Dimension reduction of images using neural networks

Master thesis

J.A. Spierenburg

July 1997

Supervisors:

Prof. Dr. J.N. Kok
Leiden University
Department of Computer Science

Dr. D.P. Huijsmans
Leiden University
Department of Computer Science

Abstract

Dimension reduction can be seen as the transformation from a high order dimension to a low order dimension. An example is the reduction of a cube in 3D (xyz) to a square in 2D (xy), to a point in 1D (x). The main goal of dimension reduction is to concentrate on vital information while redundant information can be discarded. Various ways are developed to reduce dimensions. Reduction methods can be distinguished into linear and non linear dimension reduction. In this thesis we shall present a linear and some non linear dimension reduction strategies which are mainly based on neural networks. A reduced representation of data makes the data easier to handle. With this in mind, we have developed some applications that make use of reduced representation of data. The main application is face recognition which uses a non linear dimension reduction strategy to reduce a face image into no more than say one to five vital values. These values are then used to classify the face image.

Acknowledgments

This Master thesis was developed during February - July 1997 at the department of Computer Science at Leiden University. I would like to thank my supervisors Prof.J.N. Kok and Dr.D.P. Huijsmans for their support and suggestions during this period. The various demonstrations initiated by Prof.J.N Kok were instrumental in improving my presentation skills.
I am grateful to Drs.M. Lamers for his enthusiasm and supply of the numerous papers in the early stages. Further, I would like to mention my colleagues at Leiden University who showed their interest and who helped me to relax when needed. Finally, I want to mention my family, especially my parents, who supported me and backed me up during my studies.

Joost Alexander Spierenburg
July 1997

Contents

Chapter 1

Introduction

In this master thesis research on dimension reduction of images will be described.

Generating a good data representation is of great importance before any further processing can be done. Most applications suffer from high dimensional input data; reduction of the data to a lower dimension makes the data easier to handle. Thus it is important to extract the principal features of the data and remove redundant information in such a way that structural relationships of the data distribution are preserved. This so-called "mapping method" is also known as dimension reduction. There exists a variety of ways to accomplish dimension reduction. We can classify dimension reduction methods in linear and non-linear methods. Principal Components Analysis (PCA) is a well known method for linear dimension reduction. Vector Quantization (VQ) and *five layer* auto-associative Neural Networks (5LN) are examples of non-linear dimension reduction.

We apply the dimension reduction on face images, which are to be considered as non-linear. We use face images because one can then compute and visualize the performance difference between linear and non-linear methods. Another advantage of using face images is that face recognition can be carried out. This recognition of faces is accomplished in the reduced dimension obtained by dimension reduction.

Suppose a face image has dimension 10×10, then one can see this face image as one vector of length 100 *(concatenate the rows of the face image)*. This vector of length 100 is also **one** point in dimension hundred. The PCA method adjusts the origin in dimension 100 and places the new origin at the center of all the face image points. Then the axes are realigned in such a way that the distances between all face image points and the axes are minimized. The VQ method is based on a neural network, which is a system that is capable of learning patterns. The VQ tries to find correlations among the

face images. The correlated face images found by the VQ are gathered in clusters. When these clusters are found, one can apply the PCA method within each cluster. The 5LN method is based on another type of neural network. This type of network tries to learn all of the individual face images. The network system must internally store relevant *(correlated)* information on all of the face images

In this research we have the following objectives [1]:

- Compare performance differences computed for the following dimension reduction methods
  - PCA based on the Karhunen-Loève transform
  - Vector quantization with Euclidean partitioning
  - Vector quantization with gradient partitioning *This is an own developed method that uses more information when clustering the face images*
  - Five layer feed-forward neural network
- Real time face recognition using a reduced dimension
- Morphing
- Gender recognition

We reduce the image dimension (our images are of dimension $92 \times 112 = 10304$) to the fifth dimension. The morph as well as gender recognition also use this reduced fifth dimension. The use of only 5 values in representing an image is explained in the chapter 'Five layer networks'.

This thesis is structured as follows. In Chapter 2 it is explained how the Karhunen-Loève transform can be applied to reduce the face images. In Chapter 3 an overview is given of neural networks and all related aspects such as the learning of a network. In the next Chapter 4 it is described how to use five layered networks to reduce the face images. In Chapter 5 an introduction to vector quantization neural networks is given. In Chapter 6 the use of the vector quantisizer which is based on the Euclidean distance is described and Chapter 7 is also dedicated to a vector quantisizer, but this time to a quantisizer using more information on the images when reducing the dimension. The obtained results with the various dimension reduction methods are handled in Chapter 8. The applications morphing, face recognition and gender recognition which are all based on dimension reduction are

[1] The research is mainly inspired by the paper by Kambhatla & Leen [1]

described in respectively Chapter 9, 10 and 11. Finally, the conclusions and recommendations for future research are given in the last Chapter 12.
An overview of the handled dimension reduction methods and applications in this thesis is shown in Figure 1.1.

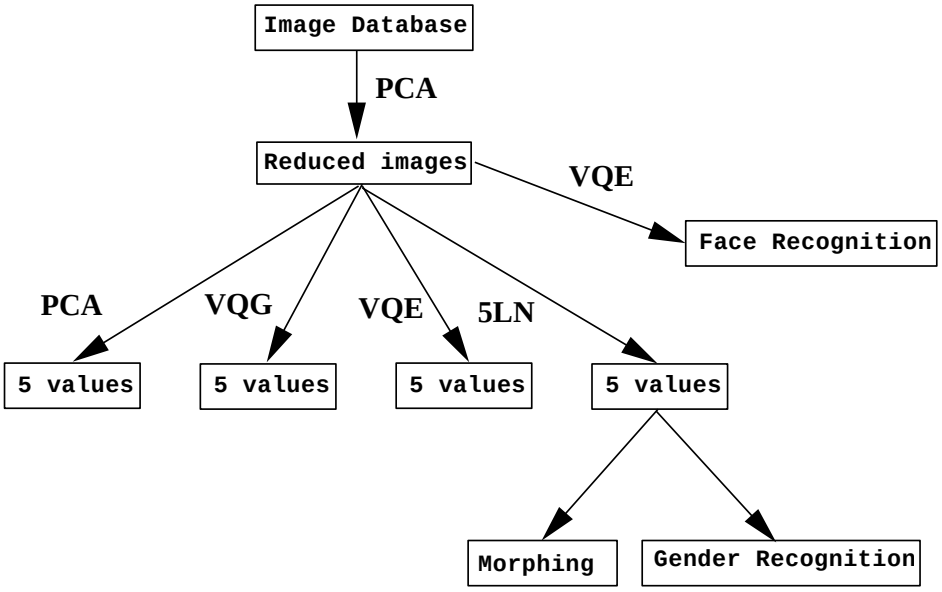

Figure 1.1: Dimension reduction tree and applications

Chapter 2

Karhunen-Loève transform

The KLT *(this transform also is commonly referred to as the eigenvector, principal component or Hotelling transform)* is a well known linear method for dimension reduction. We use the KLT based on the description by Turk & Pentland [2, 3, 4, 5, 6, 7, 8] and Gonzalez & Woods [9]. The goal of the KLT is to create a feature space that spans the significant variations among the face images. The significant features are known as "eigenfaces," because they are the eigenvectors (*principal components*) of the set of face images. The eigenvectors are ordered, such that the first eigenvector accounts for the largest variance and the last eigenvector for the smallest variance among the face images. Each $N \times N$ face image can be regarded as one point in the N^2 dimensional space.

Let the set of face images be $\Gamma_1, \Gamma_2, \Gamma_3, \ldots, \Gamma_M$. The average image of the set is defined by $\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$. Each face image differs from the average by the vector $\Phi_i = \Gamma_i - \Psi$. The covariance matrix C of the face image set is defined as

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

Because C is real and symmetric, finding the M orthonormal vectors u_n which describe the face images best is possible. Element c_{ii} is the variance of Φ_i and element c_{ij} is the covariance between elements Φ_i and Φ_j. If elements Φ_i and Φ_j are non-correlated, their covariance will be 0 and,therefore $c_{ij} = c_{ji} = 0$.

The kth vector u_k is chosen such that

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}\left(u_k^T\Phi_n\right)^2 \tag{2}$$

is maximum, subject to

$$u_l^T u_k = \begin{cases} 1, & \text{if } \; l = k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

in which the eigenvectors are u_k, and the eigenvalues are λ_k of the covariance matrix C in (1).

