ADAPTIVE METHODS OF IMAGE PROCESSING

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op 14 december 2001 om 16.00 uur
door

Dick DE RIDDER

informatica ingenieur
geboren te Rotterdam

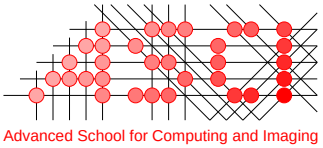
Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. L.J. van Vliet

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. L.J. van Vliet,	Technische Universiteit Delft, promotor
Dr. ir. R.P.W. Duin,	Technische Universiteit Delft, toegevoegd promotor
Dr. P.W. Verbeek,	Technische Universiteit Delft
Prof. dr. J. Kittler,	University of Surrey
Prof. dr. ir. J. Biemond,	Technische Universiteit Delft
Prof. dr. M.H. Overmars,	Universiteit Utrecht
Dr. ir. B.J.A. Kröse,	Universiteit van Amsterdam
Prof. dr. I.T. Young,	Technische Universiteit Delft, reserve lid

Dr. ir. R.P.W. Duin heeft als begeleider in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

This work was partly supported by the Foundation for Computer Science Research in the Netherlands (SION) and the Dutch Organisation for Scientific Research (NWO).



This work was carried out in graduate school ASCI.
ASCI dissertation series number 70.

ISBN 90-75691-06-8

"A PICTURE IS WORTH A THOUSAND WORDS.
IT JUST TAKES UP A LOT MORE BANDWIDTH."

ANONYMOUS

Contents

1

INTRODUCTION

1.1 Background

1.1.1 Artificial intelligence

The idea of building machines or algorithms that use artificial intelligence (i.e. mimic human intelligence) to solve problems has been attracting researchers for a long time. Although mankind has a long history in building machinery that makes certain manual or repetitive tasks lighter, little advance has yet been made in creating appliances to solve problems that normally require human ingenuity. Part of the problem is a lack of understanding of exactly what creates human intelligence: our pattern recognition capabilities; generalisation, the ability to sift through large amounts of data quickly and discard irrelevant information; creativity; a powerful associative mechanism; etc. In spite of wildly optimistic predictions, such as the Heuristic ALgorithmic computer, HAL 9000, in Arthur C. Clarke's "2001: A space odyssey" [58], we are still not anywhere near building a fully functional artificial intelligence [342].

Since the late 1950s, a large amount of research has been performed in the field of artificial intelligence (AI) [81, 281, 295]. AI can roughly be split into two subfields. The first field, *classical AI*, is concerned with constructing and studying algorithms which mimic high-level human capabilities. Examples of developments in this area are frame-based, rule-based and case-based reasoning for expert systems, logical and probabilistic inference, nonmonotonic reasoning etc. [63]. These approaches are based on the assumption that to create artificial intelligence, one has to model human intelligence; in other words, it is *model-based*. Other, newer fields stress the importance of evolution and/or embedding in the real world, such as robotics [35, 64], autonomous agents [19, 228] and artificial life [204, 218].

The second field, the one which this thesis is concerned with, *machine learning*, is more *problem-based*. Given a certain hard problem, how can we best solve it using an al-

gorithm that can learn from examples [37, 244]? The focus here is more on the outcome, a trained algorithm, than on understanding the process itself; it is inductive rather than deductive like classical AI. Most machine learning problems can be formulated as regression or classification (or pattern recognition, concept recognition) problems. Popular tools include those put forward by statistical and structural pattern recognition [85, 92], neural networks [28, 146, 148], fuzzy systems [192], rule induction [288], etc. In general, these systems share the characteristic that their behaviour is based on a number of parameters, which can be adapted by learning, i.e. set to "good" values based on a number of examples.

1.1.2 Neural networks

In the 1940s, psychologists became interested in modelling the human brain. This led to the development of the a model of the neuron as a thresholded summation unit by McCulloch and Pitts [235]. They were able to prove that (possibly large) collections of interconnected neuron models, neural networks, could in principle perform any computation, if the strengths of the interconnections (or weights) were set to proper values. In the 1950s neural networks were picked up by the growing artificial intelligence community. It also attracted the attention of researchers in statistical pattern recognition, an applied field of research born in the 1950s [2, 51, 106], based on earlier work in statistical decision theory.

In 1962, Rosenblatt [301] proposed a method to train a subset of a specific class of networks, called perceptrons. These are networks having neurons grouped in layers, with only connections between neurons in subsequent layers. However, Rosenblatt could only prove convergence for single-layer perceptrons. Although some training algorithms for larger neural networks with hard threshold units were proposed by Nilsson [257], enthusiasm waned after Minsky and Papert in 1969 showed that many seemingly simple problems were in fact nonlinear and that perceptrons were incapable of solving these [243].

Interest in artificial neural networks increased again in the 1980s, after Rumelhart et al. [306] in 1986 proposed a learning algorithm for multi-layer perceptrons, the back-propagation rule (which, as turned out later, was found before them by Parker [271] and Werbos [379]). Feed-forward networks were not the only type of network under research. In the 1970s and 1980s a number of different biologically inspired learning systems were proposed. Among the most influential were the Hopfield network [158, 159], Kohonen's self-organising map [195, 196], the Boltzmann machine [150, 152] and the Neocognitron [116, 117, 118].

The definition of what exactly constitutes a neural network is rather vague. In general it would at least require a system to

- consist of (a large number of) identical, simple processing units;

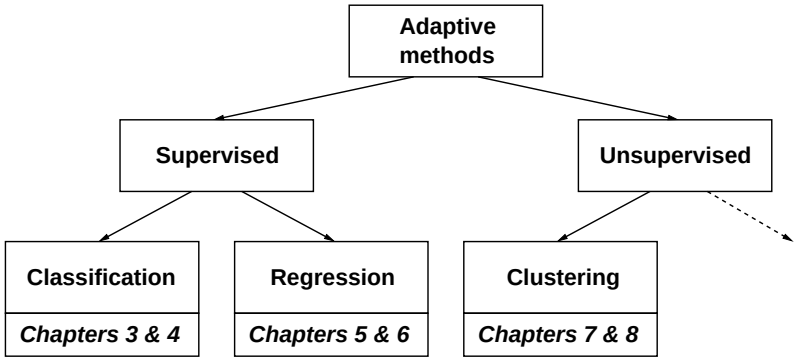


Figure 1.1: Adaptive method types discussed in this thesis.

- have interconnections between these units;
- posess tunable parameters, or weights, which define the system's function and
- lack a supervisor which tunes each individual weight.

However, not all systems that are called neural networks fit this description. Furthermore, there is a wide range of non-neural algorithms which are similar to neural networks (e.g. k-means clustering is quite similar to the self-organising map). Therefore, in this thesis we prefer to speak of *adaptive methods*, meant to be methods controlled by a set of parameters for which optimal values can be found using a learning process.

There are many possible taxonomies of adaptive methods. Here, we concentrate on *learning* and *functionality* rather than on biological plausibility, topology etc. Figure 1.1 shows the main subdivision of interest: *supervised* versus *unsupervised* learning. In unsupervised learning, there is only a data set $\mathcal{L}$ containing samples $\mathbf{x} \in \mathbb{R}^d$, where d is the number of dimensions of the data set. The goal is to construct a description of $\mathcal{L}$, based on the optimisation of some error criterion. An important application of unsupervised learning is *clustering*, in which a number of representative models are fitted to $\mathcal{L}$.

In supervised learning, for each $\mathbf{x} \in \mathcal{L}$ a dependent variable $\mathbf{y} \in \mathbb{R}^m$ has to be supplied as well. The goal of a *regression* method is then to predict this dependent variable based on $\mathbf{x}$. *Classification* can be seen as a special case of regression, in which only a single variable $t \in \mathbb{N}$ is to be predicted, the label of the class to which the sample $\mathbf{x}$ belongs.

In section 2.2, some specific neural networks falling into these categories are discussed.

1.1.3 Digital image processing

Somewhat older than artificial intelligence, digital image processing is concerned with the development of computer algorithms working on digitised images [42, 131, 282,

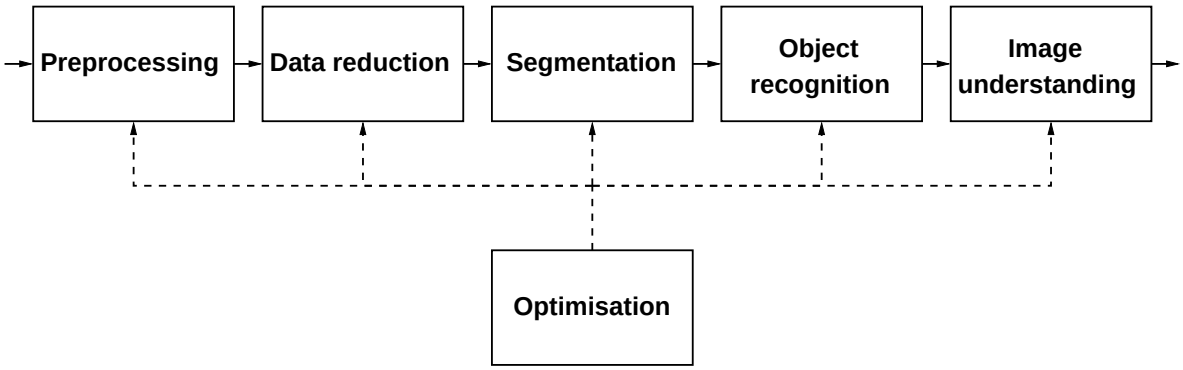


Figure 1.2: The image processing chain.

333]. It is quite a broad field, drawing upon many disciplines such as optics, signal processing, electronics, computer science, pattern recognition, perception science and cognitive science. The first work in image processing dates back to the 1920s, when automated means of image transmission were first used. In the 1950s, computers were starting to be used (see e.g. [370]). The advent of affordable computer power in that period, the questions posed by the space program around the same time, and the increasing availability of imaging and visualisation equipment, led to a large increase in the need for algorithms to process image data. The goal of image processing is usually automatic detection or recognition of image content, in which case one can speak of *machine vision*. However, the goal might also be to enhance images for further processing by humans. Its current applications are numerous, in medicine, industrial inspection, video communication, remote sensing, robot vision etc.

The range of image processing problems is wide, encompassing everything from low-level signal enhancement to high-level image understanding. In general, image processing problems are solved by a chain of tasks. This chain, shown in figure 1.2, outlines the possible processing needed from the initial sensor data to the outcome (e.g. a classification or a scene description). This pipeline consists of the steps of pre-processing, feature extraction, segmentation, object recognition and image understanding. In each step, the input and output data could either be images (pixels), measurements in images (features), decisions made in previous stages of the chain (labels) or even object relation information (graphs). What type of data is appropriate at what stage depends on the application. The image processing chain is discussed in more detail in section 2.3.

With each step in the chain the need for using prior (world) knowledge increases. For simple noise reduction, not much knowledge about the contents of the image itself needs to be known, whereas for image understanding it is imperative to limit the domain of images which can be processed. In this thesis, we deal only with image processing operations which do not presuppose any specific type of image content.

1.2 Motivation

There are many problems in image processing for which good, theoretically justifiable solutions exists, especially for problems for which linear solutions suffice. For example, for low-level operations such as image restoration, methods from signal processing such as the Wiener filter [385] can be shown to be the optimal linear approach. However, these solutions often only work under ideal circumstances; they may be highly computationally intensive (e.g. when large numbers of linear models have to be applied to approximate a nonlinear model); or they may require careful tuning of parameters. Where linear models are no longer sufficient, nonlinear models will have to be used. This is still an area of active research, as each problem will require specific nonlinearities to be introduced. That is, a designer of an algorithm will have to weigh the different criteria and come to a good choice, based partly on experience. Furthermore, many algorithms quickly become intractable when nonlinearities are introduced.

Problems further in the image processing chain, object recognition and image understanding, cannot (yet) be solved using "standard" techniques. For example, the task of recognising any of a number of objects against an arbitrary background calls for the same human capabilities investigated in artificial intelligence: the ability to generalise, associate etc.

All this naturally leads to the idea that adaptive methods might be an ideal set of tools for difficult image processing problems. Possible advantages are:

- instead of designing an algorithm, one could construct an example data set and an error criterion, and train any of a number of learning algorithms to perform the desired input-output mapping;
- for many adaptive methods, such as neural networks, the input can consist of pixels or measurements in images; the output can contain pixels, decisions, labels, etc., as long as these can be coded numerically – no assumptions are made. This means adaptive methods can perform several steps in the image processing chain at once;
- some adaptive models, such as neural networks, can be highly nonlinear; the amount of nonlinearity can be influenced by design, but also depends on the training data [291, 292];
- various methods, such as neural networks, have been shown to be universal classification or regression techniques [119, 160, 161].

This thesis investigates to what extent adaptive methods can be useful in image processing, in particular nonlinear image processing.

1.3 Main questions

The discussion above leads to two main questions:

- *Can image processing operations be learned by adaptive methods?* To what extent can adaptive methods solve problems that are hard to solve using standard techniques? Is nonlinearity a bonus?
- *How can prior knowledge be used*, if available? Can, for example, the fact that neighbouring pixels are highly correlated be used in neural network design or training?
- *What can be learned from adaptive methods trained to solve image processing problems?* If one finds an adaptive method to solve a certain problem, can one learn how the problem should be approached using standard techniques? Can one extract knowledge from the solution?

Especially the last question is intriguing. One of the main drawbacks of many learning systems is their *black-box* character, which seriously impedes their application in systems in which insight in the solution is an important factor, e.g. medical systems. If a developer can learn how to solve a problem by analysing the solution found by a learning algorithm, this solution may be made more explicit.

It is to be expected that for different types of neural networks, the answers to these questions will be different. This thesis is therefore constructed according to the taxonomy shown in figure 1.1:

- in chapters 3 and 4, neural networks for supervised classification are applied to object recognition;
- in chapters 5 and 6, neural networks for supervised regression are investigated as nonlinear image filters;
- in chapters 7 and 8 unsupervised methods are used to describe images for various applications.

Each of these methods is not only applied to real-life problems, but is also studied to answer the questions outlined above. Below, a more detailed overview is given of each chapter.

1.4 Outline of the thesis

Chapter 2 will give an overview of the literature on the application of neural networks to image processing problems. This discussion deliberately excludes other adaptive methods, as otherwise the overview would have become too extensive. For the same reason, the discussion is limited to neural networks operating on image (pixel) data directly.

After measurements have been made in an image, the problem becomes more general in nature, and in principle any adaptive method can be applied on the measurements (features).

In chapter 3, a specific feed-forward neural network, the shared weights neural network, is studied. This network's architecture is tailored for image processing operations, employing convolution-like sliding windows. This makes it especially suitable for classifying image content. Variations of the network are applied to *object recognition* problems such as handwritten digit recognition and automatic target recognition. Their performance is good, but comparable to standard pattern recognition techniques. Chapter 4 then goes on to discuss attempts to understand the internal operation of these networks, specifically their *feature extraction* capabilities. To this end, a number of small networks are trained on toy classification problems and a novel training algorithm is used which decorrelates network weights while optimising the criterion function. One of the conclusions is that due to the excessive degrees of freedom in neural networks it is hard to get an idea of what features they use. Also, the fact that the networks are trained to perform classification makes it hard to untangle the parts of the network responsible for feature extraction and classification.

Chapters 5 and 6 therefore turn to a more low-level problem: image *restoration* (preprocessing) using regression feed-forward neural networks. A specific nonlinear operator for edge-preserving smoothing, the Kuwahara filter, was implemented on a number of network architectures, ranging from a hand-optimised, modular design to a standard, fully interconnected one. Using a novel performance measure for edge-preserving smoothing, it will be shown that using prior knowledge in network design and data set construction is necessary to obtain good performance. General feed-forward networks tend to end up in linear approximations to the filtering operation, which is demonstrated using the weight decorrelating training algorithm introduced in chapter 4.

These findings lead us to return to investigating feature extraction, but leaving feed-forward networks. In chapter 7, subspace mixture models are introduced. These models cluster a data set in a number of subspaces of considerably lower dimensionality than the original space, while using the prior knowledge that image data should be recognised invariant to transformations such as translation, rotation and scaling. Several clustering and subspace-finding methods, principal component analysis (PCA) and independent component analysis (ICA) are discussed. In chapter 8, these methods are applied to image *segmentation*, *object recognition*, image database retrieval and, again, handwritten digit recognition. They are shown to give good results for such a general method.

Chapter 9 ends with some discussion, conclusions and recommendations for further work.

1.5 Sources

This thesis contains work which has been published or submitted before. The sources are, per chapter:

- Chapters 2 and 9 contain parts of a review paper submitted to *Pattern Recognition* [96].
- Chapters 3 and 4 contain parts of the author's M.Sc. Thesis [68], a publication in *Optical Engineering* [78] and some conference publications at ASCI'96, SNN'97, BMVC'97, ASCI'98 and Aerosense'98 [72, 77, 79, 93, 156, 157].
- Chapters 5 and 6 are based on a publication in *Pattern Analysis & Applications* [70] and a number of conference publications at ICONIP'98, SCIA'99 and ASCI'99 [69, 71, 87, 88].
- Chapters 7 and 8 are based on conference publications at BMVC'99, ASCI 2000, S+SSPR 2000, ICPR 2000 and BMVC 2000 [73, 74, 75, 76, 240].

This work was partly supported by the Foundation for Computer Science in the Netherlands (SION) and the Dutch Organization for Scientific Research (NWO), under project number SION 612-31-003.

Parts of chapter 3 are based on preliminary work executed in collaboration with TNO-FEL for the START consortium in the framework of EUCLID RTP-8.2 "Intelligent Sensors". The collaboration and discussions with Signaal, and in particular with the INETI-DOP, in that work is acknowledged.

Some experiments in chapter 4 were inspired by the work of Martin Wachters [367]. Parts of the work in chapters 7 and 8 were performed in co-operation with Eric Körber [198] and Olaf Lemmers [216].

The work presented in chapters 7 and 8 was performed while the author was a visitor at the Centre for Vision, Speech and Signal Processing, University of Surrey, Guilford, UK. This visit was sponsored by the EPSRC, under grant number GR/M90665.

2

ARTIFICIAL NEURAL NETWORKS IN IMAGE PROCESSING

2.1 Introduction

In this chapter, applications of artificial neural networks (henceforth: ANNs) in image processing are reviewed. First, a short introduction to feed-forward ANNs, self-organising maps and Hopfield neural networks (HNNs) will be given. These three types of ANNs are the most widely used in image processing. Second, the taxonomy of image processing techniques proposed in section 1.1.3, the image processing chain, will be discussed in more detail. This is followed by a review of applications of ANNs to each step in this chain. The discussion will be limited to applications in which the input to the ANNs consists of pixel data, as the amount of literature on using ANNs on features extracted from imagery is enormous and not relevant to this thesis.

2.2 Artificial neural networks

The most frequently used ANN architectures in image processing are feed-forward ANNs (also called multi-layer perceptrons, or MLPs) [306], self-organising maps (SOMs) [195] and HNNs [159]. Other architectures such as Boltzmann machines, cellular neural networks (CNNs), random access memory (RAM) networks etc., have been applied little in image processing. These latter architectures will be only briefly discussed in the relevant sections; for a more elaborate introduction, see e.g. [147, 148].

2.2.1 Feed-forward neural networks

A feed-forward ANN [146, 148] consists of interconnected layers of processing units or *neurons*, see figure 2.1. In this figure, the notation of weights and biases follows [148]:

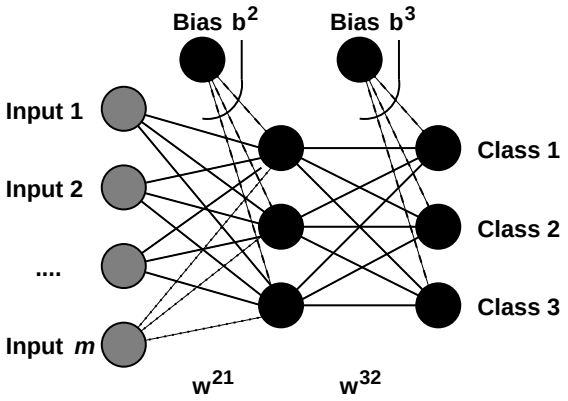


Figure 2.1: A feed-forward ANN for a three-class classification problem. The center layer is called the hidden layer.

weights of connections between layer p and layer q are indicated by $\mathbf{w}^{qp}$; the bias, input and output vectors of layer p are indicated by $\mathbf{b}^p$, $\mathbf{I}^p$ and $\mathbf{O}^p$, respectively. Basically, a feed-forward ANN is a (highly) parameterised, adaptable vector function, which may be trained to perform classification or regression tasks. A classification feed-forward ANN performs the mapping

$$N : \mathbb{R}^d \rightarrow \langle r_{min}, r_{max} \rangle^m, \tag{2.1}$$

with d the dimension of the input (feature) space, m the number of classes to distinguish and $\langle r_{min}, r_{max} \rangle$ the range of each output unit. The following feed-forward ANN with one hidden layer can realise such a mapping:

$$N(\mathbf{x};\ \mathbf{W}, \mathbf{B}) = f(\mathbf{w}^{32^T} f(\mathbf{w}^{21^T}\mathbf{x} - \mathbf{b}^2) - \mathbf{b}^3). \tag{2.2}$$

$\mathbf{W}$ is the weight set, containing the weight matrix connecting the input layer with the hidden layer ($\mathbf{w}^{21}$) and the vector connecting the hidden layer with the output layer ($\mathbf{w}^{32}$); $\mathbf{B}$ ($\mathbf{b}^2$ and $\mathbf{b}^3$) contains the bias terms of the hidden and output nodes, respectively. The function $f(a)$ is the nonlinear activation function with range $\langle r_{min}, r_{max} \rangle$, operating on each element of its input vector. Usually, one uses either the sigmoid function

$$f(a) = \frac{1}{1 + e^{-a}}, \tag{2.3}$$

with the range $\langle r_{min} = 0, r_{max} = 1 \rangle$, the double sigmoid function

$$f(a) = \frac{2}{1 + e^{-a}} - 1, \tag{2.4}$$

or the hyperbolic tangent function $f(a) = \tanh(a)$, both with range $\langle r_{min} = -1, r_{max} = 1 \rangle$.

A second often used feed-forward ANN is the radial basis function ANN, or RBF ANN. This architecture uses Gaussian transfer functions of which the centres $\mathbf{w}$ and widths $\mathbf{b}$ are learned:

$$N_{RBF}(\mathbf{x};\ \mathbf{W},\mathbf{B}) = f(||f(||\mathbf{x}\mathbf{1}^T - \mathbf{w}^{21}||^T,\mathbf{b}^2)\mathbf{1}^T - \mathbf{w}^{32}||^T,\mathbf{b}^3) \tag{2.5}$$

(where $\mathbf{1}$ is a vector of ones and the $||\cdot||$ operation works on the columns of its input matrix), with

$$f(a,b) = \exp(-\frac{a^2}{2b^2}) \tag{2.6}$$

with range $\langle r_{min} = 0, r_{max} = \infty \rangle$.

Classification

To perform classification, an ANN should compute the posterior probabilities, $P(\omega_j|\mathbf{x})$, where ω_j is the label of class j, $j = 1,\ldots,m$. Classification is then performed by assigning an incoming sample $\mathbf{x}$ to that class for which this probability is highest. A feed-forward ANN can be trained in a supervised way to perform classification, when presented with a number of training samples $\mathcal{L} = \{(\mathbf{x},\mathbf{t})\}$, with t_l high (e.g. 0.9) indicating the correct class membership and t_k low (e.g. 0.1), $\forall k \neq l$. The training algorithm, for example back-propagation [306] or conjugate gradient descent [323], tries to minimise the mean squared error (MSE) function:

$$E(\mathbf{W},\mathbf{B}) = \frac{1}{2|\mathcal{L}|}\sum_{(\mathbf{x}^i,\mathbf{t}^i)\in\mathcal{L}}\sum_{k=1}^{c}(N(\mathbf{x}^i;\ \mathbf{W},\mathbf{B})_k - t_k^i)^2, \tag{2.7}$$

by adjusting the weights and bias terms. For more details on training feed-forward ANNs, see e.g. [28, 147, 148, 297].

Richard and Lippmann showed that feed-forward ANNs, when provided with enough nodes in the hidden layer, an infinitely large training [296] and 0-1 training targets, approximate the Bayes posterior probabilities

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})},\ j = 1,\ldots,m, \tag{2.8}$$

with $P(\omega_j)$ the prior probability of class j, $p(\mathbf{x}|\omega_j)$ the class-conditional probability density function of class j and $p(\mathbf{x})$ the probability of observing $\mathbf{x}$.

Regression

Feed-forward ANNs can also be trained to perform nonlinear multivariate regression, where a vector of real numbers should be predicted:

$$R : \mathbb{R}^d \rightarrow \mathbb{R}^m, \tag{2.9}$$

with m the dimensionality of the output vector. The following feed-forward ANN with one hidden layer can realise such a mapping:

$$R(\mathbf{x};\ \mathbf{W},\mathbf{B}) = \mathbf{w}^{32^T} f(\mathbf{w}^{21^T}\mathbf{x} - \mathbf{b}^2) - \mathbf{b}^3. \tag{2.10}$$

The only difference between classification and regression ANNs is that in the latter application of the activation function is omitted in the last layer, allowing the prediction of values in $\mathbb{R}^m$. However, this last layer activation function can be applied when the desired output range is limited. The desired output of a regression ANN is the conditional mean (assuming continuous input $\mathbf{x}$):

$$\mathrm{E}(\mathbf{y}|\mathbf{x}) = \int_{\mathbb{R}^m} \mathbf{y} p(\mathbf{y}|\mathbf{x}) d\mathbf{y}. \tag{2.11}$$

A training set $\mathcal{L}$ containing known pairs of input and output values $(\mathbf{x}, \mathbf{y})$, is used to adjust the weights and bias terms such that the mean squared error between the predicted value and the desired value,

$$E(\mathbf{W},\mathbf{B}) = \frac{1}{2|\mathcal{L}|} \sum_{(\mathbf{x}^i,\mathbf{y}^i)\in\mathcal{L}} \sum_{k=1}^{m} (R(\mathbf{x}^i;\ \mathbf{W},\mathbf{B})_k - y_k^i)^2, \tag{2.12}$$

(or the prediction error) is minimised.

Several authors showed that, under some assumptions, regression feed-forward ANNs are universal approximators. If the number of hidden nodes is allowed to increase towards infinity, they can approximate any continuous function with arbitrary precision [43, 119, 160]. When a feed-forward ANN is trained to approximate a discontinuous function, two hidden layers are sufficient for obtaining an arbitrary precision [80, 334]. However, this does not make feed-forward ANNs perfect classification or regression machines. There are a number of problems:

- there is no theoretically sound way of choosing the optimal ANN architecture or number of parameters. This is called the *bias-variance dilemma* [122]: for a given data set size, the more parameters an ANN has, the better it can approximate the function to be learned; at the same time, the ANN becomes more and more susceptible to *overtraining* [122], i.e. adapting itself completely to the available data and losing generalisation;
- for a given architecture, learning algorithms often end up in a local minimum of the error measure E[1] instead of a global minimum;
- they are *non-parametric*, i.e. they do not specify a model and are less open to explanation. This is sometimes referred to as the *black box problem*. Although some

[1] Although current evidence suggests this is actually one of the features that makes feed-forward ANNs powerful: the limitations the learning algorithm imposes actually manages the bias-variance problem [291, 292].

work has been done in trying to extract rules from trained ANNs [354], in general it is still impossible to specify exactly how an ANN performs its function.

A thorough discussion of these problems will be given in section 2.5.3.

2.2.2 Self-organising maps

Self-organising maps (SOMs, also called topological maps) were introduced by Kohonen in 1982 [194, 195]. A SOM projects the d-dimensional input space onto a discrete m-dimensional lattice ($m < d$)

$$S : \mathbb{R}^d \rightarrow \mathbb{N}^m. \tag{2.13}$$

A map consists of two fully connected layers of nodes: the input layer and the output layer. The number of nodes in the input layer is given by the dimension d of the input vector $\mathbf{x}$. In contrast to feed-forward ANNs, the q nodes in the output layer are arranged in an m-dimensional grid G of size $(o_1 \times \ldots \times o_m)$. In most cases, the grid dimension m is chosen equal to 2. Each node $o_{k,l}$ contains a weight vector $\mathbf{w}^{k,l} \in \mathbb{R}^d$ (see figure 2.2 (a)). A function $S(\mathbf{x};\ \mathbf{W})$ assigns an input sample $\mathbf{x}$ to a node in the output layer. This function usually minimises the squared Euclidean distance between $\mathbf{x}$ and $\mathbf{w} \in \mathbf{W}$, i.e. it picks the node whose weight vector is closest to $\mathbf{x}$. Each node weight vector $\mathbf{w}^{k,l}$ therefore acts as a cluster centre and defines a region containing the set of vectors

$$\left\{\mathbf{x} : ||\mathbf{x} - \mathbf{w}^{k,l}|| < ||\mathbf{x} - \mathbf{w}^{p,q}||, \forall (p,q) \neq (k,l)\right\}. \tag{2.14}$$

SOMs are trained in an unsupervised manner with the goal of projecting similar (according to $S(\mathbf{x};\ \mathbf{W})$) d-dimensional input vectors to neighbouring positions on the m-dimensional map [194, 196]. Training is called *competitive*: at each time step, one winning node gets updated, along with some nodes in its neighbourhood. After training, the input space is subdivided into q regions, corresponding to the nodes in the map.

It can be shown that after training, the density of the weight vectors $\mathbf{w}^{k,l}$ corresponds to the underlying probability density function $p(\mathbf{x})$ of the input vectors [298]. An important application of SOMs in image processing is therefore unsupervised cluster analysis (see figure 1.1). This is achieved by assigning each vector to the cluster with the most similar weight vector as indicated by eqn. 2.14. The structure of the cluster borders depends on which similarity measure is used. Often the Euclidean distance is used; consequently, the cluster borders become $(d-1)$-dimensional hyperplanes. In a two-dimensional feature space, the cluster borders form a Voronoi tessellation (see figure 2.2 (b)).

Another property of SOMs, which has been used in image pre-processing applications, is that a trained map realises a nonlinear topology-preserving mapping of the d-dimensional feature space onto the m-dimensional grid. This means the SOM can be considered to be a generalisation of linear principal component analysis [298].

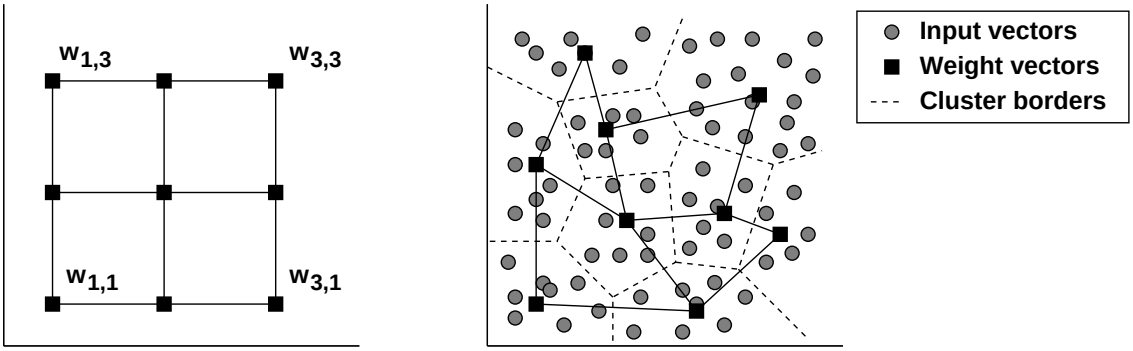


Figure 2.2: The SOM (a) before training and (b) after training, illustrating the partition of a two-dimensional space into cluster regions.

A SOM can also be used as a classifier. First, an unsupervised learning process is performed resulting in q clusters with the centres given by the weight vectors $\{\mathbf{w}^{1,1}, \ldots, \mathbf{w}^{j,k}, \ldots\}$. In a second step, each output node obtains the most frequently occurring class label of the training samples that are assigned to that node [194, 196].

2.2.3 Hopfield neural networks

The Hopfield ANN (HNN), in its basic form presented by Hopfield and Tank [158], consists of a set of q fully interconnected binary nodes, i.e. with an output of either -1 (not firing) or $+1$ (firing). The network maps binary input sets on binary output sets. An individual node fires at time $t+1$ when its weighted input at time t exceeds a certain threshold τ_j:

$$o_i(t+1) = \operatorname{sgn}\left(\left[\sum_{j \neq i} w_{ij} o_j(t)\right] - \tau_j\right), \tag{2.15}$$

where $\operatorname{sgn}(a)$ is $+1$ when its input is positive and -1 when negative. Usually, the threshold τ_j is set to 0. At any time t, the *state* of the HNN can be fully described by the vector containing the activations of all nodes,

$$\mathbf{o}(t) = \left[o_1(t), \ldots, o_q(t)\right]. \tag{2.16}$$

Each state has an associated energy level given by

$$E(t) = -\frac{1}{2}\sum_i \sum_{j \neq i} w_{ij} o_i(t) o_j(t) + \sum_i o_i(t)\tau_i. \tag{2.17}$$

It was proven by Hopfield [158] that the HNN can only change states (by iterating eqn. 2.15) in such a way that the energy $E(t)$ decreases or remains constant. Hopfield's

idea was to set the weights of the HNN such that a set of desired states of the HNN, $\mathbf{X}$, become "energy wells", i.e., states in which the energy is locally minimal. This can be achieved by setting the weights as follows, using the *generalized Hebb rule* [148]:

$$w_{ij} = \begin{cases} \frac{1}{q} \sum_{\mathbf{x} \in \mathbf{X}} x_i x_j & (i \neq j) \\ 0 & (i = j) \end{cases} \tag{2.18}$$

The Hebb rule entails setting a weight connecting two units proportional to the correlation between the output of those neurons; in other words, correlated pattern elements are given large weight.

