SAN DIEGO STATE UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

Final Project April 2004

"EXPERIMENTS ON HUMAN IRIS RECOGNITION USING ERROR BACKPROPAGATION ARTIFICIAL NEURAL NETWORK"

PAULO EDUARDO MERLOTI (PADU)

Prepared for Neural Networks Class (CS553) of Spring Semester of 2004
Prof. Dr. Roman Swiniarski

ABSTRACT

The Human Iris is one of the best biometrics features in the human body for pattern recognition. This paper provides a walkthrough for image acquisition, image segmentation, feature extraction and pattern forming based on the Human Iris imaging. It also shows experiments using Feed forward Backpropagation Neural Network on classifying the patterns formed in the first part of the paper and properly verify one's identity. This paper is submitted as partial fulfillment for class CS553 – Neural Networks.

Keywords: Iris Recognition, Biometric Identification, Image Segmentation, Singular Value Decomposition, Independent Component Analysis, Pattern Recognition, Error Backpropagation, Neural Networks.

1. Introduction

Biometrics as form of unique identification is one of the subjects of research that more rapidly grow in Computer Science. The advantages of unique identification using biometrics features are numerous, such as fraud prevention and ease of use. Although the current state of art provides reliable automatic recognition of biometric features, the field is not completely researched. Different biometric feature offer different degrees of reliability and performance.

Known biometric features include:

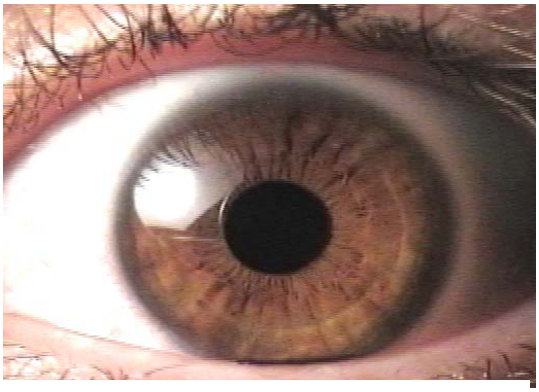


Figure 1: Human Iris with Pupil and biometric features

- Fingerprint
- Hand geometry
- Voice recognition
- Retinal Recognition
- Handwritten Patterns

In this study we choose the Human Iris image as object of recognition and some advantages will become apparent in the next section.

1.1. Iris Biometric Features

The use of the Human Iris as a biometric feature offers many advantages over other human biometric features. The iris is the only internal human body organ that is visible from the outside [1], thus well protected from external modifiers. A fingerprint for example may suffer transformations due to harm or aging, voice patterns may be altered due to vocal diseases. Yet, the human iris image is relatively simple to image and may be done so in a non-intrusive way.

The Human Iris starts forming still in the mother uterus, in the third month of gestation [3] and the visible structures are formed by the eighth month. [1] It is particularly good for automatic recognition because of its complex pattern of many distinctive features such as arching ligaments, furrows, ridges, crypts, rings, corona, freckles, and a zigzag collarette. Some of these patterns may be seen in Fig. 1.

Human Iris has epigenetic formation and it is formed part from the individual DNA, but a great deal of its final pattern is developed at random. It means that two eyes from the same individual, although they look very similar, contain internal pattern unique. Identical twins would then exhibit four different irises patterns.

1.2. Existing Products and Research

Although Automatic Iris Recognition is recently new in the biometrics market, several studies have been performed and products are already available in the market for consumption.

For instance, Panasonic [1] commercializes a line of biometric products (cameras, iris recognition software, servers) with prices ranging from US$300 up to several thousand dollars. Most of the commercial systems implement an algorithm for Iris Recognition by John Daugman [1], one of the most active researchers in this field while working for Iridian Technologies. This company retains the patent for Daugman's algorithm [4]. LG Electronics and Oki Electric also offer identification systems based on Iris Patterns.

1.3. Assumptions

This paper is not a through and complete research on the subject of Iris classification. It is a brief walkthrough on the basic steps necessary to acquire Iris image, perform image segmentation and feature extraction and how to design a Feed forward Backpropagation or Error (backprop) Neural Network to classify (identify) patterns presented to the network. Some assumptions will be made in order to simplify the research.

First, only a few words will be given towards explaining the physical constrains and problems of image acquisition. Instead, we will use the CASIA iris image database [2] collected by Institute of Automation, Chinese Academy of Sciences [5].

Methods for image segmentation and feature extraction will assume all patterns have the same rotation angle, and results may be affected by this decision.

[1] [http://www.panasonic.com/business/security/biometrics.asp]

[2] [http://www.sinobiometrics.com]

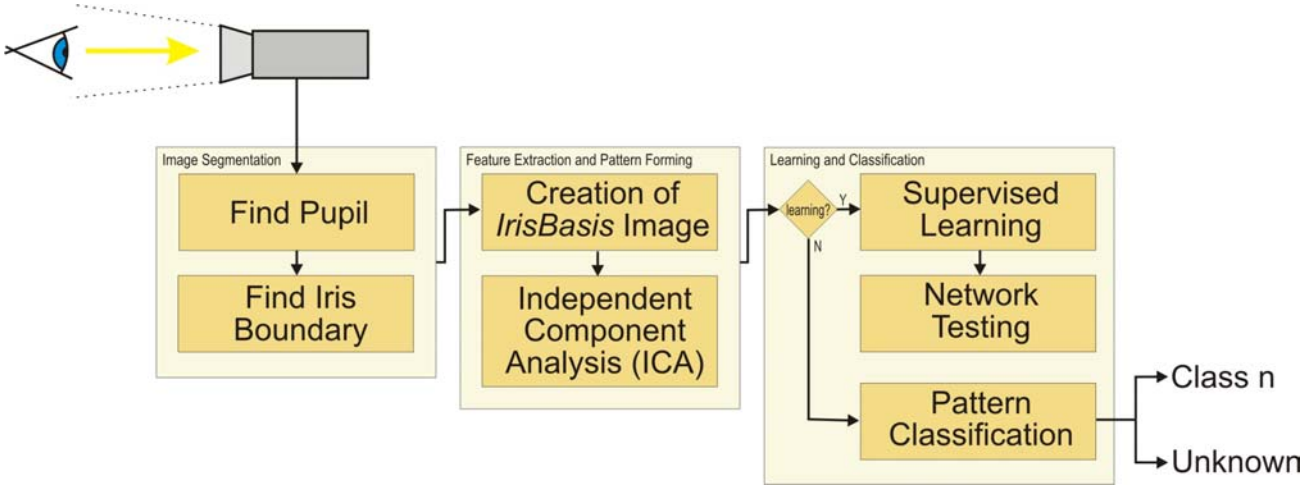


Figure 2: Diagram showing processing steps, from image capture to image segmentation, feature extraction, pattern forming and finally learning and classifying

PART I – PREPROCESSING

This paper will be divided in two parts; this first one describes the preprocessing techniques. The second one uses data generated by the first part of the paper for pattern classification using a Backpropagation Neural Network. Fig. 2 shows an overview of the processing steps taken in this paper.

2. IMAGE ACQUISITION

One of the main problems of Iris Recognition is image acquisition. With an average diameter of 12mm, a camera must have enough resolution to capture the details of the iris pattern (collarette patterns). Some commercial products solve the problem with a dual lens system. One lens is a wide angle that serves the purpose of localization of the iris in the scene. The other with a higher zoom factor (tele) is driven by the iris localization system, narrowing the focus only to the target and therefore taking a high resolution snapshot of the iris.

Another problem is illumination. The collarettes are very similar to mountain ranges, with valleys, dips and peaks. The illumination angle will determine the dark and light parts of the image. It is very important that one system implements consistent illumination, on the contrary the same iris may generate two different classes under two different illumination angles. Also, the pupil is an open door to the retina, one of the most sensitive organs of our body, and extra care must be taken when shedding direct light over it.

2.1. ILLUMINATION

CASIA Iris [5] Database uses a special camera that operates in the infrared spectrum of light, not visible by the human eye.

Although illumination is one of the most important problems on Iris Recognition, we will not attempt to tackle this issue in this paper.

2.2. DATABASE CHARACTERISTICS

This paper is based on the CASIA image database. Its ethnical distribution is composed mainly of Asians. Each iris class is composed of 7 samples taken in two sessions, three in the first session and four in the second. Sessions were taken with an interval of one month.

Images are 320x280 pixels gray scale taken by a digital optical sensor designed by NLPR (National Laboratory of Pattern Recognition – Chinese Academy of Sciences). There are 108 classes or irises in a total of 756 iris images.

3. IMAGE SEGMENTATION

Several researches were made in the subject of iris finding and segmentation. The main objective here is to remove non useful information, namely the pupil segment and the part outside the iris (sclera, eyelids, skin).

Wildes [6] uses Hough transforms to automatically detect the iris contour. Daugman [1] proposes an integro-differential operator to find both the pupil and the iris contour.

While Daugman's method is claimed to be the most effective one, we will propose a novel method in

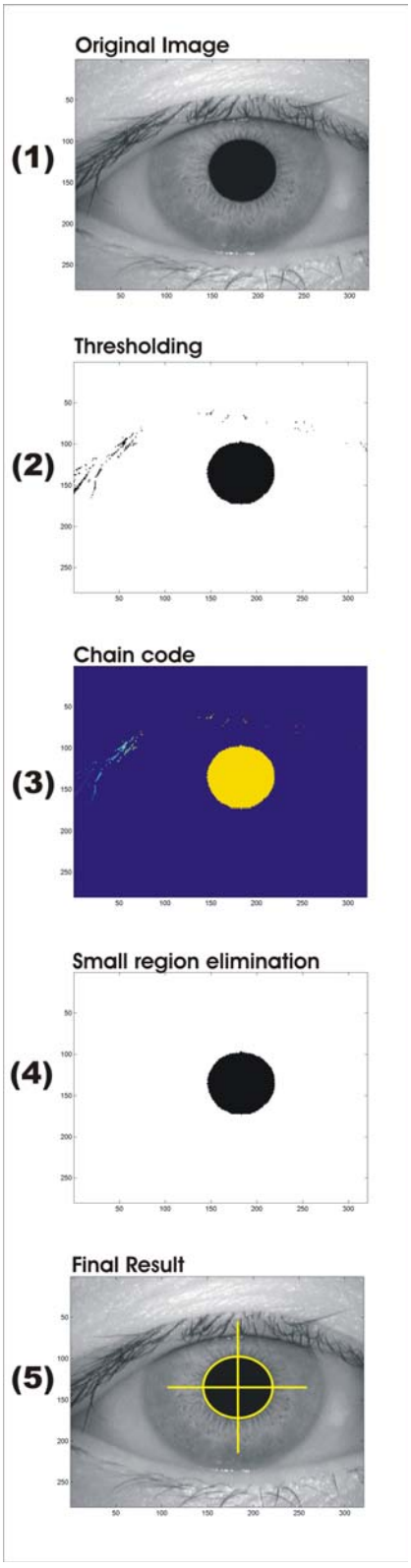


Figure 3: Five steps to perform pupil detection. The final product of the algorithm is the pupil center coordinate (x,y) and the horizontal and vertical radiuses of the pupil (rx,ry)

this paper that performs the task in hand in two phases. The algorithm used in the first phase uses the knowledge that a pupil is a very dark blob of a certain minimum size in the picture, and no other segment of continuous dark pixels are of the same size. This algorithm finds the center of the pupil and two radial coefficients as the pupil is not always a perfect circle. The second algorithm takes the information of the pupil center and tries to find edges on a one-dimensional imaginary line to each side of the pupil. The algorithms will be presented in the next two sections and tested against the iris database for validation.

3.1. PUPILLARY BOUNDARY

Assuming that the iris region is already found in the face, and the image is windowed in the eye region (iris, sclera, eyelids), the area that constitutes the iris is very distinguishable because it has a concentration of pixels all of them with very low level intensity (black or almost black).

To find the pupil, we first need to apply a linear threshold in the image,

$$g(x) = \begin{cases} f(x) > 70 : 1 \\ f(x) \leq 70 : 0 \end{cases}$$

Where f is the original image and g is the thresholded image. Pixels with intensity greater than the empirical value of 70 (in a 0 to 256 scale) are dark pixels, therefore converted to 1 (black). Pixels smaller than or equal to 70 are assigned to 0 (white).

