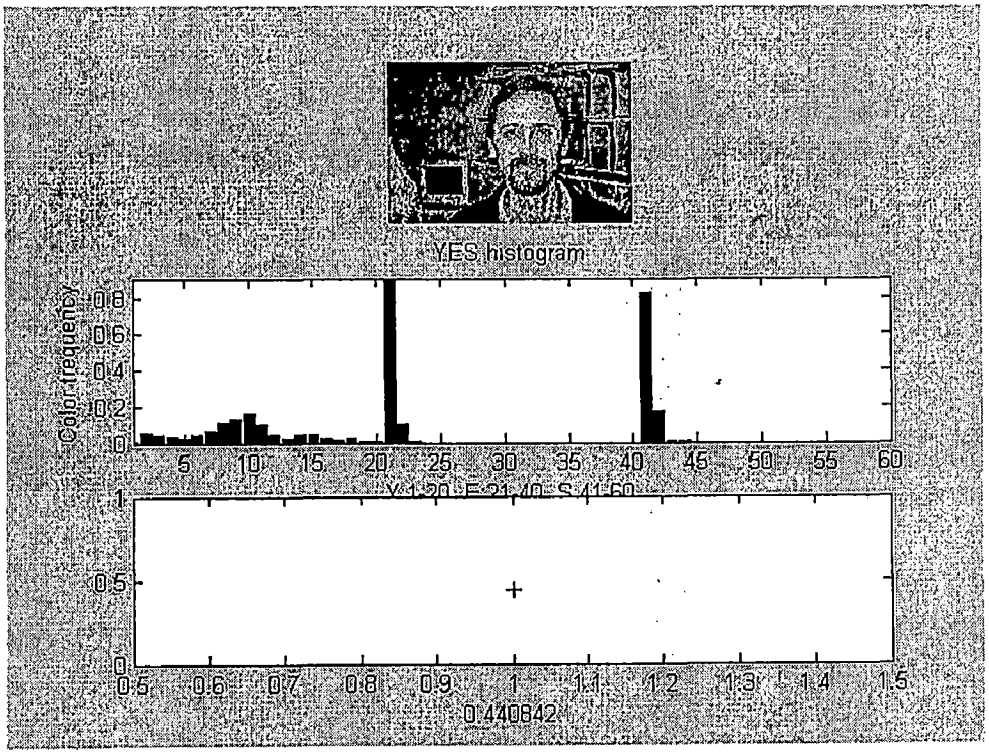


(m)

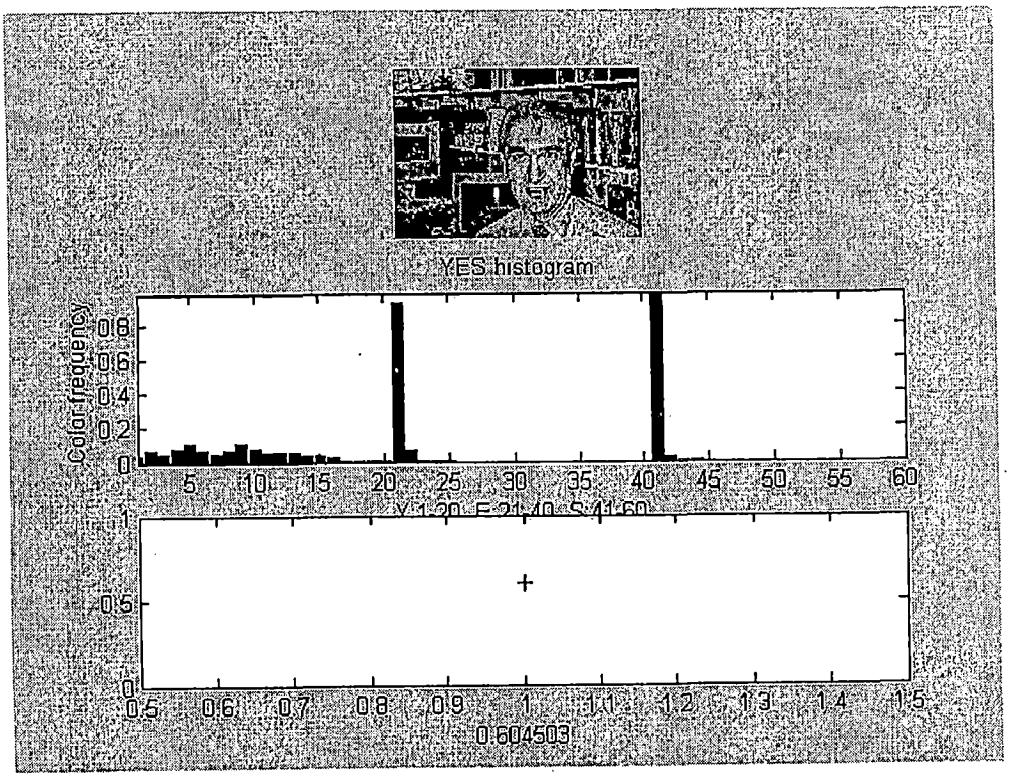
Fig 4.3 YES histogram with N=20 for training set 1



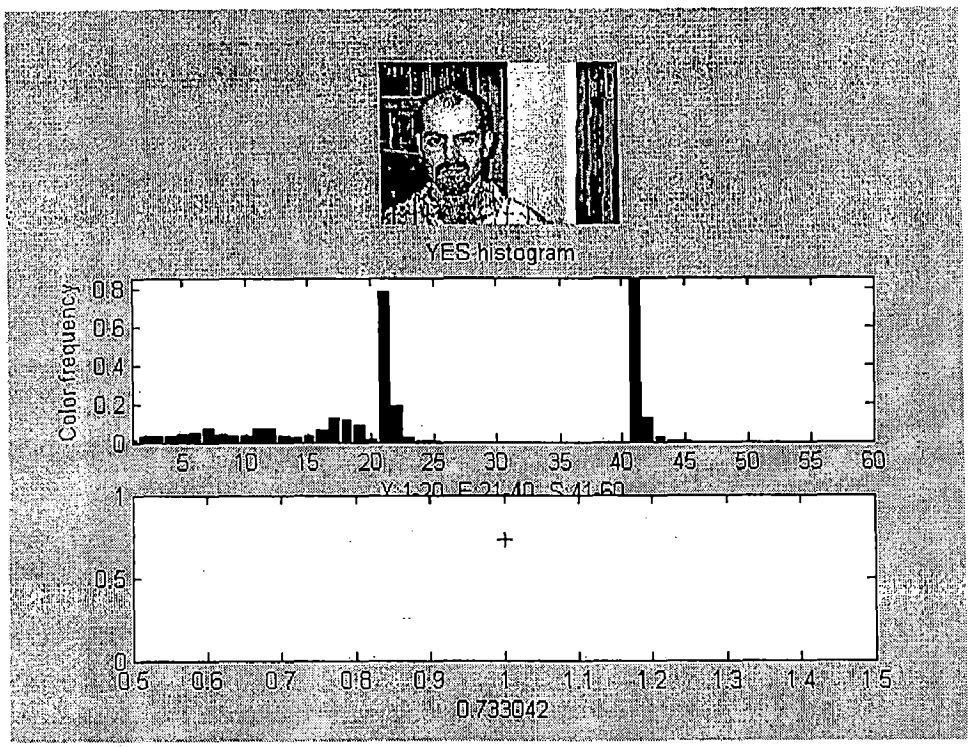
(a)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.440842

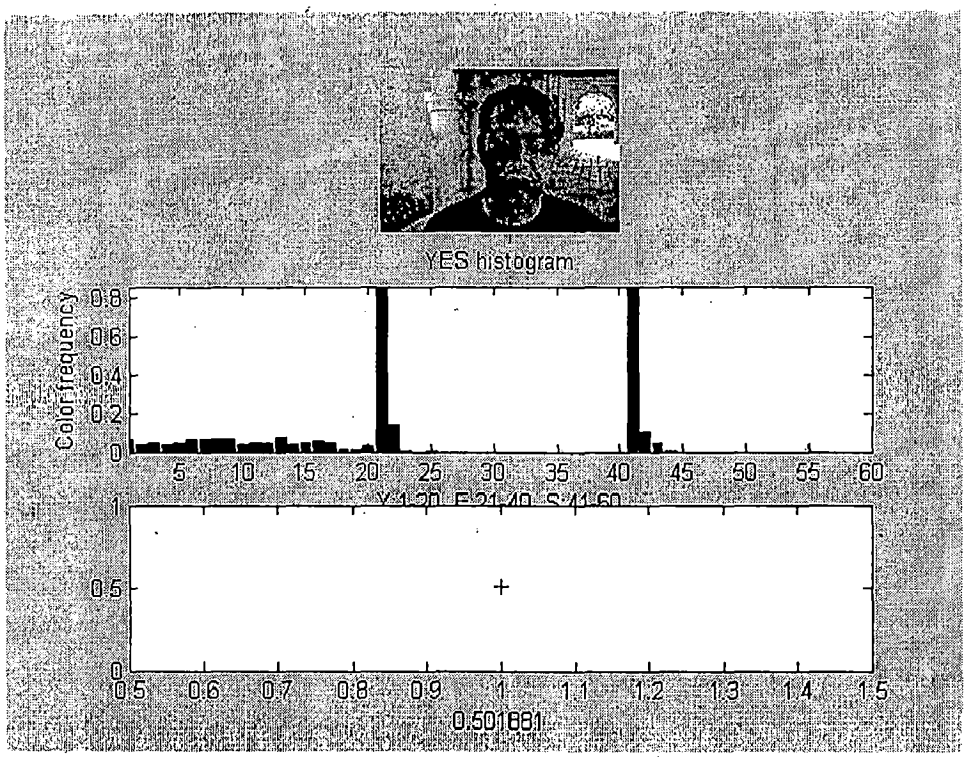
(b)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.604503

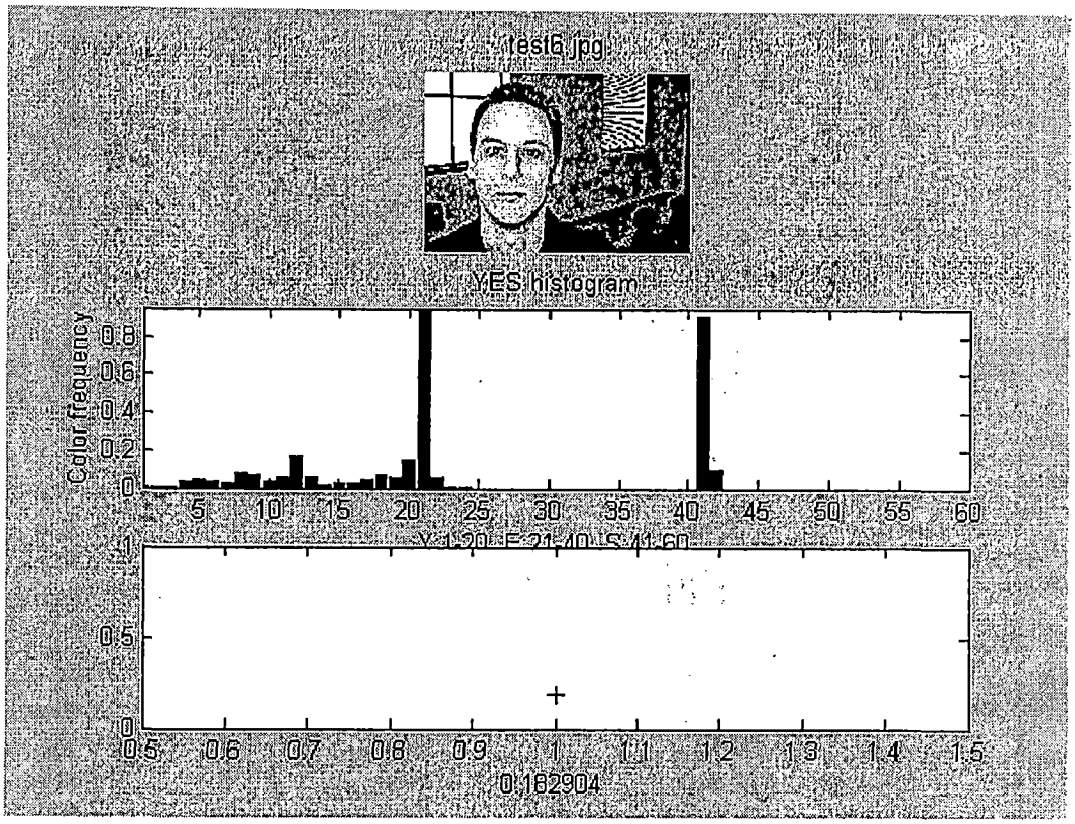
(c)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.733042