Note that HNNs are thus not trained in the same way that feed-forward ANNs and SOMs are: the weights are usually set manually. Instead, the power of the HNN lies in running it. In this so-called recall phase, a pattern is presented to the HNN. Then one node is selected at random (i.e., using an asynchronous update rule) and the output of the node is calculated. This procedure is repeated until the output of each node remains constant. Given any starting pattern close enough to one of the energy wells, the HNN will end up in that particular well. This functionality implies emergent properties, viz. content-addressable memory and error correction. Given a partially completed pattern $\mathbf{x}$, or a pattern containing a number of incorrectly set bits, the HNN is capable of restoring the originally learned pattern.

Another application of HNNs, which is quite interesting in an image processing setting [280], is finding the solution to nonlinear optimisation problems. For example, in [159] the NP-complete travelling salesman problem was mapped onto a HNN, and was shown to give near-optimal results. Such a mapping can be found by specifying a function E_p to be minimised and calculating w_{ij} such that $E(t)$ (eqn. 2.17) and E_p are the same. Initialising the network and letting it converge to a stable state then amounts to optimising E_p. However, the application of this approach is limited in the sense that the HNN minimises just one energy function, whereas most problems are more complex in the sense that the minimisation is subject to a number of constraints. Encoding these constraints into the energy function takes away much of the power of the method, by calling for a manual setting of various parameters which again influence the outcome.

One problem with the HNN [146, 345] is the occurrence of spurious states, i.e., stable states that were not foreseen. For example, if one assumes that the thresholds τ are all 0, the HNN cannot distinguish between a state $\mathbf{o}(t)$ and the same state with all outputs reversed, i.e. $-\mathbf{o}(t)$. Also, in its use as content-addressable memory, the HNN can learn stable states which correspond to mixtures of trained patterns. Thirdly, the obtained solutions depend highly on the initial state of the HNN; i.e. for optimisation problems, small differences in initialisation may lead to the HNN finding different (local) minima.

The basic Hopfield model cannot cope with nonlinear mappings as it does not contain hidden nodes. Adaptation of the HNN to cope with nonlinearity led to the development of the continuous and stochastic HNNs and, eventually, the Boltzmann machine. A

discussion of these architectures is beyond the scope of this review; see [148] for more information.

2.3 The image processing chain

Solving an image processing problem typically involves a number of different steps. For the purposes of this review, these steps can be organised into an *image processing chain*, consisting of the following steps:

1. **Pre-processing and filtering**: operations that result in a modified image with the same dimensions as the original image, e.g. contrast enhancement and noise reduction.
2. **Enhancement and feature extraction**: operations that extract significant components from an image, e.g. edges, texture characteristics or landmarks.
3. **Segmentation**: operations that partition an image into regions which are coherent with respect to some criterion. One example is the segregation of different textures.
4. **Object recognition**: determining the position and, possibly, also the orientation and scale of specific objects in an image, and classifying these objects.
5. **Image understanding**: obtaining high level (semantic) knowledge of what an image shows.
6. **Optimisation**: minimisation of a criterion function which may be used for, e.g., graph matching, segmentation or object delineation.

Optimisation techniques are not seen as a separate step in the image processing chain, but as a set of auxiliary techniques, which support all tasks.

Note that not for every problem the entire chain has to be followed; for example, many problems have segmentation or object detection as a final result. Also, some of the steps in the image processing chain have a more coherent definition than others. Preprocessing, step 1, can be almost any operation that somehow transforms the digital signal whereas techniques for object recognition, step 4, result in one or more locations of detected objects. Also, in some ANN approaches multiple steps of the chain are integrated; for example, many ANNs used for object recognition have an integrated (ANN) feature extraction stage.

Image compression, a task for which many ANN approaches have been proposed, is not considered here as its goals are quite different from the high-level steps in the image processing chain proposed here. For an overview of ANNs applied to image compression, see the reviews in [90, 96].

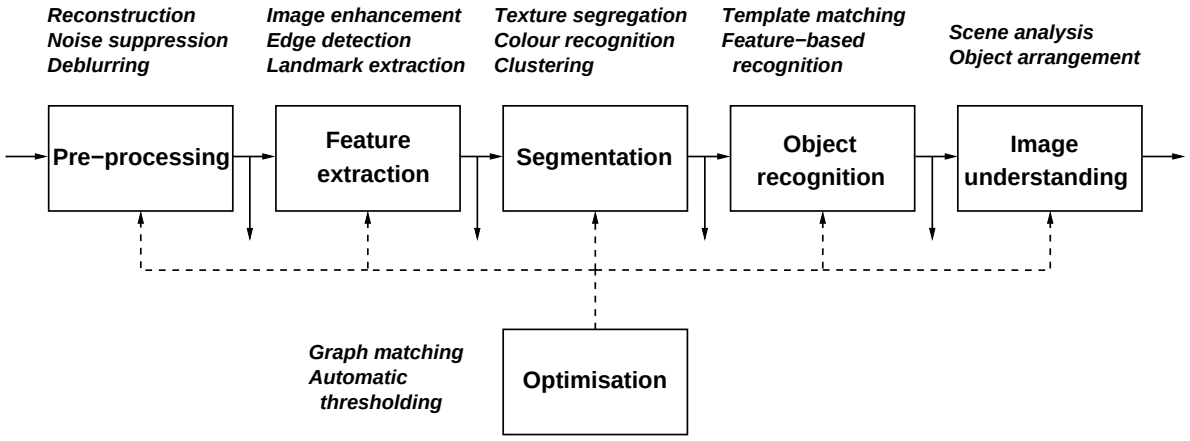


Figure 2.3: The image processing chain. Optimisation is used as an auxiliary technique.

Besides the actual task performed by an algorithm, its processing capabilities are partly determined by the abstraction level of the input data. As said before, here only applications of ANNs on raw pixel data are discussed. However, especially in the more high level steps in the image processing chain, ANNs have also been widely applied on extracted features, object characterisations etc. For a review of these applications, see [96].

2.4 Artificial neural networks in image processing

2.4.1 Pre-processing and filtering

The first step in the image processing chain consists of pre-processing images. Loosely defined, by pre-processing any operation is meant of which the input consists of sensor data, and of which the output is a full image. Preprocessing operations generally fall into one of two categories:

- image *reconstruction*: to reconstruct an image from a number of indirect sensor measurements;
- image *restoration*: to remove any aberrations introduced by the sensor (including noise);

Applications of ANNs in these categories will be discussed below.

Reconstruction

Image reconstruction problems often require quite complex computations and a different approach is needed for each application. In [1], an ADALINE network is trained to perform an EIT (electrical impedance tomography) reconstruction, i.e., a reconstruction of a 2D image based on 1D measurements on a the circumference of the image. Srinivasan et al. [339] trained a modified HNN to perform the inverse Radon transform (used for, e.g., reconstruction of CT (Computerised Tomograpy) images). The HNN contained "summation" layers to avoid an interconnection between all units. Wang and Wahl proposed a variation on the HNN for CT [375] as well. Meyer and Heindl [242] used regression feed-forward ANNs to reconstruct images from electron holograms.

Restoration

The majority of applications of ANNs in pre-processing can be found in image restoration [18, 54, 55, 105, 135, 139, 143, 210, 234, 267, 279, 287, 337, 347, 389, 392, 394]. In general, one wants to restore an image that is distorted by the measurement system, which might introduce noise, motion blur, (out-of-focus) blur etc. The restored image should resemble the original scene as good as possible. Restoration can employ all information about the distortions introduced by the system as, e.g., the point spread function. The restoration problem is ill-posed because contrasting criteria need to be fulfilled: resolution versus smoothness.

The ANN applications reviewed had various designs ranging from relatively straightforward to highly complex, modular approaches. In the most basic restoration approach, noise is removed from an image by simple filtering. Greenhil and Davies [135] used a regression feed-forward ANN with a 5×5 pixel window as input and one output node to suppress noise, by applying the ANN scan-wise to the entire image. Spreeuwers [337] used a similar ANN.

A number of more complex systems have been proposed as well. Chua and Yang [54, 55] were the first to use cellular neural networks (CNNs, [258]) for image processing. A CNN is a system in which nodes are locally connected. Each node contains a feedback template and a control template, which to a large extent determine the functionality of the CNN. For noise suppression, the templates implement an averaging function; for edge detection, a Laplacian operator. The system operates locally, but multiple iterations allow it to distribute global information throughout the nodes. Although quite fast in application, a disadvantage is that the parameters influencing CNN behaviour (the feedback and control templates) have to be set by hand. Others have proposed methods for training CNNs, e.g., using gradient descent (binary images, Schuler et al. [315]) or using genetic algorithms (grey-value images, Zamparelli [389]). CNNs were also applied for restoration of colour images by Lee and Degyvez [210].

Other examples of ANNs applied specifically to noise suppression are the neurochips described in [234], which perform smoothing; fuzzy ANNs [307, 308]; and generalised adaptive neural filters (GANFs) [143, 392]. GANFs consist of a set of neural operators, based on stack filters [8], operating on binary decompositions of grey-value data. Each neural operator gets its input from a small number of levels in the decomposition of an image window, and thus processes the image data locally in both grey-value and image position.

Traditional methods for more complex restoration problems such as deblurring and diminishing out-of-focus defects, are Maximum A Posteriori estimation (MAP) and regularisation. Applying these techniques entails solving high-dimensional convex optimisation tasks. As discussed in section 2.2.3, HNNs and variants hereof lend themselves for solving such optimisation problems: the objective functions of MAP estimation or the regularisation problem can both be mapped onto the energy function of the ANN [18, 105, 267, 347]. Often these networks had to be adapted to fit the problem.

Several specialised types of ANNs have been applied to image restoration as well. Qian et al. [287] used a hybrid approach consisting of order statistic filters for noise removal and a HNN for deblurring (by optimising a criterion function). The modulation transfer function has to be measured. Guan et al. [139] developed a so-called network-of-networks. Their system consists of loosely coupled modules, where each module is a separate ANN. The error function can be adapted to represent regularisation in a way that is similar to HNNs. Waxman et al. [376] consider the application of a centre-surround shunting feed-forward ANN (proposed by Grossberg) for contrast enhancement and colour night vision. Finally, Phoha and Oldham [279] proposed a layered, competitive ANN to reconstruct a distorted image; however, it can also perform edge detection.

Discussion

There seem to be three types of problems in pre-processing (unrelated to the three possible operation types) to which ANNs can be applied:

- optimisation of an objective function specified by a traditional pre-processing approach;
- approximation of a mathematical transformation used in reconstruction, by regression;
- general regression/classification, usually directly on pixel data (neighbourhood input, pixel output).

To solve the first type of problem, HNNs can be used for the optimisation involved in traditional methods. In most of the applications reviewed, however, mapping the actual problem to the energy function of the HNN turned out to be difficult. Occasionally, the

original problem had to be modified before it could be solved by the HNN. Having managed to map the problem appropriately, the HNN can be a useful tool in image pre-processing, although convergence to a good result is not guaranteed.

For the reconstruction problem, regression (feed-forward) ANNs can be applied. Although some applications of ANNs were indeed successful, it would seem that these applications call for more traditional mathematical techniques, because a guaranteed performance of the reconstruction algorithm is essential.

In several other applications, regression or classification ANNs were trained to perform image restoration directly on pixel data. A remarkable first finding in the literature research was that in a large number of applications, *non-adaptive* ANNs (such as CNNs) were used. Secondly, where ANNs were adaptive, their architectures usually differed much from those of the standard ANNs: prior knowledge about the problem was used to design them (e.g. in the templates used in CNNs). This indicates that the fast, parallel operation of ANNs, and the ease with which they can be embedded in hardware, can be important factors in choosing for a neural implementation of a certain pre-processing operation. However, their ability to learn from data is apparently of less importance. While it is relatively easy to construct a linear filter with a certain desired behaviour, e.g. by specifying its frequency profile, it is much harder to obtain a representative data set to learn the optimal function by using a high-dimensional regression method (see also chapters 5 and 6 of this thesis). This holds especially when the desired ANN behaviour is only critical for a small subset of all possible input samples (e.g., in edge detection). Moreover, it is not at all trivial to choose a suitable error measure for supervised training, as the mean squared error (MSE) often used in ANN training might give unwanted results in an image processing setting [337].

An important caveat is that the ANN parameters are likely to become tuned to one type of image (e.g., a specific sensor, scene setting, scale, etc.), which limits the applicability of the trained ANN. When the underlying conditional probability distribution, e.g., $p(\mathbf{y}|\mathbf{x})$ in eqn. 2.11, changes, the ANN – like all statistical models – needs to be retrained.

2.4.2 Enhancement and feature extraction

After pre-processing, the next step in the image processing chain is extraction of information relevant to later stages (e.g. subsequent segmentation or object recognition). In its most generic form, this step can extract low-level information such as edges, texture characteristics etc. This kind of extraction is also called image *enhancement*, as certain general (perceptual) features are enhanced. As enhancement algorithms operate without a specific application in mind, the goal of using ANNs is to outperform traditional methods, either in accuracy or computational speed.

Other approaches extract more application-specific geometric or perceptual features, such as corners, junctions and object boundaries. For particular applications, even more

high-level features may have to be extracted, e.g. eyes and lips for face recognition. A goal of this type of feature extraction is to lower the computational cost. However, it also serves as a means for controlling the so-called *curse of dimensionality* [2] when training ANNs for classification or regression problems. Feature extraction is usually tightly coupled with classification or regression; what variables are informative depends on the application, e.g. object recognition. Generally, one wants to extract those features that preserve the class separability as well as possible [114], i.e., minimising the within-class variability while maximising the between-class variability [85]. Some ANN approaches consist of two stages, possibly coupled, in which features are extracted by the first ANN and object recognition is performed by the second ANN. Such approaches are discussed both here and in section 2.4.4.

Enhancement

Among the applications where ANNs have been developed for image enhancement [45, 225, 246, 278, 324, 337, 338, 358, 376], one would expect most applications to be based on regression ANNs [278, 286, 338, 376]. However, several approaches rely on a classifier, typically resulting in a binary output image [45, 225, 246, 324, 337].

The most well-known enhancement problem is the detection of edges. A straightforward application of regression feed-forward ANNs, trained to behave like various edge detectors, was reported by Pugmire et al. [286] and Spreeuwers [337]. Chandrasekaran et al. [45] used a novel feed-forward architecture to classify an input window as containing an edge or not. The weights of the ANN were set manually instead of being obtained from training. A number of more complicated, modular approaches were also proposed [225, 278, 338]. Of course, if edge detection can be formulated as an optimisation problem, HNNs can be applied to image enhancement as well. This is exactly what Tsai et al. do for very precise enhancement of endocardiac borders [358].

Some approaches utilise other types of ANNs. Shih et al. [324] applied an ART network for binary image enhancement. Moh and Shih [246] describe a general approach for implementation of morphological image operations by a modified feed-forward ANN using *shunting* mechanisms, i.e. neurons acting as switches.

Feature extraction

Among the ANNs that have been trained to perform feature extraction [65, 66, 112, 113, 129, 191, 202, 203, 273, 310, 312, 325, 346, 371, 372, 380], feed-forward ANNs [112,

[2] A property of the classification or regression problem one wants to solve. Briefly, the problem is that a higher dimensionality of the feature space leads to an increased number of parameters that need to be estimated. The risk of overfitting the model will increase with the model complexity (dimensionality), which will often lead to *peaking* [369], i.e. performance decreasing when the number of dimensions grows beyond a certain point.

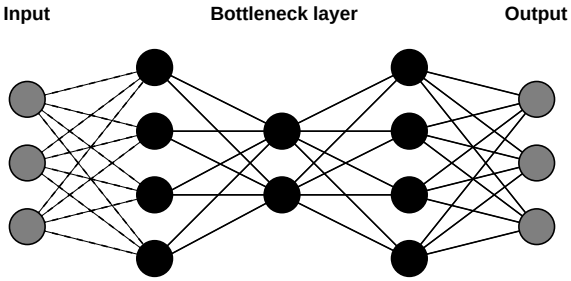


Figure 2.4: A nonlinear auto-associator ANN, mapping the 3D input to 2D in the bottleneck layer. Bias weights have not been drawn.

113, 325, 371, 372, 380], auto-associator ANNs (and variations) [273, 310, 312] and SOMs [129, 132, 191, 202, 203, 359, 393] have been used in most of the reviewed applications. The HNN [346] and the perceptron [65, 66] have also been used, to perform feature matching.

Auto-associator ANNs, a special type of feed-forward ANNs, are very applicable to feature extraction. Usually, the input signal is obtained from a convolution window. The ANNs contain at least one hidden layer with less units than the input and output layers – see figure 2.4 for an example. They are then trained to recreate the input data. The bottleneck architecture forces the ANNs to project the original d-dimensional data onto a lower, m-dimensional (possibly nonlinear) manifold from which the original data should be predicted as well as possible. Features can be extracted by supplying the ANNs with the original data and reading of the m-dimensional data. If the ANN has one hidden layer, the ANN projection can be shown to be identical to PCA [12]; when more hidden layers are used, nonlinear mappings can be learned [83, 200, 261, 360].

It is also possible to use a mixture of linear subspaces to approximate a nonlinear manifold (see e.g. [151, 356]). This approach comes close to the SOM feature extraction method, which is to is to first cluster data in the high-dimensional space, and use the cluster centres as prototype representations for the entire cluster. Chapters 7 and 8 further discuss the mixture-of-subspaces approach.

ANN feature extraction was performed for:

- subsequent segmentation of food images [273] and MR (Magnetic Resonance) images [129], automatic target recognition (ATR) [371, 372], ATR in remote sensing images (taking orientation into account) [191], recognition of characters [325, 346], medical image analysis [132, 359, 393], tracking facial movement [310] and stereo matching [65, 66];
- finding the orientation of objects [112, 247];
- finding control points of deformable models [380];

- clustering local content of an image before it was encoded [312].

In some cases, SOMs were used to cluster low-level features found by Gabor filters, e.g. in face recognition [202] and wood defect detection [203].

In most applications, extracted features were used for subsequent segmentation, image matching or object recognition. Since rotation of (anisotropic) textures or objects typically cause the largest intra-class variation, some feature extraction approaches were designed to cope explicitly with changes in orientation of objects.

Discussion

In the approaches to image enhancement, it is noteworthy that prior knowledge was again often used to restrict the ANNs. Some ANNs had manually set weights; some used the knowledge in their architecture (e.g., by using shunting mechanisms to force a feed-forward ANN to make binary decisions). The same conclusion seems to hold as for pre-processing (see 2.4.1): that operating speed and the possibility of hardware implementation are more important than the ANN ability to learn from data.

For feature extraction, a number of clustering and dimensionality reduction ANNs were discussed. It is important to make a distinction between application of supervised and unsupervised ANNs. For a supervised auto-associator ANN, the information loss implied by the data reduction can be measured directly on the predicted output variables. For unsupervised feature extraction using the SOM, the information loss is more difficult to measure as the desired output is unknown.

Both supervised and unsupervised ANN feature extraction methods have advantages compared to traditional feature extraction techniques such as PCA. Feed-forward ANNs with several hidden layers can be trained to perform nonlinear feature extraction. Moreover, a special purpose feature extractor can be trained when the desired output is known. The major disadvantage of nonlinear feature extraction ANNs is the lack of a formal, statistical basis. This issue is a subject for further research.

2.4.3 Segmentation

Segmentation is partitioning an image into parts that are coherent according to some criterion. When considered as a classification task, the purpose of segmentation is to assign labels to individual pixels or voxels. Some neural-based approaches perform segmentation directly on the pixel data, obtained either from a window (occasionally from more bands, e.g. as present in remote sensing and Magnetic Resonance (MR) images) that is slid across the image, or the information is provided to a neural classifier in the form of local features. These features typically characterise texture or geometry in a local neighbourhood around the pixel that is to be labelled.

Here only segmentation based on pixel or voxel data is considered. To this end, many ANN approaches have been presented, based on feed-forward ANNs [141, 183, 206, 226, 294, 317, 326, 327, 384], SOMs [3, 136, 193, 206, 255, 294, 368] and other self-organising ANNs [124, 127, 128, 232], HNNs [46, 125, 126, 230, 303, 373], probabilistic ANNs [206, 374], radial basis function ANNs [206], CNNs [14, 365] and RAM networks [314]. Also, biologically inspired ANN approaches have been proposed: the perception model developed by Grossberg [137, 138], which is able to segment images from surfaces and their shading, and the brain-like networks proposed by Opara and Worgotter [265]. For a review of ANN approaches to image segmentation and a comparison to traditional methods, see [268].

Hierarchical segmentation approaches have been designed to combine ANNs on different abstraction levels [317, 373]. The guiding principles behind hierarchical approaches are specialisation and bottom-up processing: one or more ANNs are dedicated to low level feature extraction/segmentation, and the results from the low-level processing ANNs are combined on a higher abstraction level where another (neural) classifier performs the final image segmentation. Reddick et al. developed a pixel-based two stage approach where a SOM is trained to segment multispectral MR-images [129]. The segments are subsequently classified into white matter, grey matter, etc., by a feed-forward ANN. Non-hierarchical, modular approaches have also been developed in which each ANN is dedicated to a special task [3, 226, 361] or where each ANN is specialised to recognise pixels belonging to a particular class [317].

In general, pixel-based ANNs have been trained to classify image content based on:

- texture [3, 129, 136, 183, 206, 226, 266, 270, 314, 326, 327, 361, 383];
- colour [125, 126, 128, 384];
- a combination of texture and local shape [50, 141, 221, 317, 374].

ANNs have also been used as pre- and post-processing steps for segmentation algorithms, e.g. for:

- clustering of pixels [255, 366, 368];
- deciding whether a pixel occurs inside or outside a segment [373];
- de-fuzzifying segmented images [124, 127];
- identification of surfaces [137, 138];
- motion segmentation [232];
- delineation of contours [49, 365];
- connecting edge pixels [14, 303].

Discussion

It is clear that ANNs are applicable to many different kinds of pixel-based segmentation tasks. In most applications, ANNs were trained as supervised classifiers to perform the desired segmentation. In these cases, all information (within a window) is provided directly to the classifier. The size of the window should be roughly comparable to that of the texture elements (*texels*) for structured textures (textures consisting of repeatedly occuring fixed elements), or large enough to obtain reliable discriminative texture statistics in case of unstructured textures. The perfect (minimal error-rate) classifier should then be able to produce the best segmentation result.

A central problem in image segmentation is that most pattern recognition techniques, among which ANNs, do not make use of the fact that neighbouring pixels are highly correlated. This hinders their application to problems requiring invariance to translation, rotation and/or scale. A solution would be to build such spatial information directly into the neural classifier (e.g., using weight sharing [207] or by taking symmetries into account [311]); alternatively, the classifier could be trained explicitly to cope with the variation by including training images in all relevant orientations and scales. Secondly, an open question is how to combine local (neighbourhood) information with context information (for example, on neighbouring areas already segmented) and prior knowledge. These problems will be considered in section 2.5.

One might argue that in view of these problems, feature-based texture segmentation is preferable, as prior knowledge can be used in the feature extraction stage and it will be easier to avoid the curse of dimensionality. Although this is true, in some cases feature-based segmentation is not feasible, since there is no limited set of images on which an algorithm has to work. In such applications, e.g., image database retrieval, prior knowledge on which features should be used is not available and adaptive pixel-based methods can prove useful. Section 8.4 discusses such an approach to image database retrieval.

A problem in performance evaluation of segmentation is how to obtain a ground truth for the (in most cases supervised) segmentation algorithms. In general, the true class membership of the pixels/voxels in the training set is known with varying degrees of confidence. In [97], this problem is addressed by letting an expert demarcate the inner parts of areas with a similar (coherent) texture but leaving the transition areas unclassified. Certainly, intra- and inter-observer variability needs to be assessed thoroughly before confident training and test sets can be compiled. Even when a reliable ground truth is available, objective performance assessment entails more than simply computing error rates on novel test images. Different segmentation algorithms could be compared using some of the proposed spatial quality measures. Generally, these measures express desirable properties such as within-region homogeneity and between-region heterogeneity [91, 163] (for an overview see [391]). However, there is not yet a single measure capable of unequivocally quantifying segmentation quality.

2.4.4 Object recognition

Object recognition consists of locating the positions and possibly orientations and scales of instances of classes of objects in an image (object detection) and classifying them (object classification). Among the ANN approaches developed for pixel-based object recognition several types of ANNs can be distinguished: feed-forward-like ANNs [82, 94, 98, 112, 113, 121, 130, 132, 168, 191, 202, 203, 220, 274, 277, 305, 309, 349, 359, 359, 371, 393, 395], variants using weight sharing [15, 109, 120, 207, 208, 209, 364], recurrent ANNs [86, 396], ART networks [39, 41, 324], bi-directional auto-associative memories [199], mixtures-of-experts [371, 372], (evolutionary) fuzzy ANNs [188], the Neocognitron [116, 118, 254] and variants hereof [16, 236], piecewise-linear neural classifiers [330], higher-order ANNs [335, 336] and HNNs [10, 386]. Besides, interesting hardware ANNs have been built for object recognition: the RAM network [52, 89] and optical implementations [185, 322]. Finally, SOMs are occasionally used for feature extraction from pixel data [132, 191, 202, 203, 359, 393]; the output of the map is then propagated to a (neural) classifier. For an overview of an important application area of ANN object recognition, automatic target recognition (ATR), see [302].

Several novel ANN architectures have been developed specifically to cope with concomitant object variations in position, (in-plane or out-of-plane) rotation and scale (in one case, an approach has been developed that is invariant to changes in illumination [322]). An interesting approach that is invariant to 2D translations, in-plane rotation and scale is the what-and-where filter [39], a combination of a multi-scale oriented filter bank (where) and an invariant matching module (what). Other approaches rely on:

- pre-processing the data to remove the invariances, e.g. removing rotation using a polar mapping [112, 113, 395];
- learning the variations explicitly in training, e.g. by adding mirrored and rotated versions of the original samples to the training set [41, 44, 223, 305] or by synthesing images [94, 98, 274];
- built-in invariance over a limited range, to both translation and rotation [82] or to translation and/or scale (e.g., the Neocognitron [116, 118, 254] and shared weight ANNs [15, 109, 120, 207, 208, 209, 364]).

Rare conditions such as object occlusion or the occurrence of multiple objects within the (sub)image that is processed by the classifier have hardly been considered explicitly. An experimental architecture developed by McQuiod is capable of recognising multiple objects simultaneously within an image [236]. Recognition of object parts (due to occlusion) is also considered in section 2.4.6.

Clearly, when object recognition is performed by teaching a classifier to recognise the whole object from a spatial pattern of pixel intensities, the complexity of the classifier grows quickly with the size of the object and with the number of dimensions (2D v 3D).

Moreover, depending on the contents of the scene (image), context information may be required before the objects of interest can be recognised with confidence. The incorporation of context information may again lead to a large number of extra parameters and thereby a more complex classifier. To cope with this problem, so-called multiresolution approaches have been developed [305, 359, 371, 372, 386, 387]. In these approaches, the ANN obtains as input the intensities from pixels located on different levels of a pyramid [181] but centred at the same location. In this way, the classifier is provided with context information but a combinatorial explosion in the number of parameters is avoided. Still, variations in scale have to be learned explicitly by the classifier through training.

Another disadvantage of ANN pyramid approaches is that they sample the *scale-space*[3] coarsely as the resolution is reduced with a factor two at each level in the pyramid. A special type of ANN that incorporates the scale information directly in a pyramidal form is the so-called higher-order ANN [335, 336]. This ANN builds up an internal scale-space-like representation by what is called coarse coding. However, higher-order ANNs need to learn variations in scale explicitly too. They should be used with caution because the coarse coding scheme may lead to aliasing as the high-resolution image is not blurred [181] before computing the coarser image on the next level.

Recurrent ANNs, i.e. ANNs with feed-back loops [147, 148], can be used to develop special approaches to object recognition [86, 396]. The added value of using a recurrent ANN lies in its memory: the current state contains information about the past, which may constitute valuable context information. The recurrent ANN developed by Ziemke [396] performs a convolution with an image in order to detect oil spills. According to the author, the recurrency principle ensures a more robust object recognition.

Several of the approaches for object detection and classification operate on binary images [16, 39, 52, 82, 89, 300]. Whereas binarising images simplifies the recognition problem considerably and facilitates inspection of the weights of the ANNs (e.g., by rule extraction), reducing the grey-level spectrum leads to a large increase in quantification noise [282]. This inevitably decreases the recognition performance of the ANN.

Discussion

The advantage of pixel-based neural approaches to object recognition is that all (relevant) information is provided as input to the classifier, at least when a sufficiently large window size can be chosen. Context information and prior knowledge can also be included using, e.g., shared weights or an image pyramid. Shared weight ANNs will be discussed in depth in chapters 3 and 4. A major disadvantage of such approaches is that object variations in rotation and scale have to be learned explicitly by the classifier

[3] A technique which entails constructing a stack of images in which details are gradually smoothed out, revealing larger structures [222].

(translation can usually be coped with by scan-wise application of the ANN). This again calls for a very large, complete training set and a classifier that can generalise well, although some model-based approaches have been presented that can generate such a complete training set [94, 274]. It is an issue for future research how to cope with occlusion or the occurrence of multiple objects within one (sub)image; in this situation many approaches may fail. Moreover, in volume data objects are three-dimensional. Very little research has been performed in application of ANNs for recognition of objects in 3D images.

2.4.5 Image understanding

Image understanding is the final step in the image processing chain, in which the goal is to *interpret* the image content. Therefore, it couples techniques from segmentation or object recognition with the use of prior knowledge of the expected image content (such as image semantics). As a consequence, there are few applications of ANNs on pixel data. In two such applications, ANNs were trained to classify ships, which were recognised from pixel data by an advanced modular approach consisting of a SOM and a feed-forward ANN [272] and to analyse satellite images [7].

A major problem when applying ANNs for high level image understanding is their black-box character. Although there are proposals for explanation facilities [95] and rule extraction [354], it is usually hard to explain why a particular image interpretation is the most likely one, Another problem in image understanding relates to the amount of input data. When, e.g., seldomly occurring images are provided as input to a neural classifier, a large number of images are required to establish statistically representative training and test sets. By contrast, for segmentation or object recognition based on low level information, much less images are needed. In conclusion, an ANN is not an optimal tool for image understanding problems.

2.4.6 Optimisation

Some image processing (sub)tasks such as stereo matching can best be formulated as optimisation problems, which may be solved by ANNs. In most of the papers reviewed, a HNN was used [56, 253, 303, 321, 344, 346, 353, 388]. In some applications, the HNN obtained pixel-based input [303, 321, 346, 388], in other applications the input consisted of local features [56, 253, 344, 353].

HNNs have been applied to optimisation problems in reconstruction and restoration (not discussed here; see section 2.4.1), segmentation, matching and recognition. The following problems were mapped onto the HNN's error function:

- segmentation of an image with an intensity gradient, by interpolating a surface, where the error function contained an interpolation term and smoothness constraints [303, 321];
- relaxation labeling for thresholding [388], in which a matrix of "compatibilities" between neighbouring pixels was used to define the error function;
- stereo matching, by establishing correspondence between features (landmarks) [253]; each node represents a possible feature match, the weights are set according to a compatibility between features, based on disparity;
- 2D and 3D object recognition as a graph matching problem, in which detected primitives and their attributes were used as in the stereo matching approach above [344, 346];
- approximation of curve of edge points by a polygon [56], where the error function is the arc-to-chord deviation between the curve and the polygon;
- establishing feature trajectories in motion images [353], again using compatibility between features, based on motion model fit, trajectory continuity and 2D geometry.

Mainly, HNNs have been applied for segmentation and recognition tasks that are too difficult to realise with conventional neural classifiers because the solutions entail partial graph matching or recognition of 3D objects. Matching and recognition are both solved by letting the ANN converge to a stable state while minimising an energy function. It was also shown that iterating the HNN can be interpreted as a form of probabilistic relaxation [142].

A disadvantage of HNNs is that training and use are both of high computational complexity. However, other more traditional algorithms for nonlinear programming, in general, also produce high computational loads [149]. In this context, it should be kept in mind that some (constrained) nonlinear programming problems can be solved optimally by traditional algorithmic approaches. For this subclass of optimisation problems, a conventional algorithmic solution should be preferred. The HNN is an interesting approach for problems that lie beyond this subclass of solvable optimisation problems.

2.5 Discussion

This literature review has shown that one of the major advantages of ANNs is that they are applicable to a wide variety of problems. There are, however, still caveats and fundamental problems that require attention. Some of these issues are general, in the sense that they are not resolved by other, competing techniques from the pattern recognition field which may be applied to the same image processing problems. Other problems are

caused by using statistical, data-oriented technique to solve image processing problems. Finally, some problems are fundamental to the way ANNs approach pattern recognition problems. General issues in pattern recognition, the problems related to the application of pattern recognition techniques on image data and the specific ANN related issues are discussed separately.

2.5.1 Issues in pattern recognition

When trying to solve a pattern recognition problem, one may be faced with several problems that are fundamental in applied statistical pattern recognition: avoiding the curse of dimensionality, selecting the most discriminative features and achieving a good *transferability*.