Next, we apply Freeman's [7] chain code to find regions of 8-connected pixels that are assigned with value equal 1. From Fig. 3 it is possible to see that eyelashes also satisfy the threshold condition, but have a much smaller area than the pupil area. Using this knowledge, we can cycle through all regions and apply the following condition:

```
for each region R
  if AREA(R) < 2500
    set all pixels of R to 0
```

Finally, we apply one last time the chain code algorithm in order to retrieve the only region in the image (hopefully the pupil). From this region, it is trivial to obtain [16, 17] its central moments. Finding the edges of the pupil involves the creation of two imaginary orthogonal lines passing through the centroid of the region. The boundaries of the binarized pupil are defined by the first pixel with intensity zero, from the center to the extremities.

This process is illustrated in detail in Figure 3, and the Matlab implementation of this algorithm can be found in Appendix A – Matlab Source Code.

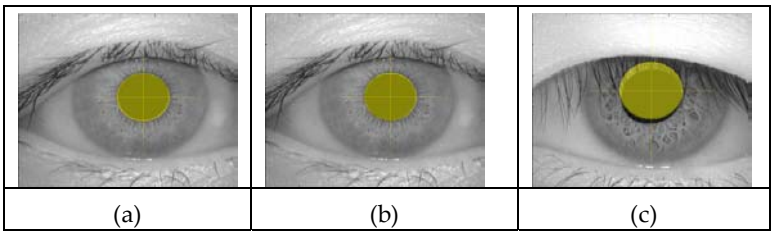


Figure 4: The left two images (a) and (b) show an example of the pupilfinder algorithm correctly locating the pupil within the iris image. Image (c) shows an example of algorithm inaccuracy when the eyelash is too close to the pupil.

This method has proven very efficient and reliable when applied to the CASIA Iris Database. From all 756 Iris images, this algorithm was able to correctly locate the center and radius of 752 exemplars (99.4%) with a very low rate of failure (only 4 exemplars). Analyzing the images where the algorithm failed, we found out that most of them come from irises with high degree of eyelash overlapping and eyelids covering part of the iris. Figure 4 shows an example of images pupils correctly located and failed ones. Eyelash is not a new problem in iris recognition research, Kong and Zhang [8] presented a study on how to detect eyelashes within an iris picture.

3.2. Iris Edge Detection

The next step towards iris segmentation is finding the contour of the iris. This may seem an easy task at first as we already have discovered the pupil location and we have the knowledge that it is concentric to the outer perimeter of the iris. The first problem comes from the anatomy of the eye and the fact that every person is different. As seen in Fig. 4c, sometimes the eyelid may occlude part of the iris, as it will occur often with the Asians, and no full circularity may be assumed in this case. Other times due to variation in gaze direction the iris center will not match the pupil center, and we will have to deal with strips of iris of different width around the pupil, as illustrated in Figure 5.

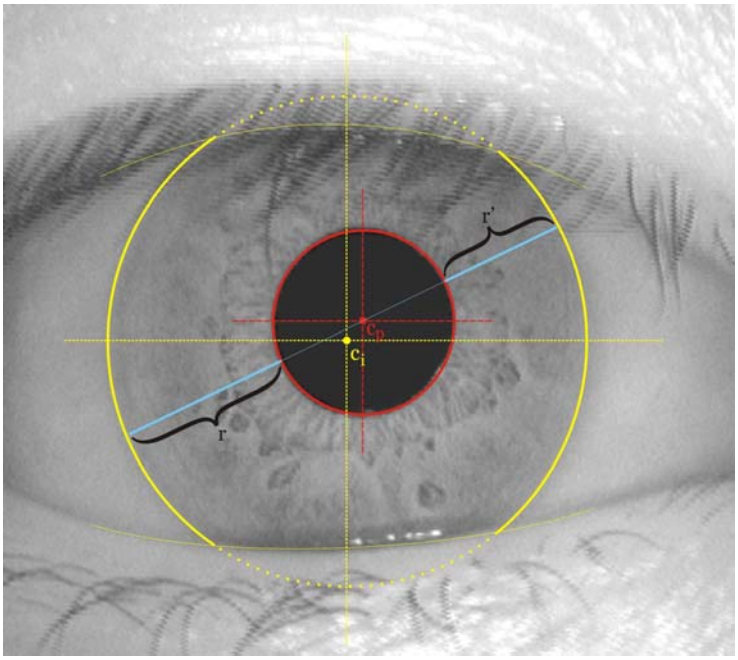


Figure 5: Centerpoint of iris (ci) does not coincide with certerpoint of pupil (cp). Consequently the iris strip of opposite sides have different widths: width(r)≠width(r')

Our method takes in consideration that areas of the iris at the right and left of the pupil are the ones that most often present visible to data extraction. The areas above and bellow the pupil also carry unique information, but it is very common that they are totally or partially occluded by eyelash or eyelid.

The strategy adopted for iris detection uses the information from section 3.1 to trace a horizontal imaginary line that crosses the whole image passing through the center of the pupil. Starting from the edges of the pupil, we analyze the signal composed by pixel intensity from the center of the image towards the border and try do detect abrupt increases of intensity level. Although the edge between the iris disk and the sclera is most of the times smooth, it is known that it always have greater intensity than iris pixels. We intensify this difference applying a linear contrast filter.

It is possible that some pixels inside the iris disk are very bright, causing a sudden rise in intensity. That could mislead the algorithm to detect that iris edge at that point. To prevent that from happening, we take the intensity average of small windows and then detect when the sudden rises occur from these intervals. This algorithm is implemented in Matlab and listed on Appendix A.

Following is a description of the

steps taken to detect the edges of the iris image I(x,y).

1. Find the center (x_{cp},y_{cp}) of the pupil and the horizontal pupil radius rx using section's 3.1 algorithm.
2. Apply a linear contrast filter on image I(*x*,*y*): $G(x,y) = I(x,y).\alpha$
   We obtained satisfactory results with α=1.4
3. Create vector V={$v_1,v_2,\ldots,v_w$} that holds pixel intensities of the imaginary row passing through the center of the pupil (rx), with w being the width of contrasted image G(*x*,*y*).
4. Create vector R={$r_{x_{cp}+rx}, r_{x_{cp}+rx+1},\ldots,r_w$} from the row that passes through the center of the pupil (y_{cp}) in contrasted iris image G(*x*,*y*). Vector R formed by the elements of the y_{cp} line that start at the right fringe of the pupil ($x_{cp}+rx$) and go all the way to the width (w) of the image. Experiences shown that adding a small margin to the fringe of the pupil provides good results as it covers for small errors of the "findpupil" algorithm.
5. Similar as described above, create vector L={$l_{x_{cp}-rx}, l_{x_{cp}-rx-1},\ldots,r_1$} from row ($y_{cp}$) of G(*x*,*y*). This time we are forming vector L which contains elements of pupil center line starting at the left fringe of the pupil and ending at the first element of that line.
6. For each side of the pupil (vector R for the right side and vector L for the left side):
   a. Calculate the average window vector A={$a_1,\ldots,a_n$} where n=|L| or n=|R|. Vector A is subdivided in i windows of size ws. For all window $i_1^{n/ws}$, elements $a_{i.ws-ws}\ldots a_{i.ws}$ will contain the average of that window. We found through experiments that a window size ws=15 provides satisfactory results for the CASIA Iris database.
   b. Identify the edge point given side of the iris (vector L or R) as the first increase of values in A_j ($1 \le j \le n$) that exceeds a set threshold t. In our experiments, a value of t equal to 10 has shown to identify the correct location of the iris edge.

The advantage of this algorithm is performance. For an image of size $m.n$, the complexity is O($m.n$) for the contrast operation and O(m) for edge finding, total complexity of O($m.n$). The reader must be warned though that algorithm efficiency and reliability highly depends on carefully chosen threshold (t) and window size (ws). Other modifications to the algorithm may also help improve the overall accuracy of the algorithm, for

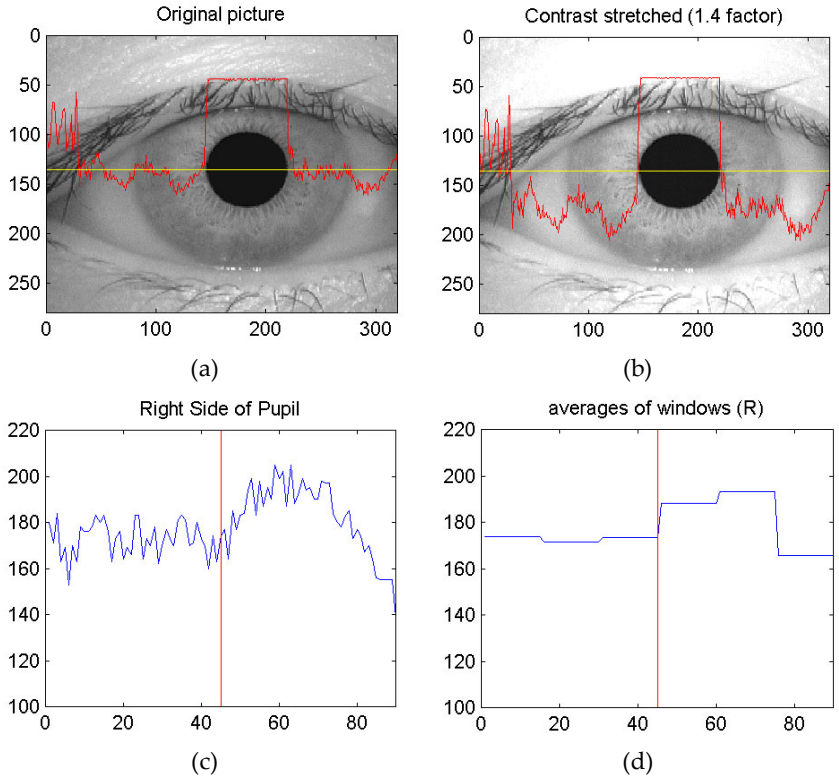


Figure 6: Example of steps taken to find the right edge of the iris shown in the image. (a) Yellow line passes through y_{cp} (center of pupil) of original image, red line shows pixel intensities for that line. (b) Effects of contrast stretching visible through red line. (c) Intensities of pixels on yellow center line y_{cp} and edge location. (d) Signal of right side of the pupil averaged through small windows.

instance adding margins to the sides of the pupil.

Figure 6 shows practical results of the algorithm implemented in Matlab.

Fast computation comes with a price, and the algorithm is very sensitive to local intensity variation (or lack of). Testing the algorithm against 756 images from CASIA Iris Database resulted in 594 successes (78.6%) and 162 failure (21.4%). From these failures, we noted that many of them occurred because of eyelashes were around the iris or the sclera was not as white as expected. As a result no significant increase in the average intensity level was perceived by the algorithm, and in some cases no edge was detected in one side of the image.

For the purpose of this paper, this technique will suffice, but it is clear in the author's vision that any serious commercial system must achieve a segmentation success close to 100%.

4. Feature Extraction

So far we have performed the segmentation of the iris, first by finding the pupil and then finding the outer edge of the iris at the line that crosses the center of the pupil. The main reason for the prior segmentation is two folded. The first reason is to isolate only information that distinguishes individuals, namely the iris patterns. The second one is the attempt to reduce the size of pattern vector. For instance, the CASIA Iris Database provides images that are 320x280 pixels. If we concatenate all rows of the image into only one vector, the dimension of the problem would be to classify a vector with 89,600 elements. This dimension is too high for today's computing power, and even though we had such capacity, satisfactory results are not guaranteed.

4.1. Forming *IrisBasis*: Overall Strategy

IrisBasis is our first attempt to reduce the dimensionality of the problem while focusing only on parts of the scene that effectively identify the individual. Also, we restrict the mapping of the iris to areas known to have less influence of eyelashes and eyelids, the sides of the iris. Assuming that intra-class rotation of iris is practically void, we propose an approach to extract pixels of either side of the pupil and forming one reduced image of the iris.

The overall strategy is as follows: given image, desired number of *IrisBasis* rows and columns,

- Retrieve pupil center and radius
- Retrieve iris endpoints
- Calculate height of pupil (2 times radius)
- Calculate space between rows (*s*) as pupil height divided by desired number of *IrisBasis* rows.
- Calculate index of the first target row as *center of pupil – vertical pupil radius*, assuming that first row is at the top of the image.
- For each side of the pupil
  - Calculate baseline width as |*iris edge of current side – pupil edge of current side*|
  - For all baselines, starting at the top of the pupil, ending at the bottom of the pupil and spaced by *s*, perform the following steps:
    - Calculate, using the equation of the circle, the (x,y) location of pixel that resides in the intersection of current baseline and circle centered at pupil center with radius equal to pupil horizontal radius.
    - Map pixels that are under the baseline to vector B with number of elements equal to half of desired number of columns. Use an average mask of 3x3 pixels to calculate pixel intensity.
    - Append vector B to *IrisBasis* matrix of respective side of the iris
- Merge the two halves of *IrisBasis* matrices side by side into one final *IrisBasis* matrix.