The eigenvectors v_i of C are used to create the eigenfaces u_l of our face images set:

$$u_l = \sum_{k=1}^{M} v_{lk} \Phi_k \quad l = 1 \dots M \tag{4}$$

A face image Γ is transformed into its eigenface components (*projected into "face space"*) by a simple operation,

$$\omega_k = u_k^T (\Gamma - \Psi) \tag{5}$$

for $k = 1, \dots, M'$. The eigenfaces span an M'-dimensional subspace of the original N^2 image space. See Figures 2.1 and 2.2 for examples of a transformation from points to eigen/face-space. The weights form a vector $\Omega^T = [\omega_1, \omega_2 \dots \omega_{M'}]$ which describes the contribution of the eigenfaces in forming the face-space projection of the input face image. So each image has it's own weight vector of length M' which describes it's contribution to each eigenface. The "inverse projection" or reconstruction, from M' to M can be reconstructed from (5) which results in

$$\Gamma = u_k^T w_k + \Psi \quad \text{for} \quad k = 1 \dots M' \tag{6}$$

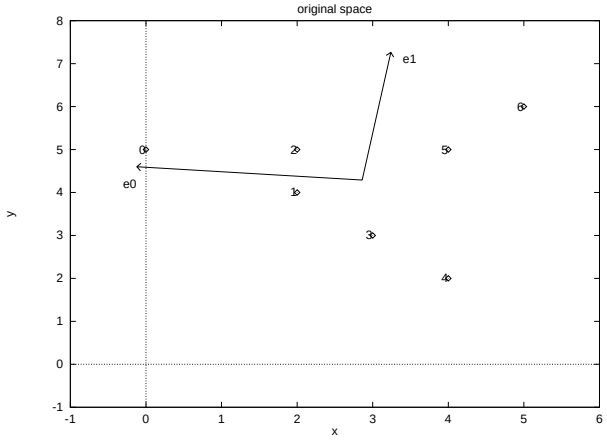

Figure 2.1: Original points

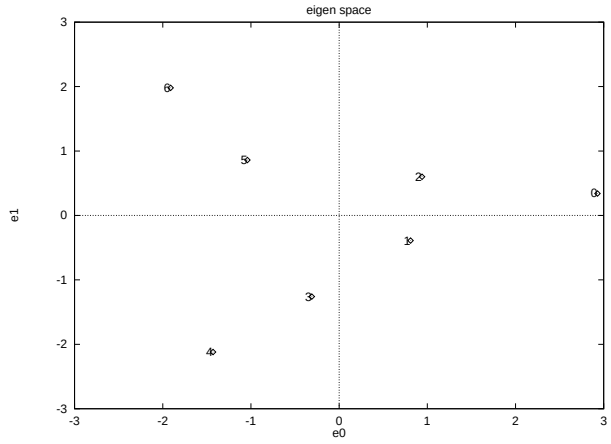

Figure 2.2: The points of Figure 2.1 in eigen/face-space

Chapter 3

Neural networks

Neural Networks can be seen as the mathematical representation of the human brain, which is capable of learning. The human brain consists of over 10^{11} neurons, which are nerve cells which respond on electrical currents. Each nerve cell contains lots of dendrites which are connected to the cell-body. Extending from the cell-body there is a single long fiber called the axon which eventually branches into strands and sub-strands. Synapses are located at the end of these strands, making it possible to transmit to other neurons. The cell-body and dendrites are the receiving ends of a neuron whereas the axon is the transmitting end. All transmissions occur through synapses. When the current in the axon is high enough, the synapse will pass it through onto the next neuron. If the current is too low, the electrical current will not be carried through that synapse. The threshold that determines whether the current will pass onto the next neuron is called the action potential.

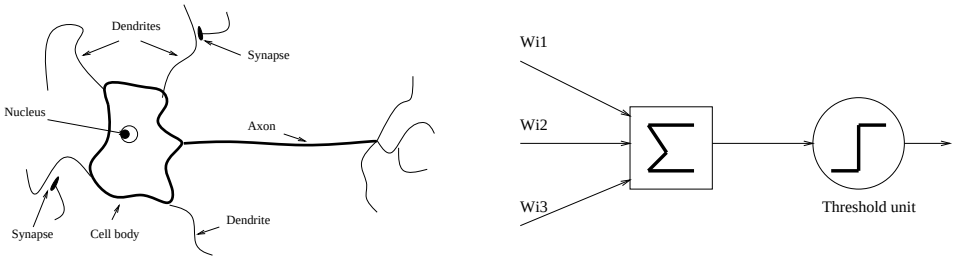

Figure 3.1: Biological and McColloch Pitts neurons

McCulloch and Pitts [1943] [10] proposed a simple model of a neuron as a binary threshold unit. Specifically, the model neuron computes the weighted sum of it's inputs from other units, and outputs a one or a zero according to whether this sum is above or below a certain threshold. This unit is the

building block of neural networks. Rosenblatt [1962] [10] was one of the first to describe a learning algorithm for neurons. He used one-way connections between neurons and concentrated on networks called perceptrons. The perceptron networks were organized into layers with feed-forward connection between a layer and the next.

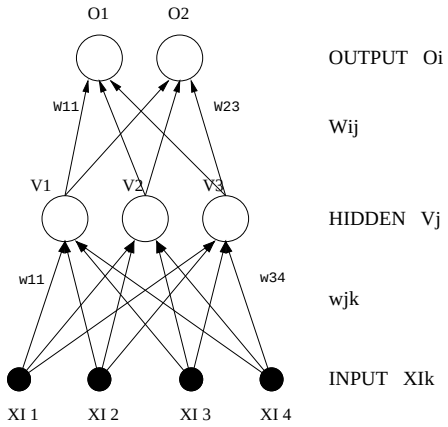

Figure 3.2: A three layer feed-forward perceptron

3.1 Network learning

Neural nets can learn by storing information on the connections (weights) between units. If a new pattern is to be learned, then weights between the units must be adjusted. There are two main learning strategies: supervised and unsupervised learning.

- Supervised learning is done on the basis of direct comparison of the output of the network with the known correct answers. It's also referred to as learning with a teacher.
- Unsupervised learning has no teacher; the only information available is in the correlations between the input data. The network is expected to create categories from these correlations, and to produce output signals corresponding to the input category.

Training a network is done by applying input and, if available, target data so that the network can learn. One aspect of a learned neural network is that it is capable of generalization. This means that the network will probably respond well on data that it has never processed during learning.

3.2 Back-propagation

A well known and frequently used learning rule is back-propagation. It is a supervised learning rule based on gradient descent. We will discuss this learning rule applied to the three layer feed-forward network in Figure 3.2. We use the following terminology:

- An input-output pair is denoted as $\{\xi_k^\mu, \zeta_i^\mu\}$
- The output units are denoted by O_i
- Hidden units by V_j
- Input terminals by ξ_k.
- Connection from unit k to unit j is denoted as w_{jk}

There are connections w_{jk} from the inputs to the hidden units, and connections W_{ij} from the hidden units to the output units. Input units are always clamped to particular values. A pattern has label μ, so input k is set to ξ_k^μ when pattern μ is being presented. We consider a network with M layers $m = 1, 2 \ldots, M$ and use V_i^m for the output of the ith unit in the mth layer. V_i^0 is the same as ξ_i, the ith input. Let w_{ij}^m mean the connection from V_j^{m-1} to V_i^m. The objective of this learning algorithm is to find

$$O_i^\mu = \zeta_i^\mu \tag{7}$$

The threshold *(also referred to as transfer or activation function)* used is the function $g(h) = \tanh \beta h$, and the learning-rate parameter is η. The transfer function is shown in Figure 3.3.

To monitor the learning of the network *(the evolution of the weights)*, a error or cost function $E[w]$ is used

$$E\Big[w\Big] = \frac{1}{2}\sum_{\mu i}(\zeta_i^\mu - O_i^\mu)^2 \tag{8}$$

The better our w_{ij}'s become, the more $E[w]$ will go to zero as we approach a solution of (8). It must be mentioned that gradient descent back-propagation is very slow, especially when more hidden layers are used. Here follows the step-by-step procedure of back-propagation, taking one pattern μ at a time (i.e., incremental updates)

1 Initialize the weights to small random values [1]

[1] To avoid local minima Palmer [10] suggests setting weights w_{ij} to be of the order of $\frac{1}{\sqrt{k_i}}$ where k_i is the number of j's which feed-forward to i (the fan-in of unit i)

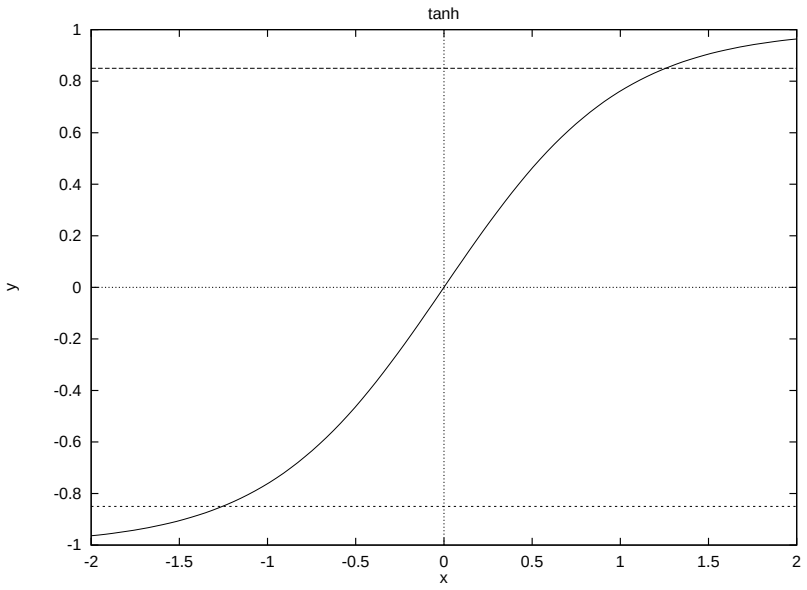

Figure 3.3: The transfer function

2 Choose a pattern ξ_k^m and apply it to the input layer (m=0) so that

$$V_k^0 = \xi_k^\mu \quad \text{for all} \quad k \tag{9}$$

3 Propagate the signal forwards through the network using

$$V_i^m = g(h_i^m) = g\Big(\sum_j w_{ij}^m V_j^{m-1}\Big) \tag{10}$$

for each i and m until the final outputs V_i^M have all been calculated.

4 Compute the deltas for the output layer

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M] \tag{11}$$

by comparing the actual outputs V_i^M with the desired ones ζ_i^μ for the pattern μ being considered.

5 Compute the deltas for the preceding layers by propagating the errors backwards

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \tag{12}$$

for $m = M, M-1, \ldots, 2$ until a delta has been calculated for every unit.