The first problem, the curse of dimensionality, occurs when too many input variables are provided to a classifier or regression function. The risk of ending up with a classifier or regressor that generalises poorly increases with the number of dimensions of the input space. The problem is caused by the inability of existing classifiers to cope adequately with a large number of (possibly irrelevant) parameters, a deficiency that makes feature extraction and/or feature selection necessary steps in classifier development. Feature extraction has been discussed in detail in section 2. Feature selection is, because of its dependence on a trained classifier, an ill-posed problem. Which features give the best performance, depends on the particular classifier that is used [99, 184, 285]. Besides offering a way to control the curse of dimensionality, feature selection also provide insight in the properties of a classifier and the underlying classification problem [99]. Also, one should keep in mind that the conditional distributions of the input data to a classifier or regression function – its features – largely determine the performance that can be obtained, rather than the choice of classifier.

A problem that is especially important in some applications, such as medical image processing or industrial inspection, is how to ensure the *transferability* of a classifier. When trained to classify samples obtained from one setting with a particular prior (class) distribution, a classifier will have a poorer and possibly unacceptably low performance when transferred to another setting where the prior class distribution is different. A question related to this is how to account for changing underlying input distributions[4], $p(\mathbf{x}|\omega_j)$ or $p(\mathbf{y}|\mathbf{x})$ (see section 2.2.1). In general, the parameters of the classifier or regression function need to be re-estimated from a data set that is representative for the novel (prior) distribution. This problem is intrinsic to all statistical models as they are based on inductive inference. Models that have not been re-trained should recognise samples falling outside the distribution they were trained on and discard them, thereby avoid-

[4]Note that for a classifier that has been trained, e.g., to recognise objects appearing at a certain scale directly from pixel data, classification of similar objects at a different scale is equivalent to classifying samples from a novel distribution $p(\mathbf{x}|\omega_j)$.

ing the assignment of "wild-guess" outputs (see e.g. [350, 352]). This is called *outlier detection*. Finally, the question of how to incorporate costs of different misclassifications or computational costs of various algorithms is not yet fully answered.

2.5.2 Obstacles for pattern recognition in image processing

Besides fundamental problems within the field of pattern recognition, other problems arise because statistical techniques are used that consider each pixel as an independent variable. Another problem is how one should incorporate prior knowledge into pattern recognition techniques. Also, the evaluation of image processing approaches is not always straightforward.

A problem in the application of pattern recognition techniques to images is how to incorporate context information and prior knowledge about the expected image content. Prior knowledge could be knowledge about the typical shape of objects one wants to detect, knowledge of the spatial arrangement of textures or objects or of a good approximate solution to an optimisation problem. According to Perlovsky [276], the key to restraining the highly flexible learning algorithms ANNs are, lies in the very combination with prior knowledge. However, most pattern recognition methods do not even use the prior information that neighbouring pixel values are highly correlated. The latter problem can be circumvented by extracting features from images first, by using distance or error measures on pixel data which do take spatial coherency into account (e.g. [151, 329]), or by designing an ANN with spatial coherency (e.g. [118, 207]) or contextual relations beween objects (e.g. [53]) in mind. On a higher level, some methods, such as the pyramid and scale-space approaches reviewed in section 2.4.4, can provide a segmentation algorithm with context information that may improve its performance.

There is a clear need for thorough validation of the developed image processing algorithms [67, 144]. In the literature reviewed, tests on a large set of independent images had only occasionally been performed. Validation and comparison between different algorithms are only possible when a reliable ground truth exists and meaningful (objective) quality measures are available. For some processing tasks like object recognition, a ground truth is in most cases easy to obtain. In other applications, different (human) observers may not fully agree about the ground truth. Even with a reliable ground truth available, it is clear that performance assessment entails much more than simply computing error rates on novel test images: it is equally important how methods deal with, e.g., noisy measurements, changes in lighting, occlusion etc.

Finally, in image processing, classification and regression problems quickly involve a very large number of input dimensions, especially when the algorithms are applied directly on pixel data. This is problematic, due to the curse of dimensionality discussed before. However, the most interesting future applications promise to deliver even more input. Whereas in almost all reviewed articles, ANNs were applied to two-

dimensional images, in e.g. microscopy and medical imaging (CT and MR imaging), three-dimensional modalities enjoy an increasingly widespread use. Adding an extra spatial dimension leads to a combinatorial explosion in the number of parameters. One way to cope with this problem is to develop feature-based pattern recognition approaches; another way would be to design an architecture that quickly adaptively downsamples the original image.

2.5.3 Artificial neural network issues

In section 2.2, a number of unresolved problems for feed-forward ANNs, SOMs and HNNs was mentioned. Here the lack of a profound theoretical basis for ANNs, the problem of choosing the best architecture and the black-box problem will be considered.

Several theoretical results regarding the approximation capabilities of ANNs have been proven. Although feed-forward ANNs with two hidden layers can approximate any (even discontinuous) function to an arbitrary precision, theoretical results on, e.g., the rate of convergence are lacking. One obstacle in developing a more profound statistical foundation for ANNs is the lack of guaranteed convergence to the global minimum of the error measure. The combination of initial parameters, the topology and the learning algorithm together determine the performance of an ANN after its training has been completed. Furthermore, there is always a danger of overtraining an ANN, as minimising the error measure occasionally does not correspond to finding a well-generalising ANN. Having said that, the large body of work on application of ANNs presented in the last decade provides even novice users with many rules-of-thumb on how to set the various parameters, and methods such as regularisation, early stopping or even ensemble training or bagging can help in avoiding the problem of overtraining.

Another problem is how to choose the best ANN architecture. Although there is some work on model selection [108, 251], no general guidelines exist which guarantee the best trade-off between model bias and variance (see page 12) for a particular size of the training set. Training unconstrained ANNs using standard performance measures such as the mean squared error might even give very unsatisfying results. This, we assume, is the reason why in a number of applications, ANNs were not adaptive at all (e.g. CNNs) or heavily constrained by their architecture (e.g., the Neocognitron, shared weight ANNs). Note that this does not automatically imply that unconstrained ANNs should not be applied to image processing problems. It does indicate, however, that great care should be taken when assessing performance of particular ANNs and that as much prior knowledge as possible should be used in both ANN design and training.

ANNs suffer from what is known as the black-box problem: the ANN, once trained, might perform well but offers no explanation on how it works. That is, given any input a corresponding output is produced, but it cannot be easily explained why this decision was reached, how reliable it is, etc. In image understanding, the black-box problem of

ANNs is certainly problematic, so their use in such applications will remain limited. In some image processing applications, e.g., monitoring of (industrial) processes, electronic surveillance, biometrics, etc. a measure of the reliability is highly necessary to prevent costly false alarms. In such areas, it might even be preferable to use other, less well performing methods that do give a statistically profound measure of reliability.

2.6 Conclusions

This survey was structured according to the six steps in the image processing chain: pre-processing, feature extraction, segmentation, object recognition, image understanding and optimisation. ANNs have been trained to perform one or more of these tasks with various degrees of success:

- In pre-processing, several regression ANNs were developed for reconstruction and restoration. Often, these ANNs were not adaptive, or only partially. A general conclusion was that neural solutions are, in general, truly interesting when existing algorithms fail or when ANNs may reduce the amount of computation considerably.
- For enhancement, prior knowledge was often used to restrict the applied feed-forward regression and classification ANNs. The two main feature extraction methods were nonlinear mapping (by auto-associator ANNs) and clustering (by SOMs).
- Many ANN approaches for segmentation have been developed. Among the diverse segmentation tasks, texture segregation is the segmentation problem that has most frequently been attacked by an ANN classifier.
- Object recognition is another problem which has received much attention in the literature on ANN applications in image processing. Although many successful applications were discussed, there are some problems left to investigate. For example, object occlusion and multiple occurrence of objects has hardly been considered.
- Image understanding is a dubious application of ANNs because of their black-box character and the need for large numbers of images as training and test sets. As long as there is no accepted facility for explaining why a particular class label has been assigned to a case, black-box classifiers will not be widely applied in image understanding.
- Finally, the HNN can solve optimisation problems. However, several issues remain problematic, such as mapping the problem at hand to the HNN architecture and bypassing the high dependency of the initial configuration. HNNs become an interesting alternative to conventional techniques when the latter cannot solve the

	Image processing task (section)							
		Reconstruction (2.4.1)	Restoration (2.4.1)	Enhancement (2.4.2)	Feature extraction (2.4.2)	Segmentation (2.4.3)	Object recognition (2.4.4)	Image understanding (2.4.5)	Optimisation (2.4.6)
ANN type	ANN architecture								
Feed-forward									
	Perceptron					•			
	Multi-layer, regression		•		•	•			
	Multi-layer, classification				•		•	•	•
	Auto-associatior					•			
	Radial basis function						•		
	Shared weights							•	
	Recursive							•	
	Neocognitron							•	
Self-organising									
	Adaptive resonance theory (ART)				•			•	
	Self-organising map (SOM)					•	•	•	•
Hopfield									
	Hopfield	•	•		•	•		•	•
Hardware-based									
	Cellular (CNN)		•		•	•			
	Generalised adaptive neural filters (GANF)		•						
	Associative memories (and RAM)						•	•	•
Other									
	Fuzzy neural/neuro-fuzzy		•			•	•	•	
	Various	•					•	•	

Table 2.1: An overview of ANN types and architectures used for image processing tasks. The three ANN architecture types introduced in section 2.2 (feed-forward, self-organising and Hopfield) are the most widely used. Clearly, some ANN architectures are applicable only to specific steps in the image processing chain.

optimisation problem, either because of its nonlinear character or because of the computational complexity.

An overview of ANN architectures used for different image processing tasks is given in table 2.1. It shows that the three main ANN models described in section 2.2 are most widely used, although a large number of more exotic architectures have been applied to different problems as well. Although supervised feed-forward networks have been applied in many steps of the image processing chain, in the earlier steps they were often restricted. Training unrestricted feed-forward ANNs only is useful when a good ground truth is available, such as in object recognition and image understanding. Obviously, unsupervised methods are useful mainly in situations where a ground truth is not known, e.g. segmentation and feature extraction. Finally, the more hardware-like ANNs are useful in any situation where large amounts of data have to be processed quickly. However, as these ANNs have limited learning capabilities, they are vehicles for using prior knowledge rather than fully adaptive algorithms.

Although one of the often mentioned advantages of ANNs is the possibility of hardware (VLSI) implementation, it would seem that this is not used in practice very often. The same holds for ANNs as for parallel computation in general: the application has to be costly enough to warrant the development of highly specialised hardware, since traditional sequential computing methods become cheaper and faster each year (cf. Moore's law [293]), and software remains much more flexible than hardware.

It can be concluded that ANNs can be useful tools in image processing problems for either classification, regression or for (supervised and unsupervised) feature extraction. One of the major advantages of using ANNs for image processing problems must be that they present the user with a very powerful tool with wide applicability. The advent of sophisticated ANN simulation packages (e.g. MATLAB [179] or SNNS [390]) makes for easy application.

As was mentioned in chapter 1, this thesis will focus both on actual applications of neural networks to image processing tasks and the problems discussed above:

- the choice of ANN architecture;
- the use of prior knowledge about the problem in constructing both ANNs and training sets;
- the black-box character of ANNs.

In the next chapters an ANN architecture developed specifically to address these problems, the shared weight ANN, will be investigated. However, these questions will play a role in all of the subjects discussed in this thesis.

3

SHARED WEIGHT NETWORKS FOR OBJECT RECOGNITION

3.1 Introduction

In this chapter, some applications of shared weight neural networks will be discussed. These networks are more commonly known in the literature as TDNNs, *Time Delay Neural Networks* [23], since the first applications of this type of network were in the field of speech recognition[1]. Sejnowski et al. used a slightly modified feed-forward ANN in their NETtalk speech synthesis experiment [316]. Its input consisted of an alpha numerical representation of a text; its training target was a representation of the phonetic features necessary to pronounce the text. Sejnowski took the input of the ANN from the "stream" of text with varying time delays, each neuron effectively implementing a convolution function; see figure 3.1. The window was 7 frames wide and static. The higher layers of the ANN were just of the standard feed-forward type. Two-dimensional TDNNs later developed for image analysis really are a generalisation of Sejnowski's approach: they used the weight-sharing technique not only after the input layer, but for two or three layers. To avoid confusion, the general term "shared weight ANNs" will be used. However, when searching the literature, one will also encounter the terms TDNNs, convolutional ANNs or space displacement ANNs.

The rest of this chapter will focus on just a few implementations of shared weight ANNs, those developed by Le Cun et al. [207, 208, 209] and Fogelman Soulie and Viennet [109, 364]. These ANN architectures are interesting, in that they incorporate prior knowledge of the problem to be solved – object recognition in images – into the structure of the ANN itself. The first few layers of these ANNs act as convolution filters on

[1]The basic mechanisms employed in TDNNs, however, were known long before. In 1962, Hubel and Wiesel introduced the notion of receptive fields in mammalian brains [164]. Rumelhart et al. in their 1986 paper [306] proposed the idea of sharing weights for solving the T-C problem, in which the goal is to classify a 3×3 pixel letter T and a 3×2 pixel letter C, independent of translation and rotation [243].

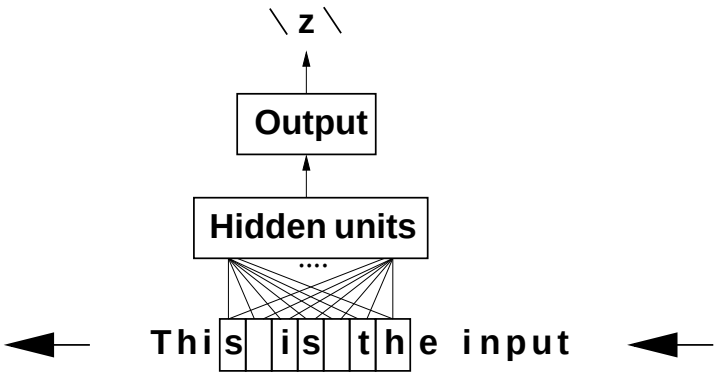


Figure 3.1: The operation of the ANN used in Sejnowski's NETtalk experiment. The letters (and three punctuation marks) were coded by 29 input units using place coding: that is, the ANN input vector contained all zeroes with one element set to one, giving $7 \times 29 = 203$ input units in total. The hidden layer contained 80 units and the output layer 26 units, coding the phoneme.

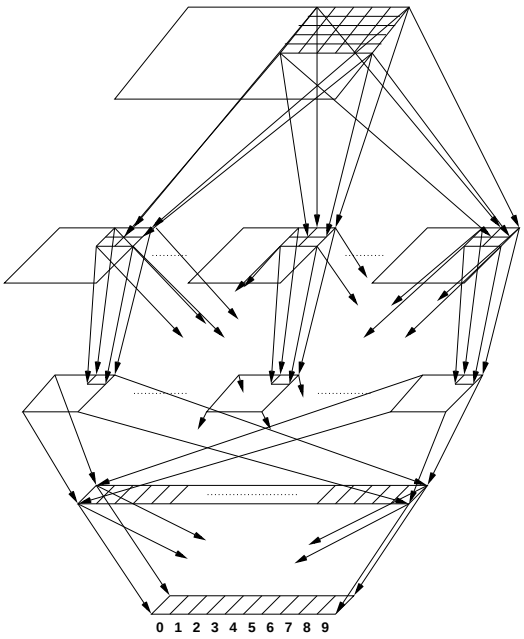
the image, and the entire ANN can be seen as a nonlinear filter. This also allows us to try to interpret the weights of the trained ANNs in terms of image processing operations.

In section 3.2, the basic shared weight architecture used in this chapter will be introduced, as well as some variations. Next, two applications, to handwritten digit recognition (section 3.3) and automatic target recognition (section 3.4), will be shown. The chapter ends with a discussion on shared weight ANNs and the results obtained in section 3.5.

3.2 Shared weight networks

The ANN architectures introduced by Le Cun et al. [207, 208, 209] use the concept of sharing weights, that is, a set of neurons in one layer using the same incoming weight. The use of shared weights leads to all these neurons detecting the same feature, though at different positions in the input image (*receptive fields*); i.e. the image is convolved with a kernel defined by the weights. The detected features are – at a higher level – combined, to obtain shift-invariant feature detection. This is combined with layers implementing a subsampling operation to decrease resolution and sensitivity to distortions.

Le Cun et al. actually describe several different architectures, though all of these use the same basic techniques. The architecture as proposed in [207] will be discussed here as an example (see figure 3.2); differences with other implementations will be discussed later.



		# Neurons	# Connections	# Weights
L1	Input layer			
	16 x 16	256		
	12 x (8 x 8) x (5 x 5 + 1)		19968	
	12 x (8 x 8) x (1) + 12 x (5 x 5)			1068
L2	Feature maps			
	12 x (8 x 8)	768		
	12 x (4 x 4) x (8 x (5 x 5) + 1)		38592	
	12 x (4 x 4) x (1) + 12 x (8 x (5 x 5))			2592
L3	Subsampling maps			
	12 x (4 x 4)	192		
	30 x (12 x (4 x 4) + 1)		5790	5790
L4	Hidden layer			
	30	30		
	10 x (30 + 1)		310	310
L5	Output layer			
	10	10		
	Total	1256	64660	9760

Figure 3.2: The LeCun shared weight ANN.

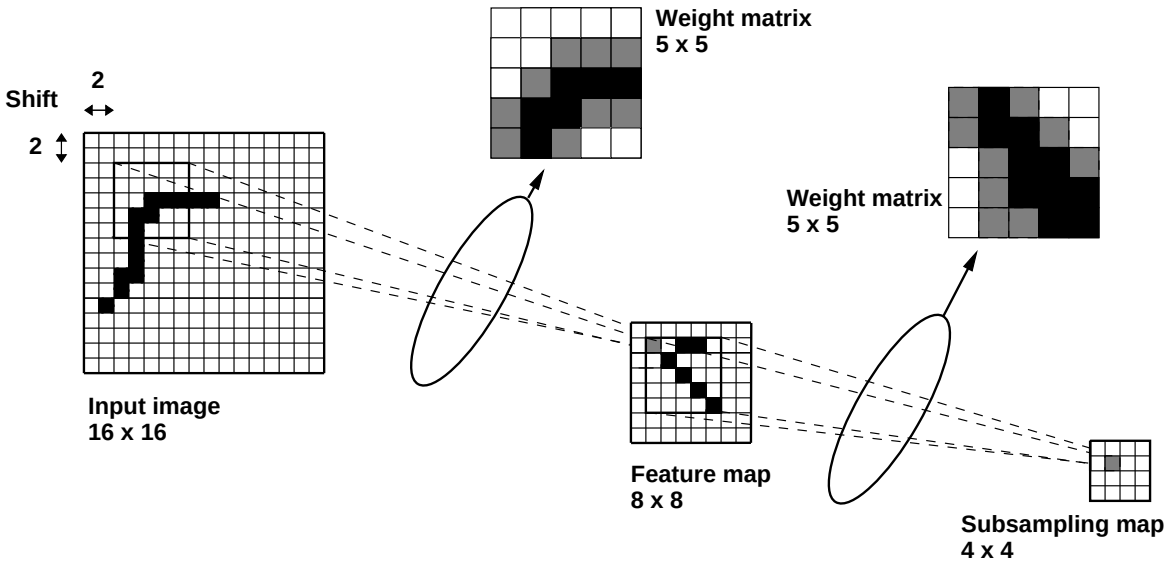


Figure 3.3: A feature map and a subsampling map.

3.2.1 Architecture

The LeCun ANN, shown in figure 3.2, comprises at least 5 layers, including input and output layers:

- The **input layer** consists of a grey-level image of 16×16 pixels.
- The second layer contains the so-called **feature maps**; see figure 3.3. Each neuron in such a feature map has the same 5×5 set of incoming weights, but is connected to a square at a unique position in the input image. This set can be viewed as a convolution filter, or template; that is, if a neuron in a feature map has high output, this corresponds to a match with the template. The place of the match in the input image corresponds to the place of the neuron in the feature map. The image is under-sampled, as the receptive field for two neighbouring neurons is shifted two pixels in the input image. The rationale behind this is that, while high resolution is important for detecting a feature, it is not necessary to know its position in the image with the same precision.

  Note that the number of connections between the input and feature map layer is far greater than the number of weights, due to the weight-sharing. However, neurons do *not* share their bias. Figure 3.2 shows the number of neurons, connections and weights for each layer.
- The third layer consists of **sub-sampling maps** (figure 3.3). This layer is included mainly to reduce the number of free parameters. The principle is the same as for the feature maps: each neuron in a sub-sampling map is connected to a 5×5 square and all neurons in one sub-sampling map share the same set of 25 weights. Here, too, the feature map is under-sampled, again losing some of the information about the place of detected features.

  The main difference however, is that each neuron in a sub-sampling map is connected to more than one feature map. This mapping of feature maps onto sub-sampling maps is not trivial; Le Cun et al. use different approaches in their articles. In [207], only the number of feature maps connected to each sub-sampling map, 8, is mentioned; it is not clear which feature maps are linked to which sub-sampling maps. In [209] however, table A.2 in appendix A.1 is given. Again, due to the use of shared weights, there are significantly less weights than connections (although biases are not shared). See figure 3.2 for an overview.
- The output of the sub-sampling map is propagated to a **hidden layer**. This layer is fully connected to the sub-sampling layer. The number of neurons is 30.
- The **output layer** is fully connected to the hidden layer. It contains 10 neurons, and uses place coding for classification; the neurons are numbered $0 \ldots 9$, and the neuron with the highest activation is chosen. The digit recognised is equal to the neuron number.

The total number of neurons in the ANN is 1256. Without weight sharing, the total number of parameters would be 64660, equal to the number of connections. However, the total number of *unique* parameters (weights and biases) is only 9760.

Shared weight ANNs can be trained by any standard training algorithm for feed-forward ANNs ([147, 148]), provided that the derivative of the cost function with respect to a shared weight is defined as the sum of the derivatives with respect to the non-shared weights [364]. The individual weight updates are used to update the bias for each neuron, since biases are not shared.

Clearly, the architecture presented uses prior knowledge (recognising local features, combining them at a higher level) about the task to solve (i.e., object recognition), thus addressing the problem discussed in section 2.5.2. In [332], the authors show that this approach indeed gives better performance. They compare three simple architectures: a standard back-propagation ANN, an ANN with one feature map and one sub-sampling map and an ANN with two feature maps, each mapped onto one sub-sampling map. It is shown that the more prior knowledge is put into the ANN, the higher its generalisation ability[2]. This experiment will be discussed in chapter 4.

3.2.2 Other implementations

Although the basics of other ANN architectures proposed by Le Cun et al. and others are the same, there are some differences to the one discussed above [207]. In [208], an extension of the architecture is proposed, which from here on will be called "LeNet". Although this ANN has a larger number of connections, the number of unique parameters is even lower than that of the LeCun ANN. LeNet further differs from the LeCun ANN in a number of ways:

- after the second layer, there are effectively two equal, independent subnetworks;
- the subsampling layer is even more constrained, by not only sharing weights between subsampling masks, but demanding that all weights in one map are equal as well (effectively implementing a uniform filter);
- there is no hidden layer before the output layer.

More details of the LeNet architecture can be found in appendix A.2.

The "LeNotre" architecture is a proposal by Fogelman Soulie et al. in [109] and, under the name Quick, in [364]. It was used to show that the ideas that resulted in the construction of the ANNs described above can be used to make very small ANNs that still perform reasonably well. In this architecture, there are only two feature map layers of

[2]Generalisation ability is defined as the probability that a trained ANN will correctly classify an arbitrary sample, distinct from the training samples. It is therefore identical to the test error for sufficiently large testing sets drawn from the same distribution as the training set.

two maps each; the first layer contains two differently sized feature maps. The LeNotre ANN is described in more detail in appendix A.3.

3.2.3 Related work

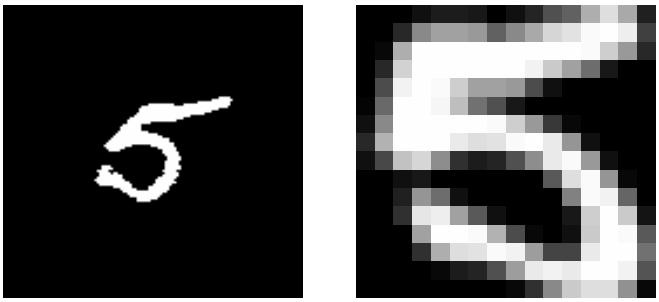
Shared weight ANNs have been applied to a number of other recognition problems, such as word recognition [24], cursive script recognition [313], face recognition [109, 205, 364], automatic target recognition [120] and hand tracking [259]. In a different application, Obellianne et al. [260] describe an architecture in which two shared weight ANNs are coupled at the output layer, in order to build an associative memory. It was used to classify noisy tomography images, and performed well in reducing noise even when the noise-level was increased.

Other architectures employing the same ideas can be found as well. In [117], an ANN architecture specifically suited to object recognition is proposed; the Neocognitron. It is based on the workings of the visual nervous system, and uses the technique of receptive fields and of combining local features at a higher level to more global features (see also 2.4.4). The ANN can handle positional shifts and geometric distortion of the input image. However, the architecture of the Neocognitron is more complicated than that of the LeCun ANN. For example, it uses two kinds of neurons; combines excitatory and inhibitory links; and mixes fixed and variable links. It also differs from the LeCun architecture in the fact that it is trained by unsupervised learning.

Others have applied standard feed-forward ANNs in a convolution-like way to large images. Spreeuwers [337] and Greenhill and Davies [135] trained ANNs to act as filters, using pairs of input-output images; for example, Spreeuwers used a ray-traced image of some boxes as input and the hand-edited edge map of the same picture as the training goal. In his experiments, simple 3-layer ANNs were used containing only one hidden layer, in contrast with the elaborate architectures described before. Furthermore, the input neurons were completely connected to the hidden layer. However, applying such ANNs to all possible input windows in an image (as in a convolution) makes it equivalent to a shared weight ANN with one feature map. Finally, some approaches use convolution-like ANNs with receptive fields, but without shared weights, e.g. the face detection ANNs of Rowley et al. [305].

3.3 Handwritten digit recognition

This section describes some experiments using the LeCun, LeNet and LeNotre ANNs in a handwritten digit recognition problem. For a more extensive treatment, see [68]. The ANNs are compared to various traditional classifiers, and their effectiveness as feature extraction mechanisms is investigated.



(a) (b)

Figure 3.4: A digit before (a) and after (b) pre-processing.

First, in section 3.3.1, the handwritten digit data set will be introduced. Next, the experiments performed will be described and discussed in section 3.3.2, and results will be compared to those found in the literature in section 3.3.3. Finally, some experiments to judge the power of shared weight ANNs as feature extractors will be discussed in section 3.3.4.

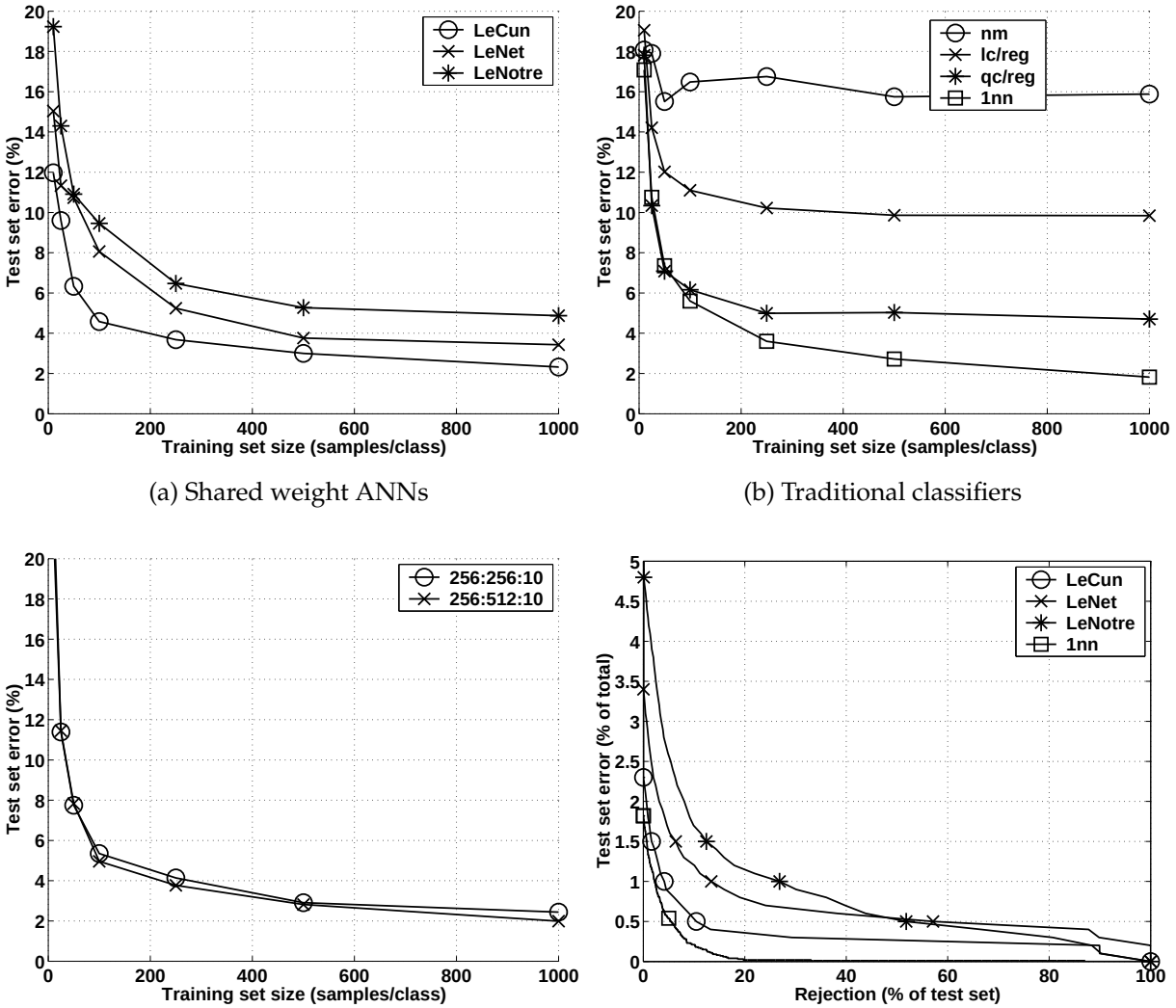
3.3.1 The data set

The data set used in the experiments was taken from the Special Database 3 distributed on CD-ROM by the U.S. National Institute for Standards and Technology (NIST) [382]. Currently, this database is discontinued; it is now distributed together with Database 7 as Database 19. Of each digit, 2,800 samples were available. After randomising the order per class, the set was split into three parts: a training set of 1,000 images per class, a testing set of 1,000 images per class and a validation set of 500 images per class. The latter set was used in the ANN experiments for early stopping: if the error on the validation set increased for more than 50 cycles continuously, training was stopped and the ANN with minimum error on the validation set was used. This early stopping is known to prevent overtraining [154, 283].

The binary digit images were then pre-processed in the following steps [68]:

- shearing, to put the digit upright;
- scaling of line width, to normalise the number of pixels present in the image;
- segmenting the digit by finding the bounding box, preserving the aspect ratio;
- converting to floating point and scaling down to 16×16 using low-pass filtering and linear interpolation.

Figure 3.4 shows an example.



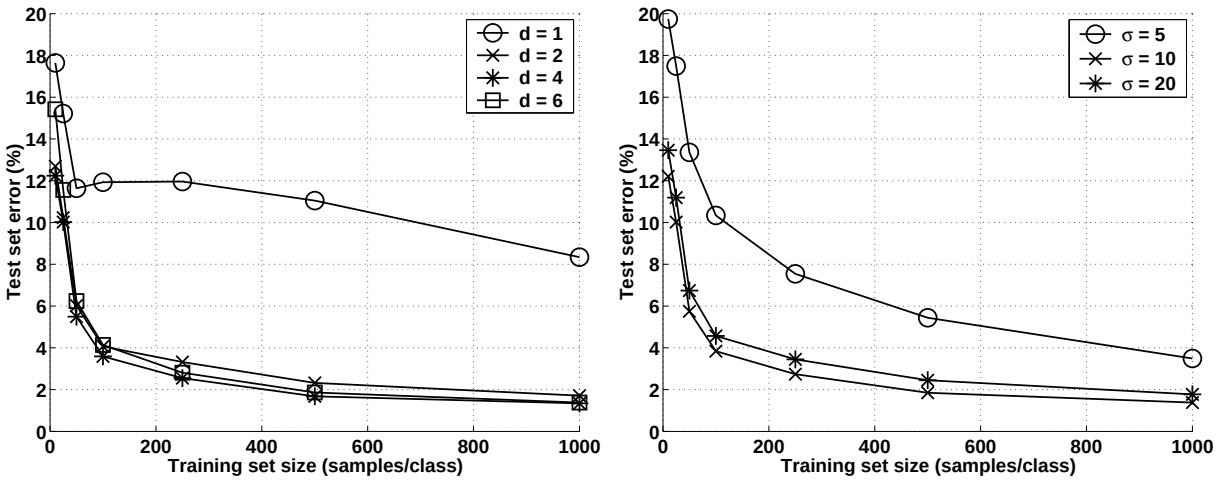
(a) Shared weight ANNs

(b) Traditional classifiers

(c) Feed-forward ANNs

(d) Reject curves

Figure 3.5: Classification errors on the testing set, for (a) the LeCun, LeNet and LeNotre ANNs; (b) the nearest mean classifier (nm), linear and quadratic Bayes plug-in rules (lc, qc) and the 1-nearest neighbour classifier (1nn); (c) standard one hidden layer feed-forward ANNs with 256 and 512 hidden units. (f) Reject-error curves on the testing set for the shared weight ANNs and the 1-nearest neighbour classifier (1nn), all trained on the entire training set (1,000 samples per class).