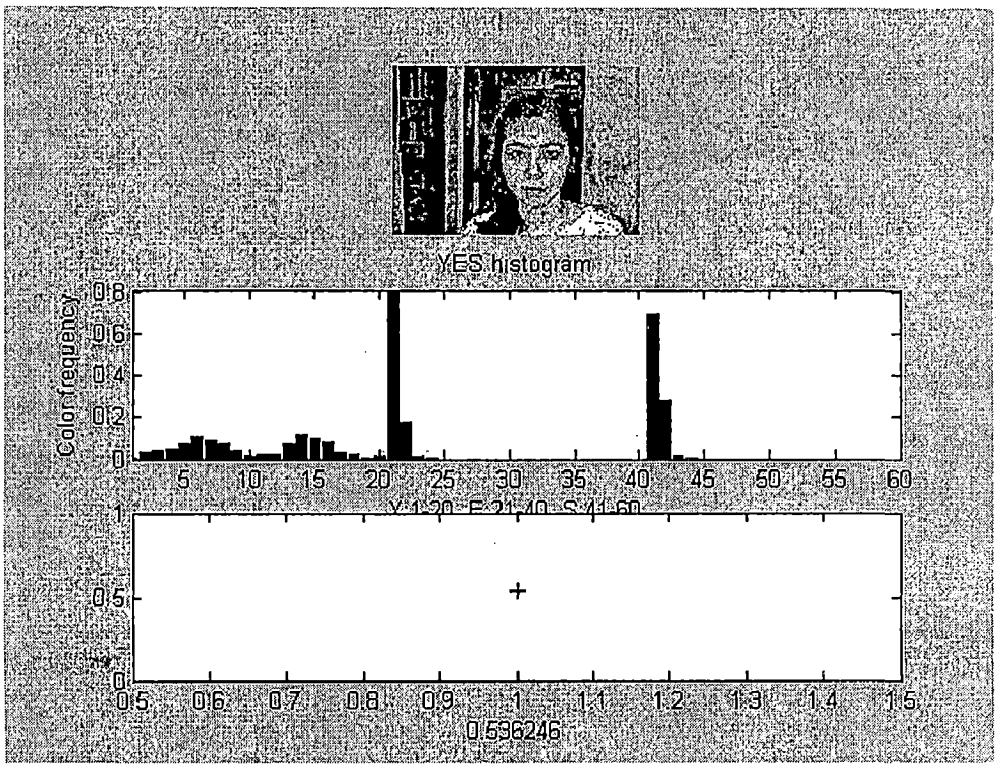
(d)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.501681

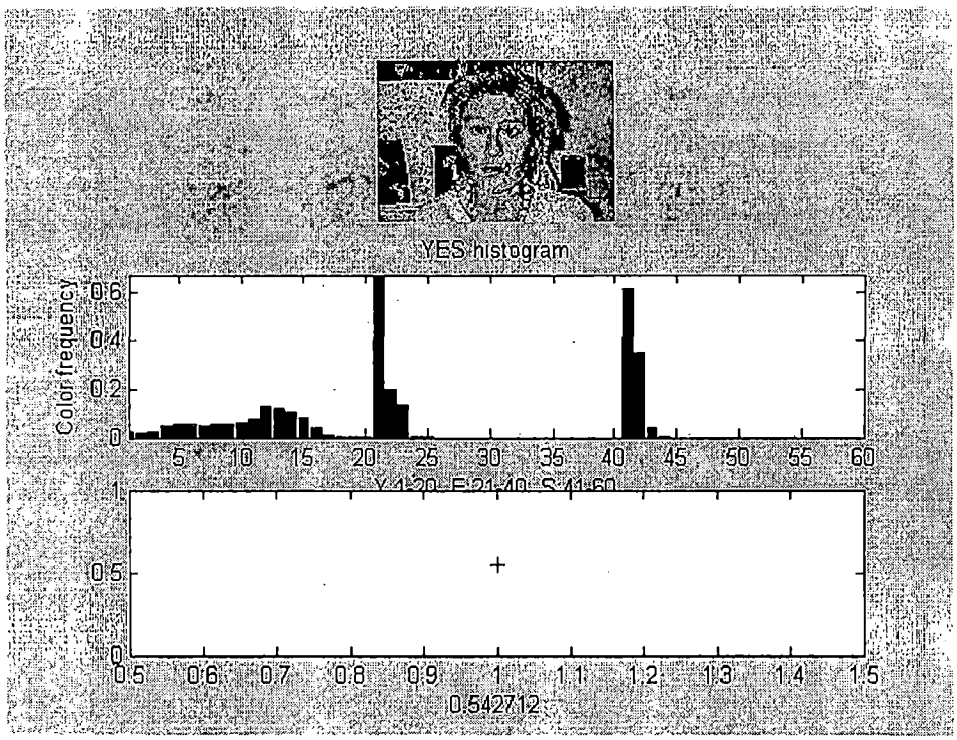
(e)


test6.jpg
YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.162904

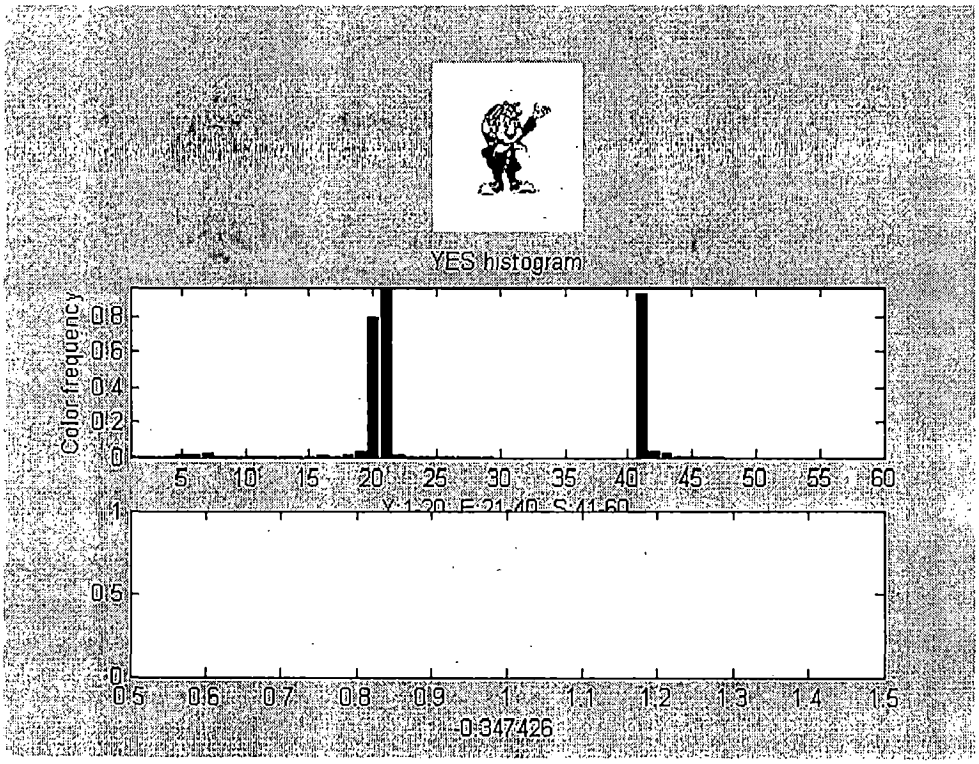
(f)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.536246

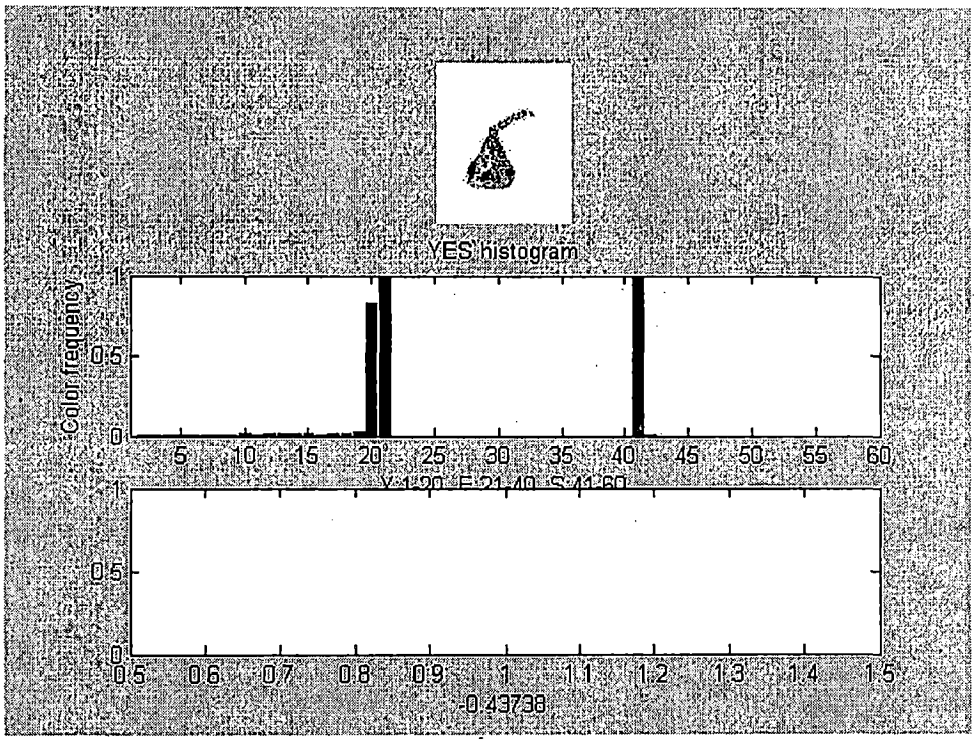
(g)


YES histogram
Color frequency
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.542712

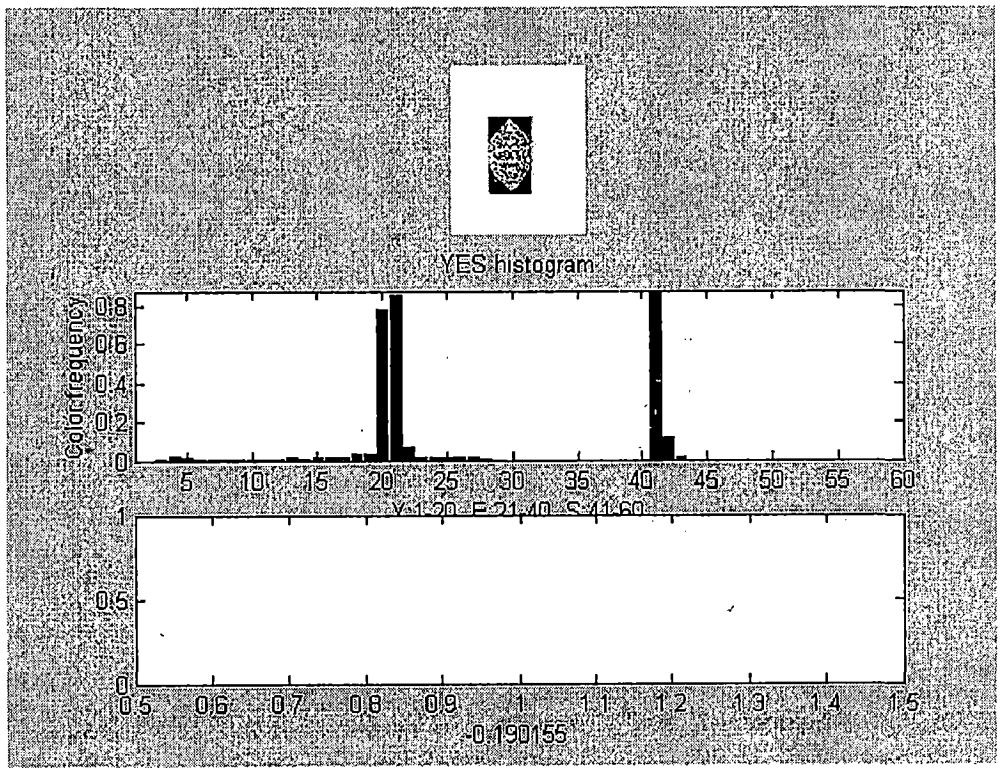
(h)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
0.347426