6 Use gradient descent

$$\Delta w_{ij}^m = -\eta \frac{\delta E}{\delta w_{ij}} = \eta \delta_i^m V_j^{m-1} \tag{13}$$

to update all connections according to $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$.

7 Go back to step 2 and repeat for the next pattern.

3.3 Auto-associative and hetero-association

Auto-associative *(also referred to as auto-encoding or identity mapping)* neural nets are neural nets that try to learn the identity mapping. This means that the input is also the required output. When the hidden layer(s) between the input and output layer have fewer units, the network gets a hour-glass shape. This kind of network can be used for dimension reduction because the network has found a more compact representation of the input data. It is clear that when fewer units in the hidden layer(s) are used the net will have to find a more complex representation of the data, so learning times will increase. At a certain point the network can no longer learn the required encoding and more units in the hidden layer(s) are needed. The hour-glass shape of the network can be regarded as a joining of two networks. The first net will perform a encoding of the data to a lower dimension, while the second net will perform a decoding of the reduced data to the original data dimension.

Hetero-associative neural nets are nets in which the number of units in the input and output layer are not equal. These kind of nets are used for classification purposes. An example is the *XOR* [10] problem which has two binary inputs and just one binary output.

3.4 Momentum

As mentioned before, back-propagation is very slow, especially when η is small. If the learning-rate η is too large, the gradient descent will oscillate widely. One technique to speedup the learning process of a neural net and to avoid undue oscillations involves the use of a momentum term. The idea is to give each connection w_{pq} a momentum , so that the gradient descent tends to change in the direction of the *average* downhill direction that it encounters, instead of oscillating. The momentum term can be applied as follows

$$\Delta w_{pq}(t+1) = -\eta \frac{\delta E}{\delta w_{pq}} + \alpha \Delta w_{pq}(t) \tag{14}$$

where t is a time step and α indicates the momentum parameter. α has domain $[0 \ldots 1]$, and is often chosen to be 0.9 [10].

3.5 Adaptive parameters

Correct values for η and α are difficult to find for particular problems. Moreover the best values at the beginning of training may not be appropriate later on. A good solution for this problem was found by introducing adaptive parameters which evolve during evolution. This approach is based on weight updates. If several, *(say K steps)* weight updates would decrease the cost-function, then may be we could learn a bit faster by increasing η. On the other hand, if a weight change would increase the costs, then η must be reduced and the step taken must be "undone" by setting $\alpha = 0$. This can be done with

$$\Delta\eta = \begin{cases} +a & \text{if} \quad \Delta E < 0 \quad \text{consistently} \\ -\beta\eta & \text{if} \quad \Delta E > 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where ΔE is the cost function change, and a and b are appropriate constants.

3.6 Learning by pattern versus learning by epoch

Basically there are two possible methods for computing and performing weight update, depending on the moment when the update is to be performed. In the "learning by pattern" method, a weight is updated after each presentation of an input-output pair. It can be seen as a method that minimizes the cost function for each individual pair, trying to minimize the overall error. An alternative method is "learning by epoch" *(also referred to as batch learning)*. The weights gather information on gradient descent for each pattern, and when the whole pattern set has been presented the weights are updated. This way of weight updating tries to minimize the summed error of the pattern set, in order to reduce the total cost function. We use this "learning by epoch" method in our research.

3.7 Generalization

Generalization is the reaction of the network on inputs that were not used during training. Generalization is a nice aspect of neural nets, but it is hard

to train a network that will master generalization. If the training time of the network is too short, the net will not find the correlations between the face images. On the other hand, if training time is too long, only the face images presented during training will be known.

To obtain a network that will have generalization, the following strategy can be followed:

```
Generalization algorithm

begin
  WHILE NOT stop DO
    Train net 1 epoch
    compute error on training set
    compute error on test set
    IF (error on test set has increased AND
        error on training set has decreased
        in the last K epochs) THEN
      stop learning
    fi
  od
end
```

This can imply that the learning process is stopped while the error of the net is quite large, but we are certain to have achieved generalization.

Chapter 4

Five layer networks

One hidden layer Auto-associating feed-forward networks are capable of extracting the principal components *(this is linear dimension reduction)* of the data [11, 12, 13]. Addition of one or more hidden layers between the inputs and representation layer *(the representation hidden layer is the middle layer of the network)*, and between the representation layer and the outputs provides a network that is capable of learning non-linear representations (*Kramer 1991, Oja 1991, Nakauchi & Nakano 1991* [10]). When the input dimension is known, the output dimension must be of the same order. The dimensions of the other three, hidden layers must be smaller than the input and output dimensions, but the exact size is not known beforehand. There are algorithms that can compute the number of units needed for the hidden layers, but these algorithms are time consuming. DeMers and Cottrell [13] used an algorithm that tried to prune units in the representation layer. They started with a large representation layer in their auto-encoding neural net, and after numerous iterations they could prune a unit. They continued with this strategy until the number of nodes in the representation layer was too small to encode the data. The conclusion from DeMers and Cottrell [13] was that their algorithm found a minimal representation layer size of five, while the identity map could still be learned. DeMers and Cottrell [13] used extra connections from the nodes in the representation layer to the nodes in the output layer. These extra connections provide a larger degree of freedom in the decoding of the data. Because we were surprised that only such small number of representation nodes appeared to be required we decided to try to achieve similar results. Thus when the input, output and representation layer dimensions are known, only two unknown sizes for the remaining hidden layers are left. Thus our five layer feed-forward auto-associative neural network has the following configuration:

$$17 - n - 5 - n - 17$$

if we have an input and output dimension of seventeen. The neural net has a faster response if the number of edges is small, and the number of nodes is minimal.

It makes no sense to reduce the representation layer to a smaller dimension than five, because five layer networks are very slow to train and are prone to become trapped in poor local minima. Moreover, the use of a smaller representation layer would prevent us from comparing our results with those found by Kambhatla and Leen [1].

Chapter 5

Vector quantization

Unsupervised competitive learning is a case of learning in which output units compete for being the one to trigger. The aim of these kind of networks is to cluster or categorize the input data. Similar inputs should be classified as being in the same category. The network must find the correlations between the input data in order to carry out classification. This kind of clustering is known as *vector quantization*. The classes obtained by quantization of the input space are called *Voronoi cells*, and each Voronoi cell has a reference vector which is the center of that cell, see Figure 5.1 for an example of a Voronoi cells. We will now discuss the working of a unsupervised competitive

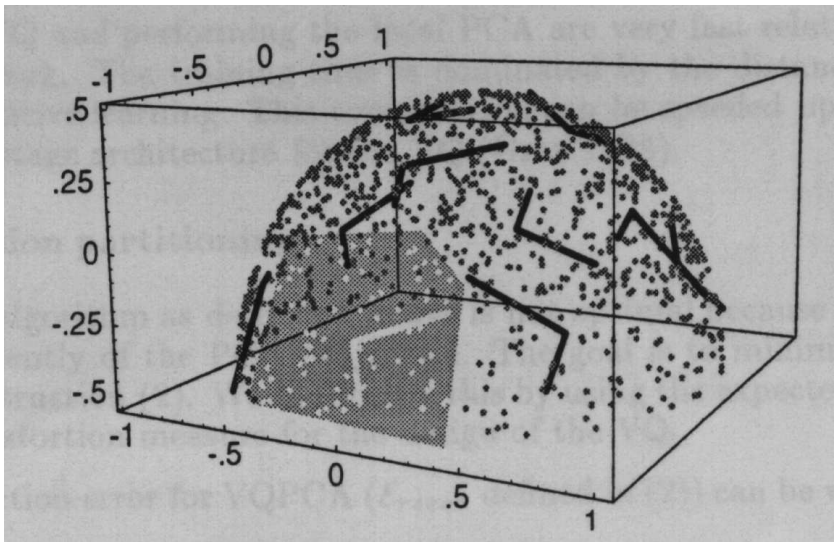

Figure 5.1: Voronoi cells

neural net.

The network consists of a single layer of output units O_i, each fully connected to a set of inputs ξ_j, via weights $w_{ij} \geq 0$. Only one of the output units, called the winner, can trigger at a time. The winner is the unit with the largest net input

$$h_i = \sum_j w_{ij}\xi_j = w_i \cdot \xi \tag{16}$$

for the current input vector ξ. Thus

$$w_{i^*} \cdot \xi \geq w_i \cdot \xi \quad \text{for all} \quad i \tag{17}$$

where the winning unit is defined as i^*. This means that the winner unit is the unit closest to the given input ξ. Now the winner unit updates it's weights w_{i^*} so that vector w_{i^*} is drawn closer to the given input vector, and it will win more easily the next time when the same input is presented to the network again. This strategy is practised for all patterns μ and repeated several times creating patches of clusters. The mentioned updating of the weights is carried out according to the standard competitive learning rule

$$\Delta w_{i^*j} = \eta(\xi_j^\mu - w_{i^*j}) \tag{18}$$

There is, however, a drawback with this method of clustering. Units w_i far from any input vector may never win, and therefore never learn *(such units are called dead units)*. One way of solving the problems connected with such *dead units* is using *leaky learning* which also updates the losing units, but with a much smaller learning-rate than η. To encourage wide exploration at the beginning of learning and refinement of the weights after a while, it is useful to decrease the large initial η during learning to a smaller η. The function that we use for modifying the learning-rate is given by

$$\eta^{(t)} = \eta^{(0)}(1 - \frac{1}{T}) \tag{19}$$

in which t is the current iteration, $\eta^{(0)}$ is the initial learning-rate value, typically between 0.2 and 0.7, and T is the total number of training iterations to be performed. The great advantage of unsupervised competitive learning is that it is very fast compared with supervised back-propagation learning. It is recommended not to use a excess of Voronoi cells, because learning becomes slow and cells with hardly any items can appear.