(a) Polynomial SVMs

(b) Radial basis SVMs

Figure 3.6: Classification errors on the testing set, for (a) SVMs with a polynomial kernel function of degrees 1, 2, 4 and 6 and (b) SVMs with a radial basis kernel function, $\sigma = 5$, 10, 20.

3.3.2 Experiments

Instances of the LeCun, LeNet and LeNotre ANNs were trained on subsets of the training set containing 10, 25, 50, 100, 250, 500 and 1000 samples per class. Following [207], weights and biases were initialised randomly using a uniform distribution in the range $\left[-\frac{2.4}{F}, \frac{2.4}{F}\right]$, where F was the total fan-in of a unit (i.e. the number of incoming weights). Back-propagation was used for training, with a learning rate of 0.5 and no momentum. Training targets were set to 0.9 for the output neuron coding the right digit class, and 0.1 for the other output neurons. After training, the testing set was used to find the error.

For comparison, a number of traditional classifiers were trained as well: the nearest mean linear classifier (which is denoted nm in the figures), the linear and quadratic *Bayes plug-in classifiers*[3] (lc and qc) and the 1-nearest neighbour classifier (1nn) (see e.g. [85, 114, 377] for a discussion on these statistical pattern classifiers). For the Bayes plug-in classifiers, regularisation was used in calculating the 256×256 element covariance matrix $\mathbf{C}$:

$$\mathbf{C}' = (1 - r - s)\,\mathbf{C} + r\,\mathrm{diag}(\mathbf{C}) + \frac{s}{256}\,\mathrm{tr}(\mathbf{C})\mathbf{I} \tag{3.1}$$

where $\mathrm{diag}(\mathbf{C})$ is the matrix containing only the diagonal elements of $\mathbf{C}$, $\mathrm{tr}(\mathbf{C})$ is the trace of matrix $\mathbf{C}$, and using $r = s = 0.1$. Furthermore, two standard feed-forward

[3]The Bayes classifier assumes models for each of the classes are known; that is, the models can be "plugged in". Plugging in normal densities with equal covariance matrices leads to a linear classifier; plugging in normal densities with different covariance matrices per class leads to a quadratic classifier.

ANNs were trained, containing one hidden layer of 256 and 512 hidden units, respectively. Finally, support vector classifiers (SVMs, [62, 351, 362]) were trained with polynomial kernels of various degrees and with radial basis kernels, for various values of σ.

Results are shown in figures 3.5 (a)-(c) and figure 3.6. The shared weights ANNs perform well, better than most traditional classifiers. Of the three ANNs, the best seems to be the LeCun ANN, which has the largest number of parameters. The smallest ANN, LeNotre, gives the largest errors. The 1-nearest neighbour classifier and the standard feed-forward ANNs perform as well as the shared weight ANNs or slightly better, as do the SVMs.

In general, classifiers performing better also have many more parameters and require more calculation in the testing phase. For example, when trained on 1,000 samples per class the LeCun ANN (2.3% error) performs slightly worse than the 1-nearest neighbour classifier (1.8% error) and the best performing SVMs (e.g. radial basis kernels, $\sigma = 10$: 1.4% error), but slightly better than the 256 hidden unit feed-forward ANN (2.4% error). The LeCun ANN has 63,660 parameters, requiring as many FLOPs (floating point operations) to test one sample. In contrast, the 1-nearest neighbour rule, trained on 1,000 samples per class, requires 10,000 distance calculations in 256 dimensions, i.e. roughly 5,120,000 FLOPs. Similarly, the SVM uses a total of 8,076 support vectors in its 10 classifiers, requiring 4,134,912 FLOPs. However, the fully connected feed-forward ANN with 256 hidden units requires $256 \times 256 + 256 \times 10 = 68{,}096$ FLOPs, a number comparable to the LeCun ANN. In conclusion, the shared weight ANNs seem to perform well given their limited number of parameters, but a standard feed-forward ANN performs equally well using the same amount of computation. This indicates that the restrictions placed on the shared weight ANNs are not quite necessary to obtain a good performance. It also contradicts the finding in [332] that the use of shared weights leads to better performance.

A reject rule was also implemented, by imposing a threshold on the relative difference between the highest and second highest output of an ANN. For the 1-nearest neighbour classifier, rejection was based on the relative difference between the distance to the nearest neighbour and the nearest neighbour of another class. Reject-error curves are given in figure 3.5 (d), for ANNs and the 1-nearest neighbour classifier trained on the entire training set. For low rejection thresholds, the ranking of the classifiers remains the same.

3.3.3 Comparison

The results obtained here, summarised in table 3.1 (a), are comparable to those found in literature. In 1992, NIST organised a competition in handwritten digit recognition in which several companies and universities competed. Performances of the systems

ANN	Error	Reject
LeCun	2.3 %	4.1 %
LeNet	3.4 %	13.4 %
LeNotre	4.9 %	26.9 %

(a) Performance of shared weight ANNs.

Institute	Feature extractor	Classifier	Error
OCR Systems	unknown	unknown	2.6 %
AT&T Bell Labs	none	k-NN with tangent distance[a]	3.2 %
AEG Electrocom	pre-processing, PCA	polynomial	3.4 %
ELSAG-BAILEY	unknown	"neural algorithm"	3.4 %
IBM	morphology, contours, ...	ANN	3.5 %
AT&T Bell Labs	LeNet	LeNet	3.7 %

(b) Performance of winning algorithms in the 1992 NIST competition.

No.	Paper(s)	ANN	Dataset	Learn set	Test set	Error	Reject
1	[207, 209]	LeCun	USPS[b]	7,291	2,007	5.0 %	n/a
2	[208]	LeNet	USPS	7,291	2,007	3.4 %	n/a
3	[109, 364]	LeNet	NIST-3	15,000	5,000	1.4 %	0.8 %
4	[109, 364]	LeNotre	NIST-3	15,000	5,000	4.1 %	8.8 %

(c) Performance of shared weight ANNs found in literature.

Table 3.1: An overview of performance of various approaches. "Reject" is the percentage of the objects it is necessary to reject to reach a 1% classification error on the total set.

[a] This technique defines a set of possible transformations on an image (translation, rotation, dilation, ...) controlled by a set of parameters. A special distance measure, the *tangent distance*, is then introduced which is locally invariant to these transformations. This distance measure can be computed efficiently by approximating the nonlinear transformation parameter manifold locally by a linear hyperplane. The tangent distance is then used as a feature vector in subsequent classification [328, 364].

[b] A database of the United States Post Office. Zip code images were extracted from envelopes passing through Buffalo, NY and segmented into 5 digit images by postal service contractors.

competing are given in table 3.1 (b). Note that these results are not directly comparable to the results obtained here for a number of reasons: competitors were free to choose their own training set and not required to disclose their algorithms, and NIST used a set of 100,000 digits written under other circumstances to test the systems.

An overview of reported performances of shared weight ANNs is given in table 3.1 (c). Again, these numbers are difficult to compare. In experiments 1 and 2, a different data set was used. The results of experiments 3 and 4, in which the database used was the same, are better than the results obtained here (table 3.1 (a)). However, the data was pre-processed and split differently and not randomised. Furthermore, the authors used the Levenberg-Marquardt training algorithm [148], did not use early stopping, and do not report how many repetitions of the experiments led to their results. Finally, they used a different reject function: the total Euclidean distance between the actual output vector of an ANN and the target vector.

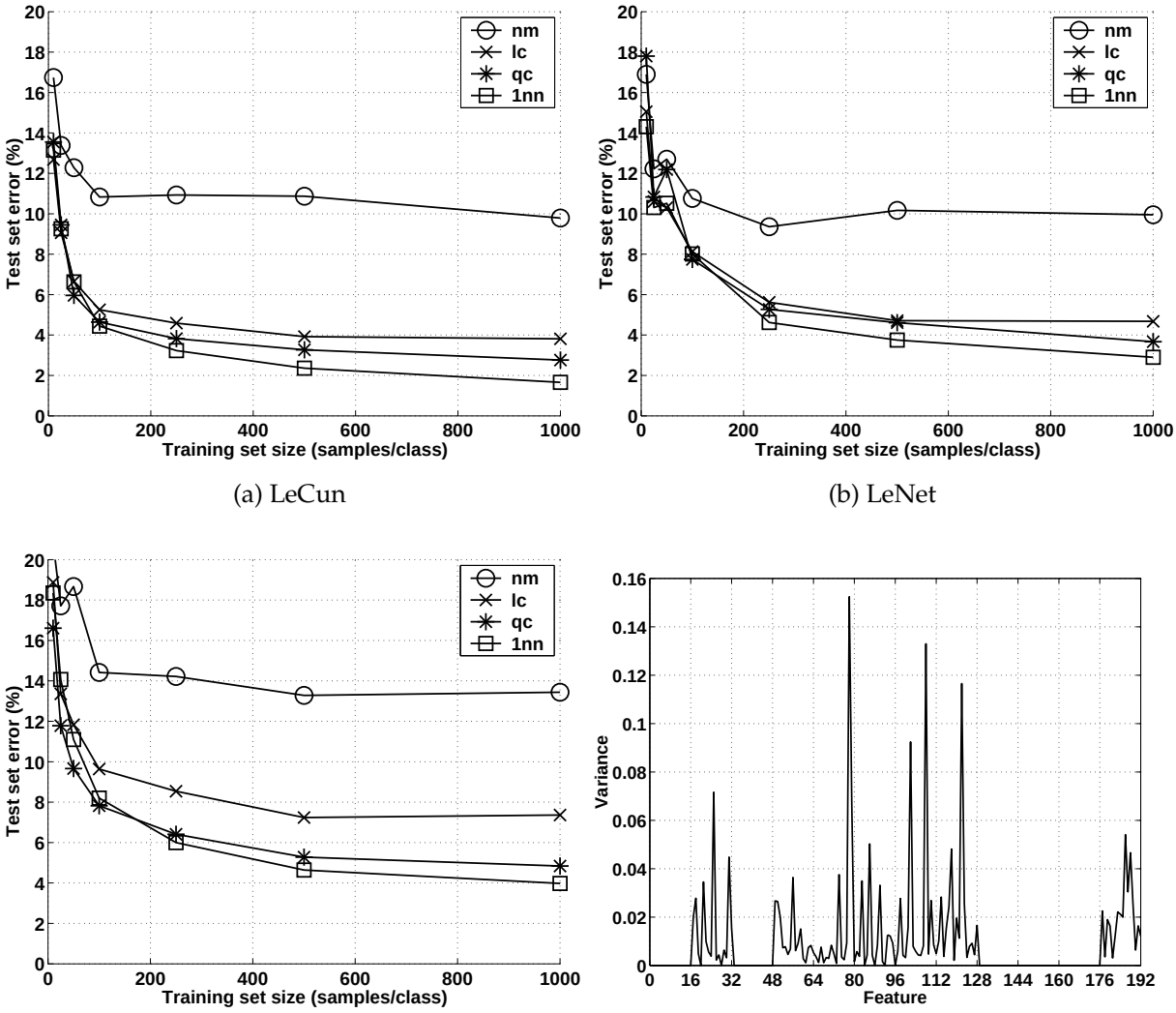
3.3.4 Feature extraction

In appendix A, images of the LeCun, LeNet and LeNotre ANNs trained on the entire training set are shown (in figures A.2, A.4 and A.6, respectively). Some feature maps seem to perform operations similar to low-level image processing operators such as edge detection. It is also noteworthy that the extracted features, the outputs of the last subsampling layer, are nearly binary (either high or low). However, visual inspection of the feature and subsampling masks in the trained shared weight ANNs in general does not give much insight into the features extracted. In the next chapter, a number of simpler problems will be studied in order to learn about the feature extraction process in shared weight ANNs.

Here another approach is taken to investigate whether the shared weight ANNs extract useful features: the features were used to train other classifiers. First, the three architectures were cut halfway, after the last layer of feature maps or subsampling maps, so that the first part could be viewed to perform feature extraction only. The original training, testing and validation sets were then mapped onto the new feature space by using each sample as input and finding the output of this first part of the ANN. This reduced the number of features to 192 for the data sets generated by the LeCun and LeNet ANNs, and 54 for those generated by the LeNotre ANN.

In the experiments, a number of classifiers were trained on these data sets: the nearest mean linear classifier (nm), the Bayes plug-in linear and quadratic classifier (lc and qc) and the 1-nearest neighbour classifier (1nn). For the Bayes plug-in classifiers, the estimate of the covariance matrix was regularised in the same way as before (3.1), using $r = s = 0.1$. Figures 3.7 (a)-(c) show the results for the features extracted by the LeCun, LeNet and LeNotre ANNs, respectively.

For one ANN, LeNet, it was found that some of the feature maps did not perform any



(a) LeCun

(b) LeNet

(c) LeNotre

(d) Variance of features extracted by LeNet

Figure 3.7: Performance of various classifiers trained on data sets extracted from the feature extraction parts of the (a) LeCun, (b) LeNet and (c) LeNotre ANNs. (d) Variance of the output of the LeNet ANN's last feature extraction layer over 10,000 samples.

function. Figure 3.7 (d) shows the variance over the data set extracted from the last subsampling map layer of this ANN: for roughly 80 units, it is near zero. As discussed in section 3.2.2, the LeNet ANN consists of a number of rather independent parts – figure A.3 shows how any map never receives input from more than one other map. In some of the parts, the weights have grown very high, pushing the summed input of the units far into the nonlinear part of the transfer function, effectively disabling them. Therefore, the aforementioned classifiers were trained on outputs with variance larger than 1.0×10^{-6} only. This also explains why the LeNet ANN performed worse than expected.

In all cases the 1-nearest neighbour classifier performs better than the classification parts of the ANNs themselves. The Bayes plug-in quadratic classifier performs nearly as well as the ANNs (compare figure 3.5 (a) to figures 3.7 (a)-(c)). This is to be expected, as for the shared weight ANNs the last layer, having only a small number of units, cannot implement highly nonlinear classifiers. Interestingly, the LeCun ANN does not seem to use its 30 unit hidden layer to implement a highly nonlinear classifier, as the difference between this ANN's performance and that of the Bayes plug-in quadratic classifier is the same as for the shared weight ANNs lacking this hidden layer. Clearly, for all shared weight ANNs, most of the work is performed in the shared weight layers; after the feature extraction stage, a quadratic classifier suffices to give good classification performance.

Most traditional classifiers trained on the features extracted by the shared weight ANNs perform better than those trained on the original feature set (figure 3.5 (b)). This shows that the feature extraction process has been useful. In all cases, the 1-nearest neighbour classifier performs best, but only when it is trained on the features extracted by the LeCun ANN is its performance (1.7% error for 1,000 samples/class, see figure 3.7 (a)) better than on the original data set (1.8% error, figure 3.5 (b)). On features extracted from both other shared weight ANNs, performance never becomes better than that of the 1-nearest neighbour classifier or the SVMs trained on the entire data set (figures 3.5 (b), 3.6).

3.3.5 Discussion

A number of shared weight ANN architectures were implemented and applied to a handwritten digit recognition problem. Although some non-neural classifiers (such as the 1-nearest neighbour classifier and some support vector classifiers) perform better, they do so at a larger computational cost. However, standard feed-forward ANNs seem to perform as well as the shared weight ANNs and require the same amount of computation. The shared weight ANN results obtained were compared to those found in the literature and seem to be slightly worse than the state-of-the-art. Differences in experimental setup partly explain these discrepancies. In the LeNet architecture, several parts do not have a function after training; the variance in unit activitity is near zero. The training parameters could probably be tuned to prevent this; however, this was not attempted.

Unfortunately, it is very hard to judge visually what features the ANNs extract. However, the ANNs were also tested on their feature extraction behaviour, by using the output of the last subsampling map layer as a new data set in training a number of traditional classifiers. The ANNs indeed act well as feature extractors, as these classifiers performed well. For the largest training set size (1000 samples/class), however, performance was never better than on the original data set, except for the 1-nearest neighbour classifier trained on features extracted by the LeCun ANN.

3.4 Automatic target recognition

This section describes another application of shared weight ANNs, to automatic target recognition (ATR, [25, 26]), in this case the task of recognising nearby tanks in infra-red (IR) images. ANNs have, often successfully, been applied to target recognition problems (for overviews, see [299, 302]). In most of these problems, however, the ANNs are rather dedicated to the task at hand. Several non-standard pre-processing or learning techniques have been developed for these problems (e.g. [290]).

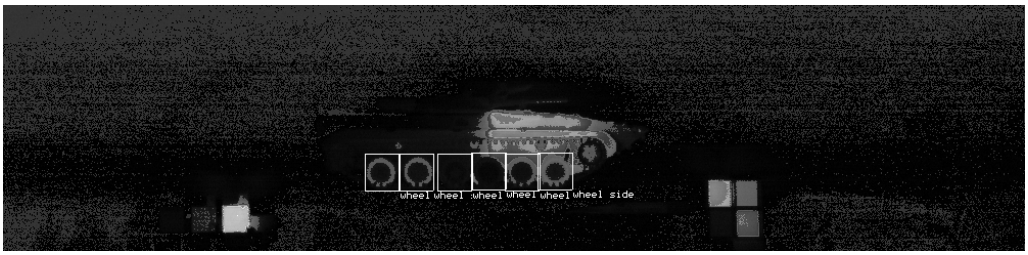
Here, the overall recognition strategy is based on the notion that vehicles include a row of wheels; the ANNs are used as feature detectors to find these wheels. Although the ANNs are trained on individual samples, they are used as nonlinear filters on entire images. After feature detection, a search is performed for a row of wheels. The advantage of such an approach is the fact that, assuming that false alarms from the ANNs are independent, any coincidental occurrence of a row of detections similar to a wheel row pattern is not very likely, thus suppressing the number of false alarms. It is assumed that the distance to the potential object is known (for example using radar, laser range finder, passive range imaging, digital terrain models, etc.), which given a known IFOV (Instantaneous Field Of View) provides the approximate scale parameter. This scale parameter is used to compute the magnification to match the input of the ANN feature detector. The procedure to search for a row of wheels is described in section 3.4.2.

In [120], an approach similar to this is reported, also utilising a shared weight ANN for the recognition of vehicles in IR images. The difference between their work and this lies in the fact that here the ANNs are used only as wheel feature detectors, whereas Gader et al. attempt to find complete vehicles and use shared weight ANNs containing morphological operations.

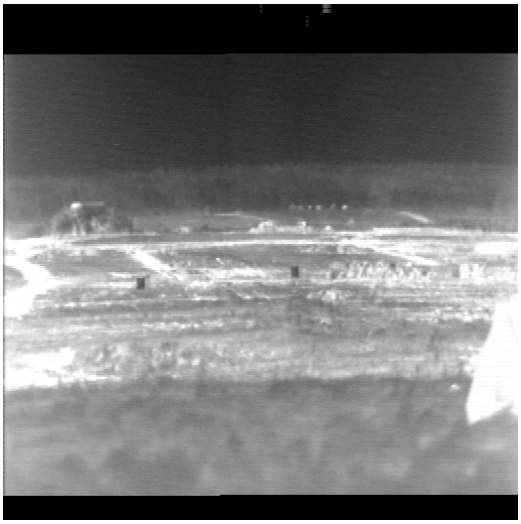
The remainder of this section will start with a discussion of the data set used, in section 3.4.1. Although shared weight ANNs can be applied without any modification, there is a major difference between standard classification problems and this particular problem, in that rejection of unknown samples is of great importance. That is, it is a form of *one-class classification* [79, 350]. This problem will be discussed in section 3.4.2. The resulting system has been tested on scale dependency and sensitivity to clutter in the background of images. Experimental results are given in section 3.4.3, showing that the system performs adequately.

3.4.1 The data set

The data set consisted of a database of infrared (IR) images. 160 12-bit 1000×240 images were available, each of which depicts one vehicle under different rotations. 80 of these images were taken in the $3 - 5\mu$ range, the other 80 in the $8 - 12\mu$ range, all using the



(a)



(b)

(c)

Figure 3.8: Example images: (a) A $3-5\mu$ IR image. Wheel sample bounding boxes are indicated. Note that all 160 images contain a number of "hot plates", which are included to calibrate the cameras. These plates were masked out when testing the system. (b) A clutter image taken with the UA-92 camera. (c) A clutter image taken with the TNO Scorpio camera.

TNO-FEL DUDA-1 camera. Figure 3.8 (a) gives an example of an image. Three extra images were used to test the sensitivity of the system to background clutter. Two of these (512×512 pixels) were taken with the $8-12\mu$ UA-92 camera; the other (512×256) was taken with the TNO Scorpio $8-12\mu$ camera. Examples are given in figure 3.8 (b) and (c). Although the number of clutter images is rather low, the images used are sufficient to get an idea of the clutter sensitivity. All images were then pre-processed by converting to floating point images, subtracting the image mean and dividing the result by two times the image standard deviation. Because the grey value distributions usually were asymmetric, the conversion resulted in images containing values roughly in the range $[-1.0, 7.0]$.

In each image, samples of wheels were manually indicated by a bounding box. Note that, since the images contained various vehicles with several orientations, there

ANN	20 unit hidden layer	10 unit RBF layer	No. of outputs	No. of units	No. of links	No. of weights
LeNotre			2	386	3556	194
LeNotre^{H}	•		2	406	5580	1206
LeNotre^{R}		•	1	395	4440	636
LeNotre^{RH}	•	•	1	415	5920	1376

Table 3.2: The ANN architectures used in the ATR experiments. The basic LeNotre architecture is shown in figure A.5 in appendix A. The other architectures contain either an added RBF layer or an added hidden layer, or both. In the latter case, the hidden layer precedes the RBF layer.

was quite some variation between samples and a large number of partially obscured samples. Then, 25% of the images (40 images, containing 290 wheels in total) was chosen at random and put aside to serve as a testing set. From the remaining 120 images, samples of wheels and background were extracted. Of each wheel in the image, a 16×16 pixel sample was created by extracting the image content in the bounding box (preserving the aspect ratio) and rescaling using low-pass filtering and linear interpolation. Note that in most of the images, the original wheel bounding box width fell in the range 30-40 pixels, meaning the samples had to be scaled by a factor of 0.4-0.6.

Of each sample, a horizontally mirrored version was added to incorporate the prior knowledge that recognition of wheels should be invariant under this transformation. As a side effect, this (artificially) enlarged the training set, which helps in training the ANNs. Furthermore, from each image a number of negative (i.e., non-wheel) samples were extracted: background samples, samples taken between the wheel centres and samples taken above or below the wheel centres. The latter two were added to force the ANN to give precise localised responses on the wheel centres only. The size of background samples was chosen to fall in the same range as the original size of the foreground samples. Of the data set created in this way, 5 random splits into a training set (83.3% of all samples) and a validation set (17.7%) were constructed. On average, the training sets contained 1,494 wheel samples and 2153 background samples and the validation sets 298 wheel samples and 431 background samples.

3.4.2 The vehicle detection algorithm

ANN feature detection

In the ATR experiments, the basic ANN architecture used was the LeNotre ANN (see section 3.2.2). However, in the case of recognising (small) objects against noisy backgrounds, it is important to use a form of rejection or *outlier detection* [350]. Many

advanced approaches to incorporate outlier detection in ANNs have been proposed, among which the use of special learning rules [249] which force ANNs to learn closed decision boundaries and advanced forms of vector quantisation such as Adaptive Resonance Theory (or *ART* [40, 250]). In this work, two simpler solutions were tested. The first is to add an output unit to the ANN which should represent all non-wheel (background) samples. The second, more sensible approach is to use a form of vector quantisation inside the ANN. The standard method of incorporating vector quantisation is to use a radial basis function layer. This layer contains units with a Gaussian transfer function, localising their responses, of which the width σ and the position μ are learned [148]. The advantage is that the response to wheel samples is automatically local, so that far-away (non-wheel) samples will give low ouput. The main disadvantage is that the number of representative vectors has to be chosen in advance.

When an ANN is built with one hidden layer of RBF units, the result is a classifier that is reminiscent of the Parzen estimator [85, 114] (although no actual density estimation is performed). The output of that layer will be low in places where no data is expected and high near data cluster centres, so one output unit can be used which should be high for normal samples and low for background samples. Of course, one could also use additional normal (i.e. sigmoid) hidden layers before the RBF layer, to pre-transform the samples to a space of lower dimensionality.

In total, four variations of the LeNotre architecture were tested, with or without a 20 unit hidden layer and/or a 10 unit RBF layer. They are listed in table 3.2.

Wheel row detection

The output of the ANN is, for each pixel, a number representing the probability of the presence of a wheel at that location. For further processing, a 5×5 pixel maximum filter was applied, after which these local maxima were thresholded using a threshold t. This threshold can be found according to some criterion; for example, in such a way that no false alarms are generated for clutter images.

For wheel row detection, the typical pattern of wheel detections has to be found, i.e. the normal configuration of wheels on the vehicle. For the vehicles considered in this application, the typical pattern is a set of wheels located on a line. This leads to a search space in the shape of a wide rectangle. For the images considered, the observer was at the same height as the object itself. This means that the rectangle always is oriented horizontally, independent of the orientation of the vehicle. Hence the search was executed as a score boarding process: each detected wheel leads to a score within a rectangular 81×9 area around it. A wheel row was detected if for a certain pixel a score higher than or equal to 4 was obtained.

A different approach should be taken if the observer is not located in the plane orthogonal to the rotation axis of the vehicle. In that case the row of wheels has a different

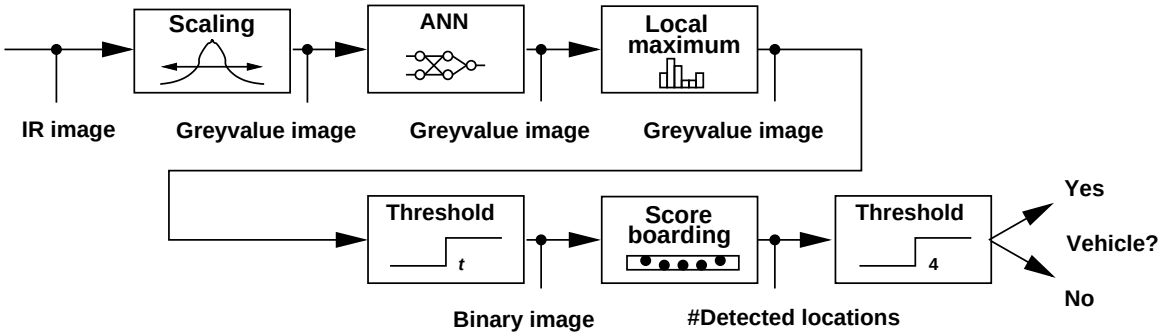


Figure 3.9: The vehicle detection algorithm.

orientation in the image, depending on its rotation angle. This means that during the detection phase, the score boarding rectangle will have to be applied for all rotation angles. However, a finite set of representative rotations of this score boarding rectangle will do. As a side effect, the false alarm rate will increase (as possibly more patterns are erroneously identified as objects). Thus, any knowledge about possible rotation angles should be incorporated in the algorithm to reduce the false alarm rate.

Figure 3.9 shows a schematic presentation of the entire recognition algorithm. Figure 3.10 shows some examples of processed images.

3.4.3 Experiments

Training

Of each of the ANN architectures listed in table 3.2, five instances were trained on differently randomised and split data sets (as discussed in section 3.4.1). In all experiments, the algorithm used was the conjugate gradient descent method [155, 323]. This algorithm has no parameters. Training targets were set to 0.25 for background samples and 0.75 for wheel samples; for the non-RBF ANNs, with two output units, place coding was used. The validation set was used to prevent overtraining, by stopping training when the error on the validation set did not decrease for 50 cycles.

The ANNs were trained on isolated samples, but should be used to "scan" an image for the presence of certain classes, i.e. as nonlinear filters. The input of the ANN feature detectors will be the content of a window sliding over the input image; the result of the filter will be an image. In this image, each pixel contains the probability of the presence of the class represented by that unit at that location for one ANN evaluation. Figures 3.10 (b) and (e) show examples of such filtered images.

In table 3.3 the results gathered during the ANN training phase are given. Note that

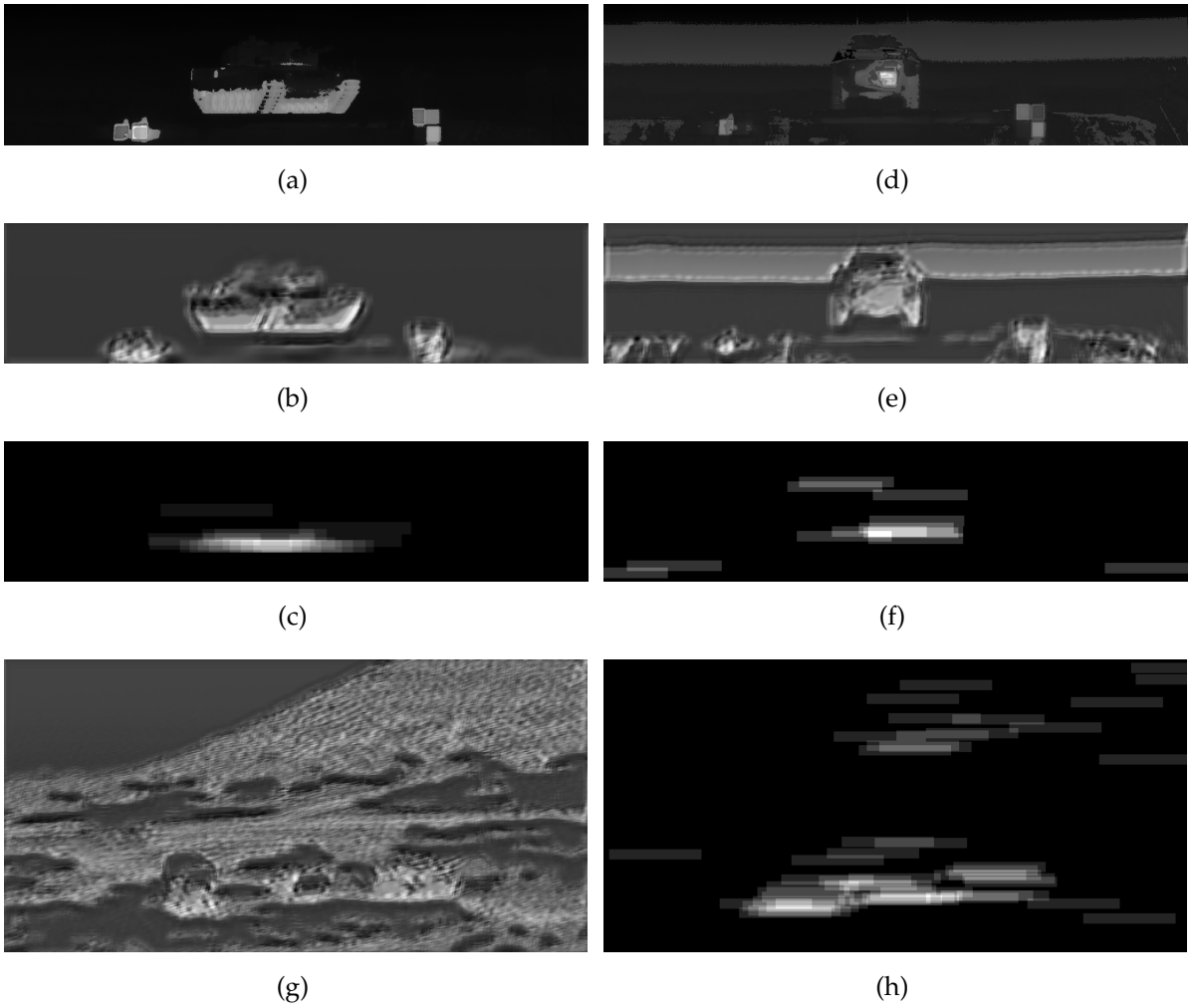


Figure 3.10: The recognition process for three images: an image containing wheels (a)-(c), an image containing no wheels (d)-(f) and the clutter image of figure 3.8 (c) in (g)-(h). The top row contains the original images, the second row contains the outputs of an ANN (in this case, LeNotre$_2^H$) and the third row images show the results of the score boarding process. In all images, the grey value range has been stretched for visualisation purposes. *Note*: for the evaluation of the sensitivity to background clutter, the vehicles in the clutter image have been masked out.

