A Neural Network Based Facial Feature Detector

Sophia Sameera Corsava

MSc Information Technology: Knowledge Based Systems

Department of Artificial Intelligence

University of Edinburgh

1993

Abstract

The principal goal of this project is to determine the likely positions of small complex features i.e. eyes, noses and mouths from intensity image data.

There are many image processing and pattern recognition systems, traditional and non-traditional. Most of them are memory consuming and need elaborate and expensive hardware. There is, however, a very big interest in these systems not only because humans need a definite improvement of pictorial information for their interpretation of image density data, but also because today's robots, have to be able to perceive their environment reliably and this has been proven feasible with the aid of such systems.

A specific area, that of facial recognition, was examined in this project, using artificial neural nets instead of the traditional pattern recognition methods.

Three pre-procesing techniques have been used in order to ensure that the produced results are reliable.

If the net topology is the optimum, and the preprocessing technique produces consistent and concise data, the net can recognise and identify the position of facial characteristics successfully.

Acknowledgements

I would like to thank my supervisor, Dr. Bob Fisher. A big thanks to Andrew Fitzgibbon for his help with matlab and the image processing part and his patience and last but not least Erdinc Dermenioglou, who provided me with the database of faces, was extremely helpful and gave me a lot of information about the entire mechanism of his project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

There is an increasing need for face recognition systems, within security facilities, in big companies, or in other places where the identity of the people entering or leaving is important. The human face is a dynamic three dimensional structure and it has been proved that traditional pattern recognition techniques fail to do the task efficiently and effectively.

The goal of this project is to combine connectionist methods with computer vision and digital image processing techniques so as to achieve the recognition of facial features, and as a verification to determine their position on the facial image.

There are already various systems used for recognising individual facial features or whole faces, like WISARD [9], or others that recognise numbers like Cognitron[7] Neocognitron[8]etc.

The most common approach has been to extract from the configuration of facial features some geometric details and then to compare these with already stored information in a database, examples include Kaya and Kobayashi[25] and Harmon.[13]

These methods, however, have not been efficient enough for commercial use, because they are either not reliable or are memory consuming and hardware expensive.

Connectionist methods have been applied before with quite successful results, which were more reliable than those obtained by other techniques in the cases of complex 2-D patterns. [13] [12] [9] [24]

Connectionist computing methods [15] [16] [14] have been proven successful in this task, but there are some disadvantages, such as the net easily forgetting already known data and taking some time to learn new ones.

This kind of problem is attributed to the usually complex structure of the neural net that is used for the recognition task and its large number of input units. In this project an attempt was made to reduce to the minimum these problems by decreasing significantly the number of inputs to the net.

This was achieved by pre-processing the image to be used as input to the net which will make some distinct areas i.e. the small complex features, 'visible' to the net and will effectively reduce the dimensionality of data.

This project investigates the neural net facial feature recognition approach in combination with three preprocessing techniques prior to entering the patterns to the neural net. More specifically these techniques are :

1. eigenvectors and alpha values

2. ft analysis along with phase angles in radians of the resulting complex matrix. The angles lie between - π and π.

3. ft analysis, with reduced frequencies.

These methods are explained further in Chapter three.

The neural net with which we experiment is the Multilayer Perceptron and it was chosen because we believe it can perform the recognition task more efficiently and effectively.

1.2 Previous Recognition Techniques

There have been quite a few attempts to automate face recognition procedures.

Examples include Goldstein *et al*, (1971,1972), who produced interesting results, Harmon et al, (1981), Takahashi et al (1990). All of them were successful in performing the recognition task, but they could not quite automate the entire procedure and the results were not as reliable as those of the connectionists. In some cases, template matching was applied and whenever the template was not correctly placed, it was adjusted manually.

Connectionist methods, on the other hand have been more successful in automating the entire procedure. Examples include Kohonen et al (1981), who attempted to do face recognition using whole images of faces without elaborate preprocessing.

WISARD [9] detected facial features with the purpose of helping the face recognition task in general.

Cognitron [7] is a self organised multi-layered neural net by Fukushima, was quite successful as well in recognition tasks, while Neocognitron [8] is a better version that can recognise patterns, without being affected by position shifts.

Vincent *et al* [12], compared traditional recognition techniques with connectionist methods, and proved that the latter were more successful, although not 100%.

Turk and Pentland [13] were also successful in recognising faces. They treated the face image as a vector and they performed eigenvector analysis on a set of facial vectors as such. They then selected the best eigenvectors by principle component analysis and they proved that their method had a big success rate.

The research reported here differs for these results in that we are looking to identify and locate individual features, such as left eyes, rather than complete faces.

1.3 Pattern Recognition

Pattern recognition is closely linked with perception and cognition. The main aim is to extract, identify, classify and describe the patterns in data gathered from real and simulated environments. [18] [16]

Scientists have been studying the human and the animal perception systems, so as to find evidence of their function [22]. It is very important that we understand how the perceptual system is structured and exactly how it works in order to be able to copy it and apply it to robots and machines [28].

This apparent lack of ability to understand the nervous system in depth has not prevented people from making perceptual models based on mathematics that provide the theoretical basis for classifier design.

Several measurements are required in order to be able to adequately distinguish inputs that belong to different categories. If we take n-measurements from the input pattern each of which is a unique feature we can then create a set of these features that form a feature vector.

There are statistical pattern recognition techniques that cannot handle very well the structural information about the interconnections in a complex pattern. Of them, the Bayesian classification technique is a powerful tool. The Problem Reduction Representations seem to work in certain cases only, and the neural nets and the genetic algorithms have succeeded repeatedly in pattern recognition tasks.[25]

It is very important to find the necessary parameters while designing a pattern recognition system. Usually these parameters are unknown and this is why non-numeric or non-parametric systems are used to find a suitable density function. A very interesting method could be estimating this function from the sample patterns.

Classifiers rely on distance metrics and probability theory. The distance metrics can be found by calculating the Hamming distance, the Eucledian distance, the city-block distance, or the square distance.[17] [15] [11]

An essential step is to train the classifiers; this can be done in two ways, either by supervised learning or unsupervised. In the first case the classifier is presented with the output that it has to produce. A problem that can occur is due to the big demand for training patterns which grows exponentially with the number of dimensions in the pattern space.

In the case of unsupervised learning, different clustering algorithms are used that seek to find clusters and/or natural groupings in the data. An example is the hierarchical clustering method, which happens to be quite complicated.

For the scope of this project supervised learning has been preferred.

The three preprocessing techniques were carefully selected so as to maintain as much as possible valuable information for the patterns to be recognised. In the case of the eigenvectors, a principal components analysis was applied in order to decide the most important of them. In the case of the fast fourier transform phase angles, the vector's direction was kept intact, and finally in the case of the fast fourier transform reduced frequencies, the frequence filtering was carefully done in order to produce a reconstructed image that would be similar to the original one. The result of this careful selection of preprocessing techniques was that the neural net was trained relatively quickly and the identification and recognition were successful.

1.4 Gray Level Representation

A way to deal with classification problems is to use an automatic scene analysis technique such as the gray level representation. [27]

Suppose that we have to analyse a black and white image. This can be represented as a real-valued function of two variables i.e f(x,y). The intensity of the of the image is the gray level or brightness. This fact enables us to represent an image in the computer. An image function to be suitable for image processing must be digitised both spatially and in amplitude. Samples of the digitised image have to be acquired and this is done by digitising the spatial co-ordinates (x,y). The amplitude digitisation is the gray-level quantisation [26].

The image is eventually represented by an array of integers, where each member of the array or pixel specifies its approximate gray level for a corresponding cell. Usually the quantities are an integer power of 2. $N = 2^n$ and $G = 2^m$, where G is the number of gray levels and N x N is the size of the image. The discrete levels are equally spaced between 0 and 255 in the gray scale. The number of bits required to store a digitised image is given by the formula bits = N x N x m.

The resolution of a picture depends on N and m. The more these values increase the better the image becomes. This implies that the storage and the processing requirements increase as a function of N and m. However, if the resolution is very low, in an effort to decrease those two, we have the checker-board effect. [27]

In order to obtain better results with this method it is necessary to normalise the image first, because there might be too many dark or very bright regions and subsequently the partitioning of the image into sets of distinct gray levels might lead to wrong decisions on behalf of the neural net. Low values in the function represent dark levels of intensity while high values represent bright ones.

The key decision with gray level representation is to select which gray level transitions are significant, because there is the danger of one being unable to reach efficient and effective solutions.

The number of gray levels for this project was decided to be 256, because we tried to maintain an acceptable quality in terms of resolution.

1.5 Artificial Neural Nets

According to neuroscience, from the study of the spatial behaviour of mammals, the spatial orientation and navigation involve the combination of sensory information along with the organism's own measurements with respect to the sources of these stimuli. More specifically spatial cognition requires the computation of either previous memories or the ones newly generated that are being experienced directly from the organism. The assumption is that such an ability must involve the internal manipulation of spatial representations, something similar to the tensorial matrix theory in mathematics. It is assumed that there is a mechanism which makes transformations from one sensory space to another. This function

also serves as a general model for the ability to foresee the consequences of particular actions in specific contexts. This ability of the human nervous system is valuable, although not very well understood. It is this exact ability that the scientists are trying to copy by having introduced the concept of the artificial neural nets. [28]

It has been proved that neural nets are very good at recognising faces. There are of course drawbacks in using them, like the net forgetting the learned patterns when trained with new ones, or sometimes being overtrained, but they are indeed very important since they can be used to perform tasks for which computational algorithms may not exist. they change and adapt their behaviour dynamically and they have a distributed nature. which can work on fault tolerant hardware. It is also important not to overtrain the net, because then instead of generalising it memorises the patterns and we want to avoid exaclty that.

The neural nets learn from example and they do not require vast amounts of memory, depending off course on the network size. The innovation here is that the net does not have anywhere an explicit model of the small complex feature that it is supposed to recognise but it has retained in memory its reaction to the feature recognition. [11]

The need to find a simple learning rule is urgent. The good behaviour - the behaviour we want - should be reinforced, while the undesirable one should be reprimanded [14]. This idea is transfered to the neural nets. Of the various types of neural nets, in this project we deal with the Multilayer Perceptron (MLP).

The MLP was chosen, because it learns fast, it does not forget easily, and as all neural nets, its distributed processing nature makes it fault tolerant. MLPs can recognise noisy patterns quite successfully. The structure and properties of the Multilayer Perceptron are explained further in Chapter Four.

1.6 Project Overview

The figures that follow figure 1–1 figure 1–2, show graphically the structure of the project. During *Phase-I* the images are normalised, the subimages containing individual features are extracted and the data files are prepared according to the three preprocessing techniques. These are described in detail in Chapters two and three. When the three data files are prepared then the net training takes place as well as various tests are performed in order to find the net configuration that will learn the training patters fast and with a small error per unit. This procedure is explained further in Chapter four. The outcome of Phase-I is the three trained nets.

During *Phase-II* the postprocessing and verification procedure take place. We select two faces that are unknown to the net. These faces are scanned and subimages are extracted starting from the first pixel until the last one. The scanning program extracts these windows as follows:

- Extract a 12x30 or a 16x32 window [1] starting from pixel Number one
- Extract a 12x30 or a 16x32 window starting from pixel Number two, and so on until you reach the last pixel

When the image scanning is complete the extracted windows are processed according to each of the three methods : eigenvectors principal components analysis and alpha values, [2] the fast fourier transform phase angles or else ft-I, and finally the fast fourier transform reduced frequencies or else ft-II.

The outcome of each of these methods is a data file containing input patterns to be fed to the corresponding trained classifier. The outcome of Phase-II is three files containing the probabilities assinged by the each of the nets and the projection

---

[1] The window/subimage dimensions depend on the preprocessing technique to be used but more about this in Chapter 3

[2] From now we shall refer to this technique as the alpha values method

of these on the actual image. So for each image there are three sets of results that show graphically the probabilities assigned by the nets for each pixel. More information about them is given in Chapter five and in Appendix A where all the net results are kept.

1.7 Chapter Overview

The thesis is organised into five chapters :

Chapter one refers to background information about Pattern recognition, previous recognition techniques, gray level representation and its significance in this project and finally, to a brief summary about artificial neural nets.

Chapter two describes the structure of the multilayer perceptron and its properties, the Matlab program and the Input Data morphology.

Chapter three explains analytically the preprocessing methods,

Chapter four refers to the net configurations, training and results

Chapter five explains in depth the post-processing and verification procedures and finally

Chapter six summarises the conclusions and talks about future work.

At the end of Chapter five there is a detailed appendix illustrating the net results graphically while at the end of the thesis in Appendix B there are all the matlab programs that were used for this project.

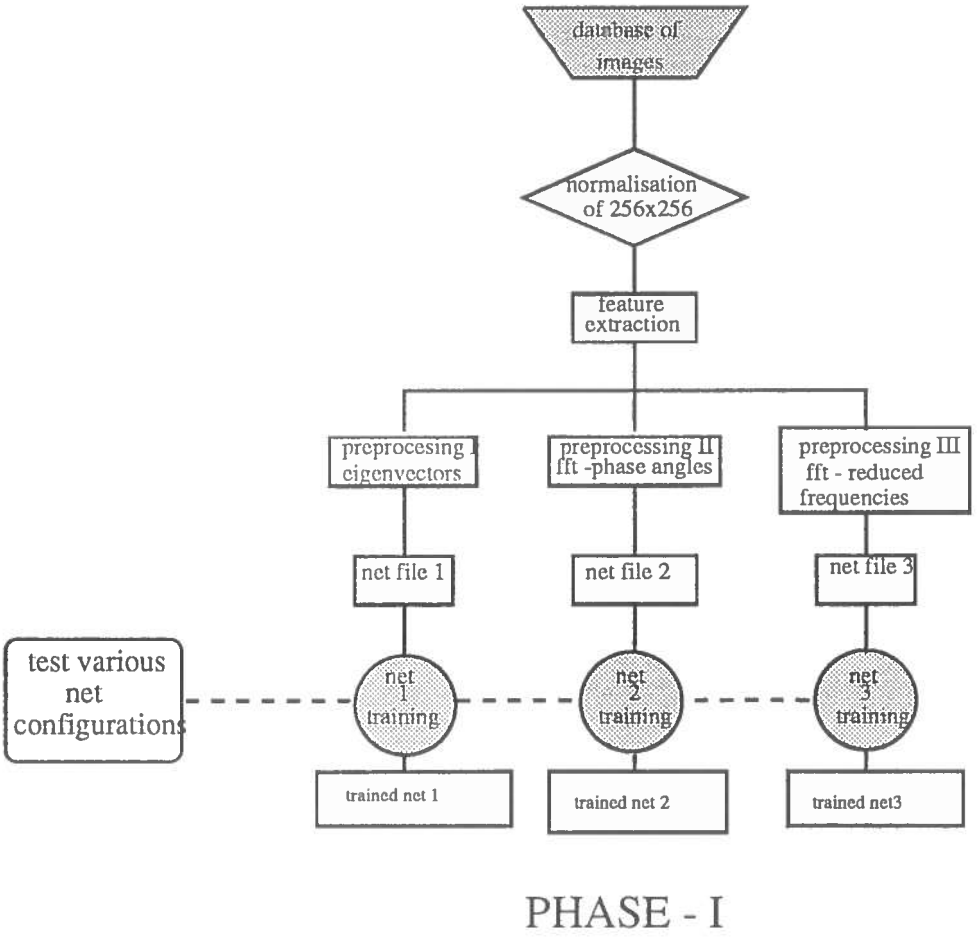
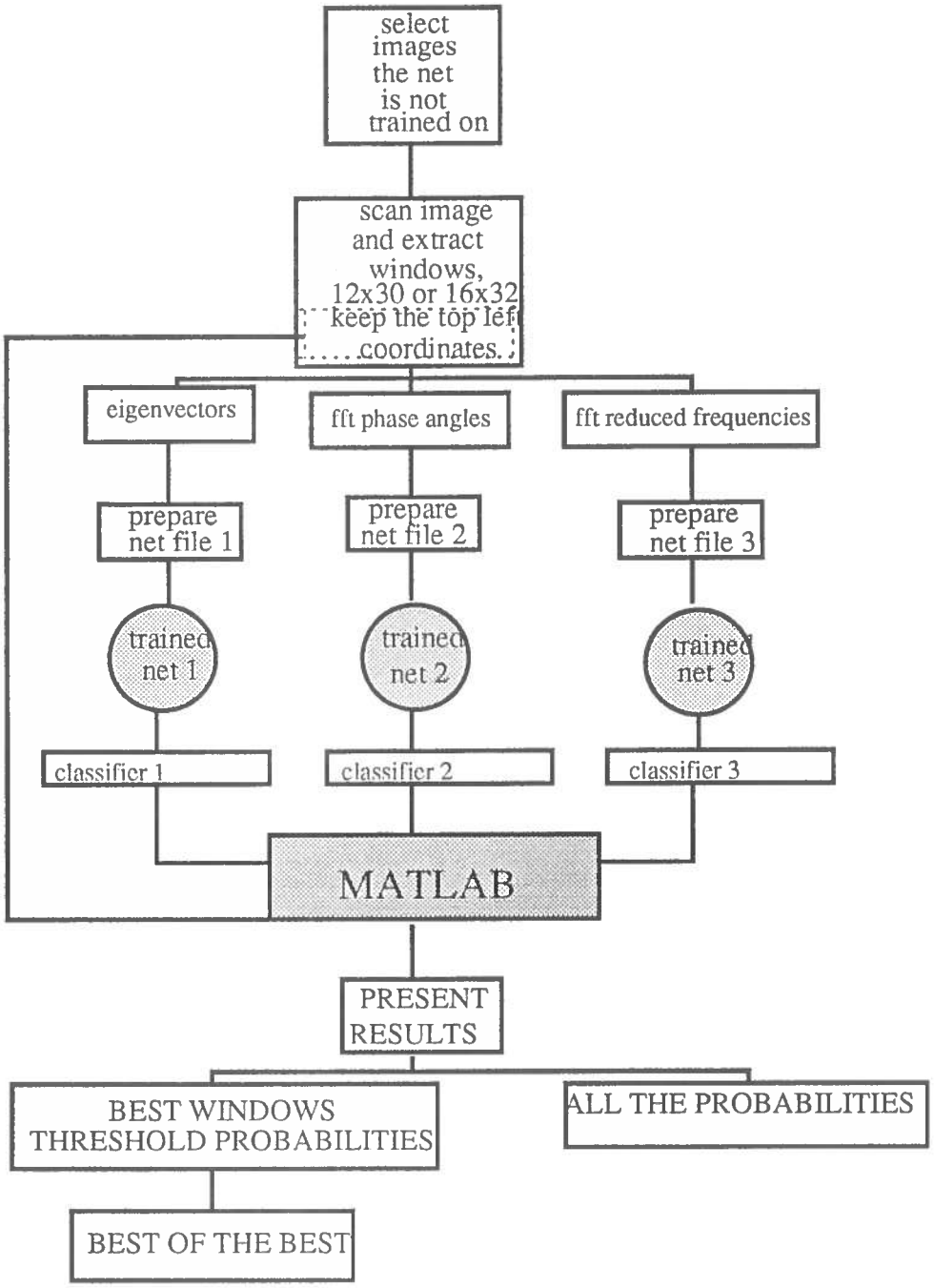