Chapter 6

Vector quantization Euclidean partitioning

Kambhatla & Leen [1] introduced a method for performing non-linear dimension reduction. Their method clusters the input space using vector quantization, and when the Voronoi cells are created *local* PCA is performed in each local region cell. The hybrid algorithm proceeds in three steps

1 Using competitive learning, train a VQ with *Euclidean* distance with Q reference vectors (weights) $(r_1, r_2, \ldots, r_Q)$. The distance measure used to determine which unit is to be the winner is computed with the *Euclidean* rule

$$||w_i - \xi|| \tag{20}$$

the winning unit i^* corresponds to the unit with the minimum *Euclidean* distance to ξ.

2 Perform a *local* PCA within each Voronoi cell of the VQ. For each cell, compute the local covariance matrix for the data with respect to the corresponding reference vector (centroid) r_c. Next compute the eigenvectors $(e_1^c, e_2^c, \ldots, e_n^c)$ of each covariance matrix.

3 Choose a target dimension m and project each data vector x onto the leading m eigenvectors to obtain the local linear co-ordinates $z = (e_1^c \cdot (x - r_c), \ldots, e_m^c \cdot (x - r_c))$.

The encoding of x consists of the index c of the reference cell closest (*Euclidean* distance) to x, together with the $m < n$ component vector z. The decoding is given by

$$\hat{x} = r_c + \sum_{i=1}^{m} z_i e_i^c \tag{21}$$

where r_c is the *reference vector* (centroid) for the cell c, and e_i^c are the leading eigenvectors of the covariance matrix of the cell c. The mean squared reconstruction error incurred by *Vector quantization Euclidean Partitioning* is

$$\varepsilon_{recon} = E\Big[||x - \hat{x}||^2\Big] = E\Big[||x - r_c - \sum_{i=1}^{m} z_i e_i^c||^2\Big] \tag{22}$$

where $E[\cdot]$ denotes an expectation with respect to x, and $\hat{x}$ is as defined in (21).

Chapter 7

Vector quantization gradient

Vector quantization uses the feature vectors which are obtained through PCA to generate clusters. With more information about the images, better clustering may be achieved.

A gradient filter can be used to detect edges in images. One can use the PCA reduced information of the gradient images and combine it with the already known PCA reduced information. One can concatenate both feature vectors and apply vector quantization, in order to obtain clusters that are based on both gradient and normal image features. This method is shown in Figure 7.1.

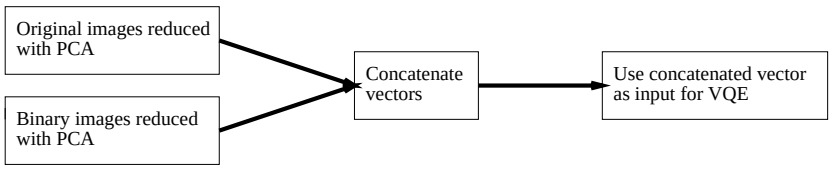

Figure 7.1: VQG graph

The gradient of an image $f(x, y)$, at coordinates (x, y) is defined as vector

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix} \tag{23}$$

This gradient vector points in the direction of the maximum change rate of f in (x, y). We approximate the gradient ∇f with absolute values:

$$\nabla f \approx |G_x| + |G_y| \tag{24}$$

We have to compute the derivatives G_x and G_y at every pixel location. Derivatives can be implemented in digital form by using the Sobel operators [14]. These operators are shown in Figure 7.2

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Figure 7.2: Sobel operators

By moving the 3x3 masks over an image, we can compute the gradient for each pixel with (24). When all possible image locations have been masked, a gradient image is obtained. A nice property of Sobel operators besides differencing (enhancing the edges) is the smoothing effect. Because all derivative filters enhance noise, we like the image to be smoothed which is done automatically by Sobel.

A threshold can be used to create a binary [0,1] image of the gradient image. The chosen threshold value is $\frac{1}{4}\times$ number of intensity levels (256) according to Egas [15]. A binary image accounts only for the most specific contours of an image, which is desired in order to accomplish that 'look alike' images appear in the same cluster.

Figure 7.3: Original image, Sobel image and Binary (thresholded) image

Chapter 8

Experimental results of dimension reduction methods

Before we discuss the obtained results, we will first explain how we pre-process our face images and how we split up our face database.

8.1 Face image database

Our face image database is a selection from the Olivetti face database. [1] Each person-image has 10 different front viewed expressions. In total there are 400 grayscale 92×112 face images. Because Kambhatla and Leen [1] use a face image database consisting of 160 front views of 20 different people (10 males and 10 females), we have decided also to create such a 1:1 environment. Due to the fact that the Olivetti database only stored 5 female-images, we found it appropriate to use 5 male-images in order to maintain a evenly balanced database. Thus we have created a face image database comprised of 100 front views of 10 different people. The 10 different people are shown in Figure 8.1. It is important to split the face image database in a test-set and a training-set. Both test and training sets must be good representations of the whole database in order to get good results with the mentioned five layer dimension reduction network. We shall use a test-set of maximum 30 images and a training-set of maximum 70 images. The test-set will only be used for determining the stop criteria for the five layer neural net and the gender net in order to obtain generalization. All results obtained are computed according to the training-set.

An important advantage of using the Olivetti database is the fact that the Olivetti database face images have been constructed with special high

[1] The Olivetti face database can be found on the Internet at http://www.cam-orl.co.uk/facedatabase.html

Figure 8.1: Training-set males and females

resolution camera's operated by professionals. In our research environment, we can use the Indy camera [2] to take pictures, but this camera suffers from a very low resolution. Moreover when creating our own face image database we recognized that it is very difficult to obtain face images that have been produced under the same light conditions; for example: a cloud before the sun can cause a shift in the pixel intensities, which has the effect that correlations among the images are harder to find and therefore our reduction methods would have an unnecessary difficult task in finding the desired correlations.

8.2 Pre-processing

Instead of taking the whole grayscale (8 bit) face images as input data for our methods, we reduce the face images with PCA *(KLT)* to a lower dimension. To determine what the reduced dimension must be, we plot the normalized sorted eigenvalues. From Figure 8.2 we can determine the required dimension by looking at which point the curve starts to 'flatten'. We found that using a threshold value of 0.04 acceptable data representations can be produced whilst only 17 eigenvectors are needed. Instead of using a threshold value for determining how many eigenvectors to use, one can also take 99% of the eigenvalues. This will increase the number of eigenvectors to be used in comparison with the threshold method. The 99% method will give a better representation of the face images but the input dimension for all the dimension reduction methods will be quite large. This is an aspect

[2]The Silicon Graphics Indy workstations are standard equipped with a camera (IndyCam)

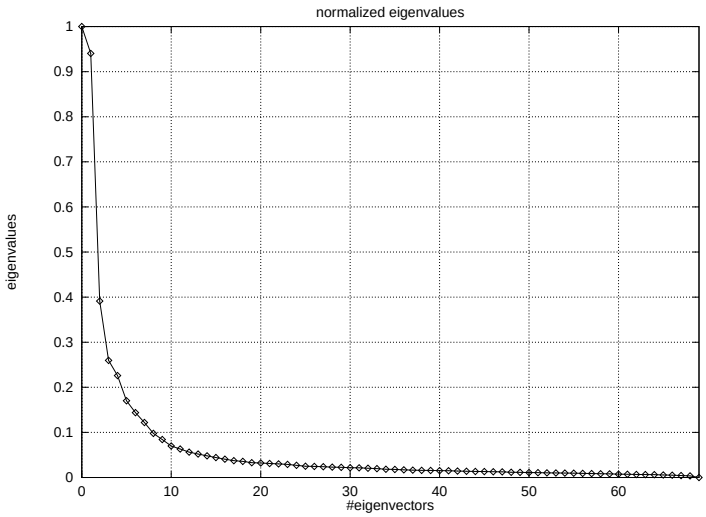

Figure 8.2: Normalized eigenvalues

that imposes major drawbacks on the five layer neural network, because the number of weights in the network and the required training time will increase considerably.

8.3 Performance measure

The error function to determine the performance of the mentioned dimension reduction methods after reconstruction is computed as follows:

$$\varepsilon_{norm} = \frac{\varepsilon_{recon}}{E\left[||x||^2\right]} = \frac{E\left[||x - \hat{x}||^2\right]}{E\left[||x||^2\right]} \tag{25}$$

Average error of a method

$$\varepsilon_{average} = \frac{\sum_1^{\#images} \varepsilon_{norm}}{\#images} \tag{26}$$

The error measure is based on the reduced representations of the images in the image database. Applying the inverse projection (6) (projection from the reduced dimension to the original face dimension (92×112)), one can visualize the performance of a method.

8.4 NeuralCam

In order to test the suggested dimension reduction methods, the program 'NeuralCam' was developed. The program was written with the programming languages C and Motif and the operating system platform was UNIX (Irix 5.3). The Motif language is used to build an user interface (UI) which was necessary because we wanted to visualize our results. A brief user manual of the 'NeuralCam' program can be found in the appendix. This manual also covers the the implemented applications morphing, face recognition and gender recognition.