ANN	Error on	Trial 1	2	3	4	5	Min. on val.	Avg.	Std. dev.
LeNotre	Train	19.0	17.5	23.9	22.2	18.1	17.5	20.1	2.8
	Valid.	20.0	18.5	26.3	22.6	19.9	18.5	21.5	3.1
LeNotre^{H}	Train	• 16.6	• 20.6	• 15.8	20.3	• 24.5	15.8	19.6	3.5
	Valid.	21.3	20.4	16.3	22.8	26.1	16.3	17.2	8.7
LeNotre^{R}	Train	• 13.2	20.2	• 19.7	30.0	• 17.4	17.4	20.1	6.2
	Valid.	18.1	22.8	22.1	31.0	17.7	17.7	22.3	5.4
LeNotre^{RH}	Train	40.6	40.9	• 23.9	• 22.5	• 21.1	23.9	29.8	10.0
	Valid.	42.7	41.1	23.2	23.5	24.0	23.2	30.9	10.1

Table 3.3: Training results in % error on the training set. Only ANNs marked with "•" were used in the subsequent testing phase; the other ANNs were judged to perform too poorly to warrant further processing. The minimum errors are those for which the error on the validation set was minimal.

the errors given in this table are *not* test errors and that anomalies may occur due to the rather arbitrary way of measurement. Performance was measured as follows. For a sample to be considered correctly classified,

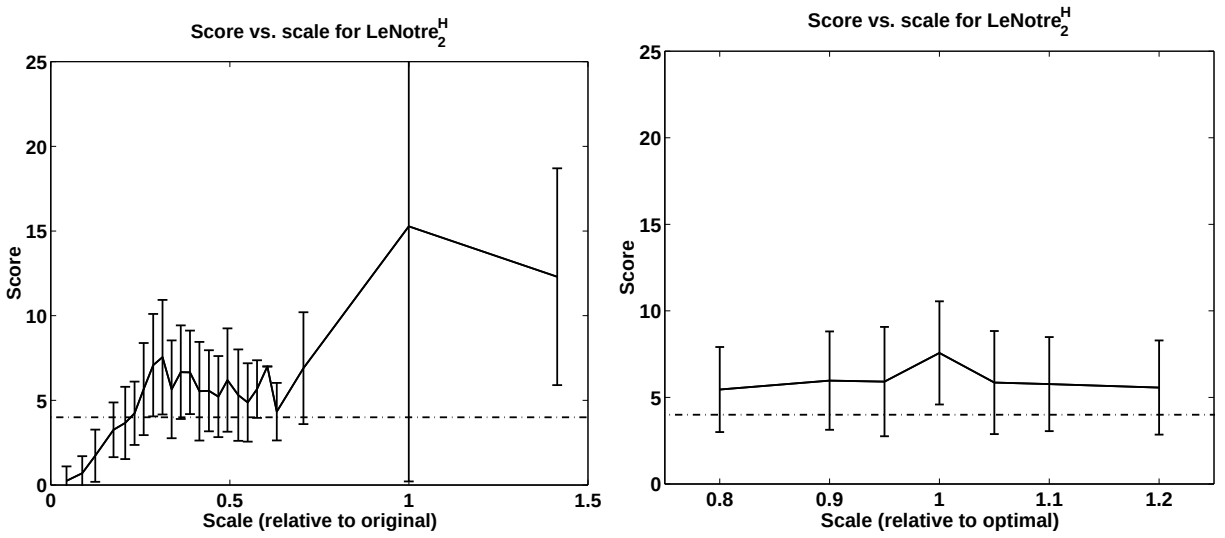
- for non-RBF ANNs (having background as an extra class), the output unit representing the wheel class should be highest;
- for RBF ANNs, the output of the ANN should be larger than 0.5 for a wheel sample, whereas for a background sample it should be smaller than 0.5.

This means that during training the RBF ANNs, 0.5 is used as a rejection threshold. This need not be the optimal choice; however, the optimal threshold can only be found after training. The error percentage can at least be used for comparison between training sessions, RBF ANN architectures and data sets.

It is evident that results are quite erratic: for different trials, very different performances are reached. There are two possible reasons:

- the different splitting of the data into a training and validation set for the five instances, which would indicate that there is not enough training data for the ANN to generalise upon;
- the difficulty of the problem easily leads the ANNs into local minima, depending on the weight initialisation.

Testing the ANNs proved to be highly time-consuming, as each ANN was tested on 40 test images at a large range of scales. To restrict the number of testing runs, only ANNs whose performance was visually judged to be reasonable were considered for further processing. These ANNs are indicated by a "•" in table 3.3. Note that none of



(a) (b)

Figure 3.11: Detection performance for LeNotre$_2^H$ using $t_{lm} = 0.67$: (a) Number of detected patterns versus scale; a scale of 1.0 means the scale of the original image (recall that most samples were scaled with factors between 0.4 and 0.6, see section 3.4.1). Over a relatively large range (0.3-0.7) the number is larger than 4, meaning a vehicle will be detected, and relatively stable. (b) Number of detected patterns around the correct scale; here a scale of 1.0 means the image scaled with the average factor used when the wheel samples were extracted (i.e. the wheels have the same size as those used to train the ANN). Both graphs indicate the average number of patterns detected over all 40 test images; the bars indicate the standard deviation.

the standard LeNotre ANNs were tested; these were all deemed to perform too poorly to be of any further interest.

Performance and scale sensitivity

Figure 3.11 shows the detection performance, using the wheel row detection algorithm, of LeNotre$_2^H$. This was the best performing ANN, but other ANNs gave similar results. Figure 3.11 (a) indicates for threshold $t = 0.67$, for a number of scales, the average number of detected patterns in the 81×9-pixel search area. The 0.67 threshold was found to be optimal for LeNotre$_2^H$: it was the highest threshold which gave no false alarms on the three clutter images. Note that this threshold need not be optimal for all ANNs – see table 3.4 for an overview. Figure 3.11 (b) shows the performance around the optimal scale, that is the scale for which the wheels in the image have on average the same size as was used to train LeNotre$_2^H$, 16×16 pixels. It clearly shows a peak around the optimal scale. However, taking into account the imprecision in the range finding procedure and the errors made when the bounding boxes were indicated in the data set, it is unlikely

ANN	Threshold	Vehicles detected	ANN	Threshold	Vehicles detected
$\text{LeNotre}_1^{\text{H}}$	0.86	23%	$\text{LeNotre}_3^{\text{R}}$	0.75	35%
$\text{LeNotre}_2^{\text{H}}$	0.67	83%	$\text{LeNotre}_5^{\text{R}}$	0.71	59%
$\text{LeNotre}_3^{\text{H}}$	0.72	60%	$\text{LeNotre}_3^{\text{RH}}$	0.76	9%
$\text{LeNotre}_5^{\text{H}}$	0.76	14%	$\text{LeNotre}_4^{\text{RH}}$	0.74	40%
$\text{LeNotre}_1^{\text{R}}$	0.79	35%	$\text{LeNotre}_5^{\text{RH}}$	0.71	49%

Table 3.4: The maximum threshold for each ANN for which no false alarm was found in the clutter images and the corresponding percentage detected of the total number of vehicles.

that one will be able to calculate this optimal scale with enough precision.

If four or more patterns are detected (the dotted line in figure 3.11), the conclusion is that a vehicle is present in the images. The graphs have been somewhat smoothed (using bins of 0.025) to present the results clearly. The figures show that, to a certain extent, the number of detected patterns per unit area is constant and independent of the scale. Only for very small scales this does not hold: the number of detected patterns drops below 4 and consequently no vehicle is detected. This means that tests on clutter images will not have to be performed on an entire range of scales, since a single scale gives a good indication for the rest of the range. Another conclusion is that the ANN feature detector does not only detect wheels; otherwise, one would expect at most 19 (and probably less) detected patterns, since that is the maximum number of wheels present in a test image.

Sensitivity to background clutter

To test the sensitivity of the system to background clutter, it was tested on three clutter images, i.e. images in which the vehicles were masked out. Figure 3.12 shows receiver-operator curves for each ANN. Note that, since there were only 3 clutter images, only 3 levels of false alarm are possible. However, the trend should be clear.

A problem with the evaluation of clutter images was the normalisation step. Recall that each image was normalised by subtracting its mean and dividing by two times its standard deviation (section 3.4.1). Since images containing vehicles have a small number of pixels with relatively high values, this results in a very non-symmetric grey-value distribution. In clutter-images however, the result is quite symmetric and the resulting normalised image has a lower maximum grey-value than an image containing a vehicle. To investigate to what extent the ANNs simply respond to image intensity variation, one ANN ($\text{LeNotre}_2^{\text{H}}$) was tested on the clutter-images for various levels of intensity multiplication. Figure 3.12 (d) shows that for small multiplications, performance remains good. For large multiplications however (larger than two), performance drops to a point where the system begins to produce a large number of false alarms. Although

this is undesirable, this may not be a problem as long as the condition is met that the vehicle to be recognised has a relatively high intensity compared to its surroundings.

3.4.4 Discussion

A system was described for detection of vehicles in IR imagery. Using shared weight ANNs as feature detectors, a simple post-processing step incorporated the prior knowledge that an image of a vehicle should contain four or more wheels on a single line. The system performs reasonably well. Comparison to other methods is hard, as there is no standard data set for ATR in IR images. Gader et al. [120] claim good results using their shared weight ANN approach, but do not test their system for sensitivity to clutter or scale.

The hypothesis that some form of rejection would be necessary was disproved by the experiments, since ANNs with an RBF layer did not perform significantly better than the other systems. Only in the case without an added hidden layer did the addition of an RBF layer contribute something. Also, the addition of an RBF layer to an ANN having an extra hidden layer does not seem to contribute anything. This can be the result of these ANNs being complex enough to solve the problem as well as possible, but is more likely to be caused by the data set size being insufficient to train these more complex ANNs. The fact that various trained instances of one ANN architecture show great variation in performance also points in this direction.

The ANNs have been tested on sensitivity to scale and seem to be sensitive to differences in small areas around the right scale, yet give responses over a large range of scales. Clutter sensitivity seems to be reasonable on the three images used. With the best ANN, $\text{LeNotre}_2^{\text{H}}$, it is possible to reach a 0% false alarm rate and still recognise 83% of the vehicles present in the testing set.

To improve performance, several steps could be taken:

- significantly enlarge the training set;
- optimise the number of hidden units and/or radial basis units;
- train a bank of ANNs on different types of wheels, or other vehicle features such as windshields or tank barrels, and aggregate their output;
- employ a more intelligent post-processing step.

Visual inspection of the trained ANNs (not shown here) to investigate their feature extraction behaviour again revealed little. In fact, as the signal-to-noise ratio of the images used in training these ANNs is much lower than it was for the handwritten digit images, very little can be deduced from the processing of the input by the feature map layer. Gader et al. [120] inspected trained feature maps and claimed they were "... suggestive

of a diagonal edge detector with a somewhat weak response" and "... of a strong horizontal edge detector with some ability to detect corners as well"; however, these maps can be interpreted to perform any of a number of image processing primitives.

3.5 Conclusions

In this chapter, shared weight ANNs were introduced and applied to two real-world problems. Although the applications showed they can be successfully applied to a range of problems and make good feature extraction mechanisms, very little could yet be learned about the way they function, i.e. what actually makes them good feature extractors.

To gain a better understanding, either the problem will have to be simplified, or the goal of classification will have to be changed. The first idea will be worked out in the next chapter, in which simplified shared weight ANNs will be applied to toy problems. The second idea will be discussed in chapters 5 and 6, in which feed-forward ANNs will be applied to image restoration (regression) instead of feature extraction (classification).

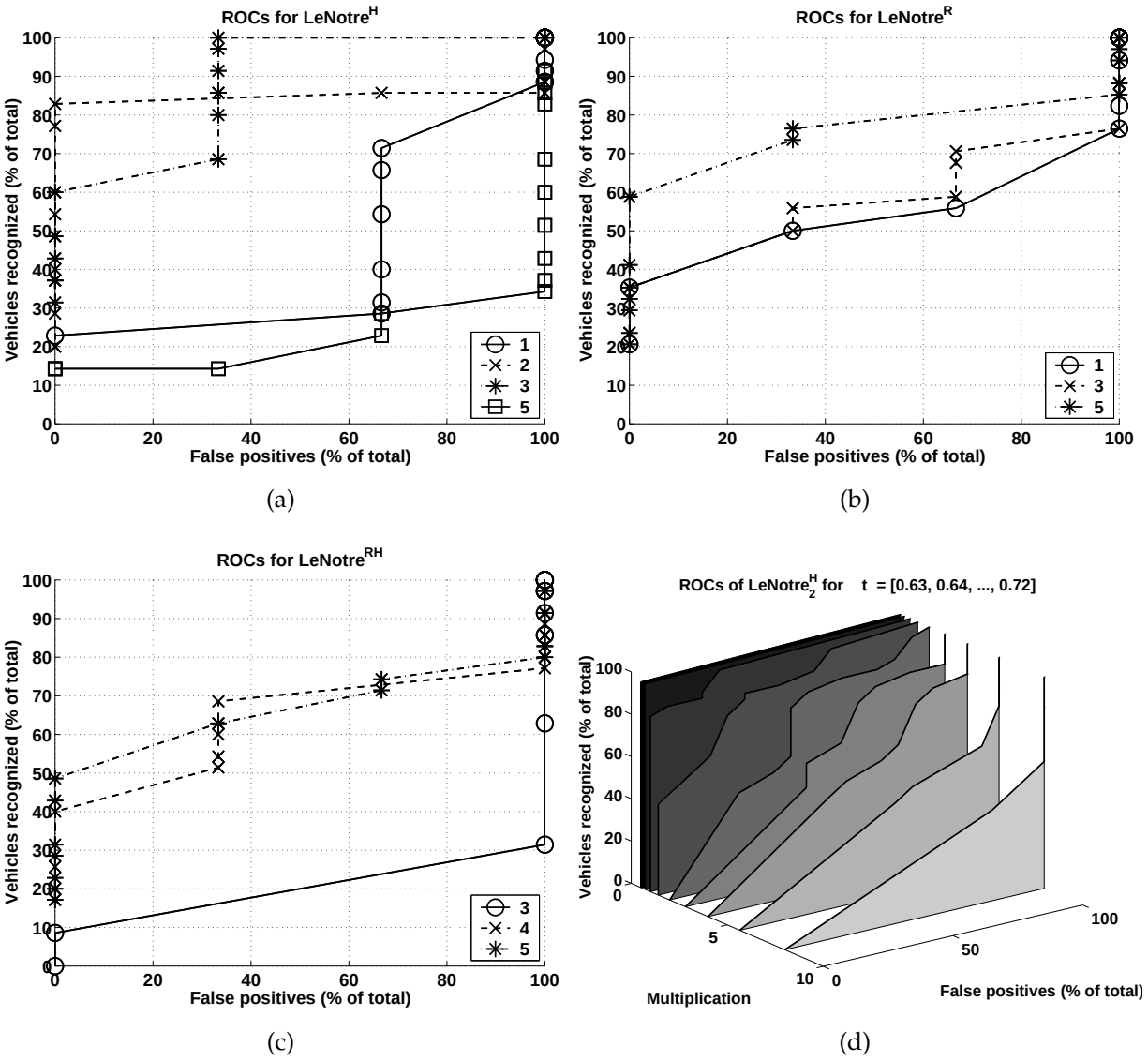


Figure 3.12: (a)-(c) Receiver-operator curves (ROCs) as a function of threshold t for each ANN architecture. (d) Receiver-operator curves for $\text{LeNotre}_2^{\text{H}}$ as a function of threshold t, for various intensity multiplication factors of the background clutter.

4

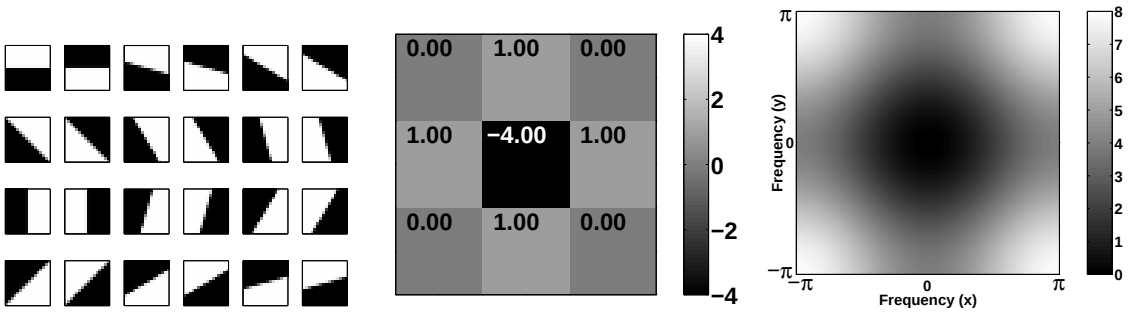
FEATURE EXTRACTION IN SHARED WEIGHT NETWORKS

4.1 Introduction

This chapter investigates whether ANNs, in particular shared weight ANNs, are capable of extracting "good" features from training data. In the previous chapter the criterion for deciding whether features were good was whether traditional classifiers performed better on features extracted by ANNs. Here, the question is whether sense can be made of the extracted features by interpretation of the weight sets found. There is not much literature on this subject, as authors tend to research the way in which ANNs work from their own point of view, as tools to solve specific problems. Gorman and Sejnowski [133] inspect what kind of features are extracted in an ANN trained to recognise sonar profiles. Various other authors have inspected the use of ANNs as feature extraction and selection tools, e.g. [99, 319], compared ANN performance to known image processing techniques [57] or examined decision regions [238]. Some effort has also been invested in extracting (symbolic) rules from trained ANNs [318, 354] and in investigating the biological plausibility of ANNs (e.g. [363]).

An important subject in the experiments presented in this chapter will be the influence of various design and training choices on the performance and feature extraction capabilities of shared weight ANNs. The handwritten digit and vehicle recognition experiments showed that, although the ANNs performed well, the complexity of both the ANN and the data set made visual inspection of trained ANNs impossible. For interpretation therefore it is necessary to bring both data set and ANN complexity down to a bare minimum. Of course, many simple problems can be created [68]; here, two classification problems will be discussed.

In section 4.2, the first problem studied will be that of edge recognition. A data set was constructed containing both edges of varying slopes and uniform regions. First, an optimal architecture and weight set is calculated, based on the Laplacian edge detector.



(a) (b) (c)

Figure 4.1: (a) The edge samples in the edge data set. (b) The Laplacian edge detector. (c) The magnitude of the frequency response of the Laplacian edge detector.

Next, a set of ANN architectures will be trained. In this set the amount of prior knowledge on the problem will be gradually increased and the operation of each ANN will be compared to the hand-crafted solution.

One of the conclusions will be that the artificial nature of the data set causes problems. Therefore, in section 4.3, the second problem discussed will be that of classifying a subset of the NIST database consisting of 10 samples of digit "1" and 10 samples of digit "7". Here, one of the goals is to use more than one receptive field and see which features are extracted by the various receptive fields. A new training algorithm will be introduced in section 4.4 which can aid in interpretation of weight sets.

Finally, section 4.5 will draw some conclusions.

4.2 Edge recognition

In this section, the problem of edge recognition is treated as a classification problem: the goal is to train an ANN to give high output for samples containing edges and low output for samples containing uniform regions. This makes it different from edge detection, in which localisation of the edge in the sample is important as well. A data set was constructed by drawing edges at $0^\circ, 15^\circ, \ldots, 345^\circ$ angles in a 256×256 pixel binary image. These images were rescaled to 16×16 pixels using bilinear interpolation. The pixel values were -1 for background and +1 for the foreground pixels; near the edges, intermediate values occurred due to the interpolation. In total, 24 edge images were created. An equal number of images just containing uniform regions of background (-1) or foreground ($+1$) pixels were then added, giving a total of 48 samples. Figure 4.1 (a) shows the edge samples in the data set.

The goal of this experiment is not to build an edge recogniser performing better than the

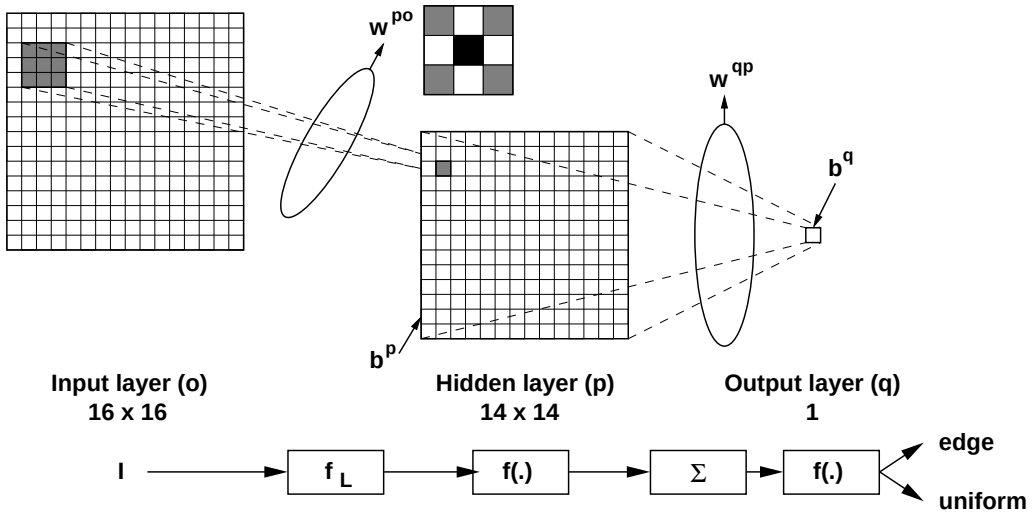


Figure 4.2: A sufficient ANN architecture for edge recognition. Weights and biases for hidden units are indicated by $\mathbf{w}^{po}$ and $\mathbf{b}^{p}$ respectively. These are the same for each unit. Each connection between the hidden layer and the output layer has the same weight $\mathbf{w}^{qp}$ and the output unit has a bias $\mathbf{b}^{q}$. Below the ANN, the image processing operation is shown: convolution with the Laplacian template f_L, pixel-wise application of the sigmoid $f(.)$, (weighted) summation and another application of the sigmoid.

traditional methods; it is to study how an ANN performs edge recognition. Therefore, first a theoretically optimal ANN architecture and weight set will be derived, based on a traditional image processing approach. Next, starting from this architecture, a series of ANNs with an increasing number of restrictions will be trained, based on experimental observations. In each trained ANN, the weights will be investigated and compared to the calculated optimal set.

4.2.1 A sufficient network architecture

To implement edge recognition in a shared weight ANN, it should consist of at least 3 layers (including the input layer). The input layer contains 16×16 units. The 14×14 unit hidden layer will be connected to the input layer through a 3×3 weight receptive field, which should function as an edge recognition template. The hidden layer should then, using bias, shift the high output of a detected edge into the nonlinear part of the transfer function, as a means of thresholding. Finally, a single output unit is needed to sum all outputs of the hidden layer and rescale to the desired training targets. The architecture described here is depicted in figure 4.2.

This approach consists of two different subtasks. First, the image is convolved with a *template* (or filter) which should give some high output values when an edge is present

and low output values overall for uniform regions. Second, the output of this operation is *(soft-)thresholded* and summed, which is a nonlinear neighbourhood operation. A simple summation of the convolved image (which can easily be implemented in a feed-forward ANN) will not do. Since convolution is a linear operation, for any template the sum of a convolved image will be equal to the sum of the input image multiplied by the sum of the template. This means that classification would be based on just the sum of the inputs, which (given the presence of both uniform background and uniform foreground samples, with sums smaller and larger than the sum of an edge image) is not possible. The data set was constructed like this on purpose, to prevent the ANN from finding trivial solutions.

As the goal is to detect edges irrespective of their orientation, a rotation-invariant edge detector template is needed. The first order edge detectors known from image processing literature [282, 385] cannot be combined into one linear rotation-invariant detector. However, the second order Laplacian edge detector can be. The continuous Laplacian,

$$f_L(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{4.1}$$

can be approximated by the discrete linear detector shown in figure 4.1 (b). It is a high-pass filter with a frequency response as shown in figure 4.1 (c). Note that in well-sampled images only frequencies between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ can be expected to occur, so the filters behaviour outside this range is not critical. The resulting image processing operation is shown below the ANN in figure 4.2.

Using the Laplacian template, it is possible to calculate an optimal set of weights for this ANN. Suppose the architecture just described is used, with double sigmoid transfer functions (eqn. 2.4). Reasonable choices for the training targets then are $t = 0.5$ for samples containing an edge and $t = -0.5$ for samples containing uniform regions. Let the 3×3 weight matrix ($\mathbf{w}^{po}$ in figure 4.2) be set to the values specified by the Laplacian filter in figure 4.1 (b). Each element of the bias vector of the units in the hidden layer, $\mathbf{b}^p$, can be set to e.g. $b^p_{opt} = 1.0$.

Given these weight settings, optimal values for the remaining weights can be calculated. Note that since the DC component[1] of the Laplacian filter is zero, the input to the hidden units for samples containing uniform regions will be just the bias, 1.0. As there are 14×14 units in the hidden layer, each having an output of $f(1) \approx 0.4621$, the sum of all outputs O^p will be approximately $196 \cdot 0.4621 = 90.5750$. Here $f(\cdot)$ is the double sigmoid transfer function introduced earlier (eqn. 2.4).

[1] The response of the filter at frequency 0, or equivalently, the scaling in average pixel value in the output image introduced by the filter.

For images that do contain edges, the input to the hidden layer will look like this:

$$\begin{array}{|c|c|c|c|c|c|}\hline -1&-1&-1&-1&-1&-1\\ -1&-1&-1&-1&-1&-1\\ -1&-1&-1&-1&-1&-1\\ 1&1&1&1&1&1\\ 1&1&1&1&1&1\\ 1&1&1&1&1&1\\ \hline\end{array} \otimes \begin{array}{|c|c|c|}\hline 0&1&0\\ 1&-4&1\\ 0&1&0\\ \hline\end{array} = \begin{array}{|c|c|c|c|}\hline 0&0&0&0\\ 2&2&2&2\\ -2&-2&-2&-2\\ 0&0&0&0\\ \hline\end{array} \tag{4.2}$$

There are $14 \times 14 = 196$ units in the hidden layer. Therefore, the sum of the output $\mathbf{O}^p$ of that layer for a horizontal edge will be:

$$\begin{aligned}\sum_i O_i^p &= 14f(2+b_{opt}^p) + 14f(-2+b_{opt}^p) + 168f(b_{opt}^p)\\ &= 14f(3) + 14f(-1) + 168f(1)\\ &\approx 14\cdot 0.9051 + 14\cdot(-0.4621) + 168\cdot 0.4621 = 82.0278\end{aligned} \tag{4.3}$$

These values can be used to find the w_{opt}^{qp} and b_{opt}^q necessary to reach the targets. Using the inverse of the transfer function,

$$f(x) = \frac{2}{1+e^{-x}} - 1 = a \quad \Rightarrow \quad f^{-1}(a) = \ln\left(\frac{1+a}{1-a}\right) = x, \quad a \in \langle -1, 1\rangle \tag{4.4}$$

the input to the output unit, $\mathbf{I}^q = \sum_i O_i^p w_i^{qp} + \mathbf{b}^q = \sum_i O_i^p w_{opt}^{qp} + b_{opt}^q = 0$, should be equal to $f^{-1}(t)$, i.e.:

$$\begin{aligned}\text{edge:} \quad t = \quad 0.5 &\Rightarrow \mathbf{I}^q = \quad 1.0986\\ \text{uniform:} \quad t = -0.5 &\Rightarrow \mathbf{I}^q = -1.0986.\end{aligned} \tag{4.5}$$

This gives:

$$\begin{aligned}\text{edge:} \quad 82.0278\, w_{opt}^{qp} + b_{opt}^q &= \quad 1.0986\\ \text{uniform:} \quad 90.5750\, w_{opt}^{qp} + b_{opt}^q &= -1.0986.\end{aligned} \tag{4.6}$$

Solving these equations gives $w_{opt}^{qp} = -0.2571$ and $b_{opt}^q = 22.1880$.

Note that the bias needed for the output unit is quite high, i.e. far away from the usual weight initialisation range. However, the values calculated here are all interdependent. For example, choosing lower values for $\mathbf{w}^{po}$ and b_{opt}^p will lead to lower required values for w_{opt}^{qp} and b_{opt}^q. This means there is not one single optimal weight set for this ANN architecture, but a range.

4.2.2 Training

Starting from the architecture described in section 4.2.1, a number of ANNs were trained on the data set discussed in section 4.2. The weights and biases of each of these ANNs can be compared to the optimal set of parameters calculated above.

An important observation in all experiments was that as more restrictions were placed on the architecture, it became harder to train. Therefore, in all experiments the conjugate gradient descent (*CGD*, [148, 323]) training algorithm was used. This algorithm is less prone to finding local minima or diverging than back-propagation, as it uses a line minimisation technique to find the optimal step size in each iteration. The method has only one parameter, the number of iterations for which the directions should be kept conjugate to the previous ones. In all experiments, this was set to 10. The CGD algorithm will be discussed in more detail in section 4.4.

Note that the property that makes CGD a good algorithm for avoiding local minima also makes it less fit for ANN interpretation. Standard gradient descent algorithms, such as back-propagation, will take small steps through the error landscape, updating each weight proportionally to its magnitude. CGD, due to the line minimisation involved, can take much larger steps. In general, the danger is overtraining: instead of finding templates or feature detectors that are generally applicable, the weights are adapted too much to the training set at hand. In principle, overtraining could be prevented by using a validation set, as was done in chapter 3. However, here the interest is in what feature detectors are derived from the training set rather than obtaining good generalisation. The goal actually is to adapt to the training data as well as possible. Furthermore, the artificial edge data set was constructed specifically to contain all possible edge orientations, so overtraining cannot occur. Therefore, no validation set was used.

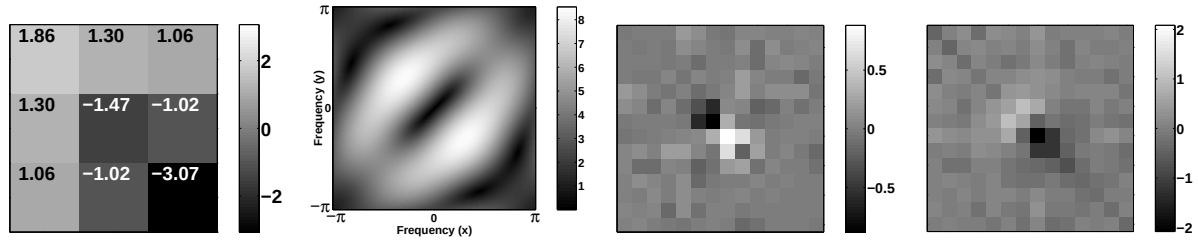
All weights and biases were initialised by setting them to a fixed value of 0.01, except where indicated otherwise[2]. Although one could argue that random initialisation might lead to better results, for interpretation purposes it is best to initialise the weights with small, equal values.

ANN_1: The sufficient architecture

The first ANN used the shared weight mechanism to find $\mathbf{w}^{po}$. The biases of the hidden layer, $\mathbf{b}^p$, and the weights between hidden and output layer, $\mathbf{w}^{qp}$, were not shared. Note that this ANN already is restricted, as receptive fields are used for the hidden layer instead of full connectivity. However, interpreting weight sets of unrestricted, fully connected ANNs is quite hard due to the excessive number of weights – there would be a total of 50,569 weights and biases in such an ANN.

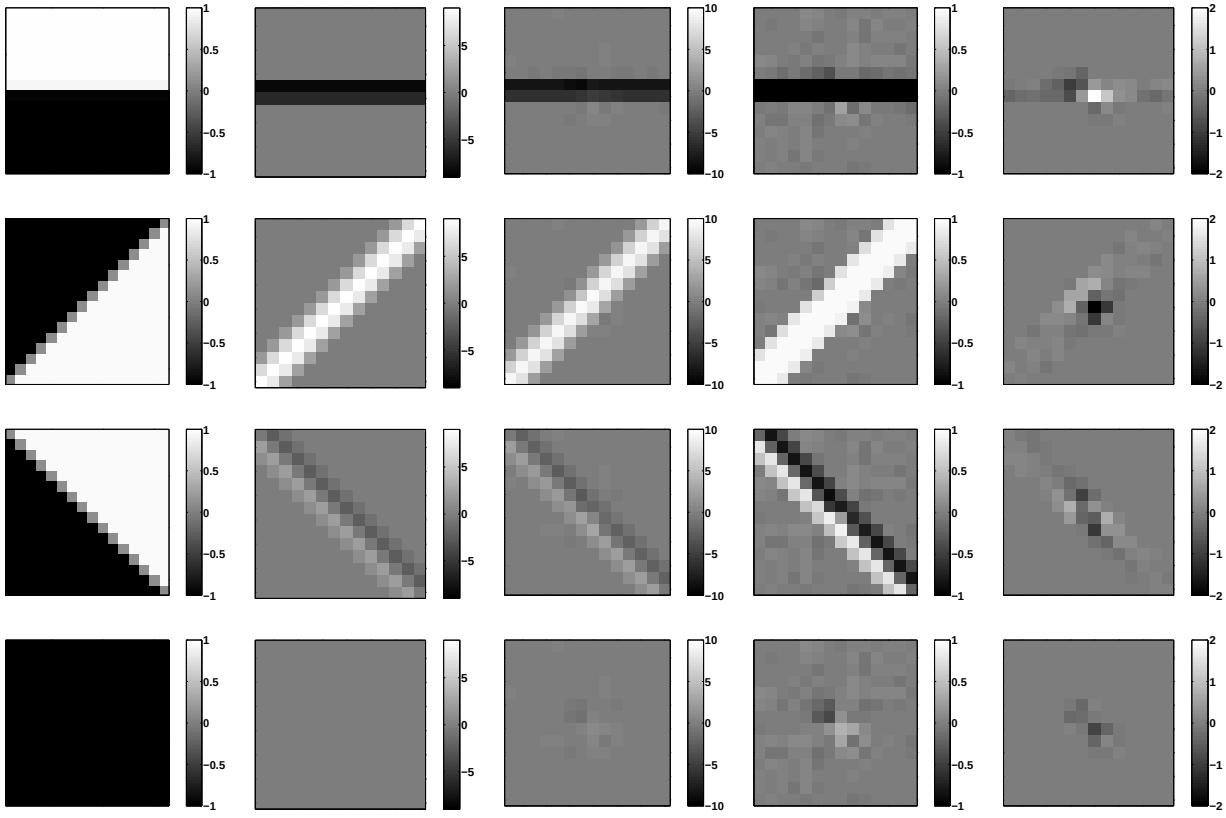
Training this first ANN did not present any problem; the MSE quickly dropped, to 1×10^{-7} after 200 training cycles. However, the template weight set found – shown in figures 4.3 (a) and (b) – does not correspond to a Laplacian filter, but rather to a directed edge detector. The detector does have a zero DC component. Noticeable is the information stored in the bias weights of the hidden layer $\mathbf{b}^p$ (figure 4.3 (c)) and the

[2]Fixed initialisation is possible here because units are not fully connected. In fully connected ANNs, fixed value initialisation would result in all weights staying equal throughout training.



