Algorithms for designing filters for optical pattern recognition

Epaminondas Stamos

A thesis submitted for the degree of
Doctor of Philosophy
of the
University of London.

Department of Electronic & Electrical Engineering
University College London

January 17, 2001

Acknowledgments

My PhD was a long engagement which lasted several years. During this period, many people have helped me in one way or another. I feel very grateful towards three people in particular, without whose help and support it would have been impossible for me to do this PhD. My parents, Giannis Stamos and Maria Stamou and my supervisor David R. Selviah. I feel gratitude towards my parents for the psychological support they gave me during the project and for their crucial financial support, since this PhD was not funded in any other way. And I am very grateful to my supervisor for his continuous support and guidance, for his relentless effort to teach me as many things as possible and for his very long patience to explain to me everything that was necessary for my progress and to endure my mistakes and inefficiencies. I thank them all very much.

During my time here in UCL I have been a member of the Optical Systems and Devices group. All of the other members of the group have been very helpful. I particularly wish to thank Dr Lawrence Commander for spending long hours from his own PhD to be my advisor and my "assistant". Many thanks to Robin Kilpatrick, Mark Gardner, Keith Forward and to the newer members of the group, Sue Blakeney and Hui-Fang Deng, for sharing their computer programs and their knowledge with me and for the interesting discussions we have had. I would also like to thank Dr Sally Day and Dr Aníbal Fernández for their valuable contributions and comments during the group meetings we have held. Laki Panteli for his help and support in the first years of my stay in England. I am also grateful to Dr Tim York for his valuable advice during the first years of my PhD.

Last but not least I want to express my appreciation to my sister Vivian who has kept me company and helped me in so many ways during the last four years.

Abstract

Matched filters for optical correlators detect the presence of objects immersed in white noise, but are unable to discriminate between similar, noisy input patterns. Also, the dynamic range of optical systems often limits the size of the images that can be recognised. We develop two algorithms for designing filters for optical pattern recognition. The first algorithm suppresses the similarities between the training images and creates a set of filters, which are mutually orthogonal to them. Our filters tolerate 7 dB more additive input white noise than matched filters and the required dynamic range is reduced by 25 dB. In addition, the filters obtained after only two iterations tolerate 2 dB more additive input white noise than linear combination filters (LCF), which results in an improvement in the probability of discrimination of about 30% for the same amount of noise. The correlation outer products for the 2 iteration Similarity Suppression (SS) algorithm are substantially lower than those for the LCFs. The second, Feature Enhancement and Similarity Suppression (FESS), algorithm designs filters for multi-class pattern recognition. Each of these filters can recognise all the members of a group and distinguish them from other groups. The probability of recognition for a training set of faces is 100% without noise, compared to 90% using matched filters and the required dynamic range is again reduced by 25 dB. We prove the mathematical equivalence between these algorithms, the back-propagation algorithm for training neural networks and the method for designing general synthetic discriminant functions (SDF). Our algorithms also design filters for two or more cascaded banks of correlators and can train multilayer neural networks. Conversely, matrix inversion methods, which are generally used for designing SDFs, can train neural networks and give the same results as obtained with the back-error propagation algorithm.

Contents

List of Figures

List of Tables

Notation

β	: Convergence parameter
η	: Convergence parameter
η_H	: Horner efficiency
θ	: Neuron threshold
λ	: Positive real number
ξ	: Uniform background image
Σ	: Noise covariance matrix
σ^2	: Variance
$\phi(\cdot)$	: Neuron activation function

$\mathbf{a}$	: Solution vector for synthetic discriminant functions
a	: Coefficient for calculating equal correlation peak filters
b	: Coefficient for calculating equal correlation peak filters
$\mathbf{C}$	: $M \times M$ matrix whose elements are the coefficients C
C	: Coefficient for calculating linear combination filters
c	: Coefficient for calculating equal correlation peak filters
d	: Desired neuron or correlator output value
D	: How often (iterations) the addition equation of the FESS algorithm is applied
$\mathbf{D}$	: $N \times N$ diagonal matrix
$\mathbf{d}$	: Vector with desired correlation outputs
$E\{\cdot\}$	: Expected value
E_{ave}	: Average correlation plane energy
e	: Neuron output error
F	: Correlation response
$\mathbf{f}$	: Input image
$\mathbf{G}$	: $M \times 1$ vector whose elements are the filters $\mathbf{g}$
$\mathbf{g}$	: Correlation filter

$\tilde{\mathbf{g}}$	: Unnormalised correlation filter
${}^{(j)}\mathbf{g}$	: $\mathbf{g}$ filter in the j_{th} bank of cascaded banks of correlators
$\mathbf{g}^{(i)}$	: $\mathbf{g}$ filter at the i_{th} application of the SS or FESS algorithm
K	: Number of classes
L_j	: Number of training patterns in the j_{th} class
M	: Total number of patterns in the training set
N	: Size (in pixels) of training patterns and filters
$\mathbf{n}$	: Noise vector
O	: Total number of neurons in the network
P	: Power of normalised training patterns
$\mathbf{R}$	: Vector inner product matrix
r	: Value of inner product of two patterns
$\mathbf{S}$	: Matrix whose lines are the training vectors $\mathbf{s}$
$\mathbf{s}$	: Training pattern
$\bar{\mathbf{s}}$	: Mean of training patterns $\mathbf{s}$
$\mathbf{T}$	: $N \times N$ diagonal matrix
t	: Neuron output target value
$\mathbf{u}$	: Pattern orthogonal to the other patterns in its set
$\tilde{\mathbf{u}}$	: Unnormalised orthogonal pattern
$\mathbf{V}$	: Set of N real numbers
v	: Neuron net internal activity level
w	: Neuron weight
x	: Neuron input
y	: Neuron output
z	: Complex number of modulus 1
$\mathbf{z}$	: Vector of real numbers
$\star$	: Symbol denoting convolution
$\otimes$	: Symbol denoting correlation
$\cdot$	: Symbol denoting inner product

Abbreviations

BPOF	:	Binary Phase Only Filter
BPO SDF	:	Binary Phase Only Synthetic Discriminant Function
CCD	:	Charge Coupled Device
ECP	:	Equal Correlation Peak
FESS	:	Feature Enhancement and Similarity Suppression
fSDF	:	filter Synthetic Discriminant Function
FT	:	Fourier Transform
GMF	:	Generalised Matched Filter
JTC	:	Joint Transform Correlator
LCF	:	Linear Combination Filter
LDF	:	Linear Discriminant Function
MACE	:	Minimum Average Correlation Energy
MHP	:	Modified Hyper Plane
MICE	:	Minimum Correlation Energy
MINACE	:	Minimum Noise And Correlation Energy
MLP	:	Multi Layer Perceptron
MOF	:	Mutually Orthogonal Filter
MOSLM	:	Magneto Optic Spatial Light Modulator
MVSDF	:	Minimum Variance Synthetic Discriminant Function
OCC	:	Optimal Characteristic Curve
OTF	:	Optimal Trade-off Filter
PCE	:	Peak to Correlation Energy
POE	:	Probability Of Error
POF	:	Phase Only Filter
PO SDF	:	Phase Only Synthetic Discriminant Function
SDF	:	Synthetic Discriminant Function
SLM	:	Spatial Light Modulator
SMACE	:	Space domain MACE

SNR	: Signal to Noise Ratio
SS	: Similarity Suppression

Chapter 1

Introduction

In this thesis we develop two algorithms for designing filters for optical pattern recognition. We investigate their performance using theoretical calculations and computer simulations. In addition, we compare our algorithms to relevant existing techniques for designing filters for optical pattern recognition and to neural network training algorithms.

The original aim of this project was to develop an optically implementable algorithm for training neural networks. This was based on an initial version of one of our algorithms for designing filters for optical pattern recognition, which was based on the Gram-Schmidt orthogonalisation procedure, and on the already known relationship between neural networks and optical correlators [4]. Our initial aims were to further demonstrate, develop and improve our algorithm and to assess its limitations. To investigate whether it could be used to train neural networks and its relationship to other neural network training algorithms. And to design and build an optical system which would implement our training algorithm.

Various reasons, most important among which being the interesting results we obtained from our computer simulations and the theoretical comparisons with other training algorithms and filter design techniques, led us to emphasise the theoretical part of the project. In addition, we focused on the optical filter design side of the project and not on the neural network side, because of the currently higher interest in optical filters rather than optical neural networks. In the following paragraphs we present some background information on the relevant fields, namely optical pattern recognition and neural networks, which will help us place our work in the context of related research.

Optical pattern recognition has been a vibrant field of research over the last forty years [5]. Correlation [6, 7], a very well known mathematical method for comparison, is very often used for optical pattern recognition. Optical correlators have the advantage that they are very fast compared to computers. This speed advantage is a consequence of the inherent parallelism of optics and the speed of light. Moreover, there are optical correlators, which can correlate many images with one in parallel [8]. The most important component of any recognition system based on a correlation is the template, or filter, with which the input pattern is compared. This filter depends on the correlator implementation (optical, electronic, hybrid etc.). Some optical correlators use filters in the space domain [9, 10] and others in the Fourier domain [5]. Furthermore, the filters depend on the particular task at hand. Some systems' primary aim is to detect the presence of an object in a noisy background. For example, matched filters, which are the complex conjugates of the spectrum of the original patterns, are optimal for detecting signals in white, Gaussian noise [11]. The aim of other systems is to recognize the presence of any one of several patterns in the input, for example, SDF filters [12, 13, 14, 2] and optimal trade-off filters [15, 16, 17]. Other systems aim to distinguish between very similar objects, for example, mutually orthogonal filters [18]. All of these are not completely different tasks, on the contrary, they are inter-related and many filters are designed with all of these aims in mind. Most of the previously mentioned filters are linear combinations of training patterns and their design methods are based on solving a set of simultaneous equations, to calculate an array of coefficients. These coefficients can then be used to linearly combine the training patterns, to create the filters in such a way that their correlations with the input patterns yield the desired output values.

Neural networks [3] are simplified models of the human brain. They consist of many simple processing units called *neurons*. These neurons are interconnected with connections of different strengths. The strengths of these interconnections are called *weights* and determine the behaviour of the network. The methods for modifying these weights are called training algorithms. Most of them are iterative and they apply a mathematical rule to modify the network's weights, usually based on a number of training examples. Sometimes, these mathematical rules are rather complicated. In addition, many iterations and a large number of training examples may be necessary for the network to yield the desired outputs. Therefore, the training of a neural network is often a time consuming process. Furthermore, as the desired network behaviour may change with time, the network may need to be retrained.

These disadvantages of the neural network training process and the known structural equivalence between optical correlators and single layer neural networks, were what initially motivated us to start this project. An algorithm which could be implemented optically and could be used to train neural networks would use the advantage of the speed and the parallelism of optics to speed up the process of training neural networks. During the project, we decided not to work on the practical optical system to implement our training algorithm, but instead to concentrate on the algorithm development. Also we shifted the emphasis from neural networks to optical filters and we investigated the relationship between neural network training algorithms and optical filter design techniques using our algorithms as an intermediate step for the comparisons, which led to some very interesting results. So our revised, final aims are summarised below:

1. To further demonstrate, develop and improve our algorithm.
2. To assess its limitations.
3. To develop, demonstrate, and assess the limitations of a second algorithm which addresses the problem of multi-class pattern recognition.
4. To investigate the relationship between our algorithms and some neural network training algorithms.
5. To compare our algorithms to some relevant optical filter design techniques.

The layout of this thesis is as follows: We start with some background theory and a review of some of the relevant research, in the next three chapters. Then we present our work and we finish the thesis with our conclusions. Specifically, the next chapter contains some introductory theory on optical correlators. We briefly describe the 4-f correlator and the joint transform correlator. This chapter is useful for the reader who has no prior knowledge of optics, and particularly correlators. A reader who is already familiar with these can proceed straight to the next chapter. Chapter 3 presents some of the most relevant methods for designing filters for optical pattern recognition. In chapter 4 we present some introductory theory on single layer and multilayer perceptrons. This is a theory chapter, aimed at the reader who has no neural network knowledge and can be omitted by a reader who is already familiar with them. The next four chapters present our work. Chapter 5 contains the derivation and theoretical analysis of our first algorithm, called the *similarity suppression* (SS) algorithm. In addition in chapter 5 we theoretically compare the similarity suppression algorithm with relevant filter design techniques and the Hebbian learning law for training neural networks. In chapter 6 we present the computer simulations of the SS algorithm. Each section of that chapter presents the simulations that prove, or investigate the accuracy of the theory that was presented in the corresponding section of chapter 5. Our second algorithm, called the *feature enhancement and similarity suppression* (FESS) algorithm, along with its theoretical analysis and comparisons with relevant filter design methods, is presented in chapter 7. The layout of chapter 7 is very similar to the layout of chapter 5, so that the reader can make comparisons between the two algorithms. Chapter 8 follows, with the computer simulations for the FESS algorithm. An overview of our work, along with a list of our main achievements and new ideas, and some proposals for further work are presented in chapter 9. Appendix A presents some mathematical preliminaries that are useful for the reading of the thesis. Appendix B presents the mathematical analysis of the changes that occur to the filters' magnitudes during the training with the SS and the FESS algorithms. Appendix C contains all of the training images used for the FESS algorithm. Finally, appendix D contains the graphs of the simulations of the FESS algorithm, which were not included in chapter 8.

Chapter 2

Optical inner product correlator fundamentals

2.1 Introduction

Optical inner product correlation is the tool on which optical pattern recognition is based. So, in this chapter we present some elementary background theory on optical correlation, starting with the optical Fourier transform and ending with some implementation considerations and some performance measures. In the second section we present the Fourier transform property of a lens. In section 2.3 we present some of the most common optical correlators. Section 2.4 describes the optical matched filters which were first used for optical pattern recognition. Section 2.5 presents some principles and important issues concerning implementations of optical correlators. Finally, section 2.6 presents some frequently used performance measures for the evaluation of optical pattern recognition filters. The theory presented in this chapter is aimed at the reader with no prior knowledge of correlators and is written with the purpose of familiarising him/her with some correlator fundamentals, which are necessary for the understanding of the main work of this thesis. If the reader is already familiar with optical correlators and filter performance measures, he/she can proceed to the next chapter, which presents some of the filters that have been designed for use with these optical correlators.

2.2 Optical Fourier transform

According to the *Fraunhofer approximation* [19], when an aperture is illuminated with coherent light the far-field diffraction pattern is the Fourier transform of the complex aperture distribution, as shown in equation 2.1

$$E(x,y) = \frac{e^{jkz}}{j\lambda z} e^{\frac{jk(x^2+y^2)}{2z}} \iint\limits_{-\infty}^{+\infty} E(\xi,\eta) e^{-j\frac{2\pi}{\lambda z}(x\xi+y\eta)}\, d\xi\, d\eta \tag{2.1}$$

where, λ is the wavelength of the light, z is the distance from the aperture and $k = \frac{2\pi}{\lambda}$ is the propagation number, the magnitude of the propagation vector, $\mathbf{k}$. If a lens is inserted immediately after the diffracting aperture, then it focuses the far-field pattern onto the focal plane. The amplitude distribution at the focal plane of the lens is

$$E(x,y) = \frac{1}{j\lambda f} e^{\frac{jk(x^2+y^2)}{2f}} \iint\limits_{-\infty}^{+\infty} E(\xi,\eta) e^{-j\frac{2\pi}{\lambda f}(x\xi+y\eta)}\, d\xi\, d\eta \tag{2.2}$$

where the constant phase factor e^{jkz} is ignored and f denotes the focal length of the lens. This equation is almost identical to the 2-dimensional Fourier transform equation shown in equation 2.3. The only difference between the two equations is the quadratic phase factor term $e^{\frac{jk(x^2+y^2)}{2f}}$.

$$F(x,y) = \int_x \int_y f(\xi,\eta) e^{-j(k_x\xi+k_y\eta)}\, d\xi\, d\eta \tag{2.3}$$

It has been shown [19] that when the diffracting aperture is located at the front focal plane of the lens, then this quadratic phase factor is removed and an exact Fourier transform relationship exists between the front and back focal planes. As far as the inverse Fourier transform is concerned, which is shown in equation 2.4,

$$f(x,y) = \int_x \int_y F(\xi,\eta) e^{j(k_x\xi+k_y\eta)}\, d\xi\, d\eta \tag{2.4}$$

this can be obtained by performing a forward Fourier transform optically and then calculating the mirror image along the x and the y axes of the output.

2.3 Optical Correlators

By making use of the convolution theorem[1], it is possible to compute the convolution of two functions much faster by performing two FFTs, one inverse FFT

[1]See appendix A

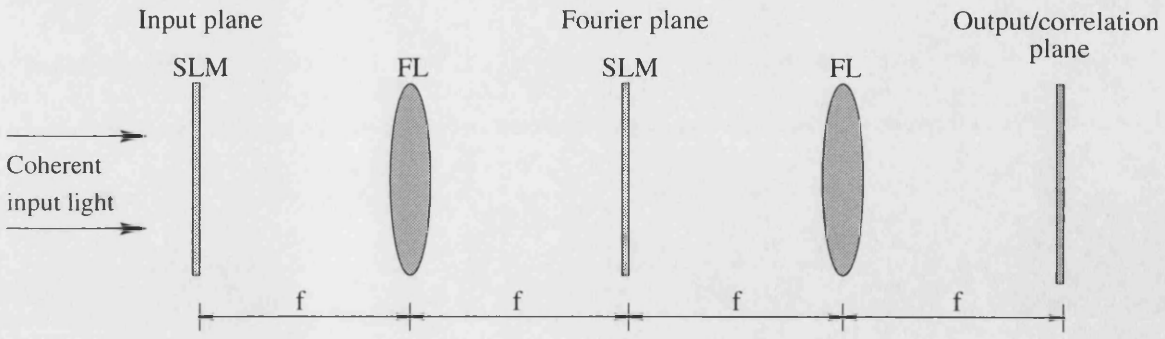


Figure 2.1: 4-f correlator after Collings 1987 [1]

and N multiplications instead of $2N^2$ multiplications that would be necessary for the convolution [20]. Even then, however, the convolution or correlation of two functions can be a time consuming calculation, and dedicated chips have been manufactured to perform them [21]. Optical convolution or correlation is very fast, because an aperture can perform a Fourier transform with a lens bringing it into the near field and giving it the correct phase. Also the multiplication is very easily implemented optically [22], by, for example, illuminating a sandwich of the two images. The speed of the optical implementation of the correlation has led to the design of several kinds of optical correlators.

2.3.1 4-f Correlator

A very simple optical correlator is shown in figure 2.1. It is called the *4-f correlator* or the *frequency plane correlator.* The first lens is performing the Fourier transform of the input function, $i(x, y)$, which is displayed on the first spatial light modulator (SLM) and is illuminated with coherent light. The complex conjugate, $F^*(u, v)$, of the Fourier transform of the filter function, $f(x, y)$, is displayed at the back focal plane of the lens using the second SLM. The two Fourier transforms are multiplied, the light leaving the second SLM is the product $I(u, v)F^*(u, v)$, and at the back focal plane of the second lens, which is performing another Fourier transform, the output is equal to the cross-correlation of the two functions in the space domain [20]. The 4-f correlator uses Fourier domain matched filtering because the Fourier transform of the filter must be displayed on the second SLM. Obviously the Fourier transform of any filter is a complex function. The photographic film that was initially used for the implementation of the 4-f correlator was not orig-

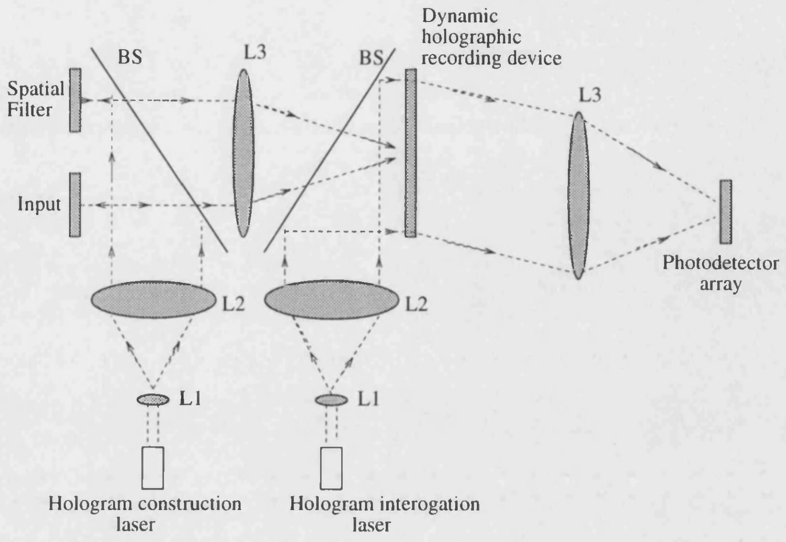

Dynamic holographic recording device
BS
L3
BS
Spatial Filter
Input
L3
L2
L2
L1
L1
Photodetector array
Hologram construction laser
Hologram interogation laser

Figure 2.2: Joint transform correlator after Collings 1987 [1]

inally [23] able to accommodate complex transmittances. Vander Lugt[2] [5] was the first to propose a method for bypassing the problem by using a holographic filter. To do that he recorded the intensity of the interference between the filter $F(u,v)$ and an off-axis reference beam. According to Kumar in [23], "*when this mask is placed in the back focal plane of the first FT lens, the light leaving it has three distinct components [23]: First is the product $kI(u,v)[A^2 + |F(u,v)|^2]$, where k is a normalising constant and A is the amplitude of the reference beam, and its inverse FT appears centered on the optical axis at the output plane. The second term is $kAI(u,v)F(u,v)e^{jau}$, where a is related to the angle of the reference beam. Its inverse FT is the convolution between the filter and the input functions, and it is placed along the x-axis on one side of the origin. The third term is $kAI(u,v)F^*(u,v)e^{-jau}$, whose inverse FT produces the desired correlation along the x-axis at the opposite side of the origin.*" Obviously the reference beam angle, a, plays an important role to the placement of the correlation at the output plane and a steep enough angle must be chosen to ensure good separation between the three terms.

2.3.2 Joint-transform correlator

The *joint transform correlator (JTC)* [9, 10] is based on a different approach, where the prior Fourier transformation of the filter is not necessary. In the joint

[2]Vander Lugt used amplitude masks made of photographic film and not SLMs in his implementation

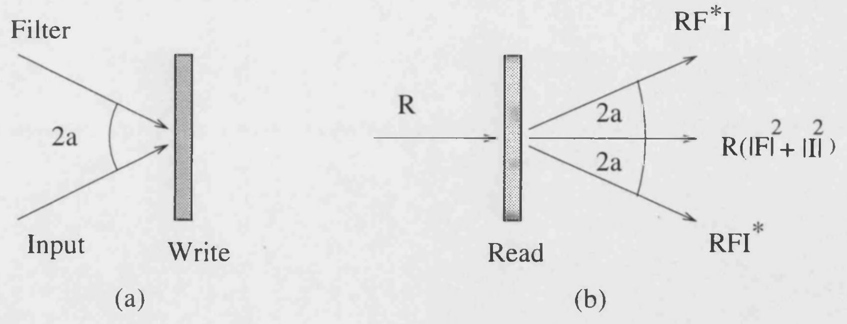


Figure 2.3: Output of the joint transform correlator after Collings 1987 [1]

transform correlator, figure 2.2, the input image and the filter are presented at the input plane of the FT lens (L3) at the same time, and are then both Fourier transformed by the lens. The interference pattern of their Fourier transforms is recorded in a real-time recording material such as a photorefractive crystal which is at the back focal plane of the FT lens. The hologram is interrogated with a collimated beam. The output beam is sent through a second FT lens and the correlation of the input and the filter patterns is obtained at the output. The reconstructed beam in the JTC consists of three terms (figure 2.3). The on-axis term is the sum of the auto-correlations of the object and the scene, $R(|F|^2+|I|^2)$. The off-axis terms are the terms of interest because they are the cross-correlations of the input and the filter, RFI^* and RF^*I, where R denotes the amplitude of the reference beam. To obtain a convolution, the mirror image of either the filter or the input function must be placed at the input of the JTC.

The main difference between the JTC and the 4-f correlator is that the JTC performs spatial-domain instead of Fourier domain filtering. In other words, the filter that is placed in the 4-f correlator must already be in the Fourier domain, while in the JTC the filter must be in the space domain. The main advantage of the JTC is that no great accuracy in the positioning of the input and the filter is required [1]. However, any change in the positioning of the input relative to the filter (or visa versa), will result in the change of the angle $2a$ between them and hence, the position of the cross-correlation peak at the output, as can be seen in figure 2.3. Provided that real-time devices are available, search routines can be performed at the frame rates of the SLMs. In addition, the JTC can be used for adaptive pattern recognition, where the input signal is continually being compared to a reference signal which is changing in time [23]. However, the optical quality of the input devices, and the FT lens used in the JTC must be high.

2.4 Optical Matched filters

A *matched filter* is a time-reversed version of the input signal $h(x) = s(-x)$, where $s(x)$ is the input signal. If the input signal is complex and 2D, then the time reversed complex conjugate of its frequency spectrum, $h(x, y) = s^*(-x, -y)$, is its matched filter. It has been proved that matched filters are optimal for detecting the presence of the signal $s(x)$ in a noisy input, when the noise is white with a constant power spectral density [11, 23]. This optimality of the matched filters in detecting signals buried in noise is proved because it maximises the output Signal-to-Noise Ratio (SNR), which leads to a minimum probability of error [24]. An optical matched filter can be implemented by a hologram containing the complex conjugate of the frequency spectrum of the pattern [1], or using an amplitude and a phase SLM.

2.5 Practical correlator implementations

Purely optical correlators have the advantage of being very fast, operating at over kHz rates [25, 26], but they suffer from several disadvantages such as the lack of versatility and programmability, and low accuracy due to the analogue nature of optics and the low dynamic range [27]. On the contrary, all of the previously mentioned deficiencies of the optical systems, are strong points of electronic computers. It is not strange, therefore, that many hybrid systems have been developed which combine the advantages of both worlds [16, 28, 29, 30, 31, 32, 33, 34, 35].

In many cases the input image must be correlated with a very large number of reference images. These reference images can either be stored in a computer and down-loaded to the correlator sequentially, or they can be stored optically. In that case the storage device is part of the correlator. Optical disks have been successfully used in correlators [36, 37, 38, 39]. Another solution is the use of photorefractive materials, which offer large storage capacity [40], high resolution and real-time recording and several correlators have been built which utilise them [41, 42, 43].

One other disadvantage of optical correlators is their bulk and large weight, as well as the fact that they are very sensitive to vibrations because precise alignment of the input and the filter image is necessary in some of them (4-f correlator) and, therefore, cannot easily be moved. In recent years, several attempts have been made to built compact correlators that are also able to endure vibrations [44, 45, 46]. Finally, several planar correlators have been built by integrating all of the optical components on the surfaces of a single substrate using lithographic fabrication techniques [47, 48, 49]. Most of these systems use spatial light modulators for the data input to the optical part of the system and CCD cameras for the correlation readout. Then the computer makes the decision based on the correlation output. In addition, all of the preprocessing of the data before it is down-loaded to the SLM and the post-processing of the correlation output is done by the computer.

It is apparent that the SLMs play a very important role in these architectures, since they depend not only on their speed, but also on their ability to modulate the amplitude or the phase of the passing light, or both [50, 51]. There is currently no SLM commercially available, which can simultaneously fully modulate the amplitude and the phase of the passing light. Therefore, several filters have been designed, which use only a part of the complex plane [16, 52]. In addition, a combination of two SLMs can be used for simultaneous amplitude and phase modulation [52, 53, 54, 55].

2.6 Performance measures

Several different performance measures have been proposed by various authors for the assessment of the performance of optical filters. The most frequently used of these performance metrics were summarised in a paper written by Kumar and Hassebrook [56]. Later in the thesis we are going to use some of these performance metrics to assess the performance of our filters and to compare it with the performance of other filters. Therefore, following the Kumar and Hassebrook paper, we present and explain the following performance metrics:

1. Signal-to-Noise Ratio

   The Signal-to-Noise Ratio (SNR) is defined as the ratio of the square of the average magnitude of the correlation peak, over the variance of the magnitude of the correlation peak:

   $$SNR = \frac{|E\{y(0)\}|^2}{var\{y(0)\}} \tag{2.5}$$

   The SNR gives us a measure of how much the correlation peak fluctuates when random noise is added to the input signal. Obviously, it is desirable to keep these fluctuations as small as possible, or in other words to maximise the SNR. It is evident from equation 2.5 that to calculate the SNR, one needs to calculate the average and the variance of the magnitude of the correlation peak. Therefore, many experiments with different noise samples have to be conducted for the SNR to be estimated.

2. Peak-to-correlation Energy

The Peak-to-Correlation Energy (PCE) is defined as the ratio of the square of the magnitude of the correlation peak, over the correlation plane energy:

$$PCE = \frac{|y(0)|^2}{E_y}, \tag{2.6}$$

where

$$E_y = \int_{-\infty}^{\infty} |y(x)|^2 \, dx \tag{2.7}$$

The PCE measures the sharpness of the correlation peak. When the correlation has a sharp high peak and low outer products, the energy of the whole correlation E_y will not be much larger than the energy concentrated on the peak, and the PCE will be large (close to 1). If the correlation peak is not sharp, or the outer products are large, then the PCE will be closer to 0.

3. Horner efficiency

   In 1982 Horner [57] introduced the Horner efficiency criterion. The Horner efficiency is the ratio of total light energy in the output plane to the light energy at the input plane and is described by the following equation

$$\eta_H = \eta_M \frac{\int |f(x) \otimes h(x)|^2 \, dx}{\int |f(x)|^2 \, dx} \tag{2.8}$$

   where $f(x)$ is the input function, $h(x)$ another function, η_M the diffraction efficiency of the recording medium, and the operator $\otimes$ indicates correlation. The Horner efficiency measures the amount of light that passes through the system.

2.7 Conclusions

In this chapter we have presented a theoretical background for optical correlators, which are the basic tools for optical pattern recognition. We explained the Fourier transform property of the lens, and described the two most important optical correlators, the 4-f correlator or frequency plane correlator, and the joint transform correlator. We introduced the concept of the matched filters, which are optimal in detecting the presence of a signal buried in white noise. We briefly reviewed the most important optical and electro-optical correlator implementations. Finally, we presented some of the most well known performance measures for the assessment of optical filters. In the next chapter we are going to review some filter design techniques.

Chapter 3

Review of spatial filter design algorithms for optical correlators

3.1 Introduction

Optical pattern recognition is a multi-faceted problem. It includes the detection of objects buried in noise [58, 59, 60, 61], the discrimination of different objects [62], the recognition of different views of a 2D [63, 64] or 3D [65, 66, 67] object and the discrimination of them from other views of a different 2D or 3D object [68]. Due to the complexity of these different recognition or discrimination problems, researchers have proposed many different methods and algorithms for the development of the appropriate filter for each case [69]. In this chapter, we review some of the algorithms proposed in the literature. Only a few of these algorithms, which are the most similar to our work and which will be compared to it in later chapters are presented in detail here. Different authors have used different notation in their publications. For the sake of clarity, we have changed that notation where necessary and we have used one set of symbols consistently throughout the chapter.

First a few words about notation: Throughout this thesis we denote patterns as vectors $\mathbf{s_j} = [s_{j1}, s_{j2}, ..., s_{jN}]^T$ of size N, where $\mathbf{s}_j$ is the j^{th} pattern of M patterns and N is the number of pixels in each image. We refer to $\mathbf{g}_j$ as being the filter for the j^{th} input vector after the application of the filter design algorithm. The un-normalised filters are denoted with $\tilde{\mathbf{g}}_j$. If the patterns are completely orthogonalised by this procedure we refer to the patterns as $\mathbf{u}_j$. $\tilde{\mathbf{u}}_j$ is used to refer to the un-normalised orthogonal patterns. The central peak of the cross correlation of two patterns is their inner product and it will be denoted by $\mathbf{g} \cdot \mathbf{s}$ and is equal to $\sum_{k=1}^{N} \mathbf{g}_k \cdot \mathbf{s}_k$ or in vector notation $\mathbf{g}^T\mathbf{s}$.

3.2 The Gram-Schmidt orthogonalisation procedure

Given a set of vectors $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_M$, in N dimensional space R^N, $M < N$, the Gram-Schmidt Ortho-normalisation procedure [70] constructs an orthonormal basis set of vectors spanning the space S=span$(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_M)$ (which is the set of all linear combinations of the vectors $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_M$). The algorithm begins by normalising the first vector $\mathbf{s}_1$,

$$\mathbf{u}_1 = \frac{\mathbf{s}_1}{||\mathbf{s}_1||_2} \tag{3.1}$$

where $|| \cdot ||_2$ denotes the Euclidean norm

$$||\mathbf{s}||_2 = \left(\sum_{i=1}^{N} s_i^2\right)^{1/2} \tag{3.2}$$

$\mathbf{s}_2$ is made orthogonal to $\mathbf{s}_1$ and it is normalised by the following two iterative steps:

$$\tilde{\mathbf{u}}_{k+1} = \left[\prod_{j=1}^{k}(\mathbf{I} - \mathbf{u}_j\mathbf{u}_j^T)\right]\mathbf{s}_{k+1}, \tag{3.3}$$

$$\mathbf{u}_{k+1} = \frac{\tilde{\mathbf{u}}_{k+1}}{||\tilde{\mathbf{u}}_{k+1}||_2} \tag{3.4}$$

Then $\mathbf{s}_3$ is made orthogonal to $\mathbf{s}_1$ and $\mathbf{s}_2$ and normalised and so on using the same iterative and normalising equations. Once k vectors have become orthogonal spanning a subspace $S_k \subset S$, $\mathbf{s}_{k+1}$ is projected onto the subspace orthogonal to S_k. Finally, all of the M vectors will be orthogonal, so S_M will be equal to S.

3.3 Linear Combination Filters

Matched filters are not very sensitive to geometric distortions and therefore, not very successful in multi-class pattern discrimination [18]. However, they can discriminate between input patterns, when these are orthogonal or can be made orthogonal. This is a result of the lower dynamic range that is required by the optical recognition system for correct discrimination of the orthogonal input patterns. A two step procedure for the design of Linear Combination Filters (LCFs) which were *mutually orthogonal*[1] (MO) was proposed by Caulfield and Maloney [18] as a solution to the discrimination problem. Each of the filters had unit output when correlated with one of the input patterns and zero with all of the others. The first step of the procedure was to calculate the vector inner-product matrix of the training patterns

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1M} \\ r_{21} & r_{22} & \cdots & r_{2M} \\ & \vdots & & \\ r_{M1} & r_{M2} & \cdots & r_{MM} \end{pmatrix} \tag{3.5}$$

where $r_{ij} = \mathbf{s}_i \cdot \mathbf{s}_j$ or in other words each element r_{ij} of the matrix was equal to the inner product between the input patterns $\mathbf{s}_i$ and $\mathbf{s}_j$. Caulfield's and Maloney's aim when testing pattern $\mathbf{s}_i$ for its identity to pattern $\mathbf{s}_k$, was to obtain an output, F_{ik}, equal to one if $i = k$ and equal to zero if $i \neq k$, i.e.,

$$F_{ik} = F_{kk}\delta_{ik} \tag{3.6}$$

They achieved their aim with the second step of the procedure, which was to form linear combinations of the responses r_{ij}. Using these linear combinations the final response when testing pattern $\mathbf{s}_i$ for its identity to $\mathbf{s}_k$ would be

$$F_{ik} = r_{ik} + \sum_{l \neq k} C_{kl} r_{il} \tag{3.7}$$

The $M - 1$ i's for which $i \neq k$ led to a set of $M - 1$ simultaneous equations with $M - 1$ unknowns, the coefficients C_{kl}. A different set of $M - 1$ simultaneous equations with $M - 1$ coefficients had to be solved for each of the M input patterns. After calculating the coefficients C_{kl}, Caulfield and Maloney used equation 3.7 to see whether an input pattern, $\mathbf{s}_i$ was the same as pattern $\mathbf{s}_k$. If the equation output was equal to one, then pattern i was the same as pattern k, and if it was equal to zero, then pattern i was different from pattern k. Caulfield's method does not actually produce new filters. Rather, it combines the inner products between the training patterns to obtain an output which will determine whether an input pattern is the same as another pattern. However, their coefficients can be used to create the actual filters, which will yield the desired outputs. Caulfield and Maloney mentioned this in their paper, but at the time that they wrote it, it was difficult to make these filters.

Later Caulfield and Haimes [71] proposed a more generalised solution to the multi-class - multi-object recognition problem with the *Generalised Matched Filter* (GMF). Their aim was to create filters which would be able to recognise all of the

[1] We call these cross-orthogonal, but our term has the same meaning.

patterns within a class and, in addition, discriminate between members of different classes. They supposed that each input object, s was of size N. For each class of objects they calculated a *Linear Discriminant Function* (LDF), which they used as the generalised matched filter. This filter would have a high correlation output with any pattern that belonged to the class it represented and a low output for any pattern which belonged to any of the other classes. The *LDF* was a real function of the training pattern s

$$LDF_i(\mathbf{s}) = \mathbf{V}_i \cdot \mathbf{s} + Q_{0i} = F_i(s) \tag{3.8}$$

where $\mathbf{V}_i = (V_1, V_2, \dots, V_N)^T$ was a set of real numbers and Q_{0i} was a real number. They chose the *LDF* which would have a high output with the members of the class it represented and a low output with all of the other input objects by maximising the equation

$$E[LDF_i(\mathbf{s} \in Class_i) - LDF_i(\mathbf{s} \notin Class_i)] \tag{3.9}$$

where $E[\cdot]$ was the expectation operator. In other words LDF_i was that linear function of s that maximised the probability to distinguish $\mathbf{s} \in Class_i$ from $\mathbf{s} \notin Class_i$. If the *LDFs* were normalised, then

$$E[LDFi(\mathbf{s} \in Class_j)] = \delta_{ij}. \tag{3.10}$$

Equations 3.10 and 3.6 show that the mutually orthogonal filters are a subset of the generalized matched filters, because if each of the classes only consists of one pattern, then equation 3.10 expresses the same condition as equation 3.6.

A filter that would have equal correlation outputs with all of the patterns representing one class in a multi-class recognition problem was proposed by Hester and Casasent [72]. It was called the *Equal Correlation Peak* (ECP) filter. Firstly the Gram-Schmidt procedure was used to orthogonalise the training images $\mathbf{s}_j$ and to produce a new set of orthogonal vectors $\mathbf{u}_j$ that formed an orthonormal basis of the space of the input and training images. Then the input images, $\mathbf{f}$, and the training images, s were expanded in this set of orthonormal vectors $\mathbf{u}_j$

$$\mathbf{f} = \sum_j a_j \mathbf{u}_j \tag{3.11}$$

$$\mathbf{s} = \sum_j b_j \mathbf{u}_j \tag{3.12}$$

and the input and training images could be represented by the coefficients a_j and b_j

$$\tilde{\mathbf{f}} = (a_1, a_2, \dots, a_k) \tag{3.13}$$

$$\tilde{\mathbf{s}} = (b_1, b_2, \dots, b_k) \tag{3.14}$$

In terms of these expansions the inner product of **f** and **s** could be described by

$$r_{fs} = \tilde{\mathbf{f}} \cdot \tilde{\mathbf{s}} = \sum_j a_j b_j \tag{3.15}$$

The objective was to design a filter, **g**, which would have equal correlation outputs with all of the inputs, $\mathbf{f}_j$, which belong to the same class. Hester and Casasent argued that this filter had to be a specific linear combination of the input images, each of which was another linear combination of the basis functions **u**

$$\tilde{\mathbf{g}} = \sum_j c_j \tilde{\mathbf{f}}_j \quad \text{or} \quad \mathbf{g} = \sum_j c_j \mathbf{u}_j \tag{3.16}$$

and the correlation outputs could then be described by

$$r_{sg} = \sum_j b_j c_j \tag{3.17}$$

So after finding the orthogonal vectors $\mathbf{u}_j$, using the Gram-Schmidt orthogonalisation procedure, and the coefficients b_j, using equation 3.12, the objective was to find the coefficients c_j and finally the filter **g** for which r in equation 3.17 would yield the correct correlation performance. If one required r_{sg} (equation 3.17) to be equal for all training patterns **s**, he could solve the resulting set of equations to obtain the coefficients c_j.

3.4 Synthetic Discriminant Functions

The work on linear combinations of training images, i.e. LCFs, ECPs and GMFs, was summarised by Caulfield [73] and Casasent and Kumar *et. al.* and it was formulated as a matrix/vector problem [13, 74]. The solution vectors $\mathbf{a}_i$ to the LCFs were described by the equation

$$\begin{aligned} \mathbf{R}\mathbf{a}_i &= \mathbf{d}_i \\ \Rightarrow \mathbf{a}_i &= \mathbf{R}^{-1}\mathbf{d}_i \end{aligned} \tag{3.18}$$

where **R** was the $M \times M$ correlation (alternatively called vector-inner product) matrix of the input images and **d** was the vector with the desired correlation outputs. The vector-inner product matrix **R** was invertible if and only if the input patterns were linearly independent [69]. Then the filters could be obtained using these solution vectors

$$\mathbf{g}_i = \sum_k a_k \mathbf{s}_k \tag{3.19}$$

where a_k are the elements of the solution vector $\mathbf{a}_i$. Depending on the desired correlation output vectors, $\mathbf{d}_i$, equation 3.18 was equivalent to equation 3.7, if mutually orthogonal filters were required.

Caulfield's and Maloney's approach (mutually orthogonal filters) meant that one filter had to be designed for each of the M patterns that one wanted to recognise. Each input pattern had to be correlated with all of them. Therefore, M correlations were necessary for correct recognition. Braunecker *et. al.* [75] suggested that M filters were redundant and that one only needed to perform at most $L = \log_2 M$ correlations to correctly recognise M filters. Braunecker's approach was based on the fact that $L = \log_2 M$ binary digits can form any number between 0 and M. For example, to recognise 4 patterns one needed only two filters, the first of which should yield a high correlation peak only with the second and the fourth input pattern and the second filter should produce a high correlation peak only with the third and the fourth input pattern. Braunecker's approach could also be applied to multi-class pattern recognition. The two previously reviewed methods, i.e. linear discriminant functions and equal correlation peak filters designed one filter for each class. Therefore K, where K is the number of classes one wants to recognise, correlations were necessary for correct recognition of an input pattern. According to Braunecker's method, only $L = \log_2 K$ correlations are necessary.

Even faster recognition could be achieved if only one filter was designed, which gave the same correlation peak value for all of the patterns that belonged to one class and a different, in intensity, correlation peak value for all of the patterns that belonged to another class and so on. This particular linear combination filter was called a *Synthetic Discriminant Function* (SDF) [13, 74]. The advantage of SDF was that only one correlation would be necessary to recognise any of the input patterns. Their disadvantage was that they required that the recognition system had a high dynamic range, because several different correlation peak values had to be correctly identified at the output plane. A year later, in 1983, the *Modified Hyperplane Method* (MHP) for more efficient design of Linear Combination Filters (LCFs) was proposed by Kumar [76]. A systematic procedure for determining the output correlation values for SDFs, instead of arbitrarily setting them to 0 and 1, was proposed by Sudharsanan and Mahalanobis *et. al.* [77]. The proposed technique provided an optimal selection of the output correlation values in the sense that they resulted in a minimization of the probability of error (POE) in detection.

Several variations of the SDFs were proposed in the following years. Kallman [78] showed that standard SDFs were less than optimal due to low output SNR. In other words, SDFs correlated very well with true targets, but also very often they gave high correlations with false targets. He observed that only intensities were detected at the correlation plane, so output correlation values could have arbitrary phase. He used this additional degree of freedom to reformulate the equation describing the SDFs (equation 3.18) in the following manner

$$\mathbf{g} \cdot \mathbf{s}_i = z_i \lambda_i \quad (1 \leq i \leq M) \tag{3.20}$$

where $\mathbf{s}_i$,$(1 \leq i \leq M)$ were the complex images of objects one wanted to recognise The λ_i were given positive numbers and z_i were complex numbers of modulus 1. Equation 3.20 gave M simultaneous equations that $\mathbf{g}$ had to satisfy and a particular solution to this equation would have the form

$$\mathbf{g}_0 = a_1\mathbf{s}_1 + \ldots + a_M\mathbf{s}_M \tag{3.21}$$

where $a_1 \ldots a_M$ were a set of complex numbers. These numbers could be found by substituting equation 3.21 into 3.20

$$(\mathbf{s}_i \cdot \mathbf{s}_j)(a_j) = (z_i \lambda_i) \tag{3.22}$$

Equation 3.22 uniquely determined the complex numbers a_i if the images $\mathbf{s}_1 \ldots \mathbf{s}_M$ were linearly independent and is identical to the general SDF solution equation shown in 3.18. Kallman proposed that one could maximise the SNR of the filters by varying the z_i phase values of the inner products and choosing the appropriate of many possible solutions to equation 3.20. Using his method, Kallman managed to construct filters with their SNR properties improved by a factor of seven [78].

3.4.1 Minimum Variance Synthetic Discriminant Function

As we saw in the previous section, SDFs yield one correlation peak with a different intensity value for each of the classes to be recognised. As the number of classes increases, the different values of the correlation peak will be closer to each other, because more of them will be needed in an overall limited range. This means that the variance[2] of the correlation peak is critical for the filter's performance. The *Minimum Variance Synthetic Discriminant Function* (MVSDF) which minimised the variance of the correlation peak, which was caused by noise, was introduced by Kumar [14]. Kumar addressed the problem where the input was one of the training images with some additive noise. In that case the output of the correlation at the origin of the correlation plane would be

$$y = \mathbf{g}^{+}(\mathbf{s}_i + \mathbf{n}) = c_i + \mathbf{g}^{+}\mathbf{n} \tag{3.23}$$

[2]For a definition look at appendix A

where $\mathbf{g}$ denoted the filter which was designed to satisfy equation 3.18, $\mathbf{g}^+$ denoted the conjugate transpose, $\mathbf{s}_i$ denoted the input pattern and $\mathbf{n}$ was a zero-mean noise vector with a covariance[3] matrix Σ. The output in this case was the desired output c_i plus an undesirable random variable $\mathbf{g}^+\mathbf{n}$. The MVSDF attempted to design the filter $\mathbf{g}$ in such a way so that the variance in the output caused by the input noise was minimised while satisfying equation 3.18. The variance of the output caused by $\mathbf{g}^+\mathbf{n}$ was

$$\sigma_y^2 = E\{|\mathbf{g}^+\mathbf{n}|^2\} = E\{\mathbf{g}^+\mathbf{n}\mathbf{n}^+\mathbf{g}\} = \mathbf{g}^+\Sigma\mathbf{g} \tag{3.24}$$

and minimising σ_y^2 shown in equation 3.24 led to the following MVSDF

$$\mathbf{g}_{MVSDF} = \Sigma^{-1}\mathbf{S}(\mathbf{S}^+\Sigma^{-1}\mathbf{S})^{-1}\mathbf{d}^* \tag{3.25}$$

where $\mathbf{d}$ denoted the vector with the desired filter outputs, $\mathbf{d}^*$ was the complex conjugate and $\mathbf{S}$ was a data matrix with the vector $\mathbf{s}_i$ as it's i_{th} column. Kumar, Bahri and Mahalanobis showed [66] that the output noise variance of minimum variance synthetic discriminant functions (MVSDFs) could be further reduced by selecting the phase values of the output correlation in an optimal fashion, an idea similar to that of Kallman [78]. They proposed using the same MVSDF as described in equation 3.25, but also to properly select the phases of the desired correlation outputs $d_i = \beta_i exp(j\theta_i)$, $i = 1, 2, \ldots, N$ in such a way so that the output variance σ^2_{MVSDF} was minimised. The exact reduction in variance could vary from being negligible to being significant and depended on the training images, the noise covariance matrix and on the constraint magnitudes. The synthesis of the MVSDF was simplified by eliminating the need to invert large noise covariance matrices when the background clutter was modeled as sample realisations of a Markov noise process by Kumar and Casasent *et. al.* [79].

SDFs which were not affected by noise with non-zero mean were proposed by Arsenault and Sheng *et. al.* [80]. They noted that any noise with a non-zero mean could be written as the sum of a zero mean noise plus a constant. Therefore, the correlation between a filter and an input image corrupted by non-zero mean noise would be

$$\begin{aligned} g \cdot (s + n + \beta) &= r_{gs} + r_{gn} + \beta \iint_{-\infty}^{\infty} g(x, y)\, dx\, dy \\ &= r_{gs} + \beta K, \end{aligned} \tag{3.26}$$

where

$$K = \iint_{-\infty}^{\infty} g(x, y)\, dx\, dy \tag{3.27}$$

[3]For a definition look at appendix A

which meant that the effect of the non-zero mean part of the noise was to add to the correlation a term which was proportional to the mean β of the noise. They proposed modifying the composite filters which were linear combinations of the training images by adding a term which discriminated against a constant background. For example, for two training images s_1, s_2 the composite filter proposed by Caulfield and Maloney [18] was

$$g_1 = as_1 + bs_2 \tag{3.28}$$

where a and b were constants which were found by solving the simultaneous equations

$$ar_{s_1s_1} + br_{s_1s_2} = 1 \tag{3.29}$$

$$ar_{s_2s_1} + br_{s_2s_2} = 0 \tag{3.30}$$

The composite filter proposed by Arsenault *et. al* was given by the equation

$$g_2 = as_1 + bs_2 + c\xi \tag{3.31}$$

where the constants a, b, c where found by solving the set of linear equations

$$g_2 \cdot s_1 = ar_{s_1s_1} + br_{s_1s_2} + cr_{s_1\xi} = 1 \tag{3.32}$$

$$g_2 \cdot s_2 = ar_{s_2s_1} + br_{s_2s_2} + cr_{s_2\xi} = 0 \tag{3.33}$$

$$g_2 \cdot \xi = ar_{\xi s_1} + br_{\xi s_2} + cr_{\xi\xi} = 0 \tag{3.34}$$

where $\xi(x, y)$ was a uniform background with an image size equal to or greater than the training images s_1 and s_2. The modified filter nullified the effect of the non-zero mean of the noise and hence the output correlation did not depend on the mean β of the noise.

A special case of input noise is the nonoverlapping target and scene noise. For example, in military applications very often one is trying to recognise armored vehicles which are positioned on a varying terrain, i.e. a noisy background, but the targets themselves are not obstructed by anything. Javidi, Réfrégier and Willet designed a filter for pattern recognition with nonoverlapping target and scene noise [81]. They showed that in this case the filter did not depend on the noise statistics.

3.4.2 Minimum Average Correlation Energy filters

The MVSDF and the other filters discussed so far only controlled one point at the origin of the correlation plane. However, in many applications, for example target detection in the military, we do not know were exactly in the input scene the target lies. Therefore, the filter must be able to locate the target and, in addition, to recognise it. In that case, a sharp correlation peak is preferable to a correlation plane full of high side-lobes. The *Minimum Average Correlation Energy* (MACE) filter which was introduced by Mahalanobis and Kumar *et. al.* [82] minimised the average correlation plane energy over all of the training images. The authors calculated the average correlation plane energy over all of the training images to be

$$E_{ave} = \mathbf{g}^{+}\mathbf{D}\mathbf{g} \tag{3.35}$$

where $\mathbf{D}$ denoted a $N \times N$ diagonal matrix. The entries along the diagonal were obtained in the following manner: first one calculated the average of the energies of the two-dimensional Fourier transforms $|S_i(u,v)|^2, i = 1, 2, \ldots, M$ of the training images $s_i(x,y)$. Then he scanned this average from left to right and from top to bottom and placed each value on the diagonal of matrix $\mathbf{D}$. N was the size of the complex column vector $\mathbf{s}_i$ obtained by sampling $S_i(u,v)$. $\mathbf{g}$ was the SDF satisfying the constraint

$$\mathbf{S}^{+}\mathbf{g} = \mathbf{d}^{*} \tag{3.36}$$

where $\mathbf{S}$ was a $N \times M$ matrix with $\mathbf{s}_i$ as its i_{th} column. Minimising E_{ave} in equation 3.35 subject to the constraints in equation 3.36 led to the following filter

$$\mathbf{g}_{MACE} = \mathbf{D}^{-1}\mathbf{S}(\mathbf{S}^{+}\mathbf{D}^{-1}\mathbf{S})^{-1}\mathbf{d}^{*} \tag{3.37}$$

MACE filters produced sharp correlation peaks but had some drawbacks. The first was that no noise tolerance had been built into these filters. The second was that MACE filters seemed to be more sensitive to intraclass variations than other composite filters [83]. Also the MACE filter was calculated in the frequency domain. However, while the MACE filter minimised the energy of the circular correlation, most optical and electronic systems generate linear correlations and ideally one would want to minimise the energy of those. The main difference between the circular and the linear correlation of two patterns of size N, is that the linear correlation has a length equal to $2N - 1$, while the circular correlation has a length equal to N. This length difference between the linear and the circular correlation results in a difference in their energies. A space-domain MACE filter termed the *SMACE* filter was proposed in 1990 by Sudharsanan *et. al.* [84]. The SMACE filter avoided the problem of circular correlations of MACE filters, however, this advantage came at the cost of having to invert a matrix which was not diagonal like the $\mathbf{D}$ matrix of the MACE filter.

In 1988 Bahri and Kumar [12] offered a general SDF solution in both spatial and frequency domains and derived and proved the uniqueness of the MVSDF and the MACE filters. This general SDF solution for the spatial domain was

$$\mathbf{g} = \mathbf{g}_0 + \mathbf{F}\mathbf{z} \tag{3.38}$$

where

$$\mathbf{g}_0 = \mathbf{S}(\mathbf{S}^T\mathbf{S})^{-1}\mathbf{d}, \tag{3.39}$$
$$\mathbf{E} = \mathbf{S}(\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T \tag{3.40}$$

and

$$\mathbf{F} = \mathbf{I}_d - \mathbf{E} \tag{3.41}$$

where $\mathbf{z}$ was an arbitrary element of $\mathbf{R}^d$. By allowing $\mathbf{z}$ to vary throughout $\mathbf{R}^d$ one could get all of the possible SDFs and by subjecting them to various performance criteria like minimum noise variance or correlation energy for example one could get specific SDF filters like the MVSDF, MACE filters, etc.

3.4.3 MICE and MINACE filters

The *Minimum Correlation Energy* (MICE) filter which provided better intraclass recognition than the MACE filter and the *Minimum Noise And Correlation Energy* (MINACE) filter which minimized the correlation plane energy resulting from the training images and the noise were proposed by Ravichandran and Casasent [2]. The authors noted that minimising the average correlation plane energy provided little control over the variance of the correlation plane energies of the training images. In other words, large side-lobes could occur even though the average energy E was minimised. The MICE filter was described by the following equation

$$\mathbf{g}_{MICE} = \mathbf{T}^{-1}\mathbf{S}(\mathbf{S}^{+}\mathbf{T}^{-1}\mathbf{S})^{-1}\mathbf{d} \tag{3.42}$$

where $\mathbf{T}$ was a $N \times N$ diagonal matrix whose diagonal elements were obtained in a similar manner to those of matrix $\mathbf{D}$ in the MACE filter. Specifically, the energies of the two dimensional Fourier spectra of the training images were calculated again, but in this case the maximum of the $|S_i(u,v)|^2, i = 1, 2, \ldots, M$ for each u, v was chosen. This was scanned from left to right and from top to bottom and the values were placed at the corresponding diagonal elements of matrix $\mathbf{T}$. The result of this difference was that the MICE filter reduced the biggest side-lobes instead of the average energy over all of the training images. Also the MICE filter provided less amplification of the input data at high frequencies and, therefore, it had reduced sensitivity to finer image details which resulted in improved intraclass recognition capability. The MINACE filter in addition minimised the correlation plane energy which resulted from the input noise. Its construction was very similar to that of the MICE filter but in this case, the diagonal elements of matrix **T** were the maximum of the corresponding elements of the spectra of the training images *and* the input noise. In the absence of input noise, the MINACE filter reduced to the MICE filter.