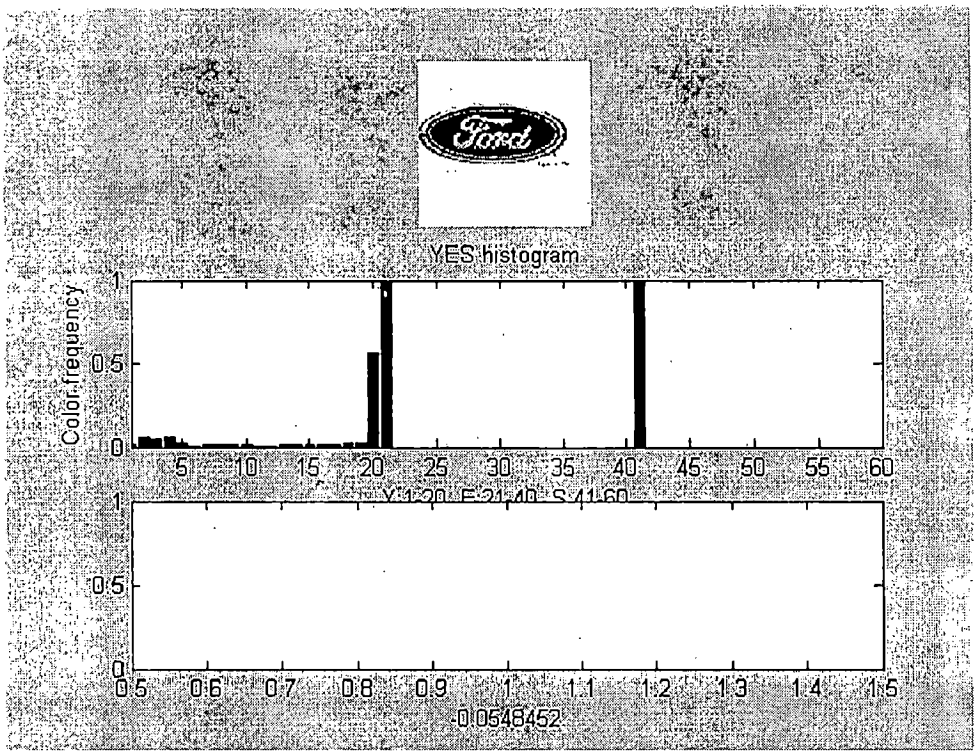
(i)


YES histogram
Color frequency
1
0.5
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
-0.43738

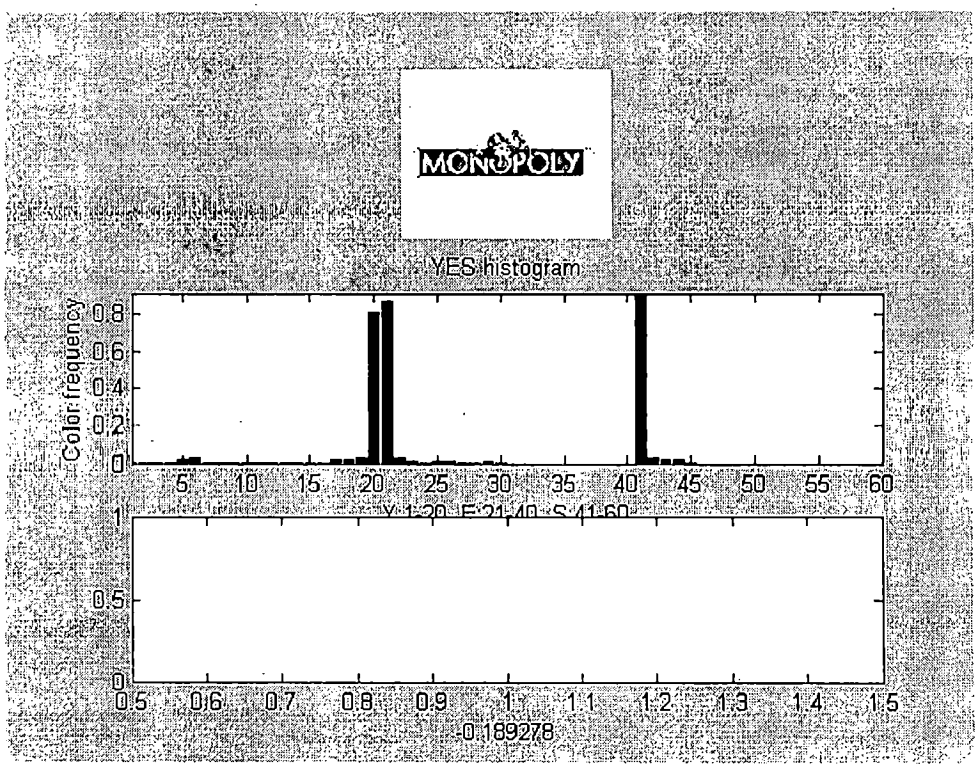
(j)


YES histogram
Color frequency
0.8
0.6
0.4
0.2
0
5 10 15 20 25 30 35 40 45 50 55 60
1
0.5
0
0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5
-0.190155

(k)



(l)



(m)

Fig. 4.4 YES Histogram approach with N=20 for training set 2

4.1.1 Network outputs

Table 4.1 Network outputs for training set 1

Image index	Actual P (output)	RGB Approach Output P	YES Approach Output P
1	1	0.619	0.593
2	1	0.558	0.792
3	1	0.605	0.937
4	1	0.635	0.753
5	1	0.580	0.755
6	1	0.569	0.680
7	1	0.612	0.905
8	1	0.643	0.800
9	1	0.621	0.490
10	1	0.491	0.682
11	0	0.383	-0.081
12	0	0.219	0.499
13	0	0.352	0.301
14	0	0.388	-0.083
15	0	0.356	-0.280
16	0	0.345	0.207
17	0	0.360	0.441
18	0	0.598	0.135
19	0	0.570	-0.095
20	0	0.613	0.031

Table 4.2 Network outputs for training set 2

Image index	Actual P (output)	RGB Approach Output P	YES Approach Output P
1	1	0.615	0.555
2	1	0.616	0.563
3	1	0.564	0.595
4	1	0.541	0.542
5	1	0.614	0.569
6	1	0.612	0.596
7	1	0.591	0.585
8	1	0.268	0.566
9	1	0.395	0.599
10	1	0.336	0.565
11	0	0.415	0.536
12	0	0.225	0.515
13	0	0.395	0.556
14	0	0.415	0.568
15	0	0.226	0.566
16	0	0.441	0.317
17	0	0.356	0.458
18	0	0.425	0.456
19	0	0.312	0.298
20	0	0.381	0.299

Table 4.3 Network outputs for test images

		Training set 1	Training set 2	Training set 1	Training set 2
1	1	0.600	0.568	0.610	0.611
2	1	0.576	0.538	0.335	0.440
3	1	0.587	0.511	0.608	0.604
4	1	0.727	0.634	0.733	0.733
5	1	0.602	0.565	0.501	0.501
6	1	0.422	0.573	0.221	0.182
7	1	0.537	0.537	0.447	0.536
8	1	0.635	0.635	0.542	0.542
9	<0.5	-2.262	-2.141	-2.094	-0.347
10	<0.5	-2.451	-2.422	-2.264	-0.437
11	<0.5	-1.006	-0.927	-1.303	-0.054
12	<0.5	-2.207	-2.095	-1.851	-0.190
13	<0.5	-2.320	-2.249	-2.348	-0.189

4.2 Appearance Based Technique for Face Detection

In Appearance based technique gray scale test images with frontal view faces is taken for face detection purpose, as shown in Fig. 4.5 first bounding boxes are formed around the images and then clear output is output is obtained in form of box placed around the face (Fig.4.6). The output is obtained by training of 4000 face images and 4000 non face images and training time is very less. Various interpretations drawn from the result output images are

- 4000 face and 4000 non-face images are taken for training purpose and time taken to complete the training process for 4000 images is only few seconds.
- Very good results is obtained by putting box around face in a gray scale frontal view face photograph, shows that training process is very efficient.
- It is applicable to only gray scale images and frontal view faces.
- Results obtained shows that give better result only for single face in an image and not for more than one face in an image.
- Results are tried on two types of test images, shows good results for gray scale jpeg format image and inaccurate result for png format image.