Figure 1–1: Conceptual Project Model - Phase I



PHASE - II

Figure 1–2: Conceptual Project Model - Phase II

Chapter 2

The Multilayer Perceptron - MLP

2.1 Structure and Properties of the MLP

The Multilayer Perceptron has three layers : an input layer, a second layer called hidden layer and finally the third one called output layer.[17] The units in the second and the third layer are perceptron units, crude models of neurons that function as thresholding units with 1 to n inputs each either 1 or 0 denoted by $x_1 \cdots x_n$. Each input is modified by the weight factor w_i so as to produce the desired output. If the actual output of the neural net is similar or the same as [1] the desired output then the learning is complete.

The perceptron cannot solve linearly inseparable problems while the MLP can cope very well with such cases. Also the Multilayer Perceptron is fault and noise tolerant and it has the ability to generalise what it has learned quite successfully. For these reasons we chose to use this net for this project, since it happens to be one of the net structures that has been successful in recognition tasks.

MLPs are used nowadays for a number of scientific and commercial applications, like NeTalk.

The function the MLP computes looks like :

$V_g^{\lambda}(u) = g(\sum_k w_{jk} V_k \lambda - 1(u))$

w_{jk} = weights g = sigmoidal threshold $V_k^{\lambda}(u)$ = value of k^{th} unit = λ^{th} logarithm on input signal u.

---

[1] Most of the times this is impossible

$V_k^1(u) = M_k = k\text{th input } \lambda = 1, 2, 3$

The thresholding function is the sigmoid one. The input layer distributes the values it receives to the hidden layer.[11]

The following figure 2–1 shows the structure of the MLP.

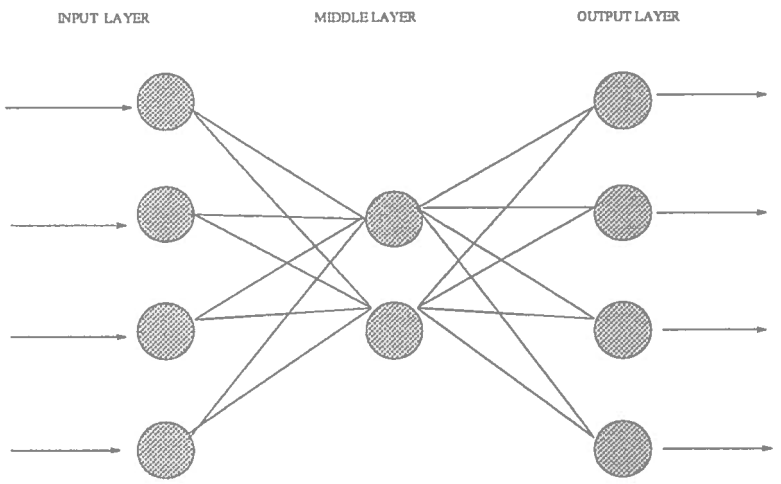


Figure 2–1: The Multilayer Perceptron

The learning rule for the MLP is the generalised delta rule or else back propagation [15] [17] [10] [11]. According to this rule, the weights of the net are adjusted until the net reaches a satisfactory level of learning the input pattern. The algorithm continuously compares the actual output with the desired one and adjusts the weights accordingly.

This is done with the help of an error function. The calculated error is back propagated from one layer to the previous one. It is through this error propagation that the net finds out about the erroneously learned pattern and thus modifies the weights until learning is achieved. If the error is big, the weight has to be altered significantly, while if the error is small, the weight adjustment has to be small as well.

This method of learning is called supervised learning, because the net has been instructed in advance to associate a given input to a given output.

The sigmoidal was selected as the thresholding function because it retains a lot of information. This function looks like :

$$a_i = g(\sum_{j=0}^{n} w_{ij} a_j) \tag{2.1}$$

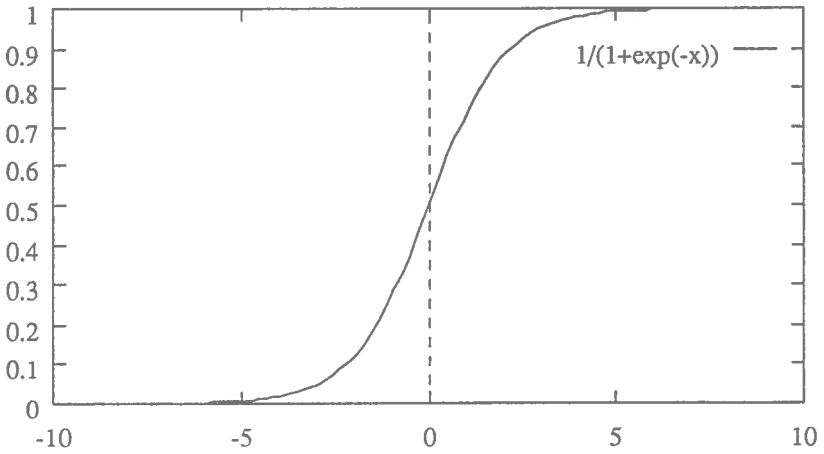


Figure 2–2: The sigmoid function

The *rbp* program in the Departmental machines was used in order to train the MLP. Rbp stands for *recurrent back propagation* and it was simulating this procedure i.e. it was back propagating the error from the output layer to the middle layer and then to the input layer until the pattern was learned as best as possible. Throughout this back propagation the weights on the arcs among the three layers were accordingly adgusted in order to reach a point at the weight space where the error was small. The only net configuration that had difficulties in getting out of the local minima was that of the ft reduced frequencies. This problem was tackled by changing the seed, and therfore the initial weights, as well as by modifying the number of the hidden units.

The recurrent back propagation usually gets stuck to local minima or maxima within the weight space. A small positive constant called *momentum* and symbolised by a is used to give the net energy boosts and in this way help it avoid situations such as these. The goal is to reach a global minimum, since the error per unit never reaches 0.

When such a minimum is reached then the net has learned the patterns as best as possible.

2.2 Matlab

MATLAB is a technical computing environment for high performance numeric computation and visualisation. It integrates numerical analysis, matrix manipulation, signal processing and graphics in a friendly environment, where traditional programming is not required; mathematical expressions are expressed as they are written [19]

The algorithms that do all the preprocessing and the postprocessing were written in matlab , which was proved to be invaluable and fast for the required complicated calculations.

2.3 Input Data and its Properties

The pictures were taken by Erdinc Dermenioglou, who tried to maintain the same light conditions, and the same distance from the camera with all the subjects during the image capturing. For this purpose a CCTV camera with a digitiser and SUN 3 workstation were used. The background lighting was minimised. [24]

Each picture was initially 512x512 pixels, each pixel 8 bits, and the gray levels totalled 256. Later on for practical reasons and to save memory the image resolution was reduced in half, which means that in the end the entire experimentation and the net result verification was done on 256x256 images.

Erdinc used ISIS as an image processing tool and a script to generate and store the captured images. Because of the technique he used with the script program (a bounding box placed around the image area to be stored), some images were smaller than 512x512. These images have not been used for this project.

The lighting was provided by two fluorescent tubes mounted on either side of the camera on the lighting rig, while the subject was sitting on a stool placed at one metre distance from the camera. He used the fluorescent tubes to provide a correctly illuminated environment and he also examined the gray level histograms of the produced images in order to make sure that they were correct.

The idea was that from these images subimages with 12x30 or 16x32 dimensions were to be extracted depending on the preprocessing technique. These windows contain the facial features individually, ie a 12x30 window contains the left eye only.

Based on these images, the centre of the window to be extracted had to be located first. This was mainly done manually, either with the use of the ISIS software, or with the xv program. The centre of that window was decided by moving the cursor to an appropriate central pixel for that window and by recording the x,y cooridnates in a text file which would be later used for the entire extraction procedure.

A crucial point was to decide where the centre of each individual feature would be. For the left and the right eyes the centre was in the pupil, in order to facilitate the net training. The window containing an eye does not include any extra pixels from around the eye. The centre for the nose was decided to be the left nostril, first because it was more prominent in most of the images and secondly because the size of the nose did not allow for a window to contain it. The centre for the mouth, was the geometrical centre of a 12x30 rectangle and it was located in between the lips.

The file containing the grouped facial features is consisted of 223 samples, which amounts to circa 45 samples of a kind. Each group has a threefold set of each facial feature, ie the left eye in three orientations, upright, tilted to the left and tilted to the right. These different orientations were considered critical to the success of this project since the neural net had to be able to cope with the various eye orientations and sizes. The same applies for the rest of the features as well.

A program that reads in the face image produces an output file with the gray level value of each pixel in numerical format.

The images are normalised before any preprocessing starts. Normalisation is discussed extensively in Chapter 3.

The extraction program given the coordinates of the centre of a 12x30 window or a 16x32 only in the case of the ft filtered data, extracts the facial features from the normalised HIPS image and gives the centered subimages which are used during the preprocessing. The images have been given numbers according to the

order they were taken, and a character as a specification of their orientation. For example the filename corrseponding to an upright normalised image that was taken tenth would be 10.norm.

As previously mentioned the same image was taken in two different orientations as well, one with the head tilted on the left and another on the right. These filenames would be 10.r.norm for the right orientation and 10.l.norm for the left.

The extracted features before they are compiled together to a single file, are kept in individual files, that are named according to this 'sorting code' as well. The left eye files begin with 'le.', then follows the number of the picture they were extracted from and in the end, is the 'norm' extension, which indicates their status (whether normalised or not).

A similar method had been followed for the rest of the features, only that the first part of the name varies according to the feature the file holds. For example the right eye files start with "ri.", the nose files with "nos.", the mouth files with "m." and the non examples with "non." This classification code was proved very efficient and effective since it saved a lot of time, throughout the project. I did not have to remember which file belongs to which image, by just looking at the name of the extracted image I knew what it represented.

Each set of files was kept in a separate directory, along with the coordinates files and all the rest of the relevant files.

All the shell programs were kept in the /bin/sun4 directory, from where the invocation was very easily done from any part of my home directory.

All the major procedures i.e. generation of gray levels matrix, feature extraction, normalisation and file compilation, were done by shell programs which were invoking either the C programs or the matlab functions. So instead of giving the coordinates to the extract program separately, there was a shell program that was reading in a file of coordinates along with the actual name of the file they refer to, and producing a series of extracted images, whose number depended on the user specifications.

For example a typical entry would be:

```
genle 1 24
```

The result of this would be the generation of gray level representations for images numbered 1 to 24 and the gray levels would be kept in files for further use.

In the end the only time-consuming operation was the training of the neural net which was took 12 to 20 hours, depending on the net configuration.

2.4 Summary

In this chapter we discussed the MLP structure and functions, the matlab package and the input data and its properties.

Chapter 3

Preprocessing Techniques

3.1 Normalisation Algorithm

The input images were normalised prior to being fed to the neural net. This had to be done because the normalised image, enhanced the features, and improved the image statistics. The neural net was then able to identify and locate more accurately facial features. In general, the homogenous picture obtained by the normalisation process gave better results when it went through the project mechanism than the non-normalised ones.

The normalisation was done with a c program called 'normalise.c'. This algorithm takes as input the HIPS image and treats it as a normally distributed matrix of integers, with variance σ^2 and mean M In order to produce the normalised image, it subtracts from each pixel the mean, then it divides this result by σ. This normalises the image. However, in order to resolve the 0-255 distribution it adds 3 and finally it multiplies the entire product with 42.6. This had to done as such because it produced a better normalised image.

So for an image of N pixels $P_i, i = 1...N$ the mean M is given by :

$$M = \frac{1}{N}\sum_{i=1}^{N} P_i \tag{3.1}$$

The variance σ^2 is given by:

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N} \cdot (P_i - M)^2 \tag{3.2}$$

the normalised intensity is given by:

$$P_i = \frac{P_i - M}{\sigma} \tag{3.3}$$

and the Final Image is :

$$F_i = 42.6(P_i + 3) \tag{3.4}$$

where M is the mean intensity, P is the pixels, and σ is the standard deviation.

The image is treated as a normal distribution and the normalisation is done on each picture separately. In this way the normalisation results improve the image globally on an individual basis.

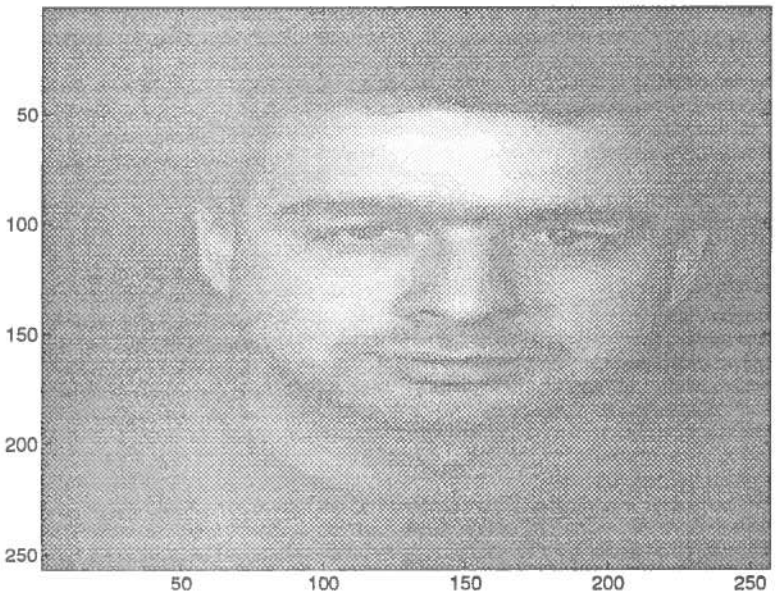


Figure 3–1: Normalised Image

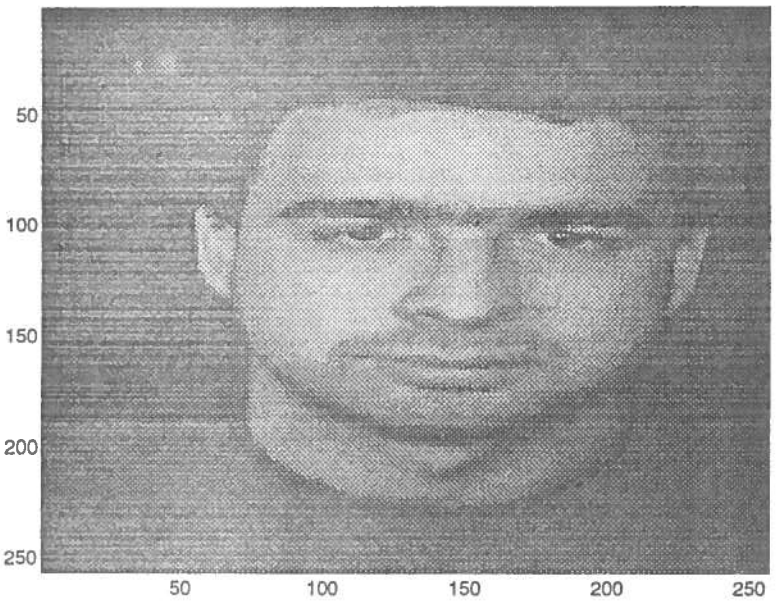


Figure 3–2: Raw Image

3.2 Preprocessing Overview

Before feeding the net with data, three preprocessing techniques were applied so as to enable us to make a constructive comparison of the results for those three methods. As it mentioned before, the preprocessing was done in order to reduce the data dimensionality and subsequently the net complexity. The alpha values method and the fourier transform reduced frequencies acheived that, the first by reducing it to 60 from the original 360 and the second to 114 from 512. The fourier transform phase angles did not attempt to do it, but instead it was used as a means to prove that even if less than the original dimensions are used the net results are quite good.