We can better visualize this strategy by looking at figure 7.

The final result of classification will depend on how close texture features are in samples of the same class. From Fig. 7, it is also possible to realize that this algorithm takes in consideration only features to the sides of the pupil. As explained before, this is an attempt to avoid eyelashes and eyelids.

Figure 8 shows three classes of patterns, each one with three instances of original image and IrisBasis image. Looking at intra class *IrisBasis* of several classes it is possible to note the similarities.

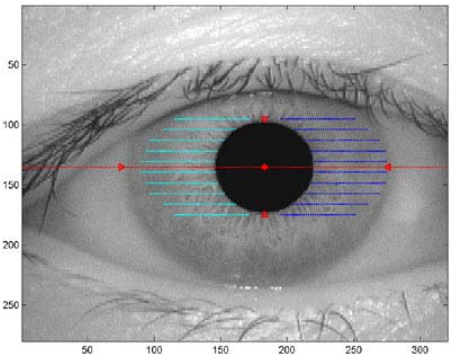


Figure 7: Image of iris with pupil center line in red. Also in red are the vertical pupil markers and the iris edge points. The blue lines form the right half of the IrisBasis matrix, while the cyan lines form the other half.

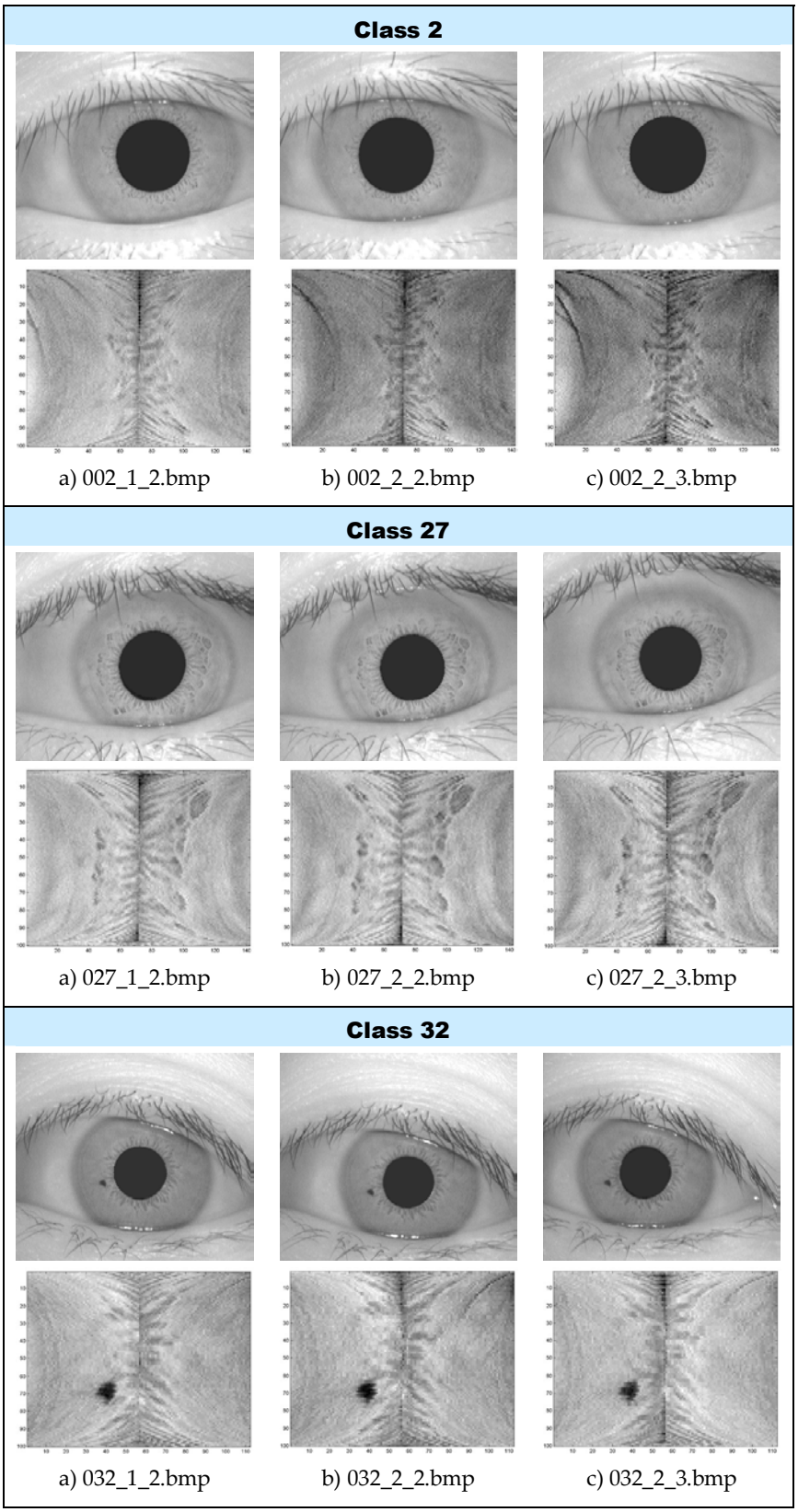


Figure 8: Exemplars of three different classes and their respective "IrisBasis" image. Each class shows three different instances of the iris. The first column show pictures taken in one session, last two columns show pictures taken one month later.

5. Pattern Forming

Dimension reduction was already achieved in the previous sections by simply isolating the area of the image that contains the characteristic information about an individual. For instance we decrease the number of elements in the input vector from 89,600 (320x280 image) to 10,000 (100x100 image), a 89% reduction in size. Still, this number is still too high for plausible classification.

Considering the fact that a large percentage of these pixels contain redundant information, we can utilize mathematical and statistical methods to reduce the dimensionality of the problem. This problem is also known as Feature Selection, from a set of *n* features, select a subset of *m* features that leads to minimum classification error.

5.1. Available Preprocessing Methods

Jain, Duin and Mao [11] wrote an excellent review on Statistical Pattern Recognition. In their review they compare and analyze the main methods for feature extraction and feature selection.

In this section we will briefly discuss the principal methods for dimensionality reduction, mainly extracted from [11].

Single Value Decomposition: Very simple and easy to implement and calculate. Retrieves a vector of singular values arranged as the main diagonal of the matrix in decreasing order of value.

Principal Component Analysis (PCA): Simple to implement and fast performance. PCA is based on linear mapping and eingenvectors. It's one of the most traditional methods around, also known as Karhunen-Loeve expansion. Good for Gaussian data.

Linear Discriminant Analysis: Also based on eigenvectors, it is faster than PCA for classification. Limited to the number of classes – 1 components with non-zero eingenvalues. Implements supervised linear mapping.

Projection Pursuit: Also a linear map, iterative and non-Gaussian. Mainly used for exploratory data-analysis.

Independent Component Analysis: Blind source separation method, very efficient de-mixing non-Gaussian distributed features. It is also an iterative, linear mapping technique.

Kernel PCA: Although also based on eingenvectors, implements non-linear mapping. It's a method based on Principal Component Analysis (PCA) that uses a kernel to replace inner products of pattern vectors.

PCA Network: Auto associative Neural Network with linear transfer functions and only one hidden layer.

Nonlinear PCA: Neural Network approach, possibly used for ICA.

Nonlinear auto-associative network: Implements a Bottleneck network with possibly many hidden layers. The nonlinear mapping is optimized by nonlinear reconstruction; input is used as target.

Multidimensional Scaling (MDS): Often poor generalization; sample size is limited and it is sensitive to noise. Main application resides on 2-D visualization.

Self Organizing Maps (SOM): Also very useful in 2-D visualization of clusters of data. It is based in a grid of neurons in the feature space.

From these techniques we explain in more details SVD and ICA, both used in the classification section.

5.2. Single Value Decomposition

Single Value Decomposition (SVD) is a powerful matrix technique with many useful applications. The main concept behind it is to expose the hidden geometry of the matrix. SVD is utilized in a variety of applications [18], from least-squares problems to solving of linear equations. We also benefit from SVD using it as a dimension reduction tool.

The basic operation of SVD relies on the factorization of an MxN matrix (M ≥ N) into three other matrices on the following form:

$$A = U\Sigma V^T$$

Where the superscript "T" denotes transpose. U is an MxM orthogonal matrix, V is an NxN orthogonal matrix and $\sum$ is an MxN diagonal matrix with $s_{ij}=0$ if $i \neq j$ and $s_{ii} \geq s_{i+1i+1}$. This SVD property is very useful for our application.

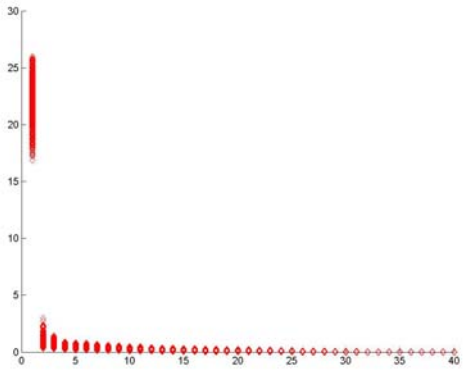


Figure 9: Plotting of first 40 dimensions of 756 SVD pattern vectors. It is possible to see that after the first 20 dimensions values are close to zero or zero.

Because the ∑ is zero everywhere except in the main diagonal, we automatically reduce the dimension of our input pattern from a matrix MxN to only a vector of N elements. The second property shown above is also very important, as it tell us that only the first *k* elements contain substantial information, and we can crop the vector tail without significant loss of generality.

The algorithm we used to convert *IrisBasis* images into SVD patterns can be found in "*Appendix A – IrisBasisToSVDPattern.m*".

5.3. Independent Component Analysis

Automatic Pattern Recognition and Classification faces one very difficult problem of dimensionality reduction, a task easily done by the human perception. Our sensory systems are constantly capturing a huge amount of information, and if not summarized, it would definitely bottleneck our whole processing capacity. Natural sensory signals are highly redundant [13], and based on these characteristic methods such as ICA, PCA, Wavelets and Fourier Transforms are studied.

Although Independent Component Analysis (ICA) and its many flavors were originally conceived for blind source separation (BSS), many studies were conducted on the application of ICA to feature extraction from time series and images [12]. Redundant distributed data from natural signals such as natural sounds or images have a high degree of redundancy. The redundancy can be translated as statistical dependency between units [13]. ICA is the redundancy reduction via linear transformations. Application of ICA on a natural scene can be compared then with edge detection methods in image processing such as Wavelets, Fourier Transforms and Convolution Kernels (Sobel, Roberts, Canny).

Gävert, Hurri, Särelä, and Hyvärinen came up with a Matlab implementation of the FastICA algorithm [12]. We will base our pre-processing of input signals on their implementation of the algorithm.

Unlike SVD, ICA depends on the correlation information between patterns, and to obtain reasonable results, we need to process the full dataset in order to obtain the Separating and the Mixing matrix, as well as the independent components.

Also, as reduction is done internally in the algorithm we need to decide on the final number of dimensions prior to processing. Some special arrangements are also necessary in order to obtain the independent components. For example, given the input image I_{MxN} and the data set D_L of *L* cases, the usual way to arrange pixels of this image is to concatenate *N* rows of *M* columns into only one row of *(M.N)* columns. The training set would have in this case *L* rows and *M.N* columns. For supervised learning, there is also one additional column labeling the case, making the total length of the pattern vector as *M.N+1*. The FastICA algorithm requires transposing the data set so it ends up with *L* columns and *M.N* rows.