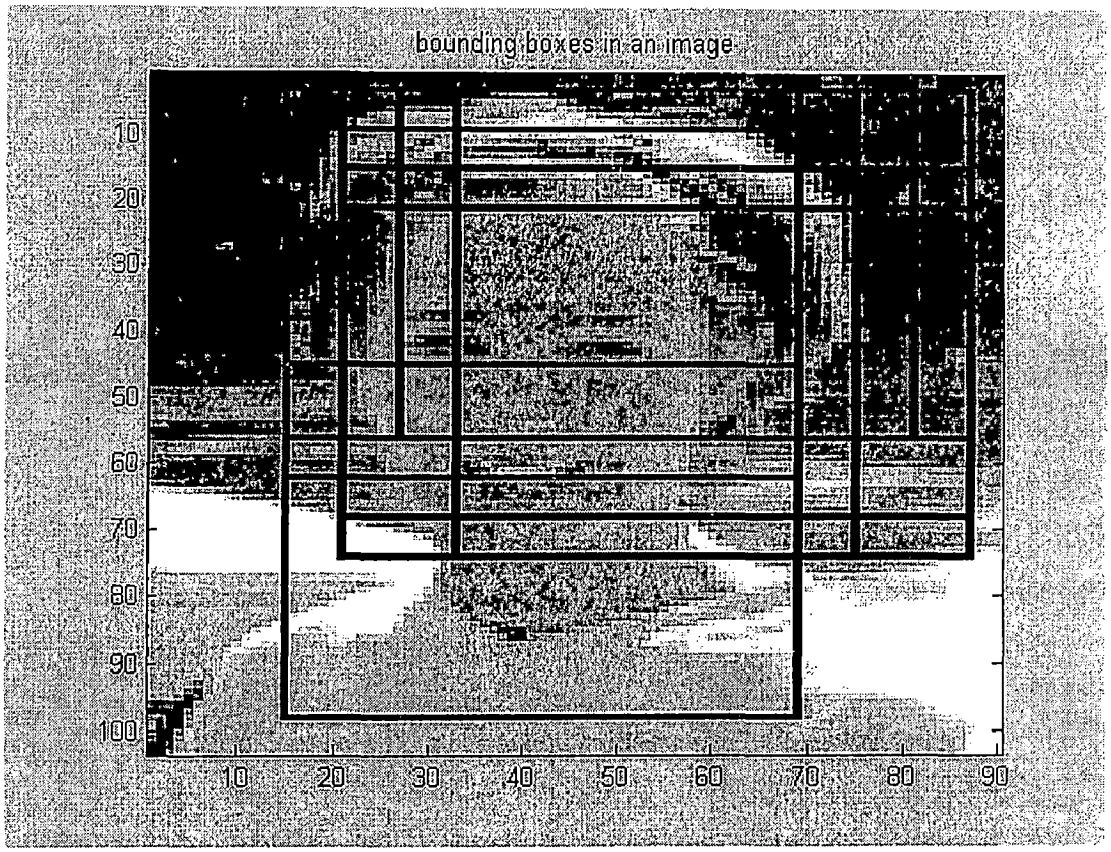


Fig. 4.5 Bounding boxes in test image 1

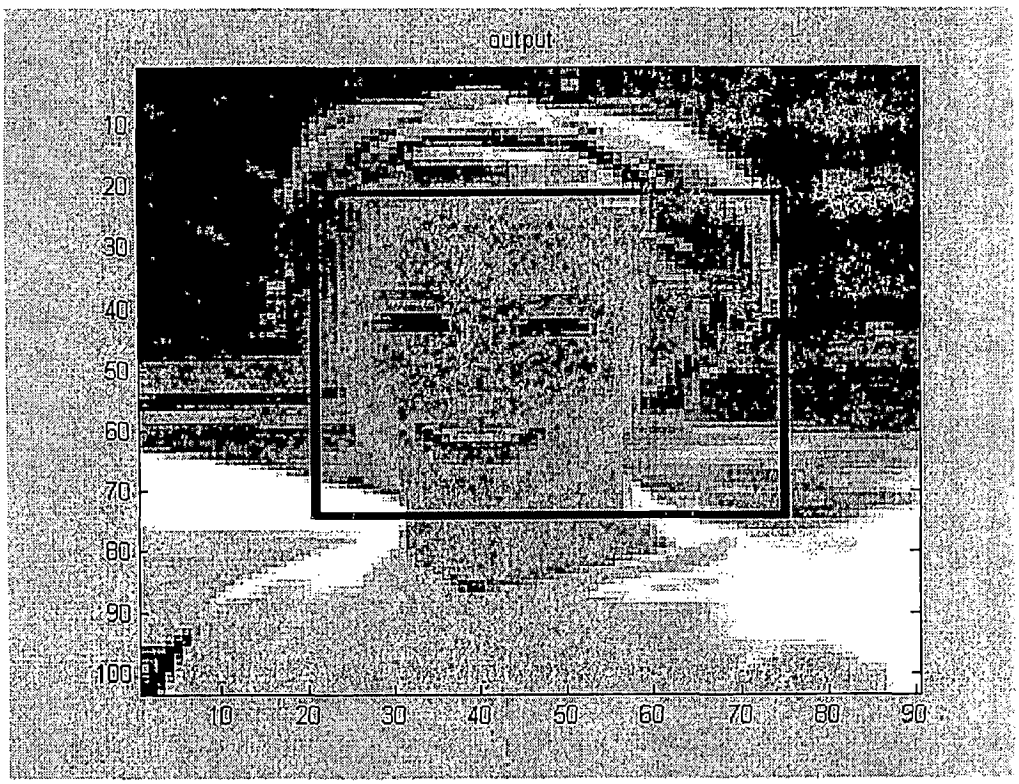


Fig. 4.6 Output of Test image1

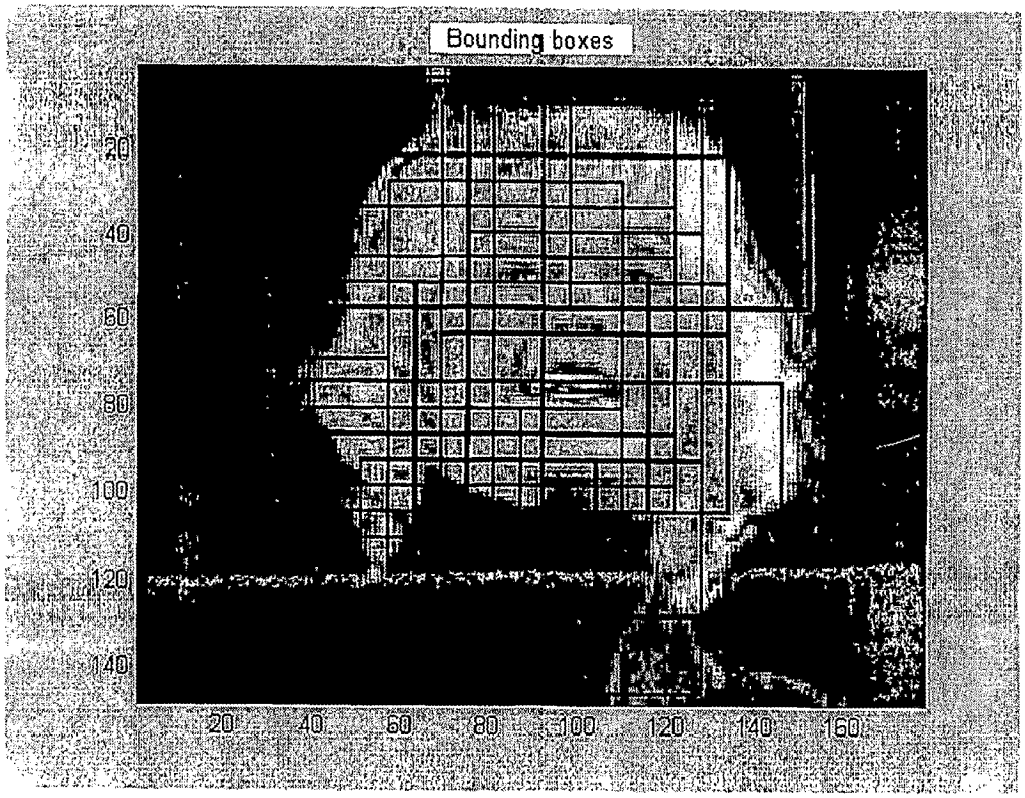


Fig. 4.7 Bounding Boxes around Test Image 2

Fig.4.8 Output of Test Image2

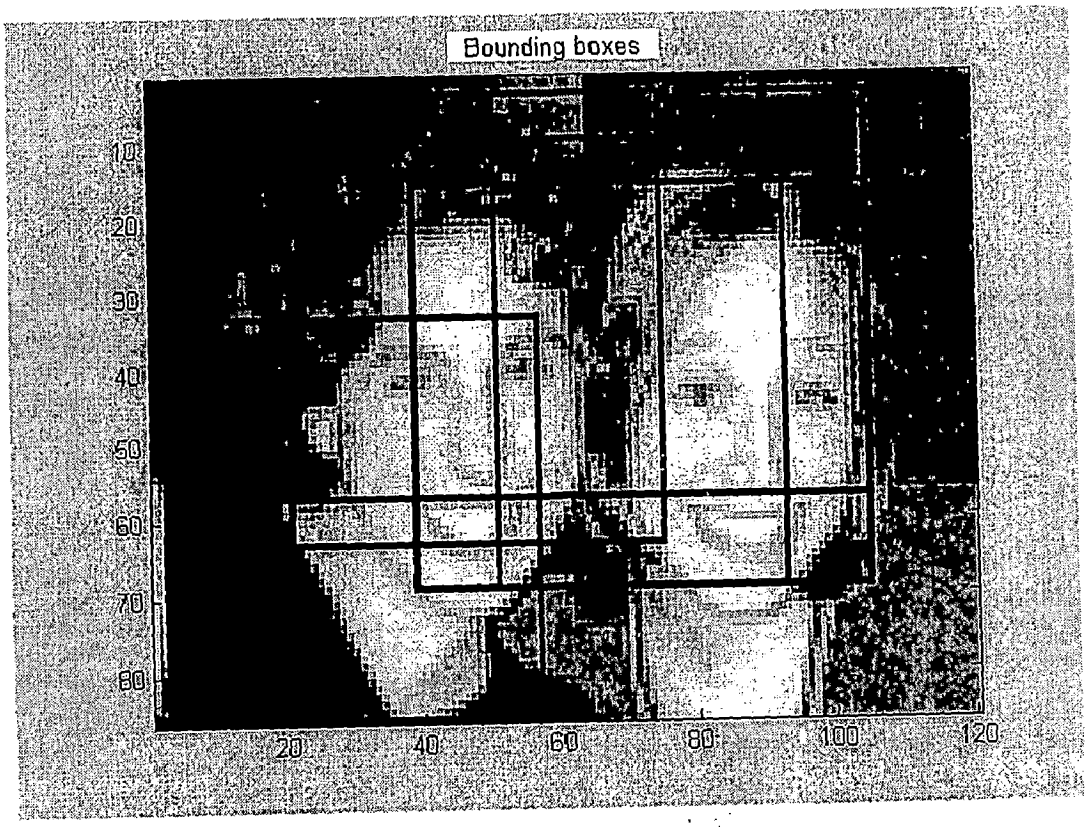


Fig. 4.9 Bounding boxes in test image3

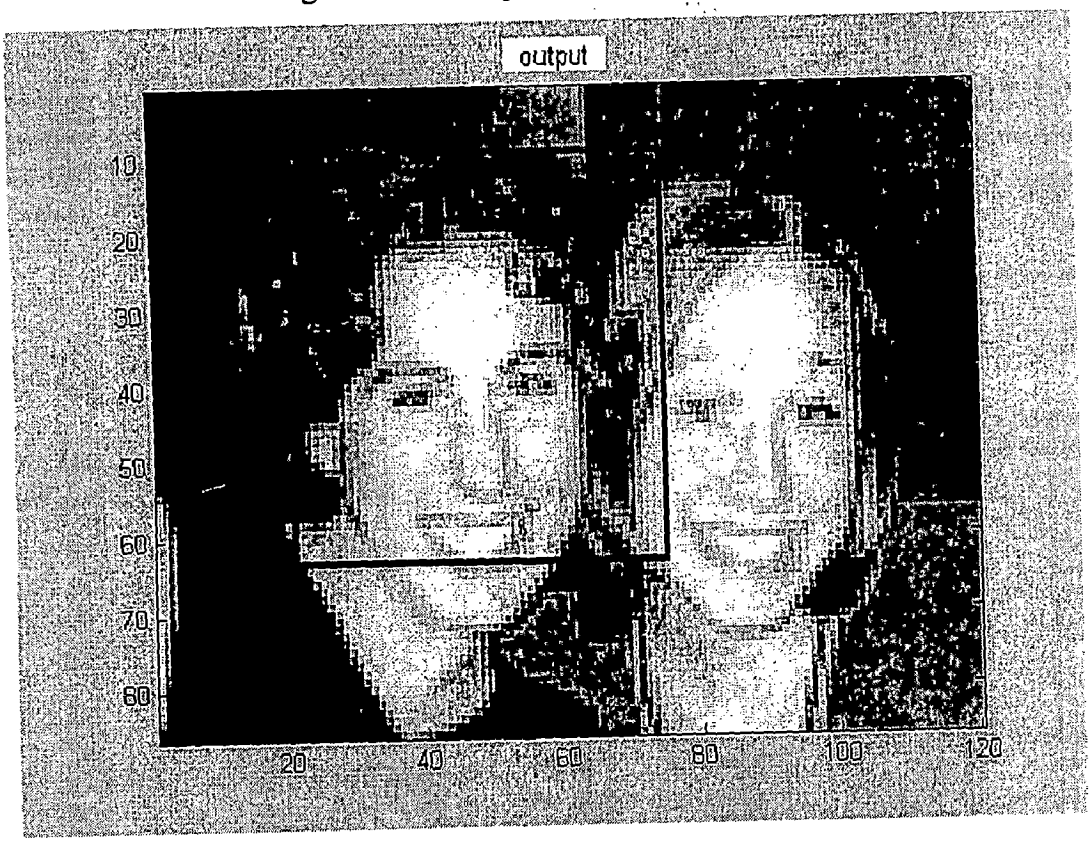


Fig. 4.10 Output of Test image 3

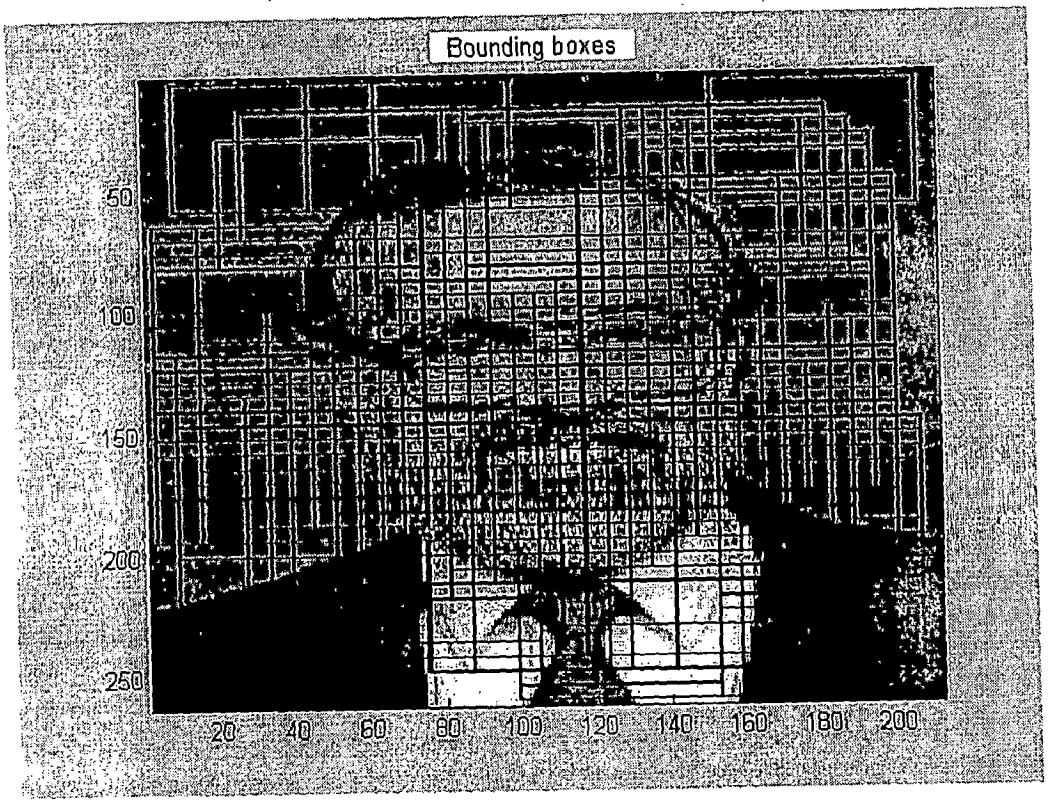


Fig. 4.11 Bounding boxes in Test Image 4

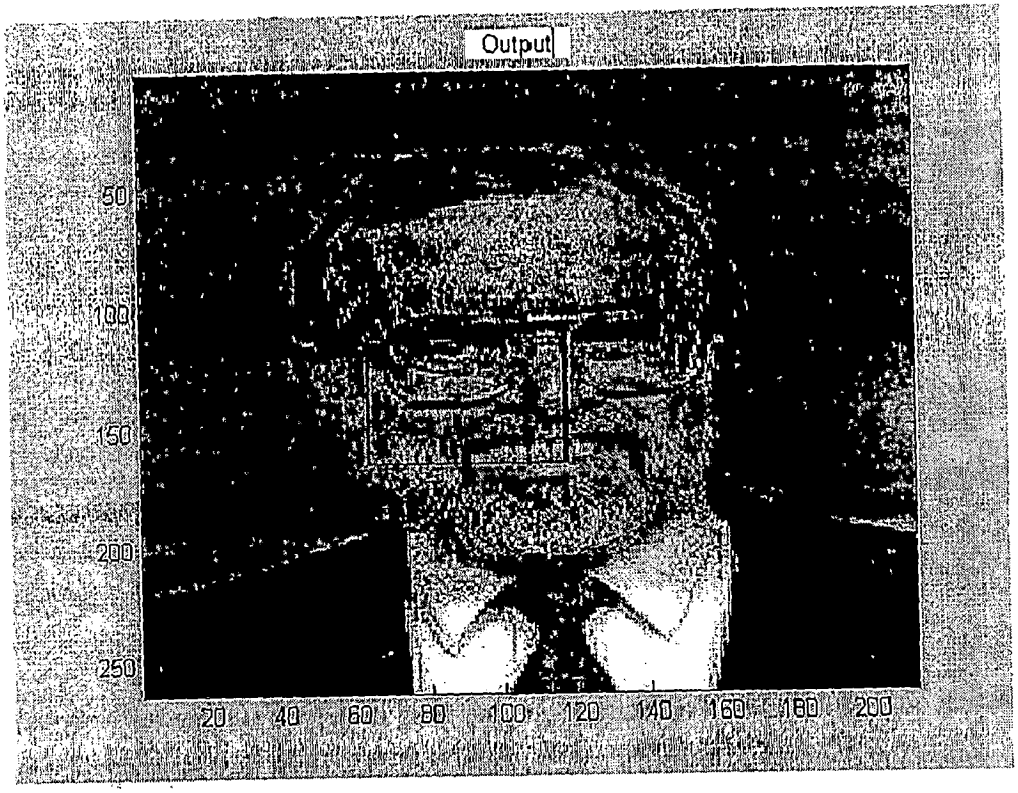


Fig. 4.12 Output of Test Image 4

4.3 Performance Parameters

- Time for training images.
- Complexity in terms of programming and implementation.
- Type of test images.
- Types of training images.
- Accuracy of result for same test images.

4.4 Comparison of Two Techniques

- Time for training is less in case of PCA training, it takes few seconds for 4000 face and 4000 non-face images. While in case of LM training it takes 1 min for 10 face and 10 non-face images.
- Complexity is large in case of appearance based technique as compared to NN color based technique and it increases as type of test image changes and number of training images increases.
- Appearance based technique is applicable only to gray scale images as it is based on feature extraction and classification, while NN technique is color based and trained for skin tone detection.
- Both technique give good results for frontal view and single faces in an image as number of faces increases and pose changes complexity of system increases rapidly.
- In color based technique, true color, jpeg format images are used for training while in appearance based technique gray scale image with any format can be used for training.
- For single and frontal view faces in an image accuracy of appearance based technique is more than color based technique.
- Output obtained in case of appearance based technique is simply a box placed around face but in case of color based technique output is obtained in form data and large calculation is required for performance evaluation.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusions:

In this dissertation color based technique using neural network [8] has been modified and implemented with new inputs and training sets using MATLAB 6.5. Further a new combination of PCA for FE and K-NN for classification has been used in appearance based technique. It is found that the result with this combination gives better face detection as compared to other combinations [16]. Other important conclusions regarding the two techniques implemented are as follows:

- The color based technique using NN is successful in detecting almost 80% of faces in color test images calculated by performance evaluation measurement.
- The color-based technique is applicable to colored, real images with frontal view faces.
- It is observed that PCA algorithm takes very less time to train thousands of face and non-face images and give very good result for gray scale, frontal view face images.
- In terms of time and accuracy appearance based technique gives very good results but in terms of complexity in implementation and variation in test images color based technique is better.

5.2 Future Scope

Since the effectiveness of the color based technique appears to be dependent on the color space, one area for further research would be to test other color spaces like HSV, LIQ etc Also, it would be interesting to explore the systems' dependence on variable parameters. Increasing the value of N, number of hidden nodes and training set would provide the neural network with more input information and and this will improve the performance of detection.

In appearance based technique, in place of PCA + K-NN combination any other combination like PCA + SVM, LDA + SVM can be used to increase the performance of face detection system at the expense of increase in complexity of implementation.

REFERENCES

[1] K.C. Yow and R. Cipolla. Feature-based human face detection. Image and Vision Computing, 15 (1997), pp. 713-735.

[2] T.F. Cootes and C.J. Taylor. Locating faces using statistical feature detectors. Proceeding of the Second International Conference on Automatic Face and Gesture Recognition, 1996, pp. 640-645.

[3] T.K. Leung, M.C. Burl, and P. Perona. Finding faces in cluttered scenes using random labeled graph matching. Proceedings of the Fifth International Conference on Computer Vision, 1995, pp. 637-644.

[4] H.A. Rowley, S. Bluja, and T. Kanade. Neural network-based face detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (1) (1998), pp. 23-38.

[5] [sung94examplebased] – Example-based learning for Human Face Detection. Kah-Kay Sung, Tomaso Poggio. MIT. IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 20, number 1, pages 39-51, January 1998.

[6] M. De Mariscoi, L. Cinque, and S. Levialdi. Indexing pictorial documents by their content: a survey of current techniques. *Image and Vision Computing*, 15 (1997)

[7] D. Androutsos, K.N. Plataniotois, and A.N. Venetsanopoulos. A novel vector-based approach to color image retrieval using a vector angular-based distance measure. Computer Vision and Image Understanding, 75 (1/2) 1999.

[8] Todd Wittman, Face Detection and Neural Networks, Department of Mathematics, University of Minnesota, December 2001.

[9] Haykin Simon, *Neural Networks - a Comprehensive Foundation*, Delhi, India: Pearson Education, Inc., 2001.

[10] Bow Sing-Tze, *Pattern Recognition and image pre-processing*, New York, Basel: Marcel Dekker, Inc., 2002.