3.5 Phase-Only Filters

Caulfield commented that an increase in Horner efficiency could result in a decrease in the quality of discrimination with many filters [85]. He also noted that the phase-only filter (POF) is the only one which can provide 100% efficiency and also lead to good discrimination. In 1984 Horner and Gianino [86] compared the classical matched filter with the amplitude- and the phase-only matched filter using the criteria of discrimination, correlation peak and optical efficiency. They came to the conclusion that the phase-only filter (POF) has higher optical efficiency and a sharper correlation peak (lower side-lobes) than the others, at a cost of lower SNR. In 1985 Horner and Leger [87] compared the phase-only filter with the binary phase-only filter (BPOF) and reported that there were several advantages in using the BPOF, mainly in their fabrication, at the cost of slightly lower SNR at the output. Horner and Gianino [88] also compared a phase-only and a binary phase-only SDF with a classical SDF and reported that the PO SDF and the BPO SDF produced sharper correlation peaks, higher SNR and increased correlation intensity.

Filter implementation constraints i.e. discrete SLMs etc. were used directly in the filter (SDF) calculation equation by Jared and Ennis [89]. One conclusion they came to, was that when doing that for POFs, one cannot set the correlation peaks to absolute values, but rather specify the proportionality between the correlation peaks for a given training set, a conclusion very similar to that of Kallman in [90]. One such filter synthetic discriminant function (fSDF) for a set of space shuttle training images and a specific magneto-optic SLM (MOSLM) was calculated by Reid and Ma *et. al.* [67] by building a correlator and using that to calculate it.

3.6 Optimal Trade-off Filters

The *Optimal Trade-off Filters* (OTFs) were introduced by Réfrégier in [17], where he used the Optimal Characteristic Curve (OCC) to design filters that were optimized between two criteria: the correlation peak sharpness and the noise robustness. Réfrégier and Huignard [91] showed that the optimization of the sharpness of the correlation peak was more fruitful and of equal complexity with the variance reduction. In 1991 [92] Réfrégier added the Horner efficiency to the criteria used for the design of the OTFs presented in [17]. Later Laude and Réfrégier [16] introduced a multi-criteria optimization method based on a geometrical interpretation of trade-offs between criteria, for any Fourier SLM coding-domain constraint and applied it to the SNR, peak-to-correlation energy (PCE), and Horner efficiency (η_H) criteria.

3.7 Discussion and summary

There are a variety of pattern recognition problems that the field of optical pattern recognition addresses, and therefore, several different filters have been designed to solve them. All of the filters that we have reviewed in this chapter fall under the general category of linear combination filters. They are all composed using different linear combinations of the training patterns. The mutually orthogonal filters are good at recognising the one pattern which they represent. They mainly address the problem of discriminating between objects, each of which is represented by one pattern only. Their main disadvantage is that a large number of them, equal to the number of the input patterns, and consequently a large number of correlations is needed for correct recognition.

Equal correlation peak filters address the opposite problem of recognising several patterns, all of which belong to the same class. When several of these classes exist, one can design an equal correlation peak filter for each one of the classes, a method proposed by Caulfield [73]. Using this method one needs a number of filters, and subsequently correlations, equal to the number of classes. Braunecker's suggestion that only $L = \log_2$(number of patterns or classes) correlations are necessary, applies to both the discrimination problem addressed by the mutually orthogonal filters and to the multi-class recognition problem addressed by Caulfield in [73]. The advantage of using Braunecker's method is that the number of necessary correlations for correct recognition is greatly reduced. In addition, each of the filters still has to produce only two outputs, one and zero. Therefore, the dynamic range that is required by the recognition system is not increased when using Braunecker's method. However, one must be very careful when choosing the patterns or classes to which each of the filters is going to respond with a high output.

Synthetic discriminant functions are also linear combinations of the training patterns. One filter is now used to recognise any number of patterns or classes. In multi-class pattern recognition, the filter is designed to produce a specific cor-

FILTER CHARACTERISTICS											
Filter		T.D.	Discr.		Rec.		N.T.	Corr. plane			D.R.R.
			M-P	M-C	M-P	M-C		P.S.	P.V.	S.R.	
LCF			√								√
GMF			√	√	√	√					
ECP				√	√	√					√
SDF	F.C.		√	√	√	√					
	POF		√	√	√	√		√			
MVSDF			√	√	√	√	√		√		
MACE		√	√	√	√	√		√		√	
MICE		√	√	√	√	√		√		√	
MINACE		√	√	√	√	√	√	√		√	
OTF		√	√	√	√	√	√	√	√	√	

Table 3.1: Summarised filter characteristics. T.D. : Target detection, Discr. : Discrimination, Rec. : Recognition, N.T. : Noise tolerance, Corr. plane : Correlation plane, M-P : Multi-pattern, M-C : Multi-class, P.S. : Peak sharpness, P.V. : Peak variance control, S.R. : Side-lobe (outer product) reduction, D.R.R. : Dynamic range reduction.

relation peak magnitude for all of the patterns that belong to one class. This output correlation peak value has to be different for each of the classes. The obvious advantage is that only one filter and one correlation is required to recognise any of the input patterns. The disadvantage of SDFs is that a higher dynamic range is now required by the recognition system because more than two different output correlation peak values will have to be distinguished. Several variations of the SDFs were proposed to improve their performance. Minimum variance SDFs (MVSDF) minimise the variance of the output correlation peak resulting from input noise. Minimum average correlation energy filters (MACE) minimise the average correlation plane energy over all of the training images. The motivation for that is to decrease the correlation side-lobes, so that the correlation peak can be easily located. Minimum correlation energy (MICE) are similar to the MACE filters, but instead of minimising the average correlation plane energy, they minimise the highest side-lobes. Minimum noise and correlation energy (MINACE)

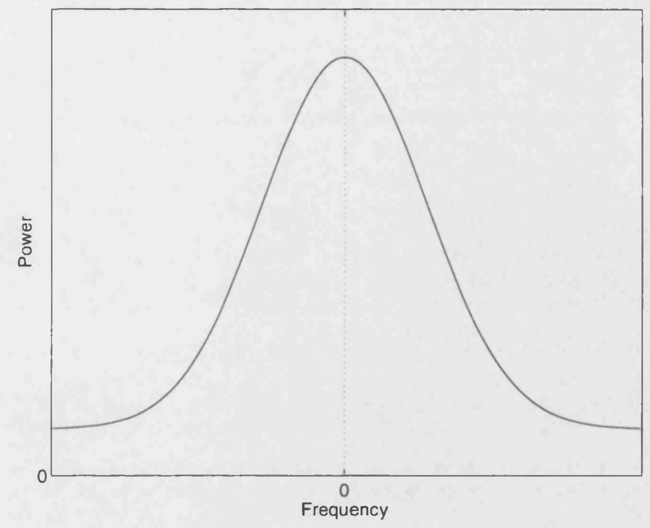


Figure 3.1: Spectral power distribution of an input pattern

filters also minimise the highest side-lobes, but in addition, they take into account the input noise.

In multiclass pattern recognition, there is a trade-off between intra-class distortion tolerance and inter-class discrimination ability. The filter must be tolerant to intra-class distortions, or in other words, must be able to recognise different patterns, which belong to the same class. In addition, the filter must be able to discriminate these patterns from other patterns belonging to other classes. Let us consider the spectral power distribution of an input pattern. For simplicity we will consider the simple case of a distribution resembling a Gaussian as shown in figure 3.1. The low frequencies in the area around the DC term represent the very general features of the pattern and are usually features which are common to many patterns of different classes. Therefore, for good inter-class discrimination ability, these frequencies must be suppressed by the filter. A typical example for such a task is the inverse filter (figure 3.2), which suppresses the low frequencies and enhances the low power, high frequencies, which express the finer image details, and, hence, has a high inter-class discrimination ability. It is however, sensitive to intra-class pattern variations. On the other hand, the matched filter, which is the complex conjugate of the pattern's frequency spectrum and which has a spectral power distribution, which is the same as that in figure 3.1, suppresses the low power, high frequencies and, hence, the finer image details. This leads to high tolerance to intra-class distortion, but low inter-class discrimination ability. Therefore, for overall high multiclass recognition performance, a bandpass filter (figure 3.3) is required, because it suppresses both the high and the low frequencies

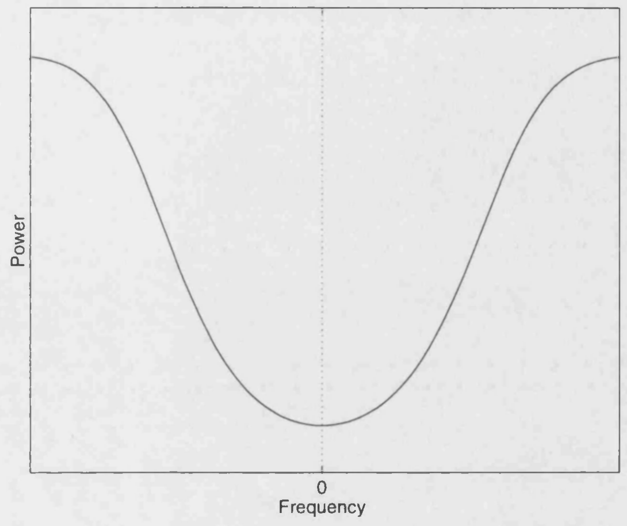


Figure 3.2: Inverse filter for the gaussian distribution of figure 3.1

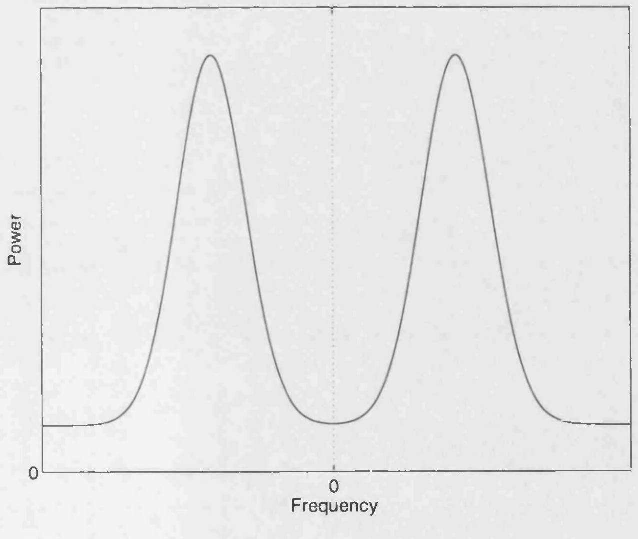


Figure 3.3: A bandpass filter for multiclass pattern recognition

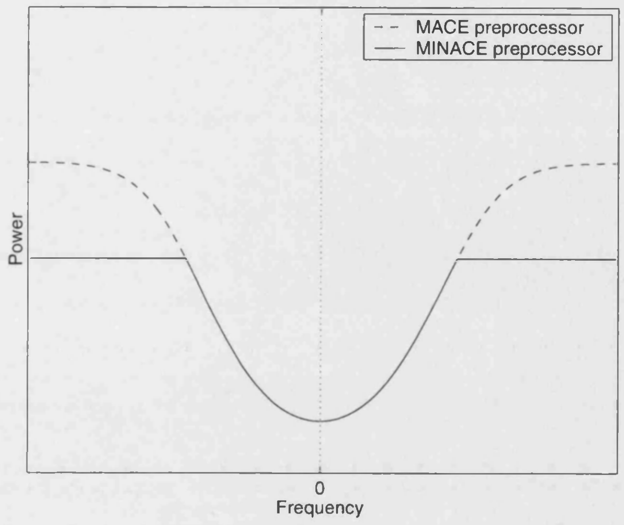


Figure 3.4: Effect of the MACE and MINACE preprocessors on the signal. Graph adapted from [2]

(including the DC term) of the input pattern's frequency spectrum.

A bandpass filter is, therefore, the goal and most of the existing filters, including the MACE and the MICE filters, are designed with that goal in mind[2]. However, they are not true bandpass filters because they suppress the low frequencies but they do not suppress the high frequencies. In that respect they are closer to inverse filters. The filter (of the ones reviewed in this chapter), which most closely approaches the bandpass goal is the MINACE filter, but even this does not suppress the high frequencies. However, it does not enhance them as much as the MACE, MICE and inverse filters do. The MINACE filter is described in equation 3.42. The relationship between the MINACE and a bandpass filter can be seen more clearly if the MINACE filter is viewed as a general SDF for recognising input patterns, which are preprocessed by a matrix $\mathbf{T}^{-1/2}$ [2].

$$\mathbf{g}_{MINACE} = \mathbf{T}^{-1/2}[\mathbf{Y}(\mathbf{Y}^{+}\mathbf{Y})^{-1}\mathbf{d}] \tag{3.43}$$

where the columns of the matrix $\mathbf{Y}$ are the preprocessed training images $\mathbf{Y}_i$, $\mathbf{Y}_i = \mathbf{T}^{-1/2}\mathbf{S}_i$. The preprocessor matrices $\mathbf{T}^{-1/2}$ for the MACE and the MINACE filters (look back in section 3.4.2 for a description of matrix $\mathbf{T}$) are shown in figure 3.4. From this figure, the preprocessor for the MACE filter has a form similar to that of an inverse filter. The preprocessor for the MINACE filter does not enhance the high frequency components of the spectrum of the input signal as much and therefore, it resemples a bandpass filter more closely.

The optimal trade-off filters (OTF) are filters which simultaneously optimise more than one criteria, for example, the correlation peak sharpness and the noise robustness. All of the reviewed filters are complex. The phase-only filters (POF) and the binary phase-only filters (BPOF) are the same as the previously described filters but with only their phase information retained. Their main advantages are that it is easier to use them in optical correlators using a phase SLM, they have higher optical efficiency compared to amplitude or fully complex filters and they produce a sharper correlation peak. Finally, in table 3.1 we can see a summary of the characteristics of all of the filters. We have marked a box in table 3.1, when the corresponding filter is capable of performing that task and not only when it is the best filter at the corresponding task.

Chapter 4

Neural networks: Perceptrons and learning algorithms

4.1 Introduction

In this chapter we present some basic theory on neural networks and in particular perceptrons. Neural networks are not the main application area for this thesis. Therefore, the theory presented here does not cover the whole field of neural networks, but rather a specific type of them: the perceptron. In addition, the perceptron is not explained in detail, only the information which is necessary for the understanding of our work is presented.

Artificial neural networks are highly parallel systems consisting of a large number of simple, interconnected, processing units. They store data in the form of the strengths of the interconnections between the units [3]. Information processing in neural networks occurs through the interactions between the processing units. They interact with each other by sending signals, either excitatory or inhibitory. The interconnection strengths are usually called weights [3]. Work on artificial neural networks started when scientists realised that the human brain works in an entirely different way from the conventional digital computer. The brain is a highly complex, nonlinear, and parallel computer. It consists of approximately $100 \cdot 10^9$ simple computational units called *neurons* [93]. The neurons are connected to each other with *synapses* and there are approximately $60 \cdot 10^{12}$ of them in total in the human brain [94]. Artificial neural networks were first designed to mimic the structure and function of the human brain. Due to their origin in human brain research, they have borrowed the biological terminology. Therefore, their processing units are called neurons and the interconnections between neurons are often referred to as synapses.

Rumelhart *et al* [95] point out that knowledge in neural networks, as in the brain and not as in conventional computers, is not an actual copy of the data that is stored. Rather, it is the strengths of the interconnections between neurons that are modified in such a way, so that the stored data can be *recreated* when it is called up. This has great implications in the way that neural networks can be trained. Learning with this model is not a matter of finding a way to represent the information to be learned with the weights, but rather tuning them in such a way so that the right pattern of activation can be created as a result of a specific input. This is a very important property of this kind of model, because it means that they can learn the interdependencies between the various activations to which they are exposed by tuning their weights during the course of processing. The procedure used to perform the learning process is called a learning algorithm and it is a simple mechanism which modulates the interconnection strength according to the information locally available at the connection.

The learning procedure can be supervised or unsupervised. When it is supervised, some form of teacher exists. The network is presented with pairs of inputs and desired outputs. The weights are modified in such a way so that the error, which is defined as the difference between the network output and the desired output, is minimised. In the unsupervised or self-organised learning there is no external teacher. Some measure of the correctness of the representation of the statistics of the environment is defined and the network modifies it's weights so that it's performance is optimised with respect to this measure [96].

4.2 The Perceptron

The *perceptron*, which was introduced by Rosenblatt [97], is the simplest form of a neural network . It only consists of one layer of neurons connected to the inputs through weighted connections. The simplest form of the perceptron constitutes only one neuron with any number of inputs as seen in figure 4.1. The perceptrons can only be used for the classification of *linearly separable* patterns [98]. Patterns are called *linearly separable* if they lie on either side of a hyperplane in N-dimensional space. A single neuron can separate two classes. An example of two linearly separable classes in two-dimensional space can be seen in figure 4.2. If the perceptron contains more than one neuron it can perform separation of more than two classes as long as they are linearly separable. The neurons output

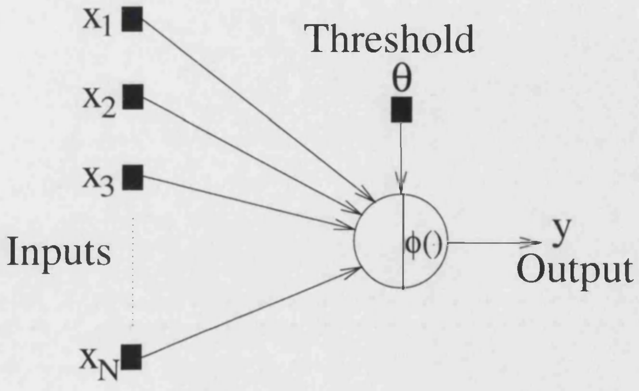


Figure 4.1: A single layer perceptron with only one neuron

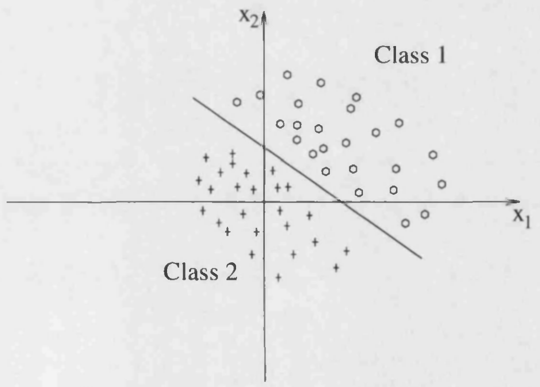


Figure 4.2: An example of two linearly separable classes in 2-D space.

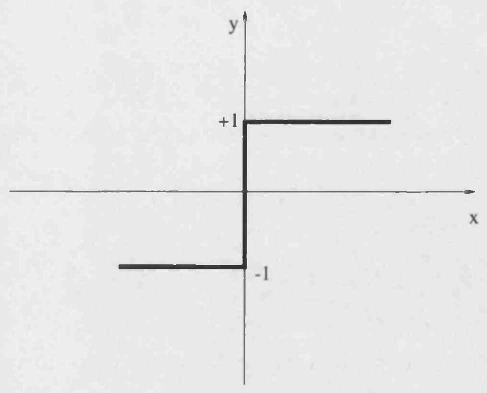


Figure 4.3: A hard limiter activation function

is given by the following equation

$$v = \sum_{i=1}^{N} w_i x_i - \theta \tag{4.1}$$

$$y = \phi(v) \tag{4.2}$$

where $x_i, i = 1, \ldots, N$ are the network inputs, $w_i, i = 1, \ldots, N$ denote the weights of the network and θ is the threshold. v is sometimes called the *net internal activity level* of the neuron. Usually function $\phi(\cdot)$ is a non linear hard limiter (See figure 4.3) and the neuron output is either 1 or -1. The separating hyperplane is defined by the equation $\sum_{i=1}^{N} w_i x_i - \theta = 0$. The first learning algorithm for the perceptron was developed by Rosenblatt [99], [100]. The proof of convergence is known as the *perceptron convergence theorem.*

4.2.1 Hebbian learning

Hebb's postulate of learning [101] is the oldest and most famous learning rule. Its purpose is to discover significant patterns of features in the input data. To do that, the algorithm is provided with a set of rules of a local nature, which enable it to learn to compute an input-output mapping with specific desirable properties. The original Hebb's rule has been expanded and rephrased by Stent [102], and Changeux and Danchin [103] and can be described as follows:

1. When two neurons on either side of a synapse are activated simultaneously, then the strength of the synapse is selectively increased.
2. If the two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

The Hebbian learning law has been expressed mathematically in various ways. One of the simplest is with the following equation

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n), \qquad 0 < \eta \leq 1 \tag{4.3}$$

where Δw_{kj} is the change that will be applied to the weight w_{kj} and η is a learning rate parameter. This particular formulation of Hebb's law is sometimes called the *activity product rule.* Repeated application of x_j leads to an *exponential growth* that finally drives w_{kj} to saturation [3]. The activity product rule described in equation 4.3 is an unsupervised rule. A supervised version of Hebb's law exists and is described by the following equation [104]

$$\Delta w_{kj}(n) = \eta(t_k(n) - y_k(n)) x_j(n), \qquad 0 < \eta \leq 1 \tag{4.4}$$

Where $t_k(n)$ denotes the neuron's target output. It is often called the *Widrow-Hoff* rule. Rumelhart *et. al.* [95], however, called it the *delta rule* because the weight change Δw_{kj} was proportional to the difference between the neuron output and the target output provided by a teacher.

4.3 Multilayer Feed-forward Networks

Perceptrons operate under the constraint that the input patterns are linearly separable [3, 98]. Whenever the input patterns are not linearly separable, the network needs to form an internal representation of the input to perform the necessary input-output mappings. This internal representation can be formed with one or more hidden layers. Multilayer feed-forward networks consist of a set of sensory units that constitute the *input layer*, one or more *hidden layers* of neurons, and an *output layer* of neurons. These networks are commonly referred to as *multilayer perceptrons* (MLPs) [3]. A multilayer perceptron has three distinctive characteristics:

1. The model of each neuron includes a *nonlinearity* at the output end. The nonlinearity must be *smooth* (i.e. differentiable everywhere). A commonly used form of nonlinearity that satisfies this requirement is a *sigmoidal nonlinearity* defined by the function:

$$y_j = \frac{1}{1 + e^{-v_j}} \tag{4.5}$$

where v_j is the net internal activity level of neuron j, and y_j is the output of the neuron.

2. The network contains one or more *hidden layers.* These enable the network to form an internal representation of the input.

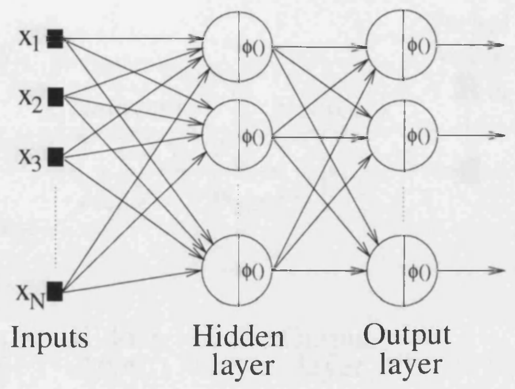


Figure 4.4: A Multilayer Perceptron with 1 hidden layer

3. There is a high degree of *connectivity* between the neurons of the network. Usually each neuron is connected to all of the neurons that are in the adjacent layers of the network.

The input signals propagate layer-by-layer through the network in a feed-forward direction. MLPs can be trained in a supervised manner with an algorithm known as the *error back-propagation algorithm* [95].

4.3.1 The Back-Propagation learning algorithm

The main concept of this algorithm is that the error of the output neurons of the network is propagated back through the network and there it is used to update all of the weights. Figure 4.4 is illustrating a MLP with a hidden layer and an output layer. Full interconnection of the neurons of the network is supposed throughout this section. The derivation of the back-propagation algorithm follows the one presented by Haykin (1994) [3] and is shown here because we use it in the next chapter to help us design some of our own filters. We first present a summary of the notation used in the presentation of the back-propagation algorithm[1].

Notation

- The indices j and k refer to different neurons in the network; with signals propagating through the network from left to right, neuron k lies in a layer to the right of neuron j, when neuron j is a hidden unit.

- The iteration n refers to the nth training pattern presented to the network.

[1]We wish to thank Simon Haykin for the adaptation of the notation and the derivation of the Back-propagation algorithm from his book [3]

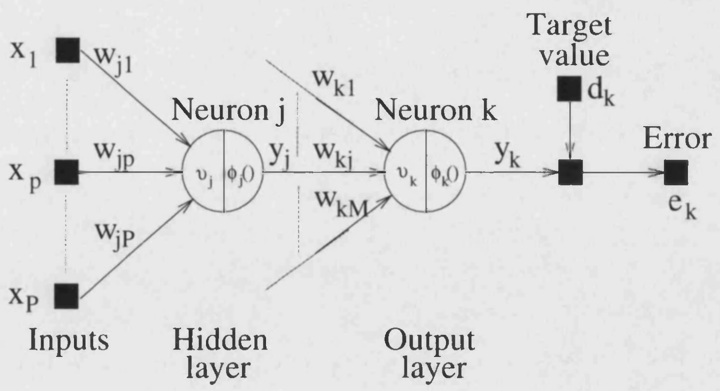


Figure 4.5: Signal-flow graph highlighting the details of output neuron k and hidden neuron j, adapted from Haykin (1994) [3]

- The symbol $e_j(n)$ refers to the error signal at the output of neuron j for iteration n.
- The symbol $d_j(n)$ refers to the desired response for neuron j and it is used to compute $e_j(n)$.
- The symbol $y_j(n)$ refers to the function signal appearing at the output of neuron j at iteration n.
- The symbol $w_{ji}(n)$ denotes the synaptic weight connecting the output of neuron i to the input of neuron j at iteration n. The correction applied to this weight at iteration n is denoted by $\Delta w_{ji}(n)$.
- The net internal activity level of neuron j at iteration n is denoted by $v_j(n)$; it constitutes the signal applied to the nonlinearity associated with neuron j.
- The activation function describing the input-output functional relationship of the nonlinearity associated with neuron j is denoted by $\phi_j(\cdot)$.
- The threshold applied to neuron j is denoted by θ_j; its effect is represented by a synapse of weight $w_{j0} = \theta_j$ connected to a fixed input equal to -1.
- The ith element of the input vector (pattern) is denoted by $x_i(n)$.
- The learning rate parameter is denoted by η.

Consider the signal-flow graph shown in figure 4.5, where the details of output neuron k for pattern n are highlighted. The inputs to neuron k are the outputs of all of the neurons in the previous layer $y_j(n)$. The *internal activity level* at the input of the non-linearity associated with neuron k is

$$v_k(n) = \sum_{j=0}^{M} w_{kj}(n) y_j(n) \tag{4.6}$$

where M is the total number of inputs excluding the threshold θ_k,which is represented by the synaptic weight w_{k0}. The associated input y_0 is fixed and equal to -1. The function signal at the output of neuron k at iteration n is

$$y_k(n) = \phi_k\big(v_k(n)\big) \tag{4.7}$$

The *error signal* at the output of neuron k at iteration n when neuron k is in the output layer is defined by

$$e_k(n) = d_k(n) - y_k(n) \tag{4.8}$$

The *instantaneous sum of squared errors* $E(n)$ is obtained by summing the squared errors of all of the neurons in the output layer

$$E(n) = \frac{1}{2} \sum_{k \epsilon C} e_k^2(n) \tag{4.9}$$

where C contains all of the neurons in the output layer. The *average squared error* E_{av} equals to the sum of $E(n)$ over all n normalized with respect to the set size N.

$$E_{av} = \frac{1}{\mathrm{N}} \sum_{n=1}^{\mathrm{N}} E(n) \tag{4.10}$$

The back-propagation algorithm updates each synaptic weight $w_{kj}(n)$ by applying to it the correction $\Delta w_{kj}(n)$ according to the delta rule [95]

$$\Delta w_{kj} = -\eta \frac{\partial E(n)}{\partial w_{kj}(n)} \tag{4.11}$$

where η is the *learning-rate parameter* of the back-propagation algorithm. The use of the minus sign accounts for *gradient descent* in weight space. We may express the instantaneous gradient $\partial E(n)/\partial w_{kj}(n)$ as follows:

$$\frac{\partial E(n)}{\partial w_{kj}(n)} = \frac{\partial E(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial w_{kj}(n)} \tag{4.12}$$

From equation 4.9 we get

$$\frac{\partial E(n)}{\partial e_k(n)} = e_k(n)$$

From equation 4.8 we get

$$\frac{\partial e_k(n)}{\partial y_k(n)} = -1$$

From equation 4.7 we get

$$\frac{\partial y_k(n)}{\partial v_k(n)} = \phi'_k\big(v_k(n)\big)$$

From equation 4.6 we get

$$\frac{\partial v_k(n)}{\partial w_{kj}(n)} = y_j(n)$$

So:

$$\frac{\partial E(n)}{\partial w_{kj}(n)} = -e_k(n)\phi'_k(v_k(n))y_k(n) \tag{4.13}$$

From equation 4.11 and equation 4.13 the weight update $\Delta w_{kj}(n)$ may be expressed as

$$\Delta w_{kj}(n) = \eta e_k(n)\phi'_k\big(v_k(n)\big)y_k(n)$$

We define the *local gradient* $\delta_k(n)$ by

$$\delta_k(n) = -\frac{\partial E(n)}{\partial e_k(n)}\frac{\partial e_k(n)}{\partial y_k(n)}\frac{\partial y_k(n)}{\partial v_k(n)} = e_k(n)\phi'_k\big(v_k(n)\big) \tag{4.14}$$

Therefore, the weight update $\Delta w_{kj}(n)$ may be expressed as

$$\Delta w_{kj}(n) = \eta\delta_k(n)y_k(n) \tag{4.15}$$

The derivation of the local gradient $\delta_k(n)$ is rather straightforward in the case that neuron k is an output neuron. We will derive the local gradient $\delta_j(n)$ for the case that neuron j is in a hidden layer, again with the help of figure 4.5, which depicts the signal-flow diagram for neuron j when this is in a hidden layer. In this case, the local gradient $\delta_j(n)$ may be redefined by

$$\begin{aligned}\delta_j(n) &= -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial E(n)}{\partial y_j(n)}\phi'_j\big(v_j(n)\big)\ , \qquad \text{where neuron } j \text{ is hidden}\end{aligned} \tag{4.16}$$

We may calculate the partial derivative $\partial E(n)/\partial y_j(n)$ as follows

$$E(n) = \frac{1}{2}\sum_{k\epsilon C} e_k^2(n)\ , \qquad \text{neuron } k \text{ is an output neuron}$$

Differentiating with respect to the output signal $y_j(n)$, we get

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{4.17}$$

However

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \phi_k\big(v_k(n)\big)\ , \qquad \text{neuron } k \text{ is an output neuron} \end{aligned}$$

Hence

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\phi'_k\big(v_k(n)\big) \tag{4.18}$$

Also:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{4.19}$$

because

$$v_k(n) = \sum_{j=0}^{M} w_{kj}(n) y_j(n)$$

where M is the total number of inputs (excluding the threshold) applied to neuron k. From equations 4.17, 4.18 and 4.19 we get

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j(n)} &= -\sum_k e_k(n)\phi'_k\big(v_k(n)\big)w_{kj}(n) \\ &= -\sum_k \delta_k(n) w_{kj}(n) \end{aligned} \tag{4.20}$$

Finally, from equations 4.16 and 4.20 the local gradient $\delta_j(n)$ for hidden neuron j is given by

$$\delta_j(n) = \phi'_j\big(v_j(n)\big) \sum_k \delta_k(n) w_{kj}(n)\ , \qquad \text{when neuron } j \text{ is hidden} \tag{4.21}$$

So to summarise, the weight update for all of the weights in the network is given by the generalised delta rule and is

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{4.22}$$

where η is the learning rate parameter, $y_i(n)$ is the input to neuron j and can be the output of neuron i in the previous layer or an input x_i, and $\delta_j(n)$ is the local gradient and is given by the equations

$$\delta_j(n) = e_j(n)\phi'_j\big(v_j(n)\big), \qquad \text{when neuron j is an output neuron} \tag{4.23}$$

$$\delta_j(n) = \phi'_j\big(v_j(n)\big) \sum_k \delta_k(n) w_{kj}(n)\ , \qquad \text{when neuron } j \text{ is a hidden neuron} \tag{4.24}$$

The training of the network occurs in two phases: During the first phase, an input pattern is presented propagated through the network and the output values are computed. In the second phase, these output values are compared to the target values and the local gradients are calculated for the output neurons using equation 4.23. Then their weights are updated (equation 4.22) and the local gradients are computed for the neurons in the previous layer using equation 4.24 and so on until all of the weights in the network are updated.

4.4 Summary

In this chapter we have introduced the basic principles of artificial neural networks. We briefly discussed two very widely used classes of neural networks, perceptrons and multilayer perceptrons. We also presented a learning algorithm for the perceptron based on the Hebbian learning law, and the error back-propagation algorithm for training multilayer perceptrons. In the following chapters we are going to derive and investigate the performance of the similarity suppression algorithm for designing pattern discrimination filters.

Chapter 5

Similarity Suppression filter design algorithm

5.1 Introduction

In the previous chapters we presented some background theory which is necessary for understanding our work and we reviewed some of the most well known filter design algorithms. In this chapter we present our similarity suppression (SS) algorithm for designing filters for optical pattern discrimination. We start with the derivation of the algorithm in section 5.2. The magnitude of the designed filters is analysed in section 5.3. In section 5.4 we compare the SS algorithm with relevant filter design algorithms, like the Gram-Schmidt orthogonalisation procedure, linear combination filters and synthetic discriminant functions and with the Hebbian learning law for training neural networks. Finally, in section 5.5 we expand the SS algorithm to design filters for 2 or more cascaded banks of correlators. In this chapter we examine the SS algorithm from a theoretical viewpoint. In the next chapter we present the computer simulations for this algorithm, which verify its performance. Before we start, we must point out that the algorithm presented here is the final product of several years of continuous changes and improvements, which were the result of computer simulations and long discussions.

5.2 Derivation of the similarity suppression algorithm

We will start (section 5.2.1) with the derivation of a slightly different algorithm from the one we use for the design of our filters. We call this, the similarity suppression (SS) orthogonalisation algorithm. It is described as a necessary precursor to the description of the algorithm which we use for the filter design (section 5.2.2), because it was the one we developed initially and the algorithm we now use was developed as an improvement on that. From this point forward, we will refer to the inner product of a pattern with itself as the auto-inner product and the inner product of a pattern and another pattern as a cross-inner product.

5.2.1 Development of the similarity suppression orthogonalisation algorithm

Our aim is to distinguish one pattern from another. This becomes difficult if the patterns are similar. Our aim then is to suppress the similarities, quantified by the inner product correlations, between *all* pairs of the known training patterns to be distinguished. However, if two patterns are similar but different from the other patterns in a group it is important not to lose the features which are common between the two patterns. These features allow each of them to be distinguished from the other members of the group. At the same time we want to make each of the two similar patterns less similar to each other to allow each of them to be distinguished from the other. This highlights the trade off that is necessary.

For simplicity just consider two patterns for now. We will generalise this to more patterns later. For just two patterns we would like to subtract the similar features of the two patterns from the first pattern, initially. The magnitude of the similar features is given by the inner product $\mathbf{s}_1 \cdot \mathbf{s}_2$ but this does not specify what the similar features are. Ideally we would like to subtract from pattern $\mathbf{s}_1$, the similar features multiplied by a weighting factor $\mathbf{s}_1 \cdot \mathbf{s}_2$, so that the similar features are removed at once.

$$\mathbf{g}_1 = \mathbf{s}_1 - (\mathbf{s}_1 \cdot \mathbf{s}_2)(normalised\ similar\ features) \tag{5.1}$$

We would also like to do this to the second pattern in a similar way

$$\mathbf{g}_2 = \mathbf{s}_2 - (\mathbf{s}_1 \cdot \mathbf{s}_2)(normalised\ similar\ features) \tag{5.2}$$

However, we do not know what the similar features are, so the best we can do is to subtract the whole of the second pattern from the first after weighting the second pattern by $s_1 \cdot s_2$, the amount of similarity.

$$g_1 = s_1 - (s_1 \cdot s_2)s_2 \tag{5.3}$$
$$g_2 = s_2 - (s_1 \cdot s_2)s_1 \qquad \text{likewise} \tag{5.4}$$

We can expect this subtraction to suppress the similarities between the two images, as similar regions in the two images will have approximately equal amplitudes and the result of the subtraction will locally be close to zero. However, this subtraction may or may not remove the similar features completely, since some features may be very similar, while others may only be slightly similar. In this case the most similar features will be suppressed. However, the only slightly similar features will be over compensated and in the worst case they may even be enhanced in magnitude although having a negative sign. The distinguishing features of each pattern which are not present in the other pattern will also be added (with negative sign) into the other pattern which is highly undesirable as we want to keep the distinguishing features of each pattern in that pattern's filter only.

Instead of making such a large change let us introduce a factor β'', less than one, giving

$$g_1 = s_1 - \beta''(s_1 \cdot s_2)s_2 \tag{5.5}$$
$$g_2 = s_2 - \beta''(s_1 \cdot s_2)s_1 \qquad \text{likewise} \tag{5.6}$$

If β'' is small enough we can be sure that all of the similar features are suppressed by a small amount and that only a small amount of the distinguishing features of one pattern are added into another pattern. If we now start afresh beginning with only the patterns g_1 and g_2, ignoring any patterns at earlier iterative steps in our algorithm and apply the same algorithm, we will suppress the similar features of patterns g_1 and g_2. However, since their similarities are mainly determined by (or inherited from) the similarities between the preceding set of patterns at the last iteration, this means that the original similarities are suppressed by a small amount on each iteration. Let us say that the new patterns are g'_1, and g'_2.

$$g'_1 = g_1 - \beta''(g_1 \cdot g_2)g_2 \tag{5.7}$$
$$g'_2 = g_2 - \beta''(g_1 \cdot g_2)g_1 \qquad \text{likewise} \tag{5.8}$$

Note that the original distinguishing features of s_1 and s_2 which are now partially present in the other pattern, g_2 and g_1 respectively, will give a negative contribution to $g_1 \cdot g_2$ since they now represent a similar feature, having some presence in each g pattern, but with an opposite sign. If the originally similar features had been completely removed by the first use of the algorithm, then the second use of the algorithm will just about remove the originally distinguishing features which had been added (with a negative sign) to the other pattern. This occurs because this algorithm suppresses similar features whether they have the same sign or not.

So, by repetitive application of the algorithm similarities are suppressed until the patterns are effectively orthogonalised, assuming β'' is chosen small enough to ensure gentle monotonic convergence and assuming that the iterations are allowed to continue to convergence. In other words, the correlation between the final pair of patterns is zero. In order to extend this to deal with more than two patterns at each iteration step we simply subtract from each pattern all of the *other* patterns, each weighted by β'' times its similarity with the pattern itself:

$$\mathbf{g}'_j = \mathbf{g}_j - \beta''(\mathbf{g}_j \cdot \mathbf{g}_1)\mathbf{g}_1 - \beta''(\mathbf{g}_j \cdot \mathbf{g}_2)\mathbf{g}_2 \cdots - \beta''(\mathbf{g}_j \cdot \mathbf{g}_M)\mathbf{g}_M \tag{5.9}$$

This is repeated in the same way for each of the $j = 1$ to M training patterns and a set of new patterns, which are denoted by the symbols $\mathbf{g}'_j, j = 1 \ldots M$, are obtained. This can be written in a more compact form

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{g}_k^{(i-1)} \right\} \mathbf{g}_k^{(i-1)} \tag{5.10}$$

where the superscript, i, denotes the iteration number. The tilde symbol over the $\mathbf{g}$ indicates that the pattern has yet to be normalised, as described later. In equation 5.10 we subtract all of the other patterns in the $\mathbf{g}^{(i-1)}$ set at the $(i-1)^{th}$ iteration, except for the pattern being processed itself, from each of the $\mathbf{g}^{(i-1)}$ patterns.

The danger of subtracting so many patterns from one pattern is that the pattern will be dominated by the effect of the subtractions and lose its own identity so we need to ensure that β'' is kept sufficiently small. Even so, the subtraction of a lot of weak pattern vectors from one pattern vector is likely to diminish its strength, or magnitude, as similarities are gradually removed. It may not matter if all of the pattern vectors are diminished by the same amount, but this is unlikely to be the case, resulting in a variation in the magnitude (in terms of the vector length or the Euclidean norm) from pattern to pattern. This is highly undesirable as it would lead to a bias or preference for some patterns if the $\mathbf{g}$ patterns were used as filters in an inner product correlator. That is to say that if white noise or completely random patterns were input, the system would indicate that more of the stronger patterns were present than the weaker ones, whereas all should be equally likely. So all of the patterns need to be normalised to have the same strength (or length), say unity. It is not sufficient to allow the algorithm equation, (5.10), to converge and then to normalise the final $\mathbf{g}$ patterns because if the preceding $\mathbf{g}$ and even $\mathbf{s}$ patterns had not been normalised the algorithm itself would not have been even handed towards each pattern so some patterns would have had an undue effect on their term in the subtractions. Therefore, in order to treat each pattern equally, throughout the process, we need to normalise the s patterns at the start and then renormalise the **g** patterns at the end of each iteration. Our aim is to produce a set of orthogonal patterns of equal strength, therefore, we need to normalise the Euclidean norm or length of the **g** vectors in the set at each iteration to be a constant:

$$\begin{aligned} \mathbf{g}_j^{(i)} &= \tilde{\mathbf{g}}_j^{(i)} \frac{\|\mathbf{g}_j^{(i-1)}\|}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \\ &= \frac{\tilde{\mathbf{g}}_j^{(i)}}{\|\tilde{\mathbf{g}}_j^{(i)}\|}, \qquad \text{assuming } \|\mathbf{g}_j^{(i-1)}\| = 1 \end{aligned} \tag{5.11}$$

So to summarise, the SS orthogonalisation algorithm is described by the following two equations:

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k\neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{g}_k^{(i-1)} \right\} \mathbf{g}_k^{(i-1)} \qquad \forall j \tag{5.12}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\|\mathbf{g}_j^{(i-1)}\|}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \qquad \forall j \tag{5.13}$$

(5.14)

5.2.2 Development of the similarity suppression cross - orthogonalisation algorithm

The SS orthogonalisation algorithm described in the last section tends to orthogonalise the original patterns and create a set of g patterns which are orthogonal to each other, without bias to any one pattern. So the final patterns would be good at recognising and discriminating the presence of any of the final patterns at the input.

We have assumed that as we were only making a series of small subtractions and then amplifying the whole by renormalisation, that we have retained all of the distinctive features of the original corresponding patterns, so that the final patterns, when used as filters, would be good at recognising the input patterns. Unfortunately, this is not necessarily the case. In N-dimensional space (N is the number of pixels in each image), which will be called pattern-space from now on, each pattern defines one point, which shows the pattern's position in the pattern-space. While the patterns are being orthogonalised, as the algorithm changes the individual pixel values, it is possible for all of the patterns to drift slowly away from their original positions, because no knowledge of the original positions is explicitly used in the algorithm to anchor them or to pull them back. At each iteration we only use the pattern set available at that iteration. So while the final patterns may no doubt be good at recognising themselves in the input, they may not be good at recognising the original patterns in the input. For example, the final $\mathbf{g}_1$ pattern may still have cross correlations with the $\mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \ldots$ patterns comparable to $\mathbf{g}_1 \cdot \mathbf{s}_1$, which is the opposite of the original aim.

It is better to develop the algorithm further by defining what we want and designing the algorithm to make it happen. *We would like each final filter* $\mathbf{g}$ *pattern to have as low an inner product correlation as possible with the original* $\mathbf{s}$ *patterns, except for the original* $\mathbf{s}$ *pattern from which it was derived, with which it must have a constant high inner product correlation.* This can be achieved by ensuring that these two types of inner product which we call respectively, the cross-inner product and the auto-inner product between the two sets, are incorporated into the algorithm itself. The aim will, therefore, be not to orthogonalise the $\mathbf{g}$ patterns as compared to other members of their own set, but to cross-orthogonalise the set of $\mathbf{g}$ patterns with respect to the set of $\mathbf{s}$ patterns. In this case the filters, $\mathbf{g}$, are said to be *mutually orthogonal* to the training patterns, $\mathbf{s}$.

This can be achieved by replacing some of the $\mathbf{g}$ patterns on the right of equation 5.9 by their corresponding original $\mathbf{s}$ patterns to give

$$\mathbf{g}'_j = \mathbf{g}_j - \beta''(\mathbf{g}_j \cdot \mathbf{s}_1)\mathbf{s}_1 - \beta''(\mathbf{g}_j \cdot \mathbf{s}_2)\mathbf{s}_2 \cdots - \beta''(\mathbf{g}_j \cdot \mathbf{s}_M)\mathbf{s}_M \tag{5.15}$$

Written in a more compact form it becomes

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.16}$$

where the superscript i denotes the iteration number. In equation 5.16 we subtract all of the other patterns in the original training set, except for the pattern being processed itself, from each of the $\mathbf{g}^{(i-1)}$ patterns. Here, as before, we take a set of patterns and derive from them another set of patterns in a one to one correspondence. At each iteration a number of terms are subtracted from one pattern which is then renormalised and this is repeated in the next iteration and so on. However, the cross-inner product correlations between the *two* sets quantify the similarities between the $\mathbf{g}_j$ pattern and all of the *other* original $\mathbf{s}_k$ patterns. Then we reduce those similarities by subtracting from the $\mathbf{g}_j$ pattern, the $\mathbf{s}_k$ pattern with which it was being compared, weighted by the inner product $\mathbf{g}_j \cdot \mathbf{s}_k$ in a similar manner as in our earlier derivation, (section 5.2.1), also including a small convergence term β''. This ties the new $\mathbf{g}_j$ patterns back to all of the original patterns (apart from the one from which it was derived) and forces the cross-correlations between the two sets to be reduced. In addition we need to tie the $\mathbf{g}_j$ pattern back to the $\mathbf{s}_j$ pattern from which it was derived and this is carried out by modifying the normalisation step to keep their inner product constant, i.e. $\mathbf{s}_j \cdot \mathbf{g}_j = \text{const}$ which can be brought about by the normalisation equation:

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.17}$$

This can be confirmed by finding the inner product of each side with $\mathbf{s}_j$. Assuming that the original training patterns are normalised to a constant value denoted henceforth by P the previous equation can be rewritten in the following manner, since $\mathbf{g}_j^{(1)} = \mathbf{s}_j$ and consequently $\mathbf{g}_j^{(i)} \cdot \mathbf{s}_j = P, \forall i$:

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{P}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.18}$$

In most cases, P is equal to 1 but this is not necessary for the convergence of the algorithm.

Returning to the full algorithm we now describe it with the following two equations:

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.19}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.20}$$

5.2.3 Advanced algorithm with improved convergence parameters

Computer simulations (section 6.2) have shown us that the algorithm is sensitive to small changes in the convergence factor β''. Large β'' values resulted in oscillations, while very small β'' values ensured convergence but the algorithm needed many iterations to converge to the desired values. We found that the algorithm converged faster and to a lower minimum if the value of β'' was changed as the algorithm was converging. Larger values of β'' at the beginning allowed faster initial convergence, while smaller values later assured a finer search for the minimum and avoided oscillations. In addition, it seems to make more sense to force the larger cross-inner products down more strongly as in MICE filters, rather than the MACE, where the largest side-lobes are forced to decrease. We can achieve both of these goals by making the β'' factor of each term depend on the difference between the inner product in that term and the desired value, in this case zero.

In order to do that β'' is calculated using the following expression (which has to be put INSIDE the summation)

$$\beta'' = \beta' \left| \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right| \tag{5.21}$$

where β' is the new convergence constant. Computer simulations have shown, that setting $\beta' < 1$ ensures that β'' does not become too large and that the algorithm does not become unstable. When β'' is substituted into the algorithm (equation 5.19) one obtains the final version of the algorithm shown in equations 5.22 and 5.23

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \left| \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right| \left((\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k) \right) \right\} \mathbf{s}_k \tag{5.22}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.23}$$

Equation 5.22 can also be written in the following form

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \pm \left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right)^2 \right\} \mathbf{s}_k \tag{5.24}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.25}$$

where in the $\pm$ sign the plus sign is used when $\left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k\right) > 0$ and the minus sign is used when $\left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k\right) < 0$. Equation 5.24 is very similar to equation 5.19 with the only differences being that the term in the brackets is now squared while preserving the sign. This version of the algorithm performed considerably better in our computer simulations. This can be explained if one considers that the term that is now squared is the difference between the current value of the cross-inner product $\mathbf{g}_j \cdot \mathbf{s}_k$ and the desired value (which is equal to zero). This difference is then used as the weight for the subtraction of the $\mathbf{s}_k$ pattern. By squaring it we emphasise the subtractions of the patterns that are most similar to the $\mathbf{g}_j$ pattern, thus forcing the largest cross-inner products, in particular, to decrease. A cubic power or higher power (while preserving the sign) would make the strength of convergence depend more strongly on the size of the cross-inner product. Computer simulations however, showed that higher powers tended to make the algorithm unstable, forced us to use a smaller convergence parameter, β' and overall did not produce a better result.

One final issue that we would like to address, is the size of β'. The size of the sum, in equation 5.24, is proportional to the total number of patterns, M and to the size of the square of the inner products, P^2. Therefore, the convergence factor has to be of the order of $\frac{1}{P^2M}$. So the algorithm (equations 5.24 and 5.25) can also be written in the form

$$\beta' = \frac{\beta}{P^2M} \tag{5.26}$$

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \frac{\beta}{P^2M}\sum_{\substack{k=1\\k\neq j}}^{M}\left\{\pm\left(\mathbf{g}_j^{(i-1)}\cdot\mathbf{s}_k\right)^2\right\}\mathbf{s}_k \tag{5.27}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)}\frac{\mathbf{g}_j^{(i-1)}\cdot\mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)}\cdot\mathbf{s}_j} \tag{5.28}$$

where now the new, final β is not related to the size of the inner products, which for binary, bipolar patterns is directly proportional to the size of the training patterns. In addition, β is not related to the number of patterns in the training set any more and it takes small values around unity.

5.3 Analysis of the normalisation step

It is logical to assume that the magnitudes of the g patterns, defined by the squared Euclidean norm shown in equation 3.2, which we rewrite here,

$$\text{Euclidean norm:} \qquad ||\mathbf{s}||_2 = \left(\sum_{i=1}^{N} s_i^2\right)^{1/2} \tag{5.29}$$

will change since we are continuously subtracting other patterns from them. In addition, any change in the g patterns will, as a result, change the magnitudes of the inner products between them and the corresponding s patterns they represent. However, we would like to keep these inner products as stable and high as possible, because that would enable us to correctly recognise a pattern by setting the appropriate threshold or just by choosing the highest inner product. To see the effect of equation 5.16 on the magnitudes of the g patterns we can rewrite it for the simple case of just two patterns $\mathbf{s}_1$ and $\mathbf{s}_2$.

$$\mathbf{g}_2^{(i)} = \mathbf{g}_2^{(i-1)} - \beta''(\mathbf{g}_2^{(i-1)}\cdot\mathbf{s}_1)\mathbf{s}_1 \tag{5.30}$$

or in matrix notation

$$\begin{aligned} \mathbf{g}_2^{(i)} &= \mathbf{g}_2^{(i-1)} - \beta''\mathbf{g}_2^{(i-1)T}\mathbf{s}_1\mathbf{s}_1 \\ \Rightarrow \mathbf{g}_2^{(i)} &= \mathbf{g}_2^{(i-1)} - \beta''\mathbf{s}_1\mathbf{s}_1^T\mathbf{g}_2^{(i-1)} \quad \text{since } \mathbf{s}^T\mathbf{g} = \mathbf{g}^T\mathbf{s} = \textit{scalar} \\ \Rightarrow \mathbf{g}_2^{(i)} &= (\mathbf{I} - \beta''\mathbf{s}_1\mathbf{s}_1^T)\mathbf{g}_2^{(i-1)} \end{aligned} \tag{5.31}$$

We can calculate the squared Euclidean norm of the $\mathbf{g}_2$ pattern in the same way as [70] to see whether it increases or decreases or remains constant when equation 5.30 is used iteratively.

$$\begin{aligned}
||\mathbf{g}_2^{(i)}||^2 &= \mathbf{g}_2^{(i)T}\mathbf{g}_2^{(i)} = \mathbf{g}_2^{(i-1)T}(\mathbf{I} - \beta''\mathbf{s}_1\mathbf{s}_1^T)^T(\mathbf{I} - \beta''\mathbf{s}_1\mathbf{s}_1^T)\mathbf{g}_2^{(i-1)} \\
&= (\mathbf{g}_2^{(i-1)T} - \beta''\mathbf{g}_2^{(i-1)T}\mathbf{s}_1\mathbf{s}_1^T)(\mathbf{g}_2^{(i-1)} - \beta''\mathbf{s}_1\mathbf{s}_1^T\mathbf{g}_2^{(i-1)}) \\
&= \mathbf{g}_2^{(i-1)T}\mathbf{g}_2^{(i-1)} - \beta''\mathbf{g}_2^{(i-1)T}\mathbf{s}_1\mathbf{s}_1^T\mathbf{g}_2^{(i-1)} \\
&\quad - \beta''\mathbf{g}_2^{(i-1)T}\mathbf{s}_1\mathbf{s}_1^T\mathbf{g}_2^{(i-1)} + \beta''^2\mathbf{g}_2^{(i-1)T}\mathbf{s}_1\mathbf{s}_1^T\mathbf{s}_1\mathbf{s}_1^T\mathbf{g}_2^{(i-1)} \\
&= ||\mathbf{g}_2^{(i-1)}||^2 - 2\beta''(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2 + \beta''^2||\mathbf{s}_1||^2(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2 \qquad (5.32) \\
\Rightarrow ||\mathbf{g}_2^{(i)}||^2 - ||\mathbf{g}_2^{(i-1)}||^2 &= \beta''^2||\mathbf{s}_1||^2(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2 - 2\beta''(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2 \\
&= \beta''(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2(\beta''||s_1||^2 - 2)
\end{aligned}$$

Equation 5.32 expresses the slope of the $\mathbf{g}_2$ squared Euclidean norm, $||\mathbf{g}_2||^2$. If the right hand side of equation 5.32 is positive then the magnitude of $\mathbf{g}_2$ will increase. If it is negative the norm will decrease and if it is equal to zero the magnitude of $\mathbf{g}_2$ will remain constant. The sign of the slope depends on the sign of the expression $(\beta''||s_1||^2 - 2)$, which depends on the value of β'', since $||s_1||^2$ is constant, and for binary patterns is equal to N.