It is the preprocessing that decides the number of input units to the neural net.

The method that gave the most accurate results was the phase angles technique, as was expected, since all the frequencies were used, thus reducing the input noise to the net (see section 3.4).

The second best was the eigenvector analysis technique (see section 3.3), and the technique that gave less accurate results among the three was the ft reduced frequencies (see section 3.5). These results are explicitely illustrated in Appendix A.

There are three matlab programs that are dedicated to the preprocessing of the net data.

Each algorithm first reads the file that contains all the extracted facial features together in groups. These features are extracted from the normalised facial images and they have size equal to 360 pixels. Each feature is read as a 360 vector long since the extracted images were 12x30 i.e 360 pixels, and as such it represents a single pattern for the neural net. Therefore each facial feature ie left eye, right eye, nose and mouth, is a 360 vector that is preprocessed prior to being fed to the net.

In the case of the eigenvector analysis, the optimum number of vector components was 60.

In the case of the fourier analysis (ft) all the 360 frequencies were used. The program that was doing the fourier transform to the set of data was deriving from the generated complex matrix the 'angle' data, in reality the phase angles in radians. These angles lie between - π and π. This had to be done in this way because the multilayer perceptron does not accept as input complex numbers; therefore a meaningful representation had to be found for the net.

The ft method was used again as a third preprocessing technique. This time a reduction in the data dimensionality was attempted through the utilisation of a smaller number of frequencies for each facial feature.

While the maximum number of frequencies was 512, in this case only 57 of them were used. The algorithm was filtering the number of frequencies above a certain limit in an effort to determine the optimum number to be used for representing eventually the image to the neural net. After a lot of experimentation it was concluded that 57 was the optimum number of frequencies to be used in order for the net to receive a meaningful representation.

The files produced by the preprocessing procedure are then fed to the corresponding neural nets that have been created according to the preprocessing specifications. At this stage, the neural net gets trained, while various configurations are tested for better results. The following figure 3–3 represents graphically the pre-processing methodology

To summarise the preprocessing stage was as follows:

- Normalise the 256x256 facial images
- Extract sequential subimages/windows starting from the first pixel until the last one. The coordinates of this pixel denoted the top left coordinates of the extracted window whose size was either 12x30 for the alpha values and the phase angles techniques or 16x32 for the ft reduced frequencies. Each extracted window was represented as a vector either 360 long for the first two cases or 512 for the last.
- In the alpha values case, find the eigenvectors of the 12x30 extracted windows, normalise them, apply principal components analysis, select the ones with the bigger eigenvalues and then prepare the file for the net training.

- In the case of the ft-phase angles, do the fourier transform of the 12x30 extracted windows, find the phase angles in radians and then prepare the file for the net.

- In the case of the ft reduced frequencies, do the fourier transform of the extracted 16x32 windows, select the thresholding value for the frequencies, separate the real from the imaginary part, put in a file the the combined thresholded matrix and prepare the file for the net.

3.3 Eigenvectors and Reduced Dimensionality

One of the major preprocessing techniques was the eigenvector analysis along with a subsequent reduced data dimensionality. This proved to be the second best method when it was coming to pattern recognition and identification on behalf of the net.

A technique very similar to Principal Components Analysis has been used quite successfully before in the face and small features recognition task, by M. Turk and A. Pentland. [13]

This method actually reduces the data dimensionality so that a suitable linear transformation of the co-ordinate system is found, such that the data variance is small and subsequently these dimensions can be ignored. Principal Components Analysis is in reality a way to make a concise representation of a set of data points. The best solution is represented by the eigenvectors corresponding to the larger eigenvalues. The eigenvectors taken in the order of size of the eigenvalues are the solutions. Note that the eigenvectors have to be orthogonal among themselves which means that $x_a^T x_b = 0$. The dimensional reduction is achieved by ignoring the eigenvectors that correspond to small eigenvalues.[15]

An important issue in this method was the speed of the entire procedure, not only throughout the preprocessing stage, where the net had to be trained, but also afterwards when the net had to classify the presented patterns.

The eigenvector analysis was accomplished much faster using the Turk and Pentland approach, in comparison to the traditional principal components analysis technique that was taking overnight to run. It was not considered practical to pursue this methodology, not only because of the actual preprocessing stage, but because of the postprocessing one, during which the net would have to be fed with extracted images from the entire face as a set of patterns. The number of these patterns was 256^2, since the scanned facial images used were 256x256.

The reason for this significant increase in speed was the different approach for the covariance matrix :

For the traditional Principal Components Analysis technique :

$C = M^T M/(m-1)$

The entry in the i-th row and j-th column is :

$c_{ij} = \frac{1}{m-1} \sum_{k=1}^{m} (p_{ki} - \mu_i)(p_{kj} - \mu_j)$

m = a set of points each n dimensional:

$$\begin{aligned} p_1 &= (p_{11}, p_{12}, \cdots, p_{1n}) \\ p_2 &= (p_{21}, p_{22}, \cdots, p_{2n}) \\ &\cdots \\ p_m &= (p_{m1}, p_{m2}, \cdots, p_{mn}) \end{aligned}$$

μ is the average point which is computed by averaging each coordinate separately :

$$\begin{aligned} \mu &= (\mu_1, \mu_2, \cdots, \mu_n) \\ &= (\frac{p_{11} + \cdots + p_{m1}}{m}, \cdots, \frac{p_{1n} + \cdots + p_{mn}}{m}) \end{aligned}$$

M is a matrix $m \times n$

$$M = \begin{pmatrix} p_{11} - \mu_1 & \cdots & p_{1n} - \mu_n \\ \cdots & \cdots & \cdots \\ p_{m1} - \mu_1 & \cdots & p_{mn} - \mu_n \end{pmatrix} \text{ [15]}$$

According to Turk and Pentland :

$C = \frac{1}{m} \sum_{n=1}^{M} \Phi_n \Phi_n^T$

The matrix C is the covariance matrix

An entry u_i in that matrix is :

$u_i = \sum_{k=1}^{M} v_{lk}\Phi_k \; l = 1, \cdots,$

$\Psi = AA^T$

Φ_i is the difference of the actual image $\Gamma_i 28$ and the average image Ψ and $A = [\Phi_1, \Phi_2, \cdots, \Phi_m]$

Their purpose was to :

... find the principal components of the distribution of faces, or the eigenvectors of the covariance matrix of a set of face images treating an image as a point (or vector) in a very high dimensional space.

... The eigenvectors are ordered each one accounting for a different amount of the variation along the image faces [13].

While it took so much time for an extravagantly memory consuming preprocessing with the pca program which is available in the Edinburgh Artificial Intelligence Departmental Machines, it took less than half an hour to obtain the same results with a matlab program that was written according to the Turk and Pentland paper specifications. The program initially reads the file containing the grouped features as a matrix of 223x360, 223 image samples of 360 pixels each (i.e.12x30).

The calculation of an average feature is done which is actually the mean of that matrix. This result is put into another matrix which is subtracted from the original one.

So :

A=matrix that contains the image samples

m=number of image samples

$S = \frac{A}{1000}$

$B = \frac{\sum S^T}{m}$

$C = S - B$

$L = C^T \times C$

$[V, D] = eig(L)$ $[V, D]$ contains the eigenvectors and the eigenvalues

$M = C \cdot V$

$T =$ normalised M

$XR = \frac{M}{T}$

$I = 60$ bigger eigenvectors of XR

$K = \sum I \times C$ find the sum of the inner product of I and C

The matrix C is used as an input argument to the functions that perform the eigenvector and the eigenvalue calculations.

When this is done, the matrix XR contains all the normalised orthogonal eigenvectors. The ones that have got the biggest eigenvalues are kept in the I matrix which is used in the image reconstruction. If the entire number of eigenvectors is used, then the reconstruction is a perfect copy of the actual image. If a smaller number of these eigenvectors are used then the resulting reconstruction is not a perfect copy, but instead is a more abstract representation, not of the specific image it was extracted from. According to Turk and Pentland [13], the number of the best eigenvectors λ [1] used is determined by :

$\lambda_k = \frac{1}{M} \sum_1^M (u_k^T \Phi_n)^2$

$u_l^T u_k =$

$\delta_{lk} = 1$

if $l = k$

0otherwise

where u_k is the k-th eigenvector.

The average face/feature A was determined by :

$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n$

Each processed face differed from the original one by :

$\Phi_i = \Gamma_i - \Psi$

---

[1] λ represents the eigenvalues whose number subsequently decides how many eigenvectors will be used

A new face image Γ was determined by :

$w_k = u_k^T(\Gamma_i - \Psi)$

where Ψ is the matrix containing the average features

This method of deciding the best eigenvectors has been followed in this case as well. The first 60 eigenvectors were considered the optimum number to be used, since they are enough to specify a facial feature.

When the eigenvector matrix is ready, the algorithm continues by creating another matrix, which is actually the inner product of the eigenmatrix and the matrix that has the result from the subtracted average features. The outcome of this operation is a set of scalar values, which are used as input to the net. The inner product was used as such because it conveys unique information about each vector and subsequently, this reduces the probability that the net will misclassify an input pattern that is generated with this technique.

In order to verify that the number of selected eigenvectors was the optimum a reconstruction of the processed images is necessary. So by finding the inner products of the eigen matrix and the matrix that contains the scalar values, which we shall call alpha, and adding each time the corresponding entry in the matrix with the average feature calculations, the resulting reconstruction when projected shows that indeed the number of eigenvectors selected as well as the rest of the processing was correct.

It is the matrix holding the set of the alpha values that will eventually be fed to the net, after preparing the file for this purpose by defining a specific number of entries in each line (the optimum was 20) [2] and by adding at the end of the defined pattern the binary output the net has to learn to associate when a similar imput pattern is fed to it. For the left eye this output was 1000, for the right eye 0100, for the nose 0010, for the mouth 0001 and for the non-examples was 0000. This representation with that particular configuration was thought to be easier to learn for the neural net.

---

[2]If a line in the file the net had to read was longer than that the rbp program was giving error messages

At the end of this chapter follows a set of examples for each of the facial characteristics , illustrating the actual image, the error which is the difference of the actual image and the reconstructed one and finally the reconstructed image figure 3–5 figure 3–6 figure 3–7. The eigenfeature is also illustrated, which is actually the image of the biggest eigenvector.

The image samples shown were randomly selected. The reconstructed images do not have a big difference from the actual ones as it can be seen from the reconstructions. In this way it was clear that the number of eigenvectors used was optimum and therefore the noise introduced because of the reduction of the data dimensionality was not affecting negatively the operation of the net.

3.4 Fourier Transform Method I

The two dimensional Fourier Transformation (ft) is a 2D transformation that analyses the spatial frequencies in an image. High frequencies are due to areas where intensities change a lot while low frequencies are due to areas of low contrast.

This technique was used in two forms, the first was with all the frequencies and the calculation of the phase angles in radians.

The two dimensional fourier transform was done on the input file. It was done individually on each 360 long pattern and the resulting complex number matrix was kept for further processing.

A Fourier transform is defined as :

$\mathcal{F}$ f(x) = F(u) = $\int_{-\infty}^{\infty} f(x)exp[-j2\pi ux]dx$

where $j = \sqrt{-1}$

The Fourier transform of a real function is generally complex :

$F(u) = R(u) + jI(u)$

$\mathcal{F}(\sqcap) = \|\mathcal{F}(\sqcap)\|]^{|\phi(\sqcap)}$

$\|\mathrm{F}(u)\| = [R^2(u) + I^2(u)]^{\frac{1}{2}}$

$\phi(u) = tan^{-1}[\frac{I(u)}{R(u)}]$

When the two dimensional fourier transform was done the data went through the final processing : the calculation of the phase angles in radians. This was accomplished using the $\phi(u)$ the phase angles.

The frequency factor is represented by u.

Verifying that the ft results were the correct ones was done by projecting the inverse ft of the image and comparing the results with the actual input pattern. The inverse Fourier transform is defined as :

$\mathcal{F}^{-1}F(u) = f(x) = \int_{-\infty}^{\infty} F(u)exp[j2\pi ux]du$

The matrix containing the phase angles in radians was then prepared for the net, following the same standards as in the previously mentioned preprocessing technique, ie 20 entries per input line and addition of the binary output at the end of the pattern for the net. Examples of how the two dimensional fourier transform of the facial features follow. Note that no errors occured between the actual and the reconstructed image, since no reduction in the data dimensionality took place.

Examples follow at the end of this chapter for each of the facial characteristics, illustrating in order the ft of the feature, the inverse ft and the fourier transform plot in figures 3–8 3–9 3–10 3–11 3–12. The ft plots show that the ft of each feature is clearly different and that the net has more chances to learn the patterns faster without getting confused as to which is which.

3.5 Fourier Transform Method II

As it was earlier mentioned the ft technique was used twice, but with a different number of frequencies each time. While in method 1 360 phase angles were used, in this case only 57 out of 512 were used, since here the extracted images were 16x32 ie 512 pixels in each picture.

Again the input file was read by the program and the ft was applied to it. Then a spatial filter was applied on the resulting matrix in an effort to determine the optimum number of frequencies to be used so as to conclude with a meaningful representation to the net. Figures 3–13 3–14 3–15 3–16 and 3–17 show clearly the reason why 57 frequencies were selected. Less than that resulted in a representation

with too much noise, the features were too abstract and therefore not appropriate for the training of the neural net.

The algorithm that was thresholding the frequencies is as follows:

1. Read in the pattern
2. Find the inverse fourier transform for it
3. Threshold the frequencies that are bigger than 500
4. Project the outcome

As soon as the number of frequencies was decided, the resulting 'mask', was kept for further use. That mask was a 360 vector long that had zeroes and ones. At the positions where the value was 0, the frequency of the input pattern was determined as being below the value used for filtering, and therefore, was rejected. In the cases where there was 1, the frequencies were accepted as valid and they were kept in to a special file that would eventually be fed to the net.

The resulting ft matrix was a complex one; as it was mentioned above the multi-layer perceptron does not accept complex values as inputs, that is why that matrix had to be presented in another way to the net.

The real part was separated from the imaginary part and the each one was kept in separate matrices. This, however meant that the number of inputs to the net had to double, since there were now two matrices that had 57x223 dimensions, (where 223 the number of input patterns, and 57 the number of ft output complex matrix).

So in this case the net had 114 inputs ie 57x2.

Again, in this case reconstruction of the initial images was attempted so as to ensure the success of this last preprocessing technique. By projecting the inverse ft of the filtered data, we made sure that this technique had indeed been successful.

The thresholding value for the mask creation was 500. This value was carefully selected in order to convey all the meaningful information for the net without distorting the facial features beyond recognition.

As it can be seen from the plot in figure 3–5 that shows the frequencies in the mask, those smaller than 500 were not important.

The error plots show the difference of the recontructed image from the actual image.

In the section that follows all the preprocessing results are explicitely stated giving the exact nature of the whole procedure. At this point the net training takes place.

3.6 Summary

In this chapter, we discussed in detail the mechanics of the three preprocessing that were used to produce the data files to be fed to the untrained net. We also illustrated the extracted features, the features generated by the preprocessing and the reconstructed ones.