[11] V. Rao and H. Rao. *C++ Neural Networks & Fuzzy Logic*. MIS Press: New York, 1995.

[12] J. Cai and A. Goshtasby. Detecting human faces in color images. Image and Vision Computing, 18 (1999), pp. 63-75.

[13] W. Zhao, A. Krishnaswamy, R. Chellappa, D.L. Swets, J. Weng. "Discriminant Analysis of Principal Components for Face Recognition", in Face Recognition: From Theory to Applications, H. Wechsler, P.J. Phillips, V. Bruce, F. Fogelman Soulie, T.S. Huang (Eds.). Springer-Verlag, pp. 73-85, 1998

[14] Prof. Jim Rehg, "Face Detection and Recognition Using PCA and PPCA" CS 7636 Computational Perception, Georgia university of Technology .

[15] Do-Joon Jung, Chang-Woo Lee, Yeon-Chul Lee, Sang-Yong Bak, Jong-Bae Kim1; Hyun Kang and Hang-Joon Kim, "PCA based real time face detection and tracking" Department of Computer Engineering, Kyungpook National University ,2002.

[16] Yi-Ting Chou, "Toward Face Detection, Pose Estimation and Human Recognition from Hyperspectral Imagery" Automated Learning Group National Center for Supercomputing Applications, 2003.

[17] Tae-Kyun Kim1,2, Hyunwoo Kim1, Wonjun Hwang1, Seok-Cheol Kee1 and Josef Kittler2, "Face description based on decomposition and combining of a facial space with LDA", University of Surrey, U.K, 2003.

[18] Tae-Kyun Kim, Sing UK Lee, Jong Ha Lee, San -Ryong Kim, " Integrated approach of multiple face detection for video surveillance" 16th international conference on pattern recognition, 2002.

[19] Shou-Der Wei and Shang-Hong Lai, "Robust Face Recognition under Lighting Variations" Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, 2003.

[20] Daniel nicorici, Sampsa hautaniemi, hakan öktem, olli yli-harja, jaakko astola, "Inferring of gene regulatory networks from Expression data using knn classifier" 1tampere international center for signal processing, Tampere university of Technology, 2002.

[21] L. Sirovich. Turbulence and the dynamics of coherent structures. Quarterly Applied. Mathematics, 45(3):561–590, 1987

[22] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society series

B, 39:1–38, 1977.

[23] Zoubin Ghahramani and Michael I. Jordan. Supervised learning from incomplete data via an EM approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, Advances in Neural Information Processing Systems, volume 6, pages 120–127. Morgan Kaufmann, 1994.

[24] Computer vision test images, Website "www-2.cs.cmu.edu/~cil/v-images.html".

APPENDICES

Program Listing of Neural Network Based Technique

Training

The following m-files were used to train the neural network. The Levenberg-Marquad code and the related files eone.m, etot.m were written by Dr. Fadil Santosa. The trainall routine takes about 5 minutes to run, but it only runs 10 iterations at a time.

Levenberg-Marquad Code :

1. **levmar**

```
function [w, res] = levmar (X, T, p, w0, lam, itMAX)
% levmar -- Levenberg-Marquad Algorithm
% Input
% (X,T) are nodes to go through
% p is number of nodes in hidden layer
% w0 is initialization for weights
% lam is initizialization for learning rate lambda
% itMAX is number of iterations

% Output
% w is the final weights matrix
% res is the list of error at each iteration


% Compute number of unknowns: nunk.
[m, nex] = size(X);
[n, nex] = size(T);
nunk = p*(m+1) + n*(p+1);

if w0 == 0
  w = 3*rand(nunk,1);    % Random weights.
else
  w = w0;
end;

% Levenberg-Marquad Algorithm
for it = 1:itMAX
   [Etot, Gtot, Jtot, R] = etot (X,T,p,w,1);
   An = Jtot' * Jtot + lam * eye (nunk, nunk);

   dw = An \ (Jtot' * reshape(R, n*nex, 1) );
   wtst = w - dw;
   [Etst, Gtot, Junk, R] = etot(X,T,p,wtst,0);

   if Etst <= Etot
     w = wtst;  lam = lam / 10;
   else
     lam = lam * 10;
   end

   res(it) = Etot;
```

```
end
```

2. etot

```
function [Etot,Gtot,Jtot,R]=etot(X,T,p,w,toggle)
% compute total misfit
%  Etot  total misfit
%  Gtot  gradient of total misfit wrt weight w
%  R     residual matrix, each column is residual vector
%  Jtot  jacobian of R wrt w
%  toggle == 1 compute jacobian
%
  [m,nex]=size(X);
  [n,nex]=size(T);
% choose number of nodes in hidden layer
  Etot=0; Gtot=zeros(size(w)); Jtot=[];
  for k=1:nex
    x=X(:,k); t=T(:,k);
    [E,G,y,J]=eone(x,t,p,w,toggle);
    Etot=Etot+E;
    Gtot=Gtot+G;
    Jtot=[Jtot;J];
    R(:,k)=y-t;
  end
```

3. econe

```
function [E,G,y,J] = eone(x,t,p,w,toggle)
% program to generate misfit and gradient of misfit given an
% input-output pair
%  x(1:m) vector corresponding to given input
%  t(1:n) vector corresponding to target output
%  p number of nodes in hidden layer
%  w(1:P) vector corresponding to weights -- P=p*(m+1)+n*(p+1)
%  y(1:n) vector corresponding to network output
%  E scalar for norm of (t-y)
%  G(1:P) gradient of E with respect to w
%  J(1:n,1:P) Jacobian of y with respect to w
%  toggle 1 -> compute Jacobian, 0 otherwise
  m=length(x);
  n=length(t);
  xp=[x;1];
  tem=w(1:p*(m+1));
  W1=reshape(tem,p,m+1);
  tem=w(p*(m+1)+1:p*(m+1)+n*(p+1));
  W2=reshape(tem,n,p+1);
  a=W1*xp;
  z=sigmoid(a);
  zp=[z;1];
  y=W2*zp;
  E=0.5*norm(t-y)^2;

% gradient with respect to W2
  del=y-t;
  tem=del*zp';
  g2=reshape(tem,n*(p+1),1);

% gradient with respect to W1
```

```
  q=W2(:,1:p)'*del;
  tem=(sigmoid(a).*q)*xp';
  g1=reshape(tem,p*(m+1),1);
  G=[g1;g2];
  J=[];
  if toggle ==1

% jacobian with respect to W2
    J2=[];
    for l=1:p+1
      J2=[J2 zp(l)*eye(n,n)];
    end

% jacobian with respect to W1
    J1=[];
    for l=1:m+1
      J1=[J1 W2(:,1:p).*(ones(n,1)*sigmoid(a'))*xp(l)];
    end
    J=[J1 J2];
  end
```