The listing below shows the output of three FastICA transformations, from 1600 dimensions to 3, 10 and 50 dimensions.

```
3-Dimension Reduction using ICA (FastICA)
[icasig5,A,W]=fastica(I,'lastEig',5,'numOfIC',5)
Number of signals: 1600
Number of samples: 756
Warning: The signal matrix may be oriented in the wrong way.
In that case transpose the matrix.
oldDimension = 1600
Calculating covariance...
Reducing dimension...
Selected [ 3 ] dimensions.
Smallest remaining (non-zero) eigenvalue [ 0.372255 ]
Largest remaining (non-zero) eigenvalue [ 3.10221 ]
Sum of removed eigenvalues [ 2.90682 ]
[ 58.5217 ] % of (non-zero) eigenvalues retained.
Whitening...
Check: covariance differs from identity by [ 4.32987e-015 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
Starting ICA calculation...
IC 1 .........computed ( 9 steps )
IC 2 .........computed ( 8 steps )
IC 3 .........computed ( 2 steps )
Done.
Adding the mean back to the data.
Note that the plots don't have the mean added.
```

```
10-Dimension Reduction using ICA (FastICA)
[icasig10,A,W]=fastica(I,'lastEig',10,'numOfIC',10)
Number of signals: 1600
Number of samples: 756
Warning: The signal matrix may be oriented in the wrong way.
In that case transpose the matrix.
oldDimension = 1600
Calculating covariance...
Reducing dimension...
Selected [ 10 ] dimensions.
Smallest remaining (non-zero) eigenvalue [ 0.0738082 ]
Largest remaining (non-zero) eigenvalue [ 3.10221 ]
Sum of removed eigenvalues [ 1.8214 ]
[ 74.0099 ] % of (non-zero) eigenvalues retained.
Whitening...
Check: covariance differs from identity by [ 4.32987e-015 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
Starting ICA calculation...
IC 1 ......computed ( 10 steps )
IC 2 ......computed ( 8 steps )
IC 3 ......computed ( 23 steps )
IC 4 ......computed ( 25 steps )
IC 5 ......computed ( 11 steps )
IC 6 ......computed ( 17 steps )
IC 7 ......computed ( 19 steps )
IC 8 ......computed ( 36 steps )
IC 9 ......computed ( 28 steps )
IC 10 .....computed ( 2 steps )
Done.
Adding the mean back to the data.
Note that the plots don't have the mean added.
```

```
50-Dimension Reduction using ICA (FastICA)
[icasig50,A,W]=fastica(I,'lastEig',50,'numOfIC',50)
Number of signals: 1600
Number of samples: 756
Warning: The signal matrix may be oriented in the wrong way.
In that case transpose the matrix.
oldDimension = 1600
Calculating covariance...
Reducing dimension...
```

```
Selected [ 50 ] dimensions.
Smallest remaining (non-zero) eigenvalue [ 0.00917543 ]
Largest remaining (non-zero) eigenvalue [ 3.10221 ]
Sum of removed eigenvalues [ 0.959531 ]
[ 86.3081 ] % of (non-zero) eigenvalues retained.
Whitening...
Check: covariance differs from identity by [ 2.02061e-014 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
Warning: There are too many signals to plot. Plot may not look good.
Starting ICA calculation...
IC 1 .......computed ( 67 steps )
IC 2 .......computed ( 26 steps )
IC 3 .......computed ( 8 steps )
IC 4 .......computed ( 72 steps )
IC 5 .......computed ( 15 steps )
IC 6 .......computed ( 13 steps )
IC 7 .......computed ( 6 steps )
IC 8 .......computed ( 6 steps )
IC 9 .......computed ( 15 steps )
IC 10 ......computed ( 26 steps )
IC 11 ......computed ( 25 steps )
IC 12 ......computed ( 17 steps )
IC 13 ......computed ( 9 steps )
IC 14 ......computed ( 21 steps )
IC 15 ......computed ( 35 steps )
IC 16 ......computed ( 12 steps )
IC 17 ......computed ( 33 steps )
IC 18 ......computed ( 7 steps )
IC 19 ......computed ( 12 steps )
IC 20 ......computed ( 10 steps )
IC 21 ......computed ( 20 steps )
IC 22 ......computed ( 9 steps )
IC 23 ......computed ( 15 steps )
IC 24 ......computed ( 29 steps )
IC 25 ......computed ( 16 steps )
IC 26 ......computed ( 19 steps )
IC 27 ......computed ( 15 steps )
IC 28 ......computed ( 24 steps )
IC 29 ......computed ( 11 steps )
IC 30 ......computed ( 41 steps )
IC 31 ......computed ( 19 steps )
IC 32 ......computed ( 56 steps )
IC 33 ......computed ( 12 steps )
IC 34 ......computed ( 17 steps )
IC 35 ......computed ( 25 steps )
IC 36 ......computed ( 25 steps )
IC 37 ......computed ( 55 steps )
IC 38 ......computed ( 58 steps )
IC 39 ......computed ( 21 steps )
IC 40 ......computed ( 19 steps )
IC 41 ......computed ( 11 steps )
IC 42 ......computed ( 11 steps )
IC 43 ......computed ( 16 steps )
IC 44 ......computed ( 20 steps )
IC 45 ......computed ( 145 steps )
IC 46 ......computed ( 16 steps )
IC 47 ......computed ( 63 steps )
IC 48 ......computed ( 12 steps )
IC 49 ......computed ( 10 steps )
IC 50 ......computed (  2 steps )
Done.
Adding the mean back to the data.
Note that the plots don't have the mean added.
```

Before going to the merits of classification, we can do a few comparisons between ICA and SVD. Looking again at Fig. 9 shows the clustering of all patterns and all cases when dimension reduced to 40 dimensions. It is very visible that only a few dimensions carry meaningful information. Only the first 5 or 6 dimensions have value not so close to zero, and only the first dimension really distinguishes itself from the others.

With ICA, as mentioned before, we don't have the ranking behavior where elements are in decreasing order, but in the other hand, there are plenty of degrees of variance between patterns and between dimensions, as noted in figure 10.

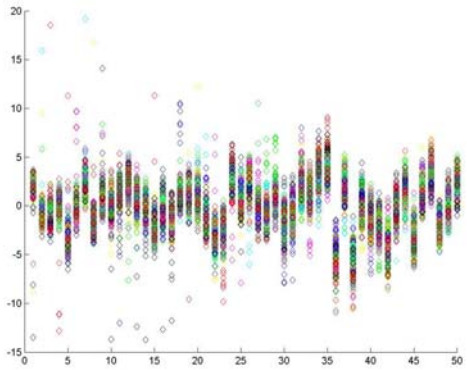


Figure 10 Clustering of all cases (756) dimension reduced. Original dimension was 1600, reduced to 50 dimensions. Different colors represent different classes. While it is still not possible to visualize cluster separation, the contrast between SVD and ICA dimension reduction effects is obvious.

PART II – CLASSIFICATION

In the next few sections we propose a method of classification of data extracted from the *IrisBasis* image and pre-processed in "**Part I"** of this paper.

Neural Networks is a very diverse field and many researchers have been investing a great deal of time trying to figure out new ideas and new networks. For the problem in hand, supervised network training and classification, several network models are available, and we've chosen "Feedforward Backpropagation of Error" because it is very well suited for supervised problems, and it is very simple to implement. If data has been properly pre-processed, the classification rate of backpropagation networks is similar to more sophisticated networks, as Radial Basis Function (RBF) networks.

6. NETWORK ARCHITECTURE

Our *backprop* network implements the classical 3-layer architecture: Input layer, Hidden layer and Output layer. The input layer contains as much neurons as the dimensionality of the pattern vector, with N neurons. In the following sections we present some numerical experiments with different number of neurons in the hidden layer and input layer. As a rule of thumb, approximately double of the neurons in the input layer usually obtain good classification results. The output layer will contain as much neurons as there are classes to recognize. In our experiments with CASIA Iris, 108 classes were utilized, therefore 108 output neurons in our network.

The section entitled "Network Design" shows with more detail how encoding and decoding is performed in order to classify input patterns. Fig. 11

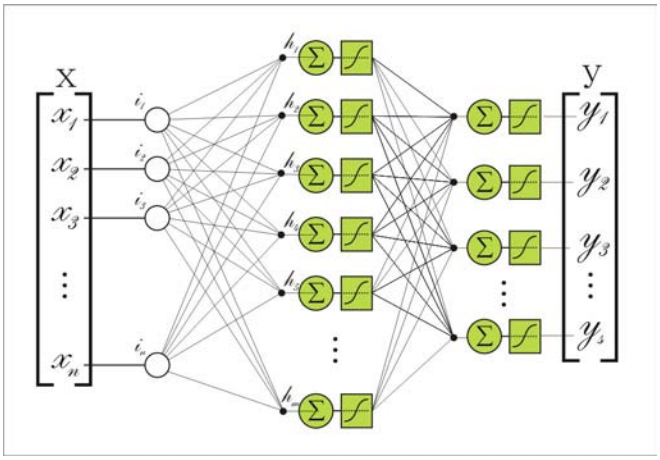


Figure 11: Feedforward Error Backpropagation Neural Network architecture showing input pattern, input layer, hidden layer, output layer and output pattern

shows an illustration of the proposed network architecture.

7. Network Design

Following are some parameters set for network training:

Training function: traingda
Adaptive learning rate

Initial learning rate: 0.2

Learning rate increment: 1.1

Epochs: 50,000

Error goal: 0.0000005

Minimum gradient: 0.000000001

As mentioned earlier, the number of neurons in the output layer corresponds to the number of classes to recognize. When the network is trained in supervised mode, a target vector is also presented to the network. This target vector T has every element set to zero, except on the position of the target class that will be set to 1.

The idea behind this design decision is that for each input pattern X presented to the network, an output vector Y is produced. This vector has the same number of elements of output neurons. Each output neuron implements a squashing function that produces a Real number in the range [0,1]. To determine which class is being indicated by the network, we select the maximum number in Y and set it to 1, while setting all other elements to zero. The element set to one indicates the classification of that input pattern.

8. Experiments and Validation

While the theory behind neural networks is well understood and mostly mathematically proven, designing a network may involve heuristic determination of parameters. Also, different problems demand different approaches, and sometimes the best way to find out a good solution to a problem is trying different scenarios, different methods and techniques. Once the results of different experiments are recorded and analyzed, one can accumulate enough knowledge in order to better decide on the best way to solve the problem in hand.

In this section we will perform some experiments with patterns extracted from the Iris segmentation phase (*IrisBasis*). The first experiment uses patterns dimensionally reduced using SVD. We will demonstrate that the number of dimensions of the input pattern influences the final classification rate. Not only the dimension of the problem, but also the number of classes to recognize.

The final experiment will try to show that using a more sophisticated feature compression technique may positively affect the classification rate, as it supposedly promotes more separation between classes.

8.1. SVD Experiments

In order to better understand the recognition problem and how important the pre-processing of data is, we show on this section classification results for input patterns that had their dimension reduced by using SVD. The SVD algorithm outputs a feature vector in decreasing order of values. This allows us

to choose how many elements the input pattern will have and how it impacts on the final classification result. The first test shows 3-Dimensional input pattern classification and following we show the same statistics for 10 dimensions. Experiments here are based on *IrisBasis* images of 40x40 pixels quantized from the original iris image with an average mask of 3x3 pixels. We also varied the number of patterns in the dataset in order to find proof that it may affect the final classification rate.

Cluster Separation

We start this experiment by plotting a scattered chart of the pattern dataset containing 3 classes, each one containing 7 cases. Figure 12 help us better visualize what to expect for class separation in the pattern space. As we can see, with 3 classes it is still possible to determine the approximate cluster boundary by visual inspection. With 5 classes in our dataset, we are still able to distinguish where the cluster centers are, but we start realizing that one small region of the graph starts getting cluttered.

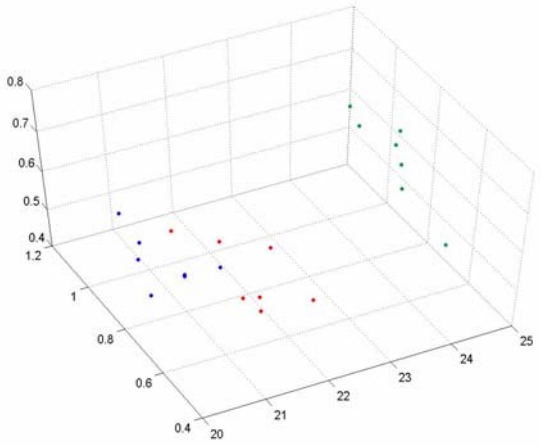


Figure 12: 3-Dimensional scatter plot of SVD input data containing 3 classes

This impression becomes more evident when we input the dataset into the network described in section 7 and start analyzing the classification rates for 3, 4, 5, 6, 7, 8, 9, 10, 20, 40 and 50 dimensions.