8.5 Network settings

Dimension reduction based on five layer neural networks requires not only a training-set but also a test-set in order to perform. As discussed in section 3.7, the test-set is used to acquire generalization. If the generalization technique is applied at the start of learning, the network shall terminate the learning process quickly because of generalization. When reviewing the results with the five layer neural network, we found that large errors occurred when face images were reconstructed. Because training times are usually high with five layer networks, it was quite clear that the generalization was applied too soon. By experimentation we found that when the error of the neural net was smaller than 2.0, acceptable face images could be reconstructed. Applying generalization when the error of the neural net is less than 2.0 is acceptable because we want to have our images reduced. The results acquired with a five layer neural network all had a threshold of less than 2.0 before putting generalization into practice. When the error of the network is less than 0.1 for the 5LN network and less than 0.01 for the 3LN gender network training is stopped. By experimentation we found that continuing training beyond the limit had no use because only the training times increased and no significantly better results were achieved. The five layer neural net is trained with the settings as presented in Table 8.1. We shall explain the various fields in Table 8.1 briefly:

- The *Test domain* indicates how large the domain is when applying generalization.
- At most, *Max Epochs* are used when training.
- The *Begin* η indicates the magnitude of the learning-rate at the start of training.

- With adaptive parameters, the learning-rate is adjusted while learning. The *End* η is used to ensure that the learning-rate does not become too small. If the learning-rate approaches zero, no progress in learning is made (learning is stalled).
- The remaining fields of Table 8.1 will be clear from the previous chapters.

Five layer settings							
Test domain	Max #Epochs	Begin η	End η	Adaptive parameter a	Adaptive parameter β	Adaptive parameter K steps	Momentum term
6	50000	0.004	0.002	0.0001	0.25	8	0.9

Table 8.1: Settings used to learn a five layer neural network

The number of Voronoi cells build with the vector quantization methods is 10% of the number of images in the image database. After the unsupervised learning session each Voronoi cell must have at least five images to be able to reduce the images to the fifth dimension. If a Voronoi cell has less than five images, the clustering must be learned again until each cell has at least five images per cell. The VQ settings used in the unsupervised networks can be found in Table 8.2

VQ settings		
η^0	leaky η	#Epochs used
0.65	0.01	1250

Table 8.2: Settings used to train a VQ network

8.6 Results

Because the five layer neural net method requires two databases it is discussed separately. The remaining methods (PCA,VQE and VQG) are tested on various sized image databases. All images are reduced with PCA to a lower dimension (chapter 8.2). The obtained reduced images are then used as input

for the mentioned methods. The methods reduce the dimension of the images further to the fifth dimension. The obtained results for the reduction to the fifth dimension can be found in Table 8.3.

	Reduction methods					
#training images	PCA $\varepsilon_{average}$	Time (s)	VQE $\varepsilon_{average}$	Time (s)	VQG $\varepsilon_{average}$	Time (s)
10	0.217	0.15	0.217	1.11	0.217	2.61
20	0.286	0.21	0.083	1.38	0.143	5.39
30	0.306	0.35	0.082	1.66	0.035	6.77
40	0.297	0.64	0.034	1.99	0.025*	7.86
50	0.293	1.05	0.035	2.10	0.019	11.20
60	0.286	1.59	0.024	2.43	0.020	14.75
70	0.288	2.27	0.019	3.03	0.019	19.00

* see page 31

Table 8.3: Reduction results

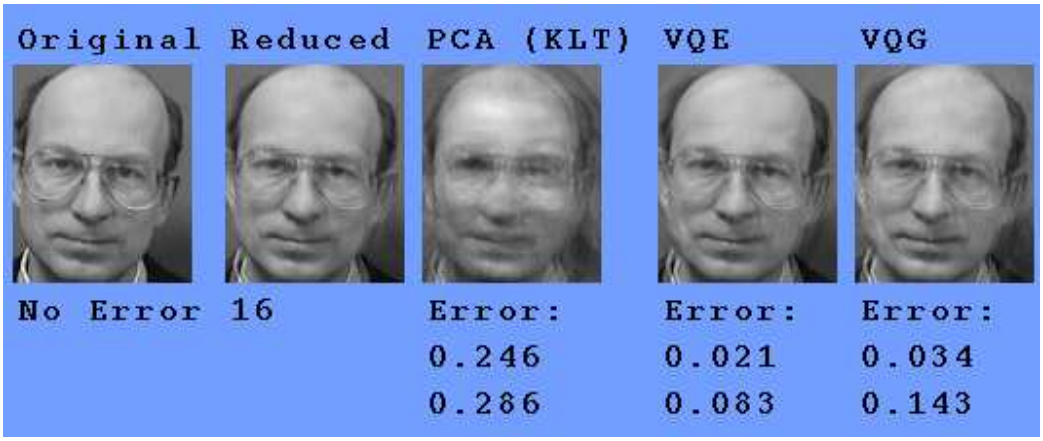

Figure 8.3: Visualized results of PCA, VQE and VQG, when applied on a database of 20 face images. The first image on the left is the original non reduced image. Next is the same image reduced with PCA to 16 values. The remaining images are the results of the reduction methods when reducing to the fifth dimension. The upper error is the error on the selected image (25), whereas the lower error indicates the error computed with (26)

From Table 8.3 it is clear that the linear reduction method PCA has the

worst performance. Vital information is lost which can be clearly seen in Figure 8.3. On the other hand, PCA is fast and when lots of eigenvectors are used, the performance is not too bad. This is why we pre-process the images with PCA. Both of the non linear vector quantization methods (VQE and VQG) show a good performance. The time consumption with VQG is much larger then with VQE because PCA is performed on the binary images before the vectors can be concatenated. This is a time-expensive operation which is not required for VQE. With only 10 images in the image database, Table 8.3 shows that there is an equal performance in dimension reduction between the methods. This is because the vector quantization methods use only 1 Voronoi cell and no clustering is obtained. This makes the vector quantization methods perform like a linear PCA. The time requirements for PCA are lower than the quantization methods because the quantization methods still have to learn the cluster. The * mark in Table 8.3 indicates that the quantization method had trouble filling the clusters with at least 5 images. An example of cell filling obtained by VQE and VQG when a database of 40 images has been applied is shown in Table 8.4. Table 8.4 shows that the VQG reduction method failed to fill the cells with at least five images.

	Cell occupation	
Cell number	VQE #images	VQG #images
0	15	8
1	9	9
2	8	19
3	8	4

Table 8.4: Comparison of cell filling between VQE and VQG while using an image database which consists of 40 face images.

The five layer neural networks are trained with three different database configurations which are displayed in Table 8.5. The dimension reduction performance of five layer neural networks is shown in Table 8.6. The time required to reduce an image is very small when using the five layered neural network. A face image only has to be propagated through the learned five layered neural net. If the weights on the edges of the neural network are stored in a file after training, then the weights can be loaded when a face image is to be reduced. This is an easy task compared with the real time learning of the vector quantization methods and the reduction method PCA. On the other hand, a time penalty is paid when training the five layered

	5LN databases	
DB number	#training images	#test images
1	30	20
2	50	20
3	70	30

Table 8.5: The various databases used with 5LN

	5LN results			
DB number	5LN $\varepsilon_{average}$	Time (s)	#Epochs used	Training time (s)
1	0.007	0.023	15504	2019.65
2	0.028	0.022	50000	10676.71
3	0.081	0.028	50000	13827.30

Table 8.6: Results of the 5LN method

neural net. The number of units in the first and third hidden layer determine the performance of the five layered network. The degree of freedom that the

	5LN results					
DB number	5LN $\varepsilon_{average}$	Time (s)	#Epochs used	Training time (s)	#Hidden units hidden layer 1	#Hidden units hidden layer 3
1	0.007	0.03	15504	2019.65	17	17
1	0.012	0.027	50000	4225.91	15	15
1	0.152	0.026	50000	4881.58	10	10

Table 8.7: Results of the 5LN method for different number of nodes in the first and third hidden layer.

neural net has to learn the images is higher when the number of nodes in these layers is high. We measured the performance and the results are shown in Table 8.7. The results displayed in Table 8.6 are obtained when a high degree of freedom was allowed. The results obtained by Kambhatla & Leen [1] are shown in Table 8.8.

When we compare our results with the results found by Kambhatla & Leen [1] we consider the quality of our results at least on a par with those

	Reduction methods					
#training images	PCA $\varepsilon_{average}$	Time (s)	VQE $\varepsilon_{average}$	Time (s)	5LN $\varepsilon_{average}$	Time (s)
120	0.458	5	0.140	163	0.350	15486
160	0.4054	7	0.0009	905	0.0729	31980

Table 8.8: Results obtained by Kambhatla & Leen

found by Kambhatla & Leen. It should be taken into account, however, that:

- Kambhatla & Leen always pre-processed their images with PCA to 50 values. We used the method described in chapter 8.2 to pre-process our images. In order to compare better, we used 50 values when pre-processing a database of 70 images. The results are displayed in Table 8.9.

	Reduction methods					
#training images	PCA $\varepsilon_{average}$	Time (s)	VQE $\varepsilon_{average}$	Time (s)	VQG $\varepsilon_{average}$	Time (s)
70	0.398	3.34	0.079	3.09	0.056	18.85

Table 8.9: Results obtained when pre-processing to 50 values is applied

- The dimensions of the images in the the database used by Kambhatla & Leen are 64×64. Recall that we used images of dimension 92×112. In smaller dimensions PCA can minimize the distances between the points easier because there are less data points (pixels) to be covered.
- The images are also normalized, which makes the variance in gray levels smaller.
- Of great importance is the fact that Kambhatla & Leen also aligned their images. They made sure that the eyes, nose and mouth of the faces in the database are aligned. This means that all eyes,noses and mouths are at the same place for each face image in the database. This aspect makes it much easier to find correlations between the face images in the database.
- In relation with the previous item, all faces are scaled.