4. Sigmoid

```
function s=sigmoid(t)
s=1./(1+exp(-t));

% Alternative
% s=(exp(t)-exp(-t))./(exp(t)+exp(-t));
```

RGB HISTOGRAN APPROACH

1. rgb_trainall1

```
function [x, outputs] = rgb_trainall1(N,p, toggle)

% Trains all images in directory images.
% We give file names of all jpg images.
% Plots final residual of each image at end.
% Given # bins N and # hidden nodes p.
% If toggle = 0, then we start with random initial weights.
% Otherwise assumes inital weights w given in file rgb_weights unless toggle=0.

warning off;

%Training rate lambda.
lam = 0.1;

% # iterations.
itMAX = 10;

% Human faces: y=1
x(:,1) = image2rgbhist ('1.jpg', N);
y(1,1) = 1.0;

x(:,2) = image2rgbhist ('2.jpg', N);
y(1,2) = 1.0;
```

```
x(:,3) = image2rgbhist ('3.jpg', N);
y(1,3) = 1.0;

x(:,4) = image2rgbhist ('4.jpg', N);
y(1,4) = 1.0;

x(:,5) = image2rgbhist ('5.jpg', N);
y(1,5) = 1.0;

x(:,6) = image2rgbhist ('6.jpg', N);
y(1,6) = 1.0;

x(:,7) = image2rgbhist ('7.jpg', N);
y(1,7) = 1.0;

x(:,8) = image2rgbhist ('8.jpg', N);
y(1,8) = 1.0;

x(:,9) = image2rgbhist ('9.jpg', N);
y(1,9) = 1.0;

x(:,10) = image2rgbhist ('10.jpg', N);
y(1,10) = 1.0;


% Not human faces: y=0
x(:,11) = image2rgbhist ('11.jpg', N);
y(1,11) = 0.0;

x(:,12) = image2rgbhist ('12.jpg', N);
y(1,12) = 0.0;

x(:,13) = image2rgbhist ('13.jpg', N);
y(1,13) = 0.0;

x(:,14) = image2rgbhist ('14.jpg', N);
y(1,14) = 0.0;

x(:,15) = image2rgbhist ('15.jpg', N);
y(1,15) = 0.0;

x(:,16) = image2rgbhist ('16.jpg', N);
y(1,16) = 0.0;

x(:,17) = image2rgbhist ('17.jpg', N);
y(1,17) = 0.0;

x(:,18) = image2rgbhist ('18.jpg', N);
y(1,18) = 0.0;

x(:,19) = image2rgbhist ('19.jpg', N);
y(1,19) = 0.0;

x(:,20) = image2rgbhist ('20.jpg', N);
y(1,20) = 0.0;
```

```
% Run through Levenberg-Marquad algorithm.
if toggle == 0
 w=0;
else
 load rgb_weights w;
end;

[w,res] = levmar (x, y, p, w, lam, itMAX);
save rgb_weights w;

% Output overall residual on final weights w.
% Residual = Network Output - Desired Output.

for i=1:20
 outputs(i) = forward(x(:,i),1,p,w);
 overall_res(i) = outputs(i) - y(1,i);
end;

plot(outputs);
xlabel('INDEX');
ylabel('P');
title('Network outputs');
```

2. image2rgbhist

```
function [x] = image2rgbhist (file_name, N)

% Writes an image at file_name (i.e. jpg) into 3 histograms.
% Each histogram has N bins.
% The vector is the appended R, G, then B histograms.
% We scale x at the end so all entries are between 0 and 1.
% This is a necessary condition for feeding into the neural network.
% The resize_matrix operation assumes each image is at least 50x50 pixels.

% Get bin centers.
for i=1:N
 Bins(i,1) = i/N - 1/(2*N);
end;

%Interpolation factor I
I = 50;

A = double(imread(file_name));
B = resize_matrix(A,I,I);
B = B/260;
[total_rows total_columns three] = size(B);

for row = 1:total_rows
 for column = 1:total_columns
  v(column+(row-1)*total_columns,1) = B(row,column,1);      % R
  v(column+(row-1)*total_columns,2) = B(row,column,2);      % G
  v(column+(row-1)*total_columns,3) = B(row,column,3);      % B
```

```
 end;
end;

total_hist = hist(v,Bins);

hist_R = total_hist(:,1);
hist_G = total_hist(:,2);
hist_B = total_hist(:,3);

x=[hist_R;hist_G;hist_B];
x = x / (I^2);
```

3. rgb_forward

```
function [y] = rgb_forward (file_name)

% rgb_forward.m

% Uses RGB histogram approach for identifying image.
% Given an image file (i.e. jpeg), returns value y
% y = 1 for face, y = 0 for not face
% Assumes weights are stored as w in rgb_weights.mat
% Plots results

x = image2rgbhist (file_name, 20);
load rgb_weights w;
y = forward (x, 1, 20 , w);

subplot (3,1,1);
imshow(file_name);
subplot(3,1,2);
bar(x);
xlabel('R:1-20  G:21-40  B:41-60');
axis([1,60,0,max(x)+0.01]);
ylabel('Color frequency')
title('RGB histogram')
subplot(3,1,3);
plot(y, '+');
axis([0.5, 1.5, 0, 1]);
xlabel(y);
```

YES HISTOGRAN APPROACH

1. yes_trainall1

```
function [x, outputs] = yes_trainall1(N,p, toggle)

% Trains all images in directory images.
% We give file names of all jpg images.
% Plots final residual of each image at end.
% Given # bins N and # hidden nodes p.
% If toggle = 0, then we start with random initial weights.
% Otherwise assumes inital weights w given in file yes_weights unless toggle=0.

warning off;
```

```
%Training rate lambda.
lam = 0.1;

% # iterations.
itMAX = 10;

% Human faces: y=1
x(:,1) = image2yeshist ('1.jpg', N);
y(1,1) = 1.0;

x(:,2) = image2yeshist ('2.jpg', N);
y(1,2) = 1.0;

x(:,3) = image2yeshist ('3.jpg', N);
y(1,3) = 1.0;

x(:,4) = image2yeshist ('4.jpg', N);
y(1,4) = 1.0;

x(:,5) = image2yeshist ('5.jpg', N);
y(1,5) = 1.0;

x(:,6) = image2yeshist ('6.jpg', N);
y(1,6) = 1.0;

x(:,7) = image2yeshist ('7.jpg', N);
y(1,7) = 1.0;

x(:,8) = image2yeshist ('8.jpg', N);
y(1,8) = 1.0;

x(:,9) = image2yeshist ('9.jpg', N);
y(1,9) = 1.0;

x(:,10) = image2yeshist ('10.jpg', N);
y(1,10) = 1.0;


% Not human faces: y=0
x(:,11) = image2yeshist ('11.jpg', N);
y(1,11) = 0.0;

x(:,12) = image2yeshist ('12.jpg', N);
y(1,12) = 0.0;

x(:,13) = image2yeshist ('13.jpg', N);
y(1,13) = 0.0;

x(:,14) = image2yeshist ('14.jpg', N);
y(1,14) = 0.0;

x(:,15) = image2yeshist ('15.jpg', N);
y(1,15) = 0.0;

x(:,16) = image2yeshist ('16.jpg', N);
y(1,16) = 0.0;
```

```
x(:,17) = image2yeshist ('17.jpg', N);
y(1,17) = 0.0;

x(:,18) = image2yeshist ('18.jpg', N);
y(1,18) = 0.0;

x(:,19) = image2yeshist ('19.jpg', N);
y(1,19) = 0.0;

x(:,20) = image2yeshist ('20.jpg', N);
y(1,20) = 0.0;


% Run through Levenberg-Marquad algorithm.
if toggle == 0
  w=0;
else
  load yes_weights w;
end;

[w,res] = levmar (x, y, p, w, lam, itMAX);
save yes_weights w;


% Output overall residual on final weights w.
% Residual = Network Output - Desired Output.

for i=1:20
  outputs(i) = forward(x(:,i),1,p,w);
  overall_res(i) = outputs(i) - y(1,i);
end;

plot(outputs);
xlabel('INDEX');
ylabel('P');
title('Network outputs');
```

2. image2yeshist

```
function [x] = image2yeshist (file_name, N)

% Writes an image at file_name (i.e. jpg) into 3 histograms.
% Each histogram has N bins.
% The vector is the appended Y, E, then S histograms.
% We scale x at the end so all entries are between 0 and 1.
% This is a necessary condition for feeding into the neural network.
% The resize_matrix operation assumes each image is at least 50x50 pixels.

% Get bin centers.
for i=1:N
  Bins(i,1) = i/N - 1/(2*N);
end;

%Interpolation factor I
```

```
I = 50;

A = double(imread(file_name));
B = resize_matrix(A,I,I);
B = B/260;
[total_rows total_columns three] = size(B);

Y = 0.253*B(:,:,1)+0.684*B(:,:,2)+0.063*B(:,:,3);
E = 0.500*B(:,:,1)-0.500*B(:,:,2);
S = 0.250*B(:,:,1)+0.250*B(:,:,2)-0.500*B(:,:,3);

% Change each YES matrix into a vector.
for row = 1:total_rows
  for column = 1:total_columns
    v(column+(row-1)*total_columns,1) = Y(row,column);      % Y
    v(column+(row-1)*total_columns,2) = E(row,column);      % E
    v(column+(row-1)*total_columns,3) = S(row,column);      % S
  end;
end;

total_hist = hist(v,Bins);

hist_Y = total_hist(:,1);
hist_E = total_hist(:,2);
hist_S = total_hist(:,3);

x=[hist_Y ; hist_E ; hist_S];
x = x / (I^2);
```

3. yes_forward

```
function [y] = yes_forward (file_name)

% yes_forward.m

% Uses YES histogram approach for identifying image.
% Given an image file (i.e. jpeg), returns value y
% y = 1 for face, y = 0 for not face
% Assumes weights are stored as w in yes_weights.mat
% Plots results

x = image2yeshist (file_name, 20);
load yes_weights w;
y = forward (x, 1, 20 , w);

subplot (3,1,1);
imshow(file_name);
subplot(3,1,2);
bar(x);
xlabel('Y:1-20  E:21-40  S:41-60');
axis([1,60,0,max(x)+0.01]);
ylabel('Color frequency')
title('YES histogram')
subplot(3,1,3);
plot(y, '+');
axis([0.5, 1.5, 0, 1]);
xlabel(y);
```

COMMON FUNCTION

1. forward

```
function [y] = forward (x, n, p ,w)

% forward.m
%  Calculates forward output of given input x with weights w.
%  x(1:m) vector corresponding to given input
%  n is # of nodes in output (T)
%  p number of nodes in hidden layer
%  w(1:P) vector corresponding to weights -- P=p*(m+1)+n*(p+1)
%  y(1:n) vector corresponding to network output

  m=length(x)
  xp=[x;1];
  tem=w(1:p*(m+1));
  W1=reshape(tem,p,m+1);
  tem=w(p*(m+1)+1:p*(m+1)+n*(p+1));
  W2=reshape(tem,n,p+1);
  a=W1*xp;
  z=sigmoid(a);
  zp=[z;1];
  y=W2*zp;
```

2. resize_matrix

```
function [B] = resize_matrix (A, r, c)

% To use type "[B] = resize_matrix (A, #, #)".
% Resizes image matrix A by picking out pixels.
% Resulting matrix B will have r rows by c columns.
% Each pixel has three values: RGB.

[rows columns three] = size(A);

row_sample = floor (rows/r);
column_sample = floor (columns/c);

for i=1:r
   for j=1:c
     B(i,j,:) = A(i*row_sample, j*column_sample, :);
end;
end;
```