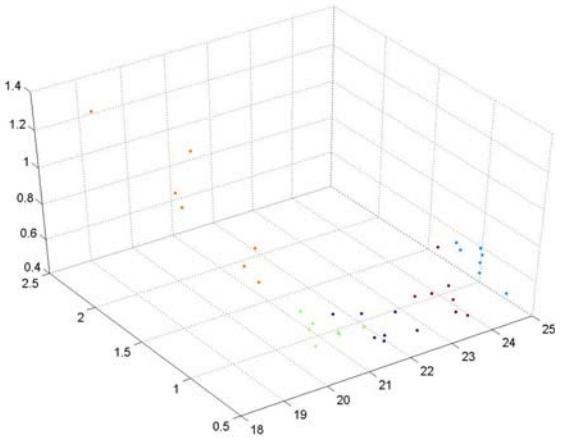


Figure 13: Same as above, 5 classes

The graph in figure 15 illustrates some points. The first is that although a high dimensional pattern is difficult to handle and process, a certain number of elements for the input pattern is necessary. In this case, 10 dimensions were always better than 3, except in the cases where the dataset had more than 20 elements. The other point is related to the visual information of figures 10, 11 and 12. As the number of classes and cases increase, the network has more difficulties in learning the proper discriminatory weights. In the tests we realized, when the number of classes was kept low (up to 6 classes) the network was able to reach the minimum squared error (MSE) goal within the specified number of epochs. As we increased the number of classes beyond 6, the MSE goal was not attained anymore, but the MSE kept decreasing until the maximum number of epochs was achieved. In this case, the classification rate followed and also decreased. We have the feeling that increasing the number of epochs may allow the network to eventually converge.

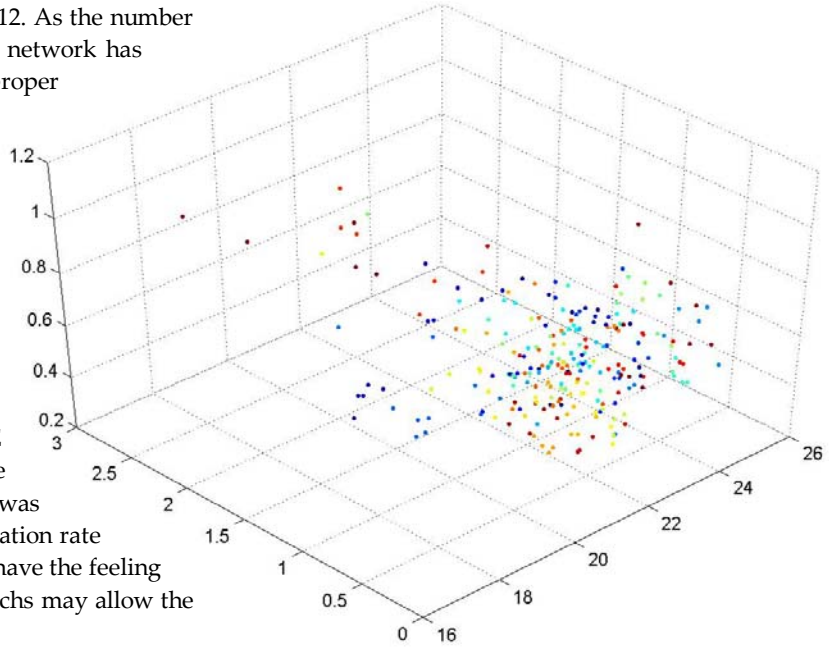
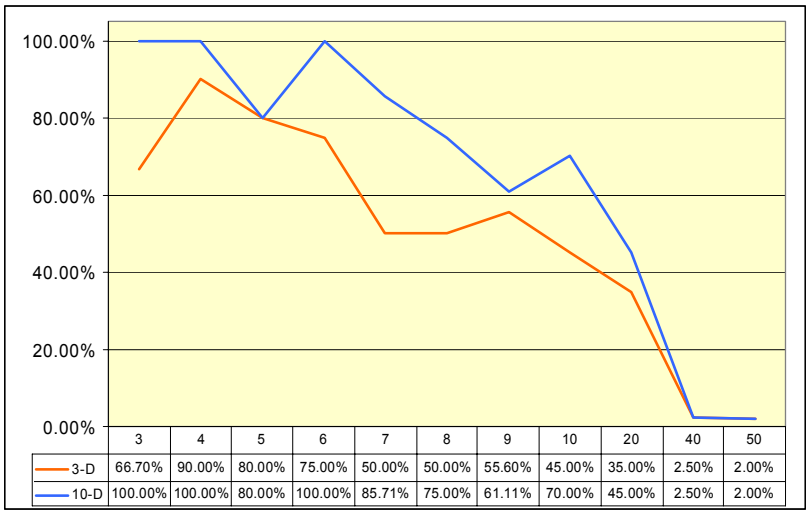


Figure 14: 50 classes, highly overlapped and cluttered

The more interesting fact happened when number of classes was higher than 20. In this case, the



	3	4	5	6	7	8	9	10	20	40	50
3-D	66.70%	90.00%	80.00%	75.00%	50.00%	50.00%	55.60%	45.00%	35.00%	2.50%	2.00%
10-D	100.00%	100.00%	80.00%	100.00%	85.71%	75.00%	61.11%	70.00%	45.00%	2.50%	2.00%

Figure 15: Classification rate chart for patterns pre-processed with SVD

network didn't achieve the MSE goal and very quickly it reached the minimum gradient parameter, meaning that learning was not being productive and the network was showing no improvement towards decreasing the minimum squared error. In this case, network performance was very mediocre, and looking at the confusion table for 50 classes only told us that every input pattern was classified as the same class. The network became biased towards only one class.

8.2. ICA Experiments

The second experiment with Backpropagation NN uses a dataset preprocessed using Independent Component Analysis (see section 5.3). The original *IrisBasis* image was 40x40 pixels, corresponding to a pattern vector of 1601 elements (including class label element). ICA was utilized to reduce the dimensionality of this dataset in three smaller datasets, with 3, 10 and 50 dimensions. We will compare the classification rates of these three datasets, and ultimately with classification rates obtained with input data pre-processed with SVD datasets.

Data Quality

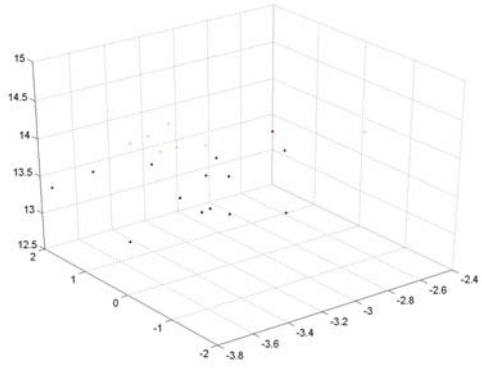
Analysis of 3 dimensional patterns using ICA pre-processing show that clusters are slightly more separated than those of SVD. Again, we plot here scattered charts of 3-dimensional patterns with 3, 5 and 50 classes, each class with 7 elements. Comparing the 50 classes' graphs of SVD and SVD, we see that values for SVD range from [16,26], [0,3] and [0.2,1.2], a solid of 30 cubic units. ICA produced a cluster space of 960 cubic units, in the ranges [-5,3], [10,16] and [-10,10].Although greater ranges are not as important as sharp cluster separation, it is already a sign that clusters may have more separation among them.

Network Training – 3 Dimension Example

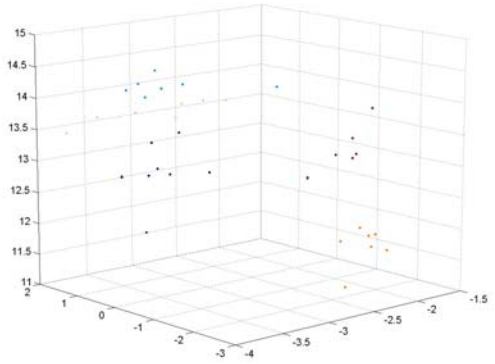
Following we present the results of classification of 3-Dimensional data prepared using ICA algorithm. The network utilized is the one designed in section 6 and 7.

Script session of training and classification follows.

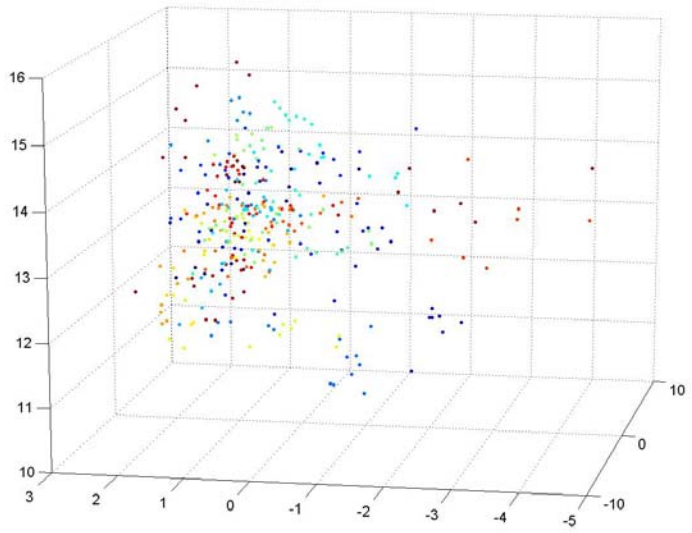
```
TS =

   -2.8640    1.2777   12.5889    1.0000
   -2.9219    1.7199   12.5172    1.0000
   -2.9494    1.4136   12.6834    1.0000
   -2.6692    0.6967   12.6527    1.0000
   -3.3305    0.6261   13.3149    1.0000
   -3.0384    0.6265   13.4116    1.0000
   -3.5804    0.8065   12.8197    1.0000
   -3.0959    0.5083   14.8158    2.0000
   -3.2020    0.7207   14.7179    2.0000
   -3.8060    0.1567   14.9651    2.0000
   -3.4906    1.0455   14.4934    2.0000
   -3.3923    0.7209   14.8060    2.0000
   -3.3070    0.7564   14.4880    2.0000
   -3.3881    0.6132   14.5679    2.0000
   -2.8283    1.2894   12.8812    3.0000
   -2.7378    1.7805   12.2221    3.0000
   -3.1117    1.4770   11.7951    3.0000
   -2.8311    1.4663   11.8054    3.0000
   -2.6279    1.6052   12.1349    3.0000
   -3.4197    1.5692   11.4336    3.0000
   -3.7768    1.7604   11.6028    3.0000

TRAINGDA, Epoch 0/50000, MSE 0.467242/5e-007, Gradient 0.316465/1e-
009
TRAINGDA, Epoch 1500/50000, MSE 0.0135942/5e-007, Gradient
0.0805861/1e-009
TRAINGDA, Epoch 4500/50000, MSE 0.00506083/5e-007, Gradient
0.0090971/1e-009
TRAINGDA, Epoch 7500/50000, MSE 0.00278679/5e-007, Gradient
0.00608816/1e-009
TRAINGDA, Epoch 10500/50000, MSE 0.00169937/5e-007, Gradient
0.0155874/1e-009
TRAINGDA, Epoch 13500/50000, MSE 0.00102513/5e-007, Gradient
0.00429258/1e-009
TRAINGDA, Epoch 16500/50000, MSE 0.000682525/5e-007, Gradient
0.00322378/1e-009
TRAINGDA, Epoch 19500/50000, MSE 0.000466332/5e-007, Gradient
0.00156951/1e-009
TRAINGDA, Epoch 22500/50000, MSE 0.000327278/5e-007, Gradient
0.00135353/1e-009
TRAINGDA, Epoch 25500/50000, MSE 0.000233602/5e-007, Gradient
0.00106353/1e-009
TRAINGDA, Epoch 28500/50000, MSE 0.00018053/5e-007, Gradient
0.0033436/1e-009
TRAINGDA, Epoch 31500/50000, MSE 0.000141906/5e-007, Gradient
0.00389465/1e-009
TRAINGDA, Epoch 34500/50000, MSE 9.52964e-005/5e-007, Gradient
0.00149548/1e-009
```



(a)



(b)



(c)

Figure 16: 3-Dimensional patterns pre-processed with Independent Component Analysis (ICA) – (a) Clusters of three classes, 7 samples each (b) Five classes, 7 samples each (c) Fifty classes, 7 samples each

```
TRAINGDA, Epoch 37500/50000, MSE 7.69979e-005/5e-007, Gradient
0.00210628/1e-009
TRAINGDA, Epoch 40500/50000, MSE 5.14733e-005/5e-007, Gradient
0.000210729/1e-009
TRAINGDA, Epoch 43500/50000, MSE 3.98173e-005/5e-007, Gradient
0.00051697/1e-009
TRAINGDA, Epoch 46500/50000, MSE 3.05609e-005/5e-007, Gradient
0.000453066/1e-009
TRAINGDA, Epoch 49500/50000, MSE 2.278e-005/5e-007, Gradient
0.000103497/1e-009
TRAINGDA, Epoch 50000/50000, MSE 2.17839e-005/5e-007, Gradient
8.52174e-005/1e-009
TRAINGDA, Maximum epoch reached, performance goal was not met.
class = 100
```

Classification rate of 100% on the classification of 3 classes of 3 dimensions.