(a) (b) (c) (d)

Figure 4.3: (a) The template and (b) the magnitude of its frequency response, (c) hidden layer bias weights and (c) weights between the hidden layer and output layer, as found in ANN_1.



(a) (b) (c) (d) (e)

Figure 4.4: Stages in ANN_1 processing, for three different input samples: (a) the input sample; (b) the input convolved with the template; (c) the total input to the hidden layer, including bias; (d) the output of the hidden layer and (e) the output of the hidden layer multiplied by the weights between hidden and output layer.

weights between the hidden layer and the output layer, $\mathbf{w}^{qp}$ (figure 4.3 (d)). Note that in figure 4.3 and other figures in this chapter, individual weight values are plotted as grey values. This facilitates interpretation of weight sets as feature detectors. Presentation using grey values is similar to the use of Hinton diagrams [153].

Inspection showed how this ANN solved the problem. In figure 4.4, the different processing steps in ANN classification are shown in detail for three input samples (figure 4.4 (a)). First, the input sample is convolved with the template (figure 4.4 (b)). This gives pixels on and around edges high values, i.e. highly negative (-10.0) or highly positive (+10.0). After addition of the hidden layer bias (figure 4.4 (c)), these values dominate the output. In contrast, for uniform regions the bias itself is the only input of the hidden hidden layer units, with values approximately in the range $[-1, 1]$. The result of application of the transfer function (figure 4.4 (d)) is that edges are widened, i.e. they become bars of pixels with values +1.0 or -1.0. For uniform regions, the output contains just the two pixels diagonally opposite at the centre, with significantly smaller values.

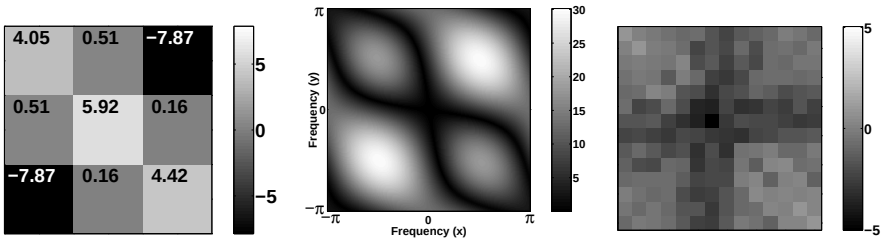
The most important region in these outputs is the centre. Multiplying this region by the diagonal +/- weights in the centre and summing gives a very small input to the output unit (figure 4.4 (e)); in other words, the weights cancel the input. In contrast, as the diagonal -/+ pair of pixels obtained for uniform samples is multiplied by a diagonal pair of weights of the opposite sign, the input to the output unit will be negative. Finally, the bias of the output unit (not shown) shifts the input in order to obtain the desired target values $t = 0.5$ and $t = -0.5$.

This analysis shows that the weight set found is quite different from the optimal one calculated in section 4.2.1. As all edges pass through the centre of the image, the edge detector need not be translation-invariant: information on where edges occur is coded in both the hidden layer bias and the weights between the hidden layer and the output layer.

ANN_2: Sharing more weights

To prevent the ANN from coding place-specific information in biases and weights, the architecture will have to be simplified further. As a restriction, in the next ANN architecture the weights between the hidden layer and output layer were shared. That is, there was one single weight shared among all 196 connections between the hidden units and the output unit. Training took more time, but still converged to a 1×10^{-6} MSE after 2,400 cycles. Still, the network does not find a Laplacian; however, the template found (figure 4.5 (a) and (b)) has a more clear function than the one found before. It is a strong detector for edges with slope $-45°$, and a weak detector for edges with slope $45°$.

In the bias weights of the hidden layer (figure 4.5 (c)), place-specific information is now stored for edges which are not amplified well by this detector. Bias weight values are



(a) (b) (c)

Figure 4.5: (a) The template, (b) the magnitude of its frequency response and (b) hidden layer bias weights as found in ANN_2.

also significantly higher than before (an average of -1.2144). This allows the ANN to use the transfer function as a threshold operation, by scaling large positive pixel values differently from large negative pixel values. In conclusion, responsibility for edge recognition is now shared between the template and the bias weights of the hidden layer.

ANN_3: Sharing bias

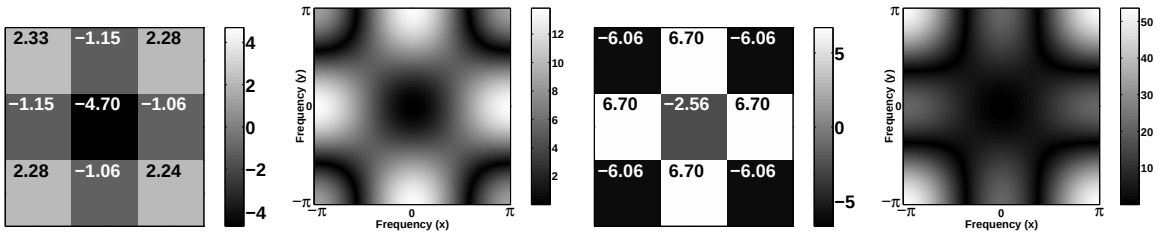
As the biases of hidden layer units are still used for storing place-dependent information, in the next architecture these biases were shared too[3]. Training became even harder; the ANN would not converge using the initialisation used before, so weights were initialised to a fixed value of 0.1. After 1,000 episodes, the MSE reached 8×10^{-4}, just slightly higher than the minimal error possible (at 3×10^{-4}, larger than zero due to the interpolation used in scaling the edge samples). The template found is shown in figures 4.6 (a) and (b).

Note that the template now looks like a Laplacian edge detector; its frequency response is similar to that of the Laplacian in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. However, there are still small differences between various weights which are equal in the true Laplacian. In fact, the filter seems to be slightly tilted, with the top left corner containing weights with higher magnitude. Also, the frequency response shows that the filter gives a bandpass response in diagonal directions. To obtain a more Laplacian-like template, further restrictions will have to be placed on the ANN.

ANN_4: Enforcing symmetry

In the last ANN, the prior knowledge that the goal is to obtain a rotation-invariant filter was used as well, by sharing weights in the filter itself. The *mask* used for this purpose

[3]Sharing biases would have required a major rewrite of the simulation package used, SPRLIB/ANNLIB [155]. Therefore, biases were shared by replacing all biases by their average after each training cycle.



(a) (b) (c) (d)

Figure 4.6: (a) The template found in ANN_3 and (b) the magnitude of its frequency response. (c) The template found in ANN_4 and (d) the magnitude of its frequency response.

was:

A	B	A
B	C	B
A	B	A

(4.7)

i.e. connections with identical mask letters used shared weights. Note that in this ANN there are only 6 free parameters left: the three weights in the mask, a bias weight for both the hidden and output layer and one weight between the hidden and output layer.

Training was again more cumbersome, but after initialising weights with a fixed value of 0.1 the ANN converged after 1,000 episodes to an MSE of 3×10^{-4}. The filter found is shown in figures 4.6 (c) and (d). Finally, a solution similar to the optimal one was found: its frequency response is like that of the Laplacian in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and the weights are symmetric.

4.2.3 Discussion

The experiments described in this section show that ANNs can be used as edge detectors. However, the presence of receptive fields in the architecture in itself does not guarantee that shift-invariant feature detectors will be found, as claimed by some [208, 209, 364]. Also, the mere fact that performance is good (i.e., the MSE is low) does not imply that such a feature extraction process is used. An important observation in ANN_1 and ANN_2 was that the ANN will use weights and biases in later layers to store place-dependent information. In such a network, where edge positions are stored, in principle any template will suffice. Obviously, this makes interpretation of these templates dubious: different observers may find the ANN has learned different templates.

One reason for the ease with which ANNs store place-dependent information might be the relative simplicity of the dataset: the fact that edges all passed through the centre of the image makes this possible. Therefore, in the next section similar ANNs will be trained on a real-world dataset.

When the ANNs were further restricted by sharing biases and other weights (ANN_3), convergence became a problem. The explanation for this is that the optimal weight set is rather special in ANN terms, as the template has to have a zero DC component (i.e., its weights have to add up to zero). Although this seems to be a trivial demand, it has quite large consequences for ANN training. Optimal solutions correspond to a range of inter-dependent weights, which will result in long, narrow valleys in the MSE "landscape". A small perturbation in one of the template weights will have large consequences for the MSE. Simple gradient descent algorithms such as back-propagation will fail to find these valleys, so the line-optimisation step used by CGD becomes crucial.

The last ANN, ANN_4, was able to find an edge detector very similar to the Laplacian. However, this architecture was restricted to such an extent that it can hardly be seen as representative for practical application of ANNs. This indicates there is a trade-off between complexity and the extent to which experiments are true-to-life on the one hand, and the possibility of interpretation on the other. This effect might be referred to as a kind of ANN *interpretability trade-off*[4]. If an unrestricted ANN is trained on a real-world data set, the setup most closely resembles the application of ANNs in everyday practice. However, the subtleties of the data set and the many degrees of freedom in the ANN prevent gaining a deeper insight into the operation of the ANN. On the other side, once an ANN is restrained, e.g. by sharing or removing weights, lowering the number of degrees of freedom or constructing architectures only specifically applicable to the problem at hand, the situation is no longer a typical one. The ANN may even become too constrained to learn the task at hand. The same holds for editing a data set to influence its statistics or to enhance more preferable features with regard to ANN training, which will be discussed in chapter 6.

4.3 Two-class handwritten digit classification

The experiments described in this section address the problem raised in section 4.2.3, i.e. that the data set used in the previous experiments may have been too simple. To construct a more real-life dataset while still maintaining the expectation that weights can be interpreted, a small NIST subset was used. This subset consisted of 10 samples each of the classes "1" and "7", shown in figure 4.7. The 16×16 pixel values were scaled linearly between -1.0 (background) and 1.0 (foreground). Training targets were set to $t = 0.5$ for class "1" and $t = -0.5$ for class "7".

For this problem, it is already impossible to find an architecture and weight set by hand which will give minimal error. The receptive fields in the ANNs are expected to act as

[4]Note that this is not precisely the same issue as addressed by the bias-variance trade-off (see page 12), which is concerned with the relation between model complexity and error. The concern here is with the specificity of the model with respect to interpretation which, in principle, is unrelated to complexity: making a model more specific need not introduce a bias.

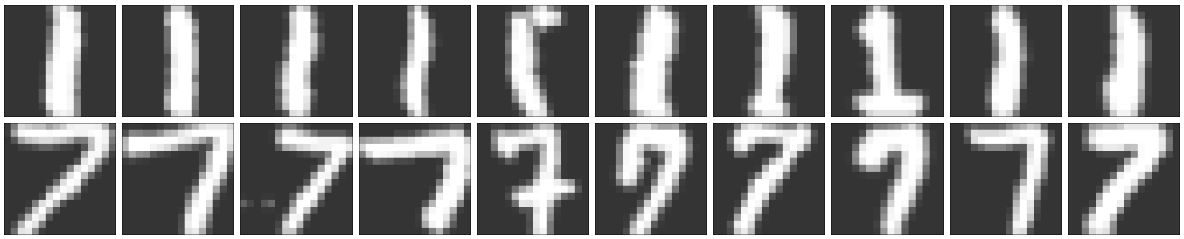


Figure 4.7: The two-class handwritten digit data set.

feature detectors, extracting characteristic shapes from the data. Beforehand, it is quite hard to indicate by hand which weight sets will detect the most salient features. However, as the width of the strokes in the digit images lies in the range $3-5$ pixels, feature detectors should have widths and heights roughly in the range $3-7$ pixels.

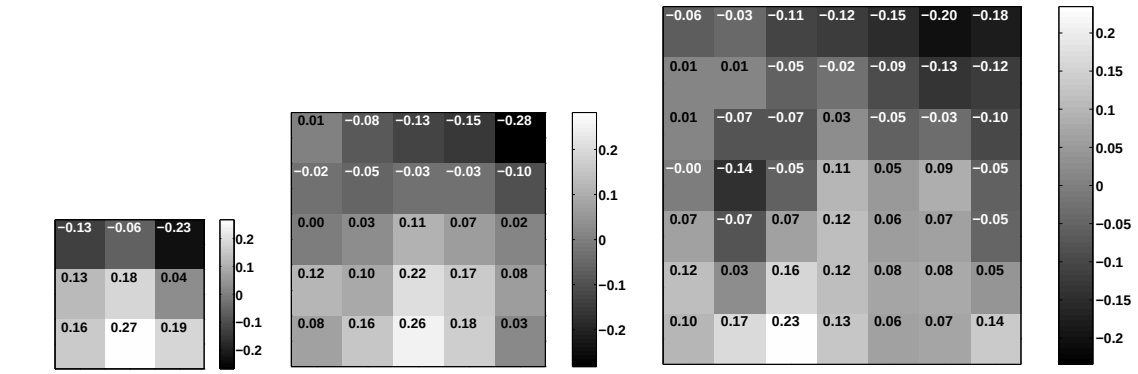
The starting point therefore will be the ANN used for edge recognition, shown in figure 4.2. However, three different architectures will be used. The first has a 3×3 pixel receptive field and $14 \times 14 = 196$ units in the hidden layer, the second contains a 5×5 pixel receptive field and $12 \times 12 = 144$ hidden units and the last contains a 7×7 pixel receptive field and $10 \times 10 = 100$ hidden units. As for this data set it is to be expected that using more than one feature map will increase performance, architectures using two feature maps were trained as well. In this case, the number of hidden units doubles.

4.3.1 Training

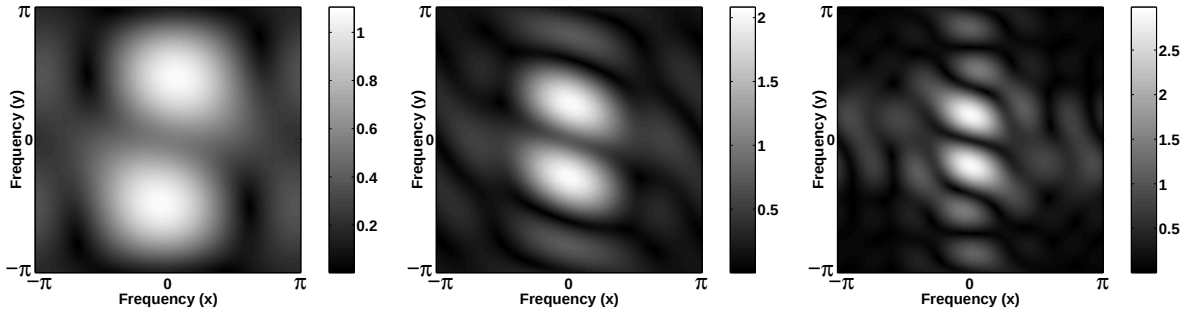
Most ANNs were rather hard to train, again due to the restrictions placed on the architecture. CGD was used with 10 steps during which directions were kept conjugate. All ANN weights and biases were initialised using a fixed value of 0.01, except where indicated otherwise. For most restricted architectures, reaching an MSE of exactly 0 proved to be impossible. Therefore, training was stopped when the MSE reached a sufficiently low value, 1.0×10^{-6}.

$\mathbf{ANN_1}$: Unrestricted

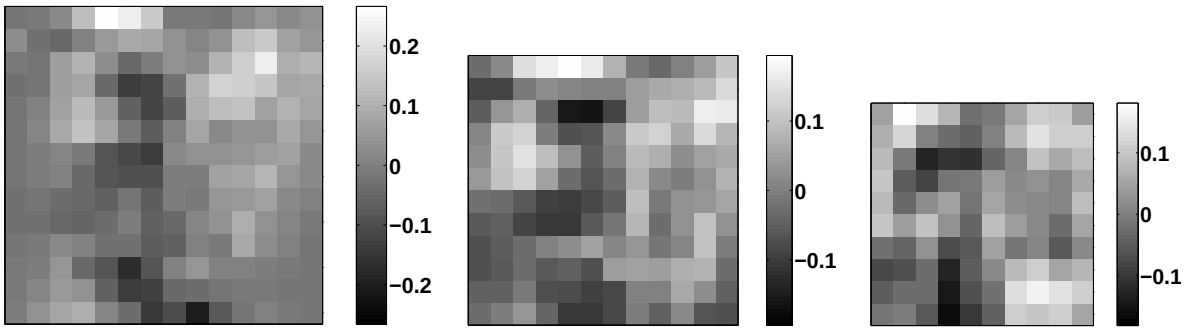
The first ANNs were identical to the one shown in figure 4.2, except for the fact that three different ANNs were trained with 3×3 ($\mathrm{ANN}_1^{3\times3}$), 5×5 ($\mathrm{ANN}_1^{5\times5}$) and 7×7 ($\mathrm{ANN}_1^{7\times7}$) pixel receptive fields, respectively. These ANNs quickly converged to a nearly zero MSE: after 250 training cycles, the MSE was in the order of 1×10^{-10}. The feature detectors found, shown in figure 4.8 (a), are not very clear however. The frequency responses (figure 4.8 (b)) give more information. The filters most closely resemble horizontal edge detectors: note the basic shape returning for the three sizes of



(a)



(b)



(c)

Figure 4.8: (a) Feature detectors found in the receptive fields of $\text{ANN}_1^{3\times3}$, $\text{ANN}_1^{5\times5}$ and $\text{ANN}_1^{7\times7}$. (b) The corresponding frequency response magnitudes. (c) Weights between hidden layer and output layer.

feature detector.

As was the case in the edge recognition ANNs, the weights between the hidden layer and the output unit have been used to store positions of the digits. Figure 4.8 (c) illustrates this. Positive weights indicate pixel positions where typically only class "7" samples have high values; negative weights indicate positions where class "1" is present. Although noisy, these same basic shapes are present for each size of the receptive field.

In contrast to what was found for the edge recognition ANNs, the bias weights in the second layer were not used heavily. Bias values fell roughly in the range $\left[-2 \times 10^{-2}, 2 \times 10^{-2}\right]$, i.e. negligible in comparison to feature detector weight values.

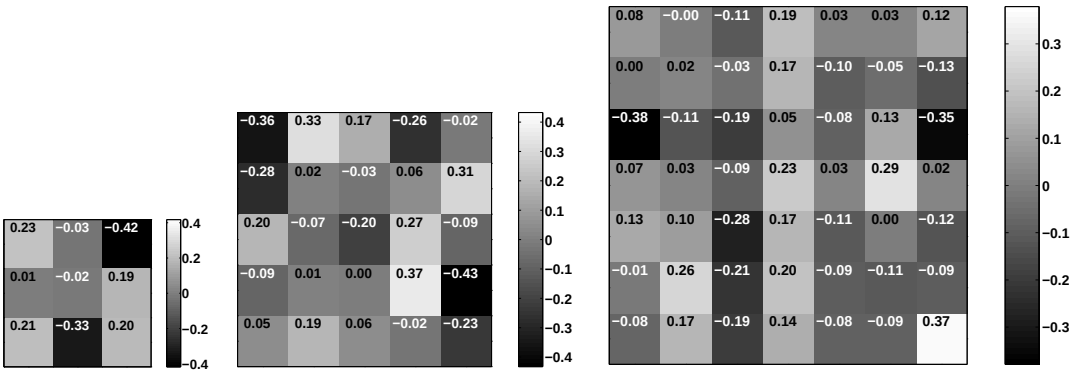
ANN_2: Fully restricted

In the next architecture, the number of weights was restricted by sharing weights between hidden layer and output layer and by sharing the bias weights in the second layer (i.e., the basic architecture was the same as ANN_3 for edge recognition, on page 71). As a consequence, there were far fewer parameters left in the ANNs: the number of weights in the feature detector plus two biases and one weight between hidden and output layer.

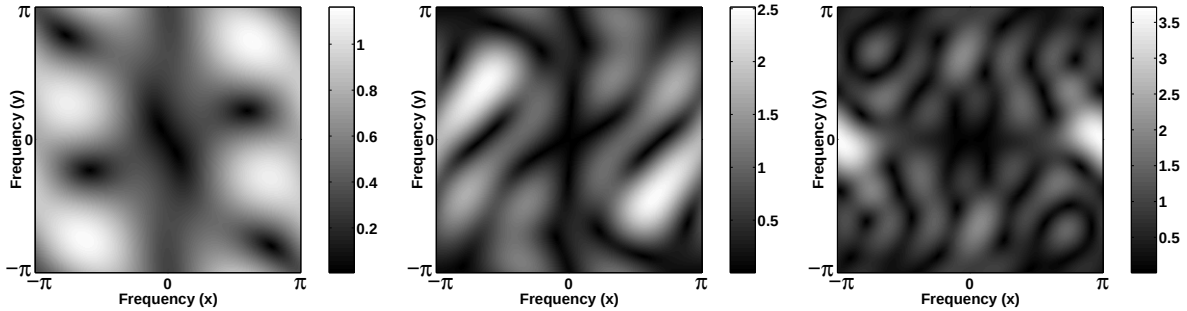
Training became quite a bit harder. It did not converge for the ANN with the 3×3 pixel receptive field; the MSE oscillated around 1.5×10^{-2}. For the other two ANNs, training was stopped when the MSE fell below 1×10^{-6}, which took 2000 cycles for the 5×5 pixel receptive field ANN and 1450 cycles for the 7×7 pixel receptive field ANN.

The feature detectors found are shown in figure 4.9. Note that since the 3×3 receptive field ANN did not converge, the resulting filter cannot be interpreted. Since the weights between hidden layer and output layer can no longer be used, the feature detectors of the other two look rather different. The 5×5 pixel feature detector is the most pronounced: it is a detector of 3-pixel wide bars with a slope of 45°. Evidence for this can also be found by inspecting the output of the hidden layer for various inputs, as shown in figure 4.10. In the location of the stem of the "7"s, output values are much higher than those in the location of the stem of the "1"s. Finally, the function of the 7×7 pixel feature detector is unclear.

From these results, it is clear that a feature detector size of 3×3 pixels is too small. On the other hand, although the 7×7 pixel feature detectors gives good performance, they cannot be interpreted well. The 5×5 pixel feature detector seems to be optimal. Therefore, from here on only 5×5 pixel feature detectors will be considered.



(a)



(b)

Figure 4.9: (a) The feature detectors found in the receptive fields of $\text{ANN}_2^{3\times3}$, $\text{ANN}_2^{5\times5}$ and $\text{ANN}_2^{7\times7}$. (b) The corresponding frequency response magnitudes.

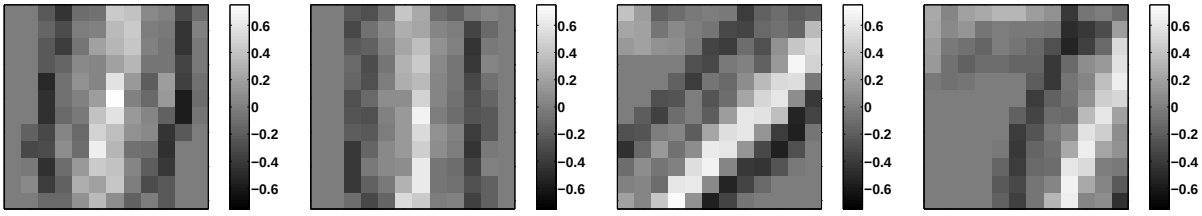


Figure 4.10: The output of the hidden layer of $\text{ANN}_2^{5\times5}$, for two samples of class "1" (left) and two samples of class "7" (right).

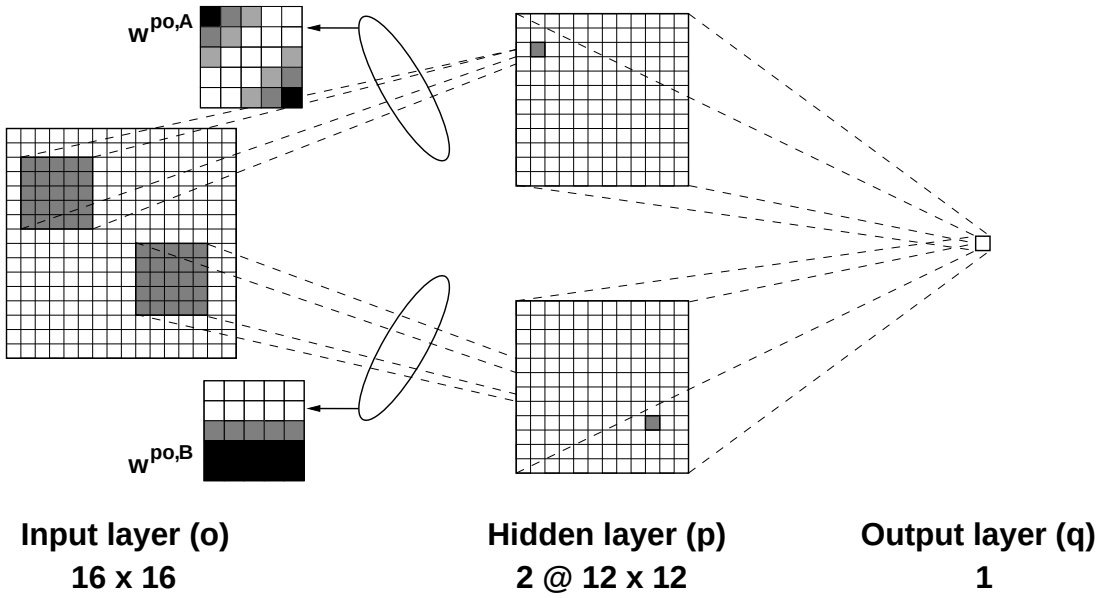
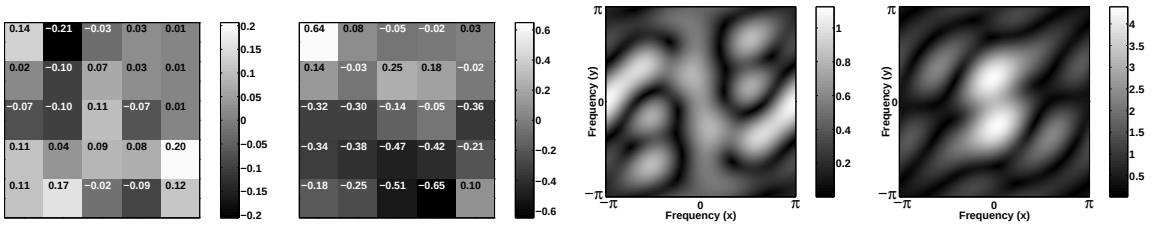


Figure 4.11: $\text{ANN}_3^{5\times5}$, with two 5×5 pixel feature detectors. Biases and weights between the hidden layer and output layer have not been indicated.

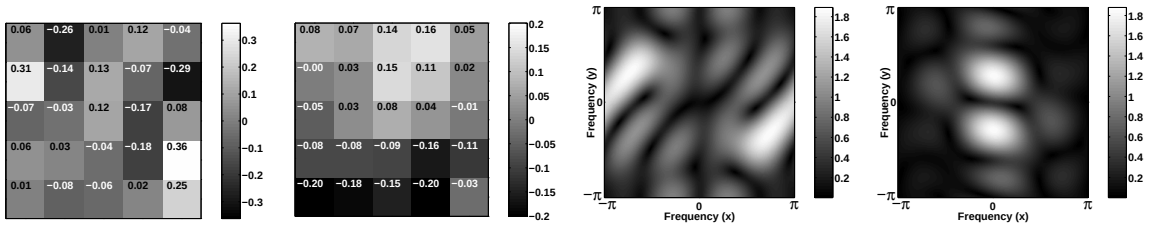
ANN_3: Two feature maps

Although the frequency response of the 5×5 pixel feature detector is clearer than the others, the filter itself is still noisy, i.e. neighbouring weights have quite different values. There is no clear global feature (within a 5×5 pixel region) that corresponds to this detector. The reason for this might be that in fact several features are detected (either amplified or attenuated) using this one set of weights. Therefore, ANN_3 contained two feature maps instead of one. In all other respects, the ANN was the same as ANN_2, as shown in figure 4.11.

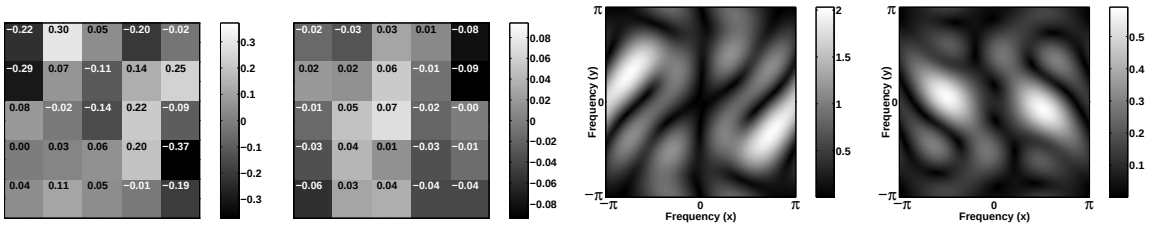
If this ANN is initialised using a fixed value, the two feature detectors will always remain identical, as each corresponding weight in the two detectors is equally responsible for the error the ANN makes. Therefore, random initialisation is necessary. This frustrates interpretation, as different initialisations will lead to different final weight sets. To illustrate this, four ANNs were trained in which weights were initialised using values drawn from a uniform distribution with range $[-0.01, 0.01]$. Figure 4.12 shows four resulting template pairs. The feature detector found before in $\text{ANN}_2^{5\times5}$ (figure 4.9) often returns as one of the two feature maps. The other feature detector however shows far more variation. The instantiation in the second row of figure 4.12 (b) looks like the horizontal edge detector found in ANN_1 (figures 4.8 (a), (b)), especially when looking at its frequency response (in the fourth column). However, in other ANNs this shape does not return. The first and fourth ANN indicate that actually multiple feature detectors may be distributed over the two feature maps.



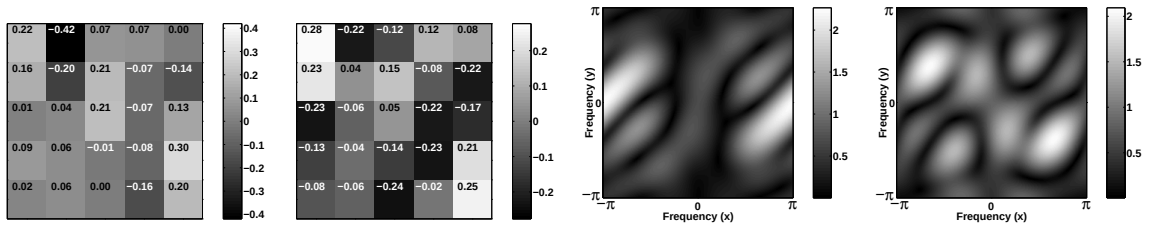
(a)



(b)



(c)



(d)

Figure 4.12: Feature detector pairs found in ANN_3, for four different random weight initialisations ((a)-(d)).

To allow inspection of weights, initialisation with fixed values seems to be a prerequisite. To allow this, the training algorithm itself should allow initially identical weights to take on different values during training. The next section will introduce a training algorithm developed specifically for training ANNs with the goal of weight interpretation.

4.4 Decorrelating conjugate gradient descent

The last experiment on the NIST subset showed that interpretation of ANNs with multiple feature detectors is difficult. The main causes are the random weight initialisation required and a tendency of the ANNs to distribute features to be detected in a non-obvious way over receptive fields. To address the latter problem, hidden units learning identical functions, a modular approach has been suggested [180]. However, this is not applicable in cases in which there is no clear decomposition of a task's input space into several domains.

To allow fixed value weight initialisation and still obtain succinct feature detectors, a new training algorithm will be proposed. The algorithm is based on CGD, but has as a soft constraint the minimisation of the squared covariance between receptive fields. In this way, the symmetry between feature detectors due to fixed value initialisation can be broken, and receptive field weight sets are forced to become orthogonal while still minimising the ANN's MSE. First, the philosophy behind the training algorithm will be discussed in section 4.4.1. Next, the algorithm itself is given in section 4.4.2. Finally, experimental results on the small digit data set are discussed in section 4.4.3.

4.4.1 Decorrelation

Note that, in trained ANNs, weight sets belonging to different receptive fields need not be exactly the same for the feature maps to perform the same function. This is because weights are interdependent, as was already noted in section 4.2.1. As an example, consider the weight vectors $\mathbf{w}^{po,A}$ and $\mathbf{w}^{po,B}$ (from here on, $\mathbf{w}^A$ and $\mathbf{w}^B$) in $\text{ANN}_3^{5\times5}$ (figure 4.11). As long as $\mathbf{w}^A = c_1\mathbf{w}^B + c_2$, biases in the hidden and output layer and the weights between these layers can correct the differences between the two weight sets, and their functionality can be approximately identical[5]. The conclusion is that to compare weight sets, one has to look at their correlation.

Suppose that for a certain layer in an ANN (as in figure 4.11) there are two incoming weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$, both with $K > 2$ elements and $\text{var}(\mathbf{w}^A) > 0$ and $\text{var}(\mathbf{w}^B) >$

[5]Up to a point, naturally, due to the nonlinearity of the transfer functions in the hidden and output layer. For this discussion it is assumed the network operates in that part of the transfer function which is still reasonably linear.