Program Listing of Appearance Based Technique for face Detection

Main M-files for Face Detection and training

1. Test

```
IM    the input test image
IM = double(imread('test2.png'));
NumFace = 1;

load model

% START can be 1, but use to ignore smaller rectangles (level to start at)
% STEP is good at 2
% SCALEFACT is good at 1.2
START=7;
STEP=2;
LEVELS=7;
SCALEFACT=1.2;

% Setup
PYR_MAX = LEVELS;
MROWS = size(MASK,1);
MCOLS = size(MASK,2);
IROWS = size(IM, 1);
ICOLS = size(IM, 2);
RECT = [];

% Build the image pyramid
SCALE = SCALEFACT; % A good choice is 1.2
PYR{1} = IM;
XRANGE{1} = 1:1:ICOLS;
YRANGE{1} = 1:1:IROWS;
[MX{1},MY{1}] = meshgrid(XRANGE{1}, YRANGE{1});
for i=2:PYR_MAX,
   XRANGE{i} = 1:SCALE.^(i-1):ICOLS;
   YRANGE{i} = 1:SCALE.^(i-1):IROWS;
   [MX{i},MY{i}] = meshgrid(XRANGE{i}, YRANGE{i});
   PYR{i} = interp2(MX{1}, MY{1}, PYR{1}, MX{i}, MY{i});
end

% View pyramid
%figure;
%colormap(gray);
%showimages(PYR, 2, 3, 1, 6, 1);
%drawnow;
%pause;

% Scan the pyramid
for im_num = START:PYR_MAX,
  fprintf(1, '\n\nImage Scale: %d\n', im_num);
  for im_row = 1:STEP:size(PYR{im_num},1)-MROWS+1,
    fprintf(1, '\n Row:%d', im_row);
```

```
  for im_col = 1:STEP:size(PYR{im_num},2)-MCOLS+1,
    TEST = 0;
    if bPCA==1
      TEST = classify_1NN(PYR{im_num}, MASK, im_row, im_col,mean_ALL, U1, S1, data_proj,
cFACEV, cNFACEV);
    elseif bPCA_LDA==1
      TEST = classify_1NN_new(PYR{im_num}, MASK, im_row, im_col, mean_ALL, meanF,
meanNF, U1, S1, U2, S2, dataF_proj, dataNF_proj);
    elseif bSVM==1
      TEST = classify_SVM(PYR{im_num}, MASK, im_row, im_col);
    elseif bPCA_SVM==1
      TEST = classify_PCA_SVM(PYR{im_num}, MASK, im_row, im_col, mean_ALL, U1, S1);
    end
    if (TEST == 1)
      fprintf(1, '\n ---(#SCALE,R,C): [%d] (%d,%d) ',im_num, im_row, im_col);
      RECT = [RECT; (im_row/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...          % top
            (im_col/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...          % left
            ((im_row+MROWS-1)/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...          %
bottom
            ((im_col+MCOLS-1)/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...    % right
            TEST];                                          % TEST value
  end
 end
end

% Plot the bounding boxes in an image
IMR = IM;
drawbox(IMR, RECT, 1);

% post-process to eliminate these false detections
RECT = postprocess(IM, RECT, NumFace);

% plot again
IMtmp = IM;
drawbox(IMtmp, RECT, 1);
```

2. Train_PCA

```
[eMin:eMax] range eigen_value & eigen_vectors we will keep
eMin = 3;
eMax = 23;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                    Image Loading
%%%
%%%
% assume training images are already preprocessed:
%  i.e. cropped (to left face pixels only) and scaled to appropriate size (18*27 i.e.)
% now load in the images and make up the training data mtx
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the image oval mask
MASK = buildmask;
```

```
NI  = size(find(MASK),1);

% load face images, normalize, and set training vectors
                                         % temp: only for current test query here!!!!
                                         % need to change back if not use current query images
FACES = loadimages('./trainset/train/face/', '', 'pgm');
%FACES = loadimages_bootstrap(FACES, './bootstrap/face/','','PNG');
%FACES = augmentlr(FACES);
[NORM_FACES, SHADING] = normalize(FACES, MASK);

clear FACES SHADING

FACEV = buildimvector(NORM_FACES, MASK);
%FACER = buildresvector(NORM_FACES, FACE_T);

clear NORM_FACES

% load non-face images, normalize, and set training vectors

NFACES = loadimages('./trainset/train/non-face/', '', 'pgm');
%NFACES = loadimages_bootstrap(NFACES, './bootstrap/non-face/', '','PNG');
%NFACES = augmentlr(NFACES);
[NORM_NFACES, NSHADING] = normalize(NFACES, MASK);

clear NFACES NSHADING

NFACEV = buildimvector(NORM_NFACES, MASK);
%NFACER = buildresvector(NORM_NFACES, FACE_F);

clear NORM_NFACES

cFACEV = size(FACEV, 2);
cNFACEV = size(NFACEV, 2);


% optional --- Display images
if 0,
disp('original image data');
showimages(NORM_FACES,  5, 10, 1, 50, 1);
% showimages(NORM_NFACES, 5, 5, 1, 25, 2);
% pause;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%          Build the sub-space with eigen- vectors & values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% FACESV & NFACESV: each face is a column vector

% collect face & non_face data together
%   : each (non-)face is a column vector
ALL = [FACEV NFACEV];

clear FACEV NFACEV
```

```
mean_ALL = sum(ALL, 2)/size(ALL,2);

% mean subtracted:
data = ALL - mean_ALL *ones(1, size( ALL,2));

% get U and S from data
% as you may try&found, when using large dataset, eig() or svd() fail to reach a converged solution
% while snap-shot method (as used by Pentland et. al.) or em-pca methods can still have a good solution

% SVD
%[U1, S1, V1] = svd(data);

% eig
%[U1, S1] = eig(data*data'); S1=sqrt(S1);

% snap shot method (not implement yet, you are encouraged to implement one here!)

% EM-PCA
                % iter: default is 20
iter = 20;
                % for em-pca, nnedn't mean-subtraction here
[U1, S1] = empca(ALL,eMax,iter);
S1 = diag(S1);          % S1 is a mtx now

% chop U&S to preserve only the [eMin:eMax] range of eigen- vectors & values
U1 = U1 (:, eMin:eMax);
S1 = S1 (eMin:eMax, eMin:eMax);

% project training data into sub-space
data_proj = (U1*inv(S1))' * data;


% optional --- eigen-spectra plots
if 0,
tmp1=diag(S1);
subplot(1,2,1),plot(1:length(tmp1), tmp1);
end

% optional --- now we can re-construct face by the remaining eigen data
if 0,
FACEV_recon = U1 *U1' *data(:,1:cFACEV)  + mean_ALL *ones(1, cFACEV);
NORM_FACE_recon = buildface(FACEV_recon, MASK);
disp('the reconstruction result');
showimagevecs(NORM_FACE_recon, size(MASK,1), size(MASK,2), 5, 10, 1, 50, 2);

end

save model

bPCA=1;
bPCA_LDA=0;
bSVM=0;
bPCA_SVM=0;
```

3. **Truepca**

```
function [evects,evals] = truepca(dataset)
% [evects,evals] = truepca(dataset)
%
% USUAL WAY TO DO PCA -- find sample covariance and diagonalize
%
% input: dataset
% note, in dataset, each COLUMN is a datapoint
% the data mean will be subtracted and discarded
%
% output: evects holds the eigenvectors, one per column
%       evals holds the corresponding eigenvalues
%
% History:
%    Feb. 21, 2002  Modified by Li to use SVD instead of eig
%    Feb. 23, 2002  Modified by Li to use JDQR instead of eig

[d,N]  = size(dataset);

mm = mean(dataset')';
dataset = dataset - mm*ones(1,N);

cc = cov(dataset',1);
% original code
[cvv,cdd] = eig(cc);
% Li: use SVD alternative
%[cvv,cdd,dummy] = svd(cc,0);
% Li: use JDQR alternative
%[cvv,cdd] = JDQR(cc);


[zz,ii] = sort(diag(cdd));
ii = flipud(ii);
evects = cvv(:,ii);
cdd = diag(cdd);
evals = cdd(ii);
```

4. Assert

```
function [] = assert(condition,message)

if nargin == 1,message = '';end
if isempty(message),message = 'Assert Failure.'; end
if(~condition) fprintf(1,'!!! %s !!!\n',message); end
```

5. empca

```
function [evec,eval] = empca(data,k,iter,Cinit)
%[evec,eval] = empca(data,k,iter,Cinit)
%
% EMPCA
%
% finds the first k principal components of a dataset
% and their associated eigenvales using the EM-PCA algorithm
%
% Inputs:  data is a matrix holding the input data
%          each COLUMN of data is one data vector
%          NB: mean will be subtracted and discarded
```

```
%       k   is # of principal components to find
%
% optional:
%       iters is the number of iterations of EM to run (default 20)
%       Cinit is the initial (current) guess for C (default random)
%
% Outputs:  evec holds the eigenvectors (one per column)
%         eval holds the eigenvalues
%


[d,N]  = size(data);
data = data - mean(data,2)*ones(1,N);

if(nargin<4) Cinit=[]; end
if(nargin<3) iter=20; end

[evec,eval] = empca_orth(data,empca_iter(data,Cinit,k,iter));



function [C] = empca_iter(data,Cinit,k,iter)
%[C] = empca_iter(data,Cinit,k,iter)
%
% EMPCA_ITER
%
% (re)fits the model
%
%   data = Cx + gaussian noise
%
% with EM using x of dimension k
%
% Inputs:  data is a matrix holding the input data
%            each COLUMN of data is one data vector
%            NB: DATA SHOULD BE ZERO MEAN!
%       k   is dimension of latent variable space
%            (# of principal components)
%       Cinit is the initial (current) guess for C
%       iters is the number of iterations of EM to run
%
% Outputs: C is a (re)estimate of the matrix C
%          whose columns span the principal subspace
%

% check sizes and stuff
[p,N] = size(data);
assert(k<=p);
if(isempty(Cinit))
  C = rand(p,k);
else
  assert(k==size(Cinit,2));
  assert(p==size(Cinit,1));
  C = Cinit;
end

% business part of the code -- looks just like the math!
```

```
for i=1:iter
    % e step -- estimate unknown x by random projection
 x = inv(C'*C)*C'*data;
    % m step -- maximize likelihood wrt C given these x values
 C = data*x'*inv(x*x');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [evec,eval] = empca_orth(data,C)
%[evec,eval] = empca_orth(data,Cfinal)
%
% EMPCA_ORTH
%
% Finds eigenvectors and eigenvalues given a matrix C whose columns span the
% principal subspace.
%
% Inputs:  data is a matrix holding the input data
%            each COLUMN of data is one data vector
%            NB: DATA SHOULD BE ZERO MEAN!
%        Cfinal is the final C matrix from empca.m
%
% Outputs: evec,eval are the eigenvectors and eigenvalues found
%        by projecting the data into C's column space and finding and
%        ordered orthogonal basis using a vanilla pca method
%

 C = orth(C);
 [xevec,eval] = truepca(C'*data);
 evec = C*xevec;
```

6. empcaol

```
function [evec,eval] = empcaol(k,iter,Cinit)
%[evec,eval] = empcaol(k,iter,Cinit)
%
% EMPCAOL (ONLINE VERSION OF EMPCA)
%
% finds the first k principal components of a dataset
% and their associated eigenvales using the EM-PCA algorithm
%
% Inputs:  k    is # of principal components to find
%
% optional:
%        iters is the number of iterations of EM to run (default 20)
%        Cinit is the initial (current) guess for C (default random)
%
% the data is provided by the function nextpoint
% nextpoint(1) re-initializes the data providing function
% nextpoint(0) to get successive datavectors
% nextpoint(0) should return 0 when it is out of data
% NB: nextpoint should return data with the mean already subtracted out
%
% Outputs:  evec holds the eigenvectors (one per column)
%          eval holds the eigenvalues
```

```
%

if(nargin<3) Cinit=[]; end
if(nargin<2) iter=20; end

[evec,eval] = empcaol_orth(empcaol_iter(Cinit,k,iter));



function [C] = empcaol_iter(Cinit,k,iter)
%[C] = empcaol_iter(Cinit,k,iter)
%
% EMPCA ONLINE ITERATIONS
%
% (re)fits the model
%
%   data = Cx + gaussian noise
%
% with EM using x of dimension k
% Gets points one at a time ONLINE. Uses the function nextpoint.m.
%
% Inputs:  k    is dimension of latent variable space
%             (# of principal components)
%        Cinit is the initial (current) guess for C
%        iters is the number of iterations of EM to run
%
% Outputs: C is a (re)estimate of the matrix C
%           whose columns span the principal subspace
%
% uses nextpoint(1); to reinitialize nextpoint each pass through the data
% uses nextpoint(0) to get successive datavectors
% NB: nextpoint should return data with the mean already subtracted out
%

% check sizes and stuff


p = nextpoint(1);

if(isempty(Cinit))
 C = rand(p,k);
else
 assert(k==size(Cinit,2));
 assert(p==size(Cinit,1));
 C = Cinit;
end

% loop for iterations
for ii=1:iter
 nextpoint(1);      % reset nextpoint
 C = empcaol1(C);   % let's do it
end

function [Cnew] = empcaol1(C)
%[Cnew] = empcaol1(C)
```

```
%
%does one complete E AND M step of empca by calling nextpoint(0)
%to get successive datapoints
%
% NB: nextpoint should return data with the mean already subtracted out

[p,k] = size(C);
CC = inv(C'*C)*C';
W = zeros(k,k);
Q = zeros(p,k);

[yi,status] = nextpoint(0);

while(status>0)
%  fprintf(1,'Now processing datapoint %d\r',status);
  xi = CC*yi;
  wi = xi*xi'; W = W+wi;
  qi = yi*xi'; Q = Q+qi;
  [yi,status] = nextpoint(0);
end

Cnew = Q*inv(W);



function [evec,eval] = empcaol_orth(C)
%[evec,eval] = empcaol_orth(Cfinal)
%
% finds ordered orthogonal basis for subspace identified in Cfinal
%
% online method
% uses nextpoint(1) to initialize data generator
% uses nextpoint(0) to provide successive data vectors
% NB: nextpoint should return data with the mean already subtracted out


[p,k] = size(C);
W = zeros(k,k);

C = orth(C);

nextpoint(1);
[yi,status] = nextpoint(0);
while(status>0)
%    fprintf(1,'Now processing datapoint %d\r',status);
  nf = status;
  xi = C'*yi;
  W = W+xi*xi';
  [yi,status] = nextpoint(0);
end

[cvv,cdd] = eig(W/nf);
[zz,ii] = sort(diag(cdd));
ii = flipud(ii);
xevec = cvv(:,ii);
```

```
cdd = diag(cdd);
eval = cdd(ii);

evec = C*xevec;
```