- If $\beta'' = \frac{2}{||\mathbf{s}_1||^2}$ then the slope is zero and the norm of $\mathbf{g}_2$ is constant.
- If $\beta'' < \frac{2}{||\mathbf{s}_1||^2}$ then the slope is negative and the norm of $\mathbf{g}_2$ decreases. Eventually, as the inner product $(\mathbf{g}_2^{(i-1)T}\mathbf{s}_1)^2$ decreases, the right hand side of equation 5.32 tends to zero. So after a number of iterations the norm of $\mathbf{g}_2$ will stabilise to a very low value. However, as $||\mathbf{g}_2||^2$ decreases, the auto-inner product $\mathbf{g}_2 \cdot \mathbf{s}_2$ will decrease as well and this not always the most desirable result.
- If $\beta'' > \frac{2}{||\mathbf{s}_1||^2}$ then the slope is positive and $||\mathbf{g}_2||^2$ will increase. This happens because at each iteration large values are subtracted from the $\mathbf{g}_2$ pixels and their sign is reversed but their absolute values increase and, therefore, $||\mathbf{g}_2||^2$ increases. This however, results in the increase of the cross-inner product $\mathbf{g}_2 \cdot \mathbf{s}_1$, which again may not be the most desirable result.

In section 5.2.3 we chose to use $\beta'' = 1/PM$, which corresponds to $\beta' = 1/P^2M$ for the improved algorithm, as our convergence parameter. This β'' is smaller than $2/||\mathbf{s}||$ ($P = ||\mathbf{s}||$) and according to the previous analyses the magnitudes of the $\mathbf{g}$ filters will decrease. That is the reason why we chose to normalise the $\mathbf{g}$ patterns in such a way so that all of the auto-inner products $\mathbf{g}_j \cdot \mathbf{s}_j$ remained constant and equal to the initial value of the auto-inner products of the s patterns. To do that we used equation 5.17 which we rewrite here

$$\mathbf{g}_j^{\prime(i)} = \mathbf{g}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\mathbf{g}_j^{(i)} \cdot \mathbf{s}_j} \tag{5.33}$$

at each iteration after equation 5.16. By using equation 5.33 at each iteration of the algorithm we force $\mathbf{g}_j \cdot \mathbf{s}_j = \mathbf{s}_j \cdot \mathbf{s}_j, \forall j$. This equation has an effect on the magnitudes of the g patterns as well. Their magnitudes now increase as long as the right hand side of equation 5.32 is not equal to zero. This happens because of equation 5.33 which amplifies them after every iteration. As the algorithm converges and $\mathbf{g}_2 \cdot \mathbf{s}_1 \rightarrow 0$, for two patterns, $||\mathbf{g}_2||$ stabilises at a higher level than what it was before the training. This analysis can be extended to an $M > 2$ number of patterns with very similar results[1]. It is logical to predict that amongst several filters, the one whose corresponding training pattern is most similar to other training patterns will have at the end of the training the largest magnitude. This will happen because the algorithm affects filters that are derived from similar patterns more and it does not induce large changes to the filters that correspond to patterns that are very different to each other.

One last thing that we would like to point out here is that there is a drawback in using this normalisation. In section 5.2.1 we said that we would prefer all of the filters g to be of equal magnitude, so that no bias for some patterns would exist. Now, however, we have shown that the magnitudes, $||\mathbf{g}||$, of some of the filters will increase, and, therefore, a bias will exist. This makes the rejection of unknown random patterns difficult because they may give a higher correlation peak with a high magnitude filter. On the other hand this normalisation enables us to correctly discriminate amongst the known patterns by setting the appropriate threshold, or just by choosing the highest peak. We could use a different normalisation equation and normalise the magnitudes $||\mathbf{g}||$ of the g filters themselves, for example equation 5.11, which we rewrite here

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{||\mathbf{g}_j^{(i-1)}||}{||\tilde{\mathbf{g}}_j^{(i)}||} \tag{5.34}$$

This normalisation would ensure that all of the filters would be of equal magnitude, and that an unknown random pattern would give approximately equal correlation peaks with all of them. The choice of which normalisation to use depends on the task at hand. If we know that all of the possible input patterns belong in a specified set and we want to discriminate among them, then the first normalisation described in equation 5.33 is preferable. If we want to detect the presence of an

[1]See appendix B for the mathematical analysis

object in the input, or if there is a chance that a pattern which does not belong to our set and must be rejected is present in the input, then it is important to have unbiased filters and the second normalisation described in equation 5.34 may be preferable.

5.4 Comparison with other filter design algorithms

In the previous sections we derived the SS orthogonalisation and the SS cross orthogonalisation algorithms. In this section we are going to compare them from a theoretical viewpoint with some relevant filter design techniques. We did these comparisons to find out how our algorithms relate to other filter design techniques, where they differ and their advantages and disadvantages. This knowledge can help one decide when to use our algorithms to design filters. In addition it has helped us improve our algorithms by borrowing ideas from similar techniques and applying them to our work. We will start by comparing the SS orthogonalisation algorithm to the Gram-Schmidt orthogonalisation procedure. We will then compare the SS algorithm with the Linear Combination Filters developed by Caulfield and Maloney [18]. Finally, we present the comparison between the SS algorithm and the Hebbian learning law for training single layer neural networks.

5.4.1 Comparison of the Gram-Schmidt orthogonalisation procedure with the similarity suppression orthogonalisation algorithm

The Gram-Schmidt orthogonalisation procedure was described in section 3.2 and the orthogonalisation equations are rewritten here

$$\mathbf{u}_1 = \frac{\mathbf{s}_1}{||\mathbf{s}_1||_2} \tag{5.35}$$

$$\tilde{\mathbf{u}}_{k+1} = \left(\prod_{j=1}^{k} (\mathbf{I} - \mathbf{u}_j \mathbf{u}_j^T) \right) \mathbf{s}_{k+1}, \tag{5.36}$$

$$\mathbf{u}_{k+1} = \frac{\tilde{\mathbf{u}}_{k+1}}{||\tilde{\mathbf{u}}_{k+1}||_2} \tag{5.37}$$

In order to compare our algorithm with it, we note that the vectors $\mathbf{u}_1, ..., \mathbf{u}_k$, are already orthonormal in the iterative equations 5.36 so that the cross-terms in the product $\prod_{j=1}^{k}(\mathbf{I} - \mathbf{u}_j^T\mathbf{u}_j)$ disappear, therefore, we can write equation 5.36 in the form [70]

$$\begin{aligned}\mathbf{u}_{k+1} &= \left(\mathbf{I} - \sum_{j=1}^{k}\mathbf{u}_j\mathbf{u}_j^T\right)\mathbf{x}_{k+1} \\ &= \mathbf{x}_{k+1} - \sum_{j=1}^{k}(\mathbf{u}_j^T\mathbf{x}_{k+1})\mathbf{u}_j\end{aligned} \tag{5.38}$$

Equations 5.10 and 5.11, which we rewrite here,

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k\neq j}}^{M} \left\{\mathbf{g}_j^{(i-1)} \cdot \mathbf{g}_k^{(i-1)}\right\}\mathbf{g}_k^{(i-1)} \tag{5.39}$$

$$\mathbf{g}_j^{(i)} = \frac{\tilde{\mathbf{g}}_j^{(i)}}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \tag{5.40}$$

$$\tag{5.41}$$

are very similar to equations 5.38 and 5.35 defining the Gram-Schmidt orthonormalisation which is not surprising as our algorithm also leads to orthogonalisation of the original patterns. However, our final set of orthogonal patterns is different to the one obtained with the Gram-Schmidt procedure. In fact the basic SS algorithm is a symmetrical and iterative version of the Gram-Schmidt process. The main differences between the two are that:

i. In the SS algorithm all of the patterns are changed by a small amount in each iteration, treating each pattern in the same equal handed way. However, in the Gram-Schmidt procedure, the first pattern is not changed at all and all of the others are changed to become orthogonal to it. The final result is, therefore, highly dependent on the order in which the patterns are chosen as first, second, and so on. A different presentation order would lead to different set of orthogonal patterns. The order is unimportant for the SS algorithm as the iterative equation does not use any patterns already modified earlier in the current iteration but only patterns from the previous iteration which are fixed throughout.

ii. The SS algorithm has a convergence factor, β'' while the Gram-Schmidt procedure has not.

If the orthogonal patterns resulting from the Gram-Schmidt orthogonalisation procedure are used to recognise and distinguish the presence of *themselves* in the input, the system would work well as the patterns are orthogonal. However, in many cases it is not possible to choose the patterns to be recognised to be members of a particular orthogonal set.

If the patterns resulting from the Gram-Schmidt orthogonalisation process were used as filters to try to distinguish the original input patterns, the later patterns have had so much removed from them that it is likely that their original own distinctive features have been obscured to such a degree that they may be considered to have been lost. With the Gram-Schmidt algorithm, in N dimensional pattern space, the first pattern remains at the same position, while subsequent patterns move further and further from their original position despite being finally resolved onto an orthogonal axis. This means that the inner products between the final and the corresponding original patterns are likely to be smaller for patterns which were towards the end of the presentation order during orthogonalisation, resulting in a bias towards the first patterns in the presentation order.

Moreover, in the Gram-Schmidt procedure the first patterns still retain the original similarities that they had with most of the other patterns so when used as filters they will register a large output when any of the other similar original patterns are input leading to incorrect discrimination.

5.4.2 The relationship between the SS cross orthogonalisation algorithm and Caulfield's and Maloney's Linear Combination Filters

The SS cross orthogonalisation algorithm is described by equation 5.16 which we rewrite here:

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.42}$$

there is also a normalisation step (equation 5.20) which is not necessary for our analysis here and it is omitted for the sake of simplicity as is the square in equation 5.27. Equation 5.42 shows that at every iteration all of the training patterns are subtracted from each of the filters, with different weights. We can consider that the training patterns that correspond to each of the filters (when $k = j$ in equation 5.42) are subtracted from them with their weights set to zero. Lets consider what happens to an individual filter throughout the training. At each iteration, all of the training images are subtracted from it, each with a different weight. After all of the iterations a total amount of each of the training images has been subtracted from it. This total amount is equal to the sum of all of the individual weights which were used for the subtraction of each training image during the training.

The same thing happens to all of the filters. Therefore, they can be given by the following equations:

$$\begin{aligned} C_{11}\mathbf{s}_1 + C_{12}\mathbf{s}_2 + \cdots + C_{1M}\mathbf{s}_M &= \mathbf{g}_1 \\ C_{21}\mathbf{s}_1 + C_{22}\mathbf{s}_2 + \cdots + C_{2M}\mathbf{s}_M &= \mathbf{g}_2 \\ &\vdots \\ C_{M1}\mathbf{s}_1 + C_{M2}\mathbf{s}_2 + \cdots + C_{MM}\mathbf{s}_M &= \mathbf{g}_M \end{aligned} \tag{5.43}$$

where each of the coefficients $C_{11}, C_{12}, \ldots, C_{MM}$ is equal to the sum of all of the individual weights that were used for the subtraction of each of the training images during the training. These coefficients are all negative, except from $C_{jj}, \forall j$ which are equal to zero. It is obvious that if the coefficients $C_{11}, C_{12}, \ldots, C_{MM}$ can be calculated then the final filters can be created without the need for an iterative procedure. The aim of the cross orthogonalisation algorithm is to design the filters in such a way so that each of them has an inner product equal to one (or some other constant) with the corresponding training pattern and equal to zero with every other pattern. These conditions can be expressed by the following set of equations:

$$\begin{aligned} \mathbf{s}_i \cdot \mathbf{g}_j &= 1 \qquad \text{if } i = j \\ \mathbf{s}_i \cdot \mathbf{g}_j &= 0 \qquad \text{if } i \neq j, \forall i, j \end{aligned} \tag{5.44}$$

The previous set of equations are written in a matrix form as follows:

$$\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_M \end{pmatrix} \cdot \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_M \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{5.45}$$

$$\Rightarrow \begin{pmatrix} \mathbf{s}_1 \cdot \mathbf{g}_1 & \mathbf{s}_1 \cdot \mathbf{g}_2 & \cdots & \mathbf{s}_1 \cdot \mathbf{g}_M \\ \mathbf{s}_2 \cdot \mathbf{g}_1 & \mathbf{s}_2 \cdot \mathbf{g}_2 & \cdots & \mathbf{s}_2 \cdot \mathbf{g}_M \\ & \vdots & & \\ \mathbf{s}_M \cdot \mathbf{g}_1 & \mathbf{s}_M \cdot \mathbf{g}_2 & \cdots & \mathbf{s}_M \cdot \mathbf{g}_M \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{5.46}$$

or,

$$\mathbf{S}\mathbf{G}^T = \mathbf{I} \tag{5.47}$$

where $\mathbf{S}$ is a Mxl vector whose elements are the training patterns $\mathbf{s}$, $\mathbf{G}$ is a Mxl vector whose elements are the filters $\mathbf{g}$ and $\mathbf{I}$ is an MxM identity matrix. Equations 5.43 can be written in a matrix form as follows:

$$\begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ & \vdots & & \\ C_{M1} & C_{M2} & \cdots & C_{MM} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_M \end{pmatrix} = \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_M \end{pmatrix} \tag{5.48}$$

or,

$$\mathbf{CS} = \mathbf{G} \tag{5.49}$$

where $\mathbf{C}$ is an MxM matrix each element of which is the coefficient C_{ij}. From equation 5.49

$$\mathbf{G}^T = \mathbf{S}^T\mathbf{C}^T \tag{5.50}$$

and substituting equation 5.50 into equation 5.47

$$\mathbf{S}\mathbf{S}^T\mathbf{C}^T = \mathbf{I} \tag{5.51}$$

$$\Rightarrow \mathbf{R}\mathbf{C}^T = \mathbf{I} \tag{5.52}$$

$$\Rightarrow \mathbf{C}^T = \mathbf{R}^{-1}\mathbf{I} \tag{5.53}$$

$$\Rightarrow \mathbf{C} = \mathbf{R}^{-1T} \tag{5.54}$$

where $\mathbf{R}$ is an MxM matrix each element of which, m_{ij}, is equal to the inner product between the training patterns $\mathbf{s}_i$ and $\mathbf{s}_j$. So, the coefficients C_{ij} can be calculated from equation 5.54 as long as matrix $\mathbf{R}$ can be inverted. In that case the final filters can be calculated directly from equation 5.49 by substituting the coefficients matrix $\mathbf{C}$ from equation 5.54

$$\mathbf{G} = \mathbf{R}^{-1T}\mathbf{S} \tag{5.55}$$

As we saw in chapter 3, Caulfield and Maloney [18] calculated their Linear Combination Filters in two steps. The first step was to calculate the vector inner product matrix, $\mathbf{R}$, of the input patterns. This matrix had each of it's elements r_{ij} equal to the inner product between the training patterns $\mathbf{s}_i$ and $\mathbf{s}_j$

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1M} \\ r_{21} & r_{22} & \cdots & r_{2M} \\ & \vdots & & \\ r_{M1} & r_{M2} & \cdots & r_{MM} \end{pmatrix} \tag{5.56}$$

where $r_{ij} = \mathbf{s}_i \cdot \mathbf{s}_j$.

In the second step they formed linear combinations of the responses $r_{ij}s$. Using these linear combinations, the final response when testing pattern $\mathbf{s}_i$ for its identity to $\mathbf{s}_k$ would be

$$F_{ik} = r_{ik} + \sum_{l \neq k} C_{kl} r_{il} \tag{5.57}$$

They imposed the constraint that F_{ik} had to be zero unless $i = k$ and nonzero if $i = k$, i.e.,

$$F_{ik} = F_{kk}\delta_{ik} \tag{5.58}$$

Equations 5.57 and 5.58 were formulated as matrix equations [74, 13] leading to the general SDF described by equation 3.18 which we rewrite here

$$\begin{aligned}\mathbf{R}\mathbf{a}_i &= \mathbf{d}_i \\ \Rightarrow \mathbf{a}_i &= \mathbf{R}^{-1}\mathbf{d}_i\end{aligned} \tag{5.59}$$

Equation 5.54 is in essence the same as equation 5.59. We have calculated the coefficients for the M filters, while equation 5.59 calculates the coefficients for a single filter. In addition, the matrix with the desired values in our case is the identity matrix, while equation 5.59 is more general. The vector-inner product matrix $\mathbf{R}$ is transposed as well as inverted in our equation (5.54) because we have defined the coefficient vector for each of the filters as a $1 \times M$ vector while in equation 5.59, $\mathbf{a}_i$ is a $M \times 1$ vector. So in effect we see that the SS cross orthogonalisation algorithm should finally converge to the solution which is obtained using general synthetic discriminant functions or Caulfield's and Maloney's method. The main difference between our algorithm and the two methods, is that our algorithm is iterative. The first question that automatically arises is whether the SS algorithm converges to exactly the same solution as the other two methods. We provide an answer to this question in the next chapter using computer simulations. The advanced form of the SS cross orthogonalisation algorithm described by equation 5.27 has the subtraction weight squared. This square does not affect the previous result, as it can be included in the coefficients C_{ij} without any change in the subsequent analysis.

5.4.3 Equivalence between a bank of correlators and a single layer of neurons

In this section we explain the equivalence between a bank of correlators and a single layer of neurons to pave the way for the comparison between the SS cross orthogonalisation algorithm and the Hebbian learning rule presented in the next section. We will follow the analysis presented in [4].

For an input pattern s of size N and a filter $\mathbf{g}$ also of size N the value of the central peak of the correlation at the output plane of a correlator is equal to their inner product which can be written

$$y_C = \sum_{i=1}^{N} s_i g_i \tag{5.60}$$

The output of a single neuron using neural network notation, where x_i denotes the input i and w_i the weight of the connection between the neuron and that input,

$\phi(\cdot)$ is the activation function (usually nonlinear) and θ is the threshold, is [3]:

$$y_N = \phi(v_N) = \phi(\sum_{i=1}^{N} x_i w_i - \theta) \tag{5.61}$$

Using our notation, equation 5.61 is written

$$y_N = \phi(v_N) = \phi(\sum_{i=1}^{N} s_i g_i - \theta) \tag{5.62}$$

Equations 5.60 and 5.62 are very similar with the only difference being the threshold and the activation function in equation 5.62. If the threshold θ is set to zero and we use the identity function $\phi(x) = x$ as our activation function, then equation 5.62 becomes identical to 5.60. On the other hand, we can include a threshold in equation 5.60 by increasing the size (the number of pixels) of the filter g and the input s and filling the remaining pixels of the filter with a constant background of the appropriate value. The inner product will then be

$$y_C = \sum_{i=1}^{M} s_i g_i = \sum_{i=1}^{N} s_i g_i + \sum_{i=N+1}^{M} s_i g_i = \sum_{i=1}^{N} s_i g_i + \theta \tag{5.63}$$

where $M, M > N$ is the new total number of pixels. By choosing the appropriate pixel background value for the filter and setting the corresponding pixels at the input image to 1, we can create any desired threshold, even a negative one. So to conclude, each correlator in a bank of correlators corresponds to a neuron in a single layer of neurons. Each individual pixel value of the filter placed in the correlator corresponds to an individual weight of the neuron. The central peak of the correlation (the inner product) between the input and the filter corresponds to the neuron output when its activation function is linear and its threshold is equal to zero.

5.4.4 Comparison of the similarity suppression algorithm with the unsupervised Hebbian learning law

Motivated by the equivalence between a single layer of neurons and a bank of correlators, in this section we are going to investigate the relationship between the similarity suppression algorithm and the unsupervised Hebbian learning law [101], which is one of the most common algorithms for training single layer neural networks. We are going to use both our and the usual neural network notation to make this comparison clearer. The unsupervised Hebbian learning law is also called the activity product rule and is expressed by the equation 4.3 which we rewrite here

$$\Delta w_{jm} = \eta y_j x_m \tag{5.64}$$

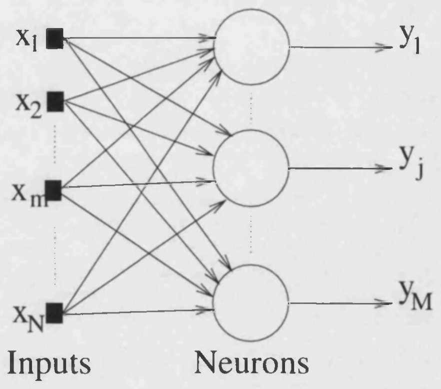


Figure 5.1: A single layer of neurons.

where (see figure 5.1) w_{jm} is the interconnection weight between the input m and neuron j. η is the convergence parameter. x_m is the value of the input, m and, y_j is the output of the neuron j and is given by the following equation

$$y_j = \sum_{m=1}^{N} w_{jm} x_m \tag{5.65}$$

where N is the total number of inputs or the number of pixels in our training patterns. Rewriting the two previous equations in our notation they take the form

$$\Delta g_{jm} = \eta y_j s_m \quad \text{or} \quad \Delta \mathbf{g}_j = \eta y_j \mathbf{s} \tag{5.66}$$

for the weight vector and

$$y_j = \sum_{m=1}^{N} g_{jm} s_m \quad \text{or} \quad y_j = \mathbf{g}_j \cdot \mathbf{s} \quad \text{or} \quad y_j = \mathbf{g}_j^T \mathbf{s} \tag{5.67}$$

where as before, we denote our training patterns with $\mathbf{s}$ and the filters, whose pixel values correspond to the weights of the network, with $\mathbf{g}$. If the network is trained in the batch mode, where the weight update is performed once after all of the training patterns have been presented to the network, then the weight update is given by (without summation convention)

$$\Delta w_{jm} = \eta \sum_{k=1}^{M} y_{kj} x_{km} \tag{5.68}$$

where k indicates the training pattern and M is the total number of training patterns. In our notation this equation can be written

$$\Delta g_{jm} = \eta \sum_{k=1}^{M} y_{kj} s_{km} \quad \text{or} \quad \Delta \mathbf{g}_j = \eta \sum_{k=1}^{M} \left\{ \mathbf{g}_j \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.69}$$

for the weight vector. The similarity suppression algorithm in its simplest form is given by equation 5.16 which we rewrite here

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.70}$$

$$\Rightarrow \Delta \mathbf{g}_j = -\beta \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{5.71}$$

Notice that we have omitted the square in equation 5.27 to make the comparison easier. Equations 5.69 and 5.71 are very similar. β and η play the same role in both algorithms i.e. convergence factors. Equation 5.69 shows that the weight update is equal to the product of the output from the neuron, $\mathbf{y}_{kj}$ and the input, $\mathbf{s}_{km}$, that is connected through the non-updated weight. This product is summed over all training examples, if the system is trained in batch mode. Similarly, the change in one of the stored filters in a bank of correlators, given in equation 5.71, is equal to the product of the central peak of the output of the correlator and the corresponding input.

There are two differences between the two algorithms.

i. The first is the different sign in equations 5.71 and 5.69. Mathematically this means that the magnitude of the weight vector in the Hebbian learning will increase while in the case of the similarity suppression algorithm the magnitude of the weight vector may increase or decrease depending on the value of the convergence factor β (as we saw in section 5.3). The plus sign in the Hebbian learning law can be interpreted as feature or similarity enhancement instead of similarity suppression that our algorithm performs. Thus the Hebbian Law performs generalisation, which means that the network is trained to give a high output for all of the training patterns and for other patterns similar to them. On the other hand, the SS algorithm performs discrimination, which means that the correlators are "trained" (or one can say that the filters placed in them are designed in such a way) to give low outputs with all but one of the training patterns.

ii. The second difference between the two algorithms is that in the Hebbian learning law the summation is done over the products of the output and *all* of the input patterns, while in the SS algorithm one product, that of the output and the input pattern corresponding to the updated filter, is excluded from the sum. This would be the biggest term in the summation provided that the weight vector (or the **g** pattern in our case) is normalised, since the normalisation changes the **g** filter in such a way so that its inner product with the corresponding s pattern is held constant. If that term was included in the SS algorithm, the result would be that the filters would be trained to discriminate against all of the input patterns and they would not produce a high output for any of them.

Both algorithms need a normalisation step to become stable and in that case the Hebbian learning law is called the normalised Hebbian rule [70].

5.5 Extension of the Similarity Suppression algorithm to train two or more consecutive banks of correlators

In this section, we extend the SS algorithm to calculate the filters for each of two cascaded banks of inner product correlators. Since one bank of correlators is mathematically equivalent to a single layer neural network, it cannot classify patterns that are not linearly separable. Two or more consecutive banks of correlators however, correspond to two or more interconnected neural network layers and can classify patterns that are not linearly separable. It is necessary however, that each correlator in the first bank is followed by a non-linear activation function, because from neural network theory we know that hidden units with linear activation functions provide no benefit in classifying patterns that are not linearly separable [3]. Based on the similarity between the SS algorithm and the Hebbian learning rule shown in the previous section, we are going to follow the well known derivation of the back-propagation learning algorithm using our filter formalism.

In figure 5.2 we can see two cascaded banks of correlators. There are T correlators in the first bank and M in the second. The same pattern s is input to all of the correlators in the first bank. The input to the correlators in the second bank, which is the same for all of them, is formed by the outputs of the correlators in the first bank. The correlation peak of each one of them, after the activation function is applied to it, corresponds to one pixel of the pattern which is input to the correlators in the second bank. Therefore, the filters in the first bank are of size N, where N is the size of the input patterns and the filters in the second bank are of size T, where T is the number of correlators in the first bank. The nonlinear activation functions between the two layers are not shown in figure 5.2

First let us define (following the definition in [95]) the average squared error to be equal to the mean of the squared errors of the outputs of all of the filters in

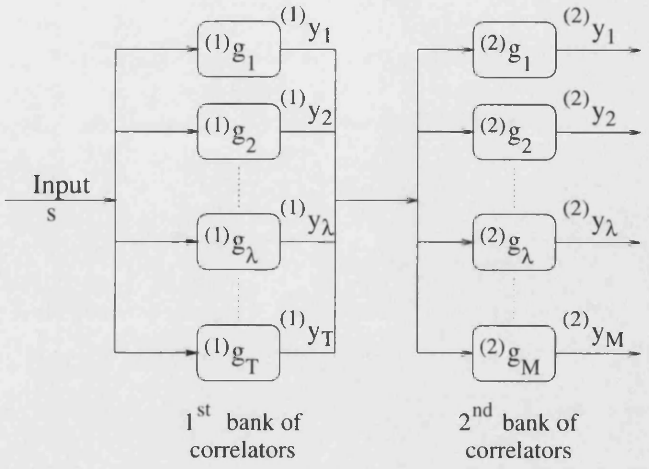


Figure 5.2: Two cascaded banks of correlators.

the output bank of correlators for all of the training patterns

$$E_{av} = \frac{1}{2M} \sum_{k=1}^{M} \sum_{\lambda=1}^{M} {}^{(2)}e_{\lambda k}^2 \tag{5.72}$$

where λ indicates the filter number in the output bank, k indicates the training pattern number and the number in the parenthesis on the top left of the each symbol (in this case (2)) indicates the bank number. The output bank is bank 2 and the hidden is bank 1. We have made the assumption that the number of the filters in the output bank is equal to the number of the training patterns. The error of each filter in the output layer is

$${}^{(2)}e_{\lambda k} = {}^{(2)}d_{\lambda k} - {}^{(2)}y_{\lambda k} \tag{5.73}$$

The change $\Delta\, {}^{(2)}g_{jm}$ to each of the pixels of filter ${}^{(2)}\mathbf{g}_j$ will be:

$$\Delta\, {}^{(2)}g_{jm} = -\beta \frac{\partial E_{av}}{\partial\, {}^{(2)}g_{jm}} = -\frac{\eta}{M} \sum_{k=1}^{M} {}^{(2)}e_{jk} \frac{\partial\, {}^{(2)}e_{jk}}{\partial\, {}^{(2)}g_{jm}} \tag{5.74}$$

where m denotes pixel number. By applying the chain rule of multi-variable differential calculus we have

$$\frac{\partial\, {}^{(2)}e_{jk}}{\partial\, {}^{(2)}g_{jm}} = \frac{\partial\, {}^{(2)}e_{jk}}{\partial\, {}^{(2)}y_{jk}} \frac{\partial\, {}^{(2)}y_{jk}}{\partial\, {}^{(2)}v_{jk}} \frac{\partial\, {}^{(2)}v_{jk}}{\partial\, {}^{(2)}g_{jm}} \tag{5.75}$$

From equation 5.73 we get

$$\frac{\partial\, {}^{(2)}e_{jk}}{\partial\, {}^{(2)}y_{jk}} = -1 \tag{5.76}$$

and from equation 5.62 ($y_N = \phi(v_N) = \phi(\sum_{i=1}^{N} s_i g_i - \theta)$) we get

$$\frac{\partial\, {}^{(2)}y_{jk}}{\partial\, {}^{(2)}v_{jk}} = {}^{(2)}\phi_j'({}^{(2)}v_{jk}) \tag{5.77}$$

and from equation 5.62 also

$$\frac{\partial\, {}^{(2)}v_{jk}}{\partial\, {}^{(2)}g_{jm}} = {}^{(1)}y_{ik} \tag{5.78}$$

where ${}^{(1)}y_{ik}$ is the output of filter i in the previous bank of correlators and the input to the second bank of correlators and corresponds to $\mathbf{s}_i$ in equation 5.62. So using equations 5.75-5.78, equation 5.74 becomes

$$\Delta\, {}^{(2)}g_{jm} = \frac{\eta}{M}\sum_{k=1}^{M} {}^{(2)}e_{jk}\, {}^{(2)}\phi_j'({}^{(2)}v_{jk})\, {}^{(1)}y_{mk} \tag{5.79}$$

so the equation for the update of the filters in the output layer when they are regarded as whole images is obtained by substituting equations 5.73 and 5.62 into equation 5.79

$$\Delta\, {}^{(2)}\mathbf{g}_j = \frac{\eta}{M}\sum_{k=1}^{M}\left\{ {}^{(2)}d_{jk} - {}^{(2)}\phi_j({}^{(2)}\mathbf{g}_j^{(i-1)} \cdot {}^{(1)}\mathbf{y}_k)\right\} {}^{(2)}\phi_j'({}^{(2)}\mathbf{g}_j^{(i-1)} \cdot {}^{(1)}\mathbf{y}_k)\, {}^{(1)}\mathbf{y}_k \tag{5.80}$$

where as usual all of the bold symbols denote vectors. The local gradient for the filter ${}^{(2)}\mathbf{g}_j$ in the output layer is defined to be

$${}^{(2)}\delta_{jk} = {}^{(2)}e_{jk}\, {}^{(2)}\phi_j'({}^{(2)}v_{jk}) \tag{5.81}$$

and the filter update equation can be rewritten by substituting equation 5.81 into equation 5.80 in the following form

$$\Delta\, {}^{(2)}\mathbf{g}_j = \frac{\eta}{M}\sum_{k=1}^{M} {}^{(2)}\delta_{jk}\, {}^{(1)}\mathbf{y}_k \tag{5.82}$$

For the update of filter ${}^{(1)}\mathbf{g}_j$ in the hidden layer we cannot use the local gradient defined in equation 5.81 because the calculation of the error ${}^{(1)}e_{jk}$ is not straightforward because we do not know what the desired outputs of the correlators in the first bank should be. We define the local gradient ${}^{(1)}\delta_{jk}$ as follows

$${}^{(1)}\delta_{jk} = -\frac{\partial E_{av}}{\partial\, {}^{(1)}y_{jk}}\frac{\partial\, {}^{(1)}y_{jk}}{\partial\, {}^{(1)}v_{jk}} = -\frac{\partial E_{av}}{\partial\, {}^{(1)}y_{jk}}\, {}^{(1)}\phi_j'({}^{(1)}v_{jk}) \tag{5.83}$$

Using equations 5.72 and 5.62, we can calculate the partial derivative $\partial E_{av}/\partial\, {}^{(1)}y_{jk}$ as follows

$$\frac{\partial E_{av}}{\partial\, {}^{(1)}y_{jk}} = \frac{1}{M}\sum_{k=1}^{M}\sum_{\lambda=1}^{M} {}^{(2)}e_{\lambda k}\frac{\partial\, {}^{(2)}e_{\lambda k}}{\partial\, {}^{(1)}y_{jk}} = \frac{1}{M}\sum_{k=1}^{M}\sum_{\lambda=1}^{M} {}^{(2)}e_{\lambda k}\frac{\partial\, {}^{(2)}e_{\lambda k}}{\partial\, {}^{(2)}v_{\lambda k}}\frac{\partial\, {}^{(2)}v_{\lambda k}}{\partial\, {}^{(1)}y_{jk}} \tag{5.84}$$

In the output layer

$$ {}^{(2)}v_{\lambda k} = \sum_{j=1}^{M} {}^{(2)}g_{\lambda j}\, {}^{(1)}y_{jk} \tag{5.85} $$

or for images

$$ {}^{(2)}v_{\lambda k} = {}^{(2)}\mathbf{g}_{\lambda} \cdot {}^{(1)}\mathbf{y}_{k} \tag{5.86} $$

From equations 5.62, 5.73 and 5.76 we get

$$ \frac{\partial\, {}^{(2)}e_{\lambda k}}{\partial\, {}^{(2)}v_{\lambda k}} = -\, {}^{(2)}\phi'_{\lambda}({}^{(2)}v_{\lambda k}) \tag{5.87} $$

and from equation 5.85 we get

$$ \frac{\partial\, {}^{(2)}v_{\lambda k}}{\partial\, {}^{(1)}y_{jk}} = {}^{(2)}g_{\lambda j} \tag{5.88} $$

so substituting equations 5.87 and 5.88 into equation 5.84 we get

$$ \frac{\partial E_{av}}{\partial\, {}^{(1)}y_{jk}} = -\frac{1}{M} \sum_{k=1}^{M} \sum_{\lambda=1}^{M} \underbrace{{}^{(2)}e_{\lambda k}\, {}^{(2)}\phi'_{\lambda}({}^{(2)}v_{\lambda k})}_{{}^{(2)}\delta_{\lambda k}} {}^{(2)}g_{\lambda j} \tag{5.89} $$

and substituting equation 5.89 into equation 5.83 we get

$$ {}^{(1)}\delta_{jk'} = \frac{1}{M} \sum_{k=1}^{M} \sum_{\lambda=1}^{M} {}^{(2)}\delta_{\lambda k}\, {}^{(2)}g_{\lambda j}\, {}^{(1)}\phi'_{j}({}^{(1)}v_{jk'}) \tag{5.90} $$

and from equation 5.82 the filters in the first bank of correlators can be updated by the following equation

$$ \Delta\, {}^{(1)}g_{jm} = \frac{\eta}{M^2} \sum_{k'=1}^{M} \sum_{k=1}^{M} \sum_{\lambda=1}^{M} {}^{(2)}e_{\lambda k}\, {}^{(2)}\phi'_{\lambda}({}^{(2)}v_{\lambda k})\, {}^{(2)}g_{\lambda j}\, {}^{(1)}\phi'_{j}({}^{(1)}v_{jk'}) s_{k'm} \tag{5.91} $$

or the same equation for the whole images can be written

$$ \Delta\, {}^{(1)}\mathbf{g}_{j} = \frac{\eta}{M^2} \sum_{k'=1}^{M} \sum_{k=1}^{M} \sum_{\lambda=1}^{M} A \tag{5.92} $$

where

$$ A = \left\{ {}^{(2)}d_{\lambda k} - {}^{(2)}\phi_{\lambda}({}^{(2)}\mathbf{g}_{\lambda} \cdot {}^{(1)}\mathbf{y}_{k}) \right\} {}^{(2)}\phi'_{\lambda}({}^{(2)}\mathbf{g}_{\lambda} \cdot {}^{(1)}\mathbf{y}_{k})\, {}^{(2)}g_{\lambda j}\, {}^{(1)}\phi'_{j}({}^{(1)}\mathbf{g}_{j} \cdot \mathbf{s}_{k'}) \mathbf{s}_{k'} \tag{5.93} $$

Equation 5.80, which describes the update of the filters in the second bank of correlators, is very similar to the basic equation of the SS algorithm with the main difference being the presence of the non-linear activation function and it's derivative. So in effect it trains the filters in the output bank to distinguish among the images that are output from the first bank. Equation 5.92 which describes the correction that must be applied to the filters in the first bank of correlators, is not as straightforward to explain. It updates the filters in the first bank, based on the average error of the filters in the second (output) bank, which can be measured. The algorithm which calculates the filters for more than two cascaded banks of correlators can be derived in a similar manner using the same chain differential rule for the other layers.

5.6 Discussion and conclusions

In this chapter we have addressed the problem of designing a set of filters which are mutually orthogonal to a set of training patterns. We developed a similarity suppression algorithm which starts from a set of training patterns, and creates a set of filters. Each of the filters has a high inner product (equal to 1, assuming that all of the training patterns are normalised) with only one of the training patterns; the one that it was derived from. In addition, each of the filters has very low inner products with all of the other training patterns.

We showed that with our choice of convergence parameter the magnitudes of the filters will decrease, unless a normalisation step is used. We chose to normalise the filters in such a way that we ensured that their auto-inner products would remain stable at a desired value. However, by doing that we increased the magnitudes of the filters themselves and we now suspect that if a random pattern, or white noise is input to the system, the output will be biased towards the filter whose magnitude is the largest. There is, however, a way around this if we use a different normalisation.

In the third section of this chapter we compared the SS algorithm to some other filter design techniques. The first version of the algorithm which orthogonalises the filters themselves is a symmetrical, iterative version of the Gram-Schmidt procedure.

We also compared the SS algorithm to the Linear Combination filters and to the general Synthetic Discriminant Function filters. The SS algorithm converges towards the LCFs and SDFs solution. The two filter design methods are, therefore, roughly equivalent. In the next chapter we are going to investigate whether the algorithm will converge to the exact solution as the LCFs with the help of computer simulations.

It is well known [4] that a bank of correlators is mathematically similar and, if the threshold is equal to zero and the activation function is linear, in some cases equivalent to a single layer of neurons. Therefore, it can be used to implement a 1-layer neural network. There is also an obvious similarity in the equations describing the SS algorithm and the unsupervised Hebbian learning law. Their main difference is that in its present form the SS algorithm performs discrimination, while the Hebbian law performs generalisation. However, a single layer of neurons and a bank of correlators can only perform correct recognition when the patterns to be recognised are linearly separable. The back-propagation learning algorithm is well known for its ability to train neural networks with hidden layers of neurons. Based on the equivalence between the SS algorithm and the Hebbian law we have extended the SS algorithm to design filters for two or more cascaded banks of correlators. In doing that, we have not devised a completely new training algorithm, but rather expressed the back-propagation algorithm in terms which refer to whole images and are better suited to the design of filters for optical correlators.

In the next chapter we are going to present some computer simulations of the SS algorithm. With these simulations we are going to verify the theoretical analyses presented in this chapter and investigate the ability of the final filters to recognise noisy patterns.

Chapter 6

Computer simulations of the Similarity Suppression algorithm

6.1 Introduction

In chapter 5 we described the similarity suppression algorithm. In this chapter we are going to present some of the computer simulations which we conducted while we were developing the algorithm. Some of these simulations helped us do some modifications to the algorithm and others verified our theoretical conclusions. In section 6.2 we use the algorithm to calculate two sets of filters to recognise two different sets of training patterns in the presence of noise. We investigate the effect of the convergence parameter on the speed of convergence and on the final solution. In addition, we observe the magnitudes of the filters to verify the conclusions we drew in section 5.3. In section 6.3 we present computer simulations which test the performance of one of the sets of filters, calculated in section 6.2, in recognising images buried in additive input white noise. Section 6.4 presents computer simulations which compare the SS algorithm with linear combination filters. These simulations verify the theoretical comparison between the SS algorithm and linear combination filters shown in the previous chapter and clarify the relationship between the two methods even further. Finally, in section 6.5 we use the results of computer simulations to optimise the number of times that the algorithm is allowed to iterate, which yields some surprisingly beneficial results.

6.2 Convergence Simulations

This section presents the performance of the algorithm during the iterative training phase. The next section presents the performance of the filters, **g**, so produced, at discriminating patterns in noise. In order to assess the efficacy of the algorithm it is necessary to choose and to devise appropriate performance measures. These are introduced below, followed by a detailed description of the simulation parameters and results.

6.2.1 Performance Measures

We define three performance metrics:

i. Cross-inner product matrix

A matrix R which we call the cross-inner product matrix was calculated at each iteration. The R_{ij} element of the cross-inner product matrix was equal to the value of the inner product of the patterns $\mathbf{g}_i$ and $\mathbf{s}_j$. The goal of the training is to minimise all of the elements of the cross-inner product matrix except from the ones that are on the diagonal, which remain constant and equal to the normalised magnitudes of the training patterns, P. In the first iteration, when the **g** patterns are identical to the **s** patterns, matrix R is the *vector (auto) inner product matrix* of the input patterns as defined in [66].

ii. Global Energy

A term which we will call the total energy of the system was defined as

$$TE = \frac{1}{M^2 P} \sum_{i=1}^{M} \sum_{j=1}^{M} |\mathbf{g}_i \cdot \mathbf{s}_j| \tag{6.1}$$

In other words the total energy of the system is equal to the normalised sum of the modulus of all of the elements of the cross-inner product matrix. The total energy is a measure of the height of all of the cross inner products. As the algorithm converges, we expect the total energy to decrease.

iii. Largest cross-inner products

In order to monitor the system's convergence, another figure of merit was calculated. This figure of merit was the average size of the modulus of the three largest cross-inner products as a fraction of P, calculated at each iteration.

6.2.2 Binary, bipolar patterns

Our aim when conducting these simulations was to see how much the cross-inner products were reduced, how many iterations it took for these reductions to take place and how the final cross-inner product values and the convergence speed were affected by the convergence parameter β. The first training set of patterns to be recognised, consisted of 32, 16x16 binary, bipolar patterns denoted by $\mathrm{s}_i, i = 1, \ldots, 32$. We chose to use binary, bipolar patterns for their simplicity which helped us to evaluate the results in the early stages of the development of the algorithm. Eight patterns in the training set were chosen to have random elements. Those were patterns numbers 1,7,10,11,17,20,21 and 22 as they appeared in the set. The other patterns were similar to one of patterns 1, 11 or 22, differing by 7, 14, 28, 56 and 112 pixels. Table 6.1 shows the order of the patterns in the training set, the similar patterns and by how many pixels they differ. The patterns that are similar to one another were created by copying the initial pattern and then randomly changing the desired number of pixels. We constructed this specific training set so that there were some very similar and some very different patterns in it. If all of the training patterns had been chosen to be different, the algorithm would not produce much benefit, because if all of the cross-inner products were already small there would not be any reason for them to change.

Pattern No:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Similar to :	-	1	1	1	1	1	-	22	22	-	-	11	11	11	11	11
Differing by:	-	7	14	28	56	112	-	7	14	-	-	7	14	28	56	112
Pattern No:	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Similar to:	-	11	11	-	-	-	22	22	22	22	22	1	11	1	22	1
Differing by:	-	7	14	-	-	-	7	14	28	56	112	7	7	7	14	14

Table 6.1: Number of pixels differing in the training set.

The advanced algorithm, described in equations 5.27 and 5.28, was tested with several different values of the convergence parameter β. Here we present some representative results for 6 different values of β, which are $\beta = 0.01, 0.1, 0.5, 1, 4$ and 6. For most of the β values the algorithm converged to a sufficiently stable solution within the first 1500 iterations.

In figure 6.1 one can see the three dimensional graph of the initial state of the cross-inner product matrix. The palest shading shows the highest peaks. The large values on the diagonal represent the auto-inner products. All of the other peaks, some of which are large (but no bigger than the auto-inner products) represent

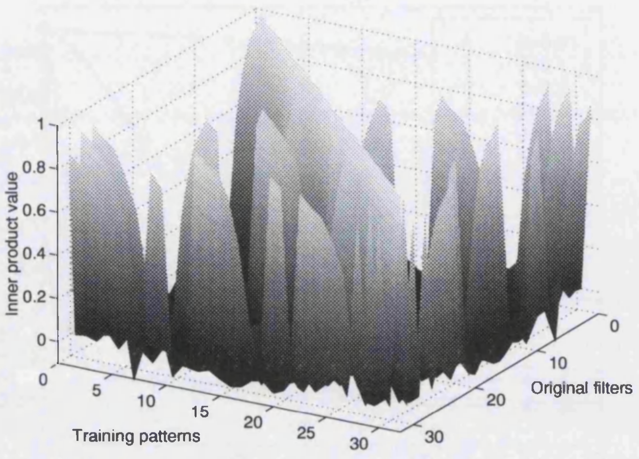


Figure 6.1: Cross-inner product matrix before the training. The graph is shown with the x and y axes reversed for clarity.

the cross-inner products and those are the ones that we want to decrease. The total energy index is plotted in figure 6.2 as a function of iteration number. For most of the β values,the total energy reduces rapidly, which means that the cross-inner products decrease. In addition, for most of the β values the total energy decreases exponentially. The decrease is very fast initially and slows down later. The average of the absolute value of the three largest cross-inner products as a fraction of the auto-inner product's normalised value P, is shown in figure 6.3, as a function of iteration number. The algorithm converged to the desired solution

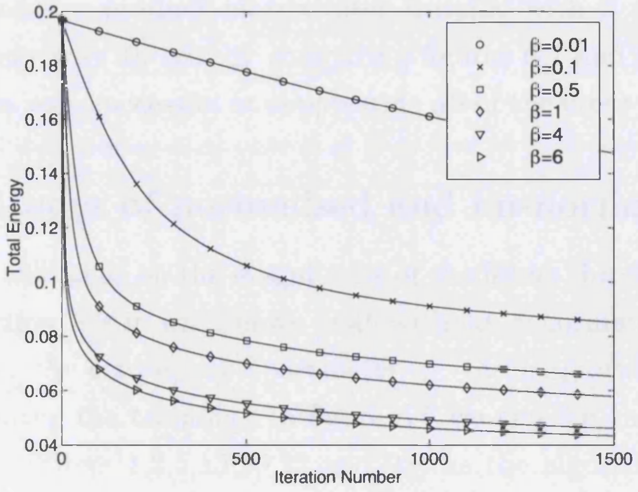


Figure 6.2: Total energy index.

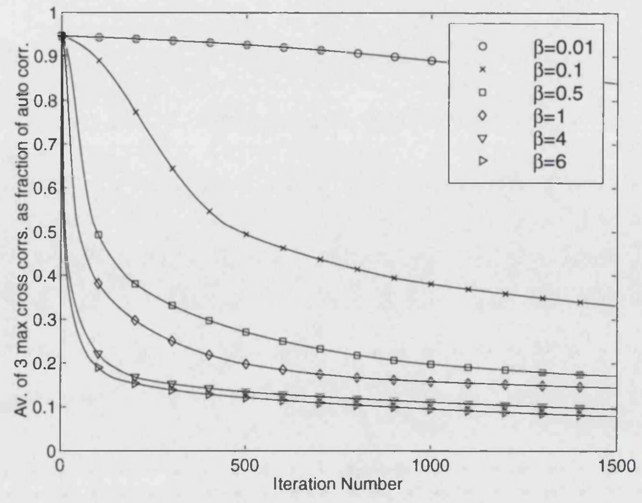


Figure 6.3: Absolute average values of the 3 largest inner products as a function of iteration number for various values of the convergence factor, β.

for most of the values of β. As we expected, convergence is a lot faster when β is large (figures 6.2 and 6.3). In addition, the cross-inner products converged to lower values as β increased. However, β values larger than 6, destabilised the algorithm and as a result the inner products oscillated between their initial and very large values. Figure 6.3 shows that even the largest cross-inner products converged to a value less than 10% of the auto-inner product value P for most of the β values (after about 800 iterations). In the best case ($\beta = 6$) most of the convergence has taken place after just 100 iterations. The three dimensional graph of the cross-inner product matrix after training with $\beta = 6$ is shown in figure 6.4. It is very easy to see, by comparing figures 6.4 and 6.1, that the SS algorithm has been very successful at suppressing all of the cross-inner products.

6.2.3 Magnitudes of normalised and un-normalised filters

In this section we will focus on the magnitudes of the filters throughout the training phase. In section 5.3 it was shown that without a normalisation step and with our choice for the convergence parameter, β, the magnitudes of the filters would decrease during the training. In figure 6.5 we can see the magnitudes of some of the filters (filters 1,2,5,13,20,22 and 23) as the algorithm converges to the final solution, when the normalisation equation is not used. We have chosen to show these particular filters because some of them (filters 1,2,5,13,22 and 23) were derived from patterns that were more or less similar to others and some

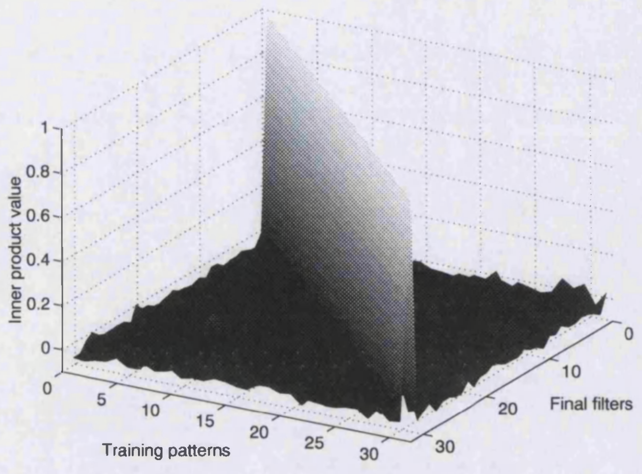


Figure 6.4: Cross-inner product matrix after 1500 iterations for a convergence factor of $\beta = 6$. The graph is shown with the x and y axes reversed.

(filter 20) were derived from patterns different to all others. First of all we must point out that in figure 6.5, the magnitudes of all of the filters have the same initial value, 256. We can see in the figure that the magnitudes of all of the filters decrease as was predicted in the theory in section 5.3. However, they do not all decrease by the same amount. A first observation one can make in this figure is that the largest drop in the filters' magnitudes takes place in the first iteration. In addition, the amount that the filters' magnitudes decrease, depends on the similarity between their corresponding initial patterns and other patterns in the set. A general trend seems to exist: the amount of the decrease of a filter's magnitude during the training without normalisation depends on the initial pattern from which it was derived and it is proportional to the similarity between that pattern and the other patterns in the set, as well as to the number of these similar patterns. This can be verified in figure 6.6 which shows the magnitudes of all of the filters, each one depicted with an "x" on the graph, after the training without normalisation. They are plotted versus the similarity amongst the corresponding training patterns.

This similarity amongst the training patterns was calculated in the following manner: for each of the training patterns, we calculated the number of pixels which had equal value to pixels in other patterns. For example, the first training pattern has 249 pixels equal to pixels in the second pattern (they differ by 7 pixels, therefore, 256-7=249), another 242 pixels equal to pixels in the third pattern and so on and in total it has at least 1803 pixels equal to pixels in all of the other

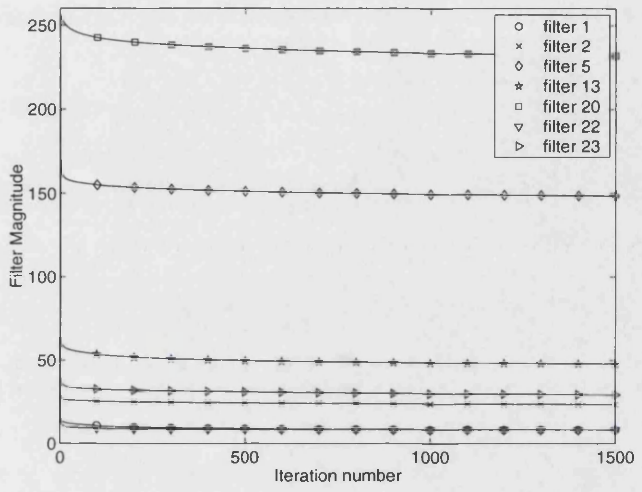


Figure 6.5: Unnormalised magnitudes of some of the filters as a function of the number of iterations.

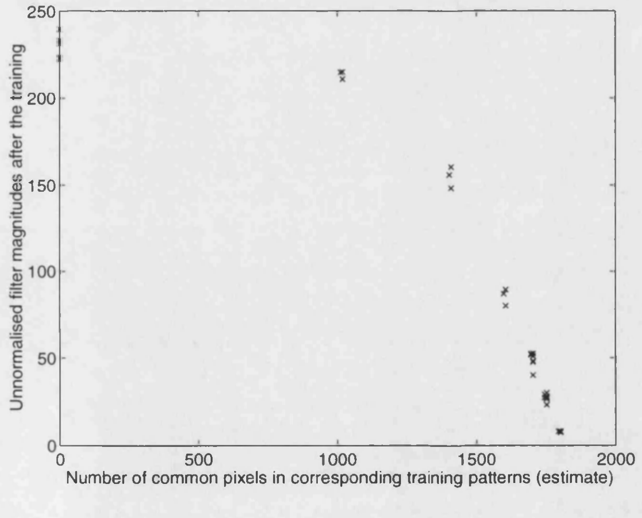


Figure 6.6: Un-normalised magnitudes of all of the filters after the training versus similarity amongst the corresponding training patterns.

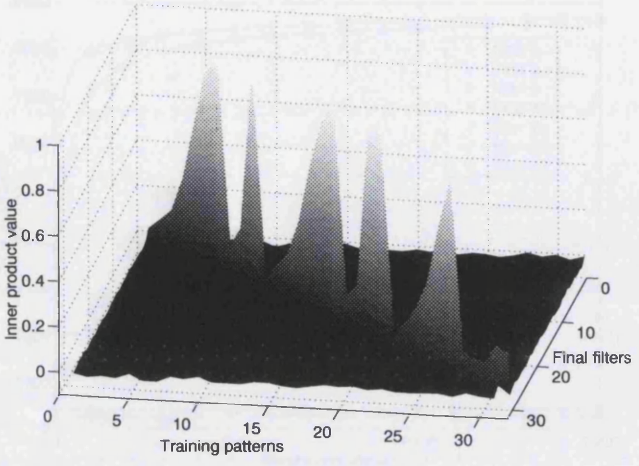


Figure 6.7: Cross-inner product matrix after 1500 iterations without normalisation for a convergence factor of $\beta = 6$. The graph is shown with the x and y axes reversed for clarity.

patterns in the training set. We used the terms "at least", because the previous calculation did not take into account the pixels which randomly happen to have equal values with pixels in other patterns. The second training pattern has 249 pixels equal to pixels in the first pattern, 256-7-14=235 pixels equal to pixels in the third training pattern and so on and in total it has 1754 pixels equal to pixels in all of the other patterns. Again, this number is not accurate, but just an estimate, but it suffices for our purpose here. The graph shown in figure 6.6 verifies our previous conclusion, that the amount that a filter's magnitude is going to decrease during the training depends on the similarity between that filter's corresponding training pattern and all of the other patterns in the training set.