From the preprocessing results we concluded that the techniques used gave patterns that were not very much different from the actual ones, thus increasing the probabilities that the nets will successfully recognise the patterns they will be tested with.

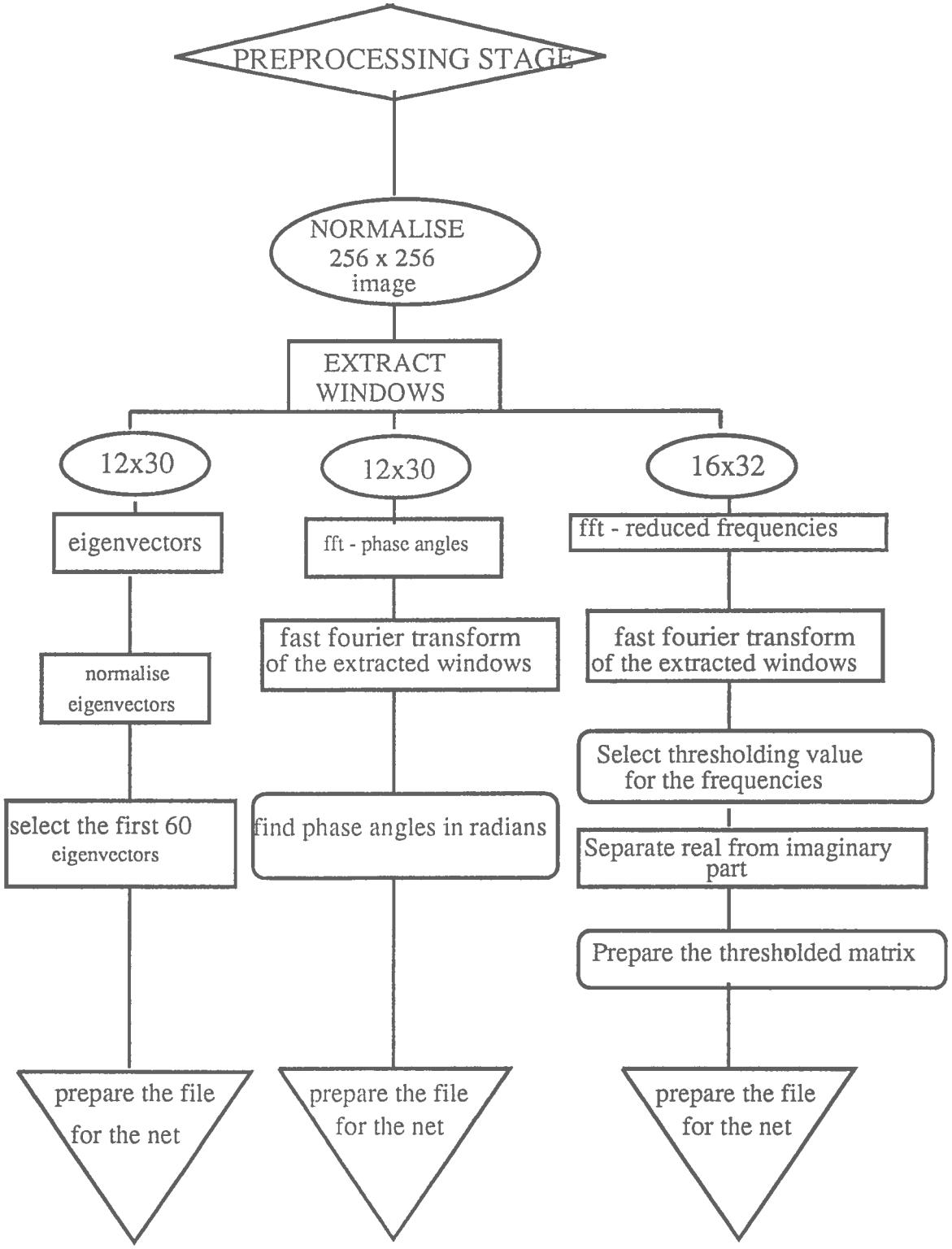


Figure 3–3: Graphical Preprocessing Overview

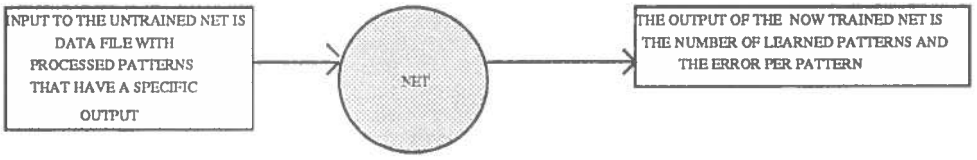


Figure 3–4: The role of the net conceptually

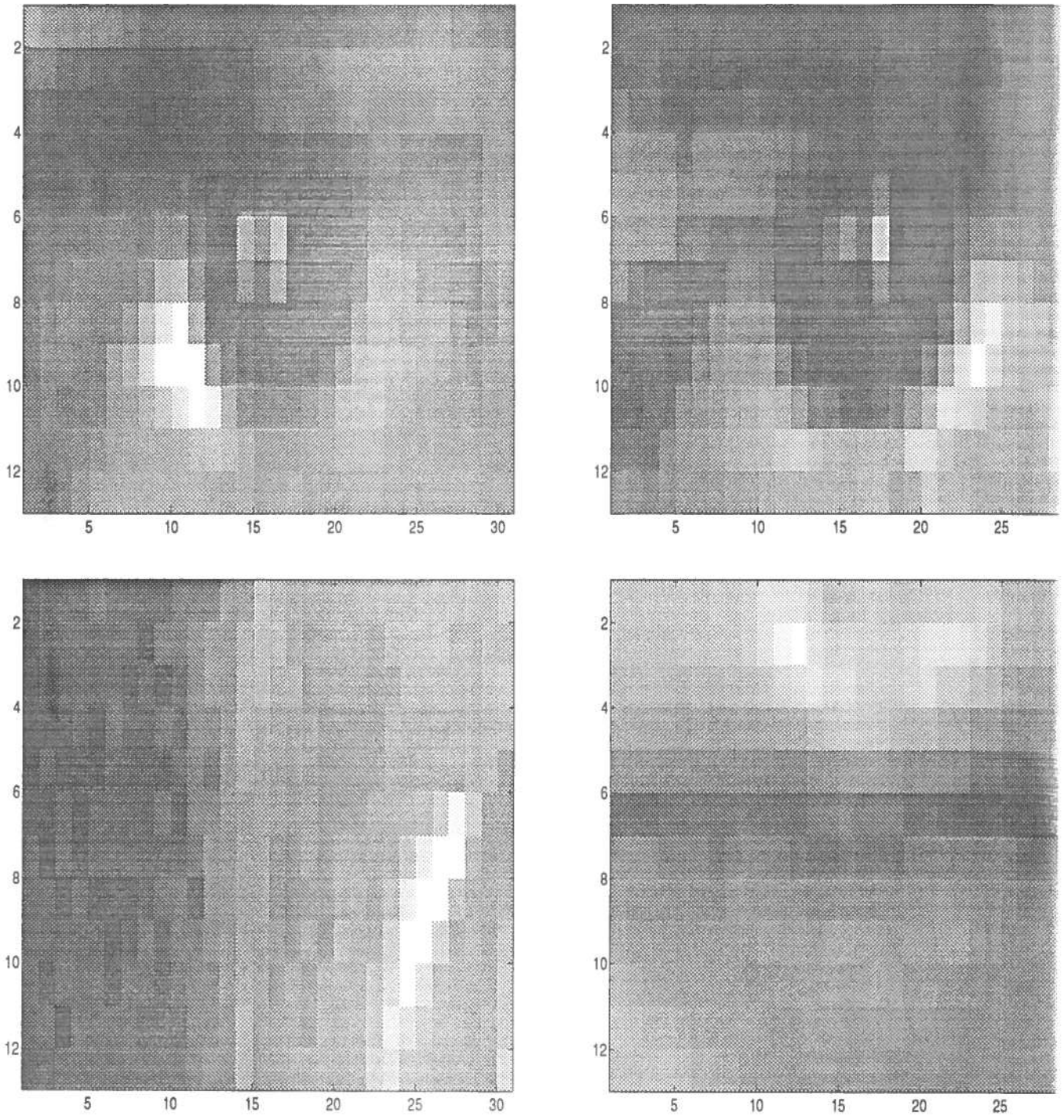


Figure 3–5: Normalised (12x30) left eye, right eye , nose and mouth

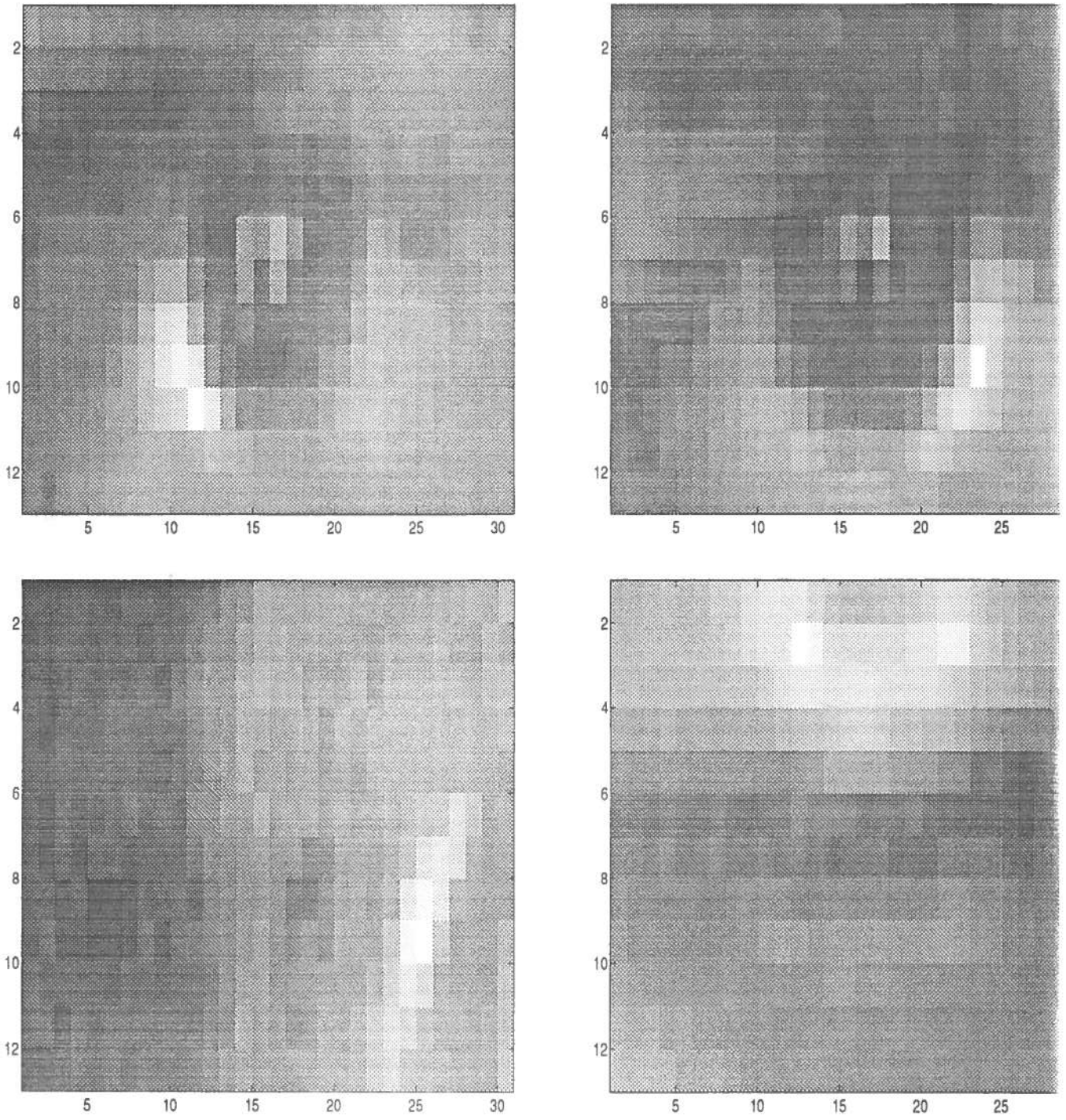


Figure 3–6: Reconstructed left eye, right eye, nose and mouth

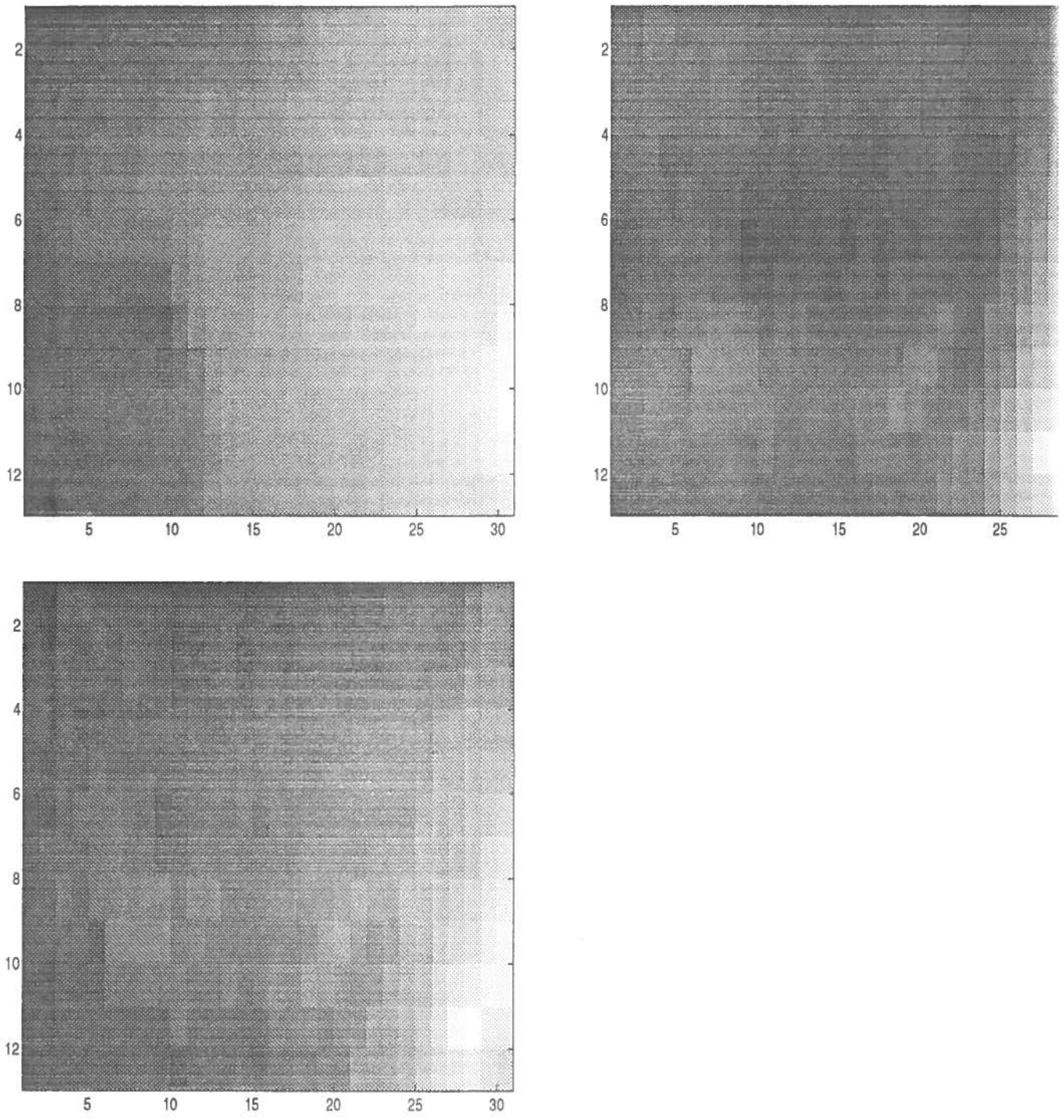


Figure 3–7: The Best Average EigenFeature, A normalised non example 12x30, and its reconstruction