7. nextpoint

```
function [datapoint,status] = nextpoint(reset)
% [datapoint,status] = nextpoint(reset)
%
% NEXTPOINT - skeleton function
%
% this function returns the next datapoint for online methods.
%
% nextpoint(1) should return the dimensionality of the data
%       and reset to the beginning of the dataset
%
% nextpoint(0) should return the next datapoint and a status flag
%       status=1 if we still have more data
%       status=0 if we are out of data
%


global dat;
global thisn;
[p,N] = size(dat);

if(reset)
 % go back to beginning of dataset and return dimensionality
 datapoint = p; status=p;
 thisn=1;
elseif(thisn<=N)
 % return next datapoint and status=1 or status=0 if at end
 datapoint=dat(:,thisn);
 thisn=thisn+1;
 if(thisn>N) status=0; else status=1; end
else
 datapoint = NaN;
 status = 0;
end
```

Image Utilities Function

1. augmentlr

```
function IM = augmentlr(IM)

num = size(IM,2);
nrows = size(IM{1},1);
ncols = size(IM{1},2);

for i=1:num,
  IM{i+num} = IM{i}(1:1:nrows,ncols:-1:1);
end
```

2. buildface

```
function res = buildface(DATA, MASK)

% DATA is the the masked data (the non-zero portion of the MASK)
% MASK is a mtx containing 0&1.

imgs = size(DATA,2);
INDICES = find(MASK);

res = zeros(size(MASK(:),1),imgs);
for i=1:imgs
   tmp = MASK(:);
   tmp(INDICES)=DATA(:,i);
   res(:,i) = tmp;
end
```

3. **buildmvector**

```
function IMVECTOR = buildimvector(IM, MASK)

pics = size(IM,2);
INDICES = find(MASK);

IMVECTOR = zeros(size(find(MASK),1),pics);
for i=1:pics,
  IMVECTOR(:,i) = IM{i}(INDICES);
end
```

4. **buildmask**

```
function MASK = buildmask()

% An 19x19 mask
MASK = ...
   [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0   ];

if 0,
% An 18x27 mask
```

```
MASK = ...
   [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; ...
    0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0; ...
    0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
    0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0; ...
    0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0; ...
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    ];
end

MASKW = size(MASK, 2);
MASKH = size(MASK, 1);
```

5. **classify_1NN**

```
function res = classify_1NN(IM, MASK, srow, scol, mean_ALL, U1, S1, data_proj, cFACEV, cNFACEV)

% 1-NN classfication scalar factor for face & non-face
% res == 1: IS a face
% res ~= 1: NOT a face
%

MASKH = size(MASK, 1);
MASKW = size(MASK, 2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                    1-NN Classification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% given a query data vector, decide if it is face by 1-NN
QUERY{1} = IM(srow:srow+MASKH-1,scol:scol+MASKW-1);          % the fixed moving-window
size
[NORM_QUERYS, SHADING] = normalize(QUERY, MASK);
QUERYV = buildimvector(NORM_QUERYS, MASK);
```

```
% project into the PCA sub-space
% QUERYV = QUERYV - mean_ALL; needn't do the mean-subtraction here!! to retain the discrimination
power
Q_proj1 = (U1*inv(S1))' * QUERYV;

Q_F = Q_proj1 * ones(1, cFACEV);
Q_NF= Q_proj1 * ones(1, cNFACEV);

dataF_proj = data_proj(:,1:cFACEV);
dataNF_proj = data_proj(:,cFACEV+1:size(data_proj,2));

clear data_proj;

% two dist. measurement, we now choose the euclidean one..
                                        % euclidean dist.
tmp_F=sum((dataF_proj-Q_F).^2, 1);
tmp_NF=sum((dataNF_proj-Q_NF).^2, 1);

                                        % cosin dist. (similarity dist.)
%tmp_F = abs(Q_F_proj'*dataF_proj)/(norm(Q_F_proj)*norm(dataF_proj,'fro'));
%tmp_NF= abs(Q_NF_proj'*dataNF_proj)/(norm(Q_NF_proj)*norm(dataNF_proj,'fro'));

mintmpF = min(tmp_F);
mintmpNF= min(tmp_NF);

% find which face/non-face is the closest
% idx_F = find(tmp_F == meantmpF );
% idx_NF= find(tmp_NF== meantmpNF);

% because non-face data tend to be scattered compared to face data,
% so here we need a scalar factor to balance them

if mintmpF < mintmpNF
  %fprintf('\n the query image IS a face image');
  res = 1;
else
  res = 0;
  %fprintf('\n the query image is NOT a face image');
end
```

6. **normalize**

```
function [OUT, SHADING] = normalize(IN, MASK)

% Retrieve the indices for the given mask
IND = find(MASK);

% Set up matrices for planar projection calculation
% i.e. Ax = B  so  x = (A'*A)^-1 * A'*B
x = 1:1:size(IN{1},2);
y = 1:1:size(IN{1},1);
[mx,my] = meshgrid(x,y);
mxc = mx(IND);
myc = my(IND);
mcc = ones(size(myc));
```

```
A = [mxc, myc, mcc];

% Cycle through each image removing shading plane
% and adjusting histogram
for i=1:size(IN,2),

  % Calculate plane: z = ax + by + c
  B = IN{i}(IND);
  x = inv(A'*A)*A'*B;
  a = x(1); b = x(2); c = x(3);

  %This is the color plane itself
  SHADING{i} = mx.*a + my.*b + c;

  %This is the image minus the color plane
  %(the constant will be normalized out in histogram recentering)
  OUT{i} = IN{i} - (mx.*a + my.*b + c);

  % Now, recenter the histogram
  maximum = max(max(OUT{i}.*MASK));
  minimum = min(min(OUT{i}.*MASK));   %minimum = min(min(OUT{i}))
  diff = maximum - minimum;

  % original one
  %OUT{i} = ((OUT{i}-minimum)./diff).*MASK;
  % Li's modification to fix divided by zero bug
  if diff==0,
    diff = 1e10; % a positive infinite value
  else
    OUT{i} = ((OUT{i}-minimum)./diff).*MASK;
  end;

end
```

7. **postprocess**

```
function res = postprocess(IM, RECT, NumFace)

[h,w] = size(IM);
NUMRect= size(RECT, 1);


% hC   : hit count
% currR : current RECT
% i,j   : current col @ row position
% currR(1), currR(3): top, bottom
% currR(2), currR(4): left, right
% currR(5): -1:false detection,
%         0: not decided,
%         1:NUMFace : the detection box for specific face

myRECT = (RECT);
myRECT(:,5) = zeros(NUMRect, 1);

for index = 1:NumFace
  %
```

```
% find current max hit position

MaxHit = 0; posH =0; posW = 0;
for i=1:h
for j=1:w
  hC = 0;
  for k=1:NUMRect
    currR = myRECT(k,:);
    if (i>currR(1) & i<currR(3) & ...
      j>currR(2) & j<currR(4) & currR(5)==0)
      hC = hC + 1;
    end
  end

  if hC > MaxHit
    MaxHit = hC;
    posH = i; posW = j;
  end

end
end

% compute the overlapped area out of these overlapped boxes
for k=1:NUMRect
  currR = myRECT(k,:);
  if (posH>currR(1) & posH<currR(3) & ...
    posW>currR(2) & posW<currR(4) & currR(5)==0)
    myRECT(k,5) = index;
  end
end

area = [1,1, size(IM,1), size(IM,2)];
for k=1:NUMRect
  if myRECT(k,5) == index
    area = myrectint(area, myRECT(k,1:4));
  end
end

center = [(area(1)+area(3))/2, (area(2)+area(4))/2];

% find the center of area (center), then elect only ONE box which's center closest to center, and
eleminate others
closest = 1e5; ind_closest=0;
for k=1:NUMRect
  if myRECT(k,5) == index
    centerR = [(myRECT(k,1)+myRECT(k,3))/2, (myRECT(k,2)+myRECT(k,4))/2];
    if norm(centerR-center) < closest
      closest = norm(centerR-center);
      ind_closest = k;
    end
  end
end

for k=1:NUMRect
  if myRECT(k,5) == index & ind_closest ~= k
    myRECT(k,5) =-1;
```

```
        end
    end
end

res = [];
for k=1:NUMRect
    if myRECT(k,5)>0
        res = [res; RECT(k,:)];
    end
end


% sub routines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = myrectint(area1, area2)
res = zeros(size(area1));

if (area1(1)>area2(1))
    res(1)=area1(1);
else
    res(1)=area2(1);
end;

if (area1(2)>area2(2))
    res(2)=area1(2);
else
    res(2)=area2(2);
end;

if (area1(3)<area2(3))
    res(3)=area1(3);
else
    res(3)=area2(3);
end;

if (area1(4)<area2(4))
    res(4)=area1(4);
else
    res(4)=area2(4);
end;
```

Image Loading and Display Files

1. **loadimages**

```
function IM = loadimages(directory, prefix, suffix)

% cd IM this directory
old_dir=pwd;
cd(directory);

% fIMd all matched filenames
dirinfo = dir([prefix,'*.',suffix]);
found = {dirinfo.name};
```

```
% Load the image set
for i = 1:size(found,2),
  if suffix=='pgm'
    IM{i} = double(pgmRead(found{i}));
  else
    IM{i} = double(imread(found{i}));
  end
end

res = IM;

clear found

% cd out
cd(old_dir);
```

2. loadimages_bootstrap

```
function res = loadimages_bootstrap(IM, directory, prefix, suffix)

% #imgs already exists IM FACES
numImgs=size(IM, 2);

% cd IM this directory
old_dir=pwd;
cd(directory);

% flMd all matched filenames
dirinfo = dir([prefix,'*.',suffix]);
found = {dirinfo.name};


% Load the image set
for i = 1:size(found,2),
  if suffix=='pgm'
    IM{i} = double(pgmRead(found{i}));
  else
   IM{numImgs+i} = double(imread(found{i}));
  end
end

res = IM;

clear found

% cd out
cd(old_dir);
```

3. scaleImg

```
function res = scaleImg(imname, cscale, rscale, imgsavename)

if(nargin<4) imgsavename='out.PNG'; end
```

```
IM = double(imread(imname));
IROWS = size(IM, 1);
ICOLS = size(IM, 2);

PYR{1} = IM;
XRANGE{1} = 1:1:ICOLS;
YRANGE{1} = 1:1:IROWS;
[MX{1},MY{1}] = meshgrid(XRANGE{1}, YRANGE{1});

XRANGE{2} = 1:cscale:ICOLS;
YRANGE{2} = 1:rscale:IROWS;
[MX{2},MY{2}] = meshgrid(XRANGE{2}, YRANGE{2});
PYR{2} = interp2(MX{1}, MY{1}, PYR{1}, MX{2}, MY{2});

imwrite(PYR{2}/255, imgsavename)
imshow(PYR{2}/255)
```

4. showImages

```
function showimages(IM, xdim, ydim, start, end1, fign)

% Show the image set if fign is valid
if (fign>0)
  figure(fign);
  for i=start:end1,
    subplot(xdim,ydim,i-start+1);
    imagesc(IM{i});
    colormap gray;
  end
end
```

5. showimagevecs

```
function showimagevecs(IM, imH, imW, xdim, ydim, start, end1, fign)

% Show the image set if fign is valid
if (fign>0)
  figure(fign);
  for i=start:end1,
    subplot(xdim,ydim,i-start+1);
    imagesc(reshape(IM(:,i), imH, imW));
    colormap gray;
  end
end
```

END