So the filter magnitudes do decrease and that has an effect on their auto-inner products with the corresponding training patterns. Since there is no normalisation step, these will decrease as well. In figure 6.7 we can see the final state of the cross-inner product matrix. All of the cross-inner products have decreased to very low values, but some of the auto-inner products have decreased as well and this is undesirable. To recognise its corresponding training pattern correctly, each of the filters must have an auto-inner product with it which is higher than the cross-inner products with the other patterns. Again one can observe that the decreased auto-inner products are the ones that correspond to initially similar patterns. We introduced the normalisation step to stabilise the auto-inner products and solve this problem, but we predicted that this normalisation would increase the

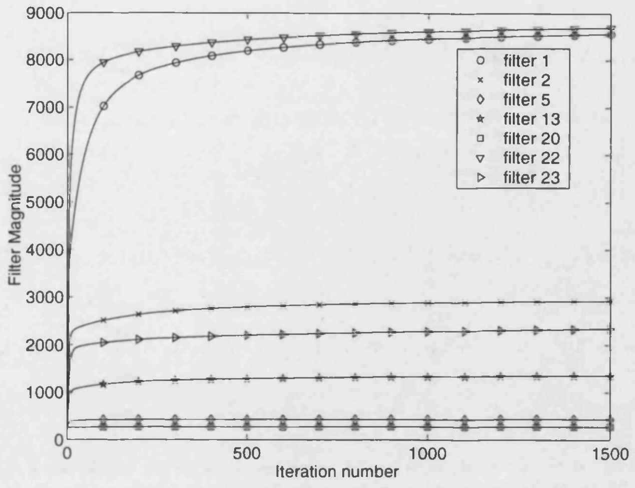


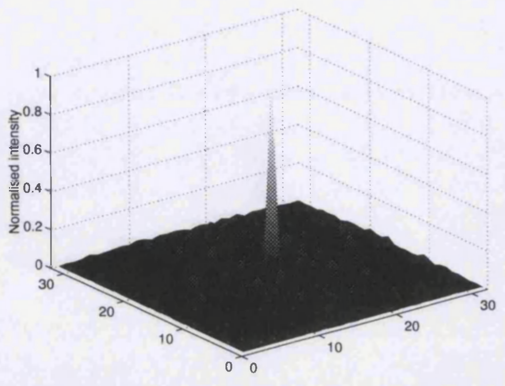
Figure 6.8: Normalised magnitudes of some of the filters as a function of the number of iterations.

magnitudes of the filters unevenly. This can be verified in figure 6.8 which shows the magnitudes of the same filters but in this case when they were normalised throughout the training. Due to the normalisation, the strongest filters are the ones that were the weakest without the normalisation step.

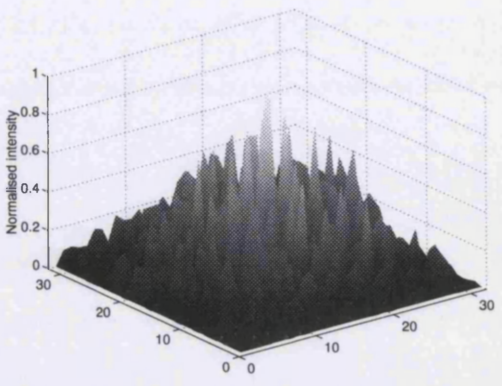
This increase in the magnitudes of some of the filters after using the normalisation is undesirable, because, as we said in section 5.3, there will now be a bias towards them if random patterns are presented into the recognition system. This bias is quite strong, as from figure 6.8 we can see that the strongest filters are roughly 33 times stronger than the weakest.

6.2.4 Peak-to-Correlation Energy of the correlations between the training patterns and the trained and untrained filters

The SS algorithm is very successful at decreasing the cross-inner products between filters and patterns that do not correspond to them. Each filter starts by being identical to one of the training patterns and then changes so that it becomes different to all of the other patterns in the set. These changes, however, must have an effect not only on the inner product between the filters and the patterns, but on the whole correlation plane. The algorithm forces the inner products to decrease but it does not place any constraints on the outer products. In figure 6.9 we can see the intensity profile in the correlation plane for two correlations. Subfigure 6.9-(a)



(a) $\mathbf{s}_1 \otimes \mathbf{s}_1$, PCE=0.53



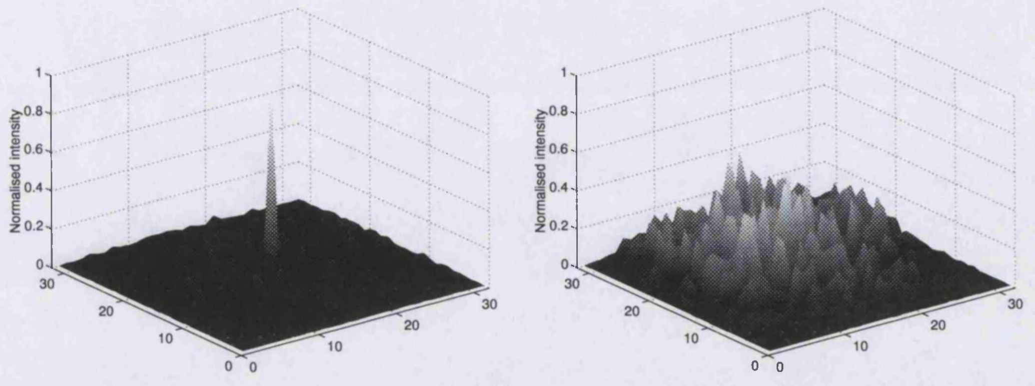
(b) $\mathbf{s}_1 \otimes \mathbf{g}_1$, PCE=0.027

Figure 6.9: Correlation plane intensity for auto-correlation of pattern 1 and correlation between pattern 1 and filter 1

depicts the auto-correlation of pattern 1. Subfigure 6.9-(b) depicts the correlation between pattern 1 and filter 1, which was obtained after 1500 iterations with $\beta = 6$. We can see that the auto-correlation of pattern 1 has a sharp peak (as expected) and a PCE (as defined in chapter 2, equation 2.6) equal to 0.53. When using the filter corresponding to pattern 1, the inner product has remained stable, but the outer products have increased a lot and the PCE is now only 0.027. For a correlation between a filter and the pattern it corresponds to, we want a high correlation peak and low side-lobes, therefore, a high PCE as close to 1 as possible is desirable. Therefore, the fact the PCE has decreased so much in the correlation between the first pattern and its corresponding filter is a disadvantage.

Figure 6.10 shows the intensity profile of the correlation plane for the correlations between pattern 1 and pattern 2 (subfigure 6.10-(a)) and between pattern 1 and filter 2 (subfigure 6.10-(b)). The correlation between the two initial patterns has a high, sharp central peak because the patterns are very similar. With filter 2, the central peak of the correlation has decreased (to less than 10% of P) but the outer products have increased to about 65% of the auto-correlation peak value, P.

Another interesting example can be seen in figure 6.11 which shows the correlations between pattern 7 and pattern 1 (in subfigure 6.11-(a)) and pattern 7 and filter 1 (in subfigure 6.11-(b)). In the correlation between patterns 7 and 1 there is no correlation peak and the outer products are all low, because the two patterns are very different. However, when using filter 1, the central point of the

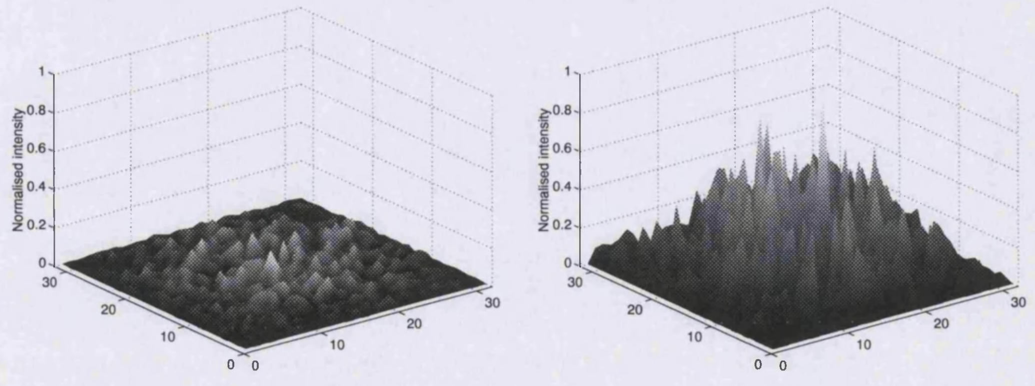


(a) $\mathbf{s}_1 \otimes \mathbf{s}_2$ (b) $\mathbf{s}_1 \otimes \mathbf{g}_2$

Figure 6.10: Correlation plane intensity for correlations between pattern 1 and pattern 2 and between pattern 1 and filter 2.

correlation plane, i.e. the inner product, may still have a very low value, but the outer products have increased dramatically and one of them is about 80% of the auto-correlation peak value, P.

To conclude, the algorithm reduces the inner products, but does not put any constraints on the outer products, so they increase. Not all of the correlations have their outer products increased by the same amount. The biggest increase in the outer products occurs in correlations of filters that were derived from patterns which were similar to others at the beginning of the training. The auto-



(a) $\mathbf{s}_7 \otimes \mathbf{s}_1$ (b) $\mathbf{s}_7 \otimes \mathbf{g}_1$

Figure 6.11: Correlation plane intensity for correlations between pattern 7 and pattern 1 and between pattern 7 and filter 1.

correlation's outer products increase a lot because of the normalisation step. A filter is changed during the training but that makes it different from the pattern from which it was derived, as well as from other patterns. When it is normalised so that it's auto-inner product reaches the desired level, it's magnitude increases as we saw in the previous section and as a result, all of the outer products of the correlation increase.

This increase in the outer products in the correlations plays no role in electronic systems, but is a major drawback in optical systems, unless the input images are always centered. In the case when the location of the object in the input scene is not known precisely, the increased outer products would make an optical system a lot less useful, because a high outer product could be mistaken for a correlation peak.

6.2.5 Real valued patterns

In addition to binary, bipolar patterns, we tested the algorithm with some real valued patterns. The training set consisted of ten patterns. Each was 112 $\times$ 92 pixels. The patterns were monopolar, grey-level and the pixels took integer values between 1 and 256. Each pattern was a photograph of a person's face. The patterns were part of the Olivetti Research Laboratory (ORL) faces database. The photographs that were used can be seen in figure 6.12. We used the algorithm on this second training set, mainly to demonstrate that it works with real valued patterns as well. In addition we use this training set to test the Feature Enhancement and Similarity Suppression (FESS) algorithm, which is presented in the next chapter and these simulations will help us to compare the two algorithms.

The patterns that are shown in figure 6.12 are not normalised. They were normalised, however, before they were presented to the algorithm. The vector-inner product matrix for the normalised patterns before the training is shown in figure 6.13. It is clear from the graph that the initial patterns are all very similar. All of the cross-inner products are high and their magnitudes are about 80-90% of the auto-inner products. We used the algorithm with a convergence factor $\beta = 0.65$. We let it run for 2000 iterations until it converged to a stable solution. The final cross-inner product matrix after the training is shown in figure 6.14. Again we can see that all of the cross-inner products have decreased to very small values. The algorithm, therefore, works equally well with real valued patterns.

Finally, figure 6.15 shows the filters that were created. The resulting filters were bipolar, real valued. A better understanding of the way that the algorithm

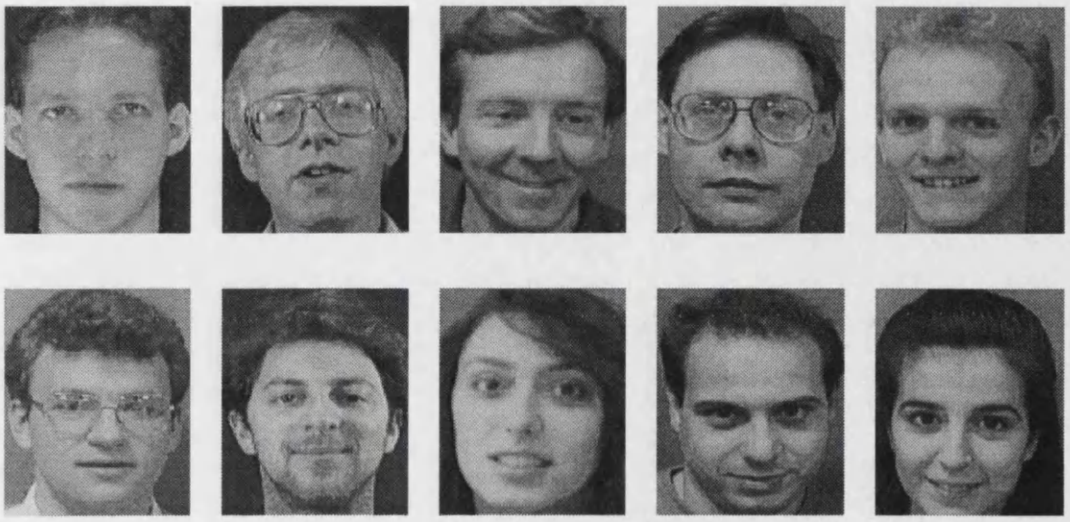


Figure 6.12: Training set consisting of ten people's faces.

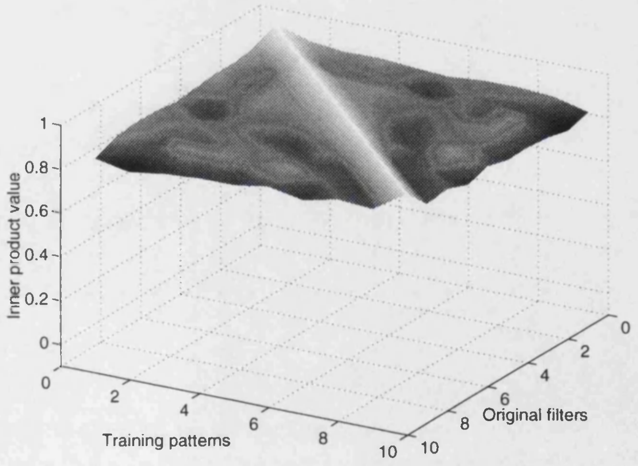


Figure 6.13: Vector-inner product matrix before the training.

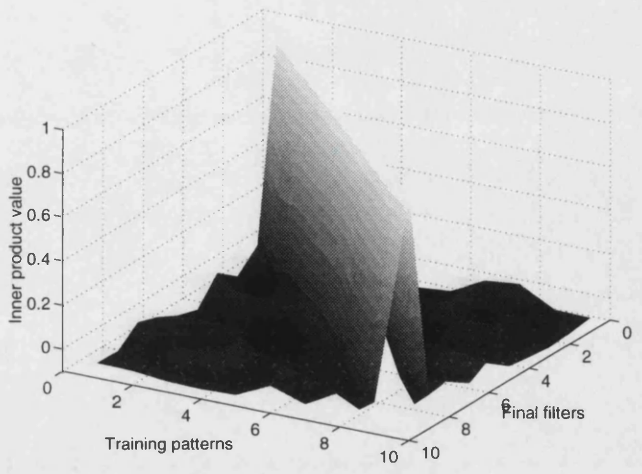


Figure 6.14: Cross-inner product matrix after 2000 iterations for a convergence factor of $\beta = 0.65$.

creates the filters can be gained from these figures because unlike the previous random patterns, these represent human faces and have a meaning to us. Some of the filters have features of other patterns in them but with a negative sign, like for example the glasses on the first filter. Most of the filters have some areas strengthened (very bright, or very dark) or weakened (grey) and usually bright areas in one filter correspond to dark areas in the others.

6.3 Probability of discrimination and dynamic range

Optical inner product correlator pattern recognition systems suffer from the limited dynamic range inherent in optics. For example, in the extreme case of two bipolar N pixel patterns, that differ only by one pixel and which need to be distinguished, the dynamic range of the optical system has to be greater than $20\log_{10} N/2$ dB in the inner-product correlator domain, for correct recognition. Taking into account that the dynamic range of a typical optical system can be about 30dB, one can see that the pattern's number of pixels cannot be greater than 64, meaning that no larger than a 8x8 pixel image can be recognised optically.

The SS algorithm minimises the cross-inner products and holds the auto-inner products constant so that they differ by a larger amount. The dynamic range required by a detector at the output inner-product plane of an optical system is

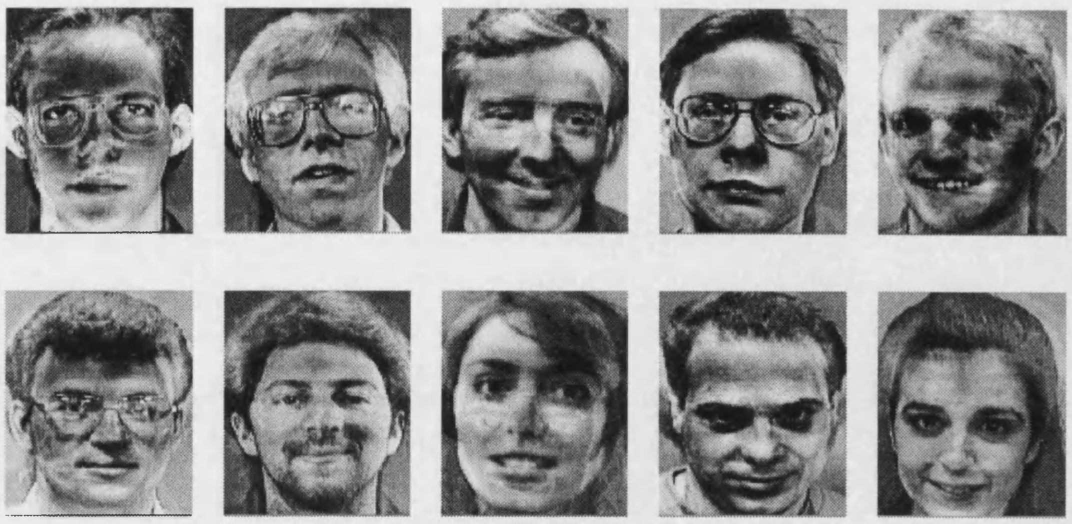


Figure 6.15: Final filters for the faces set.

reduced and less sensitive equipment is needed. So the SS algorithm allows us to increase the size of the images that can be recognised by an optical system.

In most cases the pattern that needs to be recognised will contain an amount of noise, where we are using the word "noise" in a broad sense indicating additive or multiplicative noise or distortion, rotation or a proportion of another pattern. It is important to see how much noise can be tolerated before the pattern becomes unrecognisable, and how much the required dynamic range is, for each noise level. We conducted simulations with analogue additive noise. The dynamic range requirements for correct discrimination and the probability of discrimination were calculated for different levels of noise. The noise added to the patterns was normally distributed with a zero mean. The input signal to noise ratio (SNR) varied from 20 to -10dB. The results shown in this section were obtained using the filters which were calculated with $\beta = 6$. The method for calculating the probability of discrimination, was to calculate all of the inner products between an input pattern and all of the filters and then to choose the highest of them. For correct discrimination, the highest inner product had to be the one with the filter which corresponded to the input pattern. The experiment was repeated for all of the training patterns for 5000 different samples of noise for each different noise level. We did not use a threshold because the SS algorithm addresses the problem of discrimination between patterns and not detection.

The resulting curves for the probability of discrimination before and after training are shown in figure 6.16. We can see, in that figure, that there is a signif-

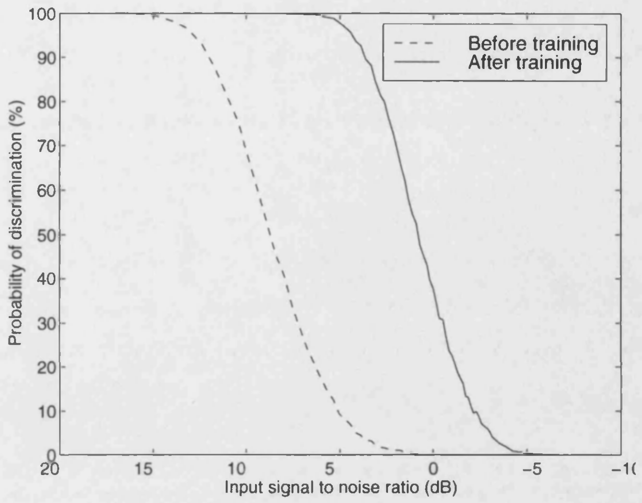


Figure 6.16: Probability of discrimination versus input signal to noise ratio

icant increase in the probability of discrimination after the training. For example, with an input signal to noise ratio of 3 dB the probability of discrimination is 2% before the training and it increases to 83% after the training. Also the probability of discrimination falls to 50% at an input signal to noise ratio of 8.9 dB before the training and 0.8 dB after the training. The curve after the training is almost a shifted version of the curve before the training, although it is a bit steeper. This means that the same pattern discrimination behaviour versus SNR can be achieved but we can tolerate 7dB more noise.

The dynamic range of the recognition system was defined to be the ratio of the difference between the auto-inner product and the maximum cross-inner product, to the corresponding auto-inner product, in decibels. This can be written as:

$$dynamic\ range = max\forall i\Bigg(-20\log_{10}\left\{\frac{\mathbf{s}_i \cdot \mathbf{g}_i - max\forall j(\mathbf{s}_i \cdot \mathbf{g}_j)}{\mathbf{s}_i \cdot \mathbf{g}_i}\right\}\Bigg),$$

$$i = 1 \ldots M,\ j = 1 \ldots M,\ j \neq i \tag{6.2}$$

This definition assumes that the system has some form of automatic gain control which, for example, scales the maximum auto-inner product to a constant near the top of the dynamic range.

The resulting plot of the dynamic range versus the signal to noise ratio before and after the training is shown in figure 6.17. The error bars in figure 6.17 indicate the standard deviation (as defined in appendix A) of the calculated dynamic range values for 5000 measurements. We can see from figure 6.17 that there is a very large reduction in the dynamic range required for correct discrimination, of the

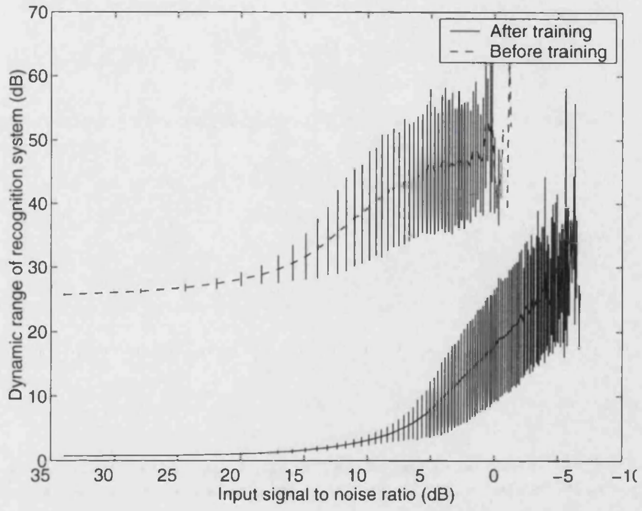


Figure 6.17: Dynamic range of the recognition system as a function of the signal to noise ratio. The error bars show the standard deviation for 5000 measurements.

order of 25 dB, after the training. The amount of reduction lessens for higher amounts of noise. The error bars increase as the noise is increased due to the random nature of the noise. The worse case after the training is better than the best case before the training, for the same amount of additive noise, because the error bars do not meet. The curve before the training does not extend to higher noise levels because, from figure 6.16, when the probability of discrimination drops to zero it is not meaningful to plot the dynamic range. From graph 6.17 we can also see that if an optical system has a dynamic range of 30dB this means that, before training, patterns can be recognised having an input SNR of 15dB upwards whereas, after training, the dynamic range of the system does not limit discrimination.

6.4 Comparison between the filters produced with the SS algorithm and the linear combination filters

In section 5.4.2 we compared the SS algorithm to the method proposed by Caulfield and Maloney in [18] for designing linear combination filters and we concluded that the SS algorithm converges to the same solution as the one provided from Caulfield's and Maloney's method. In this section we use that method to create

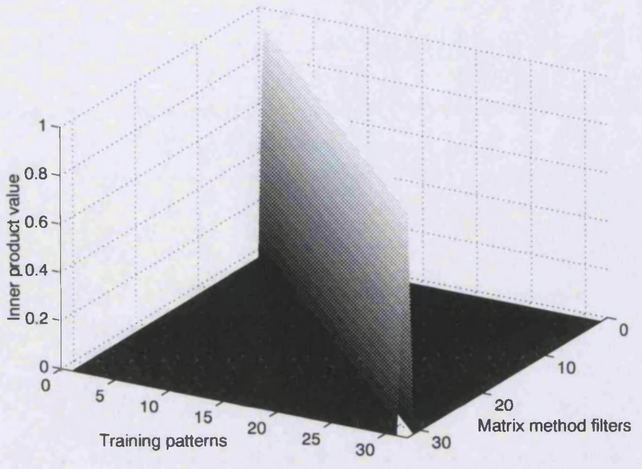
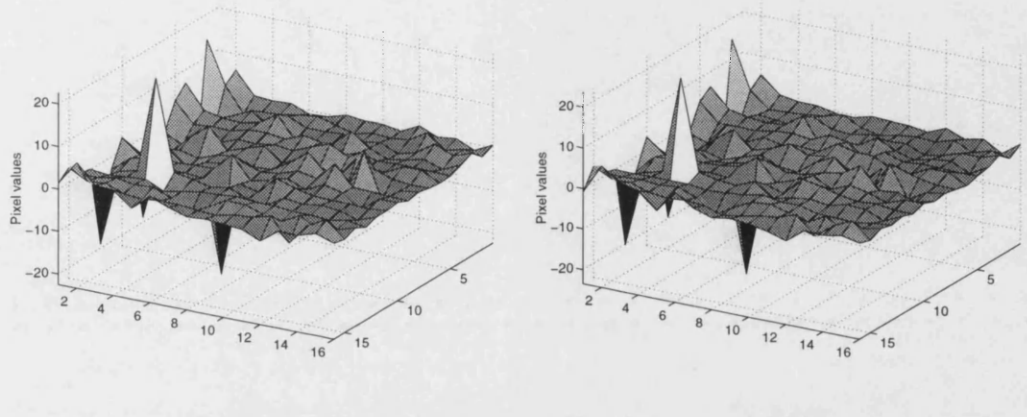


Figure 6.18: Cross-inner product matrix between the input patterns and the filters created using equation 5.55. The graph is shown with the x and y axes reversed for clarity.

filters which are mutually orthogonal to the binary, bipolar patterns in our first training set and compare them to the filters that were created with the SS algorithm. We calculated the cross-inner product matrix between the input patterns and the set of filters created with the matrix method. The three dimensional graph of this matrix can be seen in figure 6.18. This can be compared to figure 6.4 which shows the cross-inner product matrix between the input patterns and the filters that were created using the SS algorithm.

We can see from figure 6.18 that the filters created using equation 5.55 are completely cross-orthogonal to the input patterns as it was expected. The filters that were created using the SS algorithm are almost (figure 6.4) - but not completely - orthogonal and the SS algorithm may converge to the same solution if it is allowed to run for more iterations. To investigate further we looked at the actual filters. An example is shown in figure 6.19, which shows the two versions of filter 2. Subfigure (a) depicts filter 2 created by the SS algorithm and subfigure (b) depicts filter 2 created with the matrix method. The differences between the two filters are plotted in figure 6.20, which shows the filter created with the SS algorithm after we subtracted the filter which was created with the matrix method. We can see from the three graphs that the filters are very similar and it looks like the algorithm given time will converge to exactly the same solution that is obtained with the matrix method.

One might argue at this stage that there is no point in using the SS algorithm



(a) Filter created with the SS algorithm

(b) Filter created with the matrix method

Figure 6.19: Pixel values of the two versions of filter 2

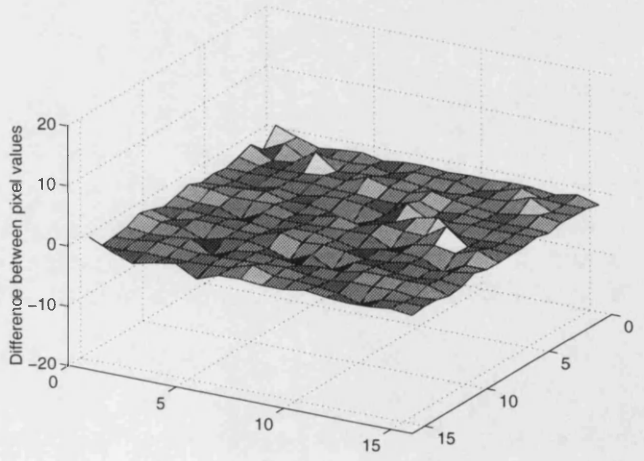


Figure 6.20: Differences between pixel values of the two versions of filter 2

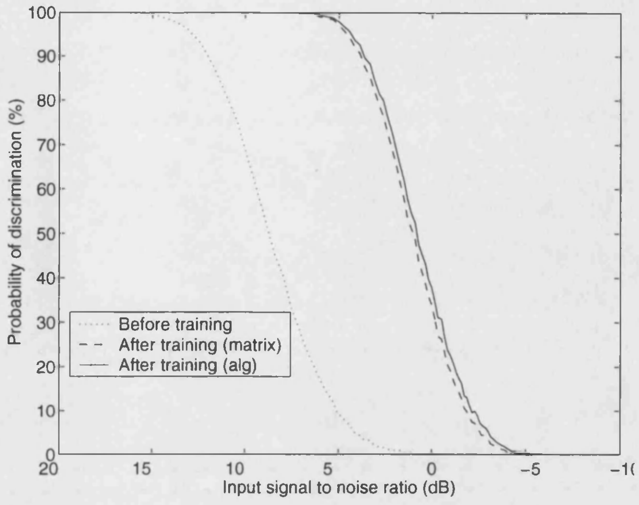


Figure 6.21: Probability of discrimination versus input signal to noise ratio

to create the filters since they can be obtained with fewer calculations, and, therefore, faster from equation 5.55. However, since the filters obtained with the two different methods are not identical, we decided to test the tolerance to input noise and the dynamic range that would be required by an optical system for correct discrimination, when using the second set of filters (the ones calculated with the matrix method). We conducted the same simulations as in the previous section.

The resulting curves for the probability of discrimination using the two filter sets are shown in figure 6.21. Also in the same graph there is a third curve which shows the probability of discrimination before the training. We can see, in that figure, that there is a significant increase in the probability of discrimination after the training whichever of the two sets of filters we use. However, the filters obtained with the SS algorithm are slightly more tolerant to noise.

The plot of the dynamic range versus the signal to noise ratio, again using both filter sets, is shown in figure 6.22. As with the probability of discrimination graph, the dynamic range graph shows us that the filters obtained with the SS algorithm are a bit more (maximum difference between two sets of filters $\sim$1 dB) noise tolerant. Obviously for small amounts of noise the filters obtained with the matrix method (equation 5.55) yield better dynamic range results because they are completely orthogonal to the input patterns. How can these results be explained? It may be that completely cross-orthogonalising the filters to the patterns is not the best solution after all. Maybe the matrix method results in some kind of overfitting to the training data which makes the final filters less able to generalise and,

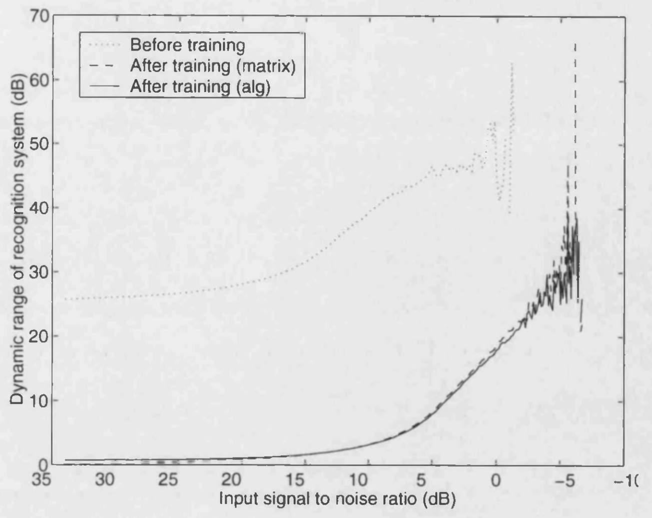


Figure 6.22: Dynamic range of the recognition system as a function of the signal to noise ratio.

therefore, less tolerant to input noise.

6.5 Optimisation of number of iterations for the similarity suppression algorithm

The results shown in the previous section motivated us to investigate the noise tolerance of the various sets of filters obtained when using the SS algorithm and allowing it to run for different numbers of iterations. To do that we used the SS algorithm to train the filters for the binary, bipolar patterns in our first training set and during the training, after each iteration, we calculated the probability of discrimination and the dynamic range required for correct discrimination with the newly produced set of filters. Each time the same amount of random noise was added to the input. As before the noise was analogue, normally distributed, with zero mean and with constant variance equal to 1. The probability of discrimination versus iteration number is shown in figure 6.23. We can see that there is a sharp increase of the probability of discrimination in the first iterations and then the probability of discrimination decreases, until it finally converges to a relatively constant level. The dynamic range required by the optical system for correct discrimination versus iteration number is shown in figure 6.24. As we can see the required dynamic range decreases very quickly and after the first few iterations it converges to a constant level. The first ten points of the two previous

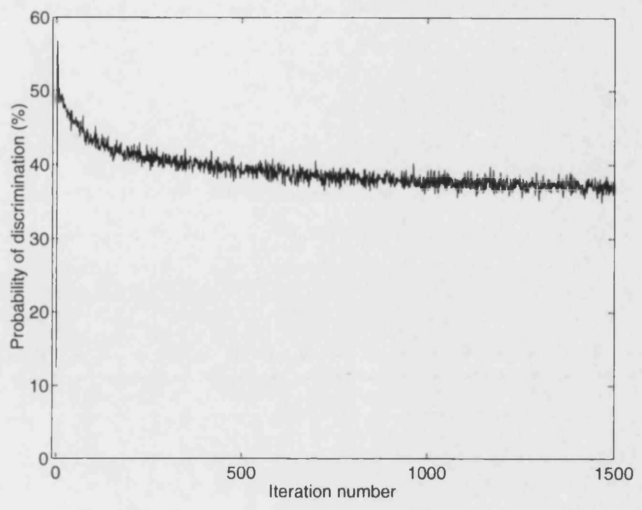


Figure 6.23: Probability of discrimination versus number of iterations.

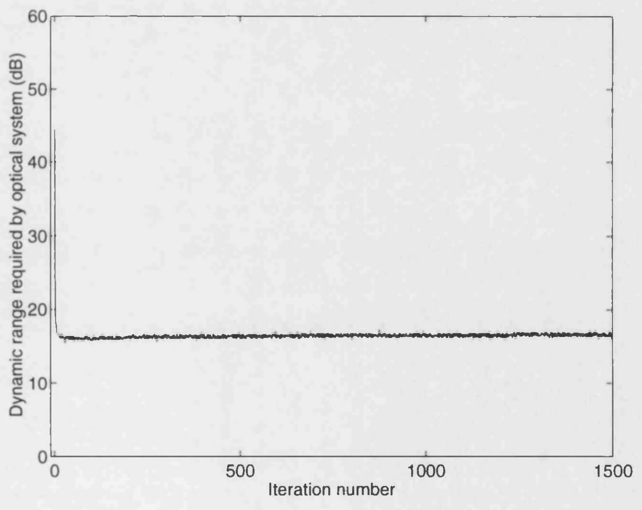


Figure 6.24: Dynamic range of the recognition system versus number of iterations.

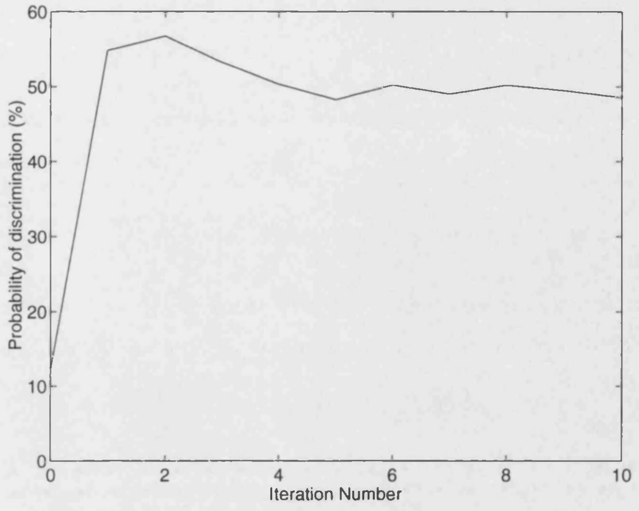


Figure 6.25: Probability of discrimination versus number of iterations for the first 10 iterations.

graphs can be seen in figures 6.25 and 6.26. In figure 6.25 we can see that the the probability of discrimination reaches a maximum at the second iteration and then it decreases. In figure 6.26 we can see that the dynamic range required by an optical recognition system decreases sharply in the first two iterations and then it gradually stabilises.

We then calculated the probability of discrimination and the dynamic range required by the optical system for correct discrimination using the filters obtained after the first few iterations. The corresponding graphs for the probability of discrimination can be seen in figure 6.27. As we can see from the probability of discrimination curves, the filters produced after only 2 or 4 iterations perform slightly worse for a higher signal to noise ratio but as the SNR worsens, these filters perform better than the ones obtained after the algorithm has converged completely (after around 1500 iterations) and better than the ones which are calculated using the matrix method of equation 5.55. In figure 6.28 we have plotted the difference between the probability of discrimination when using the filters produced after 2 iterations of the algorithm and when using the filters produced after 1500 iterations. The other curve in the same graph is the difference between the probability of discrimination when using the filters produced after 2 iterations and the filters produced with the matrix method. We can see in figure 6.28 that the largest benefit, 29%, in using the filters produced after two iterations is with an SNR of about 0 dB. When the SNR is about 6 dB it is better to use

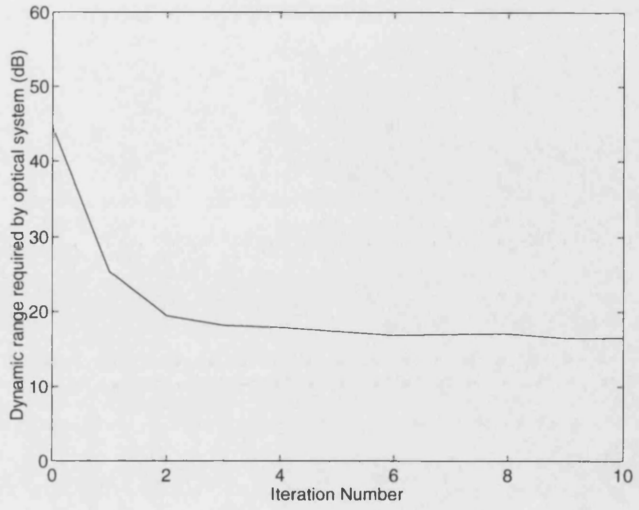


Figure 6.26: Dynamic range of the recognition system versus number of iterations for the first 10 iterations.

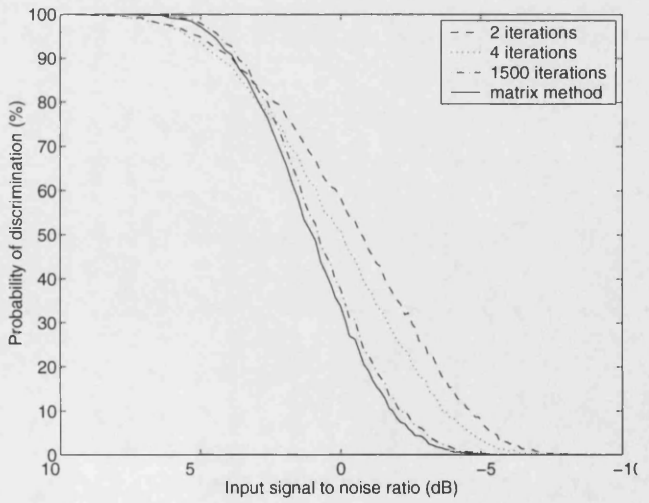


Figure 6.27: Probability of discrimination as a function of the signal to noise ratio.

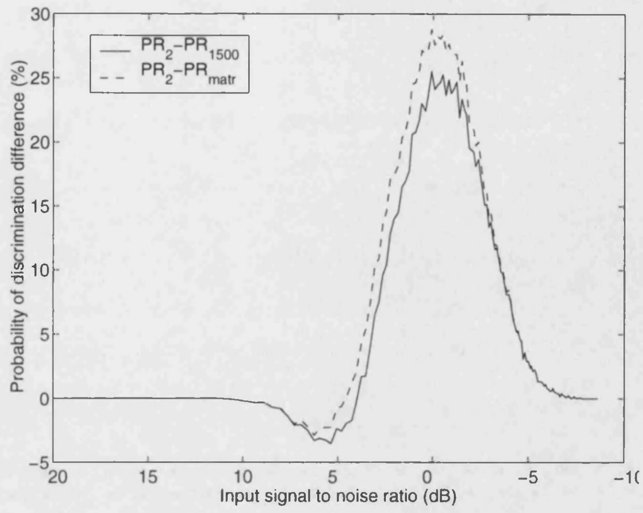
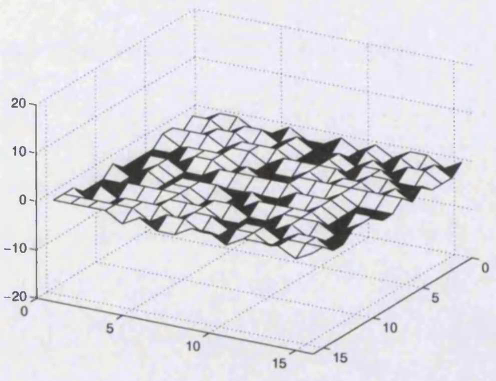


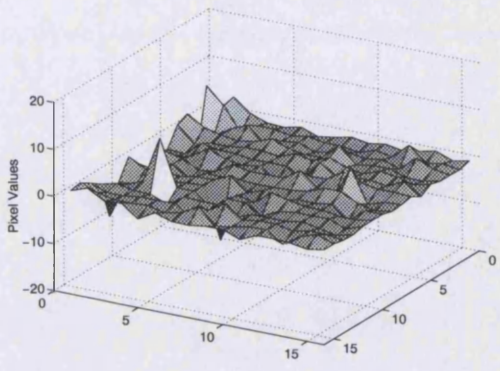
Figure 6.28: Probability of discrimination difference as a function of the signal to noise ratio.

the filters produced after 1500 iterations.

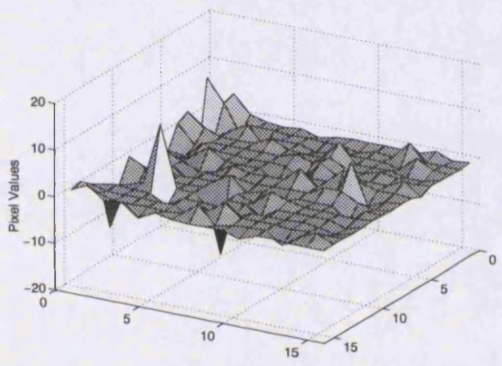
How can this improved performance, in terms of probability of discrimination in the second iteration, be explained? In figure 6.29 we plot the second training pattern and the corresponding filter in the first four and in the final iteration. In addition, in figure 6.30 we plot the differences between the individual pixels of filter 2 in the first four and in the final iteration. We can see in the two figures, that there is a big change in filter 2 in the first iteration (before the training, filter 2 is identical to the second training pattern). Some pixels take large positive or negative values and we can assume that these pixels define the features of this particular training pattern. In the next three iterations there is a steady, gradual enhancement of the same features as we can see both in figure 6.29 and in figure 6.30 where the pixel differences from one iteration to the next are shown. In figure 6.31 we can see the cross-inner product matrix before the training, in the first four iterations and after 1500 iterations. The matrix is depicted from the side to enable us to see the negative cross-inner products. We can see that initially there are cross-inner products with large positive values. In the first iteration the cross-inner products are reduced considerably and some of them increase, but with a negative sign. In the next three iterations, the main feature, apart from the reduction of the positive values, is the increase of the negative cross-inner products. However, all of the cross-inner products, positive and negative have almost disappeared in the final iteration. Although interesting,



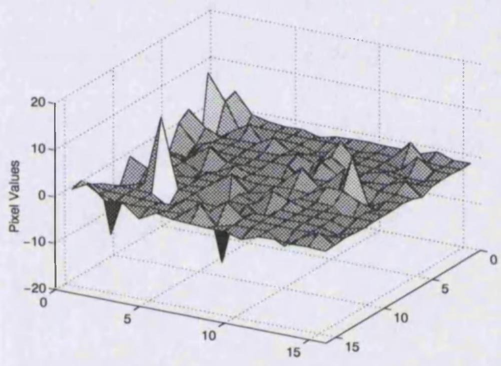
(a) Training pattern 2



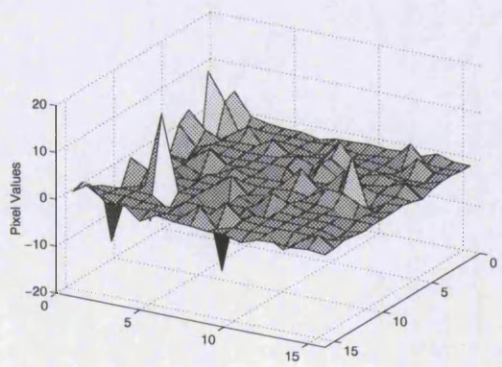
(b) Filter 2 after 1 iteration



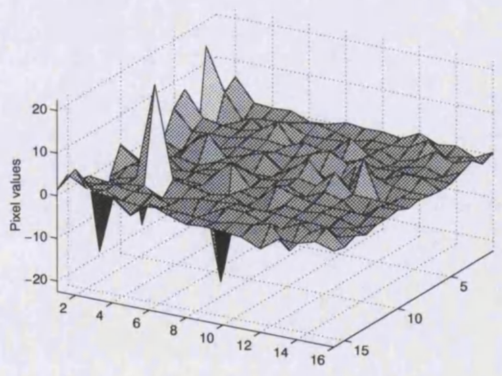
(c) Filter 2 after 2 iterations



(d) Filter 2 after 3 iterations

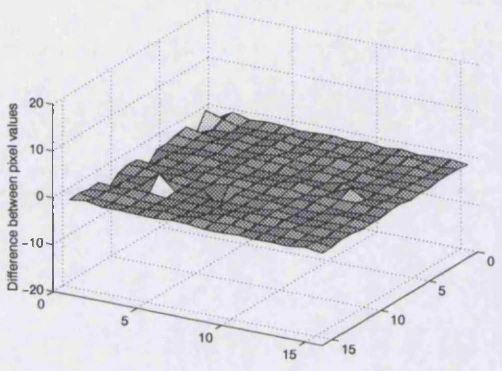


(e) Filter 2 after 4 iterations

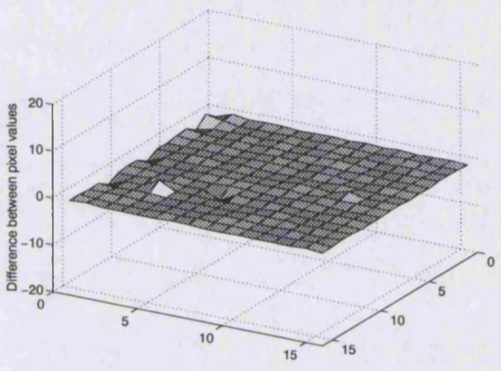


(f) Filter 2 after 1500 iterations

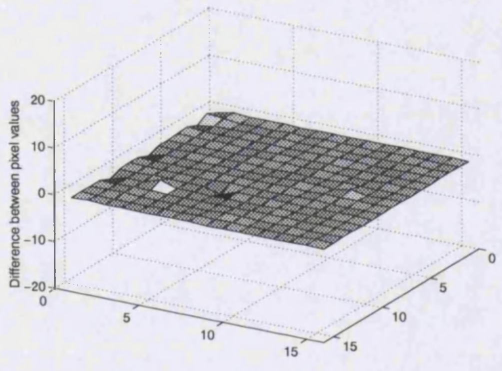
Figure 6.29: Pixel values of pattern 2 and filter 2 in the first 4 and the final iteration.



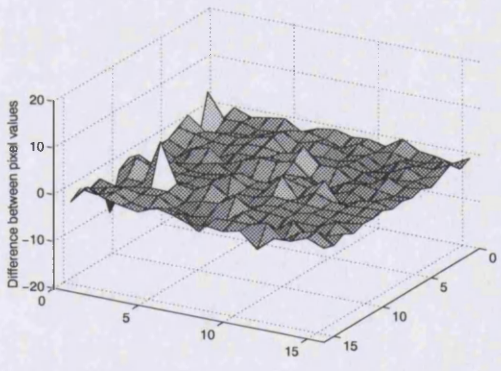
(a) Differences between filter 2 in the 2^{nd} and 1^{st} iterations.



(b) Differences between filter 2 in the 3^{rd} and 2^{nd} iterations.



(c) Differences between filter 2 in the 4^{th} and 3^{rd} iterations.



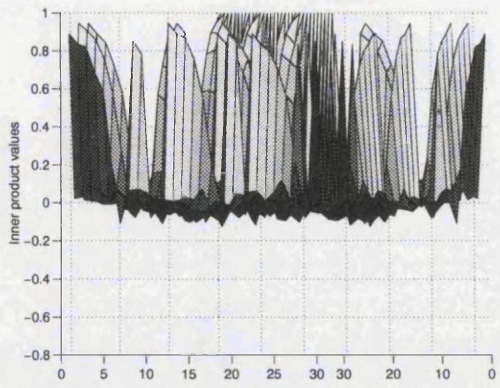
(d) Differences between filter 2 in the 1500^{th} and 2^{nd} iterations.

Figure 6.30: Differences between pixel values of the second filter in various iterations.

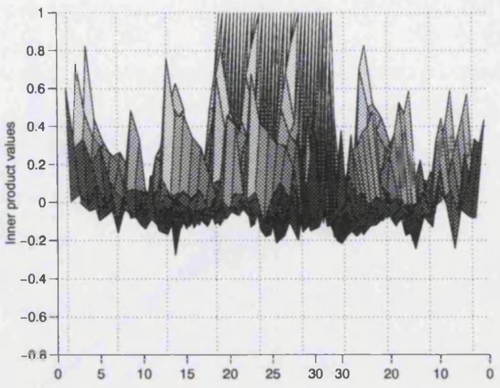
the previous observations do not shed light on the question of why the probability of discrimination is highest in the second iteration and not in the first or the third for example. There are no sharp changes either in the filters themselves, or in the cross-inner product matrix between the first and the third iterations. The negative cross-inner products may play a role but that role is not clear from the data we have here. However, we can explain why the system performs better in the initial iterations compared to the final iterations. Subfigure (b) in figure 6.31 shows that the cross-inner products decrease rapidly from the first iteration. Therefore, we can expect a higher probability of discrimination in the first iteration compared to the untrained filters. Then, as the algorithm converges, the filters become "over-trained" and they are less tolerant to input noise. The data we have here, however, does not help explain why the probability of discrimination maximum occurs in the second iteration and not in the third for example. And although we guess that the same thing will happen with other training sets as well, we have no method of predicting the exact iteration at which the system's performance will be optimised.

The dynamic range curves comparing the performance of the filters after 2, 4 and 1500 iterations with the performance of the filters calculated with the matrix method, are shown in figure 6.32 and are what one would have predicted based on the knowledge gained from the probability of discrimination curves. The filters which are obtained with the matrix method give the lowest required dynamic range for high SNR since they are orthogonal to the input patterns. However, as the SNR decreases the curves meet and at very high noise levels the filters obtained after only 2 or 4 iterations perform slightly better.

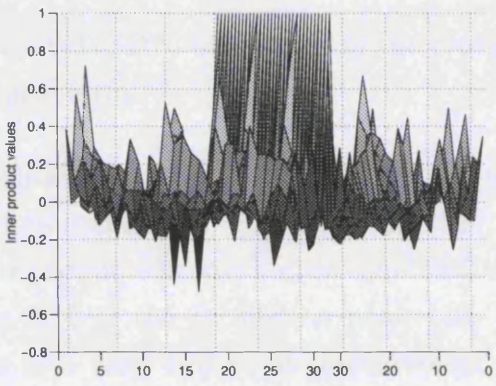
Before we discuss the trade-off between probability of discrimination and dynamic range, we are going to investigate the height of the outer products of the correlations when the 2 iteration filters are used. In figures 6.33, 6.34 and 6.35 we can see the correlation plane intensities for the correlations between some of the input patterns and the filters produced with the SS algorithm after two iterations. In the same figures we have also included the corresponding correlations with the filters that were produced after 1500 iterations of the SS algorithm. We have shown these 1500 iteration graphs before in section 6.2.4 but we plot them again here so that the reader can make a comparison. Specifically, in figure 6.33 we can see the correlations between the first training pattern $\mathbf{s}_1$ and the corresponding filter $\mathbf{g}_1$. Subfigure (a) shows the correlation of $\mathbf{s}_1$ with the filter, $\mathbf{g}_1$, obtained after 1500 iterations and subfigure (b) shows the correlation of $\mathbf{s}_1$ with the filter,



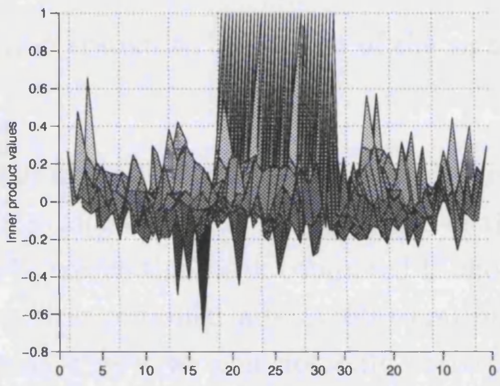
(a) C.i.p. matrix before the training



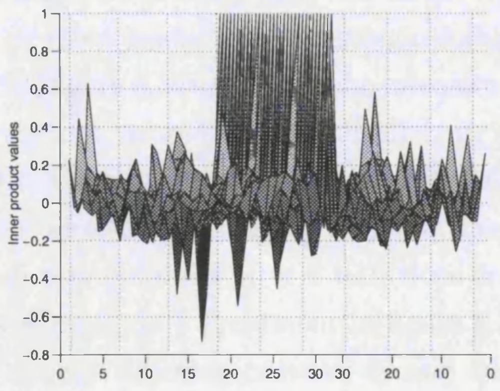
(b) C.i.p. matrix after 1 iteration



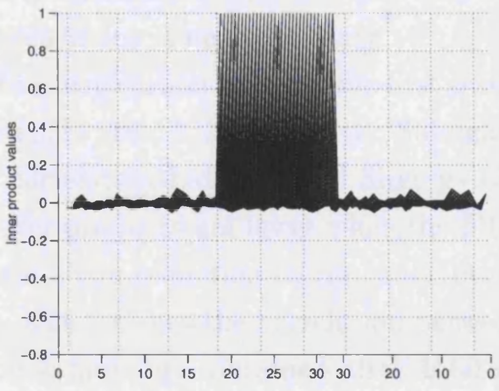
(c) C.i.p. matrix after 2 iterations



(d) C.i.p. matrix after 3 iterations



(e) C.i.p. matrix after 4 iterations



(f) C.i.p. matrix after 1500 iterations

Figure 6.31: Cross-inner product matrix before the training, in the first 4 and in the final iteration. The matrix is depicted from the side. C.i.p.: Cross-inner product.