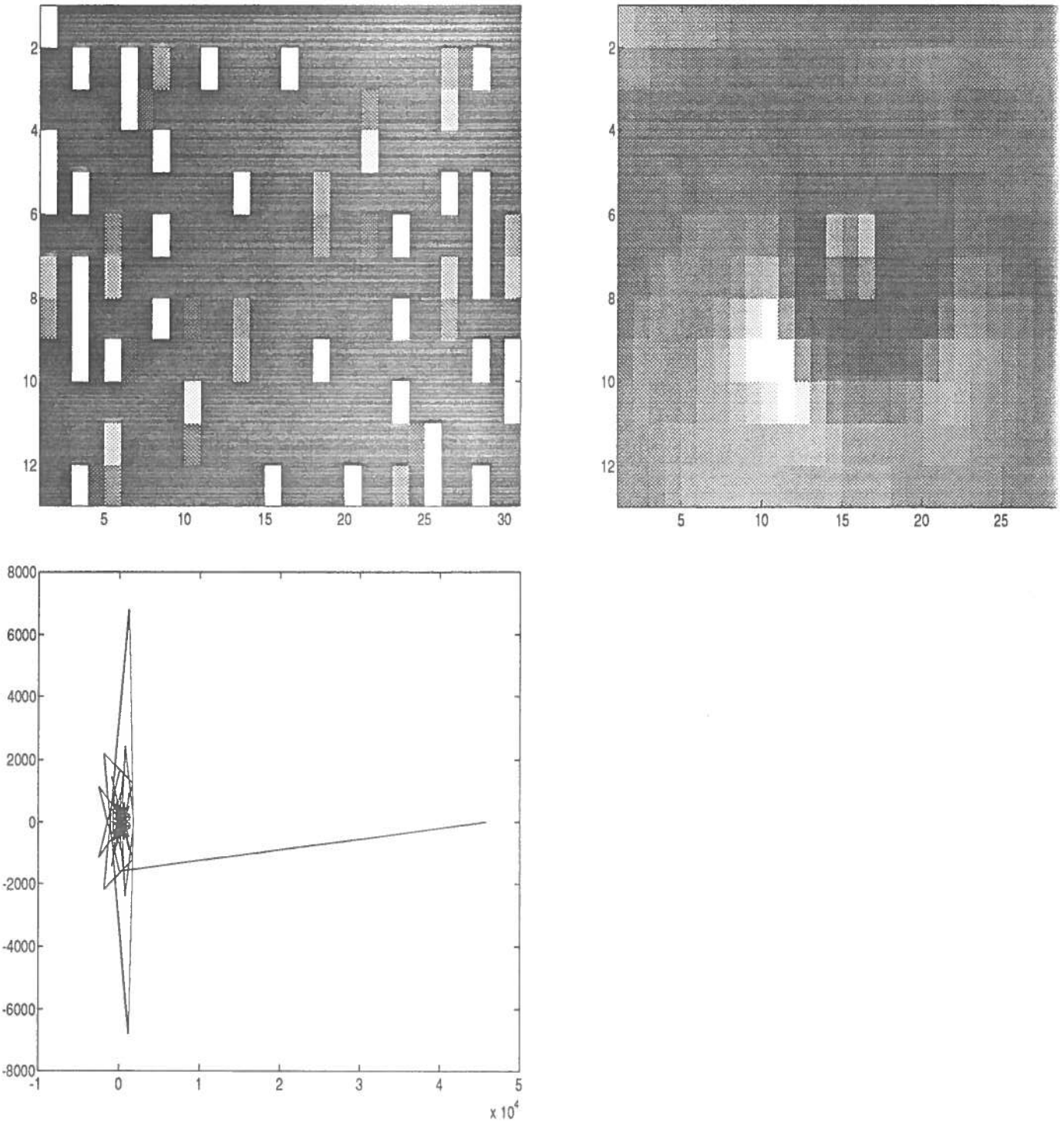


Figure 3–8: fourier Transform of a left eye, its inverse, and the ft plot

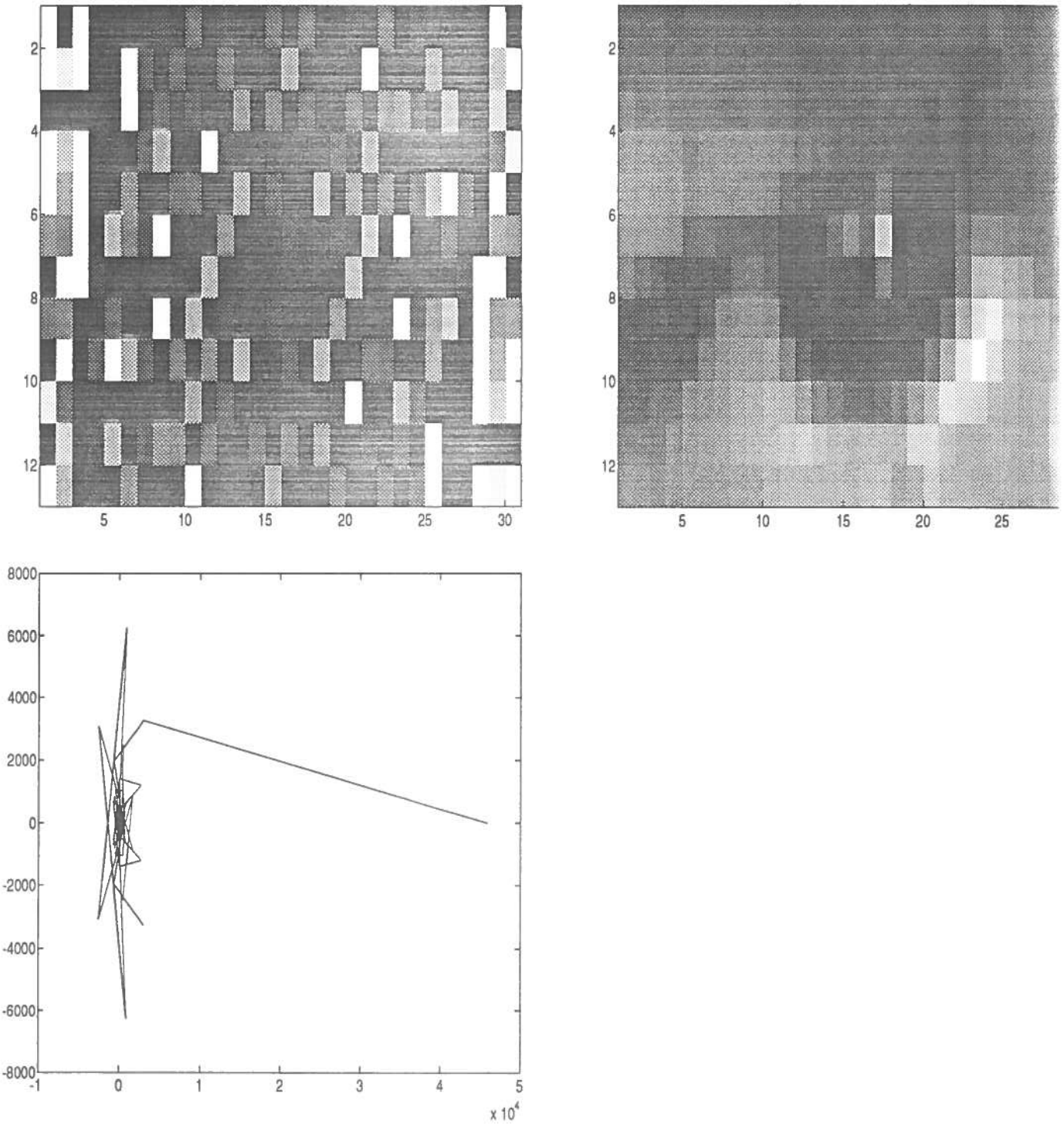
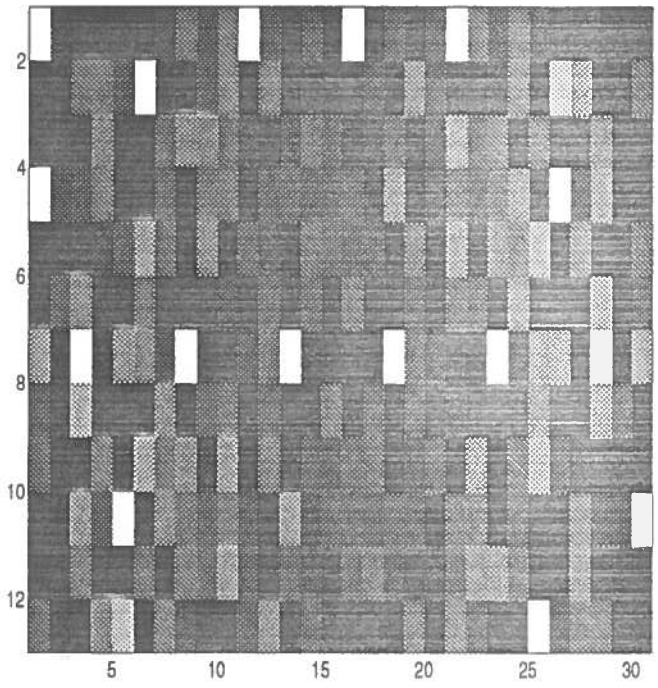
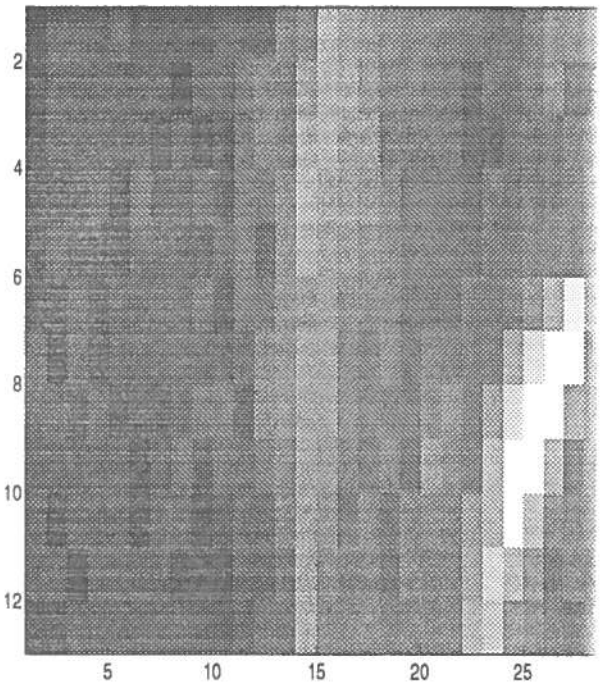