- Kambhatla & Leen did use more images than we did.

Irrespective of the results obtained by Kambhatla and Leen, we have been successful in finding good dimension reduction methods. When comparing performance and time one can conclude that the VQE method is most appropriate for the face recognition application. The face recognition application is described in chapter 10.

Chapter 9

Morphing

Changing one image into another without shocks *(shock breaks)* is called a morph. A well known morph is the one used in the music video clip 'Black or White' by Michael Jackson 1986. He used morphing to let face images of black and white males and females change into each other. Such morphing can also be done with our five layer network. After learning we have a reduced representation of our images. From these vectors we can determine the two most separate vectors using Euclidean distance. If we follow the difference path between these two vectors we will end up in the other image. Because of generalization we expect that by transversing the path the network will produce useful new images. We set the number of morph steps between the two images at 100, thus the morph will consist of 100 frames. See Figure 9.1 for an example of a morph that was made with a database of 30 face images. When analyzing the morph instances, we found that the five layered neural network first produced images that had characteristics of the third image in Figure 8.1 and later the glasses from the sixth image in Figure 8.1 is used before ending up in the final morph image.

9.1 Conclusion

We have found a way for generating a morph with a neural network. A path not directly between the extremes, but transversing the intermediate images can improve, the quality of a morph. This can be expected because the neural network has learned all of the images, and the generated images which are between the intermediate images will not differ so much from the learned images.

Figure 9.1: An example of morph instances from the upper left to the lower right corner

Chapter 10

Face recognition

Using computers to recognize faces is of great importance nowadays. An example where face recognition can be applied sensibly is at a police station. There it is used to identify suspects using faces presant in a criminal database. Normally one would have to review all the photo books available on criminals which is an immense and time-consuming task. To speed up the search, use is made of a criminal photo database equipped with face recognition: numerous faces that are rather alike are present. Now one can concentrate on comparing the found face images with that of the suspected criminal. Our results show that the VQE method proves to be the most suitable method for fast dimension reduction. We shall now explain how we use the VQE (that uses local PCA (KLT)) to perform human face recognition. We apply the VQE as described in chapter 5. We use 30% of the number of images in the image database as the maximum number of Voronoi cells. We use 30% with face recognition instead of the 10% used in the dimension reduction in chapter 8.6, because a face database must be created before one can apply face recognition. The creation of a database is time consuming, and only a few images are likely to be added to the database. If we use only 10%, hardly any clustering will be accomplished. If a cell has 0 images, we retrain the VQE. After three retrain attempts we reduce the number of allowed cells to 20% of the database. As a result we will obtain Voronoi cells which contain images. Each cell has local images which are gathered in a local database. We now explain the use of a local database: We apply the PCA (KLT) to the set of face images in the local database, to obtain feature vectors. The number of used feature vectors in a cell is $\frac{3}{4}\times$ the number of images in that cell. A new face is propagated through the VQE network, and as result the Voronoi cell where the new face would be classified is found. In the found cell the new face is projected onto the local feature vectors, and the associated weights are compared with all the known faces. A match is found when the

Euclidean distance is smaller than a desired threshold. This results in the following scheme:

Initialize:

1 Acquire an initial set of face images, the training set

2 Calculate the eigenfaces/vectors of this training set, and use M' eigenvectors (see chapter 8.2) that account for the highest eigenvalues. These eigenvectors define the "input space" of the VQE method

3 Project every person in the training set into this "face space", and obtain every one's set of weights

4 Apply VQE and obtain Voronoi cells with reduced images.

Recognize:

5 Calculate a set of weights of the input image, by projecting this images into "input space"

6 Use the VQE network to classify the input image into a Voronoi cell

7 Check if the input image is close enough to the local Voronoi cells "face space", it may not be a face!

8 If it's a face, use the input image "face space" weights to classify it as 'known' or 'unknown', depending on the distance to all the learned images in the Voronoi cell

See Figure 10.1 for an overview of the scheme.

We have implemented this scheme in the 'NeuralCam' program. We used a real time camera stream to generate input images. These camera stream images are directly applied to a VQE and the input stream image is processed to match, if possible, within a certain Voronoi cell. A brief user manual about face recognition can be found in the appendix A

10.1 Conclusion

We have developed a face recognition program before called 'FaceCam'. This program uses the same camera stream as the 'NeuralCam' program does, but it uses the linear dimension reduction strategy PCA to recognize faces. If we compare the performance obtained with the non-linear dimension reduction strategy used in 'NeuralCam' and the linear strategy used in 'FaceCam', we

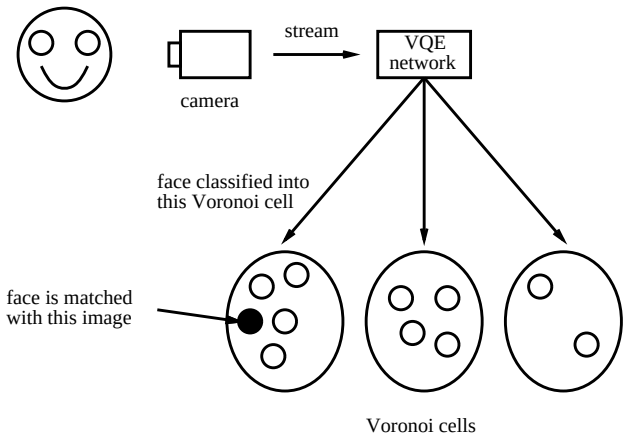

Figure 10.1: Overview of the face recognition system

observed that the non-linear approach in 'NeuralCam' showed a much better performance. When the Voronoi cells are analyzed, multiple face images of a person are often located within one Voronoi cell. One must be aware of the following items when applying face recognition:

- **Head size:** Because correlations between face images are being sought, it is important that the input stream face images are located whitin some aspect ratio of the face images in the database. This means that the distance between a face before the camera and the camera itself should be kept the same at all times. One can simply avoid this problem because one can take pictures of a face at various distances and store them in a database for proper retrieval

- **Light:** When the background light intensity changes, for example when a cloud gets before the sun, a shift in gray level intensities will occur. This shift has a huge impact on the face recognition task because the face recognition method tries to compare the camera stream faces with the face images stored in the database. Because the light intensity of the camera stream images is not within the same domain as the stored face images in the database no matches would be found. The stored database face images and the camera stream face images are not within the same region in space. A black background may be used to minimize the effect of background light variations and a room with special light conditions can also be used to improve performance.

- **Rotation:** The used method can recognize face images that are rotated a bit clockwise or anti clockwise. Much more problems occur when then input face is not taken fully faced forward. An example is a face image that is in a nodding down direction. To solve this problem, the use of another method called Fisher's [16] PCA is required. This method has not been implemented because our main objective was dimension reduction of images and not optimizing the face recognition application.

10.2 Future work

- Instead of applying PCA, use Fisher's PCA which does account better for nodded head positions.
- One can divide the face image into eyes, mouth and nose. Find best matches for all of the entities and classify on the gathered information.
- Let the recognition not depend on one camera image only, but use multiple different images from the camera stream for recognition. This prevents one from holding a single passport photograph in front of the camera.

Chapter 11

Gender recognition

One can use neural networks to recognize the gender of a person by his/her face image. Presenting the original (non reduced) images to a neural network is not recommended; this implies long training times because the order of input nodes of the neural network will be too high. To reduce training times, a smaller representation of the images is required. With dimension reduction, good compact representations of the images can be obtained. In our research, we used the five layered feedforward neural network to reduce each of the images to a vector of length five. The choice for using the five layered network, instead of one of the other methods mentioned before, is made because a five layered network was also used for this purpose by DeMers and Cottrell [13]. The reduced image representations are then used as input for a three layered feedforward gender network. The three layered gender network has only one output node which indicates true for a male and false for female. The gender network has only three layers (only linear dependencies are found) because the gender recognition task is not seen as difficult as reducing the size of images to only five values (non linear). We used a image database of 70 training images and 30 test images to train the five layered feedforward network. The 30 test images are used (as described in chapter 3.7) to obtain generalization. The reduced 70 training images are divided into 60 training images and 10 test images for the gender network. We used the same 30 test images as with the training of the five layered feedforward network to acquire generalization for the gender network. The 30 test images are also reduced to five values with the learned five layered network. In Figure 11.1 a overview of the gender recognition is shown.