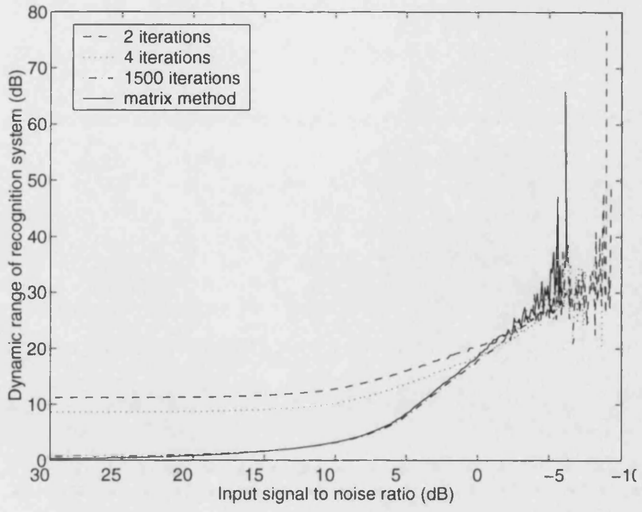
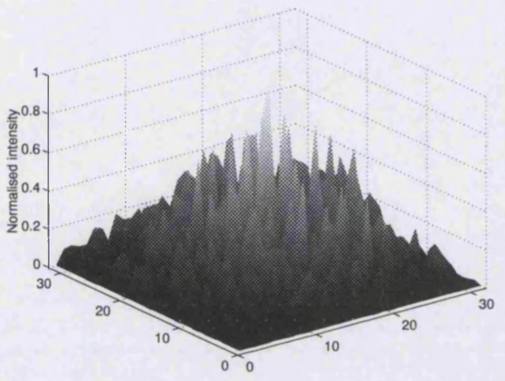


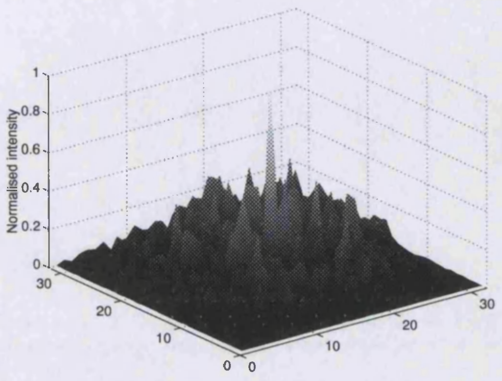
Figure 6.32: Dynamic range of the recognition system as a function of the signal to noise ratio.

$\mathbf{g}_1$, obtained after 2 iterations. We can see in figure 6.33 that the outer products are lower when the filter obtained after 2 iterations is used. In fact, none of the outer products is now higher than 50% of the correlation peak compared to more than 70% of the correlation peak with the filter obtained after 1500 iterations. This is a very important improvement because now we can not only correctly recognise the pattern, but also locate it in the input scene if its exact location is not known. In addition, we can see that the correlation peak is sharp. This is important when more than one target exist in the input scene, in which case the two or more peaks will be distinguishable even if one is near the other.

Subfigure 6.34-(a) shows the intensity of the correlation between the first training pattern, $\mathbf{s}_1$ and the second filter, $\mathbf{g}_2$ obtained after 1500 iterations. Subfigure 6.34-(b) shows the intensity for the same correlation but with the filter $\mathbf{g}_2$ obtained after 2 iterations only. Again, the outer products are lower when the filter which was obtained after 2 iterations is used. The reduction of the outer products is even more prominent in figure 6.35, which shows the correlation between the seventh training pattern, $\mathbf{s}_7$ and the first filter, $\mathbf{g}_1$, obtained after 1500 iterations (subfigure 6.35-(a)) and after 2 iterations (subfigure 6.35-(b)). None of the outer products is higher than 50% of the auto-correlation peak value, when the 2 iteration filters are used, while with the 1500 iteration filters there where outer products which were as high as 80% of the auto-correlation peak value. This reduction of the outer products allows us to use the filters obtained with the SS

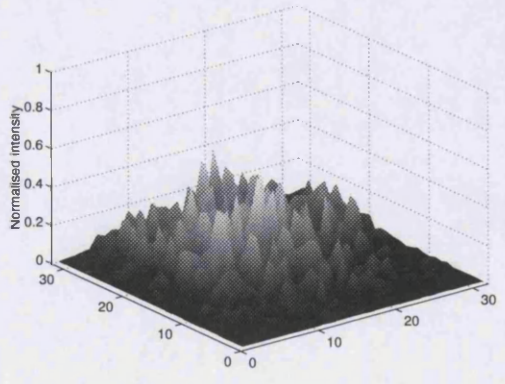


(a) $\mathbf{s}_1 \otimes \mathbf{g}_1$, 1500 iterations, PCE=0.027

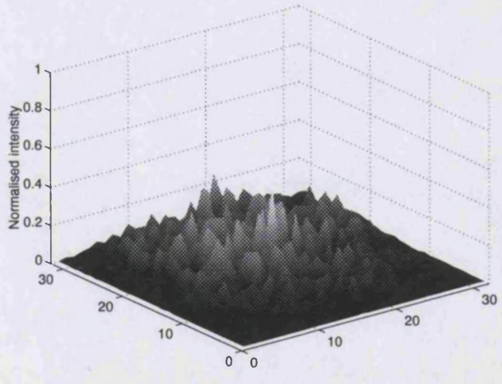


(b) $\mathbf{s}_1 \otimes \mathbf{g}_1$, 2 iterations, PCE=0.1

Figure 6.33: Correlation plane intensity for correlation between pattern 1 and filter 1 after 1500 and after 2 iterations.

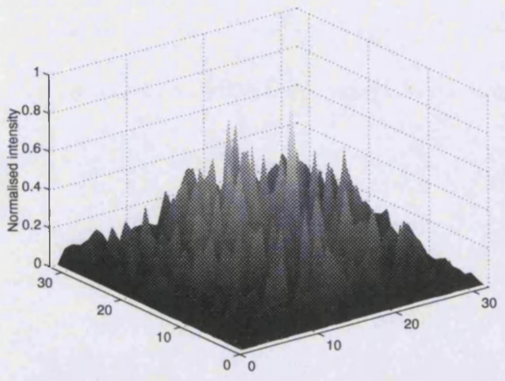


(a) $\mathbf{s}_1 \otimes \mathbf{g}_2$, 1500 iterations

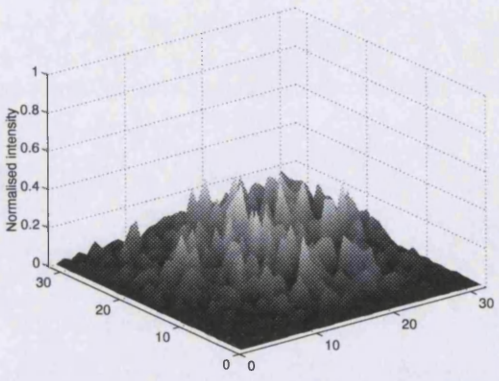


(b) $\mathbf{s}_1 \otimes \mathbf{g}_2$, 2 iterations

Figure 6.34: Correlation plane intensity for correlations between pattern 1 and filter 2 after 1500 and after 2 iterations.



(a) $\mathbf{s}_7 \otimes \mathbf{g}_1$, 1500 iterations



(b) $\mathbf{s}_7 \otimes \mathbf{g}_1$, 2 iterations

Figure 6.35: Correlation plane intensity for correlations between pattern 7 and filter 1 after 1500 and after 2 iterations.

algorithm after 2 iterations to recognise or discriminate input patterns even when the exact location of the object in the input scene is not known, at least when no noise is present in the input.

So from the two graphs, the one for the probability of discrimination (figure 6.25) and the one for the dynamic range (figure 6.26), we can see that there is a trade-off between probability of discrimination and dynamic range. If the dynamic range of the system is absolutely critical, then one can choose to use the filters which are completely cross-orthogonal to the input images, thus minimising the required dynamic range at the expense of probability of discrimination at higher noise levels. In the opposite case when one wants to maximise the probability of discrimination, then the filters obtained after only 2 iterations give the best results of all. Another consideration is the type and amount of noise present. If the main type of noise present is system noise then dynamic range is critical and the filters created with the matrix method may be the best choice. If on the other hand, there is a lot of input noise and not a lot of system noise then one can sacrifice dynamic range for a higher tolerance to input noise which is provided by the filters produced after only 2 iterations. In addition, our final decision of which filter to use must also take account of the height of the outer products. When the exact location of the object in the input scene is not known, it is better to use the filters produced with the SS algorithm after 2 iterations, even if the dynamic range required by the recognition system is higher. We summarise the previous conclusions in table 6.2

SS - LCF COMPARISON							
Use the	H.I.N.	L.I.N.	H.D.R.	L.D.R.	Loc.	High Disc.	High Rec.
SS 2 i.	✓		✓		✓	✓	✓
LCF		✓		✓		✓	

Table 6.2: Comparison between the filters produced with the SS algorithm after 2 iterations and the LCFs. SS 2 i.: SS algorithm 2 iterations, H.I.N.: High input noise, L.I.N.: Low input noise, H.D.R.: High required dynamic range, L.D.R.: Low required dynamic range, Loc.: Location detection, High Disc.: High Discrimination ability, High Rec.: High recognition ability

6.6 Conclusions

In this chapter we presented the computer simulations for the SS algorithm. We started by showing that the algorithm is actually doing what it is intended to do, that is, it is reducing the cross-inner products between filters and patterns that do not correspond to them, while it keeps the auto-inner products stable. We saw that the convergence parameter has a strong influence on the convergence speed and on the final result. In general the algorithm converges when β takes values around 1. However, values a lot smaller than that can make the algorithm too slow, and values a lot larger than that can lead to oscillations. The first few hundreds of iterations are usually enough for the algorithm to converge to a sufficiently good solution. The normalisation step is very important and also has a strong influence on the final result. If it is not used, then the auto-inner products do not remain stable and decrease as the filters are weakened from the continuous subtractions. When the normalisation step is used, the auto-inner products remain stable at the desired level, but some of the filters are strengthened unevenly compared to some others, and as a result there is a bias towards them when random patterns or noise is input into the system. The algorithm also effects the outer products of the correlations between the filters and the patterns. The algorithm places no constraint on them, so they increase as the filters are amplified during normalisation. This has the direct consequence that the PCE of the correlations with the filters decreases, compared to that of the correlations with the initial patterns.

The motivation behind the development of the algorithm was to be able to discriminate amongst similar patterns which were buried in noise. In the second section of this chapter we tested the filters produced by the SS algorithm in discriminating between patterns with additive input noise. The input SNR varied from 20 to -10 dB and we saw that the filters were tolerant to 7 dB more input noise compared to the initial patterns. Another important matter is the dynamic range of the optical system compared to the dynamic range that is required for correct discrimination. Usually after an optical system is built, it's dynamic range is fixed and cannot be increased, as a matter of fact it usually decreases due to dirt, vibrations, etc. Therefore, it is important to minimise the dynamic range that is required for correct discrimination. We saw that when using the filters, the required dynamic range is reduced by as much as 25 dB for that particular training set.

In section 6.4 we verified the conclusions of the theoretical comparison between the SS algorithm and the linear combination filters. The algorithm converges towards the same solution as the one provided with the matrix method. The filters that were produced by the SS algorithm after 1500 iterations were very similar to the filters produced using Caulfield's method. They were not identical, however, and when the two sets of filters were compared with respect to the probability of discrimination and dynamic range, the filters which were produced using the SS algorithm performed slightly better. This led us to think that it may be better not to allow the algorithm to converge fully because in that case the filters may be over-fitted to the patterns in the training set and that may make them less tolerant to noise. In the final section of the chapter we compared the filters produced only after the first few iterations to all of the other filters produced so far. The filters which were produced after only 2 iterations performed a lot better as far as probability of discrimination was concerned, particularly for high additive noise. Of course the required dynamic range when using these filter was higher because they were not allowed to become orthogonal to the training patterns. In addition, the filters produced after 2 iterations produced lower side-lobes compared to the filters produced after 1500 iterations. That allows us to use them even when the exact location of the object in the input scene is not known. One can make a choice of which filters to use, based on the application at hand, the amount of noise in the input and the amount of system noise. If the optical system has a low dynamic range then the filters produced with the matrix method may be the best choice because they are orthogonal to the input patterns and, therefore, require the minimum dynamic range. If, on the other hand, dynamic range is not critical and the inputs are buried in a lot of additive noise, or if the location of the object in the input scene is not known, then the filters produced by the SS algorithm after only 2 iterations are the best choice. For any requirements in between (lower required dynamic range - higher probability of discrimination) one can produce the appropriate filters by stoping the algorithm after a certain number of iterations.

The filters produced by the SS algorithm are very good at recognising the patterns from which they were derived. In many cases, however, two different patterns may represent the same object, for example two photographs of the same person, and the filters produced by the SS algorithm are designed to recognise one of these patterns only and reject all of the others. In the next chapter we present the Feature Enhancement and Similarity Suppression (FESS) algorithm which deals with such cases.

Chapter 7

Feature Enhancement and Similarity Suppression filter design algorithm

7.1 Introduction

In chapters 5 and 6 we developed the similarity suppression algorithm and tested it using computer simulations. Each of the filters designed by the SS algorithm can recognise one specific pattern. In some cases it is necessary to design a filter, which can recognise a group of patterns. In this chapter we introduce the Feature Enhancement and Similarity Suppression (FESS) algorithm which designs such filters. In the next section (section 7.2) we present the motivation for the development of the FESS algorithm and we describe how it is derived from the SS algorithm. Section 7.3 compares the FESS algorithm to other filter design techniques. The final section (section 7.4) of the chapter presents the expansion of the FESS algorithm to more than 1 layer. This whole chapter is a theoretical analysis of the FESS algorithm. The next chapter contains the corresponding computer simulations, which validate the theory.

7.2 Derivation of the Feature Enhancement and Similarity Suppression Algorithm

We will start by defining some terms which are necessary for the description of the FESS algorithm.

- *Class:* A group of objects, which fall under the same category and have some common characteristics. For example: doors belong in a class, chairs in another, people in another, etc. In pattern recognition, a group of patterns, which represent the same thing. For example: different photographs of the same person, different views of an airplane, etc.
- *Class representative filter:* A filter specially designed to recognise patterns, which belong to a specific class.
- *Auto-inner products:* Inner products between the class representative filter and all of the patterns that belong to the corresponding class.
- *Cross-inner products:* Inner products between the class representative filter and all of the patterns that belong to other classes.

Where possible, we are going to use the same notation as in the previous chapters but some additional symbols need to be introduced. All of the training patterns are going to be denoted by $\mathbf{s}_i, i = 1, \dots, M$, where M is the total number of training patterns. N denotes the number of pixels in the patterns and the filters. The filters themselves are denoted by $\mathbf{g}$. K denotes the number of classes. Each class i contains L_i training patterns. Obviously $\sum_{i=1}^{K} L_i = M$. The training patterns will be denoted with a second index on some occasions $\mathbf{s}_{ij}, i = 1, \dots K, j = 1, \dots L_i$. In that case, the first index refers to the class that the pattern belongs to and the second is the pattern number in that class.

7.2.1 Basic algorithm

The SS algorithm designs filters that have a high inner product with only one pattern and are orthogonal to all of the other training patterns. This means that the number of filters is equal to the number of training images. In many real life situations however, a single object can be represented by many slightly, or very, different patterns. For example, a recognition system might need to recognise rotated, scaled or shifted views of an object. There are filters designed specifically for rotation invariance [63, 67, 105] or scale invariance [106, 107], and translation invariance is an inherent characteristic of many optical correlators, but the recognition problem gets a lot more complicated when the three types of distortion are combined. In addition, there are other kinds of distortions which are found in three dimensional objects and which are a lot more difficult to describe and analyse mathematically. A typical example is the human face, which can be distorted in an infinite number of ways due to changes of expression, changes of the physical characteristics, etc. The problem becomes even more complicated when more than one class needs to be recognised. For example, in face recognition each person is a class, which contains many different patterns, which are distorted views of the same face. If we were to design filters for such a recognition problem using the SS algorithm, we would need to create one filter for each possible distorted view of the object that we were trying to recognise. This is not only unfeasible for most real life situations, but also impractical as a very large number of filters would have to be created and every input pattern would have to be compared to each and every one of them.

The ideal solution, provided by the SDF approach [74] (section 3.4), is to create one filter, which correctly recognises all of the input patterns of all classes. This filter has to produce a different output inner product magnitude value for each class. In addition, this output value has to be the same for all of the patterns that belong to a single class. The performance of such a filter depends on several parameters. First of all on the similarity between the patterns of the various classes. It is easier to design a filter which has the same inner product value with several different patterns when these patterns are similar than when these are very different to each other. Secondly, on the number of classes to be recognised and on the dynamic range of the optical system. The dynamic range required by the recognition system increases with the number of classes. The required dynamic range is equal to $-20\log(1/K)$, where K is the number of classes, and is very small for two classes, goes up to 20 dB for 10 classes, 40 dB for 100 classes and so on. Therefore, the actual dynamic range of the recognition system limits the number of classes that can be recognised by one filter. Of course these calculations were done for a completely noiseless system, which is not really feasible. In reality such a filter will not work for more than a few classes (less than 10) and most such filters are designed for two or three classes [77]. Braunecker *et. al.* [75] (section 3.4) proposed the use of $L = \log_2 K$ filters for the recognition of K classes and Mui *et. al.* [108] proposed a technique based on a tree structure where each filter would have to discriminate between two classes. Caulfield proposed the use of one filter for each class [73].

We have chosen to use one filter for each class. The filter only has to distinguish the patterns that belong to the class it represents from all of the other patterns which are members of the other classes. Therefore, only two output values are needed. This minimises the required dynamic range of each detector and the only problem remaining is to design the filter so that it produces the desired outputs with all of the patterns. Of course the number of the filters used can become very large if the number of classes is large but in such cases a single filter would not work anyway. Our technique can be easily modified to design filters that recognise more than one class. However, in this and the following chapter we are going to focus our attention on filters that only recognise one class.

So the filters that we want to design must have the following characteristics:

i. Each class representative filter must have constant, high auto-inner products with all of the patterns in its class.

ii. Each class representative filter must have very low cross-inner products with all of the patterns not in its class.

The SS algorithm can be used to reduce the cross-inner products by subtracting the training patterns, which belong to other classes, from each filter. However, if all of the training patterns in all of the classes are similar to each other, then these subtractions are going to reduce the auto-inner products. We need to modify the algorithm a little so that at the same time it enlarges the auto-inner products. We saw in chapter 5 that cross-inner products can be reduced using weighted subtraction. Based on the same logic we can increase the auto-inner products using weighted addition. The idea is to add at every iteration all of the training patterns in the class to the class representative filter.

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} + \beta' w_{j1}\mathbf{s}_{j1} + \ldots + \beta' w_{jL_j}\mathbf{s}_{jL_j} \tag{7.1}$$

β' is a convergence parameter which we are going to analyse in more detail in the next section. w_{jl} are the weights. The superscript i denotes the iteration number. Our aim when adding the patterns to the class representative filter is to copy their features into it. In order for all of the auto-inner products to become almost equal, the filter must be shifted in pattern-space towards the centre of the area, which is formed by the patterns it represents. In addition, the filter will have the smallest auto-inner products with the training patterns that are most different to it, so their weight in the addition must be the largest. Conversely, if a pattern is already similar to the class representative filter, their inner product will be large and doesn't need to increase any more, so the addition weight needs to be small. We may rewrite equation 7.1 in the following manner

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} + \beta'(P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_1)\mathbf{s}_1 + \ldots + \beta'(P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{L_j})\mathbf{s}_{L_j} \tag{7.2}$$

In equation 7.2 the weights of the additions depend on the similarity between the filter and the pattern. If a pattern is already similar to the filter, the corresponding auto-inner product will be high and the difference from the normalised auto-inner product value, P, will be small. If the pattern is different, the difference between P and the corresponding auto-inner product will be large and the pattern will be amplified before it is added to the class representative filter. If an auto-inner product between the filter and one of the training patterns is negative, then its absolute value will be added to P and, therefore, the pattern will be added to the filter with a strong weight. That will make the filter more similar to the pattern until eventually their inner product becomes positive. So the algorithm can be described by the following equations which are both used at each iteration

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta_2' \sum_{\substack{r=1 \\ r \neq j}}^{K} \sum_{k=1}^{L_r} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{rk} \right\} \mathbf{s}_{rk} \tag{7.3}$$

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} + \beta_1' \sum_{k=1}^{L_j} \left\{ P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk} \right\} \mathbf{s}_{jk} \tag{7.4}$$

where (just a notation reminder) K denotes the total number of classes, L_r denotes the number of training patterns in the r class, L_j denotes the number of training patterns in the j_{th} class and $\mathbf{s}_{jk}$ is the k_{th} training pattern in the j_{th} class. The tilde over the symbol of the new filters shows that the filters are not yet normalised. Equation 7.3 is the SS algorithm applied to all of the patterns of all of the other classes, without the normalisation.

We know from the analysis of the SS algorithm that unless we normalise the filters at every iteration, their magnitudes are going to decrease due to the continuous subtractions. Consequently equation 7.3 will have the same effect on these filters, i.e. their magnitudes are going to decrease. The FESS algorithm, however, adds some patterns to the filters and obviously equation 7.4 will have the opposite effect on the filters, i.e. it increases their magnitudes.

The filter's magnitudes are going to increase or decrease depending on the number of patterns and on the similarity between patterns. Usually, but not always, the patterns that belong to each class will be a subset of the total training set so in most cases the number of patterns that are subtracted from the filter will be larger than the number of patterns that are added to it. In addition, if we assume that at the beginning of the training most of the patterns are similar, then the subtraction weights are going to be larger than the addition weights. Hence the effect of the subtractions will be stronger than that of the additions and the filter's magnitudes are going to decrease[1]. We may be able to alleviate with a careful choice of the two convergence parameters, β_1' and β_2'. In that case, a normalisation of the filters' magnitudes should not be necessary. However, if the filters' magnitudes cannot be stabilised that way, then a normalisation step is necessary after each iteration.

As we said earlier, in the SS algorithm there was a straightforward solution to that problem, and that was to normalise the auto-inner product between the filter and the pattern it represented to the desired value using equation 5.17 which we rewrite here

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \mathbf{s}_j} \tag{7.5}$$

We cannot do the same thing here. If we normalise the auto-inner product of the class representative filter with one of the patterns in the class, then all of the other auto-inner products will be different, usually lower, because if $\mathbf{g}_j \cdot \mathbf{s}_{jk} = P$ then $\mathbf{g}_j \cdot \mathbf{s}_{jl} \neq P$, for $\forall l \neq k$. Another solution is to normalise the inner product between the filter and the mean of all of the patterns in the class:

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{\mathbf{g}_j^{(i-1)} \cdot \bar{\mathbf{s}}_j}{\tilde{\mathbf{g}}_j^{(i)} \cdot \bar{\mathbf{s}}_j} \tag{7.6}$$

where

$$\bar{\mathbf{s}}_j = \frac{1}{L_j} \sum_{k=1}^{L_j} \mathbf{s}_{jk} \tag{7.7}$$

However, if the training patterns are all very similar then by doing that we will probably increase all of the cross-inner products as well. This may happen because the mean pattern $\bar{\mathbf{s}}_j$ may also be similar to patterns which belong to other classes. We chose to normalise the filters to themselves to keep their magnitudes stable and equal to the magnitude of the training patterns, P, using equation 7.8, because by doing that we avoid any bias towards any specific filter.

$$\begin{aligned} \mathbf{g}_j^{(i)} &= \tilde{\mathbf{g}}_j^{(i)} \frac{\|\mathbf{g}_j^{(i-1)}\|}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \\ &= \tilde{\mathbf{g}}_j^{(i)} \frac{P}{\|\tilde{\mathbf{g}}_j^{(i)}\|}, \qquad \text{assuming } \|\mathbf{g}_j^{(i-1)}\| = P \end{aligned} \tag{7.8}$$

By using equation 7.8 together with equations 7.3 and 7.4, we keep all of the filters normalised, and with the additions and the subtractions we move them in

[1]The corresponding mathematical analysis is presented in appendix B

N-dimensional space, until their position is such that the auto-inner products are maximised and the cross-inner products are minimised.

One final issue that needs to be addressed at this stage is the initial value of each of the class representative filters. Each of the filters can initially be one of the patterns that belong to the corresponding class. Or it can be equal to the mean of all of the patterns that belong to the corresponding class. Or it can be random. In the next chapter we are going to compare the results for these initial filter values. So, to summarise, the FESS algorithm is described by the following equations:

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} + \beta_1' \sum_{k=1}^{L_j} \left\{ P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk} \right\} \mathbf{s}_{jk} \tag{7.9}$$

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta_2' \sum_{\substack{r=1 \\ r \neq j}}^{K} \sum_{k=1}^{L_r} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{rk} \right\} \mathbf{s}_{rk} \tag{7.10}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{P}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \tag{7.11}$$

We deliberately wrote equations 7.3 and 7.4 in the opposite order (equations 7.9, 7.10) because we want to point out that the order in which these equations are applied does not matter as long as they are both applied at each iteration before the normalisation.

7.2.2 Advanced algorithm with improved convergence parameters

Based on the analysis presented in section 5.2.3, we chose the following values for the convergence parameter in each equation so that it is inversely proportional to the total number of training patters used in that equation, times the square of the power, P, of the normalised patterns.

$$\beta_1' = \beta_1 \frac{1}{L_j P^2} \left| P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk} \right| \tag{7.12}$$

$$\beta_2' = \beta_2 \frac{1}{(M - L_j) P^2} \left| \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{rk} \right| \tag{7.13}$$

and after inserting the convergence parameters described in the previous equations the final algorithm is:

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} + \frac{\beta_1}{L_j P^2} \sum_{k=1}^{L_j} \left\{ \left(P - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk} \right)^2 \right\} \mathbf{s}_{jk} \tag{7.14}$$

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \frac{\beta_2}{(M - L_j) P^2} \sum_{\substack{r=1 \\ r \neq j}}^{K} \sum_{k=1}^{L_r} \left\{ \pm \left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{rk} \right)^2 \right\} \mathbf{s}_{rk} \tag{7.15}$$

$$\mathbf{g}_j^{(i)} = \tilde{\mathbf{g}}_j^{(i)} \frac{P}{\|\tilde{\mathbf{g}}_j^{(i)}\|} \tag{7.16}$$

In the $\pm$ sign in equation 7.15, the plus sign is used when $\left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk}\right) > 0$ and the minus sign is used when $\left(\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{jk}\right) < 0$. The parameters β_1 and β_2 take small values around 1.

7.3 Comparison of the FESS algorithm with relevant filter design and neural network training algorithms

In the previous section (section 7.2) we described the FESS algorithm. In this section we are going to compare the FESS algorithm with some relevant filter design techniques and neural network training algorithms. These comparisons are going to help us study the relationship between the FESS and other algorithms and find its advantages and disadvantages. Our aim is to gain a better insight and improve our algorithm.

7.3.1 Comparison of the FESS algorithm with the Similarity Suppression Algorithm

We have already described how the FESS algorithm is an extension of the SS algorithm. To compare the FESS algorithm with the SS algorithm we are first going to combine the two equations describing the FESS algorithm (7.3 and 7.4) into one, equation 7.17. It can be written as follows:

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta \sum_{k=1}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k - d_{jk} \right\} \mathbf{s}_k \tag{7.17}$$

where β is the convergence parameter, and d_{jk} is the desired value for each inner product. If we substitute $d_{jk} = P$ for the training patterns $\mathbf{s}_k$ that belong to class

j and $d_{jk} = 0$ for the training patterns $\mathbf{s}_k$ that don't belong to class j, equation 7.17 can be split back to equations 7.3 and 7.4.

Equation 7.17 is a supervised version of the SS algorithm (equation 5.19), which we rewrite here

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{7.18}$$

This can be easily seen if we consider that in the SS algorithm the desired value for all of the cross inner products ($k \neq j$ in equation 7.17) is zero. Therefore, $\forall k \neq j$ equation 7.17 becomes

$$\mathbf{g}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{7.19}$$

which is the same as equation 7.18. When $k = j$ in equation 7.17, the pattern $\mathbf{s}_k$ which is subtracted is the one that corresponds to the filter $\mathbf{g}_j$ and in that case the desired value for the auto-inner product is equal to P. However, the auto-inner product $\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{k=j}$ is already equal to P because the normalisation equation (in the SS algorithm) set it to that value in the previous iteration. Therefore, the whole term $\mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_{k=j} - d_{jk}$ is equal to zero and the pattern $\mathbf{s}_{k=j}$ is not subtracted from the filter $\mathbf{g}_j$. There are, however, some differences between the two algorithms:

- The initial filters for the SS algorithm are the training patterns. For the FESS algorithm the initial filters can be one of the training patterns of each class, or the average of the training patterns of each class, or random. In the first case the choice of which pattern to use as the initial filter for each class, could be random. The second choice at least ensures that the initial filter is going to contain the features of all of the patterns in the class. However, if the training patterns are very similar, there is not much difference between the two cases.

- Both algorithms move the filters in N-dimensional space until the constraints that they impose are satisfied. These constraints are different for each algorithm. The FESS algorithm imposes the constraint that the filter must contain all of the features of the patterns of the class it represents. It is this constraint that forces the auto-inner products to converge to the desired value and not the normalisation step. The SS algorithm forces each of the filters to become orthogonal to all of the other training patterns, the ones it does not represent. Initially each of the filters is made identical to the pattern it represents. After that, however, the SS algorithm equations do not force any of the filters to keep any of the features of the pattern they represent. The normalisation step ensures that the auto-inner product between the filter and the pattern it represents has the correct value. This indicates that the filter that is created by the SS algorithm does not *need* to contain any features of the pattern it represents. However, we have not yet conducted any simulations which prove the previous argument.

- The SS algorithm normalises the inner product between the filter and the pattern it represents. The FESS algorithm normalises the filter itself to the mean power of all of the training patterns.

The FESS algorithm, like the SS algorithm, can cross-orthogonalise the filters to the training patterns that belong to other classes. Since we are normalising the magnitude of each of the filters to the normalised power of all of the training patterns, P, that is $\mathbf{g}_j \cdot \mathbf{g}_j = P, \forall j$, it is not possible for the algorithm to force all of the auto-inner products to converge to P. They will converge to a value which is lower than P, because if $\mathbf{g}_j \cdot \mathbf{g}_j = P, \forall j$ and $\mathbf{s}_k \cdot \mathbf{s}_k = P, \forall k$, then $\mathbf{g}_j \cdot \mathbf{s}_k < P, \forall j, k$. This means that a higher dynamic range will be required by the recognition system compared to the dynamic range that is required when the SS algorithm is used. The advantage when using the FESS algorithm is the lower number of filters necessary for recognition.

7.3.2 Comparison of the FESS algorithm with Synthetic Discriminant Functions

We saw in section 5.4.2 that the SS algorithm converges to the same solution that is provided by the method proposed by Caulfield and Maloney for designing mutually orthogonal linear combination filters. In this section we are going to investigate the relationship between the FESS algorithm and synthetic discriminant functions. The FESS algorithm can be seen as a more general version of the SS algorithm and following a similar analysis to the one we followed for the SS algorithm we can derive the filters by solving a similar set of linear equations:

$$\begin{aligned} C_{11}\mathbf{s}_1 + C_{12}\mathbf{s}_2 + \cdots + C_{1M}\mathbf{s}_M &= \mathbf{g}_1 \\ C_{21}\mathbf{s}_1 + C_{22}\mathbf{s}_2 + \cdots + C_{2M}\mathbf{s}_M &= \mathbf{g}_2 \\ &\vdots \\ C_{K1}\mathbf{s}_1 + C_{K2}\mathbf{s}_2 + \cdots + C_{KM}\mathbf{s}_M &= \mathbf{g}_K \end{aligned} \tag{7.20}$$

where M is the total number of training patterns, K is the number of classes and consequently filters, and each of the coefficients $C_{11}, C_{12}, \ldots, C_{KM}$ is equal to the sum of all of the individual weights that were used for the addition or subtraction of each of the training images during the training. The constraints that the FESS algorithm imposes on these filters are the following:

$$\begin{aligned} \mathbf{g}_j \cdot \mathbf{s}_i &= P \qquad \text{if } \mathbf{s}_i \in \text{class } j, \\ \mathbf{g}_j \cdot \mathbf{s}_i &= 0 \qquad \text{if } \mathbf{s}_i \notin \text{class } j \end{aligned} \tag{7.21}$$

Equations 7.21 can be written in a matrix form as follows:

$$\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_M \end{pmatrix} \cdot \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_K \end{pmatrix} = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1K} \\ d_{21} & d_{22} & \cdots & d_{2K} \\ & \vdots & & \\ d_{M1} & d_{M2} & \cdots & d_{MK} \end{pmatrix} \tag{7.22}$$

$$\tag{7.23}$$

or

$$\mathbf{S}\mathbf{G}^T = \mathbf{D} \tag{7.24}$$

where $\mathbf{S}$ is a $M \times 1$ vector whose elements are the training patterns $\mathbf{s}$, $\mathbf{G}$ is a $K \times 1$ vector whose elements are the class representative filters $\mathbf{g}$ and $\mathbf{D}$ is a $M \times K$ matrix whose elements d_{ij} are equal to the desired values of the inner products between pattern $\mathbf{s}_i$ and filter $\mathbf{g}_j$. These values are subject to the constraints shown in equations 7.21. The set of equations 7.20 can also be written in a matrix form

$$\mathbf{C}\mathbf{S} = \mathbf{G} \tag{7.25}$$

where $\mathbf{C}$ is a $K \times M$ matrix whose each element is the corresponding coefficient C_{ij}. From equations 7.24 and 7.25 we get

$$\mathbf{C} = \mathbf{D}^T\mathbf{R}^{-1T} \tag{7.26}$$

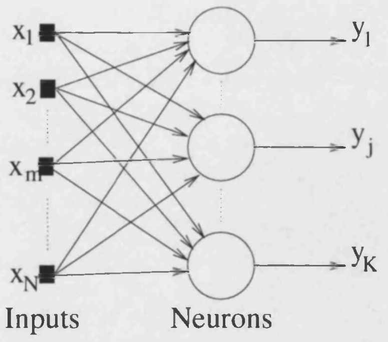


Figure 7.1: A single layer of neurons.

where $\mathbf{R} = \mathbf{S}\mathbf{S}^T$ is the $M \times M$ vector-inner product matrix of the training patterns. The final class representative filters can be calculated using equations 7.25 and 7.26

$$\mathbf{G} = \mathbf{D}^T\mathbf{R}^{-1T}\mathbf{S} \tag{7.27}$$

Equation 7.26 is very similar to equation 3.18 which describes the SDF approach and we rewrite here:

$$\begin{aligned} \mathbf{R}\mathbf{a}_i &= \mathbf{d}_i \\ \Rightarrow \mathbf{a}_i &= \mathbf{R}^{-1}\mathbf{d}_i \end{aligned} \tag{7.28}$$

The main difference is that with the SDF method, one filter is synthesised, while our method creates a number of filters equal to the number of classes. However, our method reduces to the SDF if one filter is created for all of the classes. In addition, the FESS algorithm is iterative while the SDF method is not. Equations 7.26 and 7.28 however, show that if the number of filters (=1) and the desired correlation peak values are the same, the FESS algorithm will finally converge to the solution given by the SDF method. The squares in equations 7.14 and 7.15 can be included in the coefficients C_{ij} without changing the previous results.

7.3.3 Comparison of the FESS algorithm with the supervised Hebbian law

In this section we compare the FESS algorithm with the supervised Hebbian law, which is also called the Widrow-Hoff rule or the delta rule and is described by equation 4.4 which we rewrite here

$$\Delta w_{jm} = \eta(d_j - y_j)x_m, \qquad 0 < \eta \leq 1 \tag{7.29}$$

where (see figure 7.1) Δw_{jm} is the change applied to the weight w_{jm} between neuron j and the input m, d_j is the target value for the output of neuron j, y_j, and x_m is the value of input m. Using our notation we can rewrite equation 7.29 in the following manner:

$$\Delta g_{jm} = \eta(d_j - y_j)s_m \tag{7.30}$$

and considering that $y_j = \mathbf{g}_j \cdot \mathbf{s}$ (equation 5.67), the weight update for the whole weight vector (or filter) $\mathbf{g}$ becomes

$$\Delta \mathbf{g}_j = \eta(d_j - \mathbf{g}_j \cdot \mathbf{s})\mathbf{s} \tag{7.31}$$

In the batch mode of training, the weight update is described by

$$\Delta w_{jm} = \eta \sum_{k=1}^{M}(d_{kj} - y_{kj})x_{km}, \qquad 0 < \eta \leq 1 \tag{7.32}$$

In our notation and for the whole weight vector equation 7.32 is written

$$\Delta \mathbf{g}_j = \eta \sum_{k=1}^{M} \left\{ d_{jk} - \mathbf{g}_j \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{7.33}$$

The weight update described in equation 7.33 is identical to that given in equation 7.17 which describes the FESS algorithm:

$$\begin{aligned} \mathbf{g}_j^{(i)} &= \mathbf{g}_j^{(i-1)} + \beta \sum_{k=1}^{M} \left\{ d_{jk} - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \\ \Rightarrow \Delta \mathbf{g}_j &= \beta \sum_{k=1}^{M} \left\{ d_{jk} - \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \end{aligned} \tag{7.34}$$

since η and β are both convergence parameters. The two algorithms are equivalent and will create the same filters if the same target values are given. The only difference between them stems from the manner of presentation. The Hebbian law refers to individual weights. The FESS algorithm refers to whole images.

7.4 Extension of the FESS algorithm to two or more consecutive banks of correlators

In section 5.5 we used the insight gained in the previous sections (5.4.3 and 5.4.4 to derive a SS algorithm which created filters for two or more cascaded banks of correlators. In that derivation we included the desired values so the derivation

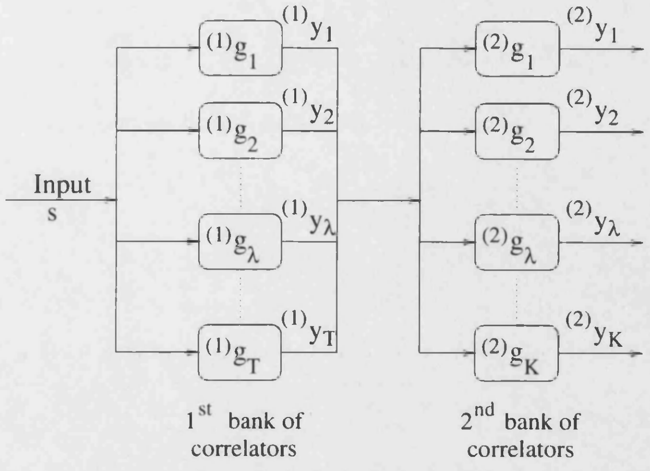


Figure 7.2: Two cascaded banks of correlators.

applies for the FESS algorithm with the following minor changes. In section 5.5 we assumed that the number of filters in the output bank was equal to the number of training patterns. This is not now the case. We now want to design one filter for each class so the number of filters in the output bank of correlators will be equal to the number of classes, K.

In figure 7.2 we can see two cascaded banks of correlators. There are T correlators in the first bank and K in the second. The same pattern $\mathbf{s}$ is input to all of the correlators in the first bank. The input to the correlators in the second bank, which is the same for all of them, is formed by the outputs of the correlators in the first bank. The correlation peak of each one of them, after the activation function is applied to it, corresponds to one pixel of the pattern which is input to the correlators in the second bank. Therefore, the filters in the first bank are of size N, where N is the size of the input patterns and the filters in the second bank are of size T, where T is the number of correlators in the first bank. The non-linear activation functions are not shown in figure 7.2.

The average squared error is given by the following equation [95]

$$E_{av} = \frac{1}{2M} \sum_{k=1}^{M} \sum_{\lambda=1}^{K} {}^{(2)}e_{\lambda k}^{2} \qquad (7.35)$$

where λ indicates the filter number in the output bank and k indicates the training pattern number. The derivation of the filter update equations is exactly the same as in section 5.5 from this point forward and will not be repeated here. The final filter update equation for the output bank is:

$$\Delta\, {}^{(2)}\mathbf{g}_j = \frac{\eta}{M}\sum_{k=1}^{M}\left\{ {}^{(2)}d_{jk} - {}^{(2)}\phi_j({}^{(2)}\mathbf{g}_j^{(i-1)} \cdot {}^{(1)}\mathbf{y}_k)\right\} {}^{(2)}\phi'_j({}^{(2)}\mathbf{g}_j^{(i-1)} \cdot {}^{(1)}\mathbf{y}_k)\, {}^{(1)}\mathbf{y}_k \tag{7.36}$$

and for the hidden bank the filter update equation is:

$$\Delta\, {}^{(1)}\mathbf{g}_j = \frac{\eta}{M^2}\sum_{k'=1}^{M}\sum_{k=1}^{M}\sum_{\lambda=1}^{K} A \tag{7.37}$$

where

$$A = \left\{ {}^{(2)}d_{\lambda k} - {}^{(2)}\phi_\lambda({}^{(2)}\mathbf{g}_\lambda \cdot {}^{(1)}\mathbf{y}_k)\right\} {}^{(2)}\phi'_\lambda({}^{(2)}\mathbf{g}_\lambda \cdot {}^{(1)}\mathbf{y}_k)\, {}^{(2)}g_{\lambda j}\, {}^{(1)}\phi'_j({}^{(1)}\mathbf{g}_j \cdot \mathbf{s}_{k'})\mathbf{s}_{k'} \tag{7.38}$$

Equation 7.36, which describes how the filters in the output bank of correlators must be updated, is very similar to equation 7.17 which describes the FESS algorithm. The only difference between the two equations is that equation 7.36 is more general and takes into account the non-linear activation functions that may exist after each of the correlators. It trains the filters in the second bank to recognise the output produced by the correlators in the hidden bank. Equation 7.37 updates the filters in the first bank of correlators based on the error of the correlators in the output bank. The two equations (7.36 and 7.37) enable us to create filters which can be used in two consecutive banks of correlators. These are mathematically equivalent to a 2-layer perceptron and, therefore, can be used to recognise patterns which are not linearly separable [3], although they cannot solve all solvable problems.

7.5 Discussion and conclusions

In this chapter we have designed filters for multi-class pattern recognition. In multi-class pattern recognition the task is to design one or more filters, which can discriminate one class from another. We developed an algorithm, which we called the Feature Enhancement and Similarity Suppression (FESS) algorithm. We used it to design filters that can discriminate each class from all of the others. This means that the number of necessary correlations is equal to the number of classes. However, the algorithm can also design filters, which recognise more than one class if needed. The FESS algorithm is iterative and is based on the SS algorithm. It uses weighted additions to combine the features of all of the patterns that belong to one class into the class representative filter. As in the SS algorithm, weighted subtractions are used to orthogonalise each class representative filter to the patterns that belong to other classes. In addition, at each iteration the filters are normalised so that their magnitudes remain stable. This normalisation ensures that no bias towards any of the filters will exist if random patterns are input.

As we already said, the normalisation keeps the magnitudes of the filters stable throughout the training. However, it does not set the auto-inner products of the filter with the patterns it represents to a specific value. These are forced to converge towards the desired value by the addition equation (7.4). Since we are normalising each of the filters to the power of the normalised training patterns, P, it is not possible to set all of it's auto-inner products equal to the same value, P, which is the desired one. So we expect them to be lower than P, but higher than the cross-inner products, which will converge to zero. The difference between the value that the auto-inner products will converge to, and the value that the cross-inner products will converge to, will define the dynamic range that will be required by the optical system for correct recognition.

The third section of the chapter presented the comparisons between the SS algorithm and relevant filter design techniques. The FESS algorithm can be seen as a supervised version of the SS algorithm. The SS algorithm forces the cross-inner products to decrease. It does not copy the features of the training pattern, that the filter represents, to the filter at every iteration. The normalisation step ensures that the auto-inner product will take the desired value. The filters that are created by the SS algorithm, yield exactly the desired value for the auto-inner product and very close to the desired values for the cross-inner products. They are very good at discriminating but cannot generalise and that is a necessary attribute for recognising classes of patterns. The FESS algorithm places the additional constraint that the class representative filter must contain all of the features of the patterns that belong to that class. It normalises the filter itself and not an inner product between the filter and one of the patterns. The filter definitely will not be able to discriminate between individual patterns within a class, which is not what we want anyway, but it will be able to generalise and recognise all of the patterns that belong to the class, even the ones that were not used in the training, provided that the training patterns span the class space.

We transformed the algorithm equations into a matrix form and saw that the FESS algorithm is very similar mathematically to the synthetic discriminant functions (SDF) approach. The SDF method designs only one filter for all of the classes, and the FESS algorithm can do that without modifications.

We have also shown that the algorithm is mathematically equivalent to the supervised Hebbian law, also known as the Widrow-Hoff rule. By doing that we have clarified the relationship between neural networks and optical correlators. Each neuron in a layer of a neural network corresponds to a correlator in a bank of correlators. The inner product between the input and the filter is equal to the internal activity level of the neuron. Neural network training algorithms like the Hebbian law or the back-error propagation algorithm change the weights of the neurons. Each individual weight corresponds to a pixel of the digitised filter. By using a simple change of notation we can rewrite these neural network training algorithms in such a way so that they refer to whole images and use them to create filters for optical correlators. The equivalence between the FESS algorithm and the supervised Hebbian law and between the FESS algorithm and the SDF method suggests that the SDF method can be used to calculate the weights of a single layer neural network without the need for an iterative procedure. Finally, like the SS algorithm, the FESS algorithm can be extended to design filters for 2 or more cascaded banks of correlators, which compared to a single bank of correlators, have the advantage that they can be used to recognise patterns that are not-linearly separable.

This chapter presented the development and the theoretical analysis of the FESS algorithm. The computer simulations, which verify our theoretical conclusions for the algorithm will be presented in the following chapter.

Chapter 8

Computer simulations of the FESS algorithm

8.1 Introduction

In the previous chapter we presented the theoretical analysis of the FESS algorithm. In this chapter we describe computer simulations, which helped us assess the performance of the FESS algorithm. In the second section (section 8.2) we use the algorithm to create filters to recognise a set of faces, which is a typical problem of multi-class pattern recognition. As we saw in the previous chapter, the initial filters before the training can be random, or equal to the mean of all of the training patterns within their class, or equal to just one training pattern. Here we describe the results of the training with all of the different initial values for the filters. In addition, we show the effect of the algorithm on the auto- and cross inner products and also on the outer products of the correlations between the filters and the training patterns. In section 8.3 we calculate the probability of recognition, false positives and false negatives and the dynamic range required by the optical system for the training set and for a test set. We finish the chapter with the conclusions.

8.2 Computer Simulations

In this section we evaluate the performance of the FESS algorithm during the training phase, to see whether the algorithm converges to the desired solution, how many iterations it takes to do that, which is the best choice for the convergence parameter and which initial filter values lead to the best performance after the training. We define the following performance metric, which will help evaluate the convergence of the FESS algorithm:

- Energy ratio

  A term which is equal to the ratio of the normalised sum of the auto-inner products to the normalised sum of the cross-inner products of all of the filters and is described by the following equation:

$$\begin{aligned} r &= \frac{\text{normalised sum of all of the auto-inner products}}{\text{normalised sum of all of the cross-inner products}} \\ &= \frac{\sum_{i=1}^{K}\sum_{j=1}^{L_i}|\mathbf{g}_i\cdot\mathbf{s}_{ij}|/M}{\sum_{i=1}^{K}\sum_{\substack{k=1\\k\neq i}}^{K}\sum_{j=1}^{L_k}|\mathbf{g}_i\cdot\mathbf{s}_{kj}|/(M(K-1))} \end{aligned} \tag{8.1}$$

The energy ratio gives us a measure of how much the auto-inner products increase in comparison to the cross-inner products. We expect it to increase as the algorithm converges.

8.2.1 Training set description

We used the algorithm to create filters for face recognition. Face recognition is one of the typical problems the algorithm is designed to tackle, because many different patterns can all represent the same person, in other words belong to the same class, and one filter has to be designed to recognise all of them. Each person's face can be distorted in many different ways. In addition to in and out of plane rotations, translation and scale variations, facial distortions also include changes of expression, elastic distortions and changes in the facial characteristics due to ageing, fattening etc. All these types of distortions are very difficult to express mathematically. The training set[1] we used was part of the Olivetti Research Laboratories faces database and it consisted of faces of ten people. Each person was represented by six photographs. So there were sixty images in total in the training set. For some of the subjects, the images (see figure 8.1 for a sample of the training set) were taken at different times, with a slightly varying lighting, different facial expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no-glasses). All of the images were taken against homogeneous backgrounds and the subjects were in upright, frontal position (with tolerance for some side movement). The images were grey level and each pixel had an integer value between 1 and 256. There are, in theory, two ways to represent these images

[1]See appendix C for the complete training set.

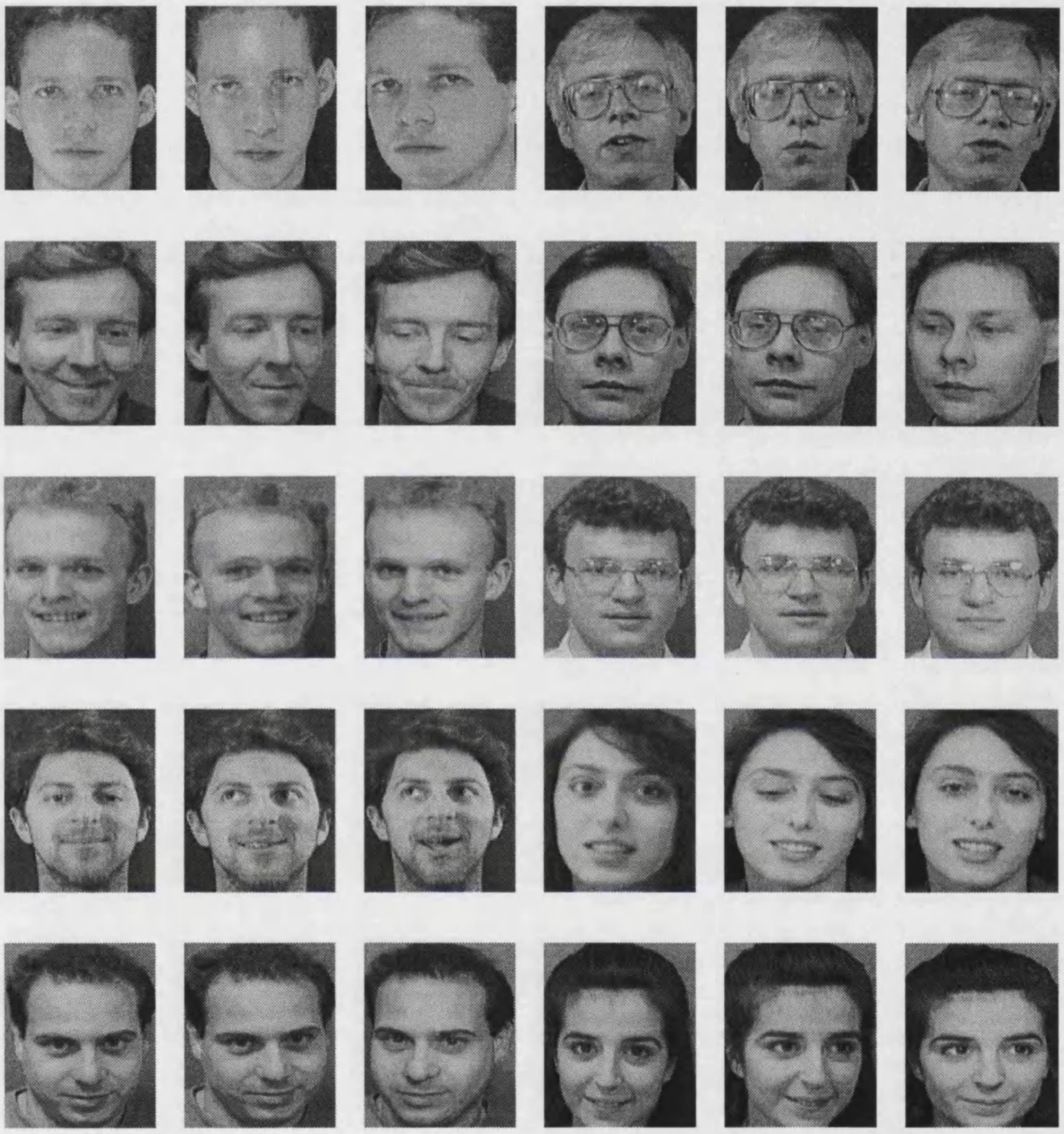


Figure 8.1: A sample of the training set, which consists of six pictures of each person. Only three examples of each subject are shown in this figure.

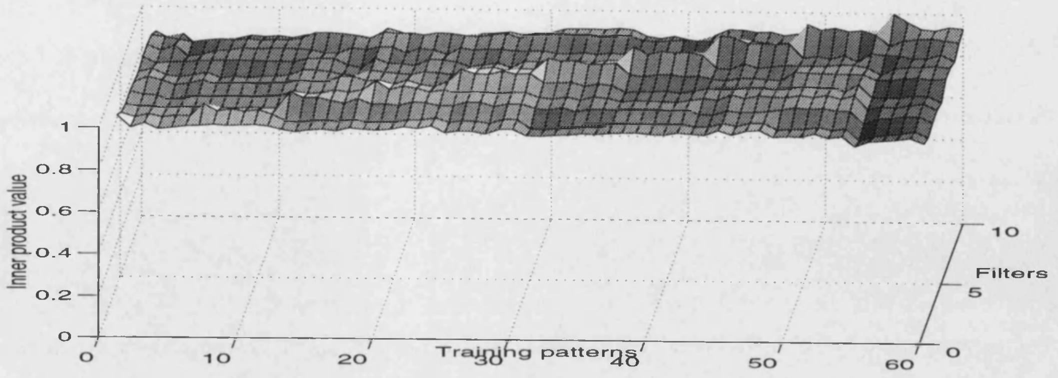


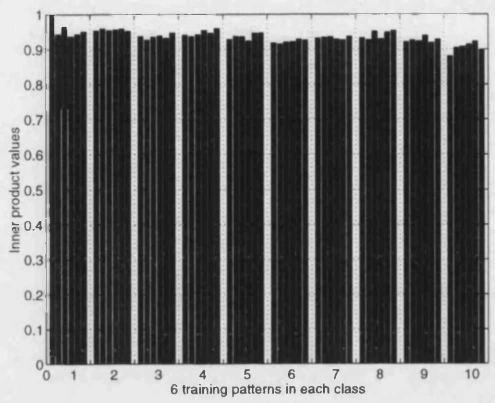
Figure 8.2: Cross-inner product matrix for the monopolar patterns before the training. The filters are initially equal to the first pattern of each of the classes.

in an optical system using SLMs. One can use a multi-level amplitude or phase SLM with 256 levels and represent the images in intensity or phase. Or, two SLMs can be used, one amplitude and one phase SLM and the pixels can be represented with values between -127 and 128, using a binary phase SLM to represent the sign and the grey level amplitude SLM to represent pixel values. Of course the first way which needs only one SLM is easier to implement, but in this chapter we will present simulations for both optical representations. We will refer to the patterns whose individual pixels have values between 1 and 256 as monopolar patterns and to the patterns whose pixels have values between -127 and 128 as bipolar patterns.