We repeat the experiment for 5, 10, 50 and 100 classes.

5 Classes

```
TS =
   -2.8640    1.2777   12.5889    1.0000
   -2.9219    1.7199   12.5172    1.0000
   -2.9494    1.4136   12.6834    1.0000
   -2.6692    0.6967   12.6527    1.0000
   -3.3305    0.6261   13.3149    1.0000
   -3.0384    0.6265   13.4116    1.0000
   -3.5804    0.8065   12.8197    1.0000
   -3.5320   -0.1272   14.3211    2.0000
   -3.5393    0.0190   14.5835    2.0000
   -3.6124    0.5899   14.2146    2.0000
   -2.7127   -1.0967   14.2888    2.0000
   -3.5662    0.3894   14.3364    2.0000
   -3.5048    0.3569   14.1155    2.0000
   -3.4719   -0.5152   14.4167    2.0000
   -2.7858    0.5911   13.8751    3.0000
   -3.2954   -0.0743   14.0237    3.0000
   -3.0190   -0.5156   14.0726    3.0000
   -3.3452   -0.0341   13.8091    3.0000
   -3.4677    0.7145   13.8164    3.0000
   -3.6950    1.1517   13.7373    3.0000
   -3.7970    1.7171   13.4339    3.0000
   -2.5641   -2.3234   12.0185    4.0000
   -2.1742   -2.1767   11.9652    4.0000
   -2.0781   -2.1391   11.9493    4.0000
   -2.0433   -2.3396   11.7218    4.0000
   -2.2805   -2.1836   12.1220    4.0000
   -2.3761   -2.6459   11.9223    4.0000
   -2.2946   -1.8731   11.1135    4.0000
   -3.5990    0.1367   12.0472    5.0000
   -2.3262   -1.0105   12.7387    5.0000
   -1.9585   -1.3212   13.0025    5.0000
   -2.0231   -1.5385   13.1215    5.0000
   -1.5670   -0.9779   13.6690    5.0000
   -1.8867   -0.7663   12.9566    5.0000
   -2.0030   -1.4028   13.3437    5.0000
TRAINGDA, Epoch 0/50000, MSE 0.491738/5e-007, Gradient 0.457461/1e-
009
```

```
TRAINGDA, Epoch 799/50000, MSE 4.94541e-007/5e-007, Gradient
9.8422e-007/1e-009
TRAINGDA, Performance goal met.
class =  80
```

Classification rate of 80% on the classification of 5 classes of 3 dimensions.

10 Classes

```
TRAINGDA, Epoch 0/50000, MSE 0.491869/5e-007, Gradient 0.367179/1e-
009
TRAINGDA, Epoch 2572/50000, MSE 4.98162e-007/5e-007, Gradient
5.95151e-007/1e-009
TRAINGDA, Performance goal met.
class = 75
```

Classification rate of 75% on the classification of 10 classes of 3 dimensions.

20 Classes

```
TRAINGDA, Epoch 0/50000, MSE 0.491869/5e-007, Gradient 0.367179/1e-
009
TRAINGDA, Epoch 2572/50000, MSE 4.98162e-007/5e-007, Gradient
5.95151e-007/1e-009
TRAINGDA, Performance goal met.
class = 75
```

Classification rate of 75% on the classification of 10 classes of 3 dimensions.

40 Classes

```
TRAINGDA, Epoch 0/50000, MSE 0.491869/5e-007, Gradient 0.367179/1e-
009
TRAINGDA, Epoch 2572/50000, MSE 4.98162e-007/5e-007, Gradient
5.95151e-007/1e-009
TRAINGDA, Performance goal met.
class = 75
```

Classification rate of 75% on the classification of 10 classes of 3 dimensions.

108 Classes

```
TRAINGDA, Epoch 0/50000, MSE 0.491869/5e-007, Gradient 0.367179/1e-
009
TRAINGDA, Epoch 2572/50000, MSE 4.98162e-007/5e-007, Gradient
5.95151e-007/1e-009
TRAINGDA, Performance goal met.
class = 75
```

Classification rate of 75% on the classification of 10 classes of 3 dimensions.

Final Classification Results

We also performed classification experiments with datasets of 10 and 50 dimensions, and the results are presented below:

	Dimensions		
	3	10	50
3 classes	**100.0%**	**100.0%**	**100.0%**
5 classes	**80.0%**	**90.0%**	**100.0%**
10 classes	**75.0%**	**90.0%**	**95.0%**
20 classes	**62.5%**	**70.0%**	**95.0%**
40 classes	**2.5%**	**68.8%**	**95.0%**
108 classes	**0.9%**	**0.9%**	**92.1%**

9. Final Conclusions

In this paper we proposed a number of different techniques for image segmentation for isolation of the iris and posterior extraction of fundamental classification data. We faced some problems inherent of imaging the Human Eye, as interference of eyelashes and eyelids over the iris. Depending on the illumination scheme, light speckles may also affect the amount of epigenetic material visible to the classification system.

The algorithm designed to extract the fundamental information of the iris is also a resampling algorithm, as it is possible to choose an output dimension of the picture (*IrisBasis*) smaller than its original size and the algorithm will use a 3x3 averaging mask to perform resampling. We must keep in mind that the average iris diameter noted in the CASIA Iris database is 230 pixels, and the iris strip can vary anything between 30 pixels and 80 pixels wide. If we resample down the number of pixels too much, we will lose vital epigenetic information and have worst classification results. In our tests, we used 40x40 pixels *IrisBasis* images. Transforming such image into a patter vector leaves us with a vector of 1601 elements per instance (including the class label element). This vector still contains a great number of redundant information, and pre-processing methods must be applied. In this paper we chose Singular Value Decomposition (SVD) and Independent Component Analysis (ICA) for comparison purposes. We used native Matlab SVD algorithm, while we used Hyvärinen and Oja [14, 15] FastICA algorithm.

Although we got much better classification results by using ICA instead of SVD, we were still not able to get satisfactory results for a large number of classes. We were able to observe that SVD did not provide good results because separation between clusters was poor. Figure 10 and 9 serve as additional evidence that ICA provides more degrees of freedom over SVD. Still, there are some advantages of SVD, namely performance and the ability to choose the number of dimensions after the SVD algorithm is run. For problems where number of classes is small (up to 6), the use of SVD is acceptable. Unfortunately this type of recognition problem demands a very high number of classes.

ICA demanded more computing power to reduce the dimensionality of the input pattern, and in extremely high dimension cases (100x100 images or 10,000 elements input pattern) the calculation of the covariance matrix showed extremely poor results on a Pentium IV 2.8GHz and 1GByte of RAM memory, forcing us to resample the input image to 40x40. The advantage or ICA was also obvious, as classification results were very good with the 50 dimensions experiment.

We finish this paper by stating that this architecture of NN is probably not the best for iris recognition: In real world applications, the number of classes (individuals) may be in the order of millions (i.e.:

airport arrivals control) and inclusion of new individuals should be fast and easy.

The problem of Iris Recognition is not simple, and although we got good results on the classification, it was only when we used 50-dimension input vectors that we started achieving these results. Performance is always an issue in real time applications.

REFERENCES

[1] Daugman, J., *How Iris Recognition Works,* IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, Number 1, January 2004.

[2] Masek L., *Recognition of Human Iris Patterns for Biometric Identification,* [http://www.csse.uwa.edu.au/~pk/studentprojects/libor].

[3] P. Kronfeld, *Gross anatomy and embryology of the eye.* In *The Eye*, H. Davson, Ed. London, U.K.: Academic, 1962.

[4] Daugman, J., *Biometric Personal Identification System based on Iris Analysis,* United States Patent, Patent Number 5,291,560, March 1994.

[5] *CASIA iris image database*, Institute of Automation, Chinese Academy of Sciences, [http://www.sinobiometrics.com]

[6] R. Wildes. *Iris recognition: an emerging biometric technology*. Proceedings of the IEEE, Vol. 85, No. 9, 1997.

[7] Herbert Freeman. *Computer processing of line-drawing images*. Computing Surveys, 6(1):57-97, March 1974.

[8] W. Kong, D. Zhang. *Accurate iris segmentation based on novel reflection and eyelash detection model*. Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong, 2001.

[9] W. W. Boles, *A security system based on human iris identification using wavelet transform*, Engineering Applications of Artificial Intelligence, 11:77-85, 1998.

[10] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, New York, 1995.

[11] A. K. Jain, R. P. W. Duin, J. Mao, *Statistical Pattern Recognition: A Review,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 1, January 2000.

[12] R. Swiniarski, *Independent Component Analysis,* CS553 Class Notes, San Diego State University, 2004.

[13] S. A. Abdallah, M. D. Plumbey, *If the independent component of natural images are edges, what are the independent components of natural souds?*, 3rd International Conference on Independent Component Analysis and Blind Source Separation, San Diego, California, December 2001.

[14] The FastICA Matlab package, *[http://www.cis.hut.fi/projects/ica/fastica]*

[15] A. Hyvärinen, E. Oja, *Independent Component Analysis – A Tutorial*, Helsinki University of Tecnology, Espoo, Finland, April 1999.

[16] G. W. Awcock, R. Thomas, *Applied Image Processing*, McGraw Hill, 1996.

[17] K. R. Castleman, *Digital Image Processing,* Prentice Hall, New Jersey, 1996.

[18] Sonia Leach, *Singular Value Decomposition – A Primer*, Department of Computer Science, Brown University, Providence

APPENDIX A – MATLAB SOURCE CODE

FUNCTION PUPILFINDER

```
function [cx,cy,rx,ry]=pupilfinder(imagename)
% USE:                    [cx,cy,rx,ry]=pupilfinder(imagename)
%
% Name:                   pupilfinder
%
%
% Author:          Padu (Paulo Merloti)
%                                padu@ieee.org
%
%
% Date:                   v.1     03/24/04
%
% Arguments:       imagename: is the input image of an human iris
%
% Purpose:
%                  perform image segmentation and finds the center and two
%                  (vertical and horizontal) radius of the iris pupil
%
% Dependencies:None
%
% Example:         [cx,cy,rx,ry]=pupilfinder('image.bmp')
%                  cx and cy is the position of the center of the pupil
%                  rx and ry is the horizontal radius and vertical radius of the pupil
%
G=imread(imagename);
bw_70=(G>70);
bw_labeled=bwlabel(~bw_70,8);
mr=max(bw_labeled);
regions=max(mr);
for i=1:regions
   [r,c]=find(bw_labeled==i);
   if size(r,1) < 2500

      region_size=size(r,1);

      for j=1:size(c,1)
            bw_labeled(r(j),c(j))=0;
      end;
   end;
end;
bw_pupil=bwlabel(bw_labeled,8);

%get centroid of the pupil
stats=regionprops(bw_pupil,'centroid');
ctx=stats.Centroid(1);
cty=stats.Centroid(2);

hor_center = bw_pupil(round(cty),:);
ver_center = bw_pupil(:,round(ctx));

%from the horizontal center line, get only the left half
left=hor_center(1:round(ctx));
%then flip horizontally
left=fliplr(left);
%get the position of the first pixel with value 0 (out of pupil bounds)
left_out=min(find(left==0));
```

```
%finally calculate the left pupil edge position
left_x = round(ctx-left_out);

%from the horizontal center line, get only the right half
right=hor_center(round(ctx):size(G,2));
%get the position of the first pixel with value 0 (out of pupil bounds)
right_out=min(find(right==0));
%finally calculate the left pupil edge position
right_x = round(ctx+right_out);

%adjust horizontal center and radius
rx = round((right_x-left_x)/2);
cx = left_x+rx;

%from the vertical center line, get only the upper half
top=ver_center(1:round(cty));
%then flip horizontally
top=flipud(top);
%get the position of the first pixel with value 0 (out of pupil bounds)
top_out=min(find(top==0));
%finally calculate the left pupil edge position
top_y = round(cty-top_out);

%from the vertical center line, get only the upper half
bot=ver_center(round(cty):size(G,1));
%get the position of the first pixel with value 0 (out of pupil bounds)
bot_out=min(find(bot==0));
%finally calculate the left pupil edge position
bot_y = round(cty+bot_out);

%adjust horizontal center and radius
ry = round((bot_y-top_y)/2);
cy = top_y+ry;
```

PUPILTESTER

Load an iris image and uses the function **pupilfinder** to find the pupil center and its radiuses (horizontal and vertical). Plots a circle and lines to best visualize these parameters

```
fname='001_1_1.bmp';

F=imread(fname);

colormap('gray');
imagesc(F);
hold;

[cx,cy,rx,ry]=pupilfinder(fname);

%plot horizontal line
x=[cx-rx*2 cx+rx*2];
y=[cy cy];
plot(x,y,'y');

%plot vertical line
x=[cx cx];
y=[cy-ry*2 cy+ry*2];
plot(x,y,'y');

circle([cx cy], rx, 1000, 'y');
```