One could think why divide the 70 images into a training- and a test-set for the gender network. Why not use two separate five layered networks to reduce to size of the training- and the test-images separately to five values followed by training the gender network with the training values and testing

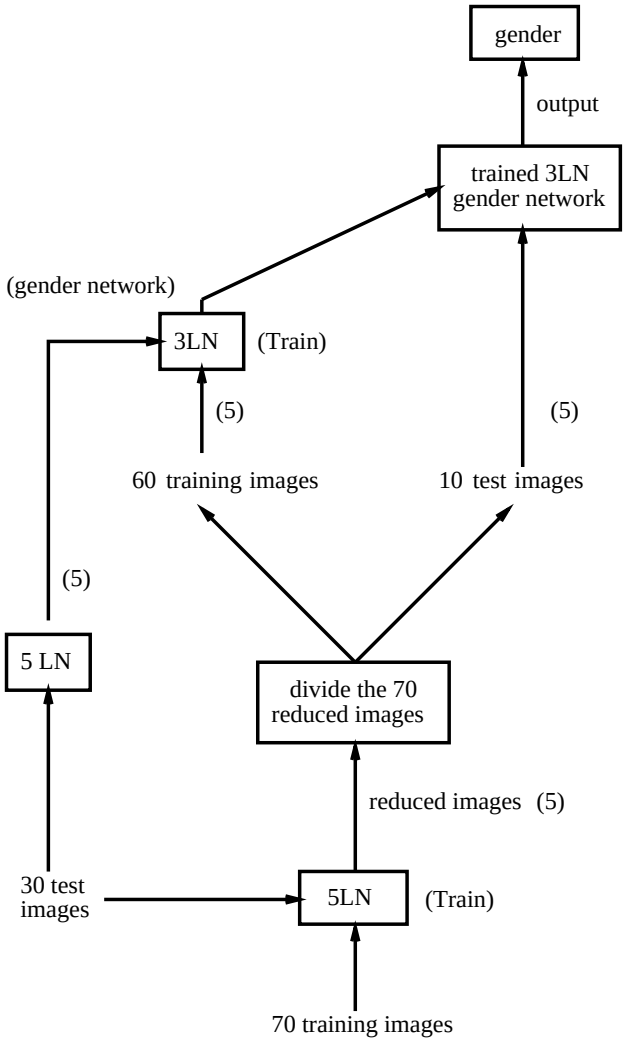

Figure 11.1: The gender recognition system

the gender network with the test values. Well, we tried this and we discovered that the input dimension of the learned gender network was not the same as the dimension of the test values. This is because the training- and test-images were reduced to five values with two different unrelated five layered networks. If a set of images is reduced to five values, the next time the same set of images is reduced to five values, the values may not have the same domain. This is because the weights on the edges of a neural network are initialized with random values. When we divide the learned 70 images, we are sure that all reduced images are within a domain. We have tested the

gender recognition with the mentioned image database which consisted of 60 training images and 10 test images. We found that all of the training images were classified correctly as male or female. The test images caused the gender network to classify 1 image incorrectly which is 10%. Full details on the training settings can be found in Table 11.1.

Three layer gender settings							
Test domain	Max #Epochs	Begin η	End η	Adaptive parameter a	Adaptive parameter β	Adaptive parameter K steps	Momentum term
6	50000	0.04	0.02	0.0001	0.25	8	0.9

Table 11.1: Settings used to learn a three layered gender neural network

11.1 Conclusion

From the results, one can conclude that it is possible to do gender recognition with three layered feedforward neural networks. No addition of hidden layers is required which indicates that the gender recognition task is linear.

Chapter 12

Conclusions and future research

We have researched the use of linear and non linear dimension reduction methods on images. Our results show that the images can be reduced better with the non linear methods. Even the five layered non linear neural net has difficulties with the learning of the image function. From the small learning-rate settings, we can conclude that the image function is very steep which makes it hard to be learned. Clustering of the images proved promising because little time is required and good results are obtained. Because good results were achieved with face images which are known from literature not to be easily reduced, the used non linear methods VQE and VQG are likely to perform well in other areas. The implemented applications show that reducing the data makes the data easier to handle and that good data representations can be used for various applications.

At the end of this thesis we propose some suggestions for future work:

- Research on the dimension reduction of speech and sound. How well would the used non linear methods perform on sound waves. One can also try to perform speech recognition, and even better, combine image and speech to identify a person.
- Instead of preprocessing (chapter 8.2) the images with PCA, use Fisher's [16] PCA to reduce the images. This could lead to even better results because more rotations are taken into account.
- Use a Sanger [10] network for PCA. This network can use the stored weights when new patterns are to be learned. This can be an advantage to the PCA used in this research, which calculates everything again [1].

[1] We investigated the use of a Sanger's network, but no negligible time improvement was achieved. This could be because our face image database was limited in size.

Appendix A

NeuralCam user manual

A.1 Introduction

The 'NeuralCam' program has many features which we shall discuss briefly.

A.2 Menubar

The menubar is where one can select options. In the following sections all of the menubar options are briefly described.

A.2.1 Options

The **Options** menu contains the options Recognition, Reduction and Quit.

- **Recognition:** This enables the face recognition feature. All previously loaded databases are cleared. The camera window (top left-hand square) will display the current camera stream. See section A.3 for more on face recognition.
- **Reduction:** This option sets the program in the reduction mode. This means that the bottom row of rectangles is activated. In these rectangles the results of the various reduction methods will appear. See section A.4 on how to manage reduction.
- **Quit:** This option allows one to exit the 'NeuralCam' program.

A.2.2 Detection

The Fixed and Motion option are located in the **Detection** menu. These options are only available when the recognition mode is active.

- **Fixed:** This holds the white rectangle in the camera window at a fixed position.
- **Motion:** The white rectangle in the camera window tries to follow any objects in motion. This method uses a number of camera stream frames to locate an object (face).

A.2.3 Database

The options 'Load Database', 'Save Database', 'Clear Database' and 'Add Rotations' are all in the **Database** menu. This menu can be accessed when the reduction mode or the recognition mode is active.

- **Load Database:** This menu option allows the user to load a database (*.db) file. A database file exists of face image data and of the matching names of the faces. When a database is successfully loaded, the menu options **Reduction** and **Settings** and **Neural** are accessible.
- **Save Database:** One can save a database in order to store the pictures that are taken with the camera. The extension of the database filename to save should be .db.
- **Clear Database:** This option clears the listbox which holds the names of the faces and the existing loaded database is wiped. A new database should be created or loaded if one wants to continue.
- **Add Rotations:** This option changes the mode of the **Add Rotations** toggle button. When in toggled mode, a new image which is added to the database is automatically rotated five degrees to the left as well as five degrees to the right.

A.2.4 Reduction

The options PCA, VQE, VQG and 5LN are located the **Reduction** menu. These menu options can be reached when the reduction mode is active and when a database has been loaded.

- **PCA:** This option sets the reduction method to PCA. All reduction results will appear in the PCA labeled window, which is located in the bottom area of the 'NeuralCam' main window.
- **VQE:** Like above, but now reduction is done with VQE.
- **VQG:** The VQG reduction method shall be applied.

- **5LN:** Selecting this option will make the **Neural** and **Settings** menu options in the menubar accessible.

A.2.5 Settings

The **Settings** menu has the following menu items: Settings, Gender and 'Apply gender'. These items can only be selected when a database has been loaded and the reduction method is **5LN**.

- **Settings:** When selecting this item, a pop-up window will appear. This pop-up window displays the settings of the five layered neural net. One can change a value by moving a slider to the left or the right. With the slider '#units hidden 2' the number of units in the representation layer can be adjusted; normally this value should be kept as 5. At the right bottom of this window, the following buttons are to be found:

  - **Load TestSet**: This button allows to load a test database, which is used for generalization when training a five layered network.
  - **Train Network**: This option allows starting the training of the five layered network. The settings from the settings pop-up box are used. A testset database must be loaded before the training can start.
  - **Load Weights for Network**: This option is used to load a weights file (*.dat). With the weights, a neural network is created and the reduction with **5LN** is used for reducing images. It is important to load a weights file which was created when the database was used.
  - **Show Morph**: This will show a morph player which can play the morph at different frame-rates (frames/sec). The weights of the five layered net must be loaded before the morph can be made and shown.

- **Gender:** This option pops-up a box that looks like the pop-up box used in **Settings**. With the sliders in this box, the learning settings can be adjusted which are used to train the three layered gender network. Located in the right bottom corner, various buttons are to be found:

  - **Load TestSet**: This button allows loading a test database, which is used for generalization when training a three layered gender network.

  - **Train gender Network**: Train the gender network. The user will be asked to divide the training-set used with 5LN into a training-set and a test-set for the three layered gender network. The gender of all images will be asked before training can start.
  - **Load Weights for Training Network**: This option allows the user to load a weights file which was created after the training of the 5LN was finished. The weights from this file are then used to build a five layered network, which is used to reduce all of the gender images to five values.

- **Apply Gender:** This is an option which can be toggled on or off. When the **Apply Gender** is on, the reduction of an image is done and the same image is applied to the gender network. The results of gender recognition are shown in the gender window which is located at the right side of the 'NeuralCam' main window. The **#TEST** and **#TRAINING** labels indicate how many training-images and test-images are used with gender recognition. The **TestWC** and **TrainWC** indicate how many test-images and training-images are classified incorrectly. The test-images are not used when training the three layered gender network.

A.2.6 Help

Only the About menu item can be found in the **Help** menu.

- **About:** Some information about the author is shown.

Figure A.1 displays the NeuralCam program.

A.3 Recognition

If the **Recognition** option is active, the camera will display images in the camera window, which is located at the top left corner of the 'NeuralCam' main window. One can load a previously created database for recognition usage, but adding images with the snapshot button (which is located just beneath the camera window) to a new database is also possible. The latter is recommended, because the difference in light intensity between the camera stream images and the database images will be smaller. One can use the two sliders next to the camera window to change the distances between the images in the database and the camera stream images. The **Space Threshold** indicates how far the current camera stream image is from the space described by the faces in the database. When this threshold is lowered, only

Figure A.1: The NeuralCam program

camera stream images that are face-like, are allowed. The **Class Threshold** indicates how far the camera stream image is from the face images in the database. If this threshold is to high, faces can be classified incorrectly. The **Class Threshold** will only be operative when the **Space Threshold** has been satisfied.

A.4 Reduction

When a database is loaded and the **Reduction** option is active one can perform dimension reduction to the fifth dimension on the face images in the database. The listbox containing all the face names can be used to reduce a face image. Double click the left mouse button on the selected name and the corresponding face image will be reduced.