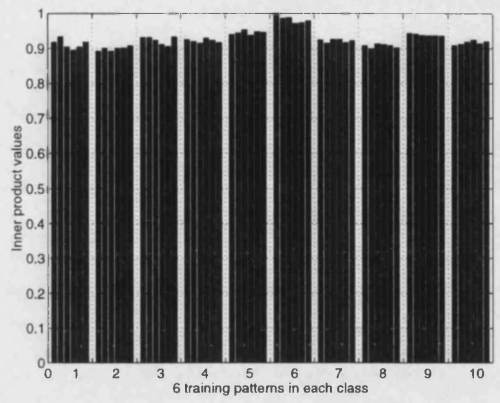
The cross-inner product matrix before the training, for the monopolar patterns, can be seen in figure 8.2. The first photo of each of the people was used as the initial, untrained filter for that class in the calculation of this cross-inner product matrix. The cross-inner product matrix is not square any more, since there are ten filters and sixty training patterns. Also each of the filters has six auto-inner products so the final cross-inner product matrix is not diagonal.

The surface graph shown in figure 8.2 does not give us a very clear view of all of the auto- and cross-inner products. We are going to create one graph for each of the rows of the cross-inner product matrix. Each row contains the inner products between the corresponding filter and all of the training patterns. It will be depicted as a bar chart. Each bar represents the value of an inner product. The bars are normalised to 1. They are divided into groups of six. Each group of bars represents the inner products with the six patterns that belong to that corresponding class.

In figure 8.3, subfigure (a) shows the first row of the cross-inner product matrix

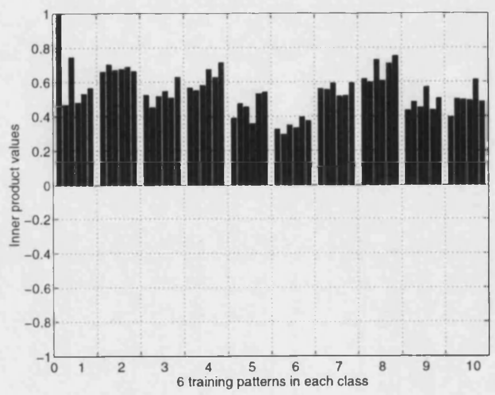


(a) Inner products of the 1st untrained filter

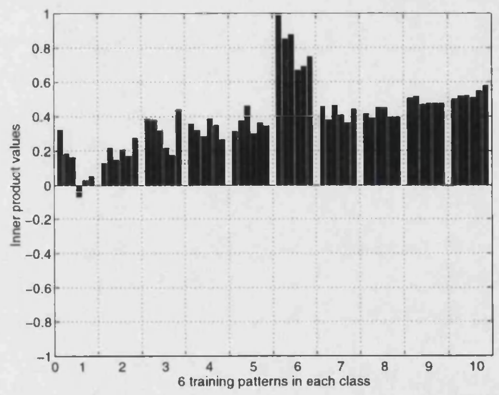


(b) Inner products of the 6th untrained filter

Figure 8.3: First and sixth row of the initial cross-inner product matrix of the monopolar patterns.



(a) Inner products of the 1st untrained filter



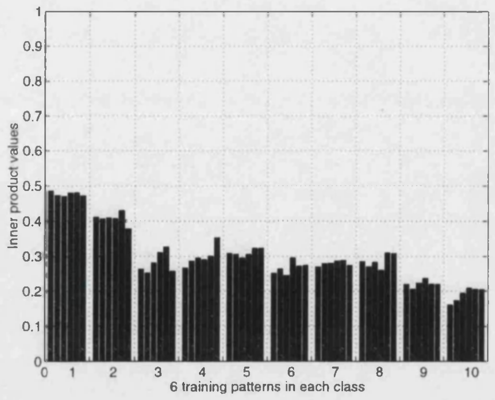
(b) Inner products of the 6th untrained filter

Figure 8.4: First and sixth row of the initial cross-inner product matrix of the bipolar patterns.

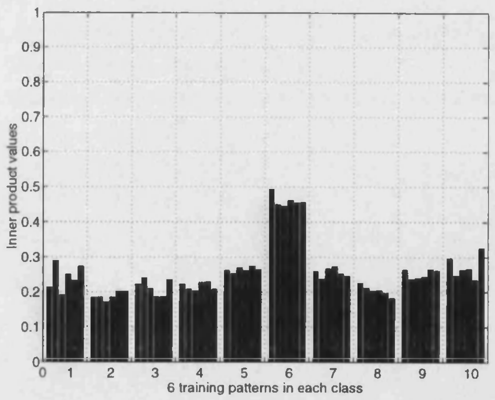
for the monopolar patterns. Subfigure (b) shows the sixth row of the same matrix. Subfigures (a) and (b) in figure 8.4 show the first and sixth row of the cross-inner product matrix for the bipolar patterns respectively. The similarity between all of the training patterns is evident in figure 8.3, because all of the auto- and cross-inner products are of almost equal magnitudes. Since the patterns are all normalised, and we used the first example of each subject as a filter, only the first auto-inner product is equal to one in all of the graphs shown in figures 8.3 and 8.4. A second observation we can make is that the bipolar patterns seem to be a lot less similar to each other than the monopolar patterns. This happens because due to the shifting and sign change, similar pixels in the monopolar patterns may have an opposite sign in the bipolar patterns and this reduces some of the inner products. Another observation one can make, is that the first filter, which represents the first class, seems to be more similar to all of the training patterns than the sixth filter which represents the sixth class. This means that we can set the appropriate threshold and use the sixth filter to successfully recognise all of the patterns that belong to the sixth class and reject all of the others. However, the same thing is not possible with the first filter which would give wrong results. In fact we chose to show these particular rows of the cross-inner product matrices because they represent the worse (1^{st}) and best (6^{th}) filters in terms of similarity to other patterns. Finally, we should note that some of the cross-inner products for the bipolar patterns are negative. If we set a threshold to distinguish between the auto- and the cross-inner products, these negative cross-inner products will be below the threshold and will be correctly rejected only in a system, which can detect their sign, for example an electronic recognition system. In an optical recognition system which only detects intensity on the output plane, these negative inner products would also be considered positive. Therefore, when using such a system there is no benefit in letting the cross-inner products converge to large negative values. All of the rows of the cross-inner product matrices before the training, for both the monopolar and bipolar patterns are shown in appendix D.

8.2.2 Training

For the training we let the FESS algorithm run until it converged to a relatively stable solution. In most of the simulations this happened within 30000 iterations. For the training we used equations 7.14, 7.15 and 7.16 with a minor modification. Instead of using two different convergence parameters β_1 and β_2, the convergence parameter was the same in equations 7.14 and 7.15, and equation 7.15 was applied



(a) Inner products of the 1st filter



(b) Inner products of the 6th filter

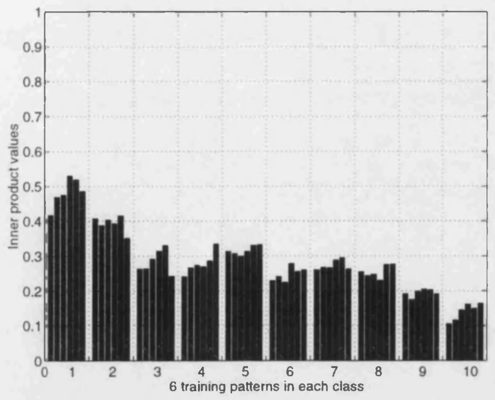
Figure 8.5: First and sixth row of the final cross-inner product matrix of the monopolar patterns after the training with the FESS algorithm. The initial filters were equal to the first example of the corresponding classes.

only once every D iterations. The final results were similar for D values between 15 and 30. The results presented here were obtained with an D value of 25.

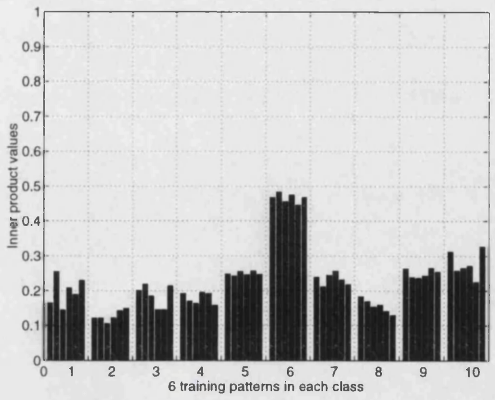
We conducted simulations with all three different values for the initial, untrained class representative filters: each one of them equal to one of the patterns belonging to the corresponding class, or the mean of all of the training patterns belonging to the corresponding class, or a random pattern. Here we present the results for all three cases[2]. We have to point out that regardless of whether the initial filters are monopolar or bipolar, they are going to end up with their pixels having both positive and negative values.

Figure 8.5 shows the bar charts for the first and sixth rows of the cross-inner product matrix, which was calculated using the monopolar patterns and the filters, which initially were equal to the first training pattern of the corresponding class. The first observation one can make looking at these graphs, is that all of the auto- and cross-inner products have decreased. The cross-inner products are now consistently lower than the auto-inner products. They are on average lower for the 6th filter than for the 1st filter. The first of the auto-inner products for each filter is slightly larger than the others because the filter was derived from the corresponding training pattern, but this difference is not large. A threshold can be set now to correctly recognise all of the training patterns.

[2]Look in appendix D for the graphs of all of the rows of all of the cross-inner product matrices after the training.



(a) Inner products of the 1st filter



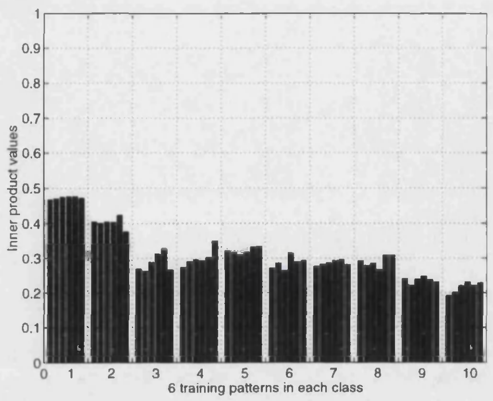
(b) Inner products of the 6th filter

Figure 8.6: First and sixth row of the final cross-inner product matrix of the monopolar patterns after the training with the FESS algorithm. The initial filters where equal to the mean of all of the examples of the corresponding classes.

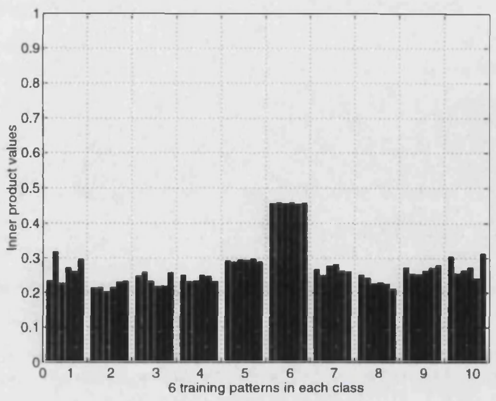
The cross-inner product matrix, whose first and sixth rows are shown in figure 8.6, was calculated using the monopolar patterns and the filters, which initially were equal to the mean of the training patterns of the corresponding class. The auto-and cross inner products are very similar to the previous case. There seem to be slightly larger differences between the auto-inner products of each filter but the magnitudes of the auto- and cross-inner products look the same on average.

Figure 8.7 shows the first and sixth rows of the cross-inner product matrix, which was calculated using the initially random filters and the monopolar patterns. The only thing that needs to be pointed out here is that the magnitude variations among the auto-inner products of each filter are very small. It is clear that the original filters had no individual features of the training patterns in them. They gained them during the training by the addition equation 7.14. These additions are weighted in such a way so that each of the filters finally becomes equally similar to all of the training patterns it represents. That is a possible explanation why the auto-inner products are almost equal in figure 8.7.

Figures 8.8, 8.9 and 8.10 show the first and sixth rows of the cross-inner product matrices for the bipolar matrices and the different initial filter values. In figure 8.8 we can see that the first auto-inner product is a lot higher than the others for both filters, obviously because they were derived from the corresponding pattern. Most of the cross-inner products have become negative or zero. In addition, there is a considerable difference between the behaviour of filter one (subfigure 8.8-(a)) and

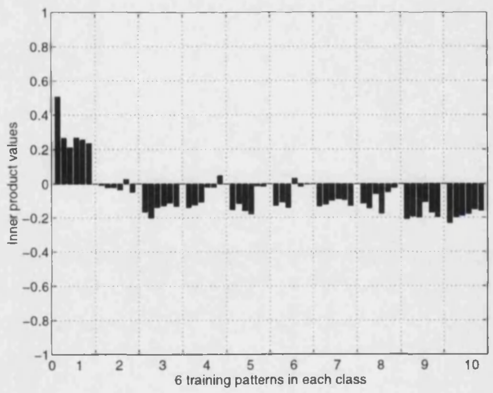


(a) Inner products of the 1st filter

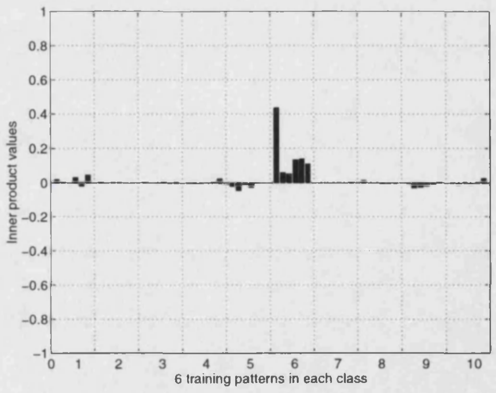


(b) Inner products of the 6th filter

Figure 8.7: First and sixth row of the final cross-inner product matrix of the monopolar patterns after the training with the FESS algorithm. The initial filters were random.

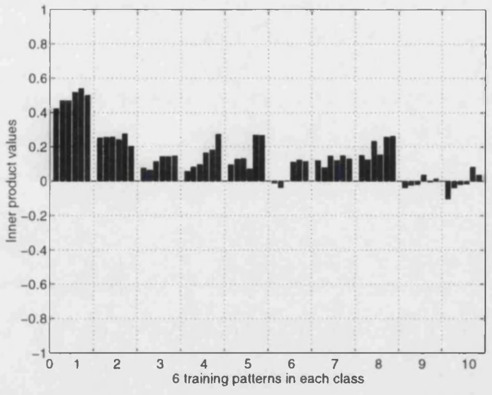


(a) Inner products of the 1st filter

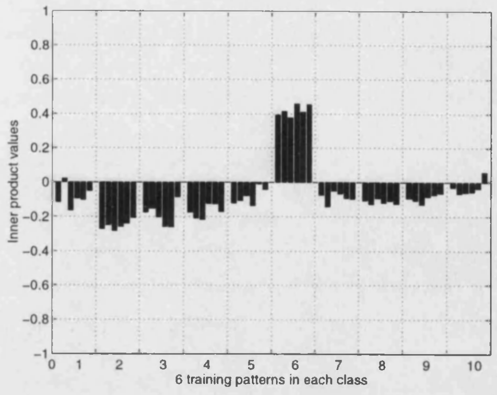


(b) Inner products of the 6th filter

Figure 8.8: First and sixth row of the final cross-inner product matrix of the bipolar patterns after the training with the FESS algorithm. The initial filters where equal to the first example of the corresponding classes.



(a) Inner products of the 1st filter



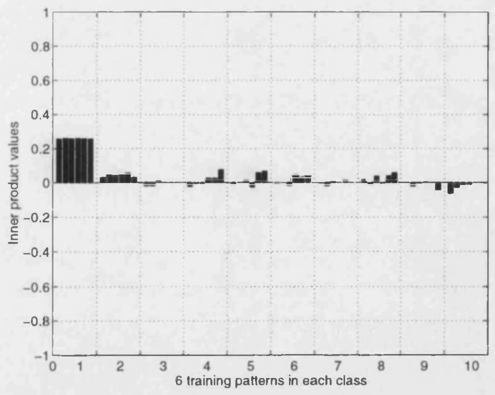
(b) Inner products of the 6th filter

Figure 8.9: First and sixth row of the final cross-inner product matrix of the bipolar patterns after the training with the FESS algorithm. The initial filters where equal to the mean of all of the examples of the corresponding classes.

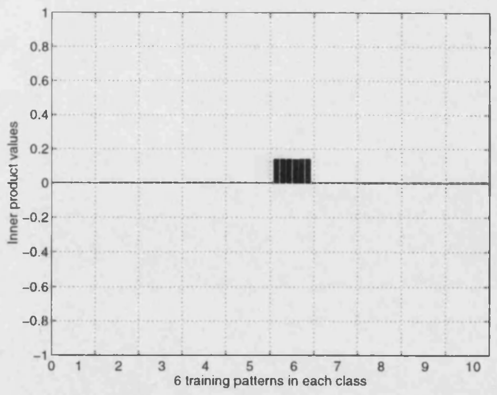
filter six (subfigure 8.8-(b)). Most of the auto-inner products for the first filter are almost double the size of the auto-inner products of the sixth filter. Most of the cross-inner products of the first filter have negative values, while most of the cross-inner products of the sixth filter are equal to zero. Consider what would happen if we used these filters to recognise the second training example of the sixth class. Its auto-inner product with the sixth filter is smaller (in absolute values) than its cross-inner product with the first filter. In an optical system which only detects intensity on the correlation plane, this would result in incorrect recognition.

The filters' performance is quite different when they are initially equal to the mean of all of the training patterns they represent (figure 8.9). The auto-inner products have similar values and they are all higher than all of the cross-inner products. Another observation we can make is that in this case all of the auto-inner products have values which are very close to the value of the first auto-inner product for each class in figure 8.8. Finally, the cross-inner products have reduced, but they have higher absolute values compared to the ones in figure 8.8. Again, a threshold would allow correct recognition. With random initial filters, (figure 8.10) all of the auto-inner products of each filter have the same magnitude and the cross-inner products are very low or zero.

Essentially the information contained in all of the bar charts (the six in this chapter and the ones in appendix D) can be summarised in the following tables. Table 8.1 shows the mean and standard deviation of all of the auto- and cross



(a) Inner products of the 1st filter



(b) Inner products of the 6th filter

Figure 8.10: First and sixth row of the final cross-inner product matrix of the bipolar patterns after the training with the FESS algorithm. The initial filters where random.

inner products between the three different filters and the monopolar patterns. The same information for the bipolar patterns is displayed in table 8.2.

After studying the bar charts and tables 8.1 and 8.2 we can make the following statements for the filters after the training using the FESS algorithm:

- Monopolar patterns
  - All of the cross-inner products are smaller than all of the auto-inner products for all of the classes for all three different initial filter values. Therefore, a threshold can be set to distinguish the training patterns at least.

Monopolar patterns					
	Auto-inner products		Cross-inner products		$\frac{Auto}{Cross}$
	MEAN	STD DEV	MEAN	STD DEV	
Initial Filters	0.9706	0.0061	0.9020	0.0347	1.076
Filter 1	0.4612	0.0156	0.2500	0.0387	1.845
Filter Mean	0.4660	0.0244	0.2288	0.0541	2.037
Filter Rand	0.4567	0.0024	0.2702	0.0354	1.690

Table 8.1: Mean value and standard deviation of the auto- and cross-inner products for the monopolar patterns. The last column in the table shows the ratio of the mean of the auto-inner products over the mean of the cross-inner products.

Bipolar patterns					
	Auto-inner products		Cross-inner products		$\frac{Auto}{Cross}$
	MEAN	STD DEV	MEAN	STD DEV	
Initial Filters	0.8077	0.0384	0.4653	0.0180	1.735
Filter 1	0.1693	0.1325	0.0351	0.0246	4.823
Filter Mean	0.4203	0.0370	0.1009	0.0618	4.165
Filter Rand	0.1498	0.0007	0.0062	0.0058	24.161

Table 8.2: Mean value and standard deviation of the auto- and cross-inner products for the bipolar patterns. The last column in the table shows the ratio of the mean of the auto-inner products over the mean of the cross-inner products.

  - The mean magnitude of the auto- and cross-inner products is very similar for all three different initial filter values. However, the filters, which were initially equal to the mean of all of the patterns of the corresponding classes, produced the highest mean for the auto-inner products and the lowest mean for the cross-inner products.
  - The standard deviation of the auto-inner products is an order of magnitude smaller for the filters that were initially random, than for the other two. The mean filters produced auto- and cross-inner products with the highest standard deviation.

- Bipolar patterns
  - The filters that were derived from one training pattern only, produced some cross-inner products higher than auto-inner products. Subsequently, 100% correct discrimination could not be achieved by using a threshold.
  - In addition, the filters that were derived from one training pattern only, produced the auto-inner products with the highest standard deviation, which was almost equal to the mean of the magnitudes.
  - The filters that were derived from the mean of the training patterns of the corresponding classes, produced auto-inner products significantly higher than the other two kinds of filters. However, their cross-inner products were higher as well, although always lower than the auto-inner products.

- The filters that were initially random, forced almost all of the cross-inner products to zero. Their auto-inner products had the lowest mean but also by far the lowest standard deviation. In addition they were consistently higher than the cross-inner products.

Figures 8.11 and 8.12 show all of the final filters that were created using the monopolar and the bipolar patterns and the three different initial filter values. The first observation one can make, is that areas that are enhanced in some images, are also enhanced in some others, but with an opposite sign. For example, the area of the hair in the first subject is very bright and the same area in the sixth subject is very dark. In the monopolar, initially random filters the features of each subject are now identifiable. Some features are identifiable in the bipolar, initially random filters but not as many as in the filters derived from the monopolar patterns. The bipolar filters, which were initially equal to one training pattern are all blurred in a very similar fashion and only have some form of edge enhancement. We note that the final filters are similar to their initial value. Therefore, the choice of the initial filters is very important when the initial filters are equal to one of the patterns of the class they represent. Finally, several superimposed images can be seen in the filters derived from the mean of all of the corresponding training patterns for both the monopolar and the bipolar filters.

8.2.3 Convergence speed

In this section we discuss the convergence speed of the algorithm for the bipolar and monopolar patterns and for the different initial filters. In the beginning of section 8.2 we introduced the *energy ratio* figure of merit. We have plotted the energy ratio as a function of iteration for all of our simulations.

In figure 8.13 we can see the energy ratio plotted against iteration number for the monopolar patterns and the three initial filter values. Notice that the x axis in subfigures (a) and (b) extends only to 10000 iterations. This is because in those simulations the algorithm had already converged within the first 10000 iterations. The energy ratio converges to about the same value for all three different initial filter values, although a lot slower for the initially random filters. Figure 8.14 shows the same plots for the bipolar patterns. These graphs are more interesting. First of all we see that the graph for the random filters is thick as if the energy ratio oscillated. This is actually true, not only for the energy ratio of these filters but also for the energy ratio of all of the filters, for the bipolar and the monopolar

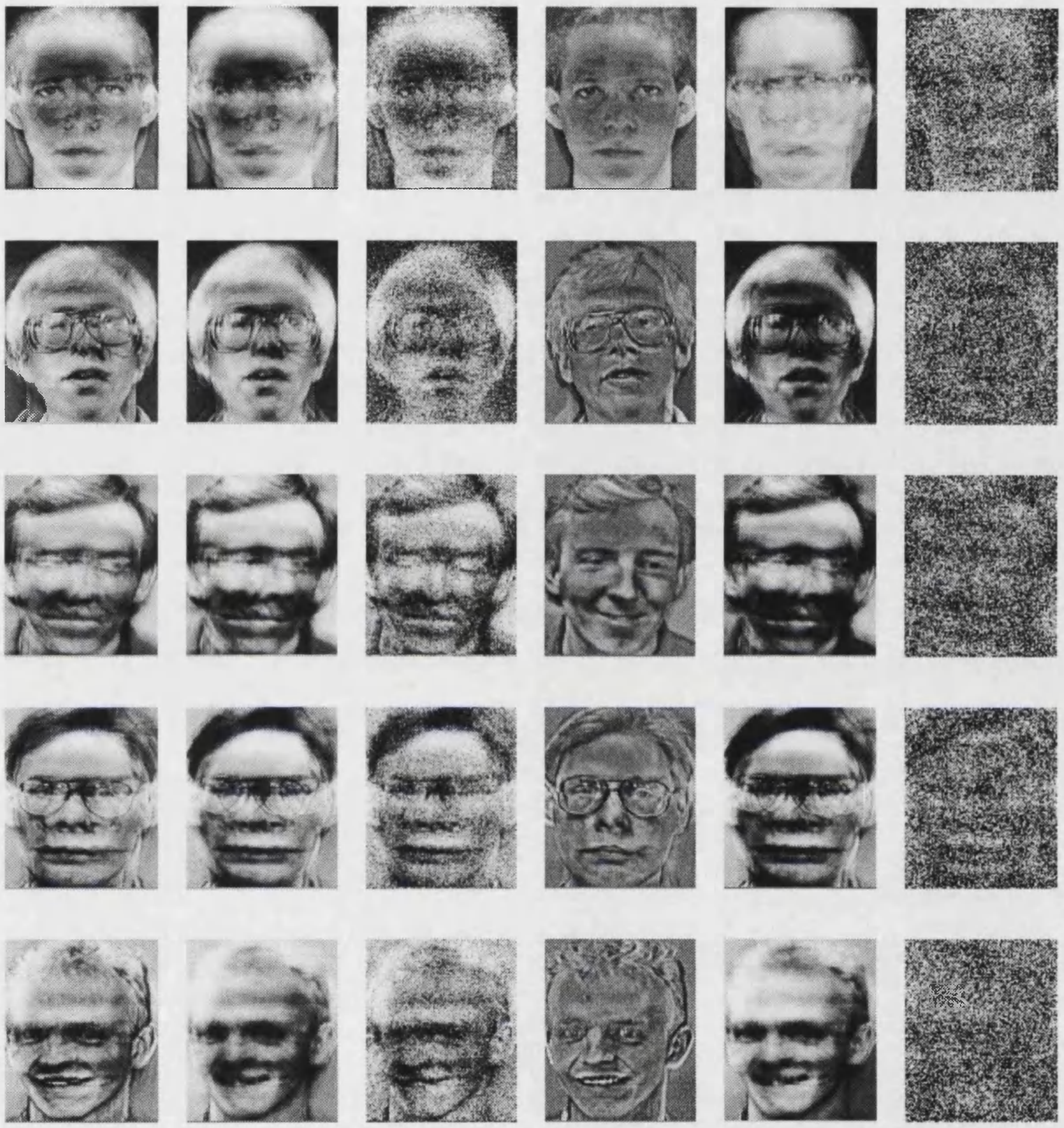


Figure 8.11: Final filters for the first five subjects, for monopolar and bipolar patterns and for all three initial filter values. 1st Column : monopolar patterns, initial filters equal to one pattern. 2nd Column : monopolar patterns, initial filters equal to the mean of the patterns. 3rd Column : monopolar patterns, random initial filters. 4th Column : bipolar patterns, initial filters equal to one pattern. 5th Column : bipolar patterns, initial filters equal to the mean of the patterns. 6th Column : bipolar patterns, random initial values.

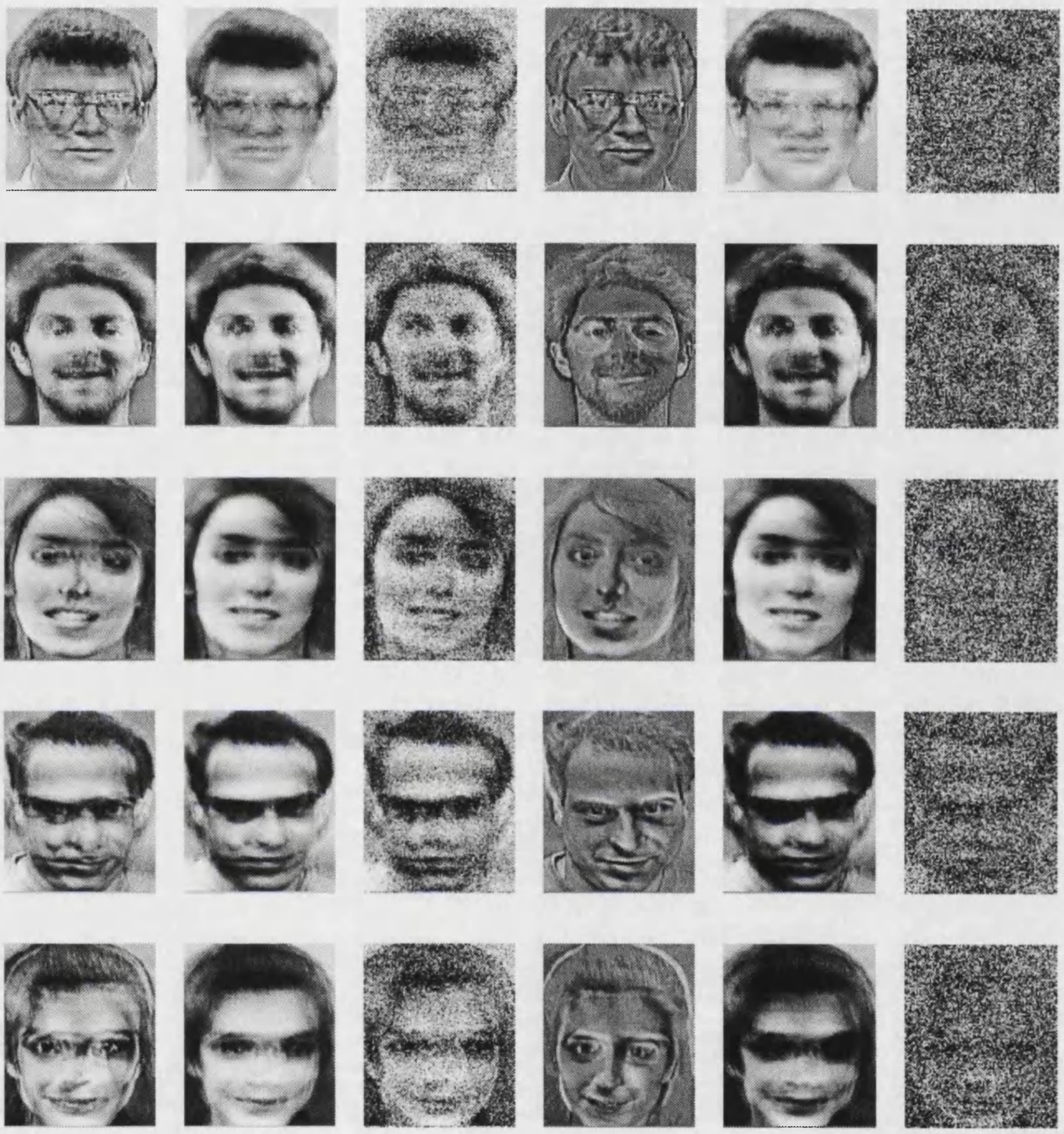
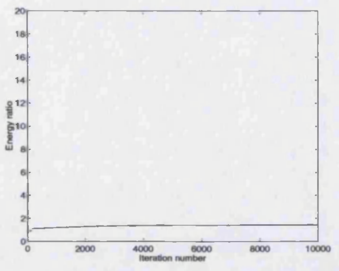
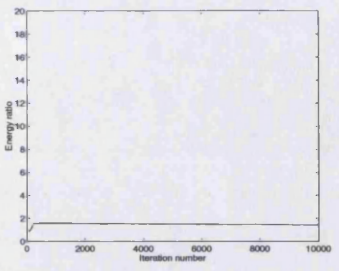


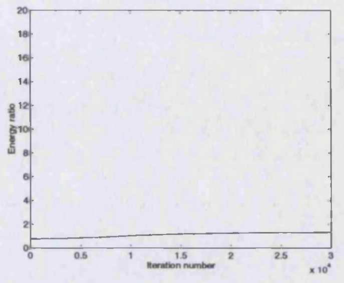
Figure 8.12: Final filters for the last five subjects, for monopolar and bipolar patterns and for all three initial filter values. 1st Column : monopolar patterns, initial filters equal to one pattern. 2nd Column : monopolar patterns, initial filters equal to the mean of the patterns. 3rd Column : monopolar patterns, random initial filters. 4th Column : bipolar patterns, initial filters equal to one pattern. 5th Column : bipolar patterns, initial filters equal to the mean of the patterns. 6th Column : bipolar patterns, random initial values.



(a) Initial filters equal to one of the training patterns

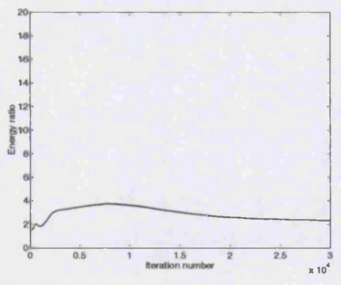


(b) Initial filters equal to the mean of the training patterns

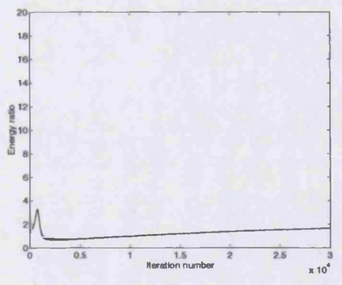


(c) Initial filters random

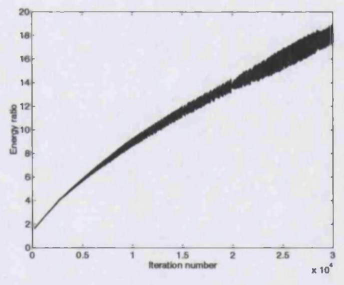
Figure 8.13: Energy ratio for the FESS algorithm plotted against number of training iterations for the monopolar patterns.



(a) Initial filters equal to one of the training patterns of the corresponding classes



(b) Initial filters equal to the mean of the training patterns of the corresponding classes



(c) Initial filters random

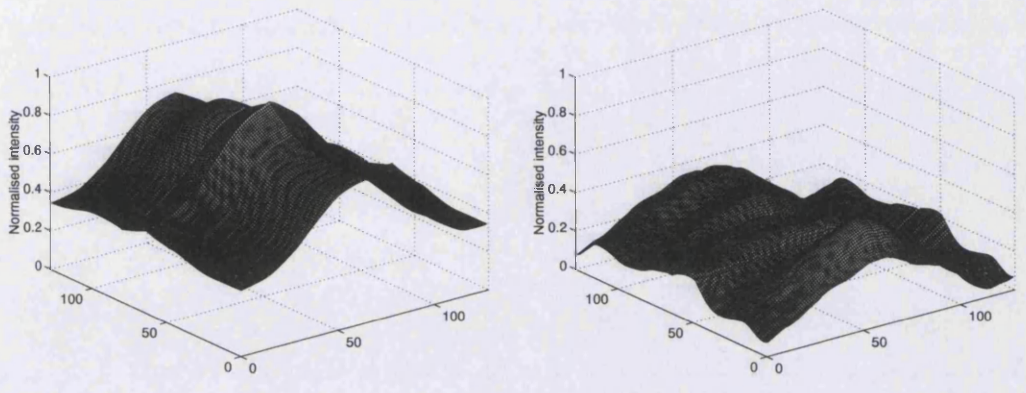
Figure 8.14: Energy ratio for the FESS algorithm plotted against number of training iterations for the bipolar patterns.

patterns. These oscillations are not visible in the other curves because they were a lot smaller and all of the curves were plotted on the same scale for comparison. These oscillations happened because the subtraction equation was used only once every 25 iterations. The energy ratio actually decreased as the addition equation was applied and it jumped to a higher value every time the subtraction equation was applied. The other two graphs in subfigures (a) and (b) show a bump at the beginning of the training (after 351 iteration in subfigure 8.14-(b)). We found these hard to explain because the cross-inner product matrices at those particular iterations were not very different from the matrices at the final iteration. We believe, however, that they resulted from relatively large inner product value fluctuations due to sign changes.

8.2.4 Peak to Correlation Energy (PCE) of correlations between the initial patterns and the final, trained class filters

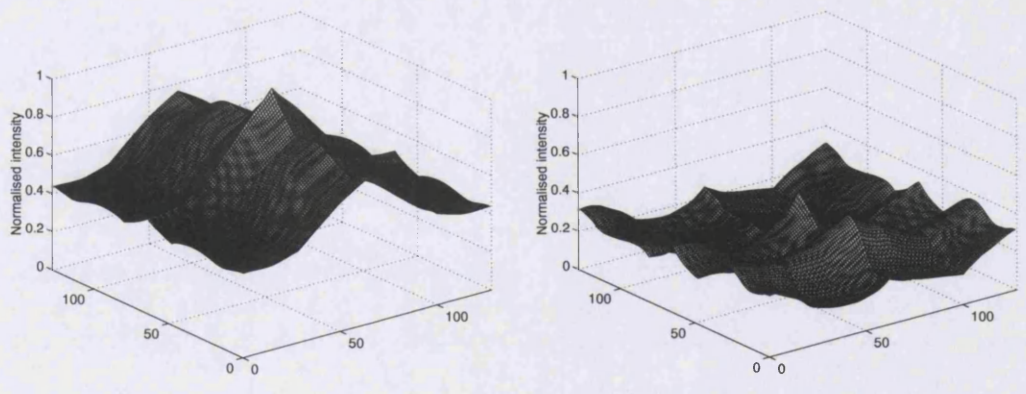
Like the SS algorithm, the FESS algorithm does not place any constraints on the outer products of the correlations between the filters and the training patterns. In this section we investigate whether the outer products increase or decrease or remain stable after the training. We use the peak to correlation energy (PCE), defined in chapter 2, section 2.6, equation 2.6, to measure the sharpness of the correlation peaks, which are located in the centre of the correlation plane at point (65,65) in each graph. In all of the following simulations in this section and in the next one, we have used the first training example of each of the classes as the representative filter for that class before the training.

Figure 8.15 shows the correlation plane intensity for the correlations between the monopolar training pattern $\mathbf{s}_{61}$, which belongs to the sixth class and the untrained (subfigure a) and trained (subfigure b) filters representing the first class. The trained filter used for this correlation was the one that was derived from the mean of all of the training patterns of the first class. The results using the other methods for the initial values of the trained filters were very similar for the monopolar patterns. We can see from figure 8.15 that the outer products have not increased. On the contrary they have decreased. The PCE for this correlation was 0.018 before and $4.8 \cdot 10^{-7}$ after the training. This is good, rather than bad however, because we do not want this correlation to produce a correlation peak, because filter $\mathbf{g}_1$ must reject pattern $\mathbf{s}_{61}$.



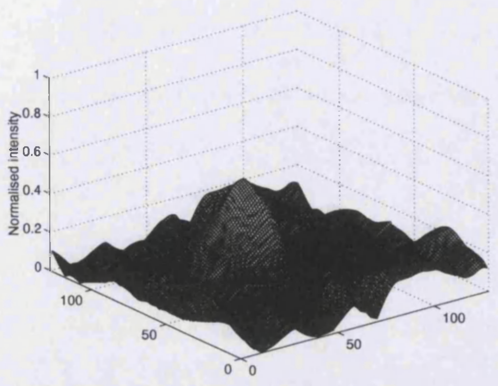
(a) $\mathbf{s}_{61} \otimes \mathbf{s}_{11}$, PCE=0.018 (b) $\mathbf{s}_{61} \otimes \mathbf{g}_1$, PCE=$4.8 \cdot 10^{-7}$

Figure 8.15: Correlation plane intensity for correlations between a photo of the sixth subject and the untrained (a) and the trained (b) filter for the first subject using the monopolar patterns. Filter $\mathbf{g}_1$ was initially equal to the mean of all of the training patterns in the 1st class. Correlation peak location: (65,65)

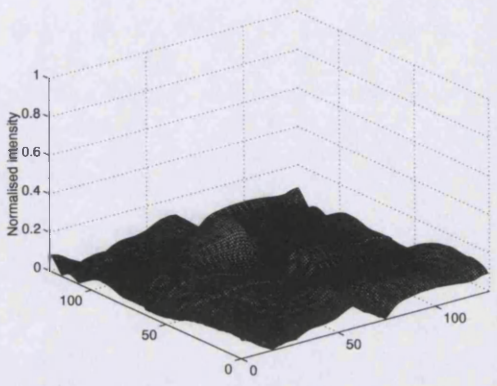


(a) $\mathbf{s}_{61} \otimes \mathbf{s}_{61}$, PCE=0.024 (b) $\mathbf{s}_{61} \otimes \mathbf{g}_6$, PCE=0.006

Figure 8.16: Correlation plane intensity for correlations between a photo of the sixth subject and the untrained (a) and the trained (b) filter for the sixth subject using the monopolar patterns. Filter $\mathbf{g}_6$ was initially equal to the mean of all of the training patterns in the 6th class. Correlation peak location: (65,65)



(a) $\mathbf{s}_{61} \otimes \mathbf{s}_{11}$, PCE=0.001

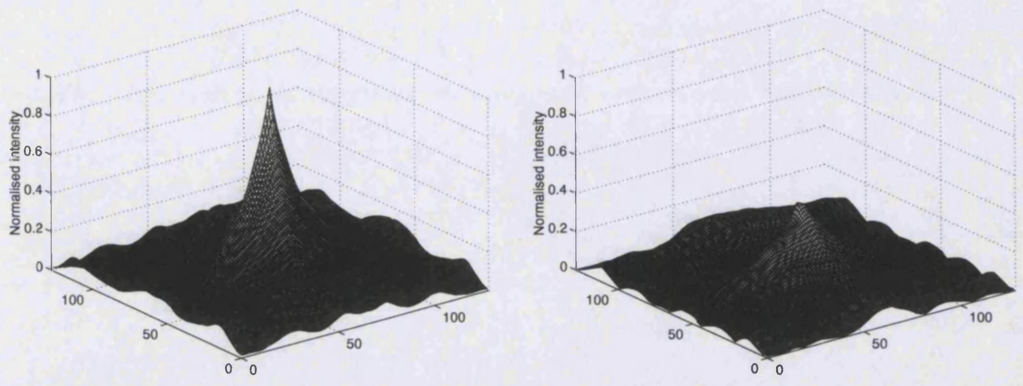


(b) $\mathbf{s}_{61} \otimes \mathbf{g}_1$, PCE=$4.6 \cdot 10^{-9}$

Figure 8.17: Correlation plane intensity for correlations between the sixth subject and the untrained (a) and the trained (b) filter for the first subject using the bipolar patterns. The trained filter was initially equal to the mean of the training patterns it represents. Correlation peak location: (65,65)

The same cannot be said for the correlations depicted in figure 8.16. In that figure, subfigure (a) shows the auto-correlation of pattern $\mathbf{s}_{61}$ and subfigure (b) shows the correlation between pattern $\mathbf{s}_{61}$ and the trained filter for the sixth class. Again the monopolar pattern was used for these correlations and the trained filter was initially equal to the mean of the patterns of the sixth class. We observe the same outer-product behaviour for these correlations as well. They have decreased in absolute terms, but they have increased relative to the correlation peak. The PCE for the auto-correlation was 0.024 and for the correlation with the final filter, 0.006. In this case, we do want an existing correlation peak so the reduced PCE is a disadvantage.

Figures 8.17 and 8.18 show the same correlations for the bipolar patterns. In both figures the initial values of the trained filters were equal to the mean of the training patterns in the corresponding classes. First of all we can see that the outer-products are a lot lower in general when using the bipolar patterns. Particularly in the auto-correlation of $\mathbf{s}_{61}$ shown in subfigure (a) of figure 8.18, the correlation peak is a lot sharper than in the same auto-correlation using the monopolar version of the pattern. The PCE is 0.071. Using the trained filter for the sixth class (subfigure 8.18(b)), the PCE falls to 0.004. However, the correlation peak is clearly higher than all of the outer products. For the correlation between pattern $\mathbf{s}_{61}$ and the untrained and trained filters representing the first class (figure



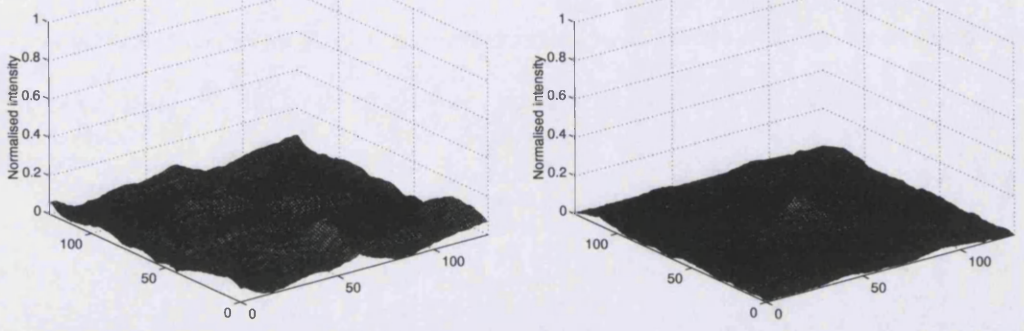
(a) $\mathbf{s}_{61} \otimes \mathbf{s}_{61}$, PCE=0.071 (b) $\mathbf{s}_{61} \otimes \mathbf{g}_6$, PCE=0.004

Figure 8.18: Correlation plane intensity for correlations between a photo of the sixth subject and the untrained (a) and the trained (b) filter for the sixth subject using the bipolar patterns. Before the training, $\mathbf{g}_6$ was equal to the mean of all of the training patterns it represents. Correlation peak location: (65,65)

8.17), the PCE is 0.001 and $4.6 \cdot 10^{-9}$ respectively.

Finally, figure 8.19 shows the correlations between pattern $\mathbf{s}_{61}$ and the trained filters representing the first (subfigure a) and the sixth (subfigure b) classes. The filters used for these correlations were the initially random filters. Both correlations have very low outer products. The correlation with the filter representing the first class has no correlation peak at all and, therefore, a very low PCE, $5.99 \cdot 10^{-11}$ and the correlation with the filter representing the sixth class has a discernible although very low in magnitude correlation peak and a PCE equal to 0.028. The PCE is relatively high, compared to the other correlation in subfigure (a), although the correlation peak is so low, due to the almost complete absence of outer products.

To conclude, we have seen through these examples that for monopolar patterns the training increases the outer products relative to the correlation peak and reduces the peak-to-correlation energy. For the bipolar patterns, the correlation peaks are a lot sharper before the training. The outer products do not increase with the training, but the PCE decreases because the correlation peaks decrease. However, for all of the cases for the monopolar and the bipolar patterns, after the training, the auto-correlations have a higher PCE than the cross-correlations.



(a) $\mathbf{s}_{61} \otimes \mathbf{g}_1$, PCE=$5.99 \cdot 10^{-11}$ (b) $\mathbf{s}_{61} \otimes \mathbf{g}_6$, PCE=0.078

Figure 8.19: Correlation plane intensity for correlations between a photo of the sixth subject and the trained filter for the first subject (a), and trained filter for the sixth subject (b) using the bipolar patterns and the initially random filters. Correlation peak location: (65,65)

8.3 Probability of recognition and dynamic range

In the previous section we described the simulations of the FESS algorithm during the training. In this section we present the results of the probability of recognition and the dynamic range tests. These were calculated for both the training set and for a test set. The test set will be described later in the section after we present the results for the training set. Equation 6.2, which we rewrite here was used for the calculation of the dynamic range required by the optical system

$$\mathit{dynamic\ range} = max\forall i\Bigg(-20\log_{10}\left\{\frac{\mathbf{s}_i \cdot \mathbf{g}_i - max\forall j(\mathbf{s}_i \cdot \mathbf{g}_j)}{\mathbf{s}_i \cdot \mathbf{g}_i}\right\}\Bigg),$$

$$i = 1 \ldots M,\ j = 1 \ldots M,\ j \neq i \tag{8.2}$$

First in table 8.3 we present the results for the training set.

Table 8.3 shows the probability of recognition and dynamic range for the monopolar patterns and for the bipolar patterns for the following two cases. The first is when the optical system used (if an optical system is used) has some kind of interferometric phase detection at the correlation plane, which allows us to detect the phases (+,-) of the correlation peaks. The second case is for when there is no phase detection and we can only measure the intensity of the correlation peaks. The method for calculating the probability of recognition for the tests with the training set, was to calculate all of the cross-inner products between an

TRAINING SET		Prob. Rec. (%)	Dyn. Range. (dB)
Monopolar patterns	Initial filters	88.3	30.8
	Filter 1	100.0	9.06
	Filter Mean	100.0	8.88
	Filter Rand	100.0	9.74
Bipolar patterns, phase detection	Initial filters	95.0	8.59
	Filter 1	95.0	-
	Filter Mean	98.3	4.79
	Filter Rand	100.0	0.72
Bipolar patterns, no phase detection	Initial filters	95.0	8.59
	Filter 1	55.0	-
	Filter Mean	98.3	4.79
	Filter Rand	100.0	0.72

Table 8.3: Probability of recognition and dynamic range using the training set.

input pattern and all of the filters and then to choose the highest of them. For correct recognition, the highest inner product had to be the one with the filter which corresponded to the input pattern. No threshold was used, since there is no need to reject any patterns when using the training set. We can see that when using the monopolar patterns, the results are very similar for all three initial filter values. In all cases the probability of recognition has increased to 100% after the training and the required dynamic range has decreased by about 21 dB and is now around 9 dB.

When using the bipolar patterns, the results are different for the three initial filter values. Only the initially random filters increase the probability of recognition to 100% after the training. The filters that were initially equal to the mean of the corresponding training patterns, increase the probability of recognition to 98.3%. The filters that were initially equal to one pattern only, produce the worst results: when using phase detection the probability of recognition is 95%, equal to what it was before the training. When not using phase detection, the probability of recognition falls to 55% with the filters that were initially equal to the first example of the classes they represent. The required dynamic range is a lot lower for the bipolar patterns compared to the monopolar, before and after the training. The initially random filters produce the lowest required dynamic range of all.

The test set did not include any common patterns with the training set. It consisted of sixty faces in total. Forty of these patterns belonged to the same ten people that the system was trained to recognise, but they were different from the corresponding examples in the training set. The remaining twenty patterns were faces of five other people. In this case, we used a threshold to calculate the probability of recognition. Apart from the probability of recognition and dynamic range, we have also calculated the *False Positive* and *False Negative* percentages for the test set. We get a false positive when the input pattern does not belong to the memorised set, in other words, it does not belong to any of the classes our filters were trained to recognise and one of the correlations with the filters has a central peak higher than the designated threshold. In that case the system wrongly recognises an unknown pattern. A false negative is registered when the input pattern represents one of the subjects of the memorised set, but the correlations with all of the filters have peaks lower than the designated threshold. In that case the system wrongly rejects a known pattern. We can trade these off against one another by raising or lowering the threshold. Therefore, to calculate the probability of recognition, the False Positives and the False Negatives for the test set, we used the following method: we calculated all of the cross-inner products between an input pattern and all of the filters and then chose the largest of them. Then we had the following cases:

- Correct recognition when:
    i. The input pattern belongs to one of the memorised classes.
    ii. Its largest inner product is with the filter which represents the class in which the input pattern belongs.
    iii. Its largest inner product is larger than the set threshold.
- Wrong recognition when:
    i. The input pattern belongs to one of the memorised classes.
    ii. Its largest inner product is not with the filter which represents the class in which the input pattern belongs but with another filter.
    iii. That largest inner product is larger than the set threshold.
- False positive when:
    i. The input pattern does not belong to one of the memorised classes.

TEST SET						
		Thr	PR (%)	FP (%)	FN (%)	DR (dB)
Monopolar patterns	Initial filter	0.922	93.3	0.0	0.0	26.2
	Filter 1	0.391	91.6	3.3	3.3	8.38
	Filter Mean	0.385	85.0	8.3	6.6	7.66
	Filter Rand	0.397	96.6	0.0	3.3	9.35
Bipolar patterns, sign detection	Initial filter	0.717	70.0	8.3	18.3	10.1
	Filter 1	0.053	66.6	23.3	3.3	-
	Filter Mean	0.358	70.0	3.3	26.6	3.25
	Filter Rand	0.089	85.0	8.3	3.3	1.96
Bipolar patterns, no sign detection	Initial filter	0.753	71.6	0.0	26.6	10.1
	Filter 1	0.089	38.3	30.0	5.0	-
	Filter Mean	0.358	70.0	3.3	26.6	4.37
	Filter Rand	0.089	85.0	8.3	3.3	2.35

Table 8.4: Probability of recognition, false positives, false negatives and dynamic range using the test set. Thr: Threshold, PR: Probability or recognition, FP: False Positives, FN: False Negatives, DR: Dynamic Range.

ii. One of its inner products with one of the filters is larger than the set threshold.

- False negative when:

  i. The input pattern belongs to one of the memorised classes.

  ii. Its largest inner product is lower than the set threshold.

The probability of recognition was equal to the number of correct recognitions expressed as a percentage. Obviously, the percentages of correct and wrong recognition, along with the percentages of false positives and false negatives add up to a hundrend percent. One other thing that we must point out is that we used the same threshold for all of the filters.

Table 8.4 shows the corresponding results when using the test set. Again starting with the monopolar patterns, we see that the initially random filters provide the highest probability of recognition, 96.6%. The filters that were initially equal to the mean of the corresponding training patterns produce the worse results mainly because of the high number of false positives and false negatives. All of the filters for the monopolar patterns reduce the required dynamic range by about 17 dB.

The probability of recognition is in general lower when using the bipolar filters. However, the required dynamic range is also lower. The worst results are produced by the filters that were initially equal to only one training pattern and the best results are produced by the initially random filters which increase the probability of recognition to 85% and require very low dynamic range, around 2 dB. The filters that were originally equal to the mean of the corresponding training patterns, do not provide any significant performance improvement over the untrained filters.

To summarise, we saw that the dynamic range requirements are greatly lowered when using the trained filters for both the training and the test set. The probability of recognition rises to 100% for the training set, but does not improve greatly when using the trained filters to recognise the patterns in the test set. This did not happen because the patterns were wrongly recognised, on the contrary the filters displayed very good generalisation ability and were able to correctly recognise most of the test set patterns, which represented the subjects that they were trained on. The low performance resulted from the high number of false positives and false negatives (table 8.4). We might be able to improve the performance if we used a different threshold for each of the filters. The high number of false positives and false negatives was expected. As we have said before this bank of correlators corresponds to a single layer of neurons. It is known from neural network theory, that a single layer of neurons cannot solve non-linear problems. To obtain better results we must use at least two cascaded banks of correlators and thresholds, which correspond to a multilayer network. Even in that case, however, it has been proved [109] that the network cannot form closed separation surfaces around the classes, which would eliminate the false positives, if the number of neurons in the first hidden layer is lower or equal to the number of inputs. Even if the number of neurons in the first hidden layer is higher than the number of inputs, Gori *et. al.*[109] prove that the network may or may not form closed surfaces. In other words, the number of correlators in the first hidden bank must be at least equal to the size of the images ($T \geq N$ in figure 7.2) and even that does not ensure that the false positives will be eliminated.

Another issue that we would like to discuss briefly, is the necessary training time for the convergence of the extended SS and FESS algorithms, which train two consecutive banks of correlators. We have already shown that these algorithms are mathematically equivalent to the BEP algorithm for training neural networks.