PUPILVALIDATOR

Helps manually test the pupilfinder function. The program cycles through all the images of the database and present the iris image with indicative lines showing center and radius of the pupil. An input dialog asks if the lines are centered with the pupil and it counts the number of correct instances against the misclassifications. A list of failed classifications is stored in the "fail" matrix for posterior analysis.

```
%set base directory of iris database
d='D:\MasterCS\CS553\IrisDatabase\CASIA';
clc;
%read class directories (each directory contains images of one class)
files=dir(d);

colormap('gray');
counter = zeros(1,3);
fail=[];

for i = 1:size(files,1)
   if not(strcmp(files(i).name,'.')|strcmp(files(i).name,'..'))

      for s=1:2
         %validate
         classdir = [d, '\', files(i).name, '\', mat2str(s)];
         irisFiles = dir(classdir);
         for j=1:size(irisFiles,1)
            if not(strcmp(irisFiles(j).name,'.')
            |strcmp(irisFiles(j).name,'..'))
               irisFileName = [classdir, '\', irisFiles(j).name];

               F=imread(irisFileName);
               imagesc(F);
               title(irisFiles(j).name);
               colormap('gray');
               hold on;

               [cx,cy,rx,ry]=pupilfinder(irisFileName);

               %plot horizontal line
               x=[cx-rx*2 cx+rx*2];
               y=[cy cy];
               plot(x,y,'y');

               %plot vertical line
               x=[cx cx];
               y=[cy-ry*2 cy+ry*2];
               plot(x,y,'y');

                        circle([cx cy], rx, 1000, 'y');

               dialog_title = irisFiles(j).name;
               dialog_prompt = 'Are the lines aligned?';
               n_lines = 1;
               def = {'Y','N'};
               result=inputdlg(dialog_prompt,dialog_title,n_lines, def);

               counter(1) = counter(1) + 1;
               if strcmp(result,'Y')
                  counter(2) = counter(2) + 1;
               else
                  if strcmp(result,'N')
                     counter(3) = counter(3) + 1;
                     fail = [fail;irisFiles(j).name];
```

```
                    else
                       return;
                    end
                end

                counter
                hold off;
                clf reset;
            end
        end
    end

  end
end
```

IRISFINDER

```
function [right_x,right_y,left_x,left_y]=irisfinder(imagename)
% USE:                    [rx,ry,lx,ly]=irisfinder(imagename)
%
% Name:                   irisfinder
%
%
% Author:          Padu (Paulo Merloti)
%                                 padu@ieee.org
%
%
% Date:                   v.1     04/20/04
%
% Arguments:       imagename: is the input image of an human iris
%
% Purpose:
%                  perform image segmentation and finds the edgepoints of
%                  the iris at the horizontal line that crosses the center
%                  of the pupil
%
% Dependencies:None
%
% Example:         [rx,ry,lx,ly]=irisfinder('image.bmp')
%                  rx and ry is the edge point of the iris on the right side
%                  lx and ly is the edge point of the iris on the left side
%

%read bitmap
F=imread(imagename);

%find pupil center and radius
[cx,cy,rx,ry]=pupilfinder(imagename);

% Apply linear contrast filter
D=double(F);
G=uint8(D*1.4-20);

%obtain the horizontal line that passes through the iris center
l=G(cy,:);

margin = 10;

% Right side of the pupil
R=l(cx+rx+margin:size(l,2));
[right_x,avgs]=findirisedge(R);
```

```
right_x=cx+rx+margin+right_x;
right_y=cy;

% Left side of the pupil
L=l(1:cx-rx-margin);
L=fliplr(L);
[left_x,avgs]=findirisedge(L);
left_x=cx-rx-margin-left_x;
left_y=cy;
```

FINDIRISEDGE

```
function [x,avgs]=findirisedge(V)
% USE:                    [x,avgs]=findirisedge(V)
%
% Name:                   findirisedge
%
%
% Author:          Padu (Paulo Merloti)
%                                 padu@ieee.org
%
%
% Date:                   v.1     03/31/04
%
% Arguments:       V: is intensity vector that starts on the pupil fringe and
%                  go outbound
%
% Purpose:
%                  given a vector of intensities, divide it in small
%                  windows and return the point where a sudden raise has
%                  occurred
%
% Dependencies:None
%
% Example:         [x,avg]=findirisedge(LINE)
%                         x is the point of sudden change (rapid increase)
%                         avg is the vector V transformed with windowed average
%                         values
%

windowSize = 15;
i=1;
avgs=zeros(1,size(V,2));

while i < (size(V,2)-windowSize)
   w=V(i:i+windowSize-1);
   avgs(i:i+windowSize-1)=mean(w);
   i = i+windowSize;
end

w=V(i:size(V,2));
avgs(i:size(V,2))=mean(w);

x=size(avgs,2);
for i=1:size(avgs,2)-1
   dif=avgs(i+1)-avgs(i);
   if (dif>10)&(i>25)
      x=i;
      break;
   end
end
```

IRISFIGURE

Reproduces the internal working of "IrisFinder" algorithm and produces visual information (plots) along the way.

```
imagename='D:\MasterCS\CS553\Matlab\001_1_1.bmp';

%read and plot image
F=imread(imagename);

colormap('gray');

%-------------------- original image
[cx,cy,rx,ry]=pupilfinder(imagename);

D=double(F);
cl_x=[0 320];
cl_y=[cy cy];

subplot(2,2,1);
imagesc(F);
hold on;
l=F(cy,:);
plot(l,'r');
plot(cl_x, cl_y, 'y');
title('Original picture');
axis([0 320 0 280]);
hold on;

[right_x,right_y,left_x,left_y]=irisfinder(imagename);

G=uint8(D*1.4-20);

%-------------------- image with contrast changed (x 1.4)
subplot(2,2,2);
imagesc(G);
hold on;
l=G(cy,:);
plot(l,'r');
plot(cl_x, cl_y, 'y');
axis([0 320 0 280]);
title('Contrast stretched (1.4 factor)');
hold on;

%-------------------- center line (yellow) intensities of the right
%-------------------- side of the pupil
margin = 10;
R=l(cx+rx+margin:size(l,2));
subplot(2,2,3);
plot(R);
axis([0 90 100 220]);
title('Right Side of Pupil');


%-------------------- averages of windows - right side
[rex,avgs]=findirisedge(R);

divx=[rex rex];
divy=[0 220];

subplot(2,2,4);
plot(avgs);
axis([0 90 100 220]);
```

```
title('averages of windows (R)');
rex=cx+rx+margin+rex;

%-------------------- plot division lines
hold;
plot(divx,divy,'r');
axis([0 90 100 220]);

subplot(2,2,3);
hold;
plot(divx,divy,'r');
axis([0 90 100 220]);
```

IRISTESTER

Cycles through all images of the database and gives the final count of segmentation successes or failures.

```
%set base directory of iris database
d='D:\MasterCS\CS553\Iris Database\CASIA';
clc;
%read class directories (each directory contains images of one class)
files=dir(d);

colormap('gray');
counter = zeros(1,3);
fail=[];

for i = 1:size(files,1)
   if not(strcmp(files(i).name,'.')|strcmp(files(i).name,'..'))

      for s=1:2
         %validate
         classdir = [d, '\', files(i).name, '\', mat2str(s)];
         irisFiles = dir(classdir);
         for j=1:size(irisFiles,1)
            if not(strcmp(irisFiles(j).name,'.')|strcmp(irisFiles(j).name,'..'))
               irisFileName = [classdir, '\', irisFiles(j).name]

               F=imread(irisFileName);
               imagesc(F);
               title(irisFiles(j).name);
               colormap('gray');
               hold on;

               [right_x,right_y,left_x,left_y]=irisfinder(irisFileName);
               [cx,cy,rx,ry]=pupilfinder(irisFileName);

               %plot line on the right side
               x=[right_x right_x];
               y=[1 280];
               plot(x,y,'r');

               %plot line on the left side
               x=[left_x left_x];
               y=[1 280];
               plot(x,y,'r');

               dialog_title = irisFiles(j).name;
               dialog_prompt = 'Are the lines aligned?';
               n_lines = 1;
               def = {'Y','N'};
               result = inputdlg(dialog_prompt, dialog_title, n_lines, def);
```

```
                counter(1) = counter(1) + 1;
                if strcmp(result,'Y')
                   counter(2) = counter(2) + 1;
                else
                   if strcmp(result,'N')
                      counter(3) = counter(3) + 1;
                      fail = [fail;irisFiles(j).name];
                   else
                      return;
                   end
                end

                counter
                hold off;
                clf reset;
             end
          end
       end

   end
end
```

IRISBASIS

Given a gray iris image, locates the fundamental points of the image (pupil center, iris edges) and resample only the

```
function [IB]=irisbasis(imagename,NCOLS,NROWS,useAvg)
% USE:                    [IB]=irisbasis(imagename)
%
% Name:                   irisbasis
%
%
% Author:          Padu (Paulo Merloti)
%                                 padu@ieee.org
%
%
% Date:                   v.1     04/22/04
%
% Arguments:       imagename: is the input image of an human iris
%                                 NCOLS: number of columns per side of the iris
%                                 NROWS: number of rows to transfer from top to bottom
of
%                                        pupil
%                   useAvg: 0 - don't use average matrix
%                           1 - use average matrix
%
% Purpose:
%                                 given the iris image, extracts the irisBasis image
%                    the irisBasis is a selection of pixels on the left and
%                    right side of the pupil
%
% Dependencies:      pupilfinder, irisfinder
%
% Example:         [IB]=irisbasis('image.bmp',50,100)
%                                 IB is the resulting image matrix of NROWS x 2*NCOLS
%                                 containing essential image of the iris in file
%                    'image.bmp. NCOLS is the number of columns per side of
%                    the iris, therefore 50,100 will result an irisbasis of
%                    100x100 elements

%read image
F=imread(imagename);
```

```
%find pupil information
[cx,cy,rx,ry]=pupilfinder(imagename);

%find iris edges
[right_x,right_y,left_x,left_y]=irisfinder(imagename);

%base extraction lines
height = 2 * ry;
baseline = height / (NROWS-1);
row = (cy-ry);

ibr=[];
ibl=[];
ib=[];

width_r = right_x-cx-rx;
width_l = cx-rx-left_x;

for i=1:NROWS
    %right side of the pupil
    start_posR=abs(sqrt(rx^2-(row-cy)^2))+cx;
    rs=fix(start_posR);
    re=fix(start_posR+width_r);
    copyv=linspace(rs,re,NCOLS);
    L=zeros(1,NCOLS);
    for c=1:NCOLS
        if (useAvg==1)
            M=F(fix(row)-1:fix(row)+1,fix(copyv(c))-1:fix(copyv(c))+1);
            %pure mean is too blurring
            L(c)=double((mean(mean(M)))+double(F(fix(row),fix(copyv(c)))))/2;
        else
            L(c)=F(fix(row),fix(copyv(c)));
        end
    end
    ibr=[ibr; L];

    %left side of the pupil
    start_posL=-abs(sqrt(rx^2-(row-cy)^2))+cx;
    rs=fix(start_posL-width_l);
    re=fix(start_posL);
    copyv=linspace(rs,re,NCOLS);
    L=zeros(1,NCOLS);
    for c=1:NCOLS
        if (useAvg==1)
            M=F(fix(row)-1:fix(row)+1,fix(copyv(c))-1:fix(copyv(c))+1);
            L(c)=double((mean(mean(M)))+double(F(fix(row),fix(copyv(c)))))/2;
        else
            L(c)=F(fix(row),fix(copyv(c)));
        end
    end
    ibl=[ibl; L];

    row = row + baseline;
end;
IB=[ibl ibr];
```