Figure 3–9: fourier Transform of a right eye, its inverse, and the ft plot





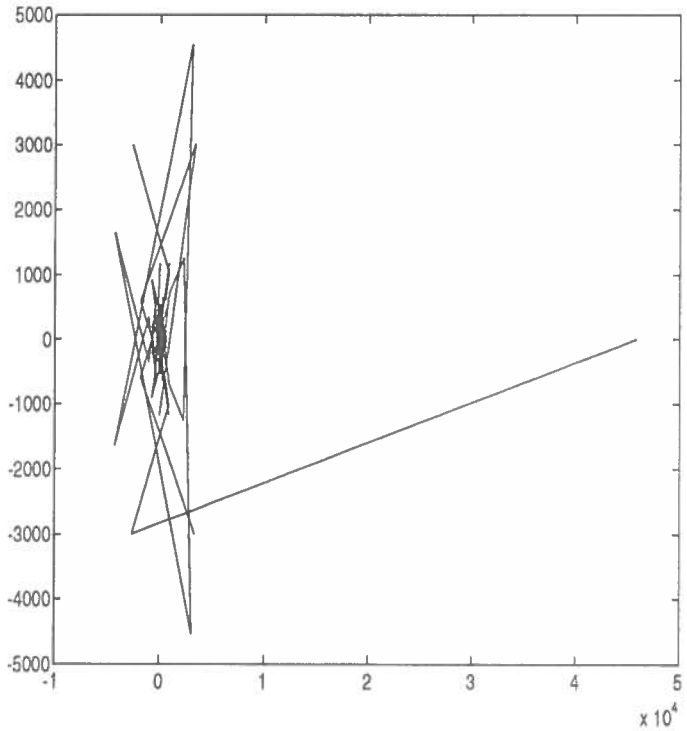


Figure 3–10: fourier Transform of a nose, its inverse, and the ft plot

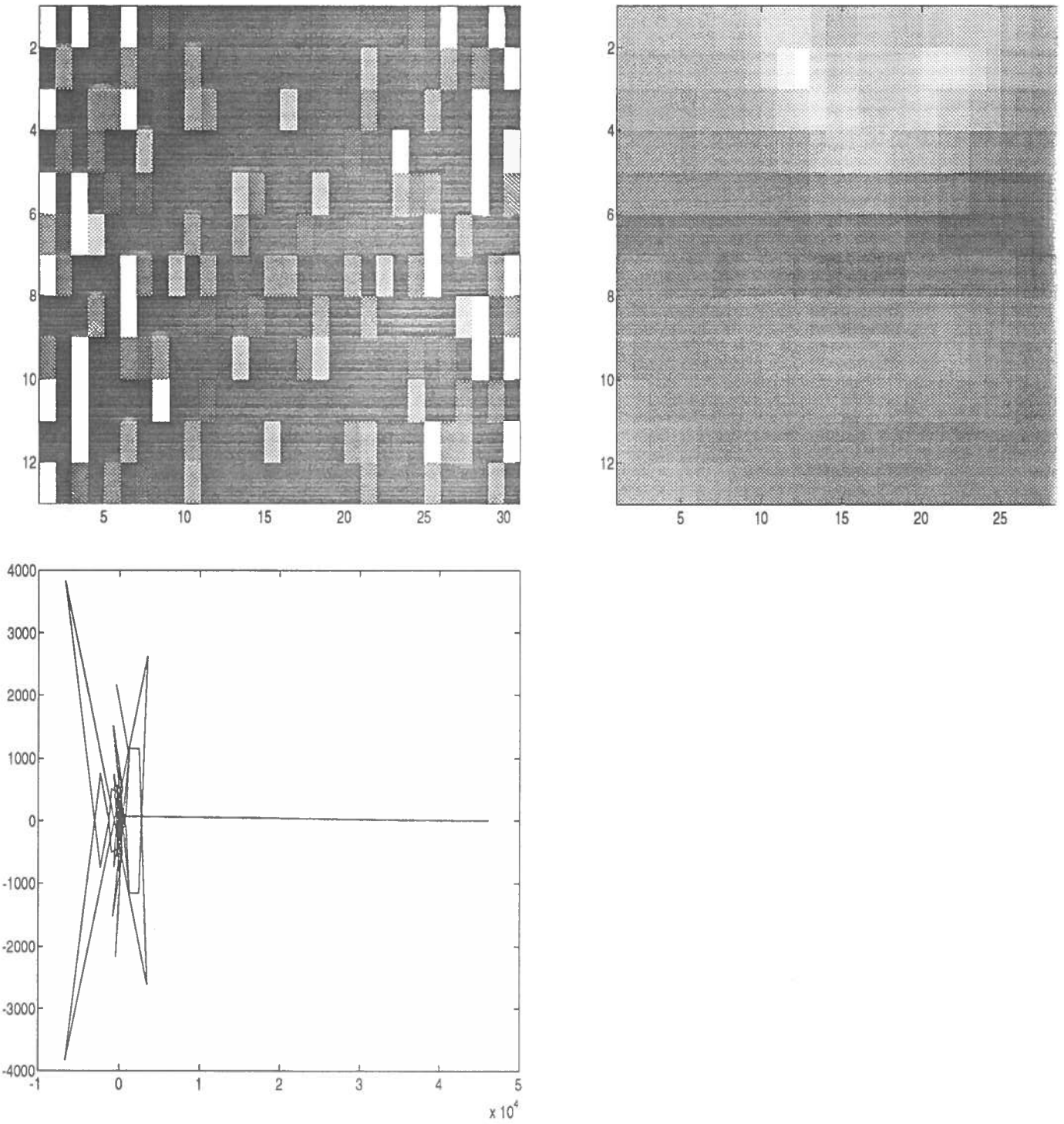


Figure 3–11: fourier Transform of a mouth, its inverse, and the ft plot

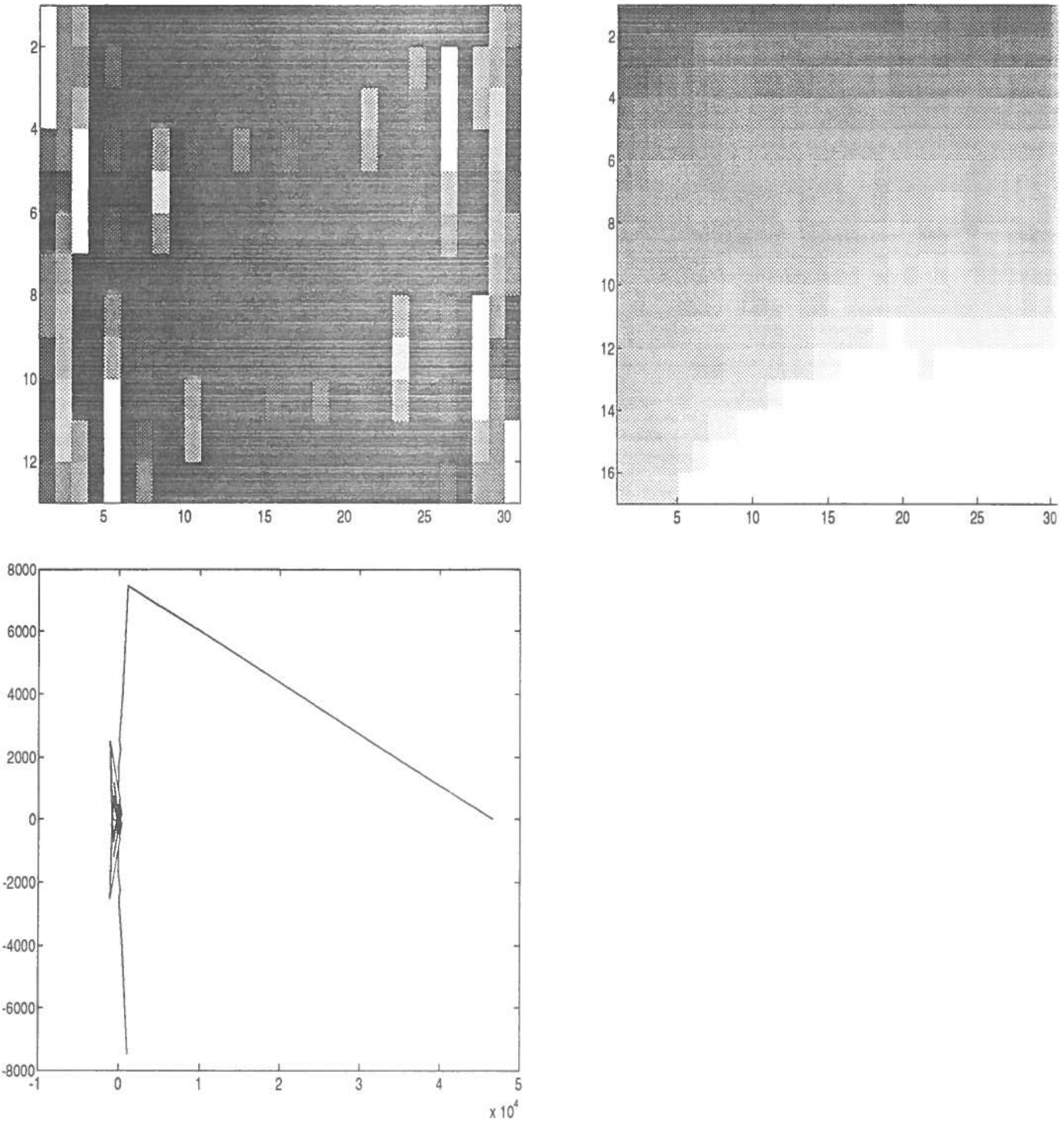


Figure 3–12: Fourier Transform of a non example, its inverse, and the ft plot

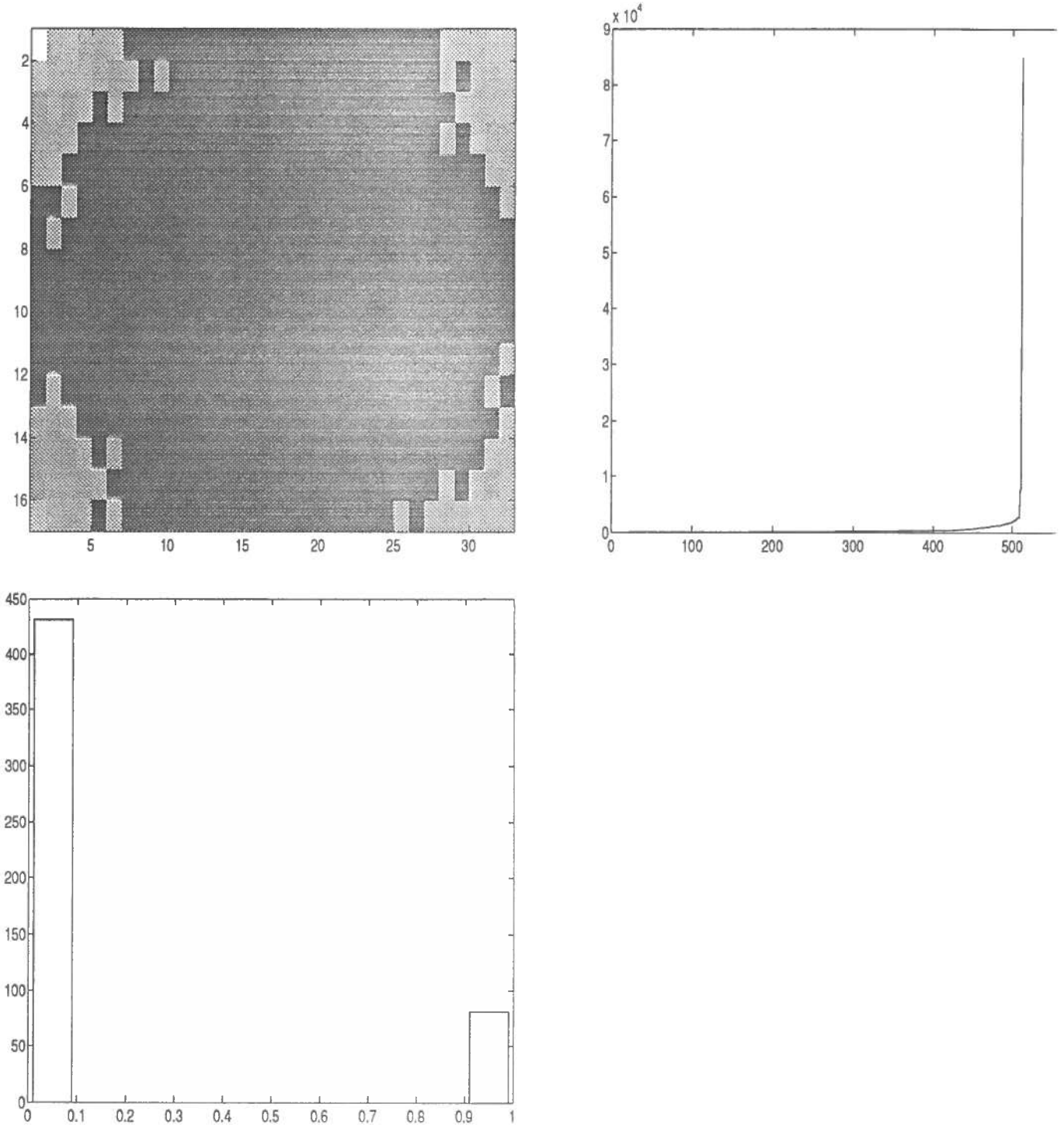


Figure 3–13: The Image of the Mask,The Frequency histogram and the plot indicating why this frequency was selected