The computation of the weights of a multilayer feed-forward neural network, using the BEP algorithm is NP-complete [110, 111, 112, 113]. This means that the computing time necessary for the training, scales with $O!$, where O is the number of neurons, or in our case correlators, in the total network. Since the number of correlators in the hidden bank must be greater than or equal to the number of pixels of an input image, the training period required for complete convergence is going to be very lengthy for any reasonable input image size, for our extended SS and FESS algorithms. Our simulations in chapter 6 have clearly shown that at least for a single layer of correlators, complete convergence leads to over-fitting of the filters to the training patterns and lower generalisation ability. It is more fruitful, therefore, to stop the algorithm after a few iterations only. It may be better to stop the training of the two consecutive banks of correlators after a few iterations as well. Whether the undertraining of the two consecutive banks of correlators will also produce better results can only be confirmed with further computer simulations. In addition, we must point out that the SS algorithm incorporates a nonlinearity in the form of the squared term, which speeds up the convergence, particularly in the initial iterations, when the cross-inner products are large. This results in the convergence index (total energy) curve having a shape similar to that of the solid curve in figure 8.20. This figure shows two representative convergence index curves for the SS algorithm with and without the squared term. When the weighting term in the SS algorithm is not squared the total energy curve is similar to the dashed curve in figure 8.20. The steeper shape of the convergence index curve of the algorithm with the squared term, means that the algorithm needs a very small number of iterations to reach a sufficiently good solution. It will probably be necessary to incorporate this square in the extended SS and FESS algorithms, in order to obtain similar convergence behaviour.

8.4 Conclusions

In this chapter we have presented computer simulations for the FESS algorithm. We showed that the algorithm forces the cross-inner products to decrease, although not always to the desired value, which is zero. We saw that the auto-inner products decrease as well, but converge to higher values than the cross-inner products. The convergence procedure is rather slow and usually takes several thousand iterations. The outer products do not increase, as they did with the SS algorithm, but because the inner-products decrease, the peak-to-correlation energy measure

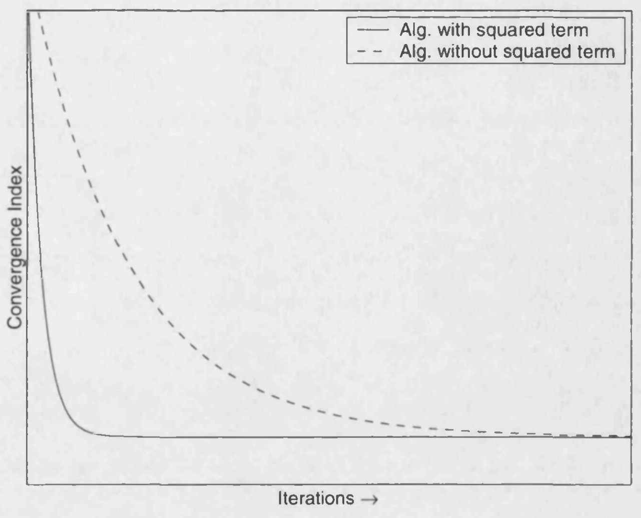


Figure 8.20: Convergence index curves of the SS algorithm with and without the squared term.

is in general lower after the training. This means that locating the correlation peak will be more difficult when using the final filters. The dynamic range required by the optical system is greatly reduced when using the final filters. Therefore, a system with lower dynamic range can be used to recognise our patterns, or more system noise, which reduces the system's dynamic range, can be tolerated. The probability of recognition for patterns in the training set is increased to 100% after the training. However, for the test set, the probability of recognition although increased, does not reach 100%, because this is impossible with only one bank of correlators. With this chapter we have concluded the presentation of the filter design algorithms and their simulations. In the next chapter we present our conclusions and make some suggestions for continuing this work.

Chapter 9

Conclusions and Future Work

9.1 Introduction

In this final chapter of the thesis we present the conclusions and achievements of our work. We start with an overview (section 9.2) of the general achievements of our work, which underline and provide a framework for understanding the more specific achievements presented in the next section, 9.3. Some recommendations on how this work should be continued are presented in the last section of the chapter.

9.2 General achievements

Based on the structural equivalence between optical correlators and neural networks [4], specifically perceptrons, we used the SS and FESS algorithms as intermediate formulations to compare the existing training algorithms for neural networks and filter design. These comparisons (chapters 5 and 7) led to the framework shown in figure 9.1.

We have created a bridge between these two domains, neural networks and optical filters, in the form of two new algorithms. Some of the new knowledge in this project has resulted from transferring well known results from the neural network field to the optical filter field, across this bridge (figure 9.1). We have shown in this thesis that:

- The matrix method for designing LCF filters or SDFs can be written in an iterative way. That in itself is not particularly useful. However, the algorithm can be stopped after one or two iterations, resulting in a filter

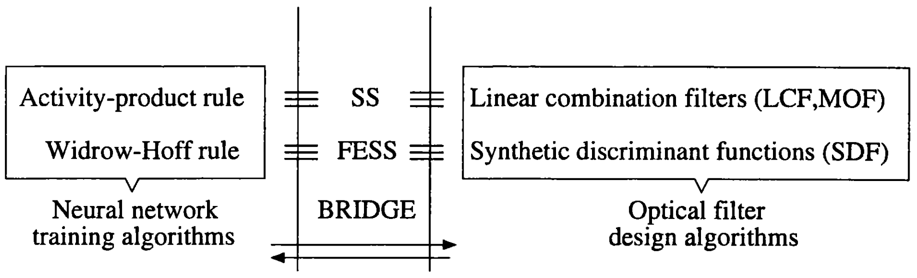

Activity-product rule
Widrow-Hoff rule
Neural network training algorithms
SS
FESS
BRIDGE
Linear combination filters (LCF,MOF)
Synthetic discriminant functions (SDF)
Optical filter design algorithms

Figure 9.1: Relationships between our algorithms, neural network training algorithms and filter design techniques

with better generalisation behaviour, more tolerance to additive input noise and lower outer products. The improved performance arises from avoiding the over-training of the filters, a problem which is very well known in the neural network field. The 2 iteration training procedure of the SS algorithm has the additional benefit of being as fast to perform as the earlier matrix method. This has been observed in our computer simulations.

- All of the existing optical filters, which are designed for a single correlator, or a single bank of correlators, suffer from the limitations that hamper single layer neural networks. In particular, a bank of correlators, like a single layer of neurons, cannot always correctly recognise non-linearly separable patterns [3]. In neural networks, this problem is sometimes solved by using more than one layers of neurons. We extended our algorithms to design filters for two cascaded banks of correlators (chapters 5 and 7). To our knowledge, non of the existing optical filter design techniques can calculate such filters.

This transfer of knowledge from one field to the other also works the opposite way, from optical filters to neural networks:

- The SDF method of optical filters can be used in preference to the iterative supervised Hebbian law to train neural networks faster, when over-training does not pose a problem for the application.

Overall, it is well known that transferring existing results from one academic field to another can result in new discoveries. It can also help avoid making the same mistakes, or following research paths that lead to dead ends. Therefore, the fundamental framework achievement in this PhD was that of devising a bridge, in the form of the SS and FESS algorithms, between the two disciplines, neural network training algorithms and filter design techniques. In the next section we will talk about more detailed results arising from this general framework.

9.3 Specific achievements

In this section we present the conclusions and specific achievements of our work.

1. We developed the *Similarity Suppression* (SS) algorithm, which starts from a set of training patterns and calculates a set of filters, which are cross-orthogonal, otherwise called mutually orthogonal, to these training patterns. The algorithm is iterative and is based on the idea that the similarities between two patterns can be suppressed if the patterns are continually subtracted from each other using the magnitude of their inner product as a weight. The algorithm results in the suppression of the cross-inner products between the training patterns and the final filters. We kept the inner product between each of the filters and the corresponding training pattern to a constant high value by using a normalisation step after each iteration.

- We presented a theoretical analysis of the changes in the filters' magnitudes during the training and verified it using computer simulations.
- We compared, using computer simulations, the filters produced with the SS algorithm with matched filters. Our filters can tolerate 7 dB more additive input white noise for the same probability of discrimination. The dynamic range required by the recognition system is reduced by 25 dB.
- We proved theoretically and verified by simulations that the SS algorithm is an iterative procedure for calculating the linear combination filters proposed by Caulfield and Maloney [18], which are mutually orthogonal to the patterns used for their creation.
- We discovered, using computer simulations, that the filters produced by the SS algorithm after only 2 iterations perform better than the ones produced by Caulfield's and Maloney's matrix method. 2 dB more additive input white noise can be tolerated for the same performance, which results in an improvement by almost 30% in the probability of discrimination. In addition, the filters produced with the SS algorithm after 2 iterations produce lower outer products than the filters produced after 1500 iterations and the filters produced with the matrix method. This is very important because it

means that the filters produced by the SS algorithm after 2 iterations can be used for pattern recognition or discrimination even when the exact location of the object in the input is not known. In addition, the correlation peak is sharp and allows the detection of other peaks nearby if more than one objects are present in the input. All of the previous benefits come at the cost of a higher required dynamic range. This conclusion also alleviates the one major disadvantage of the SS algorithm compared to the matrix method, which is the computing time it takes to create the filters. The computing time needed for two iterations is comparable to the time it takes for the matrix inversion in Caulfield's method.

- We methodically investigated the reason for the improved performance after 2 iterations and came to some general conclusions.
- We theoretically compared the SS algorithm with a simple formulation of the Hebbian learning law, the "activity-product rule" [3] and showed that the two algorithms are mathematically very similar.
- We used the insight gained by the comparison of the SS algorithm with the Hebbian learning rule, to extend the algorithm to design filters for two cascaded banks of inner product correlators.

2. We also developed the *Feature Enhancement and Similarity Suppression* (FESS) algorithm, which designs filters for multi-class pattern recognition. In multi-class pattern recognition several patterns, which belong to one of several classes, must be recognised and distinguished from patterns belonging to the other classes. Each of the filters designed by the FESS algorithm represents one class. These have high inner products with the patterns that belong to that class and low inner products with all of the other patterns. The FESS algorithm is based on the same principle as the SS algorithm, plus the idea that the features of one pattern can be copied onto another if it is continually added to it.

- We showed that the FESS algorithm can be viewed as a supervised version of the SS algorithm, where desired target values are defined for the inner products between the filters and the training patterns.
- We proved the equivalence between the FESS algorithm and the method for designing synthetic discriminant functions.

- We proved the equivalence between the FESS algorithm and the Widrow-Hoff rule for training neural networks.
- We extended the FESS algorithm to design filters for two cascaded banks of correlators.
- We verified most of our theoretical conclusions for the FESS algorithm using computer simulations.

3. We also feel obliged to point out some limitations of our algorithms and of the filters they design.

- The established equivalences between a bank of correlators and a single layer of neurons [4], and between our algorithms and the unsupervised and supervised formulations of the Hebbian learning law, mean that the shortcomings of the single layer neural networks apply to our systems as well. For example, they can only separate linearly separable sets of patterns. In addition although two or more cascaded banks of correlators can overcome this problem, they cannot be used for verification purposes because they cannot always form closed surfaces around each of the classes, thus eliminating the false positives. Closed surfaces can only be formed if the number of correlators in the first hidden bank is equal to or larger than the number of inputs, which is equal to the size, in pixels, of the input image. Such an optical system is difficult to build for a reasonable image size.

So after completing this project, my personal opinion is that our algorithms need further development to produce useful filters for optical pattern recognition. If we want to perform pattern recognition optically and only one bank of correlators can be built, then at this stage the best choice is probably to use a MICE, or OTF, or MVSDF, or another filter not reviewed in this thesis according to the specific application. Our filters, particularly after 2 iterations, perform better than the equivalent filters produced with the matrix methods (LCFs and SDFs). However, we have not compared them with the more advanced filters, which probably perform even better, because they are designed to tolerate noise, reduce the outer products, etc. while our algorithms do not take these into account. However, I strongly believe that based on the existing framework, we can now improve our filters by borrowing the ideas used in the other filter design techniques (MACE, OTF, MVSDF etc) and adapting them to our iterative algorithms. If two or more cascaded banks of optical correlators can be built, then my opinion is that the filters produced with our algorithms for two banks of correlators will produce the best results of all. If an electronic system is used for pattern recognition, then other advanced techniques exist, which can take advantage of the abilities of the electronic system and perform better than the filters reviewed and developed in this thesis.

Finally, we would like to point out that although the original intention was to design optically implementable algorithms and filters for optical pattern recognition, the results attained are in no way restricted to that type of implementation and so are more generally useful. Nevertheless, the representation of neural network learning algorithms as image operations in the SS and FESS formulations, makes them more suited to optical implementation where operations like Fourier transforms, multiplications and correlations between pairs of images are more easily performed than individual weight updates. We finish this section with table 9.1, which is similar to table 3.1, which summarises the filters' attributes, but with two extra rows, one for the SS algorithm and one for the FESS algorithm.

9.4 Future work

In this section we present some proposals and ideas of how this work should continue. Some of these ideas have not been tested yet by us. On some others, we have already done some work but we did not have the time to investigate them further.

i. System noise analysis and simulations

   So far we have simulated the filter's performance with additive input noise. We think however, that the inherent noise of an optical system will degrade the performance of our filters if it is not taken into consideration at the design stage. A theoretical analysis of the system noise, backed up by the proper computer simulations and then, the modification of the filters during the training to compensate for the system noise, are necessary for good agreement between simulation and experimental results. Neifeld *et. al.* [114] got a 60% disagreement between simulation and experimental results and by including system noise in their analysis, they reduced this gap between the simulated and experimental performance to less than 10%.

ii. Investigation of the FESS algorithm for a small number of iterations

FILTER CHARACTERISTICS											
Filter		T.D.	Discr.		Rec.		N.T.	Corr. plane			D.R.R.
			M-P	M-C	M-P	M-C		P.S.	P.V.	S.R.	
LCF			✓								✓
GMF			✓	✓	✓	✓					
ECP				✓	✓	✓					✓
SDF	F.C.		✓	✓	✓	✓					
	POF		✓	✓	✓	✓		✓			
MVSDF			✓	✓	✓	✓	✓		✓		
MACE		✓	✓	✓	✓	✓		✓		✓	
MICE		✓	✓	✓	✓	✓		✓		✓	
MINACE		✓	✓	✓	✓	✓	✓	✓		✓	
OTF		✓	✓	✓	✓	✓	✓	✓	✓	✓	
SS		✓	✓		✓						✓
FESS			✓	✓	✓	✓					✓

Table 9.1: Summarised filter characteristics. T.D. : Target detection, Discr. : Discrimination, Rec. : Recognition, N.T. : Noise tolerance, Corr. plane : Correlation plane, M-P : Multi-pattern, M-C : Multi-class, P.S. : Peak sharpness, P.V. : Peak variance control, S.R. : Side-lobe (outer product) reduction, D.R.R. : Dynamic range reduction.

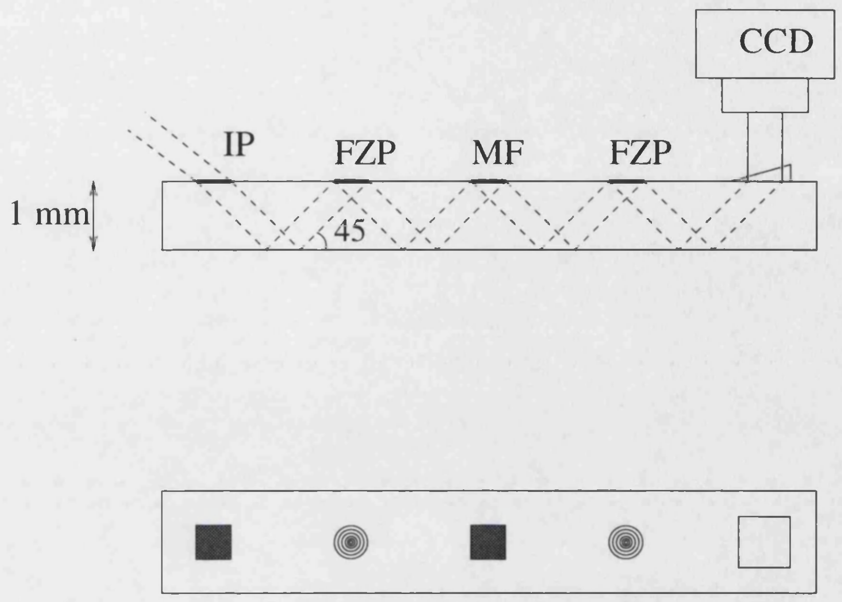


Figure 9.2: Planar correlator. FZP: Fresnel zone plate, MF: Matched filter, IP: Input pattern.

We saw in chapter 6 that the filters produced with the SS algorithm after 2 iterations perform a lot better than the fully converged filters. This implies that we may get similarly beneficial results with the FESS algorithm. Therefore, we need to investigate, mainly using computer simulations, the performance of the filters obtained in the initial iterations of the FESS algorithm.

iii. Comparison of the SS and the FESS algorithms with the more advanced filters

We have proved the mathematical equivalence between the SS algorithm and the LCFs and the FESS algorithm and the SDFs. These comparisons also show us the relationship between our algorithms and the more advanced filter design techniques such as MACE, MICE, etc. However, we need to conduct computer simulations to quantify the performance differences between our filters and the MVSDF, MACE etc. filters.

iv. Optical implementation

The filters were always designed with an optical implementation of the recognition system in mind. We think that one of the most important directions for further work on the subject is to design and implement an optical recog-

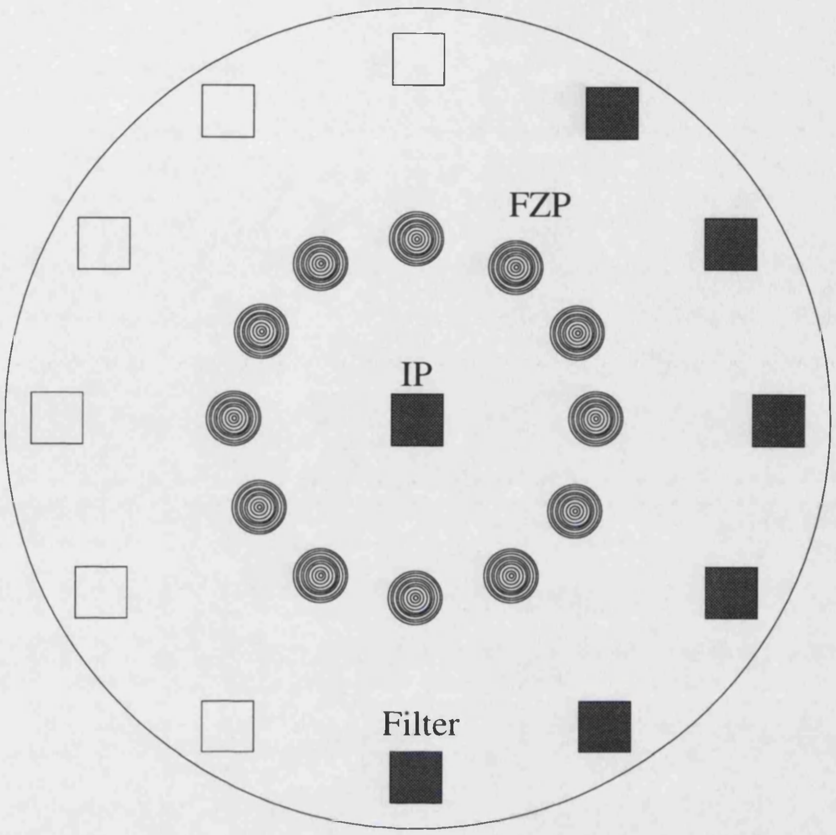


Figure 9.3: Disk planar correlator. FZP: Fresnel zone plate, IP: Input pattern.

nition system. The algorithms have now been developed to some extent, but still require further improvement. Any further theoretical development, however, may be wasted if the filters are not tested in a real optical system. That is the only way to identify all of their weaknesses and strengths. As far as the optical system itself is concerned, we strongly believe that any such system must be compact and versatile, where versatility in this case is the ability to update the filters and the input images dynamically and also change some of their characteristics such as image size, number of grey-levels etc. Although I have no practical experience, I believe that the planar correlator systems [47, 48, 49] hold a lot of promise. I think that the planar correlator design shown in figure 9.2 can be combined with a spinning disc correlator design [36, 37, 38, 39] in a system such as the one shown in figure 9.3. In the correlator shown in figure 9.3, the input image is placed at the centre of the disk. The optimum solution would be to be able to dynamically record the input image on that location. If that is not possible, then an SLM has to be attached there, but then problems will arise when trying to spin the disc with all of the SLM cables attached. The filters are placed around the disc. Again the optimum solution is to be able to record the filters dynamically. The alternative is to etch the filters onto the disc, but this solution drastically reduces the versatility and, therefore, usefulness of the correlator. As the disk spins, a laser beam will enter it from the filter side, bounce in it and exit from the opposite side were a CCD camera can obtain the correlation.

v. Outer product reduction

The increase of the outer products and consequently, the low PCE of the correlations when using our filters, is another issue that must be addressed. Based on an idea similar to that of the MACE filter, which is to try to minimise the average correlation energy, we can add a second weighting term to the first algorithm equation (for the SS algorithm). This second weight can be the total energy of the correlation plane. By using a different convergence factor for each weight, we can emphasise inner product or side-lobe reduction. The first algorithm equation then would take the form:

$$\tilde{\mathbf{g}}_j^{(i)} = \mathbf{g}_j^{(i-1)} - \beta'' \sum_{\substack{k=1 \\ k \neq j}}^{M} \left\{ \mathbf{g}_j^{(i-1)} \cdot \mathbf{s}_k + CE \right\} \mathbf{s}_k \tag{9.1}$$

where CE denotes the energy of the correlation. A different method, but a lot more computationally intensive is to use a number of additional weights, each of which will be equal to the magnitude of one of the largest side-lobes at each iteration.

vi. Simulations of two layer algorithms

We have developed the theory for the two layer versions of the SS and FESS algorithms but we have not done any computer simulations, which will verify it and give us new insights on their performance. We have a fairly good idea of what level of performance to expect based on the performance of similar neural networks, such as better recognition of non-linearly separable patterns, but the issues that are related with the optical implementation of such filters, such as SNR, side-lobe magnitudes, input and system noise tolerance, etc., have not been investigated.

vii. Initially random filters for the SS algorithm

Up to now, we have used the training patterns as the initial versions of the filters in the SS algorithm. We suspect, however, that these filters do not need to contain any of the features of the patterns that they represent. A few simple computer simulations can verify whether initially random filters, which do not contain any of the features of the training patterns, will perform equally well.

viii. Use of a different threshold for each of the correlators in the FESS algorithm

In our recognition tests with the FESS algorithm we have used the same threshold for all of the correlators. However, the differences in the magnitudes of the auto- and cross-inner products of the various filters suggest that we will get better results if each of the correlators has its own threshold.

ix. Different normalisation methods for the FESS algorithm

We have normalised the magnitudes of the filters in the FESS algorithm to keep them stable. We have proved that the FESS algorithm is mathematically equivalent to the Widrow-Hoff algorithm for training neural networks. However, the Widrow-Hoff rule does not have a normalisation equation. This suggests, that the FESS algorithm may be equally successful without a normalisation step. The normalisation step was necessary in the SS algorithm because the filters' magnitudes were decreasing due to the continuous subtractions. However, in the FESS algorithm we add patterns to the filters in addition to subtracting some patterns from them. Therefore it may be possible to keep the magnitudes of the **g** filters stable by carefully choosing the convergence parameters β_1 and β_2.

Another change that we can make to the normalisation in the FESS algorithm, is to normalise the magnitudes of the filters to a value larger than P, for example $2P$, instead of P. Our aim is to increase the auto-inner products and to make them converge as close to P as possible. However, this is not possible when the filters are normalised to P and the auto-inner products have to converge to a lower value. If the filters were normalised to a higher value, that would allow the auto-inner products to increase further. For example, if we multiply all of the pixels of a filter with the number 2, then its magnitude will be equal to $4P$ and its auto-inner products will all double. However, its cross-inner products are going to become two times as large as they were before also. In that case there is no benefit, it is like normalising our correlation outputs to higher values. If we normalise the filters during the training however, at every iteration, we may be able to further decrease the, now doubled, cross-inner products. We are not sure that this is going to happen, but we think that it is an idea worth investigating further.

x. Increase tolerance to additive input noise

The inner product between a filter $\mathbf{g}$ and a pattern with additive noise $\mathbf{s}+\mathbf{n}$ is: $\mathbf{g} \cdot (\mathbf{s}+\mathbf{n}) = \mathbf{g} \cdot \mathbf{s} + \mathbf{g} \cdot \mathbf{n}$. The aim is to eliminate the term $\mathbf{g} \cdot \mathbf{n}$. The main question that arises here is, how can we train our filters to recognise the *statistical characteristics* of the noise, and not specific noise samples. A solution may be to "copy" the MVSDF method which uses the noise covariance matrix to increase the filter's SNR. Because we have established the equivalence between our algorithms and the basic SDFs, we may be able to do a reverse transformation starting from the MVSDF equation and find the terms that must be added to the algorithm equations to get the same result. In addition, we may be able to increase the tolerance to non-zero mean noise by adding constant background images to the training set, a procedure similar to Arsenault's method described in chapter 3.

xi. Selection of optimum number of filters in FESS algorithm

There is a debate on the optimum number of filters for multi-class pattern recognition. We have thought of a method, which although computationally intensive, may guarantee that the minimum number of filters is used for the best possible performance. We can start by creating only one filter and use the FESS algorithm to force the cross-inner products between this filter and all of the patterns of all of the classes to converge to the desired values (a different value for every class). After the algorithm converges, in the case that the result is not acceptable we divide the classes of the training patterns into two groups, create two filters and train the first filter to recognise all of the objects of all of the classes of the first group and the second filter to recognise all of the objects of all of the classes of the second group. We continue with the same procedure, creating additional filters until our requirements (probability of recognition, dynamic range, etc.) are satisfied.

xii. Different encoding method for the training patterns in the FESS algorithm

We used a very simple method to convert the monopolar training patterns into bipolar before using the FESS algorithm. We have thought of a different conversion method, which is more of an interesting experiment: We can multiply each of the monopolar patterns with a different random binary bipolar pattern, thus making the training pattern itself bipolar. Then we use these bipolar patterns for the training. We can repeat the experiment with different random patterns until we get the best results.

xiii. Addition of new patterns and filters after the training.

Another direction of future work is the update of our database of filters and training patterns after the training. Lets assume that at a given time, we have a set of training patterns and that we use the SS algorithm to create the corresponding set of filters. If in the future we want to add an additional pattern and filter pair to our database, it is best if we can do that without having to train the whole system from scratch. Let us call our initial training patterns $\mathbf{s}_1 \ldots \mathbf{s}_M$ and the corresponding filters $\mathbf{g}_1 \ldots \mathbf{g}_M$ as usual. The new pattern that we want to add can be denoted by $\mathbf{s}_{M+1}$. It is straightforward to create the corresponding filter $\mathbf{g}'_{M+1}$ just by applying the algorithm to it for a number of iterations equal to what was used for the original set

$$\mathbf{g}'^{(i)}_{M+1} = \mathbf{g}'^{(i-1)}_{M+1} - \beta \sum_{\substack{k=1 \\ k \neq M+1}}^{M+1} \left\{ \mathbf{g}'^{(i-1)}_{M+1} \cdot \mathbf{s}_k \right\} \mathbf{s}_k \tag{9.2}$$

The normalisation equation is still necessary although it is not written above. The main problem is updating the initial set of filters, $\mathbf{g}_1 \ldots \mathbf{g}_M$. If pattern $\mathbf{s}_{M+1}$ was in the set from the beginning, then instead of the existing set of filters $\mathbf{g}_1 \ldots \mathbf{g}_M$, we would have a slightly different set, $\mathbf{g}'_1 \ldots \mathbf{g}'_M$. The aim is to get this new set of filters without repeating the whole training process. In section 5.4.2 we said that each of the filters can be expressed as a weighted sum of the training patterns:

$$\mathbf{g}_j = c_{j1}\mathbf{s}_1 + \ldots + c_{jM}\mathbf{s}_M \tag{9.3}$$

where

$$c_{jk} = -\beta \sum_{i=1}^{D} \mathbf{g}_j^{(i)} \cdot \mathbf{s}_k \tag{9.4}$$

and D denotes the total number of iterations. If the $\mathbf{s}_{M+1}$ pattern had been part of the initial training set, then at every iteration it would have been subtracted from each of the $\mathbf{g}'$ filters, an extra subtraction compared to those that led to the initial set of $\mathbf{g}$ filters

$$-\beta \left(\mathbf{g}_j'^{(i-1)} \cdot \mathbf{s}_{M+1} \right) \mathbf{s}_{M+1} \tag{9.5}$$

The effect of this extra subtraction on each of the coefficients c_{jk} would be equal to

$$\begin{aligned}\Delta c_{jk} &= -\beta \sum_{i=1}^{D} \left(\mathbf{g}_j^{'(i)} \cdot \mathbf{s}_{M+1}\right)\mathbf{s}_{M+1} \cdot \mathbf{s}_k \\ &= -\beta \mathbf{s}_{M+1} \cdot \mathbf{s}_k \sum_{i=1}^{L} \left(\mathbf{g}_j^{'(i)} \cdot \mathbf{s}_{M+1}\right)\end{aligned} \tag{9.6}$$

If we can calculate $\Delta c_{jk}, \forall j, k$ and then use these coefficients as weights for one more subtraction of patterns $\mathbf{s}_k$ from the filters $\mathbf{g}_j$ we should end up with the updated filters $\mathbf{g}'_j$. The problem with this solution is that we do not have the filters $\mathbf{g}_j^{'(i)}$ for every iteration i, nor is it easy to store all of the initial filters $\mathbf{g}_j^{(i)}, \forall i$, so we cannot calculate Δc_{jk} from equation 9.6. There are several ways however, with which we can approximate the sum in equation 9.6. We know from the analysis of the algorithm carried out in chapter 6, that the cross-inner products $\mathbf{g}_j'(i) \cdot \mathbf{s}_{M+1}$ are rather small for every iteration, except for the first couple of iterations. We have the values of the cross-inner products at the first iteration because initially $g_j = s_j$. We may be able to get a good enough approximation just by using that cross-inner product and the one at the final iteration, which can also be calculated, multiplied by the number of iterations, D. An even better approximation can be achieved if we use our knowledge of the convergence of the algorithm, which is expressed by the total energy index. We can use the initial and final values of the cross-inner products and the total energy index to extrapolate the values in between. Better ways may exist to calculate the correct coefficient correction Δc_{jk}, maybe by using the initial coefficients c_{jk}, which can easily be stored. Simulations are necessary to see the effectiveness of the methods proposed here and the error they introduce.

xiv. Use of $\mathbf{g}$ filters for SDF synthesis

This is more an interesting experiment, than a direction for further work. We can use the FESS algorithm to create our filters and then combine them into an SDF. The performance of this SDF can then be compared to the performance of the SDF created using the original training patterns. The filters created by the FESS algorithm contain only the main features of the patterns which they represent and have discarded the similarities between patterns that belong to different classes. Therefore it is conceivable that the SDF which will be created by them will perform better since it will not contain the unnecessary information that is contained in the original training patterns.

Appendix A
Mathematical definitions

In this appendix we present some mathematical definitions which are necessary for the understanding of the thesis.

A.1 Expected value - variance - standard deviation

The **expected value**[1] *or* **mean** *of a random variable is defined by the integral :*

$$E\{\mathrm{x}\} = \int_{-\infty}^{\infty} x f(x)\, dx \tag{A.1}$$

where $f(x)$ is the **probability density function** *of the random variable* x *and is defined by*

$$f(x) = \frac{dF(x)}{dx} \tag{A.2}$$

and $F(x)$ is the **distribution function** *of* x

$$F(x) = P\{\mathrm{x} \leq x\} \tag{A.3}$$

defined for every x from $-\infty$ to ∞. For discrete type random variables, the expected value is given by a sum:

$$E\{\mathrm{x}\} = \sum_i p_i x_i, \qquad p_i = \mathbf{P}\{\mathrm{x} = x_i\} \tag{A.4}$$

The **variance** *of a random variable* x *is by definition the integral*

$$\sigma^2 = \int_{-\infty}^{\infty} (x - E\{\mathrm{x}\})^2 f(x)\, dx \tag{A.5}$$

The variance is also given by the following equation

$$\sigma^2 = E\{\mathrm{x}^2\} - E^2\{\mathrm{x}\} \tag{A.6}$$

The positive constant σ is called the **standard deviation** *of* x.

A.2 Correlation and covariance matrices

The **covariance**[1] *C of two random variables* x *and* y *is by definition the number*

$$\begin{aligned} C &= E\big\{(\mathrm{x} - E\{\mathrm{x}\})(\mathrm{y} - E\{\mathrm{y}\})\big\} \\ &= E\{\mathrm{xy}\} - E\{\mathrm{x}\}E\{\mathrm{y}\} \end{aligned} \tag{A.7}$$

[1]The following definitions are adapted from Papoulis, 1991 [115]

For complex random variables, the covariance is

$$\begin{aligned} C &= E\big\{(\mathrm{x} - E\{\mathrm{x}\})(\mathrm{y}^* - E\{\mathrm{y}\}^*)\big\} \\ &= E\{\mathrm{xy}^*\} - E\{\mathrm{x}\}E\{\mathrm{y}^*\} \end{aligned} \tag{A.8}$$

The **correlation matrix** *of the random vector* $\mathbf{X} = [\mathrm{x}_1, \dots, \mathrm{x}_n]$ *is by definition*

$$R_n = \begin{pmatrix} R_{11} & \cdots & R_{1n} \\ \vdots & & \vdots \\ R_{n1} & \cdots & R_{nn} \end{pmatrix} \tag{A.9}$$

where $R_{ij} = E\{\mathrm{x}_i\mathrm{x}_j^*\} = R_{ji}^*$. *The* **covariance matrix** *of the random vector* $\mathbf{X} = [\mathrm{x}_1, \dots, \mathrm{x}_n]$ *is*

$$C_n = \begin{pmatrix} C_{11} & \cdots & C_{1n} \\ \vdots & & \vdots \\ C_{n1} & \cdots & C_{nn} \end{pmatrix} \tag{A.10}$$

where $C_{ij} = R_{ij} - E\{\mathrm{x}_i\}E\{\mathrm{x}_j\}^* = C_{ji}^*$. *The correlation matrix can also be written*

$$R_n = E\{\mathbf{X}^T\mathbf{X}^*\} \tag{A.11}$$

where $\mathbf{X}^T$ *is the transpose of* $\mathbf{X}$. *The covariance matrix* C_n *is the correlation matrix of the "centered" random variables* $\mathrm{x}_i - E\{\mathrm{x}_i\}$. *If* $E\{\mathrm{x}_i\} = 0, \forall i, C_n = R_n$.

A.3 Fourier transform

Given[2] *an arbitrary, complex-valued function* $f(x)$, *the integral*

$$F(\xi) = \int_{-\infty}^{\infty} f(\alpha)e^{-j2\pi\xi\alpha}\,d\alpha \tag{A.12}$$

is called the **Fourier transform** *of* $f(x)$. *The integral of equation A.12 exists*[3] *for every function* $f(x)$ *which accurately describes a real physical quantity [116].* $f(x)$ *can be obtained from* $F(\xi)$ *if equation A.12 is inverted*

$$f(x) = \int_{-\infty}^{\infty} F(\alpha)e^{j2\pi\alpha x}\,d\alpha \tag{A.13}$$

assuming that $F(\alpha)$ *exists.* $f(x)$ *is called the* **inverse Fourier transform** *of* $F(\xi)$ *and the two functions are known as a* **Fourier transform pair**.

[2]The following definitions have been adapted from Gaskill, 1978 [6]

[3]See Gaskill, 1978 [6] for a description of the conditions required

Given the function $f(x, y)$, its two-dimensional Fourier transform is given by the integral

$$F(\xi, \eta) = \iint_{-\infty}^{\infty} f(\alpha, \beta) e^{-j2\pi(\alpha\xi + \beta\eta)} \, d\alpha \, d\beta \tag{A.14}$$

The inverse Fourier transform of $F(\xi, \eta)$ is

$$f(x, y) = \iint_{-\infty}^{\infty} F(\alpha, \beta) e^{j2\pi(\alpha x + \beta y)} \, d\alpha \, d\beta \tag{A.15}$$

A.4 Convolution and correlation

The convolution and the correlation are two mathematical computations between two functions. They are described by equations A.16 and A.17 respectively

$$g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} f(\alpha) h(x - \alpha) \, d\alpha \tag{A.16}$$

$$g(x) = f(x) \otimes h(x) = \int_{-\infty}^{\infty} f(\alpha) h(x + \alpha) \, d\alpha \tag{A.17}$$

For two-dimensional functions $f(x, y)$ and $h(x, y)$, the convolution and correlation operations are described by equations A.18 and A.19 respectively

$$g(x, y) = \iint_{-\infty}^{\infty} f(\alpha, \beta) h(x - \alpha, y - \beta) \, d\alpha \, d\beta \tag{A.18}$$

$$g(x, y) == \iint_{-\infty}^{\infty} f(\alpha, \beta) h(x + \alpha, y + \beta) \, d\alpha \, d\beta \tag{A.19}$$

The convolution and correlation equations may also be written in a discrete form [7]

$$g(m, n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(i, j) h(m - i, n - j) \tag{A.20}$$

$$g(m, n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(i, j) h(m + i, n + j) \tag{A.21}$$

One of the most important theorems of signal processing is the **convolution theorem**, *which states that the convolution of two functions in the time domain is equal to the inverse Fourier transform of the multiplication of these two functions in the frequency domain, that is the multiplication of their Fourier transforms [117].*

$$g(x) = f(x) * h(x) = IFT\{F(\xi) H(\xi)\} \tag{A.22}$$

A.5 Inner and outer products

The correlation of two two-dimensional functions, is itself a two-dimensional function. The value of that function at the origin $g(0,0)$ *is called the* **central peak** *of the correlation or* **inner product** *and it is equal to*

$$g(0,0) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(i,j)h(i,j) \tag{A.23}$$

in the discrete form. All of the other points of the function $g(m,n)$ *are called the* **outer products** *of the correlation.*

Appendix B
Magnitudes of the un-normalised filters

The magnitudes of the filters are equal to the squared Euclidean norm shown in the following equation:

$$\text{Euclidean norm:} \qquad \|\mathrm{s}\|_2 = \left(\sum_{i=1}^{N} s_i^2\right)^{1/2} = (\mathrm{s}\cdot\mathrm{s})^{1/2} = (\mathrm{s}^T\mathrm{s})^{1/2} \tag{B.1}$$

In this appendix we are going to investigate whether these magnitudes are going to increase, or decrease, or stay constant during the training with the SS algorithm (equation 5.19) and the FESS algorithm (equations 7.1 and 7.2), without the normalisation step. Equations 7.1 and 7.2 can be combined into the following equation:

$$\mathrm{g}_j^{(i)} = \mathrm{g}_j^{(i-1)} + \beta_1 \sum_{k=1}^{L_j}\left(P - \mathrm{g}_j^{(i-1)T}\mathrm{s}_k\right)\mathrm{s}_k - \boldsymbol{\beta}_2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)\boldsymbol{s}_m \tag{B.2}$$

We have used a different font to write the third term on the right hand side of equation B.2. We did that because that term describes the SS algorithm. In other words, if we used the SS algorithm to design the filters, we would only subtract patterns from the filters, using the third term but with slightly different summation limits. In the following mathematical analysis we are going to use different fonts for the terms that are derived from the FESS algorithm terms and for the terms that would be there if only the SS algorithm was used. We use two different symbols for example, g_j and $\boldsymbol{g}_j$, to denote the j_{th} filter. Both symbols refer to the same filter. The only criterion of which symbol to use each time is whether the whole term that the symbol is in, would exist if only the SS algorithm was used for the training. At the end of our mathematical analysis we are going to isolate these terms and discuss the effect of the SS algorithm on the filters' magnitudes when more than two training patterns are used. Equation B.2 can also be written in the following form:

$$\begin{aligned} \mathrm{g}_j^{(i)} &= \mathrm{g}_j^{(i-1)} + \beta_1 \sum_{k=1}^{L_j}\mathrm{s}_k\left(P - \mathrm{s}_k^T\mathrm{g}_j^{(i-1)}\right) - \boldsymbol{\beta}_2 \sum_{m=1}^{M-L_j}\boldsymbol{s}_m\left(\boldsymbol{s}_m^T\boldsymbol{g}_j^{(i-1)}\right) \\ \Rightarrow \mathrm{g}_j^{(i)} &= \beta_1 \sum_{k=1}^{L_j}\mathrm{s}_k\left(P - \mathrm{s}_k^T\mathrm{g}_j^{(i-1)}\right) + \left(I - \boldsymbol{\beta}_2 \sum_{m=1}^{M-L_j}\boldsymbol{s}_m\boldsymbol{s}_m^T\right)\boldsymbol{g}_j^{(i-1)} \end{aligned} \tag{B.3}$$

The filters' magnitudes can be calculated in the following manner[1]:

$$
\begin{aligned}
\mathbf{g}_j^{(i)T}\mathbf{g}_j^{(i)} \overset{(i-1)}{\overset{\rightarrow}{=}} & \left[\beta_1 \sum_{k=1}^{L_j} \mathbf{s}_k\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)+\left(I-\boldsymbol{\beta}_2\sum_{m=1}^{M-L_j}\boldsymbol{s}_m\boldsymbol{s}_m^T\right)\boldsymbol{g}_j\right]^T \\
& \cdot\left[\beta_1 \sum_{k'=1}^{L_j} \mathbf{s}_{k'}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)+\left(I-\boldsymbol{\beta}_2\sum_{m'=1}^{M-L_j}\boldsymbol{s}_{m'}\boldsymbol{s}_{m'}^T\right)\boldsymbol{g}_j\right] \\
= & \left[\beta_1 \sum_{k=1}^{L_j} \mathbf{s}_k^T\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)+\boldsymbol{g}_j^T\left(I-\boldsymbol{\beta}_2\sum_{m=1}^{M-L_j}\boldsymbol{s}_m\boldsymbol{s}_m^T\right)\right] \\
& \cdot\left[\beta_1 \sum_{k'=1}^{L_j} \mathbf{s}_{k'}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)+\left(I-\boldsymbol{\beta}_2\sum_{m'=1}^{M-L_j}\boldsymbol{s}_{m'}\boldsymbol{s}_{m'}^T\right)\boldsymbol{g}_j\right] \\
= & \beta_1^2\sum_{k=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\mathbf{s}_k^T\sum_{k'=1}^{L_j}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)\mathbf{s}_{k'}+\beta_1\sum_{k=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\mathbf{s}_k^T\mathbf{g}_j \\
& -\beta_1\beta_2\sum_{k=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\mathbf{s}_k^T\sum_{m'=1}^{M-L_j}\mathbf{s}_{m'}\mathbf{s}_{m'}^T\mathbf{g}_j+\mathbf{g}_j^T\beta_1\sum_{k'=1}^{L_j}\mathbf{s}_{k'}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right) \\
& +\boldsymbol{g}_j^T\boldsymbol{g}_j-\boldsymbol{g}_j^T\boldsymbol{\beta}_2\sum_{m'=1}^{M-L_j}\boldsymbol{s}_{m'}\boldsymbol{s}_{m'}^T\boldsymbol{g}_j-\mathbf{g}_j^T\beta_2\sum_{m=1}^{M-L_j}\mathbf{s}_m\mathbf{s}_m^T\beta_1\sum_{k'=1}^{L_j}\mathbf{s}_{k'}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right) \\
& +\boldsymbol{g}_j^T\boldsymbol{\beta}_2\sum_{m=1}^{M-L_j}\boldsymbol{s}_m\boldsymbol{s}_m^T\boldsymbol{g}_j+\boldsymbol{g}_j^T\boldsymbol{\beta}_2^2\sum_{m=1}^{M-L_j}\boldsymbol{s}_m\boldsymbol{s}_m^T\sum_{m'=1}^{M-L_j}\boldsymbol{s}_{m'}\boldsymbol{s}_{m'}^T\boldsymbol{g}_j \\
= & \beta_1^2\sum_{k=1}^{L_j}\sum_{k'=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)\mathbf{s}_k^T\mathbf{s}_{k'}+\beta_1\sum_{k=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\mathbf{s}_k^T\mathbf{g}_j \\
& -\beta_1\beta_2\sum_{k=1}^{L_j}\sum_{m'=1}^{M-L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j\right)\mathbf{s}_{m'}^T\mathbf{g}_j\mathbf{s}_k^T\mathbf{s}_{m'}+\beta_1\sum_{k'=1}^{L_j}\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)\mathbf{g}_j^T\mathbf{s}_{k'} \\
& +\boldsymbol{g}_j^T\boldsymbol{g}_j-\boldsymbol{\beta}_2\sum_{m'=1}^{M-L_j}\left(\boldsymbol{s}_{m'}^T\boldsymbol{g}_j\right)^2-\beta_1\beta_2\sum_{m=1}^{M-L_j}\sum_{k'=1}^{L_j}\mathbf{g}_j^T\mathbf{s}_m\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j\right)\mathbf{s}_m^T\mathbf{s}_{k'} \\
& -\boldsymbol{\beta}_2\sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^T\boldsymbol{s}_m\right)^2+\boldsymbol{\beta}_2^2\sum_{m=1}^{M-L_j}\sum_{m'=1}^{M-L_j}\boldsymbol{g}_j^T\boldsymbol{s}_m\boldsymbol{s}_{m'}^T\boldsymbol{g}_j\boldsymbol{s}_m^T\boldsymbol{s}_{m'}
\end{aligned}
$$

So the magnitude of the filter $\mathbf{g}_j$ is:

$$
\begin{aligned}
\mathbf{g}_j^{(i)T}\mathbf{g}_j^{(i)} = & \boldsymbol{g}_j^{(i-1)T}\boldsymbol{g}_j^{(i-1)}+\boldsymbol{\beta}_2^2\sum_{m=1}^{M-L_j}\sum_{m'=1}^{M-L_j}\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\boldsymbol{s}_{m'}^T\boldsymbol{g}_j^{(i-1)}\boldsymbol{s}_m^T\boldsymbol{s}_{m'}-2\boldsymbol{\beta}_2\sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2 \\
& +\beta_1^2\sum_{k=1}^{L_j}\sum_{k'=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j^{(i-1)}\right)\left(P-\mathbf{s}_{k'}^T\mathbf{g}_j^{(i-1)}\right)\mathbf{s}_k^T\mathbf{s}_{k'}+2\beta_1\sum_{k=1}^{L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j^{(i-1)}\right)\mathbf{s}_k^T\mathbf{g}_j^{(i-1)} \\
& -2\beta_1\beta_2\sum_{k=1}^{L_j}\sum_{m'=1}^{M-L_j}\left(P-\mathbf{s}_k^T\mathbf{g}_j^{(i-1)}\right)\mathbf{s}_{m'}^T\mathbf{g}_j^{(i-1)}\mathbf{s}_k^T\mathbf{s}_{m'}
\end{aligned}
\tag{B.4}
$$

[1]The $\mathbf{g}_j$ filters on the right hand side of the following equations are all in the $i_{th}-1$ iteration. However, for the sake of clarity, the $^{(i-1)}$ index will be omitted until the last equation.

B.1 Analysis for the SS algorithm

The first three terms in the right hand side of equation B.4 are the ones that would result if we had used the SS algorithm instead of the FESS algorithm to design the filters. We can see that they are very similar to the terms in equation 5.32 in section 5.3. We can isolate them from the remaining terms in equation B.4 and estimate the slope, or rate of change, of the magnitudes of the **g** filters when the SS algorithm is used and more than two training patterns exist

$$\mathbf{g}_j^{(i)T}\mathbf{g}_j^{(i)} - \boldsymbol{g}_j^{(i-1)T}\boldsymbol{g}_j^{(i-1)} = \beta_2^2 \sum_{m=1}^{M-L_j}\sum_{m'=1}^{M-L_j} \boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\boldsymbol{s}_{m'}^T\boldsymbol{g}_j^{(i-1)}\boldsymbol{s}_m^T\boldsymbol{s}_{m'} - 2\beta_2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2 \tag{B.5}$$

The first term on the right hand side of equation B.5 can be expanded to give the following terms:

$$\beta_2^2 \sum_{m=1}^{M-L_j}\sum_{m'=1}^{M-L_j} \boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\boldsymbol{s}_{m'}^T\boldsymbol{g}_j^{(i-1)}\boldsymbol{s}_m^T\boldsymbol{s}_{m'} = \beta_2^2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2 \|\boldsymbol{s}_m\|^2$$

$+$ a sum of cross terms of the form:

$$\beta_2^2\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_1\boldsymbol{s}_2^T\boldsymbol{g}_j^{(i-1)}\boldsymbol{s}_1^T\boldsymbol{s}_2 \qquad \text{for example when } m = 1 \text{ and } m' = 2 \tag{B.6}$$

So using equation B.6, equation B.5 becomes:

$$\begin{aligned}\|\mathbf{g}_j^{(i)}\|^2 - \|\boldsymbol{g}_j^{(i-1)}\|^2 &= \beta_2^2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2\|\boldsymbol{s}_m\|^2 - 2\beta_2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2 \\ &+ \quad \text{sum of cross terms} \\ &= \beta_2 \sum_{m=1}^{M-L_j}\left(\boldsymbol{g}_j^{(i-1)T}\boldsymbol{s}_m\right)^2\left(\beta_2\|\boldsymbol{s}_m\|^2 - 2\right) \\ &+ \quad \text{sum of cross terms}\end{aligned} \tag{B.7}$$

The investigation of the sign of the right hand side of equation B.7 is very similar to that shown in section 5.3. The right hand side of the equation is two sums of terms. We can ignore the cross terms for the time being and investigate the sign of each of the other terms, which depends on the value of β_2.

- If $\beta_2 = \frac{2}{\|\boldsymbol{s}_m\|^2}$ then the corresponding term is equal to zero. Since all of the training patterns are normalised and $\|\boldsymbol{s}_m\|^2$ is constant $\forall m$, if $\beta_2 = \frac{2}{\|\boldsymbol{s}_m\|^2}$ then all of the terms in the sum will be equal to zero and the slope of the norm of $\mathbf{g}_j$ will depend on the cross terms.

- If $\beta_2 < \frac{2}{||s_m||^2}$ then the terms in the sum are all negative.
- If $\beta_2 > \frac{2}{||s_m||^2}$ then all of the terms in the sum are positive.

In section 5.2.3 we chose to use a convergence parameter which corresponds to $\beta_2 = 1/PM$. That is, β_2 is smaller than $2/||s||$ ($P = ||s||$) and according to the previous analyses the terms in the sum in the right hand side of equation B.7 will be negative. We have to consider the sum of the cross terms to find the sign of the slope. If the training patterns are monopolar, then the cross terms will all be positive. If the training patterns are bipolar, then the sum will consist of positive and negative terms and overall it will be smaller. In either case however, this sum of cross terms will probably be lower than the first sum in the right hand side of the equation because of the squared β_2 in front of it. Therefore, we can conclude that with our choice of convergence parameter, the slope of the $\mathbf{g}_j$ norm will be negative and the norm decreases. Eventually, as the inner products $\left(g_j^{(i-1)T} s_m\right)^2$ decrease, the right hand side of equation B.7 tends to zero. So after a number of iterations the norm of $\mathbf{g}_j$ will stabilise to a very low value.

B.2 Analysis for the FESS algorithm

The analysis is similar for the FESS algorithm, when all of the terms in equation B.4 must be considered. We saw in the SS analysis that with our choice of convergence parameter, the second and third term in the first line of the right hand side of equation B.4 are negative and cause the decrease of the filters' magnitude. The next two terms in the second line in the same expression, which we can see again below

$$\beta_1^2 \sum_{k=1}^{L_j} \sum_{k'=1}^{L_j} \left(P - \mathbf{s}_k^T \mathbf{g}_j^{(i-1)}\right)\left(P - \mathbf{s}_{k'}^T \mathbf{g}_j^{(i-1)}\right)\mathbf{s}_k^T \mathbf{s}_{k'} + 2\beta_1 \sum_{k=1}^{L_j} \left(P - \mathbf{s}_k^T \mathbf{g}_j^{(i-1)}\right)\mathbf{s}_k^T \mathbf{g}_j^{(i-1)} \tag{B.8}$$

are the corresponding terms of the FESS algorithm. These terms both have a plus sign in front of them, however, we can be sure that they are positive only in the case when the training patterns are monopolar and all of the cross-inner products $\mathbf{s}_k^T \mathbf{g}_j^{(i-1)}$ are positive. For bipolar training patterns, we cannot predict with certainty whether these two terms are going to be positive or negative. The last term in equation B.4 is usually smaller compared to the other terms due to the product $\beta_1\beta_2$ in front of it. The overall sign of the slope of the g norm, $||\mathbf{g}||^2$, depends on the number of patterns in each of the classes compared to the total number of patterns. Assuming that the training patterns are monopolar, the ones that belong to other classes and are subtracted from the filter will cause it's magnitude to decrease according to the second and third term in the first line in equation B.4. The ones that belong to the class that the filter represents, will cause it's magnitude to increase according to the next two terms in equation B.4. Usually, but not always, the total number of patterns will be a lot larger than the number of patterns in each class, therefore the negative terms will be larger and the magnitudes of the filters will decrease.

Appendix C

Training set

C.1 Training set for the FESS algorithm

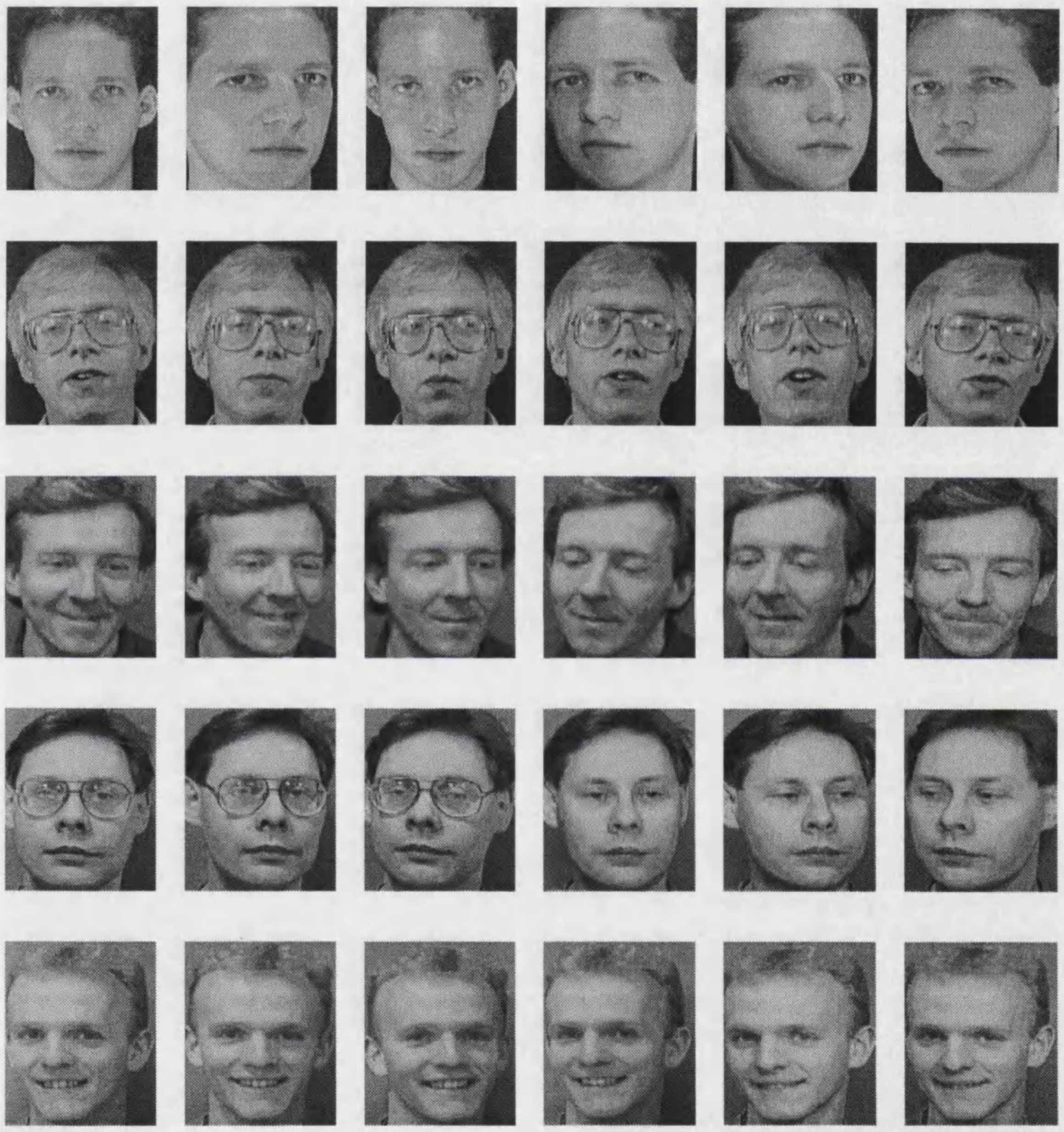


Figure C.1: The examples that were used in the training set for the first five people.