0. The correlation coefficient C between these vectors can be calculated as:

$$C(\mathbf{w}^A, \mathbf{w}^B) = \frac{\text{cov}(\mathbf{w}^A, \mathbf{w}^B)}{\sqrt{\text{var}(\mathbf{w}^A)\text{var}(\mathbf{w}^B)}} \tag{4.8}$$

The correlation coefficient $C(\mathbf{w}^A, \mathbf{w}^B)$ is a number in the range $[-1, 1]$. For $C(\mathbf{w}^A, \mathbf{w}^B) = \pm 1$, there is a strong correlation; for $C(\mathbf{w}^A, \mathbf{w}^B) = 0$ there is no correlation. Therefore, the squared correlation $C(\mathbf{w}^A, \mathbf{w}^B)^2$ can be minimised to minimise the likeness of the two weight sets.

Although this seems a natural thing to do, a problem is that squared correlation can be minimised either by minimising the squared covariance or by maximising the variance of either weight vector. The latter is undesirable, as for interpretation the variance of one of the weight vectors should not be unnecessarily increased just to lower the squared correlation. Ideally, both weight vectors should have comparable variance. Therefore, a better measure to minimise is just the squared covariance. To do this, the derivative of the covariance w.r.t. a single weight $\mathbf{w}_i^A$ has to be computed:

$$\begin{aligned} \frac{\partial \text{cov}(\mathbf{w}^A, \mathbf{w}^B)^2}{\partial \mathbf{w}_i^A} &= \frac{\partial}{\partial \mathbf{w}_i^A} \left(\frac{1}{K} \sum_{k=1}^{K} (\mathbf{w}_k^A - \overline{\mathbf{w}}^A)(\mathbf{w}_k^B - \overline{\mathbf{w}}^B) \right)^2 \\ &= \frac{2}{K} \text{cov}(\mathbf{w}^A, \mathbf{w}^B)(\mathbf{w}_i^B - \overline{\mathbf{w}}^B) \end{aligned} \tag{4.9}$$

This derivative can then be used in combination with the derivative of the MSE w.r.t. the weights to obtain a training algorithm minimising both MSE and squared covariance (and therefore squared correlation, because the variance of the weight vectors will remain bounded since the ANN still has to minimise the MSE).

Correlation has been used before in neural network training. In the cascade correlation algorithm [101], it is used as a tool to find an optimal number of hidden units by taking the correlation between a hidden unit's output and the error criterion into account. However, it has not yet been applied on weights themselves, to force hidden units to learn different functions during training.

4.4.2 A decorrelating training algorithm

Squared covariance minimisation was incorporated into the CGD method used before. The pseudo code of the original CGD algorithm is given in figure 4.13. Note that the derivative of the error function E to be minimised (usually the MSE) is used only to update $\mathbf{g}$, and the function E itself only in the line minimisation [155, 284].

The squared covariance term was integrated into the derivative of the error function as an additive criterion, as in weight regularisation [28]. A problem is how the added term should be weighted (cf. choosing the regularisation parameter). The MSE can

function $\mathbf{w} := \mathit{ConjugateGradientDescent}(\mu, \sigma, N)$
  $\mathbf{w}(0) := \mathit{UniformRandom}(\mu - \sigma, \mu + \sigma)$ *Uniform distribution, offset μ, range $[-\sigma, \sigma]$*
  $\mathbf{d} := \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}(0))$ *Derivative of MSE*
  $\mathbf{g}(0) := \mathbf{h}(0) := -\mathbf{d}$
  for $t := 1$ **to** N **do**
    $\mathbf{w}(t) := \mathit{LineMinimise}(E, \mathbf{w}(t-1), \mathbf{h}(t-1))$
      Minimise error E from point $\mathbf{w}(t-1)$ along direction $\mathbf{h}(t-1)$
    $\mathbf{d} := \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}(t))$ *Derivative of MSE*
    $\mathbf{g}(t) := -\mathbf{d}$ *Gradient*
    $\gamma := \frac{(\mathbf{g}(t)-\mathbf{g}(t-1))^T \mathbf{g}(t)}{\mathbf{g}(t-1)^T \mathbf{g}(t-1)}$ *Polak-Ribiere update rule*
    $\mathbf{h}(t) := \mathbf{g}(t) + \gamma \mathbf{h}(t-1)$ *New direction*
  od
end

Figure 4.13: The conjugate gradient descent (CGD) algorithm. The parameters, μ, σ and N, determine the offset and range of the uniform distribution from which $\mathbf{w}$ is initialised and the number of directions kept conjugate, respectively.

start very high but usually drops rapidly. The squared covariance part also falls in the range $[0, \infty\rangle$, but it may well be the case that it cannot be completely brought down to zero, or only at a significant cost to the error. The latter effect should be avoided: the main training goal is to reach an optimal solution in the MSE sense. Therefore, the covariance information is used in the derivative function *only*, to determine the direction in which steps are taken. It is not used in the absolute minimisation function *LineMinimise*. Furthermore, the squared covariance gradient, $\mathbf{d}^{\text{cov}}$, is normalised to the length of the ordinary gradient $\mathbf{d}^E$ (i.e. just its direction is used) and weighed with a factor λ. The pseudo code of the adapted algorithm, called the decorrelating conjugate gradient descent (DCGD) method, is given in figure 4.14.

Note that the derivative of the squared covariance is only calculated once for each pair of weight sets and attributed to only one of the weight sets. This allows one weight set to learn a globally optimal function, while the second set is trained to both lower the error and avoid covariance with the first set. It also allows initialisation with fixed values, since the asymmetrical contribution of the squared covariance term provides a symmetry breaking mechanism (which can even improve performance in some classification problems, see [71]). However, the outcome of the DCGD training process is still dependent on the choice of a number of parameters. DCGD even introduces a new one (the weight factor λ). If the parameters are chosen poorly, one will still not obtain understandable feature detectors. This is a problem of ANNs in general, which cannot be solved easily: a certain amount of operator skill in training ANNs is a prerequisite for obtaining good results. Furthermore, experiments with DCGD are reproducable due to the possibility of weight initialisation with fixed values.

function $\mathbf{w} := \mathit{DecorrelatingConjugateGradientDescent}(\mu, \sigma, \lambda, N)$
  $\mathbf{w}(0) := \mathit{UniformRandom}(\mu - \sigma, \mu + \sigma)$ *Uniform distribution, offset μ, range $[-\sigma, \sigma]$*
  $\mathbf{d}^E := \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}(0))$ *Derivative of MSE*
  $\mathbf{d}^{\text{cov}} := \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \frac{\partial}{\partial \mathbf{w}} C(\mathbf{w}^k(0), \mathbf{w}^l(0))^2$ *Derivative of squared covariance*
  $\mathbf{g}(0) := \mathbf{h}(0) := -(\mathbf{d}^E + \lambda \frac{||\mathbf{d}^E||}{||\mathbf{d}^{\text{cov}}||} \mathbf{d}^{\text{cov}})$ *Normalised sum of both derivatives*
  for $t := 1$ **to** N **do**
    $\mathbf{w}(t) := \mathit{LineMinimise}(E, \mathbf{w}(t-1), \mathbf{h}(t-1))$
      Minimise error E from point $\mathbf{w}(t-1)$ along direction $\mathbf{h}(t-1)$
    $\mathbf{d}^E := \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}(t))$ *Derivative of MSE*
    $\mathbf{d}^{\text{cov}} := \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \frac{\partial}{\partial \mathbf{w}} C(\mathbf{w}^k(t), \mathbf{w}^l(t))^2$ *Derivative of squared covariance*
    $\mathbf{g}(t) := -(\mathbf{d}^E + \lambda \frac{||\mathbf{d}^E||}{||\mathbf{d}^{\text{cov}}||} \mathbf{d}^{\text{cov}})$ *Gradient: normalised sum of derivatives*
    $\gamma := \frac{(\mathbf{g}(t) - \mathbf{g}(t-1))^T \mathbf{g}(t)}{\mathbf{g}(t-1)^T \mathbf{g}(t-1)}$ *Polak-Ribiere update rule*
    $\mathbf{h}(t) := \mathbf{g}(t) + \gamma \mathbf{h}(t-1)$ *New direction*
  od
end

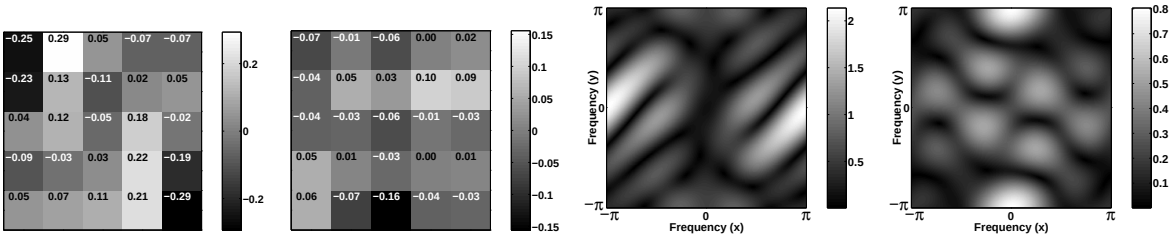
Figure 4.14: The decorrelating conjugate gradient descent (DCGD) algorithm. The parameters μ and σ again control initialisation, λ determines the relative weight of the derivative of the squared covariance and N determines the number of directions kept conjugate.

The DCGD algorithm is computationally expensive, as it takes covariances between all pairs of receptive fields into account. Due to this $\mathcal{O}(n^2)$ complexity in the number of receptive fields, application of this technique to large ANNs is not feasible. A possible way to solve this problem would be to take only a subset of covariances into account.

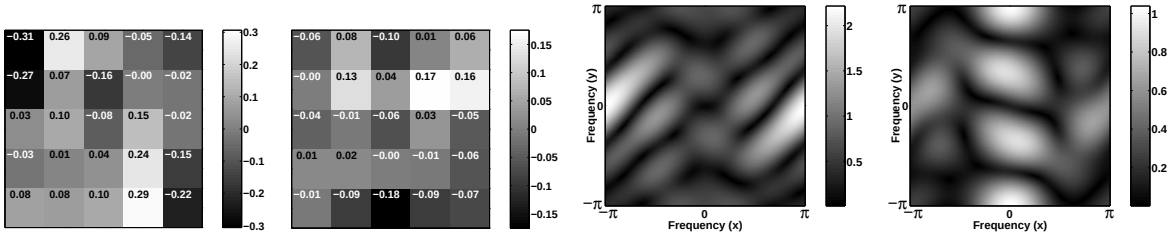
4.4.3 Training $\text{ANN}_3^{5\times5}$ using DCGD

The ANN trained using CGD in section 4.3.1, $\text{ANN}_3^{5\times5}$, was trained using DCGD. Weights and biases were initialised to a fixed value of 0.01 (i.e. $\mu = 0.01$, $\sigma = 0.0$) and $N = 10$ directions were kept conjugate at a time. The only parameter varied was the weighting factor of the squared covariance gradient, λ, which was set to 0.5, 1, 2 and 5. Training converged but was slow. The MSE eventually reached the values obtained using CGD (1.0×10^{-6}, cf. section 4.3.1); however, DCGD training was stopped when the MSE reached about 1.0×10^{-5}, after about 500-1000 cycles, to prevent overtraining. In all cases, classification was perfect.

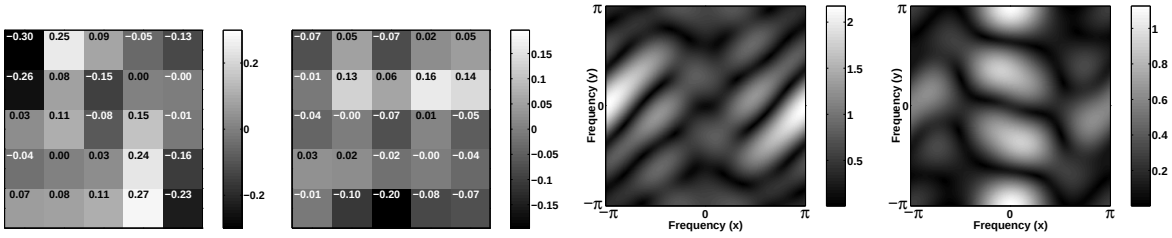
Figure 4.15 shows the feature detectors found in $\text{ANN}_3^{5\times5}$ trained using DCGD. Squared correlations C^2 between them are very small, showing that the minimisation was succesful (the squared covariance was, in all cases, 0). For $\lambda = 1$ and $\lambda = 2$, the feature



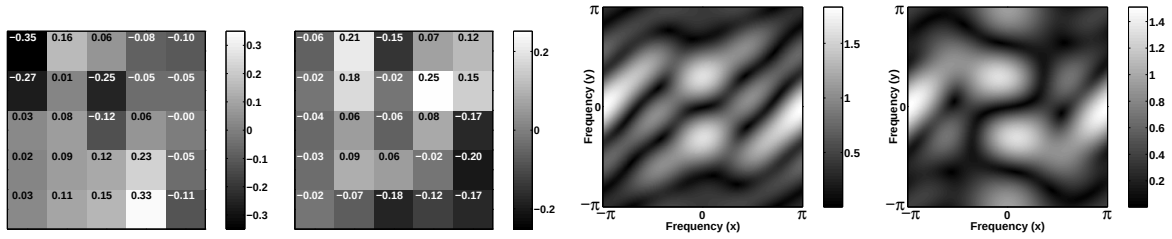
(a) $\lambda = 0.5$: $C^2 = 1.1 \times 10^{-4}$



(b) $\lambda = 1$: $C^2 = 1.7 \times 10^{-6}$



(c) $\lambda = 2$: $C^2 = 1.0 \times 10^{-6}$



(d) $\lambda = 5$: $C^2 = 4.0 \times 10^{-8}$

Figure 4.15: Feature detector pairs found in $\text{ANN}_3^{5\times5}$ using DCGD with various values of weight factor λ ((a)-(d)). C^2 is the squared correlation between the feature detectors after training.

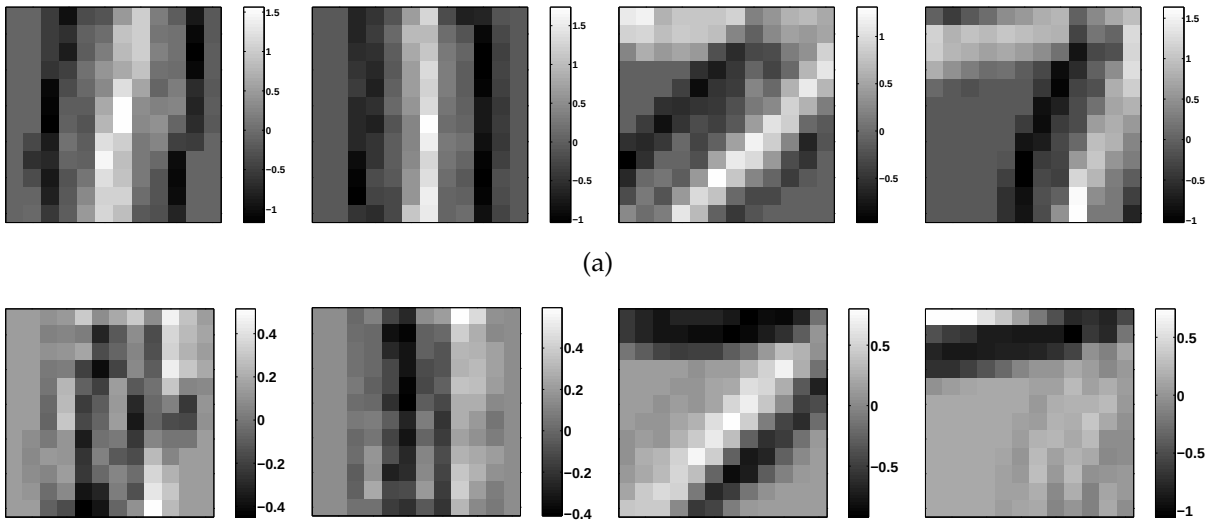


Figure 4.16: The output of (a) the first and (b) the second feature map of $\text{ANN}_3^{5\times5}$ trained with DCGD ($\lambda = 1$), for two samples of class "1" (left) and two samples of class "7" (right). The samples used were, for both digits, the leftmost two in figure 4.7.

detectors are more clear than those found using standard CGD, in section 4.3.1. Their frequency responses resemble those of the feature detectors shown in figure 4.12 (b) and, due to the fixed weight initialisation, are guaranteed to be found when training is repeated. However, λ should be chosen with some care; if it is too small ($\lambda = 0.5$), the squared covariance term will have too little effect; if it is too large ($\lambda = 5$), minimisation of the squared covariance term becomes too important and the original functionality of the network is no longer clearly visible.

The features detected seem to be diagonal bars, as seen before, and horizontal edges. This is confirmed by inspecting the output of the two feature maps in $\text{ANN}_3^{5\times5}$ trained with DCGD, $\lambda = 1$, for a number of input samples (see figure 4.16). For samples of class "1", these outputs are lower than for class "7", i.e. features specific for digits of class "7" have been found. Furthermore, the first feature detector clearly enhances the stem of "7" digits, whereas the second detector amplifies the top stroke.

Finally, versions of $\text{ANN}_3^{5\times5}$ with three and four feature maps were also trained using DCGD. Besides the two feature detectors found before no clear new feature detectors were found.

4.5 Conclusions

Two cases were discussed in which small ANNs were trained on relatively simple data sets. The goal was to find whether ANNs can detect useful features and, if so, whether

they can be interpreted by a human observer.

The first case, edge recognition, showed that for some problems ANN architectures and corresponding weight sets giving minimum error can be found by hand. However, training the architectures does not guarantee that these weight sets will be found. First, there may be a large range of weight sets optimising the MSE, which makes interpretation harder. This makes initialisation of weights with low, fixed values necessary. Second, the presence of bias weights and non-shared weights allows an ANN to store place-specific information instead of finding shift-invariant templates. When more and more prior knowledge is used in restricting the architecture (by sharing biases and weights, or even enforcing a mask) the network finds the optimal weight set specified. However, training the ANNs becomes very hard. Furthermore, these ANNs can hardly be seen as representative of normal practice anymore. This was called the interpretability trade-off, i.e. the trade-off between the possibility of interpretation and the danger of experiments not being representative of real-life problems.

A possible reason for an ANN finding place-dependent solutions might be the artificial nature of the edge data set, in which all edges pass through the centre of the image. Therefore, a problem more true to life was studied by training similar ANNs on a subset of the NIST database containing just 10 samples each of the digits "1" and "7". However, these networks were still able to store place-dependent information in the weights between the hidden layer and output layer. Only after restricting the network did the ANN find a more pronounced feature detector in its receptive field.

Experiments on the digit data set with an ANN with two receptive fields showed that the distribution of feature detectors over the receptive fields is unclear. As random initialisation is necessary, different experiments lead to different results, which give the impression that multiple feature detectors are presented in each receptive field. To gain a better understanding, a new training algorithm was proposed. In this algorithm, decorrelating conjugate gradient descent (DCGD), the squared covariance between weight sets is minimised along with the MSE. It was shown that, mainly because it allows weight initialisation with fixed values, DCGD can lead to recognisable feature detectors that are distributed clearly over the receptive fields. Note that covariance may not always be the best criterion for obtaining interpretable weight sets. There may very well be problems for which the optimal weight sets are different, yet highly covariant. However, DCGD does show that taking criteria other than output error into account is feasible.

The experiments in this chapter were performed to find whether training ANNs with receptive field mechanisms leads to the ANN finding useful, shift-invariant features and if a human observer could interpret these features. In general, it was shown that the mere presence of receptive fields in an ANN and a good performance do not mean that shift-invariant features are detected. Interpretation was only possible after severely restricting the ANN architecture, data set complexity and training method. One thing all experiments had in common was the use of ANNs as classifiers. Classification is a

"derived" goal, i.e. the task is assigning (in principle arbitrary) outputs, representing class labels, to input samples. The ANN is free to choose which features to use (or not) to reach this goal. Therefore, to study the way in which ANNs solve problems moving to regression problems might yield results more fit for interpretation, especially when a regression problem can be decomposed into a number of independent subproblems. The next chapter will study the use of ANNs as nonlinear filters for image enhancement.

5

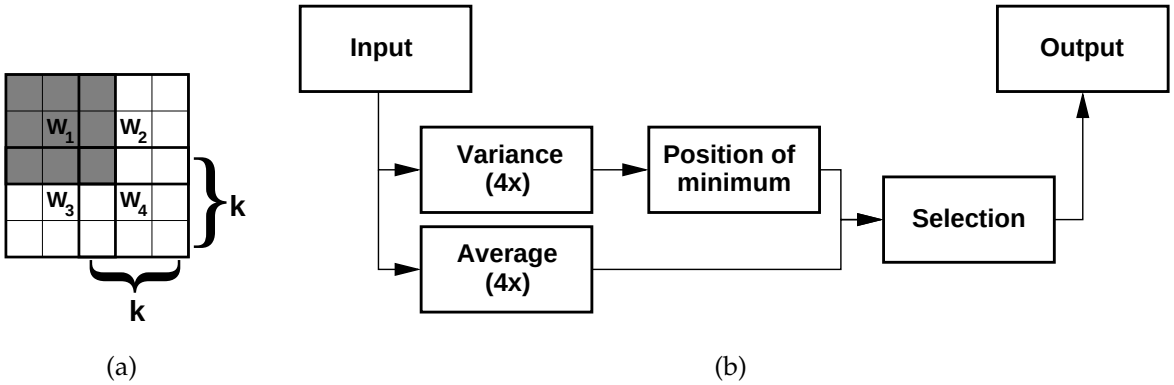
REGRESSION NETWORKS FOR IMAGE RESTORATION

5.1 Introduction

In this chapter and the next, the applicability of neural networks to nonlinear image processing problems will be studied. Whereas in the previous chapter ANNs were trained to perform classification, here they will be used for regression. The primary goal is to see whether standard feed-forward ANNs can be applied successfully to a nonlinear image filtering problem. If so, what are the prerequisites for obtaining a well-functioning ANN? Secondly, the question (as in the previous chapter) is whether these ANNs correspond to classic image processing approaches to solve such a task. Note that the goal here is not to simply apply ANNs to an image processing problem, nor to construct an ANN that will perform better at it than existing techniques. Instead, the question is to what extent ANNs can learn the nonlinearities needed in some image processing applications.

To investigate the possibilities of using feed-forward ANNs and the problems one might encounter, the research concentrates on a single example of a nonlinear filter: the Kuwahara filter for edge-preserving smoothing [201]. Since this filter is well-understood and the training goal is exactly known, it is possible to investigate to what extent ANNs are capable of performing this task. The Kuwahara filter also is an excellent object for this study because of its inherent modular structure, which allows splitting the problem into smaller parts. This is known to be an advantage in learning [6] and gives the opportunity to study subproblems in isolation. Pugmire et al. [286] looked at the application of ANNs to edge detection and found that structuring learning in this way can improve performance; however, they did not investigate the precise role this structuring plays.

ANNs have previously been used as image filters, as discussed in section 2.4.1. However, the conclusion was that in many applications the ANNs were non-adaptive. Furthermore, where ANNs *were* adaptive, a lot of prior knowledge of the problem to be



(a) (b)

Figure 5.1: (a) The Kuwahara filter: $k \times k$ subwindows in a $(2k-1) \times (2k-1)$ window; here $k = 3$. (b) Kuwahara filter operation as a sequence of operations.

solved was incorporated in the ANN's architectures. Therefore, in this chapter a number of modular ANNs will be constructed and trained to emulate the Kuwahara filter, incorporating prior knowledge in various degrees. Their performance will be compared to standard feed-forward ANNs. Based on results obtained in these experiments, in chapter 6 it is shown that several key factors influence ANN behaviour in this kind of task.

This chapter starts by discussing the Kuwahara filter, in section 5.2. Next, the various ANN architectures used are introduced in section 5.3. In section 5.4, the experiments performed using these ANNs are described, followed by a discussion of the questions raised by the results in sections 5.5 and 5.6. Chapter 6 then discusses possible causes of the problems encountered in the experiments and investigates solutions.

5.2 Kuwahara filtering

The Kuwahara filter is used to smooth an image while preserving the edges [201, 252, 357]. Figure 5.1 (a) illustrates its operation. The input of the filter is a $(2k-1) \times (2k-1)$ pixel neighbourhood around the central pixel. This neighbourhood is divided into 4 overlapping subwindows W_i, $i = 1, 2, 3, 4$, each of size $k \times k$ pixels. For each of these subwindows, the average μ_i and the variance σ_i^2 of the k^2 grey values is calculated. The output of the filter is then found as the average μ_m of the subwindow W_m having the smallest grey value variance ($m = \arg\min_i \sigma_i^2$). This operation can be applied in a scan-wise manner to filter an entire image. For an example of the effect of the filter, see figure 5.2.

The filter is nonlinear. As the selection of the subwindow based on the variances is data-driven, edges are not blurred as in normal uniform filtering. Since a straight edge will

(a) Image A (b) Image B (c) Image C

Figure 5.2: Images used for (a) training and (b)-(c) testing purposes. The top images are the originals; the bottom images are the Kuwahara filtered versions (for image A, the training target). For presentation purposes, the contrast of the images has been stretched [385].

always lie in at most three subwindows, there will always be at least one subwindow that does not contain an edge and therefore has low variance. For neighbouring pixels in edge regions, different subwindows will be selected (due to the minimum operation), resulting in sudden large differences in grey value. Typically, application of the Kuwahara filter to natural images will result in images which have an artificial look but which may be more easily segmented or interpreted.

This filter was selected for this research since:

- It is *modular* (figure 5.1 (b) illustrates this). This means the operation can be split into subtasks which can perhaps be learned more easily than the whole task at once. It will be interesting to see whether an ANN will need this modularity and complexity in order to approximate the filter's operation. Also, it offers the opportunity to study an ANN's operation in terms of the individual modules.

- It is *nonlinear*. If ANNs can be put to use in image processing, the most rewarding application will be one to nonlinear rather than linear image processing. ANNs are most often used for learning (seemingly) highly complex, nonlinear tasks with many parameters using only a relatively small number of samples.

5.3 Architectures

Although an abundance of ANN architectures has been applied to image filtering, in the experiments in this chapter only the most widely used type of ANN was used, the standard feed-forward ANN (as discussed in section 2.2.1). This type of ANN is well-studied; the learning algorithms and pitfalls are well known and it has been shown that a feed-forward ANN is a universal approximator [160], making it applicable to the image filtering problem. However, relatively little is known about the actual operation of the ANN, that is, the inner workings of the *"black box"* many consider it to be (for a rather polemic discussion on this topic, see the excellent paper by Green [134]).

In the previous chapter, it was shown that when studying ANN properties, such as internal operation (which functions are performed by which hidden units) or generalisation capabilities, one often encounters a phenomenon which could be described as an ANN interpretability trade-off (section 4.2.3). This, trade-off, controlled by restricting the architecture of an ANN, is between the possibility of understanding how a trained ANN operates and the degree to which the experiment is still true-to-life. In order to cover the spectrum of possibilities, a number of modular ANNs with varying degrees of freedom was constructed. The layout of such a modular ANN is shown in figure 5.3. Of the modular ANNs, four types were created, $\text{ANN}_1^{\text{M}} \ldots \text{ANN}_4^{\text{M}}$. These are discussed below in descending order of artificiality; i.e., the first is completely hand-designed, with every weight set to an optimal value, while the last consists of only standard feed-forward modules.

5.3.1 Modular networks

Each modular ANN consists of four modules. In the four types of modular ANN, different modules are used. These types are:

- For $\mathbf{ANN_1^M}$, the modules were hand-designed for the tasks they are to perform. In some cases, this meant using other than standard (i.e. sigmoid, linear) transfer functions and very unusual weight settings. Figure 5.5 shows the four module designs and the weights assigned to their connections:
  - The **average module** (MOD^{Avg}, figure 5.5 (a)) uses only linear transfer functions in units averaging the inputs. Four of these modules can be used to calculate $\mu_1, \ldots, \mu_4$.
  - The **variance module** (MOD^{Var}, figure 5.5 (b)) uses a submodule (on the left) to calculate the average of the subwindow it is presented. The other submodule

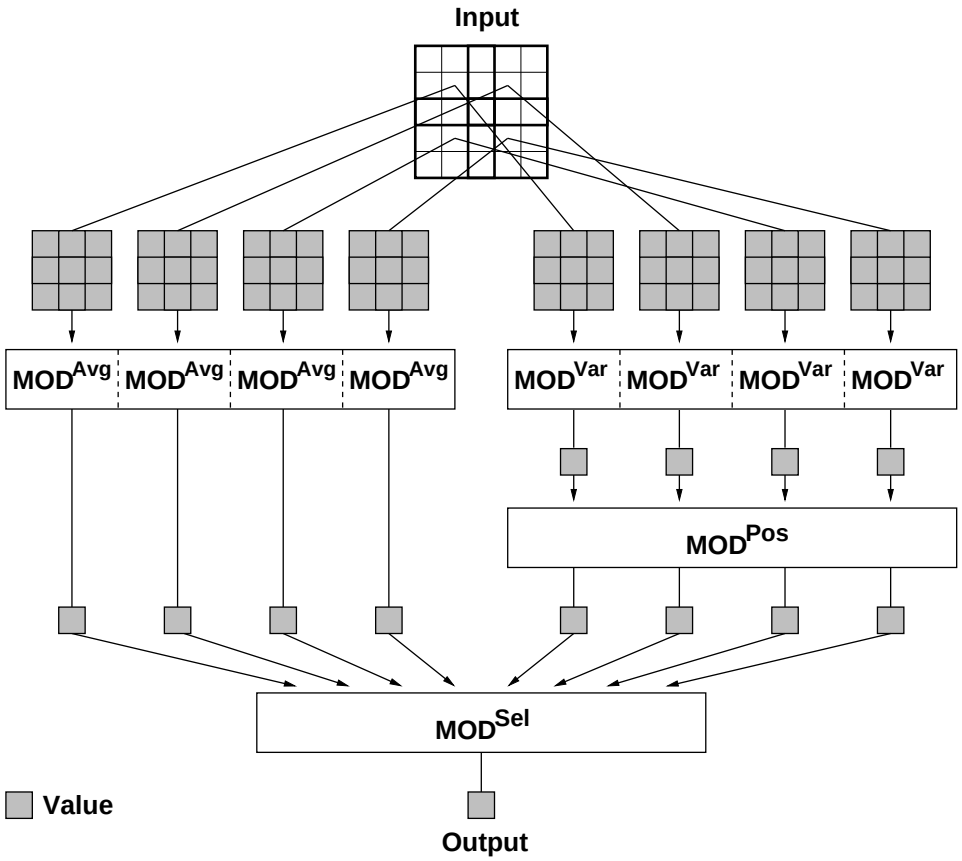
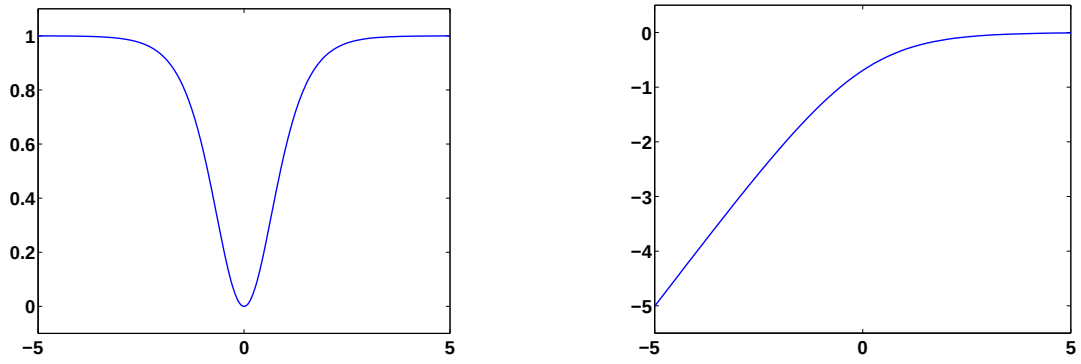
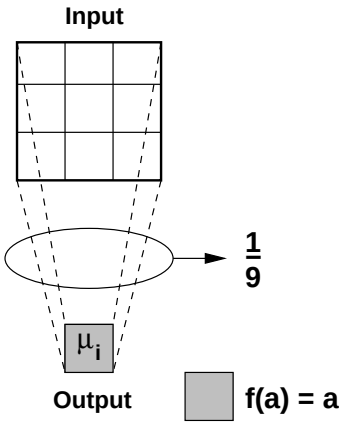


Figure 5.3: A modular ANN. MOD^{Avg}, MOD^{Var}, MOD^{Pos} and MOD^{Sel} denote the ANN modules, corresponding to the operations shown in figure 5.1 (b). The top layer is the input layer. In this figure, shaded boxes correspond to values transported between modules, not units.