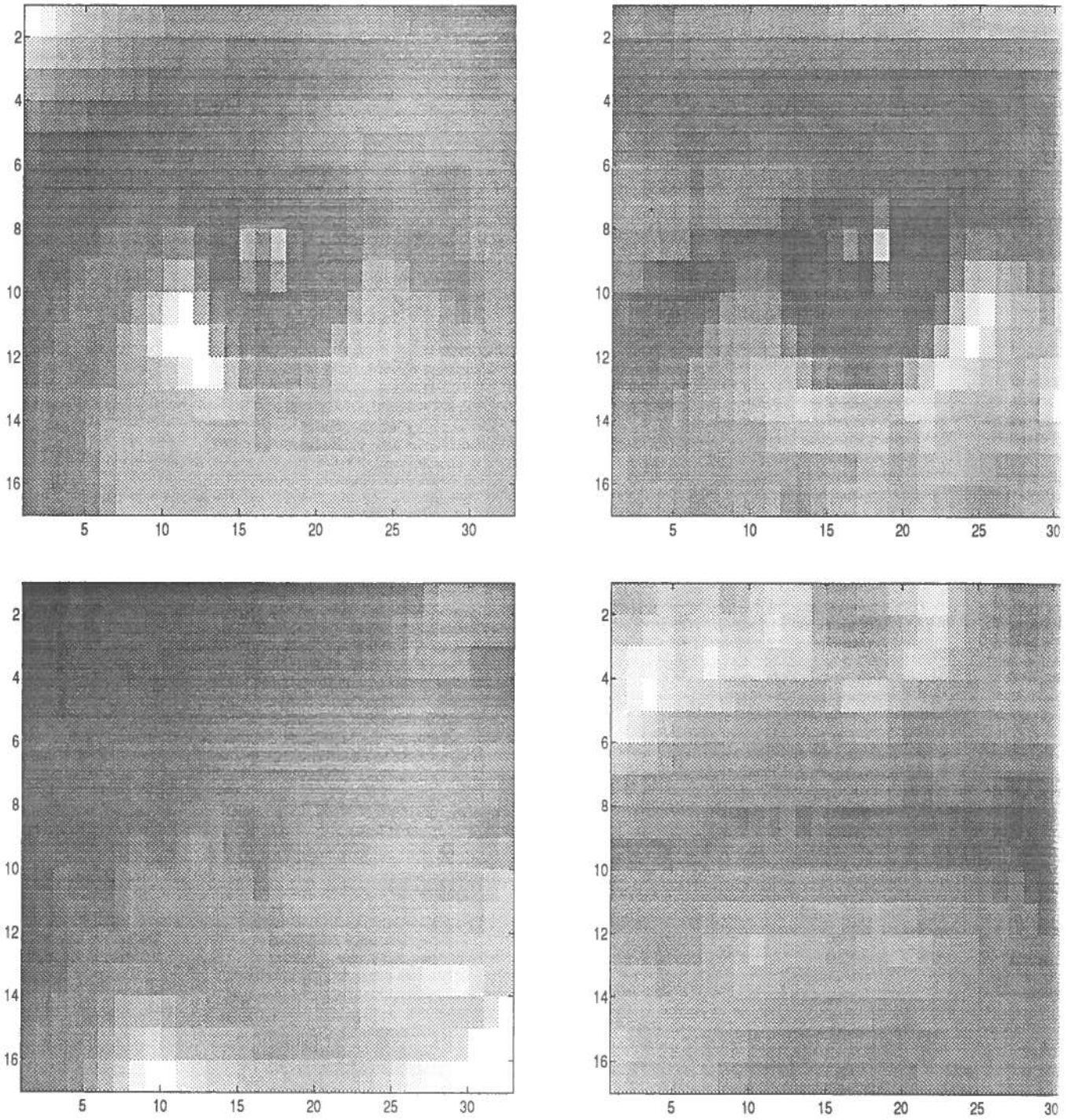


Figure 3–14: Normalised 16x32 left eye, right eye, nose and mouth

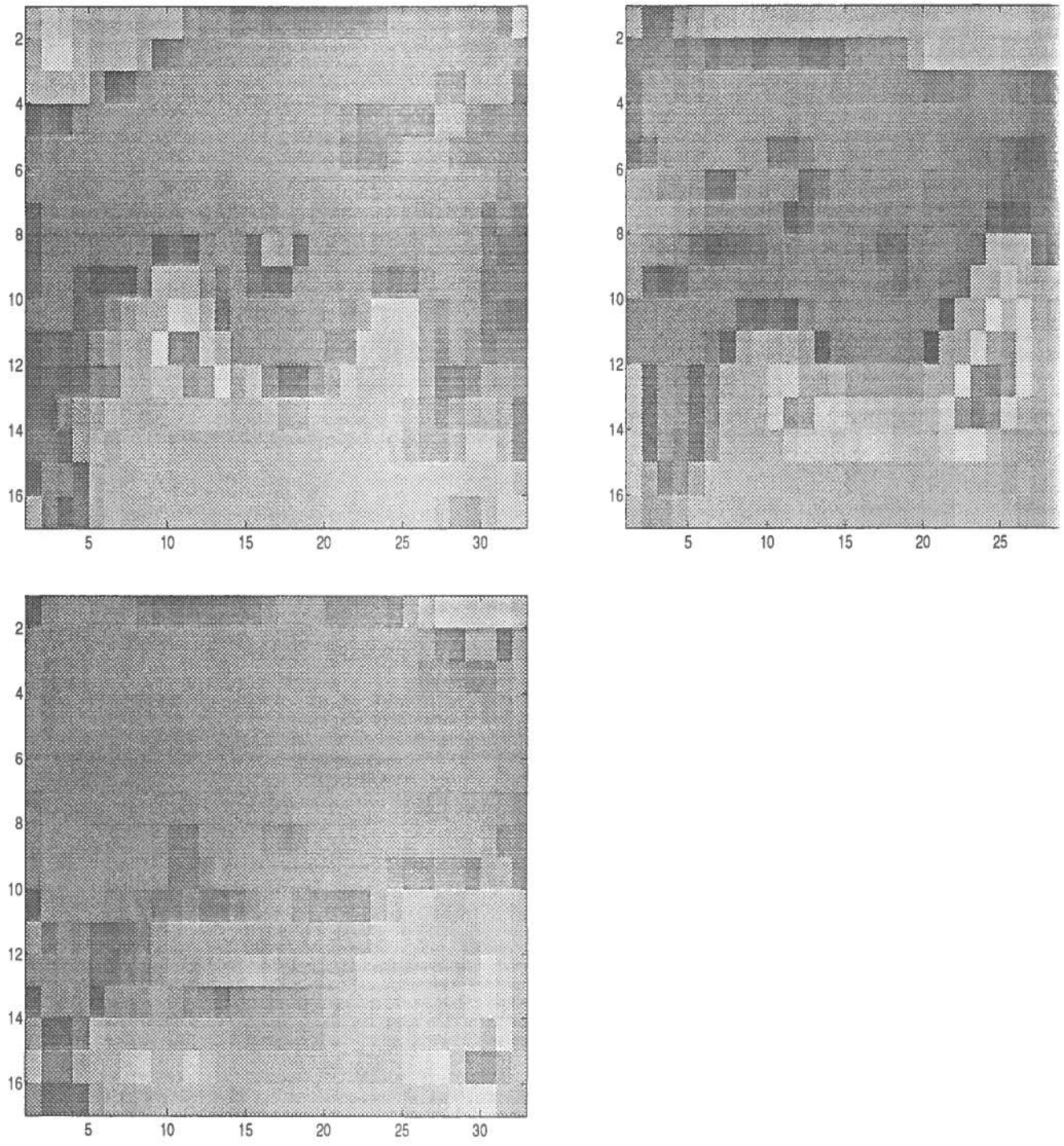


Figure 3–15: Masked left eye, right eye, nose and mouth

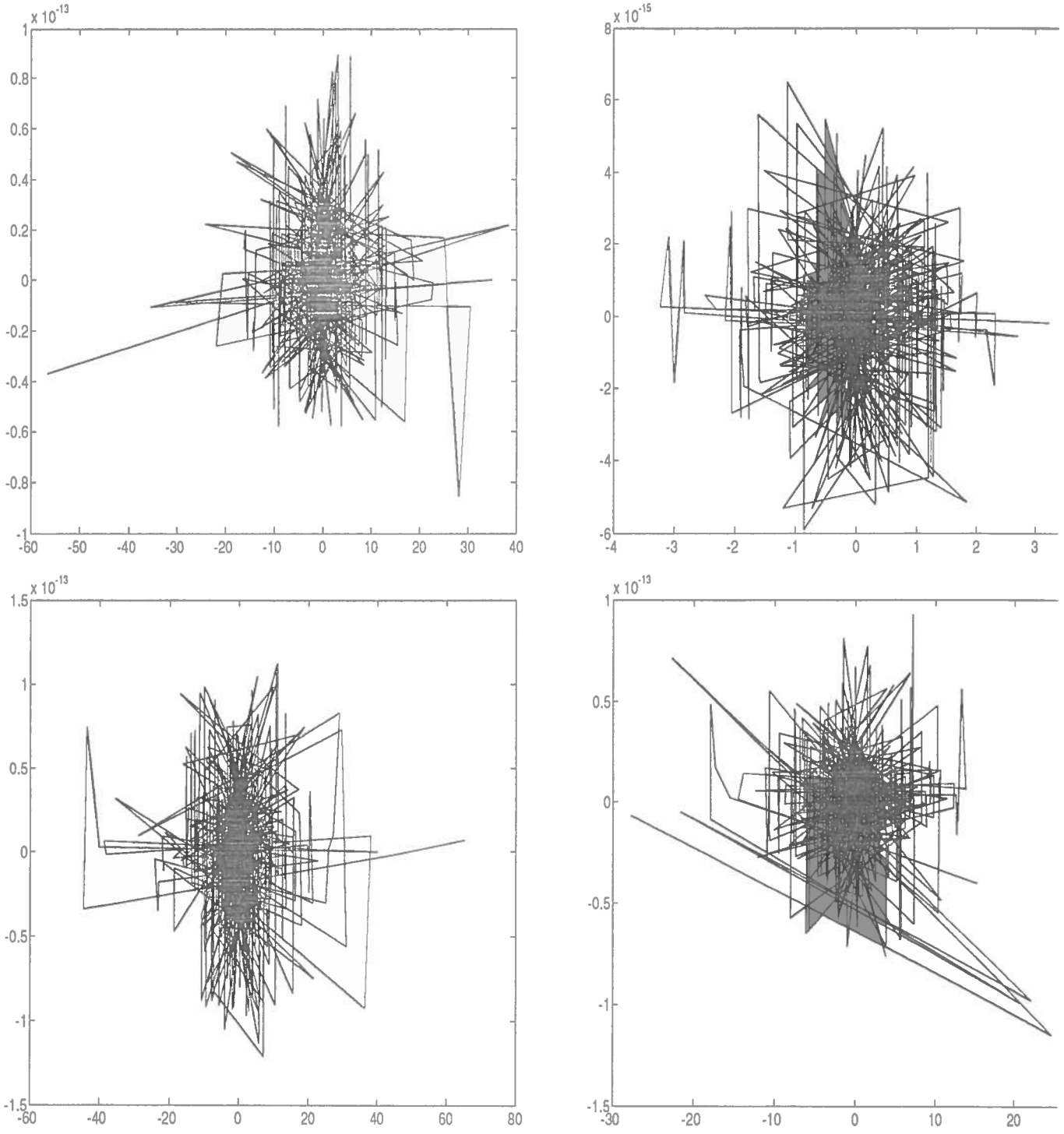


Figure 3–16: Error for the ft -II left eye, right eye, nose and mouth

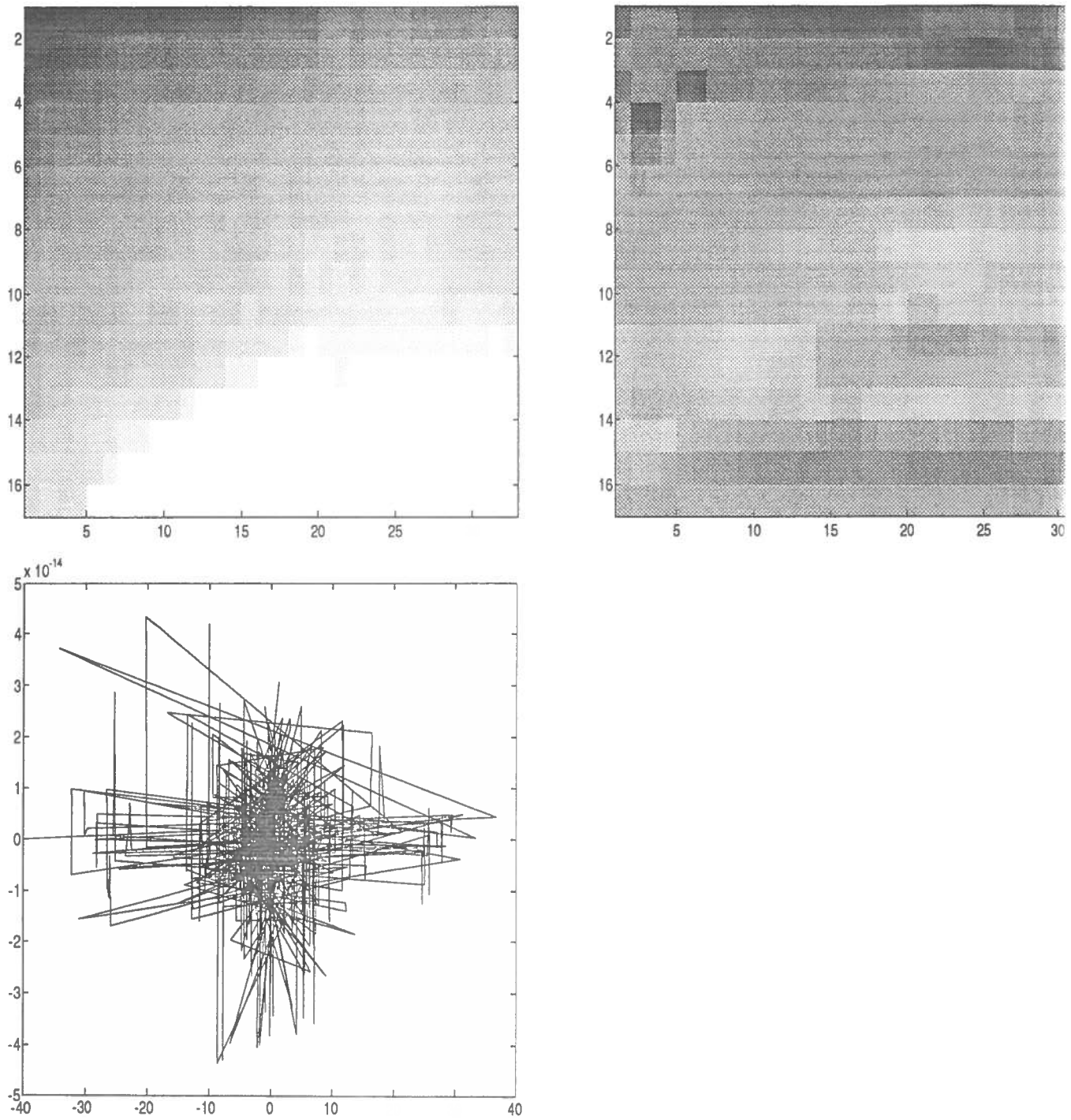


Figure 3–17: A normalised non example 16x32, it reconstruction, and the error

Chapter 4

Neural Net

4.1 Topology, Learning Methods and Training

As has already been mentioned the pattern length determines exclusively the number of input units to the net. The output units are mostly up to the user. For the scope of this project the output units for the three net configurations were four, one for the left eye, one for the right eye, one for the nose and one for the mouth. When the net was outputting 0 probabilities in all four cases during the final pattern recognition, this meant that the input pattern belonged to the group of non examples.

There are three net configurations each one pertinent to a preprocessing technique. The net is a multilayer perceptron [11] and it was trained with recurrent back propagation [11] [15]. The reason for this single training method was that more emphasis was placed on the preprocessing techniques than the neural net configurations and training methods.

Throughout the net training various factors were tested, ie the number of hidden units, the number of connected units and various possible configurations, the η and η 2 values, the momentum, the learning rule, the sharpness, the thresholding function, the seed, the tolerance and in general an effort was made to test as many as possible conigurations in order to find the one that would enable the net to learn as fast as possible.

The results that follow show more explicitly the effect of alterating the net parameters. All the changes were done methodically, meaning that not more than one configuration values were changed prior to testing, and that various combinations

were attempted of these values based on the results given during the training of the net with the previous configuration.

In some cases where the net was showing an almost steady undesirable behaviour, random changes were attempted, so as to observe the outcome. In some cases this bore success, because of the decreased training time and the learning of the patterns.

From the experiments performed with the net configurations, it became evident that what really affected considerably the speed of the training was the number of units in the second level, and the tolerance. If the number of units in the second level was too small or too big, the net had very big difficulties in learning, even the 40 from the 223 training samples. When the tolerance was increased, then the speed of learning was increased as well, but the error per unit was quite big for some tolerance values. For this reason, it was decided that the tolerance throughout all the tests, be kept at 0.1 level. The purpose of the net was not only to be able to recognise a facial feature, but also to be able to distinguish it among other facial features with a small error rate.

This tactic was rewarded when after a lot of experimentations and considerable training the net was able to give very big probabilities overall, and make very few mistakes in classification. As it was estimated, only 2-4% of the classification of patterns the net had not been trained on produced erroneous results. This estimation was based on the thresholded best windows where the classification errors occured in the cases of the left and right eyes. This was a phenomenon observed for all three nets; they were misclassifying left for right eyes and vica versa. These classification errors were not too many.

This was attributed to the careful selection of the components of the net configuration, to the preprocessing techniques as well as the monitoring of the image capturing.

The major parts of these procedures, were automated in order to exclude the possibility of a human error in the training data, and the verification procedure. The training of the net was always done in the background, because of the long hours it required.

All three final net configurations, have learned the input patterns 100% correctly, with very small errors per unit.

4.2 Net Configurations

The best results were obtained in the case of the ft phase angles net. It's configuration was :

```
m 360 20 4
s 7
k 0 0.4
t 0.1
e 0.01 0.1
a 0.3
A as
f ir or
f w b
n f fft
```

This was given as input to the rbp program.*m 360 20 4* makes a net with 360 units in the first layer 20 in the second and 4 in the third; *s 7* sets the initial seed to 7; *k 0 0.4* initialises the weights with values from 0 to 0.4; *t 0.1* sets the tolerance to 0.1; *e 0.01 0.1* η of the first layer is 0.01 η 2 (second layer) 0.1; *a 0.3* sets the momentum to 0.3; *A as* the thresholding function sigmoid; *f ir or* the output must be in real format; *f w b* keeps the weights in binary format to save space; *n f fft* the input file called fft.

The input to this net consisted of 360 long vectors. When the net training was over, another test was made in order to observe which and how many patterns the net had learned correctly. The file containing the original input patterns was again fed to the net, only that in this case, the binary output at the end of each pattern was replaced by the letter *p*, which stated to the net that was expected to assign a probability to it. The output of this procedure was a five column file,

where each column represented each feature accordingly : left eye, right eye, nose, mouth, and the probability of having made an error during the recognition. If this error was smaller than 0.050 then the net was also commenting the learned pattern as *ok*, meaning that it had learned it adequately.

This specific configuration assigned the biggest probabilities overall and made the fewest errors in the classification of unlearned patterns. This is attributed to the fact that all the frequencies were used as input to the net and not selected parts of it or reduced dimensionality data were given as input.

The second best net configuration was the alpha net. This also gave very good results, and clusters with big probabilities. In this case the erroneous classification was bigger than that of the ft phase angles and the net wrongly classified left eyes for right eyes and vice versa.

The input to this net consisted of 60 long vectors, formed by sixty alpha values that corrseponded to the 60 best eigenvectors.

```
m 60 15 4
k 0 5
t 0.1
e 0.01 0.1
a 0.5
A as
f ir or
f w b
f p G
n f alphas
```

As it is shown by figure 4–1, if less than 60 eigenvectors were used, the error between the reconstructed image and the actual image increased rapidly. That error was calclulated by:

image error(M,K)

a_i = calculate i-th projection of the k-th image $i = 1, \cdots, M$

$New\ Image = \sum_{i=1}^{M} a_i u_i + A$

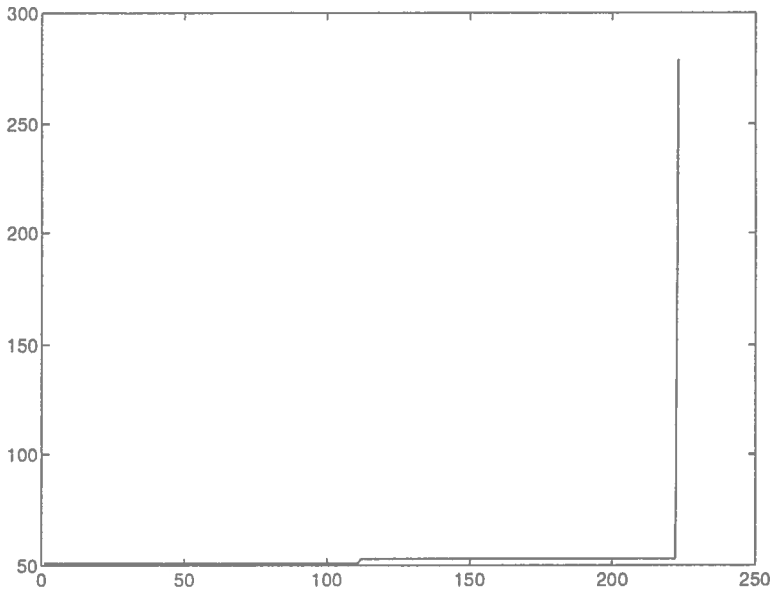


Figure 4–1: Error plot indicating the best eigenvectors

A=*average image*, u_i = the i-th eigenvector

$Actual\ Error = \sum_{i=1}^{12} \sum_{j=1}^{30} \|(originalimage(K)_{ij} - reconstructedimage_{ij}\|$

Finally the third net configuration that gave results that were less accurate than the previous two, was the ft reduced frequencies technique.

The input ot this net consisted of 114 long vectors that were produced by the combination of the thresholded real and imaginary parts of the fast fourrier transform matrix of the actual normalised input pattern.

Although the net was fully trained, and the error per unit was not much bigger compared to the others, the net assigned lower probabilities overall than the previously described configurations. An important issue was that the net took almost 4 times longer to train than the other two did.

The configuration of the net was :

```
m 114 87 4
s 7
k 0 0.4
t 0.1
e 0.01 0.1
```

```
a 0.5
A as
f ir or
f w b
n f data
```

4.3 Testing

Various components were tested in the net configuration. The two most important features that immediately affected the net performance were the number of units in the second layer of the net as well as the learning method. The latter was affecting significantly the net's speed of learning the input patterns.

The seed for the random weights was changed with test values ranging from 7 to 4 until the best seed was decided. Then three seed numbers were tested together so as to decide upon the efficiency of them. The best value was found to be 7.

The best initial weights were obtained when the *initial random weight* option in the rbp program was set within the range of 0 to 4. We concluded there after testing other ranges e.g from 0 to 0.5, from -4 to +4, from 0 to 9, from 0 to 1 or 0 0.5.

The activation function was changed as well. The kinds of thresholding functions tested were mainly the sigmoid and the radial ones since past experience has indicated that for this kind of problem they give the best results. The sigmoid thresholding function gave the best results.

The tolerance was kept at all times the same 0.1, because we wanted the net to learn the patterns as best as possible. When it was changed to 0.2, then for the same number of learned patterns the error per unit was bigger.

The a value (the *momentum*) was changed from 0.1 to 0.2, 0.3, 0.4, 0.5, 0.9, but the best value was the 0.5 one. When the alpha value was changed alone in the current net configuration no big differences were observed. When it was changed along with the rest the resulting balance of the proportions, it was proven a decisive component for the speedy learning of the net.

The η values were also adjusted accordingly. Different η values were tried for the two layers such as 0.1 for the second and 0.01 for the first. This technique was proved to be successful, instead of having the same η value for both layers.

The quick propagation algorithm was not proved effective enough although various tests were performed with different net configurations. The delta-bar-delta rule was not very successful either.

Instead a gradual pruning of the number of units in the second layer was done and this seemed to be quite effective since we concluded with a net configuration that was eventually a winner.

4.4 Net Results

All three nets were tested with images they had not encounter before. All the net configurations were fully trained i.e. they had all learned the input patterns during training 100% correctly, with a very small error rate per unit.

As it was earlier mentioned the three nets were also tested in the patterns they were trained on. In all the cases no erroneous classification occurred and all of the patterns were confirmed as learned as it can be seen from the following net outputs for all the configurations.

This applies for all three net configurations. The following tables, show the training resuts for each net configuration.