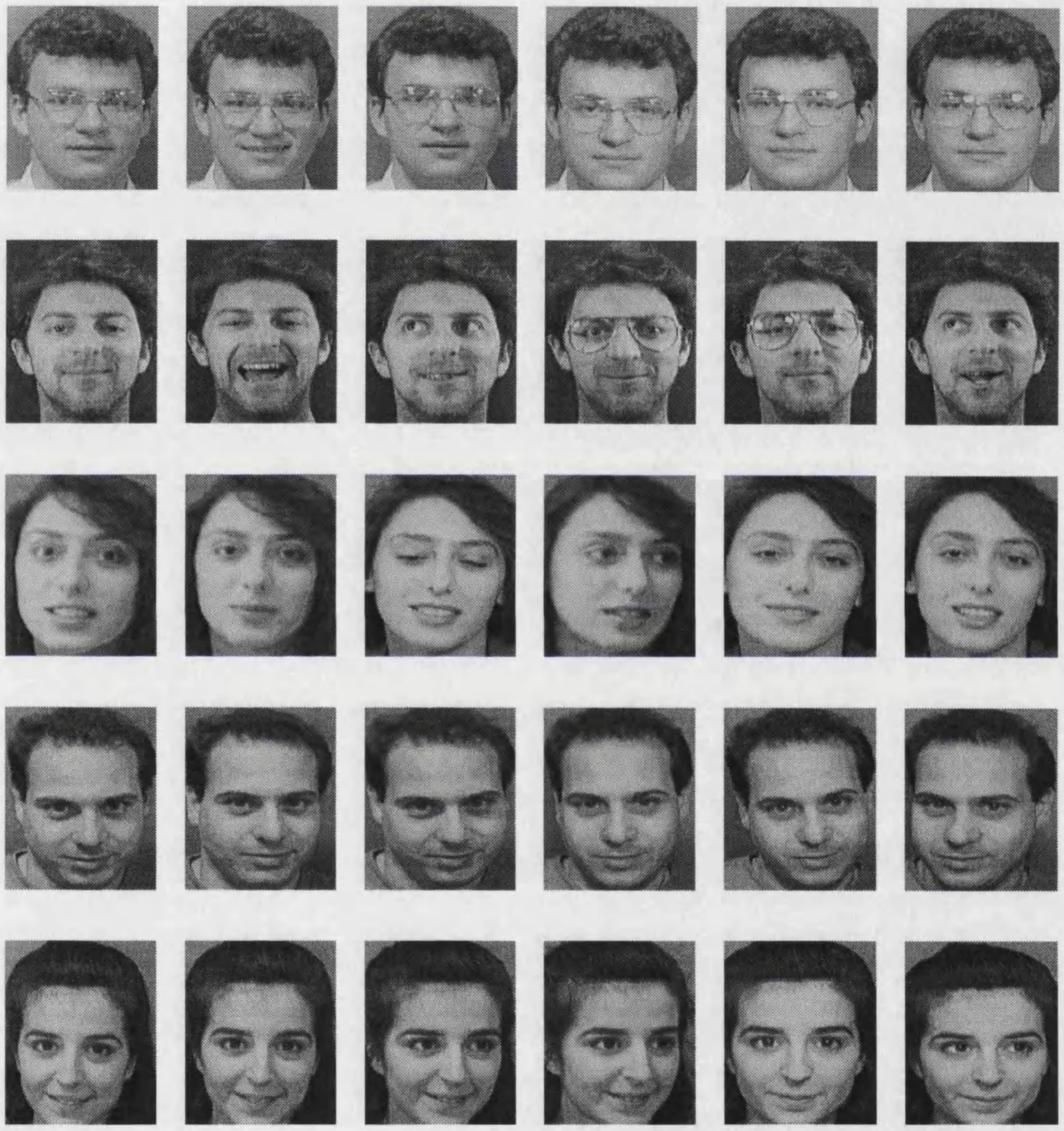
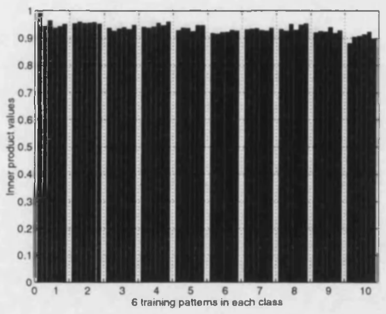
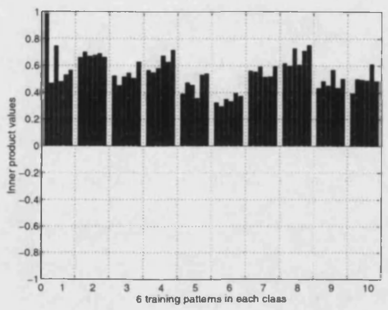


Figure C.2: The examples that were used in the training set for the last five people.

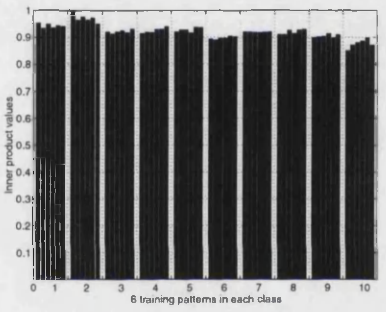
Appendix D
Cross inner product matrices



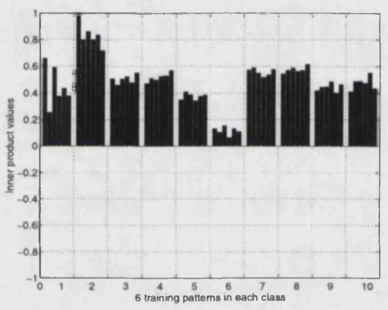
(a) Filt. 1 Mon



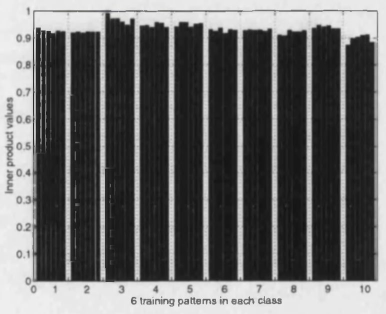
(b) Filt. 1 bip



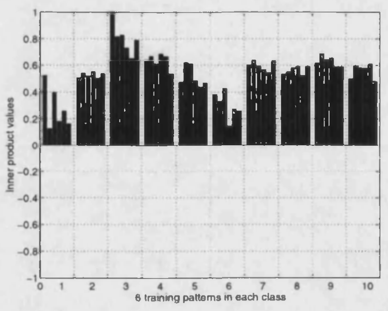
(c) Filt. 2 Mon



(d) Filt. 2 bip

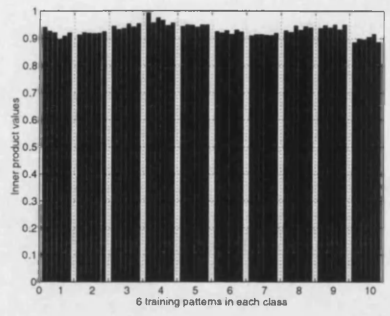


(e) Filt. 3 Mon

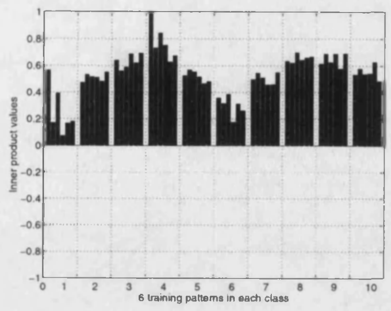


(f) Filt. 3 bip

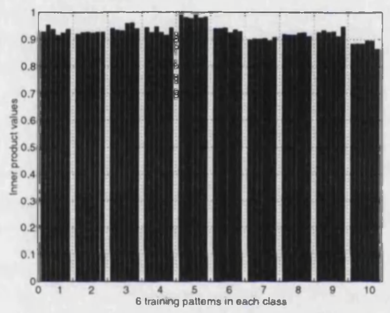
Figure D.3: First three rows of the initial cross-inner product matrices for monopolar and bipolar patterns.



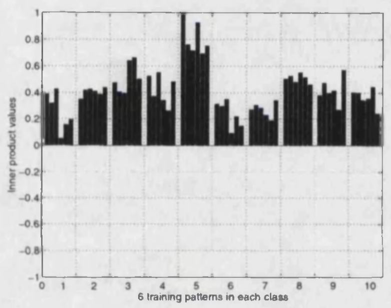
(a) Filt. 4 Mon



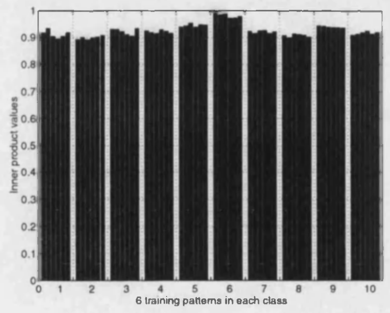
(b) Filt. 4 bip



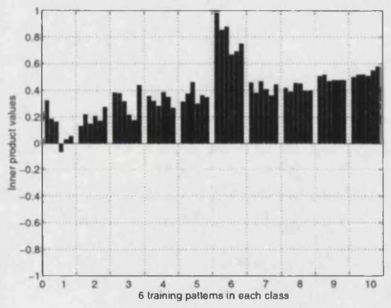
(c) Filt. 5 Mon



(d) Filt. 5 bip

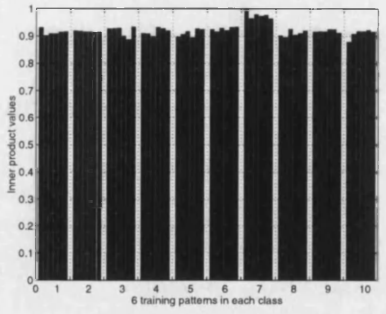


(e) Filt. 6 Mon

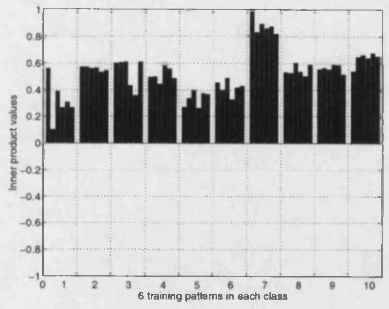


(f) Filt. 6 bip

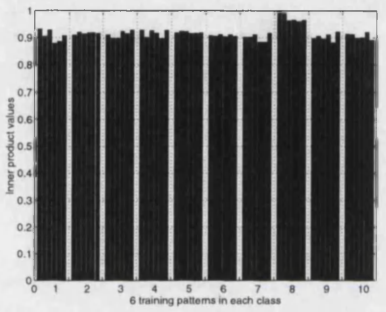
Figure D.4: Fourth to sixth rows of the initial cross-inner product matrices for monopolar and bipolar patterns.



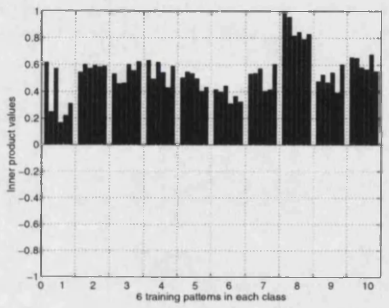
(a) Filt. 7 Mon



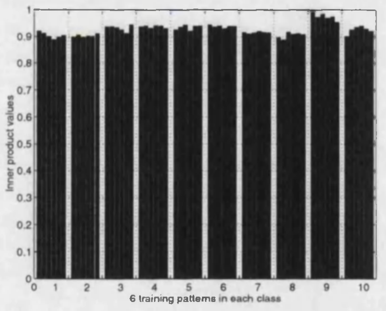
(b) Filt. 7 bip



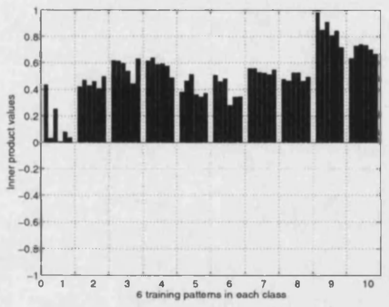
(c) Filt. 8 Mon



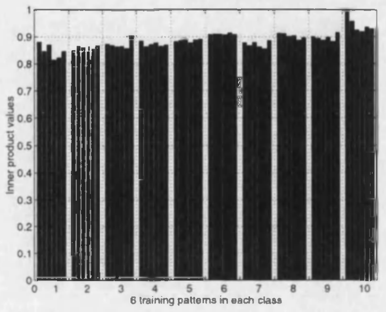
(d) Filt. 8 bip



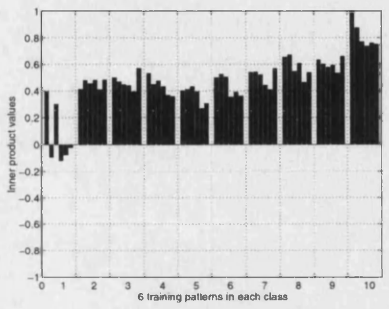
(e) Filt. 9 Mon



(f) Filt. 9 bip

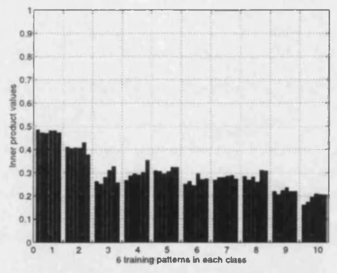


(g) Filt. 10 Mon

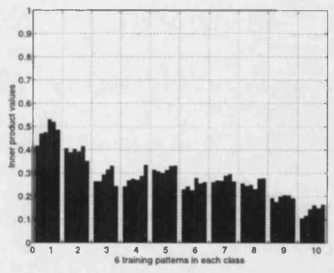


(h) Filt. 10 bip

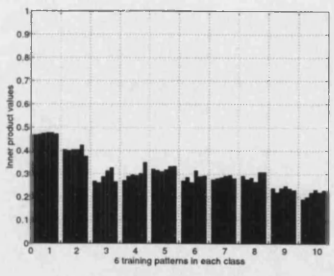
Figure D.5: Last four rows of the initial cross-inner product matrices for monopolar and bipolar patterns.



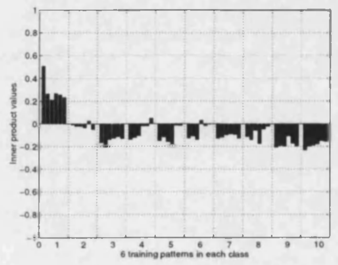
(a) Filt. 1 Mon 1



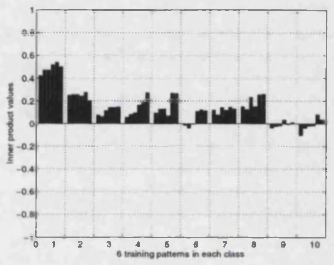
(b) Filt. 1 Mon Mean



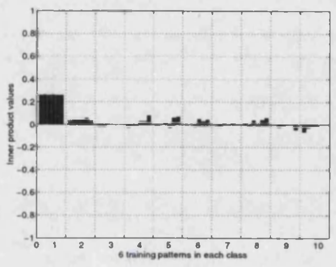
(c) Filt. 1 Mon Rand



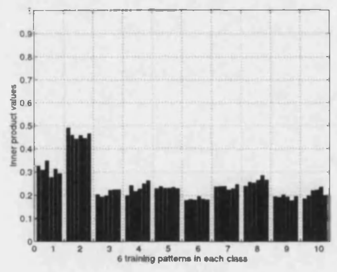
(d) Filt. 1 Bip 1



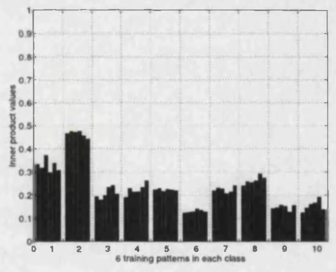
(e) Filt. 1 Bip Mean



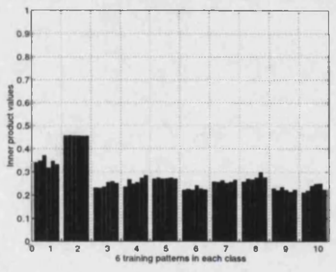
(f) Filt. 1 Bip Rand



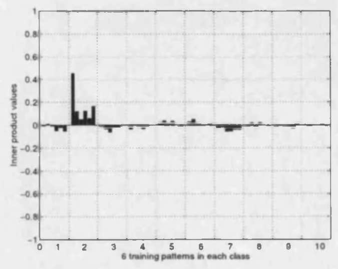
(g) Filt. 2 Mon 1



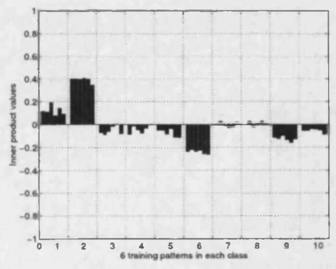
(h) Filt. 2 Mon Mean



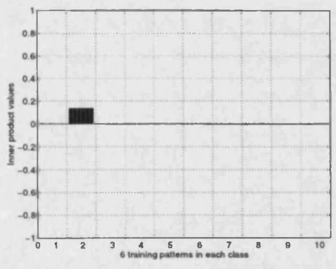
(i) Filt. 2 Mon Rand



(j) Filt. 2 Bip 1

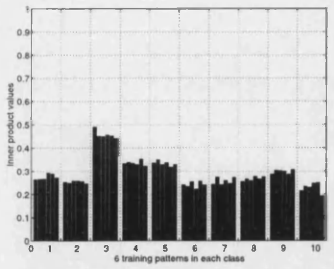


(k) Filter. 2 Bip Mean

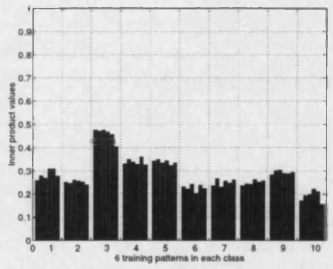


(l) Filt. 2 Bip Rand

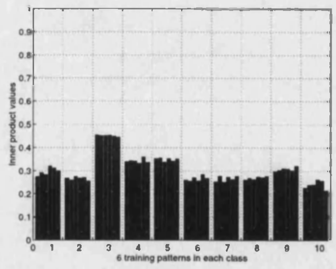
Figure D.6: First and second rows of the cross-inner product matrices for monopolar and bipolar patterns and all three initial filter values.



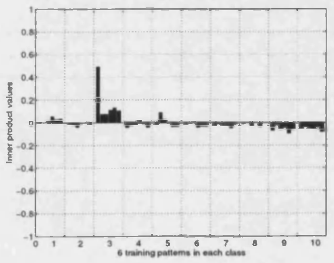
(a) Filt. 3 Mon 1



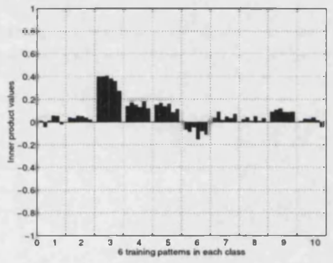
(b) Filt. 3 Mon Mean



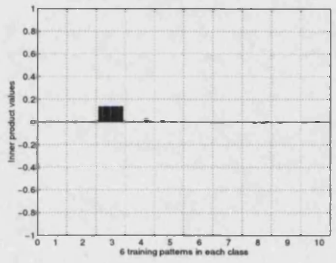
(c) Filt. 3 Mon Rand



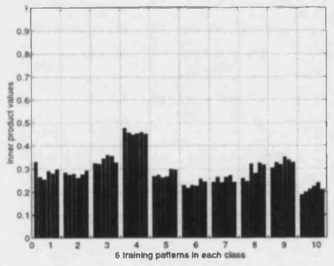
(d) Filt. 3 Bip 1



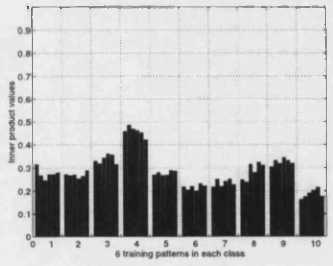
(e) Filt. 3 Bip Mean



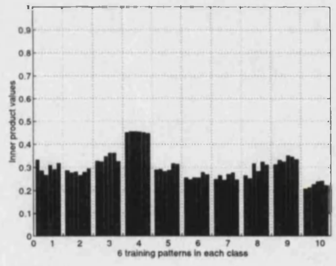
(f) Filt. 3 Bip Rand



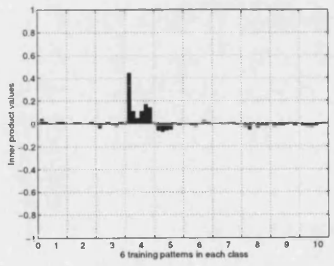
(g) Filt. 4 Mon 1



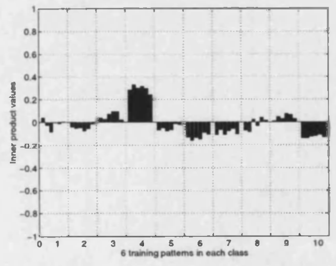
(h) Filt. 4 Mon Mean



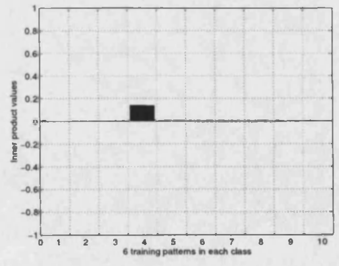
(i) Filt. 4 Mon Rand



(j) Filt. 4 Bip 1

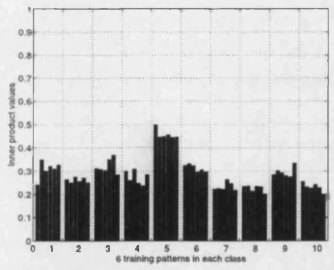


(k) Filter. 4 Bip Mean

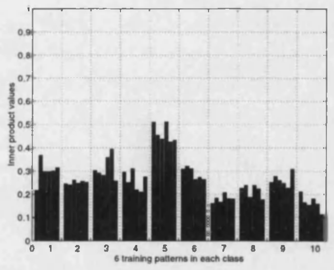


(l) Filt. 4 Bip Rand

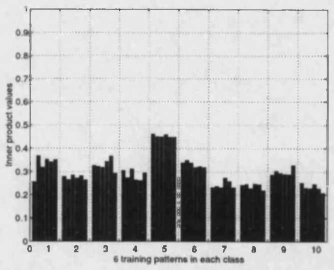
Figure D.7: Third and fourth rows of the cross-inner product matrices for monopolar and bipolar patterns and all three initial filter values.



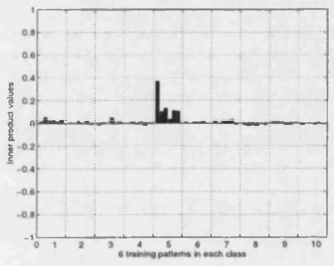
(a) Filt. 5 Mon 1



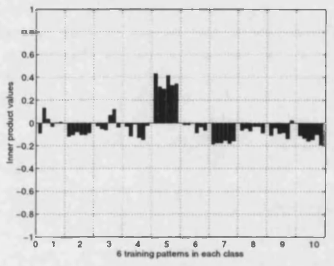
(b) Filt. 5 Mon Mean



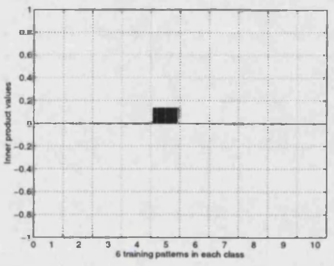
(c) Filt. 5 Mon Rand



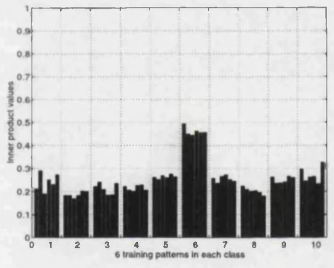
(d) Filt. 5 Bip 1



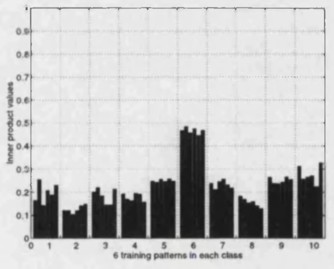
(e) Filt. 5 Bip Mean



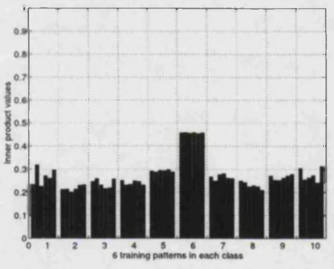
(f) Filt. 5 Bip Rand



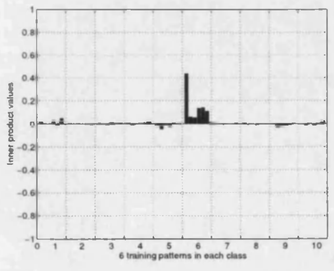
(g) Filt. 6 Mon 1



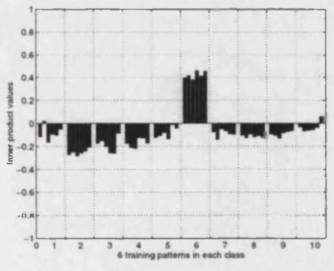
(h) Filt. 6 Mon Mean



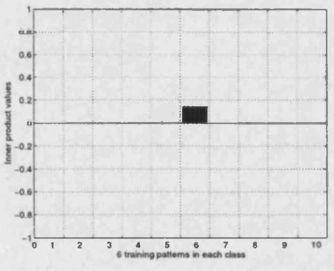
(i) Filt. 6 Mon Rand



(j) Filt. 6 Bip 1

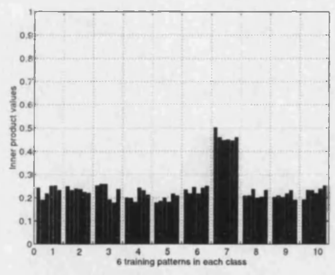


(k) Filter. 6 Bip Mean

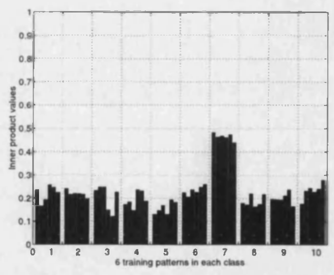


(l) Filt. 6 Bip Rand

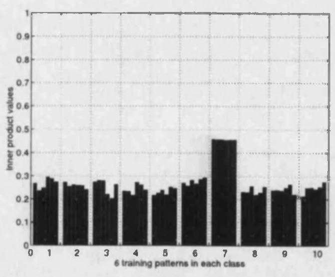
Figure D.8: Fifth and sixth rows of the cross-inner product matrices for monopolar and bipolar patterns and all three initial filter values.



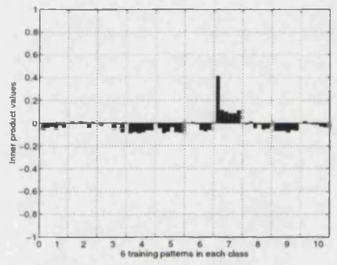
(a) Filt. 7 Mon 1



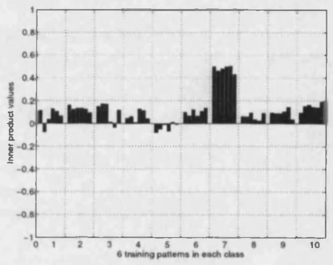
(b) Filt. 7 Mon Mean



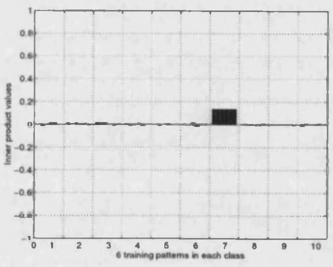
(c) Filt. 7 Mon Rand



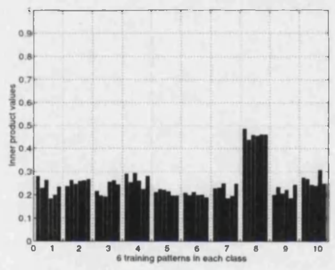
(d) Filt. 7 Bip 1



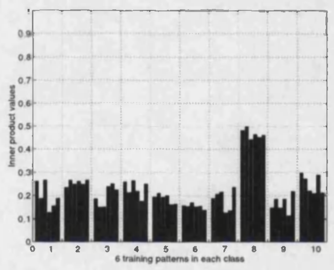
(e) Filt. 7 Bip Mean



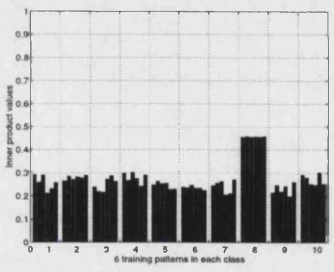
(f) Filt. 7 Bip Rand



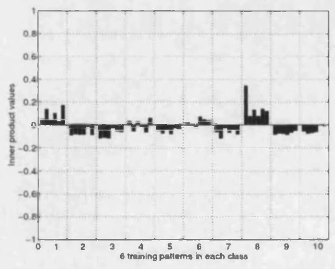
(g) Filt. 8 Mon 1



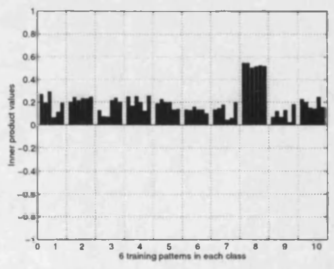
(h) Filt. 8 Mon Mean



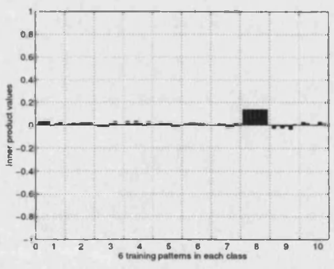
(i) Filt. 8 Mon Rand



(j) Filt. 8 Bip 1

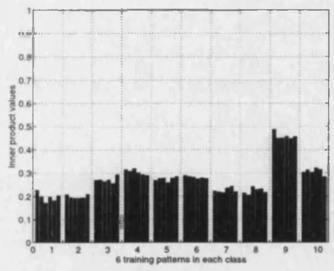


(k) Filter. 8 Bip Mean

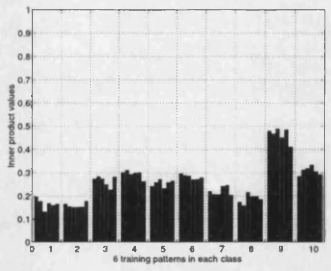


(l) Filt. 8 Bip Rand

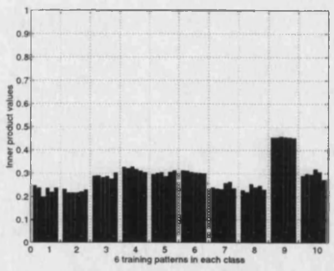
Figure D.9: Seventh and eighth rows of the cross-inner product matrices for monopolar and bipolar patterns and all three initial filter values.



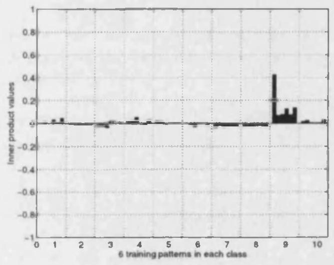
(a) Filt. 9 Mon 1



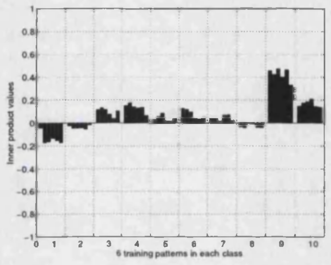
(b) Filt. 9 Mon Mean



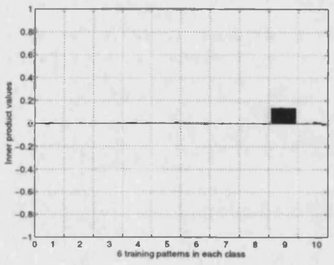
(c) Filt. 9 Mon Rand



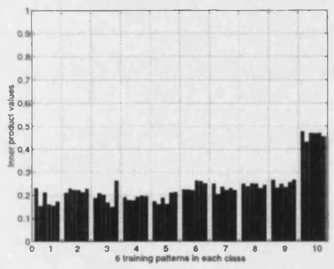
(d) Filt. 9 Bip 1



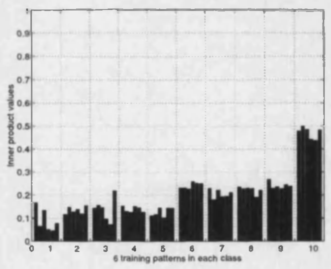
(e) Filt. 9 Bip Mean



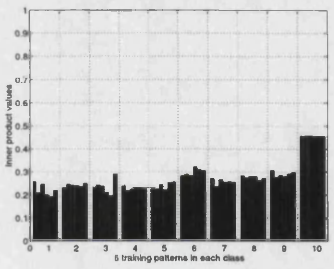
(f) Filt. 9 Bip Rand



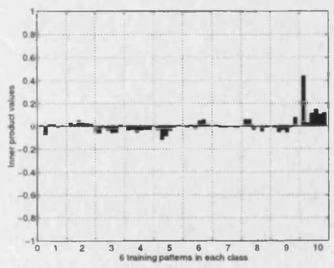
(g) Filt. 10 Mon 1



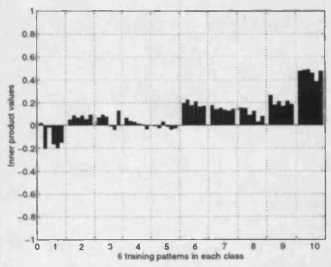
(h) Filt. 10 Mon Mean



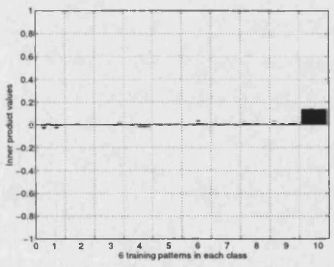
(i) Filt. 10 Mon Rand



(j) Filt. 10 Bip 1



(k) Filter. 10 Bip Mean



(l) Filt. 10 Bip Rand

Figure D.10: Ninth and tenth rows of the cross-inner product matrices for monopolar and bipolar patterns and all three initial filter values.

Bibliography

[1] Neil Collings. *Optical pattern recognition using holographic techniques.* Addison-Wesley, Reading, Massachusetts, 1987.

[2] Gopalan Ravichandran and David Casasent. Minimum noise and correlation energy optical correlation filter. *Applied Optics*, 31(11):1823–1833, April 1992.

[3] Simon Haykin. *Neural networks A Comprehensive Foundation.* Macmillan College Publishing Company, Inc., 866 Third Avenue, New York, New York 10022, 1994.

[4] David R. Selviah, John E. Midwinter, Antony W. Rivers, and K.W. Lung. Correlating matched filter model for analysis and optimisation of neural networks. *IEE Proceedings*, 136, No. 3:143–148, 1989.

[5] A. Vander Lugt. Signal detection by complex spatial filtering. *IEEE Trans. Inf. Theory.*, IT10:139–145, April 1964.

[6] Jack D. Gaskill. *Linear Systems, Fourier Transforms and Optics.* John Willey & Sons, New York, 1978.

[7] Robert J. Schalkoff. *Digital image processing and computer vision.* John Wiley & Sons, New York, 1989.

[8] David R. Selviah and Chi-Ching Chang. Self-pumped phase conjugate resonators and mirrors for use in optical associative memories. *Optics and Lasers in Engineering*, 23:145–166, 1995.

[9] C. Weaver and J. Goodman. A technique for optically convolving two functions. *Applied Optics*, 5(7):1248–1249, July 1966.

[10] J.E. Rau. Detection of differences in real distributions. *J. Opt. Soc. Am.*, 56:1490–1494, 1966.

[11] H. L. Van Trees. *Detection, Estimation, and Modulation Theory: Part I.* Wiley, New York, 1968.

[12] Z. Bahri and B. V. K. Vijaya Kumar. Generalized synthetic discriminant functions. *J.Opt.Soc.Am.*, A 5(4):562–571, April 1988.

[13] D. Casasent. Unified synthetic discriminant function computational formulation. *Applied Optics*, 23:1620–1627, 1984.

[14] B.V.K. Vijaya Kumar. Minimum variance synthetic discriminant functions. *J.Opt.Soc.Am.*, A 3:1579–1584, 1986.

[15] J. Figué and Ph. Réfrégier. Optimality of trade-off filters. *Applied Optics*, 32(11):1933–1935, April 1993.

[16] V. Laude and Ph. Réfrégier. Multicriteria characterization of coding domains with optimal Fourier spatial light modulator filters. *Applied Optics*, 33(20):4465–4471, July 1994.

[17] Ph. Réfrégier. Filter design for optical pattern recognition: multicriteria optimization approach. *Optics Letters*, 15(15):854–856, August 1990.

[18] H. J. Caulfield and W. T. Maloney. Improved discrimination in optical character recognition. *Applied Optics*, 8(11):2354–2356, November 1969.

[19] J. W. Goodman. *Introduction to Fourier Optics.* McGraw-Hill, second edition, 1996.

[20] Alastair D. McAulay. *Optical computer architectures.* John Wiley & Sons, Inc., 1991.

[21] D.M. Budgett, P.C. Tang, J.H. Sharp, R.K. Wang, and B.F. Scott. Parallel pixel processing using programmable gate arrays. *Electronics Letters*, 32(17):1557–1559, 1996.

[22] Dror G. Feitelson. *Optical Computing.* MIT Press, Cambridge, Massachusetts, first edition, 1988.

[23] B.V.K. Vijaya Kumar. Optical pattern recognition. In Bahram Javidi and Joseph L. Horner, editors, *Real-time optical information processing*, chapter 2, pages 39–88. Academic Press, Inc., San Diego, CA., 1994.

[24] B.V.K. Vijaya Kumar and J. Brasher. Relationship between maximizing the signal-to-noise ratio and minimizing the classification error probability for correlation filters. *Optics Letters*, 17:940–942, 1992.

[25] Jehad Khoury, Jonathan S. Kane, George Asimellis, Mark Cronin-Golomb, and Charles Woods. All-optical nonlinear joint fourier transform correlator. *Applied Optics*, 33(35):8216–8225, December 1994.

[26] Jerome Colin, Nicolas Landru, Vincent Laude, Sebastien Breugnot, Henri Rajbenbach, and Jean-Pierre Huignard. High-speed photorefractive joint transform correlator using nonlinear filters. *J. Opt. A: Pure Appl. Opt.*, 1:283–285, 1999.

[27] Demetri Psaltis and Ravindra A. Athale. High accuracy computation with linear analog optical systems: a critical study. *Applied Optics*, 25, No. 18:3071–3077, 1986.

[28] A. Partovi, A.M. Glass, T.H. Chiu, and D.T.H. Liu. High-speed joint-transform optical image correlator using GaAs/AlGaAs semi-insulating multiple quantum wells and diode lasers. *Optics Letters*, 18(11):906–908, June 1993.

[29] Peter S. Erback, Don A. Gregory, and Jeffery B. Hammock. Phase-only joint-transform correlator: analysis and experimental results. *Applied Optics*, 35(17):3091–3096, June 1996.

[30] Chi-Ching Chang, Yuh-Ping Tong, and Hon-Fai Yau. Rotational invariant pattern recognition using photorefractive correlator. *Jpn. J. Appl. Phys.*, 31:L43–L45, January 1992.

[31] Laurent Bigué, Michel Fraces, and Pierre Ambs. Experimental implementation of a joint transform correlator using synthetic discriminant functions. *Optics and Lasers in Engineering*, 23:93–111, 1995.

[32] Michael E. Lhamon and Laurence G. Hassebrook. Translation-invariant optical pattern recognition without correlation. *Optical Engineering*, 35(9):2700–2709, September 1996.

[33] Bahram Javidi. Generalization of the linear matched filter concept to nonlinear matched filters. *Applied Optics*, 29(8):1215–1224, March 1990.

[34] C. Halvorson, T.W. Hagler, D. Moses, Y. Cao, and A.J. Heeger. 160 femtosecond optical correlator. *Chem. Phys. Lett.*, 200:132–133, 1992.

[35] David Casasent. General-purpose optical pattern recognition image processors. *Proceedings of the IEEE*, 82(11):1724–1734, November 1994.

[36] D. Psaltis, M.A. Neifeld, and A. Yamamura. Image correlators using optical memory disks. *Optics Letters*, 14:429–431, 1989.

[37] T. Lu. Optical disk based neural network. *Applied Optics*, 28:4722–4724, 1989.

[38] F.T.S. Yu. Optical disk based joint transform correlator. *Applied Optics*, 30:915–916, 1991.

[39] Mark A. Neifeld and Demetri Psaltis. Programmable image associative memory using an optical disk and a photorefractive crystal. *Applied Optics*, 32:4398–4409, 1993.

[40] S. Tao, Z.H. Song, D.R. Selviah, and J.E. Midwinter. Spatioangular-multiplexing scheme for dense holographic storage. *Applied Optics*, 34, 1995.

[41] F.T.S. Yu, S. Wu, S. Rajan, and D.A. Gregory. Compact joint transform correlator using a thick photorefractive crystal. *Applied Optics*, 31:2416–2418, 1992.

[42] Q. He *et. al.* Shift invariant photorefractive joint transform correlator using fe:linbo$_3$ crystal plates. *Applied Optics*, 32:3113–3115, 1993.

[43] F.T.S. Yu, S. Wu A Mayers, and S. Rajan. Wavelength-multiplexed reflection-type matched spatial filtering using LiNbO$_3$. *Optics Communications*, 81:343–346, 1991.

[44] David T. Carrott, Gary Mallaley, R. Barry Dydyk, and Stuart A. Mills. Third generation Miniature Ruggedized Optical Correlator (MROC^{TM}) module. In David P. Casasent and Tien-Hsin Chao, editors, *Optical Pattern Recognition IX*, volume 3386, pages 38–44. SPIE, 1998.

[45] Kipp A. Bauchert and Steven A. Serati. Data flow architecture for high-speed optical processors. In David P. Casasent and Tien-Hsin Chao, editors, *Optical Pattern Recognition IX*, volume 3386, pages 50–58. SPIE, 1998.

[46] Tien-Hsin Chao, Gerge Reyes, and Youngchul Park. Grayscale optical correlator. In David P. Casasent and Tien-Hsin Chao, editors, *Optical Pattern Recognition IX*, volume 3386, pages 60–64. SPIE, 1998.

[47] A.K. Ghosh, M.B. Lapis, and D. Aossey. Planar integration of joint transform correlators. *Electronic Letters*, 27:871, 1991.

[48] Seok Ho Song, Suntak Park, El-Hang Lee, Pill Soo Kim, and Cha Hwan Oh. Planar optical implementation of multichannel fractional Fourier transforms. *Optics Communications*, 137:219–222, May 1997.

[49] Seok Ho Song, Jong-Sool Jeong, Suntak Park, and El-Hang Lee. Planar optical implementation of fractional correlation. *Optics Communications*, 143:287–293, October 1997.

[50] V. Laude, S. Maze P. Chavel, and Ph. Réfrégier. Amplitude and phase coding measurements of a liquid crystal television. *Optics Communications*, 103:33–38, 1993.

[51] Bjorn Lofving. Measurement of the spatial phase modulation of a ferroelectric liquid-crystal modulator. *Applied Optics*, 35(17):3097–3103, June 1996.

[52] Robin E. Kilpatrick, John H. Gilby, Sally E. Day, and David R. Selviah. Liquid crystal televisions for use as spatial light modulators in a complex optical correlator. In *Optical Pattern Recognition IX*, volume 3386, pages 70–77. SPIE, April 1998.

[53] D. A. Gregory, J. A. Kirsch, and E. C. Tam. Full complex modulation using liguid-crystal televisions. *Applied Optics*, 31:163–165, 1992.

[54] R. D. Juday. Correlation with a spatial light modulator having phase and amplitude cross coupling. *Applied Optics*, 28:4865–4869, 1989.

[55] Mark C. Gardner, Robin E. Kilpatrick, Sally E. Day, Robert E. Renton, and David R. Selviah. Experimental verification of a computer model for optimizing a liquid crystal display for spatial phase modulation. *J. Opt. A: Pure Appl. Opt.*, 1:299–303, 1999.

[56] B.V.K. Vijaya Kumar and L. Hassebrook. Performance measures for correlation filters. *Applied Optics*, 29(20):2997–3006, July 1990.

[57] Joseph L. Horner. Light utilization in optical correlators. *Applied Optics*, 21(24):4511–4514, December 1982.

[58] Philippe Réfrégier, Vincent Laude, and Bahram Javidi. Nonlinear joint-transform correlation: an optimal solution for adaptive image discrimination and input noise robustness. *Optics Letters*, 19(6):405–407, March 1994.

[59] Philippe Réfrégier. Bayesian theory for target location in noise with unknown spectral density. *J. Opt. Soc. Am A*, 16(2):276–283, February 1999.

[60] Paul C. Miller. Comparison of automatic target recognition system performance with full- and reduced-resolution correlators. *Applied Optics*, 38(23):5014–5018, August 1999.

[61] Henrik Sjoberg, Francios Goudail, and Philippe Réfrégier. Comparison of the maximum likelihood ratio test algorithm and linear filters for target location in binary images. *Optics Communications*, 163:252–258, May 1999.

[62] Zu-Han Gu and Sing H. Lee. Classification of multiclassed stochastic images buried in additive noise. *J. Opt. Soc. Am. A*, 4(4):712–719, April 1987.

[63] Y. N. Hsu and H. H. Arsenault. Optical pattern recognition using circular harmonic expansion. *Applied Optics*, 21:4016–4019, 1982.

[64] Abhijit Mahalanobis, B. V. K. Vijaya Kumar, and David Casasent. Spatial-temporal correlation filter for in-plane distortion invariance. *Applied Optics*, 25(23):4466–4472, December 1986.

[65] R. Wu and H. Stark. Three-dimensional object recognition from multiple views. *J. Opt. Soc. Am. A*, 3(9):1543–1557, September 1986.

[66] B.V.K. Vijaya Kumar, Zouhir Bahri, and Abhijit Mahalanobis. Constraint phase optimization in minimum variance synthetic discriminant functions. *Applied Optics*, 27(2):409–413, January 1988.

[67] Max B. Reid, Paul W. Ma, John D. Downie, and Ellen Ochoa. Experimental verification of modified synthetic discriminant function filters for rotation invariance. *Applied Optics*, 29(8):1209–1215, March 1990.

[68] A. Vargas, J. Campos, C. Iemmi, S. Ledesma, and M.J. Yzuel. Optical codification for multiclass pattern recognition using a parallel correlator. *Optics Communications*, 162:121–129, April 1999.

[69] B. V. K. Vijaya Kumar. Tutorial survey of composite filter designs for optical correlators. *Applied Optics*, 31(23):4773–4801, August 1992.

[70] K. I. Diamantaras and S. Y. Kung. *Principal component neural networks: theory and applications.* Wiley, New York, 1996.

[71] H. J. Caulfield and Robert Haimes. Generalized matched filtering. *Applied Optics*, 19(2):181–183, January 1980.

[72] Charles F. Hester and David Casasent. Multivariant technique for multiclass pattern recognition. *Applied Optics*, 19(11):1758–1761, June 1980.

[73] H. J. Caulfield. Linear combinations of filters for character recognition: a unified treatment. *Applied Optics*, 19(23):3877–3878, December 1980.

[74] D. Casasent, B. V. K. Vijaya Kumar, and V. Sharma. Synthetic discriminant functions for 3 dimensional object recognition. In *Proc. Soc. Photo-Opt. Instrum. Eng.*, volume 360, 1982.

[75] B. Braunecker, R. Hauck, and A. W. Lohmann. Optical character recognition based on nonredundant correlation measurements. *Applied Optics*, 18(16):2746–2753, August 1979.

[76] B.V.K Vijaya Kumar. Efficient approach to designing linear combination filters. *Applied Optics*, 22(10):1445–1448, May 1983.

[77] S. I. Sudharsanan, A. Mahalanobis, and M. K. Sundareshan. Selection of optimum output correlation values in synthetic discriminant function design. *J. Opt. Soc. Am.*, 7(4):611–616, April 1990.

[78] Robert R. Kallman. Construction of low noise optical correlation filters. *Applied Optics*, 25(7):1032–1033, April 1986.

[79] B. V. K. Vijaya Kumar, David P. Casasent, and Abhijit Mahalanobis. Correlation filters for target detection in a Markov model background clutter. *Applied Optics*, 28(15):3112–3119, August 1989.

[80] Henri H. Arsenault, Yunlong Sheng, and Jean Bulabois. Modified composite filter for pattern recognition in the presence of noise with a non-zero mean. *Optics Communications*, 63(1):15–20, July 1987.

[81] Bahram Javidi, Philippe Réfrégier, and Peter Willett. Optimum receiver design for pattern recognition with nonoverlapping target and scene noise. *Optics Letters*, 18(19):1660–1662, October 1993.

[82] A. Mahalanobis, B. V. K. Vijaya Kumar, and D. Casasent. Minimum average correlation energy filters. *Applied Optics*, 26:3633–3640, 1987.

[83] D. Casasent and G. Ravichandran. Advanced distortion invariant mace filters. *Applied Optics*, 31:1109–1116, 1992.

[84] S. I. Sudharsanan, A. Mahalanobis, and M. K. Sundareshan. Unified framework for the synthesis of synthetic discriminant functions with reduced noise variance and sharp correlation structure. *Optical Enginnering*, 29(9):1021–1028, September 1990.

[85] H. J. Caulfield. Role of the horner efficiency in the optimization of spatial filters for optical pattern recognition. *Applied Optics*, 21(24):4391–4393, December 1982.

[86] Joseph L. Horner and Peter D. Gianino. Phase-only matched filtering. *Applied Optics*, 23(6):812–816, March 1984.

[87] Joseph L. Horner and James R. Leger. Pattern recognition with binary phase-only filters. *Applied Optics*, 24(5):609–611, March 1985.

[88] Joseph L. Horner and Peter D. Gianino. Applying the phase-only filter concept to the synthetic discriminant function correlation filter. *Applied Optics*, 24(6):851–854, March 1985.

[89] David A. Jared and David J. Ennis. Inclusion of filter modulation in synthetic-discriminant-function construction. *Applied Optics*, 28(2):232–239, January 1989.

[90] Robert R. Kallman. Direct construction of phase-only filters. *Applied Optics*, 26(24):5200–5201, December 1987.

[91] Philippe Réfrégier and Jean-Pierre Huignard. Phase selection off synthetic discriminant function filters. *Applied Optics*, 29(32):4772–4778, November 1990.

[92] Ph. Réfrégier. Optimal trade-off filters for noise robustness, sharpness of the correlation peak, and Horner efficiency. *Optics Letters*, 16(11):829–831, June 1991.

[93] R. F. Thompson. *The Brain: An Introduction to Neuroscience.* W. H. Freeman & Company, New York, 1985.

[94] G.M. Shepherd and C. Koch. *The Synaptic Organization of the Brain.* Oxford University Press, New York, 1990.

[95] James L. McClelland, David E. Rumelhart, and the PDP Research Group. *Parallel distributed processing Explorations in the Microstructure of Cognition Volume 1: Foundations.* The MIT Press, Cambridge, Massachusetts, 1986.

[96] Gustavo Deco and Dragan Obradovic. *An Information-Theoretic Approach to Neural Computing.* Springer-Verlag, New York, 1996.

[97] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[98] M. L. Minsky and S. A. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[99] F. Rosenblatt. On the convergence of reinforcement procedures in simple perceptrons. Technical Report VG-1196-G-4, Cornell Aeronautical Laboratory, Buffalo, NY, 1960.

[100] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, Washington, DC, 1962.

[101] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley, New York, 1949.

[102] G.S. Stent. A physiological mechanism for Hebb's postulate of learning. In *Proceedings of the National Academy of Sciences of the U.S.A*, volume 70, pages 997–1001, 1973.

[103] J. P. Changeux and A. Danchin. Selective stabilization of developing synapses as a mechanism for the specification of neural networks. *Nature*, 264:705–712, 1976.

[104] R. S. Sutton and A. G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88:135–170, 1981.

[105] H. H. Arsenault. Rotation invariant composite filters. In P. A. Yeh, editor, *Nonlinear Optics and Applications*, volume 613, pages 239–244. Soc. Photo-Opt. Instrum. Eng., 1986.

[106] D. Casasent and D. Psaltis. Scale invariant optical correlation using mullin transforms. *Optics Communications*, 17:59–63, 1976.

[107] Y. Sheng and H. H. Arsenault. Experiments on pattern recognition using invariant Fourier-Mullin descriptors. *J. Opt. Soc. Amer.*, 3:771–776, 1986.

[108] Jack K. Mui and King-Sun Fu. Automated classification of nucleated blood cells using a binary tree classifier. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-2(5):429–443, September 1980.

[109] Marco Gori and Franco Scarselli. Are multilayer perceptrons adequate for pattern recognition and verification? *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1121–1132, November 1998.

[110] Chuanyi Ji and Sheng Ma. Performance and efficiency: Recent advances in supervised learning. *Proceedings of the IEEE*, 87(9):1519–1536, September 1999.

[111] A.L. Blum and R.L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5:117–127, 1992.

[112] S. Judd. *Neural Network Design and The Complexity of Learning.* MIT Press, 1990.

[113] D. Haussler, M. Kearns, and R. Shapire. Bounds on the sample complexity of bayesian learning using information theory and vc-dimension. In *The Fourth Workshop on Computational Learning Theory*, 1991.

[114] Wesley E. Foor and Mark A. Neifeld. Adaptive optical, radial basis function neural network for handwritten digit recognition. *Applied Optics*, 34, 1995.

[115] Athanasios Papoulis. *Probability, random variables, and stochastic processes.* McGraw-Hill, New York, third edition, 1991.

[116] R. Bracewell. *The Fourier Transform and Its Applications.* McGraw-Hill, New York, 1965.

[117] Alan V. Oppenheim and Ronald W. Schafer. *Discrete - Time Signal Processing.* Prentice-Hall International, Inc., 1980.