IRISBASISFIGURE

Used to visualize how algorithm works.

```
%irisBasis construction and visualization
imagename='001_1_1.bmp';
```

```
%read image
F=imread(imagename);

colormap('gray');
subplot(2,1,1);
imagesc(F);
hold on;

%find pupil information
[cx,cy,rx,ry]=pupilfinder(imagename);

%plot center line
plot(cx,cy,'rh-');
plot([1 size(F,2)],[cy cy],'r:');
plot(cx,cy-ry,'rv');
plot(cx,cy+ry,'r^');

%find iris edges
[right_x,right_y,left_x,left_y]=irisfinder(imagename);

%plot iris edges
plot(right_x,right_y,'r<');
plot(left_x,left_y,'r>');

%plot base extraction lines
NROWS=200;
NCOLS=100;
height = 2 * ry;
baseline = height / (NROWS-1);
row = (cy-ry);

ibr=[];
ibl=[];
ib=[];
width = right_x-cx-rx;

for i=1:NROWS
    start_posR=abs(sqrt(rx^2-(row-cy)^2))+cx;
    plot([start_posR start_posR + width],[row row],'b:');
    rs=fix(start_posR);
    re=fix(start_posR+width);
    ibr=[ibr; F(fix(row),rs:re)];

    start_posL=-abs(sqrt(rx^2-(row-cy)^2))+cx;
    plot([start_posL start_posL - width],[row row],'c:');
    rs=fix(start_posL-width);
    re=fix(start_posL);
    ibl=[ibl; F(fix(row),rs:re)];

    row = row + baseline;
end;

ib=[ibl ibr];
subplot(2,1,2);
imagesc(ib);
```

IRISVIEWALL

Cycle through all irises in the database, extract the *irisbasis* image and save it in one only directory as a jpeg file.

```
clc;

%set base directory of iris database
d='D:\MasterCS\CS553\Iris Database\CASIA';
destDir = 'D:\MasterCS\CS553\Iris Database\IrisBasisAll';

%read class directories (each directory contains images of one class)
files=dir(d);
size(files)

for i = 1:size(files,1)
   if not(strcmp(files(i).name,'.')|strcmp(files(i).name,'..'))

      for s=1:2
         %validate
         classdir = [d, '\', files(i).name, '\', mat2str(s)];
         irisFiles = dir(classdir);
         for j=1:size(irisFiles,1)
            if not(strcmp(irisFiles(j).name,'.')|strcmp(irisFiles(j).name,'..'))
               irisFileName = [classdir, '\', irisFiles(j).name]

               F=imread(irisFileName);

               [cx,cy,rx,ry]=pupilfinder(irisFileName);
               [right_x,right_y,left_x,left_y]=irisfinder(irisFileName);
               [IB]=irisbasis(irisFileName,50,100);

               %write file
               irisBasisFile = [destDir,'\',irisFiles(j).name(1:7),'.jpg'];
               imwrite(uint8(IB),irisBasisFile,'jpg');
            end
         end
      end

   end
end
```

IRISBASISTOPATTERN

Given a directory containing a list of irisbasis images of size *nxm*, converts them into a dataset of patterns, each pattern with one row and (*nxm*) columns + 1. The last column contains the pattern number, extracted from the three first letters of the file name. Pixel intensity is scaled to the [0,1] range.

The dataset is saved to one only file in the Matlab format.

```
clc;
clear;

%set base directory of irisBasis directory
irisDir  = 'D:\MasterCS\CS553\Iris Database\IrisBasisAll';
destDir  = 'D:\MasterCS\CS553\Iris Database\IrisBasisPattern';
clc;

T=[];

irisFiles = dir(irisDir);
for i=1:size(irisFiles,1)
  if not(strcmp(irisFiles(i).name,'.')|strcmp(irisFiles(i).name,'..'))
    irisFileName = [irisDir, '\', irisFiles(i).name];

    F=imread(irisFileName);
    G=im2double(F);
    P=[];
```

```
    for j=1:size(F,2)
        P = [P G(j,:)];
    end
    P=[P str2num(irisFiles(i).name(1:3))];

    T=[T;P];

    irisFileName
    [size(P) size(T)]
  end
end
save destDir T –MAT
```

IRISBASISTOSVDPATTERN

Given a directory containing a list of irisbasis images of size *nxm*, converts them into a dataset of reduced patterns by using Matlab's SVD (singular value decomposition) funtion. The last column contains the pattern number, extracted from the three first letters of the file name. Pixel intensity is scaled to the [0,1] range.

```
clc;
clear;

%set base directory of irisBasis directory
irisDir  = 'D:\MasterCS\CS553\IrisDatabase\IrisBasisAll40';
clc;

T=[];

irisFiles = dir(irisDir);
for i=1:size(irisFiles,1)
  if not(strcmp(irisFiles(i).name,'.')|strcmp(irisFiles(i).name,'..'))
    irisFileName = [irisDir, '\', irisFiles(i).name];

    F=imread(irisFileName);
    G=im2double(F);

    %perform singular value decomposition
    xpattern=svd(G);

    P=[xpattern' str2num(irisFiles(i).name(1:3))];

    T=[T;P];

    irisFileName
    [size(P) size(T)]
  end
end


save irisBasisSVD T –ASCII
```

TESTICA, TESTICA10, TESTICA50

Reduces the dimension to 3, 10 and 50 of an input pattern created from the *IrisBasis*.

TESTICA.M

```
clear;
clc;
load irisBasisVector1600
```

```
I=irisBasisVector1600(:,1:1600);
I=I';

[icasig3, A, W] = fastica(I,'lastEig', 3, 'numOfIC', 3);

icasig3=icasig3';
icasig3=[icasig3 irisBasisVector1600(:,1601)];

save icasig3 -ascii;
```

TESTICA10.M

```
clear;
clc;
load irisBasisVector1600

I=irisBasisVector1600(:,1:1600);
I=I';

[icasig10, A, W] = fastica(I,'lastEig', 10, 'numOfIC', 10);

icasig10=icasig10';
icasig10=[icasig10 irisBasisVector1600(:,1601)];

save icasig10 icasig10 -ascii;
```

TESTICA50.M

```
clear;
clc;
load irisBasisVector1600

I=irisBasisVector1600(:,1:1600);
I=I';

[icasig50, A, W] = fastica(I,'lastEig',50, 'numOfIC', 50);

icasig50=icasig50';
icasig50=[icasig50 irisBasisVector1600(:,1601)];

save icasig50 icasig50 -ascii;
```

NN_SVD

Given a dataset pre-processed with SVD, separates it in training and testing dataset, encode class label column, designs and trains a Feedforward Backpropagation of Error ANN and performs classification test.

```
clear;
clc;

load irisBasisSVD -ascii;

%get only first 3 dimensions'
nclasses=50;
TS=[irisBasisSVD(1:nclasses*7,1:3) irisBasisSVD(1:nclasses*7,41)];

%display TS (full dataset)
TS

%display scatter points
scatter3(TS(:,1),TS(:,2),TS(:,3),8,TS(:,4),'filled');

%pause;

%form training set with the first 5 instances of each class
Training=[];
```

```
for i=1:nclasses
    Training=[Training;TS((i*7)-6:(i*7)-2,:)];
end;

scatter3(Training(:,1),Training(:,2),Training(:,3),8,Training(:,4),'filled');

%form the pattern matrix (patterns in columns - no class information)
P=Training';
P=P(1:3,:);

%get class column
targetTr=Training(:,4);

%convert sequential numbered classes to power of two:
%  class 1 = 1
%  class 2 = 2
%  class 3 = 4
targetDec=2.^(targetTr-1);

%convert decimal to binary
%  class 1 = 001
%  class 2 = 010
%  class 3 = 100
targetBin=dec2bin(targetDec);

%separate in columns
targetClass=[];
for i=1:nclasses
    targetClass=[targetClass double(str2num(targetBin(:,i)))];
end;

%transpose
T=targetClass';

%------------------------------------------------------------ training

S1 = 300;     % Number of neurons in the first hidden  layer - changes according to
test
S2 = nclasses;     % Number of neurons in the output  layer

net=newff(minmax(P),[S1 S2],{'logsig' 'logsig'}, 'traingda');

%setup network parameters
net.trainFcn = 'traingda';          % Training function type
net.trainParam.lr = 0.1;           % Learning rate
net.trainParam.lr_inc = 1.05;       % Increment of a learning rate
net.trainParam.show = 300;         % Frequency of progress displays (in epochs).
net.trainParam.epochs = 50000;       % Maximum number of epochs to train.
net.trainParam.goal = 0.0000005;  % Mean-squared error goal.
net.trainParam.min_grad=0.000000001;

net=init(net);

net=train(net,P,T);

inter=sim(net,P);

%find out winner class
[Y,I]=max(inter);

%pause;
```

```
%------------------------------------------------------------ simulation
%form testing set with the remaining 2 instances of each class
Test=[];
for i=1:nclasses
    Test=[Test;TS((i*7)-1:(i*7),:)];
end;


scatter3(Test(:,1),Test(:,2),Test(:,3),8,Test(:,4),'filled');

%form the pattern matrix (patterns in columns - no class information)
P=Test';
P=P(1:3,:);

%get class column
targetTs=Test(:,4);

%convert sequential numbered classes to power of two:
%  class 1 = 1
%  class 2 = 2
%  class 3 = 4
targetDec=2.^(targetTs-1);

%convert decimal to binary
%  class 1 = 001
%  class 2 = 010
%  class 3 = 100
targetBin=dec2bin(targetDec);

%separate in columns
targetClass=[];
for i=1:nclasses
    targetClass=[targetClass double(str2num(targetBin(:,i)))];
end;

%transpose
T=targetClass';

%perform network simulation
a=sim(net,P);

%find out winner class
[Y,I]=max(a);

I=I';
c=nclasses-I;
res=log2(2.^c)+1;
match=[targetTs res targetTs-res];


%correctly classified patterns in percentage - 100%=perfect match
class=(size(find(match(:,3)==0),1)/size(targetTs,1))*100
```

NN_ICA_3D, NN_ICA_10D, NN_ICA_50D

Given a dataset pre-processed with ICA in 3, 10 or 50 dimensions, separates it in training and testing dataset, encode class label column, designs and trains a Feedforward Backpropagation of Error ANN and performs classification test.

```
clear;
clc;
```

```
load icasig50 -ascii;

%get only first 50 dimensions'
nclasses=108;
TS=[icasig50(1:nclasses*7,1:50) icasig50(1:nclasses*7,51)];

%display TS (full dataset)
TS

%form training set with the first 5 instances of each class
Training=[];
for i=1:nclasses
    Training=[Training;TS((i*7)-6:(i*7)-2,:)];
end;

%form the pattern matrix (patterns in columns - no class information)
P=Training';
P=P(1:50,:);

%get class column
targetTr=Training(:,51);

%convert sequential numbered classes to power of two:
%  class 1 = 1
%  class 2 = 2
%  class 3 = 4
targetDec=2.^(targetTr-1);

%convert decimal to binary
%  class 1 = 001
%  class 2 = 010
%  class 3 = 100
targetBin=dec2bin(targetDec);

%separate in columns
targetClass=[];
for i=1:nclasses
    targetClass=[targetClass double(str2num(targetBin(:,i)))];
end;

%transpose
T=targetClass';

%------------------------------------------------------------ training

S1 = 300;     % Number of neurons in the first hidden  layer - changes according to
test
S2 = nclasses;     % Number of neurons in the output  layer

net=newff(minmax(P),[S1 S2],{'logsig' 'logsig'}, 'traingda');

%setup network parameters
net.trainFcn = 'traingda';          % Training function type
net.trainParam.lr = 0.1;           % Learning rate
net.trainParam.lr_inc = 1.05;       % Increment of a learning rate
net.trainParam.show = 500;         % Frequency of progress displays (in epochs).
net.trainParam.epochs = 50000;       % Maximum number of epochs to train.
net.trainParam.goal = 0.0000005;  % Mean-squared error goal.
net.trainParam.min_grad=0.000000001;

net=init(net);
net=train(net,P,T);
```

```
inter=sim(net,P);

%find out winner class
[Y,I]=max(inter);

%------------------------------------------------------------ simulation
%form testing set with the remaining 2 instances of each class
Test=[];
for i=1:nclasses
    Test=[Test;TS((i*7)-1:(i*7),:)];
end;

%form the pattern matrix (patterns in columns - no class information)
P=Test';
P=P(1:50,:);

%get class column
targetTs=Test(:,51);

%convert sequential numbered classes to power of two:
%  class 1 = 1
%  class 2 = 2
%  class 3 = 4
targetDec=2.^(targetTs-1);

%convert decimal to binary
%  class 1 = 001
%  class 2 = 010
%  class 3 = 100
targetBin=dec2bin(targetDec);

%separate in columns
targetClass=[];
for i=1:nclasses
    targetClass=[targetClass double(str2num(targetBin(:,i)))];
end;

%transpose
T=targetClass';

%perform network simulation
a=sim(net,P);

%find out winner class
[Y,I]=max(a);

I=I';
c=nclasses-I;
res=log2(2.^c)+1;
match=[targetTs res targetTs-res];

%correctly classified patterns in percentage - 100%=perfect match
class=(size(find(match(:,3)==0),1)/size(targetTs,1))*100
```