Iterations	% of learned patterns	Number of patterns	Error/Unit
1000	52.47%	117 right 106 wrong	0.08995
2000	55.61%	124 right 99 wrong	0.08796
3000	78.92%	176 right 4 wrong	0.06076
4000	20.63%	46 right 177 wrong	0.21674
5000	40.36%	90 right 133 wrong	0.06839
6000	79.37%	177 right 46 wrong	0.06353
7000	79.37%	77 right 46 wrong	0.05890
8000	40.36%	90 right 133 wrong	0.14423
9000	78.92%	176 right 47 wrong	0.05847
10000	76.68%	171 right 52 wrong	0.06689
11000	94.17%	210 right 13 wrong	0.00805
12000	98.65%	220 right 3 wrong	0.00334
13000	99.55%	222 right 1 wrong	0.00183
14000	99.55%	222 right 1 wrong	0.00121
14111	100.00%	223 right 0 wrong	0.00117

Table 4–1: The rbp output of the trained $60 - 15 - 4$ net

Iterations	% of learned patterns	Number of patterns	Error/Unit
100	83.86%	187 right 36 wrong	0.02382
200	90.58%	202 right 21 wrong	0.01634
300	91.93%	205 right 18 wrong	0.01338
400	92.38%	206 right 17 wrong	0.01031
500	95.96%	214 right 9 wrong	0.00578
600	95.52%	213 right 10 wrong	0.00558
700	95.96%	214 right 9 wrong	0.00465
800	95.96%	214 right 9 wrong	0.00406
900	96.86%	216 right 7 wrong	0.00360
1000	97.76%	218 right 5 wrong	0.00296
1100	97.76%	218 right 5 wrong	0.00269
1200	97.76%	218 right 5 wrong	0.00247
1300	98.65%	220 right 3 wrong	0.00220
1400	98.65%	220 right 3 wrong	0.00200
1500	99.10%	221 right 2 wrong	0.00170
1600	98.65%	220 right 3 wrong	0.00150
1700	98.65%	220 right 3 wrong	0.00142
1783	100.00%	223 right 0 wrong	0.00126

Table 4–2: The rbp output of the trained $360 - 20 - 4$ net

Iterations	% of learned patterns	Number of patterns	Error/Unit
100	83.86%	170 right 53 wrong	0.02456
200	90.58%	190 right 23 wrong	0.02323
300	91.93%	200 right 18 wrong	0.01967
400	92.38%	205 right 18 wrong	0.01759
500	95.96%	212 right 11 wrong	0.01563
600	95.52%	213 right 10 wrong	0.01309
700	95.96%	214 right 10 wrong	0.05234
800	95.96%	215 right 9 wrong	0.00507
900	96.86%	216 right 7 wrong	0.00465
1000	97.76%	217 right 6 wrong	0.00396
1100	97.76%	218 right 5 wrong	0.00269
1200	97.76%	218 right 5 wrong	0.00247
1300	98.65%	219 right 4 wrong	0.00230
1400	98.65%	220 right 3 wrong	0.00220
1500	99.10%	221 right 2 wrong	0.00200
1600	98.65%	220 right 3 wrong	0.00198
1700	98.65%	219 right 4 wrong	0.00150
1800	99.65%	222 right 0 wrong	0.00130
1900	100.00%	222 right 0 wrong	0.00121

Table 4–3: The rbp output of the trained $114 - 87 - 4$ net

The first set of figures 4–2 illustrates the behaviour of the three nets during training, by comparing in a single graph the percentage of patterns learned, the number of iterations for these patterns and the tolerance. Also each net performance is shown individually. The second set of figures 4–3 4–4 shows the the performances of the three nets after training, when they were asked to recognise the patterns they were trained on. The figures refer to the probability distribution assigned by the net, having in mind that all the patterns were correctly identified. The first 49 are left eyes, the next 47 are right eyes, the next 46 were noses, the next 49 were mouths and the rest of them non-examples.

It has to be noted that all the nets successfully recognised the patterns they were trained on, and learned them all 100%.

The following table shows the percentages of the probabilities assigned by the various net configurations after training:

	Probabilities			
Net Type	over 0.9%	over 0.8%	over 0.6%	less than 0.6%
eigen	30%	40%	50%	50%
phase angles	20%	35%	40%	60%
fft II	15%	27%	60%	40%

Table 4–4: Probability distributions each net assigned to the test images

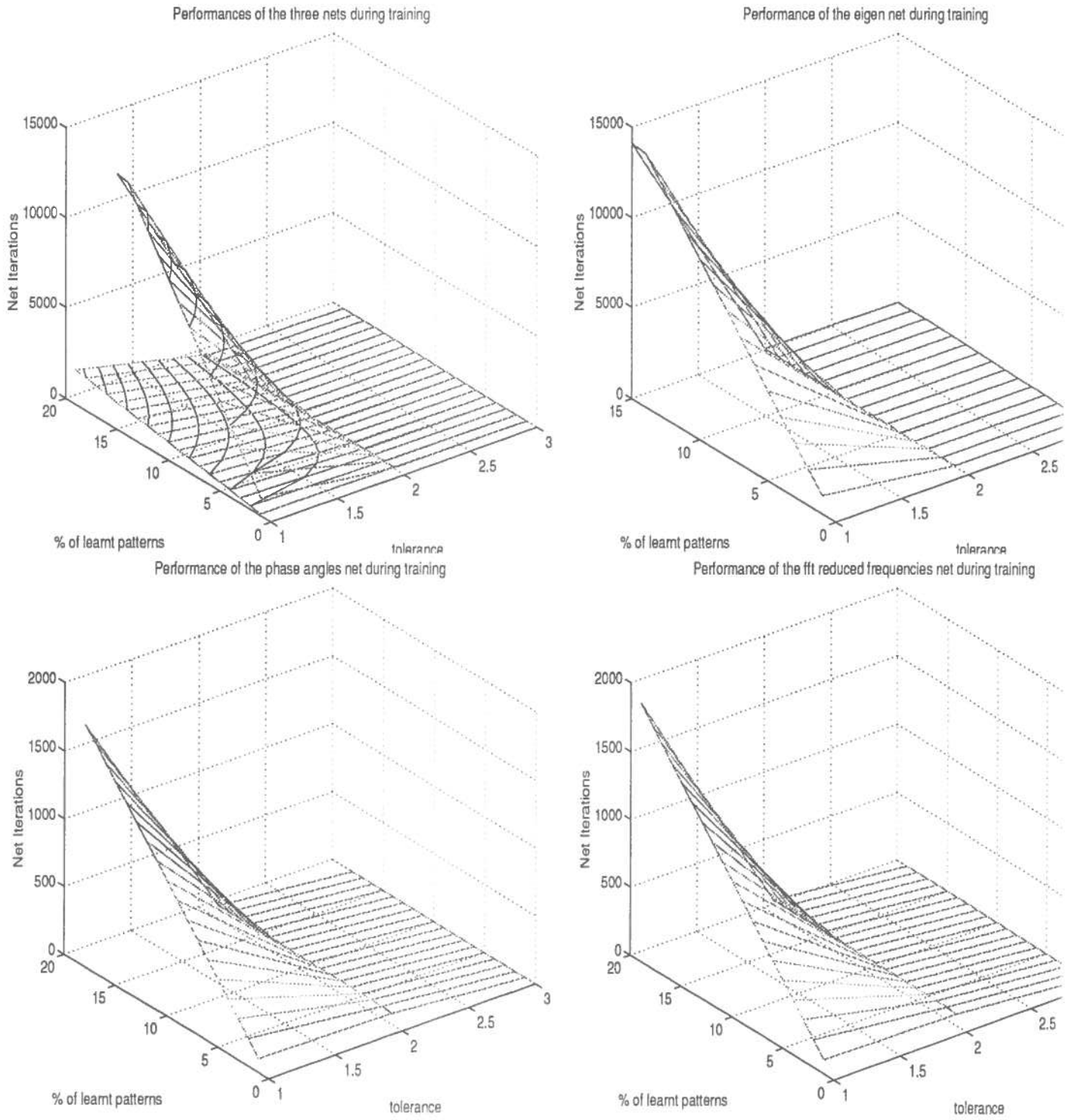


Figure 4–2: Comparative three nets resuts,eigen net, phase angles net and fft reduced frequencies net, during training

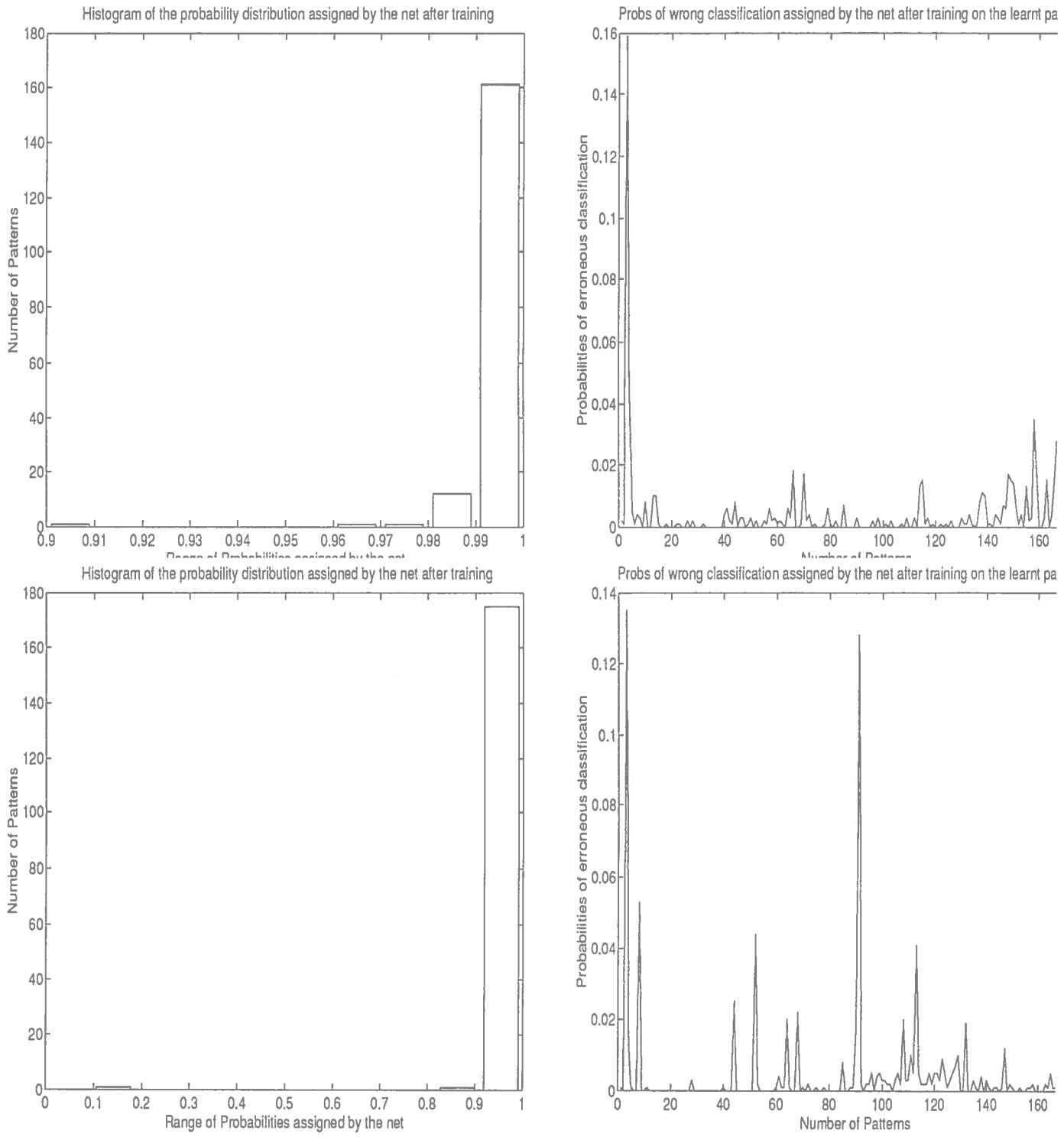


Figure 4–3: Probabilities of correct classification of learned patterns after training and Probabilities of erroneous classification for the same patterns for the Alpha Net, and the Phase Angles Net

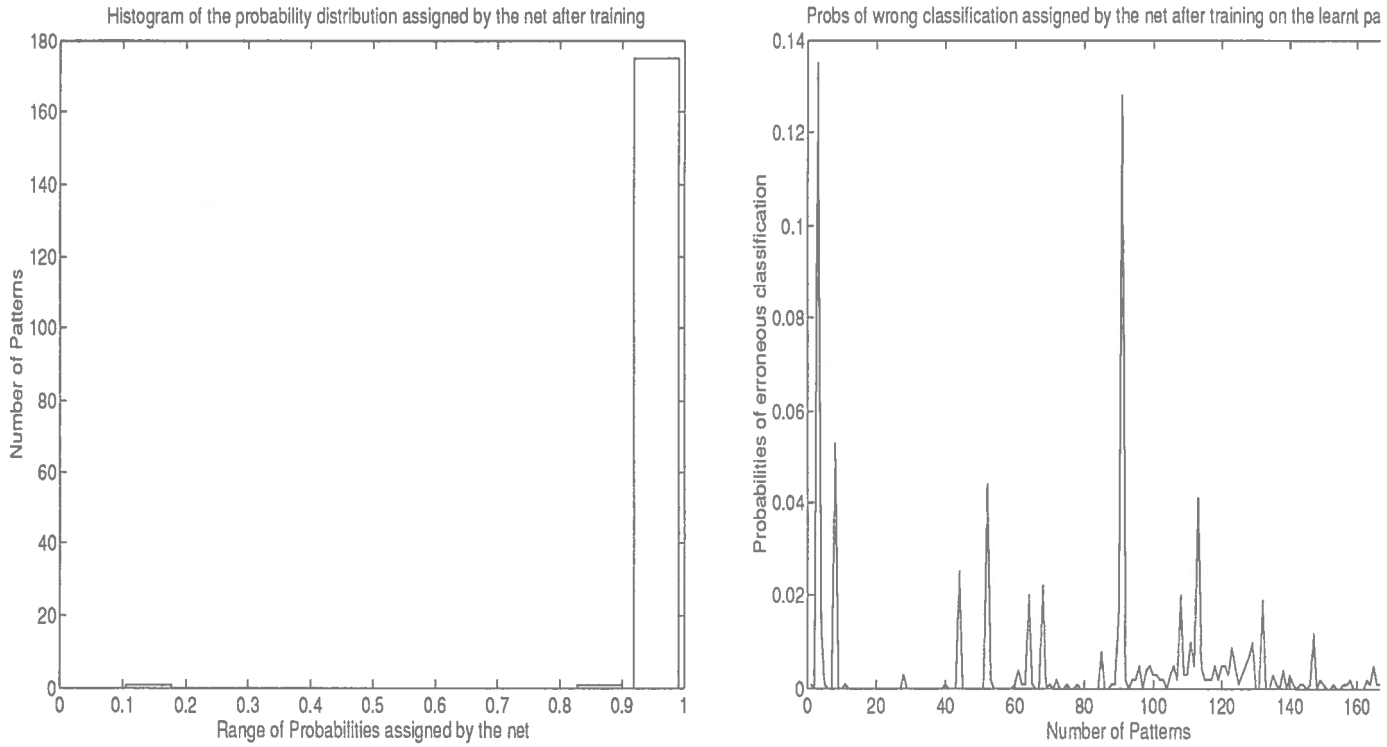


Figure 4–4: Probabilities of correct classification of learned patterns after training and Probabilities of erroneous classification for the same patterns for the FFT Reduced Frequencies Net

From all the tables we can conclude that :

- All three nets learned 100% of the patterns
- The error per unit was small

Chapter 5

Results

5.1 Postprocessing and verification procedure

When the net was fully trained and the weights file complete, a verification procedure took place during which, a matlab program scanned a face the net had not been trained with and extracted windows. That program saved all the extracted windows with dimensions 12x30 or 16x32 accordingly in a big file, as well as the coordinates of the top left corner of each window in a separate file. That second file was later used for drawing up the best windows and projecting them on the normalised facial picture.

The file that was holding the extracted windows was then processed by each of the three previously mentioned techniques; eigenvectors, fft ft angle data and fft ft reduced frequencies. The processing was done in stages. Matlab could not read big files, therefore it was read and processed consequtive parts of the extracted output. The results of this processing were kept in another file. Another matlab program was reading in the latter and was delimiting the pattern lengths with the letter 'p', something that was vital to the net. In this way the net was informed that it was fed with patterns that it was expected to recognise. The pattern lengths depended on the processing technique, ie 60 for the eigenvectors, 360 for the fft ft angle data and 114 for the fft ft reduced frequencies. When about 1000 extracted images were processed in this way, the entire procedure had to terminate. The coordinates of the last extracted window were increased by one and were used again as a starting point for a new set of extractions. This was done manually, since matlab was unable to run for long periods of time unattended since it was reducing dangerously the 40Mb swap space of the Sun IPX SPARC model used

to a few Kbs. The entire identification procedure for each face was taking a little less than 5 days to complete.

The trained net was then fed with these patterns and it was expected to identify each pattern as either left eye, right eye, nose, mouth, or nothing.

The result of this identification procedure was held in a file as well, since it was necessary for feedback.

The coordinates file that was created during the scanning of the normalised image along with the probabilities given by the net, was giving eventually information about each window and the probabilities assigned by the net. The probabilities file was laid out in four columns. The position of each column represented respectively a feature. The features had been coded as :

1. left eye 1
2. right eye 2
3. nose 3
4. mouth 4

In the case somebody wanted to identify what probabilities had been assigned by the net to each feature, s/he just had to look at this file.

From the coordinates file and the above mentioned file, another matlab program derived the top left corner window coordinates along with the probabilites assigned for each window. With this information, the program was projecting the exact position of the identified window on the actual image.

The probabilities assigned by the second net configuration, the one corresponding to the eigenvectors' technique, were quite big. For this reason a filtering was done on the initially produced probabilities; probabilities over 0.9, were selected and were kept along with the coordinates of the windows they corresponded to in a file called *best window*. It was this file that was eventually projected on the actual image, in order to show the successes and/or failures of the net. For practical reasons, instead of the top left corner, the center of the chosen window was projected; in this way the outside viewer can have a better understanding about the success or failure of the net.