Appendix B

NeuralCam source files

B.1 Introduction

In this appendix, the NeuralCam source files and other related files will be described. The NeuralCam program is developed in a UNIX environment, using the programming languages C and Motif. The Motif language is used for building an X-windows user interface.

B.2 Database files

The face images and associated information are stored in database files. The database files have the extension *.db. The following database files are used:

- Test_*_*.db
- Train_*_*.db.

The Test_*_*.db holds the test images, whereas the Train_*_*.db holds the training images. A test database is only used during the training of a neural network. The database files have the following internal structure:

```
string        /* database id */
int           /* width of images */
int           /* height of images */
int           /* number of persons */
int           /* total number of images */
string        /* name person 1 */
int           /* number of images of person 1 */
float         /* gender of person 1 */
int           /* image 1 used in training or in testing */
```

```
string        /* name person 2 */
int           /* number of images of person 2 */
float         /* gender of person 2 */
int           /* image  2 used in training or in testing */
.
.
string        /* name person n */
int           /* number of images of person n */
float         /* gender of person n */
int           /* image  n used in training or in testing */
unsigned char /* image data */
```

B.3 Gender files

The files **male_100x100.pgm**, **fem_100x100.pgm** and **malefem_100x100.pgm** contain images of the various genders. The dimensions of the images are 100×100. The images are shown in Figure B.1

Figure B.1: Gender pictures (Female, Male and Unknown)

B.4 C source files

In this section the NeuralCam source (*.c) and header files (*.h) are briefly described. The following files are implemented:

- **5ln.c 5ln.h**: This module handles the five layered neural networks. The neural networks are allocated dynamically, and when the networks have become redundant, the networks are deallocated *(destroyed)* from memory. This module also contains a function that scales the PCA reduced images into the neural network transfer function domain ([-0.85...0.85]).

- **camera.c camera.h**: The Silicon Graphics Indy camera device is controlled with this module. The camera uses interrupts to control the stream of camera images. The NeuralCam program only requires grayscaled images, but the module can also generate 256 colour images.

- **database.c database.h**: The database images from the *.db files are handled by this module. Because of the complexity of the data, a separate appendix C is dedicated to the database structures.

- **fileio.c fileio.h**: This module is the interface between the database files and the database structures. The module loads the data from the database files into the database structures in memory. The gradient images are created when a database is loaded into memory.

- **gradient.c gradient.h**: The gradient reduction method is handled in this module. The gradient images in memory are pre-processed with PCA and then concatenated with the pre-processed original image. This new vector can then be applied to the vector quantization network. The network for gradient reduction is described in module **vqg.c**.

- **pca.c pca.h**: The PCA reduction is described in this module. It is a module that uses the Jacobi method in **maths.c** to calculate the eigenvectors and eigenvalues.

- **vq.c vq.h**: The vector quantization network with Euclidean distance is handled in this module. The network is allocated dynamically, when the network becomes redundant, it is deallocated. The network is only in memory when needed.

- **vqg.c vqg.h**: Same as the module **vq.c** but it uses the gradient information to build the unsupervised vector quantization network.

- **timing.c timing.h**: This module is used to time the duration of the various methods.

- **maths.c maths.h**: This module handles various mathematical functions. The function Jacobi is optimized, it is forced to use registers. There is also a sorting algorithm, which is used to order the eigenvalues.

- **neuralcam.c**: This is the **main** NeuralCam module. It is where the execution of the NeuralCam program begins.

- **bitmaps.h**: The bitmaps (pictures) for the various buttons are stored in this header file.

- **help.h**: The 'about' text that is displayed when the user activates the about button in the help menu.
- **motifui.c ui.h**: This module creates the Motif based user interface. It is the largest module, because all interaction is handled by this module. The module uses the **motif.c** library file to create the various widgets.
- **motif.c motif.h**: This module was initially created in 1993, but has been updated and maintained in 1995. In 1997 it was extended for the NeuralCam program. The module is a library of various Motif based functions that can be used to create an user interface.
- **draw.c draw.h**: This is also a Motif library. It handles all drawing events. It was designed for the Silicon Graphics Indy workstations that use the Irix 5.3 platform. It can easily updated for a Hewlett Packard machine which uses a different colour table.
- **Makefile**: This file contains the appropriate compiler options, which makes it easy to build the NeuralCam program.

Appendix C

Data structures

C.1 Database structures

Struct of stored images

```
typedef struct  {
  int   rows;   /* image dimensions */
  int   cols;
  float *data;  /* image data */
} float_image;
```

Struct of eigenface data

```
typedef struct  {
  int         imagenum;   /* image index number */
  float_image image;      /* image */
  float       eigenvalue; /* image eigenvalue */
} ef_image;
```

Struct to store eigenface images

```
typedef struct {
  ef_image    *images;      /* image  */
  float       *eigenvalues; /* image eigenvalues */
} Eigenfaces;
```

```
Struct for training set images

typedef struct  {
  int         imagenum; /* image index number */
  float_image image;    /* original image */
  float_image diff;     /* difference image */
  float_image gradient; /* gradient image of image */
} ts_image ;


Struct that holds person data

typedef struct {
  char  name[MAXNAME];          /* person's name */
  int   nimages;                /* #images of person */
  int   *images;                /* id's of person's images */
  float *projection;            /* person's projection  */
  float *scaled_projection;     /* scaled weights for net */
  float *VQEinverse_projection; /* obtained with VQE */
  float *PCAinverse_projection; /* used with PCA */
  float *VQGinverse_projection; /* used with VQG */
  float *FIVELNinverse_projection;/* used with 5LN */
  float gender;                 /* 0 ? .85 male -.85 female */
  int   train_or_test;          /* 1 gender train 0 test */
} person_data;


Struct that holds the trainingset

typedef struct {
  int         dimensions[2]; /* width,height of faces */
  int         npersons;      /* # persons in TS */
  int         nimages;       /* # images in TS */
  int         nefs;          /* # eigenfaces to use */
  person_data *persons;      /* data per person */
  float_image mean;          /* average face image */
  float       mean_dot_mean; /* dot(mean,mean) */
  ts_image    *images;       /* detected faces */
  Eigenfaces  efs;           /* related eigenfaces */
```

```
} TrainingSet;
```

C.2 Neural structures

The settings structure

```
typedef struct {
  int    InputSize;          /* # input nodes */
  long   MaxEpochs;          /* max #epochs */
  double BeginLearningRate; /* begin eta */
  double EndLearningRate;   /* end eta */
  double Momentum;           /* momentum parameter */
  double AdaptiveA;          /* adaptive parameter a */
  double AdaptiveB;          /* adaptive parameter b */
  int    AdaptiveK;          /* adaptive parameter k */
  int    UnitsHidden1;       /* # nodes hidden 1 */
  int    UnitsHidden2;       /* # nodes representation */
  int    UnitsHidden3;       /* # nodes hidden 3*/
} SettingType;
```

Bibliography

[1] N. Kambhatla and T.K. Leen, *Fast Non-Linear Dimension Reduction*, Advances in Neural Information Processing Systems 6, 1994, OR 97291-1000

[2] M. Turk and A. Pentland, *Eigenfaces for Recognition*, Journal of Cognitive Neuroscience Volume 3 ,No. 1, pp. 71-86, 1991

[3] A. Pentland, T. Starner, N. Etcoff, A. Masoiu, O. Oliyide and M. Turk, *Experiments with Eigenfaces*, M.I.T. Media Laboratory Perceptual Computing Technical Note No. 194, August 1992

[4] A. Pentland, R. W. Picard and S. Sclaroff, *Photobook: Content-Based Manipulation of Image Databases*, M.I.T. Media Laboratory Perceptual Computing Technical Report No. 255, Nov. 1993

[5] A. Pentland, B. Moghaddam, T. Starner, *View-Based and Modular Eigenspaces for Face Recognition*, IEEE Conf. Computer Vision and Pattern Recognition, pp. 84-90, Seattle, WA, June 1994

[6] B. Moghaddam, A. Pentland, *Probabilistic Visual Learning for Object Detection*, The 5th International Conf. on Computer Vision, Cambridge, MA, June 1995

[7] M. Lew, *Transform Theory and Application*, Beeldreeksverwerking Seminar sheets, pp. 4-5, 1995

[8] H. Anton, C. Rorres *Elementary Linear Algebra*, Applications version, sixth edition, Wiley & Sons

[9] R.C Gonzalez, R.E Woods, *Digital Image Processing*, September 1993 pp. 148-156, Addison Wesley

[10] J. Hertz, A. Krogh, R.G. Palmer *Introduction to the theory of neural computation*, November 1995 eleventh printing, Addison Wesley

[11] G.W. Cortrell, P. Munro, D. Zipser *Image compression by back-propagation: an example of extensional programming*, Advances in Cognitive Science, Volume 3, 1988

[12] P. Baldi, K. Hornik *Neural Networks and principal component analysis: learning from examples without local minima*, Neural Networks 2:53-58, 1989

[13] D. DeMers, G. Cottrell *Non-Linear Dimensionality Reduction*, Dept. of Computer Science & Engr., 0114 Institute for Neural Computation, 92093-0114 1993

[14] J.A. Spierenburg, D.P. Huijsmans *VOICI: Video Overview for Image Cluster Indexing a swift browsing tool for large digital image database using similarities*, 1997, Computer Science Department, Leiden University

[15] R. Egas *Benchmarking of Visual Query Algorithms*, Master Thesis, 1997, Leiden University Department of Computer Science & Philips Research

[16] M.S. Lew, D.P. Huijsmans, D. Denteneer *Content Based Image Retrieval: KLT, Projections, or Templates*, Proceedings of the First International Workshop, IDB-MMS August 1996, pp. 27-34