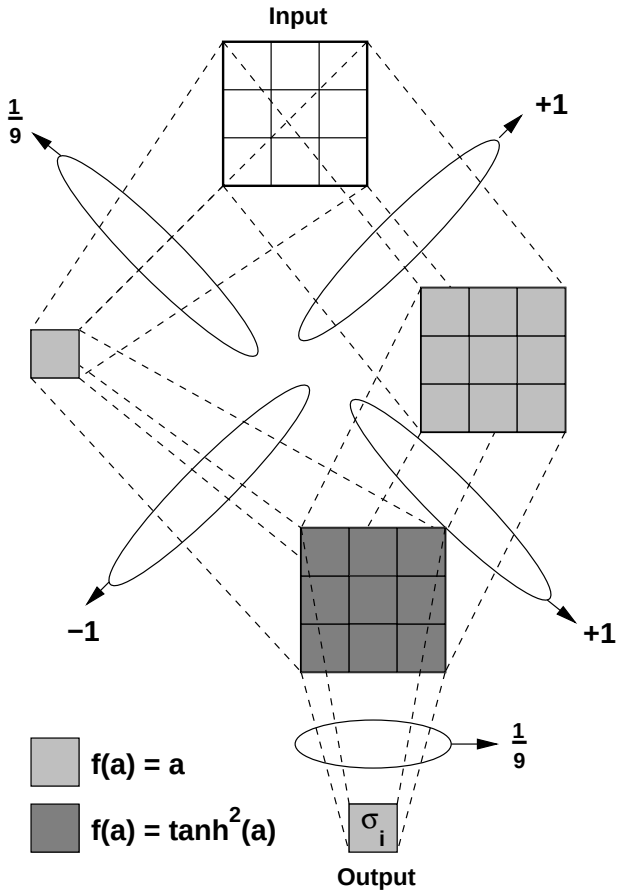


(a) $f(a) = \tanh^2(a)$ (b) $f(a) = \ln \frac{1}{1+\exp(-a)}$

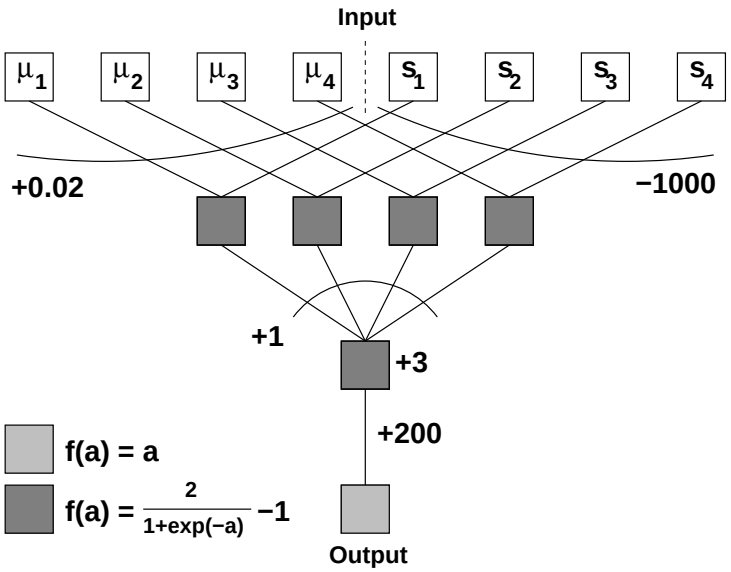
Figure 5.4: The non-standard transfer functions used in (a) MOD^{Var} and (b) MOD^{Pos}.



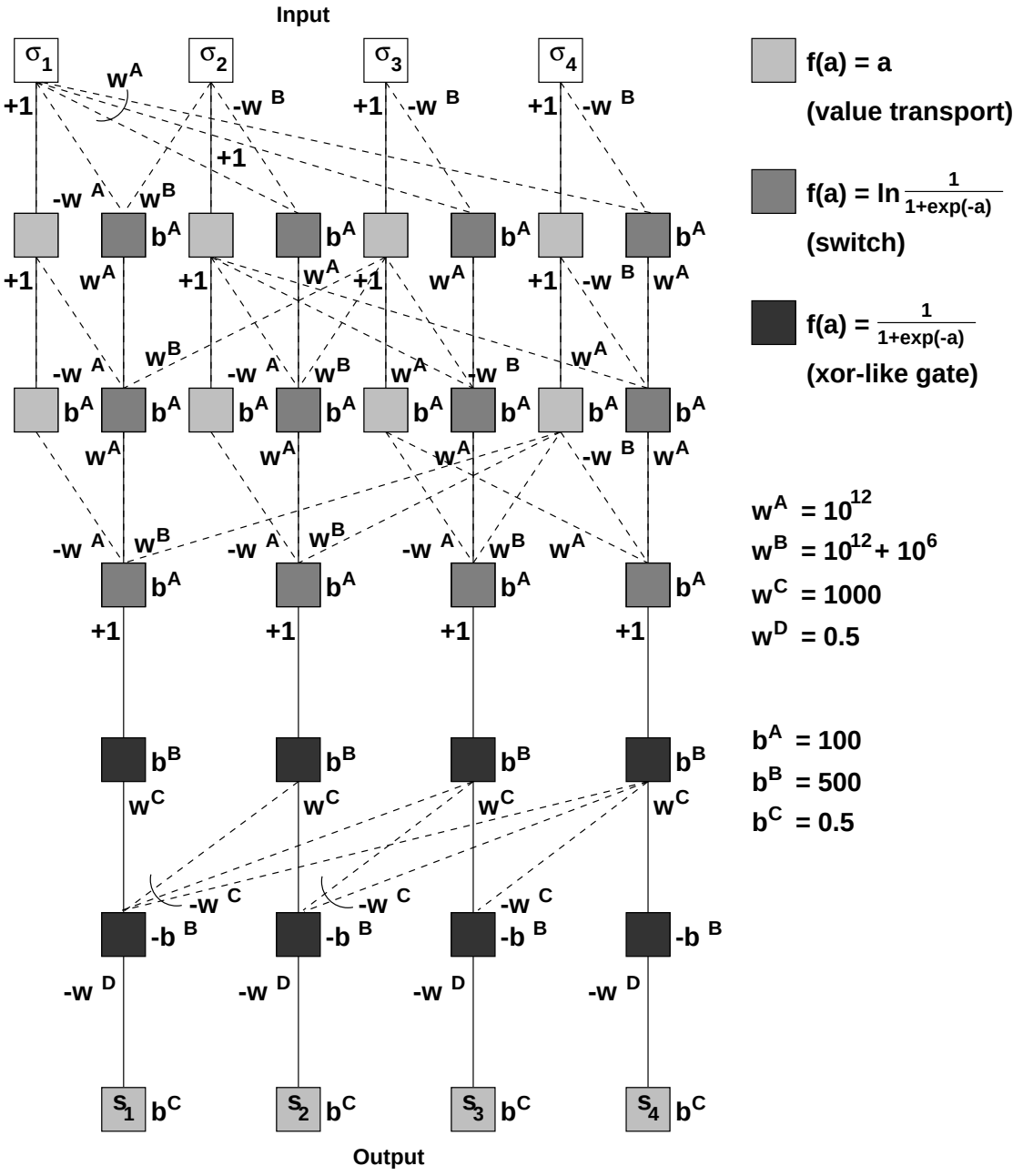
(a) MOD$^{\text{Avg}}$



(b) MOD$^{\text{Var}}$



(d) MOD$^{\text{Sel}}$



(c) $\mathrm{MOD}^{\mathrm{Pos}}$

Figure 5.5: The modules for (a) calculating the average, (b) calculating the variance, (c) finding the position of the minimum variance and (d) selecting the right average. In all modules, the top layer is the input layer. Differently shaded boxes correspond to units with different transfer functions.

(on the right) just transports the original data to lower layers[1]. The calculated averages are then subtracted from the original inputs, followed by a layer of units using a $f(a) = \tanh^2(a)$ transfer function to approximate the square of the input[2] (see figure 5.4 (a)). Four of these modules can be used to find $\sigma_1^2, \ldots, \sigma_4^2$.

– The **position-of-minimum module** for selecting the position of the minimum of four inputs (MOD^{Pos}, figure 5.5 (c)) is the most complicated one. Using the logarithm of the sigmoid (eqn. 2.3) as a transfer function,

$$f(a) = \ln \frac{1}{1 + \exp(-a)} \tag{5.1}$$

(see figure 5.4 (b)), units in the first three hidden layers act as *switches* comparing their two inputs. Alongside these switches, linear transfer function units are used to *transport* the original values to deeper layers. Weights w^A and w^B are very high to enable the units to act as switches. If the input connected using weight w^A (input I^A) is greater than the input connected using weight w^B (input I^B), the sum will be large and negative, the output of the sigmoid will approach 0.0 and the output of the unit will be $-\infty$. If $I^B > I^A$, on the other hand, the sum will be large and positive, the output of the sigmoid part will approach 1.0 and the final output of the unit will be 0.0. This output can be used as an inhibiting signal, by passing it to units of the same type in lower layers. In this way, units in the third hidden layer have as output – if inputs are denoted as $\sigma_1, \sigma_2, \sigma_3$ and σ_4 – :

$$s_i = \begin{cases} 0.0 & \sigma_i < \min_{m=1,\ldots,4 \wedge m \neq i} \sigma_m \\ 0.5 & \text{otherwise} \end{cases} \tag{5.2}$$

Weights w_A and w_B are slightly different to handle cases in which two inputs are exactly the same but one (in this case arbitrary) minimum position has to be found. The fourth and fifth hidden layer ensure that exactly one output unit will indicate that the corresponding input was minimal, by setting the output of a unit to 0.0 if another unit to the right has an output $\neq$ 0.0. The units perform an *xor-like* function, giving high output only when exactly one of the inputs is high. Finally, biases (indicated by b^A, b^B and b^C next to the units) are used to let the outputs have the right value (0.0 or 0.5).

– The **selection module** (MOD^{Sel}, figure 5.5 (d)) uses large weights coupled to the position-of-minimum module outputs (inputs s_1, s_2, s_3 and s_4) to suppress the unwanted average values μ_i before adding these. The small weights with which

[1] This part is not strictly necessary, but was incorporated since links between non-adjacent layers are difficult to implement in the software package used [155].

[2] This function is chosen since it approximates a^2 well on the interval it will be applied to, but is bounded: it asymptotically reaches 1 as the input grows to $\pm\infty$. The latter property is important for training the ANN, as unbounded transfer functions will hamper convergence.

the average values are multiplied and the large incoming weight of the output unit are used to avoid the nonlinearity of the transfer function.

Since all weights were fixed, this ANN was not trained.

- $\mathbf{ANN_2^M}$ modules have the same architectures as those of ANN_1^M. However, in this case the weights were not fixed, hence the modules could be trained. These modules were expected to perform poorly, as some of the optimal weights (as set in ANN_1^M) were very high and some of the transfer functions are unbounded (see figure 5.4 (b)).
- In $\mathbf{ANN_3^M}$ modules, non-standard transfer functions were no longer used. As a result, the modules MOD^{Var} and MOD^{Pos} had to be replaced by standard ANNs. These ANNs contained 2 layers of 25 hidden units, each of which had a double sigmoid transfer function (eqn. 2.4). This number of hidden units was thought to give the modules a sufficiently large number of parameters, but keeps training times feasible.
- In the final type, $\mathbf{ANN_4^M}$, all modules consisted of standard ANNs with 2 hidden layers of 25 units each.

With these four types, a transition is made from a fixed, hard-wired type of ANN (ANN_1^M), which is a hard-wired implementation of the Kuwahara filter, to a free type (ANN_4^M) in which only the prior knowledge that the filter consists of four subtasks is used. The goal of the exercise is to see a gradual change in behaviour and performance.

Note that the ANN_1^M architecture is probably not the only error-free implementation possible using ANN units. It should be clear from the discussion, though, that any architecture should resort to using non-standard transfer functions and unconventional weight settings to perform the nonlinear operations error-free over a large range of input values. In this respect, the exact choices made here are less important.

5.3.2 Standard networks

As chapter 3 showed, the use of prior knowledge in ANN design will not always guarantee that such ANNs will perform better than standard architectures. To validate results obtained with the ANNs described in the previous section, experiments were also performed with standard, fully connected feed-forward ANNs. Although one hidden layer should theoretically be sufficient [119, 160], the addition of a layer may ease training or lower the number of required parameters (although there is some disagreement on this, see [48, 80, 119]). Therefore, ANNs having one or two hidden layers of 1, 2, 3, 4, 5, 10, 25, 50, 100 or 250 units each were used. All units used the double sigmoid transfer function (eqn. 2.4). These ANNs will be referred to as $ANN_{L\times U}^S$, where L indicates the number of hidden layers (1 or 2) and U the number of units per hidden layer. ANN_L^S will be used to denote the entire set of ANNs with L hidden layers.

5.4 Experiments

5.4.1 Data sets

To train the ANNs, a training set was constructed by drawing samples randomly, using a uniform distribution, from image A (input) and its Kuwahara filtered version (output), both shown in figure 5.2 (a). The original 8-bit 256 grey value image was converted to a floating point image and rescaled to the range $[-0.5, 0.5]$. Three data sets were constructed, containing 1,000 samples each: a training set, a validation set and a testing set. The validation set was used to prevent overtraining (see section 3.3.1): if the error on the validation set did not drop below the minimum error found so far on that set for 1,000 cycles, training was stopped. Since in all experiments only $k = 3$ Kuwahara filters were studied, the input to each ANN was a 5×5 region of grey values and the training target was 1 value. For the modular ANNs, additional data sets were constructed from these original data sets to obtain the mappings required by the individual ANNs (average, variance, position-of-minimum and selection).

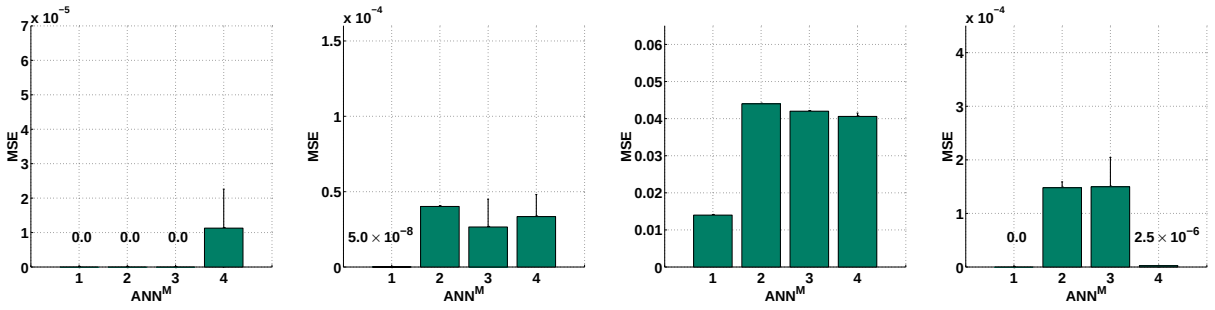
5.4.2 Training

For training, the standard stochastic back propagation algorithm [306] was used. Weights were initialised to random values drawn from a uniform distribution in the range $[-0.1, 0.1]$. The learning rate was set to 0.1; no momentum was used. Training was stopped after 25,000 cycles or if the validation set indicated overtraining, whichever came first. All experiments were repeated five times with different random initialisations; all results reported are averages over five experiments. Where ever appropriate, error bars indicate standard deviations.

Results are given in figures 5.6 and 5.7. These will be discussed for the different architectures in the next sections.

5.4.3 Modules

The different modules show rather different behaviour (figure 5.6). Note that in these figures the MSE was calculated on a testing set of 1,000 samples. As was to be expected, the MSE is lowest for the hand-constructed ANN_1^{M} modules: for all ANNs except MOD^{Pos}, it was 0. The error remaining for the MOD^{Pos} module may look quite high, but is caused mainly by the ANN choosing a wrong minimum when two or more input values σ_i are very similar. Although the effect on the behaviour of the final module (MOD^{Sel}) will be negligible, the MSE is quite high since one output which should have



(a) MOD$^{\text{Avg}}$ (c) MOD$^{\text{Var}}$ (e) MOD$^{\text{Pos}}$ (g) MOD$^{\text{Sel}}$

Figure 5.6: Performance of the individual modules on the testing set in each of the modular ANNs, $\text{ANN}_1^{\text{M}} \ldots \text{ANN}_4^{\text{M}}$.

been 0.5 is incorrectly set to 0.0 and vice versa, leading to an MSE of 0.25 for that input pattern. For the other ANNs, it seems that if the manually set weights are dropped (ANN_2^{M}), the modules are not able to learn their function as well as possible (i.e., as well as ANN_1^{M}). Nonetheless, the MSE is quite good and comparable to ANN_3^{M} and ANN_4^{M}.

When the individual tasks are considered, the average is obviously the easiest function to approximate. Only for ANN_4^{M}, in which standard modules with two hidden layers were used, is the MSE larger than 0.0; apparently these modules generalise less well than the hand-constructed, linear MOD$^{\text{Avg}}$s. The variance too is not difficult: MSEs are $\mathcal{O}(10^{-5})$. Clearly, the position-of-minimum task is the hardest. Here, almost all ANNs perform poorly. Performances on the selection problem, finally, are quite good. What is interesting is that the more constrained modules (ANN_2^{M}, ANN_3^{M}) perform less well than the standard ones. Here again the effect that the construction is closely connected to the optimal set of weights plays a role. Although there is an optimal weight set, the training algorithm did not find it.

5.4.4 Modular networks

When the modules are concatenated, the initial MSEs of the resulting ANNs are poor: for ANN_2^{M}, ANN_3^{M} and ANN_4^{M} $\mathcal{O}(1)$, $\mathcal{O}(10^{-1})$ and $\mathcal{O}(10^{-2})$ respectively. The MOD$^{\text{Pos}}$ module is mainly responsible for this; it is the hardest module to learn due to the non-linearity involved (see the discussion in section 5.4.3). If the trained MOD$^{\text{Pos}}$ in $\text{ANN}_2^{\text{M}} \ldots \text{ANN}_4^{\text{M}}$ is replaced by the constructed ANN_1^{M} module, the overall MSE always decreases significantly (see table 5.1). This is an indication that, although its MSE seems low ($\mathcal{O}(10^{-2})$), this module does not perform well. Furthermore, it seems that the overall MSE is highly sensitive to the error this module makes.

However, when the complete ANNs are trained a little further with a low learning rate (0.1), the MSE improves rapidly: after only 100-500 learning cycles training can

Type	MSE	MSE with MOD^{Pos} of ANN_1^{M}
ANN_2^{M}	$9.2 \times 10^{-1} \pm 5.2 \times 10^{-1}$	$8.7 \times 10^{-4} \pm 1.7 \times 10^{-4}$
ANN_3^{M}	$1.2 \times 10^{-1} \pm 1.2 \times 10^{-1}$	$1.0 \times 10^{-3} \pm 2.0 \times 10^{-4}$
ANN_4^{M}	$3.6 \times 10^{-2} \pm 1.7 \times 10^{-2}$	$1.2 \times 10^{-3} \pm 2.4 \times 10^{-4}$

Table 5.1: Dependence of performance, in MSE on the image A testing set, on the MOD^{Pos} module. Values given are average MSEs and standard deviations.

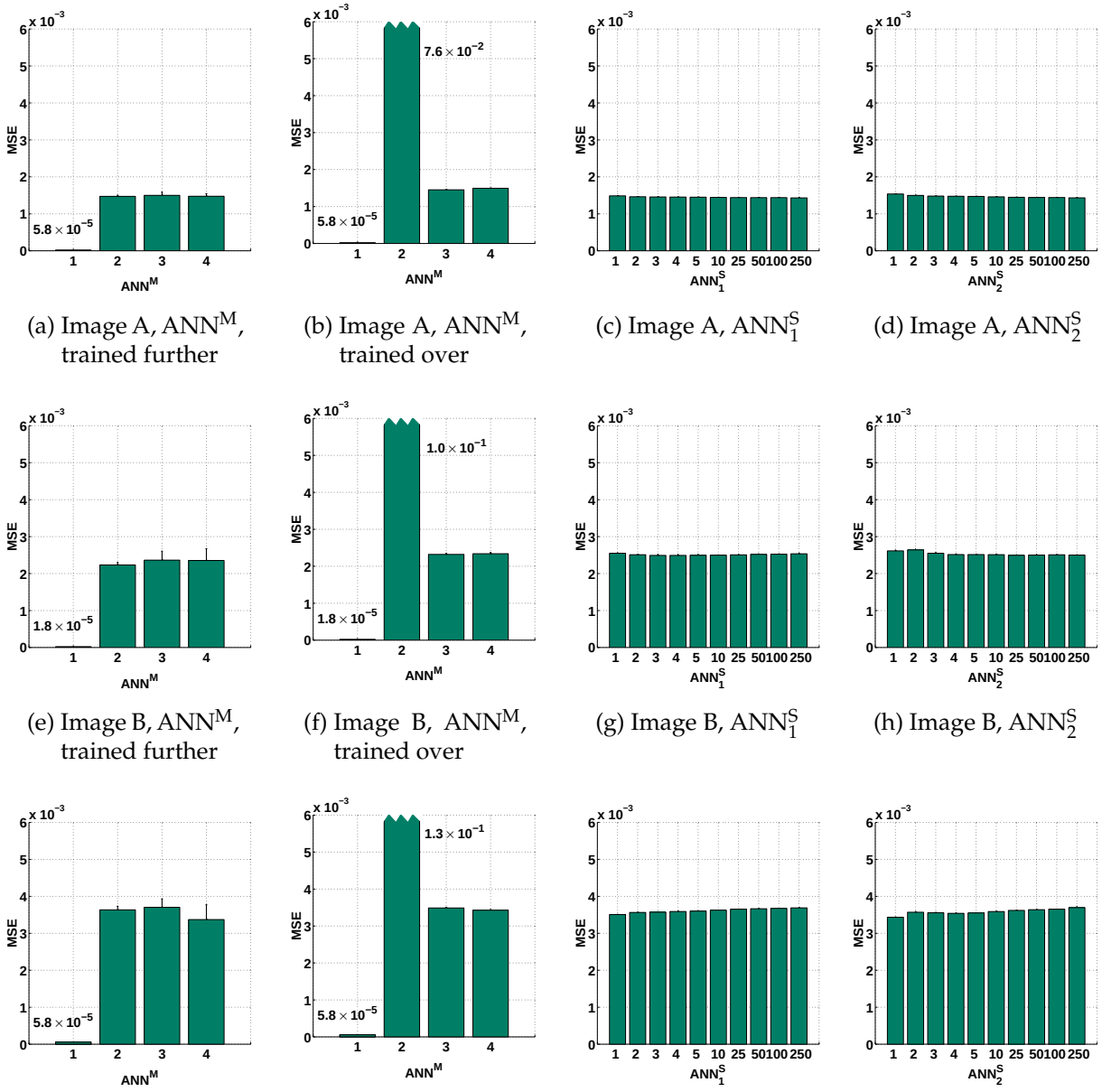
be stopped. In Pugmire et al. [286], the same effect occurs. The MSEs of the final ANNs on the entire image are shown in figures 5.7 (a), (e) and (i) for images A, B and C, respectively. Images B and C were pre-processed in the same way as image A: the original 8-bit (B) and 5-bit (C) 256 grey value images were converted to floating point images, with grey values in the range $[-0.5, 0.5]$.

To get an idea of the significance of these results, re-initialised versions of the same ANNs were also trained. That is, all weights of the concatenated ANNs were initialised randomly without using the prior knowledge of modularity. The results of these training runs are shown in figures 5.7 (b), (f) and (j). Note that only ANN_2^{M} cannot be trained well from scratch, due to the non-standard transfer functions used. For ANN_3^{M} and ANN_4^{M} the MSE is comparable to the other ANNs. This would indicate that modular training is not beneficial, at least according to the MSE criterion.

The ANNs seem to generalise well, in that nearly identical MSEs are reached for each network on all three images. However, the variance in MSE is larger on Image B and Image C than it is for Image A. This indicates that the modular networks may have become slightly too adapted to the content of Image A.

5.4.5 Standard networks

Results for the standard ANNs, ANN^{S}s, are shown in figure 5.7 (c)-(d), (g)-(h) and (k)-(l) for images A, B and C. In each case, the first figure gives the results for ANNs with one hidden layer; the second figure for ANNs with two hidden layers. What is most striking is that for almost all sizes of the ANNs the MSEs are more or less the same. Furthermore, this MSE is nearly identical to the one obtained by the modular ANNs ANN_2^{M} ... ANN_4^{M}. It also seems that the smaller ANNs, which give a slightly larger MSE on Image A and Image B, perform a bit worse on Image C. This is due to the larger amount of edge pixels in Image C; the next section will discuss this further.



(a) Image A, ANN^{M}, trained further

(b) Image A, ANN^{M}, trained over

(c) Image A, ANN_1^{S}

(d) Image A, ANN_2^{S}

(e) Image B, ANN^{M}, trained further

(f) Image B, ANN^{M}, trained over

(g) Image B, ANN_1^{S}

(h) Image B, ANN_2^{S}

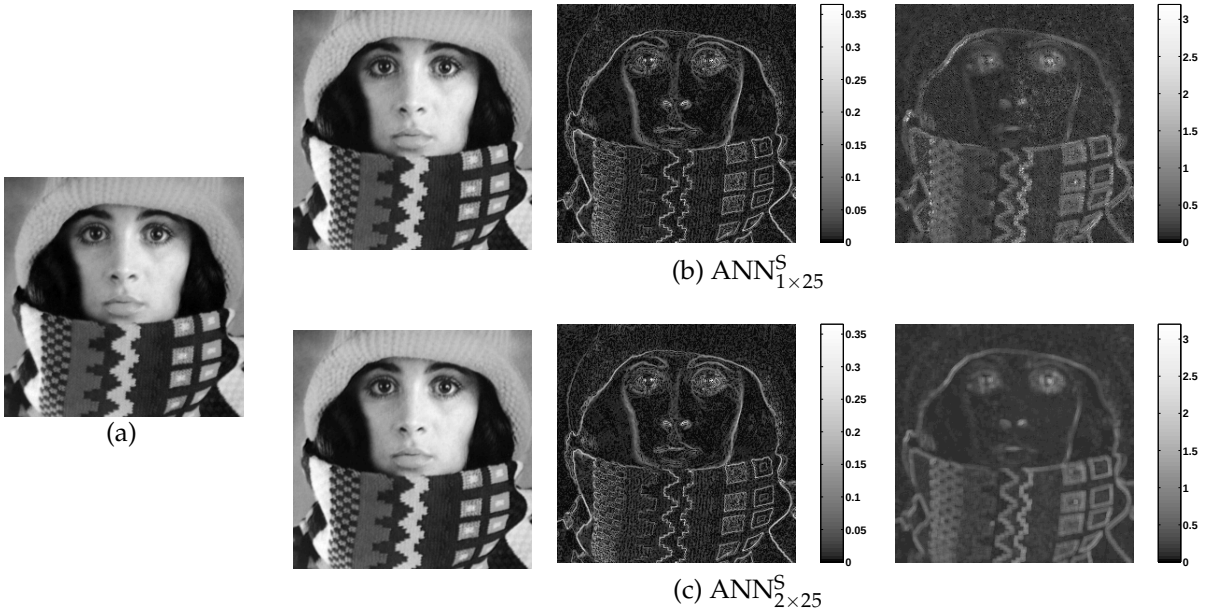
(i) Image C, ANN^{M}, trained further

(j) Image C, ANN^{M}, trained over

(k) Image C, ANN_1^{S}

(l) Image C: ANN_2^{S}

Figure 5.7: Performance of all ANN^{M}s and ANN^{S}s on the three images used: (a)-(d) on image A (fig. 5.2 (a)), (e)-(h) on image B (fig. 5.2 (b)) and (i)-(l) on image C (fig. 5.2 (c)). For the ANN^{S}s, the x-axis indicates the number of hidden units per layer.



(a)

(b) $\text{ANN}^{\text{S}}_{1\times25}$

(c) $\text{ANN}^{\text{S}}_{2\times25}$

Figure 5.8: (a) The original Image A. (b) and (c), from left to right: outputs of two ANN^{S}s on Image A; absolute differences between target image and ANN output and ANN output error bar widths plotted as grey values.

5.5 Investigating the error

The experiments in the previous section indicate that, no matter which ANN is trained (except for ANN^{M}_1), the MSE it will be able to reach on the images is equal. However, visual inspection shows small differences between images filtered by various ANNs; see e.g. the left and centre columns of figure 5.8. To gain more insight in the actual errors the ANNs make, a technique can be borrowed from the field of Bayesian learning, which allows the calculation of error bars for each output of the ANN [28]. This technique is explained in appendix B. The computation is based on the Hessian of the ANN output w.r.t. its weights $\mathbf{w}$, $\mathbf{H} = \nabla^2_{\mathbf{w}} R(\mathbf{x};\ \mathbf{w})$, which needs to be found first. Using $\mathbf{H}$, for each input $\mathbf{x}$ a corresponding variance σ^2_{tot} can be found. This makes it possible to create an image in which each pixel corresponds to $2\sigma_{tot}$, i.e. the grey value equals half the width of the error bar on the ANN output at that location. Conversely, the inverse of σ_{tot} is sometimes used as a measure of *confidence* in an ANN output for a certain input.

For a number of ANNs, the Hessian was calculated using a finite differencing approximation [28]. To calculate the error bars, this matrix has to be inverted first. Unfortunately, for the ANN^{M}s, inversion was impossible as their Hessian matrices were too ill-conditioned because of the complicated architectures, containing fixed and shared weights. Figures 5.8 (b) and (c) show the results for two standard ANNs, $\text{ANN}^{\text{S}}_{1\times25}$ and $\text{ANN}^{\text{S}}_{2\times25}$. In the left column the ANN output for Image A (5.2 (a)) is shown. The centre

column shows the absolute difference between this output and the target image. In the third column the error bars calculated using the Hessian are shown.

The figures show that the error the ANN makes is not spread out evenly over the image. The highest errors occur near the edges in Image A, as can be seen by comparing the centre column of figure 5.8 with the gradient magnitude of $|\nabla I_A|$ of Image A, shown in figure 5.9 (a). This gradient magnitude is calculated as ([385])

$$|\nabla I_A| = \sqrt{\left(\frac{\delta I_A}{\delta x}\right)^2 + \left(\frac{\delta I_A}{\delta y}\right)^2} \tag{5.3}$$

where $\frac{\delta I_A}{\delta x}$ is approximated by convolving Image A with a $[-1 \quad 0 \quad 1]$ mask, and $\frac{\delta I_A}{\delta y}$ by convolving Image A with its transpose.

The error bar images, in the right column of figure 5.8, show that the standard deviation of ANN output is also highest on and around the edges. Furthermore, although the output of the ANNs look identical, the error bars show that the ANNs actually behave differently.

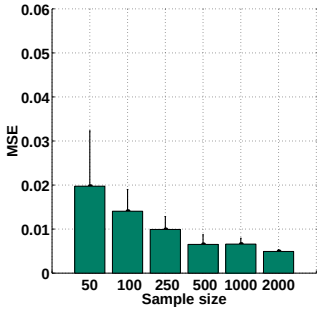
These results lead to the conclusion that the ANNs have learned fairly well to approximate the Kuwahara filter in flat regions, where it operates like a local average filter. However, on and around edges they fail to give the correct output; most edges are sharpened slightly, but not nearly as much as they would be by the Kuwahara filter. In other words, the linear operation of the Kuwahara filter is emulated correctly, but the nonlinear part is not. Furthermore, the error bar images suggest there are differences between ANNs which are not expressed in their MSEs.

5.6 Conclusions

A number of experiments on implementing a basic nonlinear filter, the Kuwahara filter, were presented. Since this filter is of a modular algorithmic nature, modular versions of the ANN were constructed and trained. A gradual shift in performance was expected as ANNs were less and less constrained, at the same time losing the possibility of understanding the workings of the ANNs. To compare this to approaches not using prior knowledge, a number of standard ANNs were trained as well.

The most noticeable result of the experiments is that whatever ANN is trained, be it a simple one hidden unit ANN or a specially constructed modular ANN, approximately the same performance (measured in MSE) can be reached. Modular training does not seem to boost performance at all. However, inspection of error images and standard deviation of ANN outputs suggests that there are differences between ANNs. Furthermore, the errors made by ANNs are concentrated around edges, i.e. in the part where the Kuwahara filter's nonlinearity comes into play.

(a)



(b)

Figure 5.9: (a) The gradient magnitude of Image A, $|\nabla I_A|$. (b) Performance of $\text{ANN}^{\text{S}}_{1\times 50}$ for various training set sample sizes.

There are a number of hypotheses as to what causes all ANNs to seemingly perform equally well, some of which will be investigated in the next chapter:

- *the problem may simply be too hard to be learned by a finite-size ANN*. This does not seem plausible, since even for a two-hidden layer ANN with 250 hidden units per layer, resulting in a total of 69,000 free parameters, the MSE is no better than for very simple ANNs. One would at least expect to see some enhancement of results;
- *it is possible that the sample size of 1,000 is too small*, as it was rather arbitrarily chosen. An experiment was performed in which $\text{ANN}^{\text{S}}_{1\times 50}$ was trained using training sets with 50, 100, 250, 500, 1,000 and 2,000 samples. The results, given in figure 5.9 (b), show however that the chosen sample size of 1,000 seems sufficient. The decrease in MSE when using 2,000 samples in the training set is rather small;
- *the training set may not be representative for the problem*, i.e. the nature of the problem may not be well reflected in the way the set is sampled from the image. An experiment to test this hypothesis is discussed in the next section (section 6.2);
- *the error criterion may not be fit for training the ANNs or assessing their performance*. It is very well possible that the MSE criterion used is of limited use in this problem, since it weighs both the interesting parts of the image, around the edges, and the less interesting parts equally. The role of the MSE as an assessment criterion is investigated in section 6.3 and its use in training in section 6.4;
- *the problem may be of such a nature that local minima are prominently present in the error surface*, while the global minima are very hard to reach, causing suboptimal ANN operation. This hypothesis is tested in section 6.5.

Besides testing the hypotheses put forward above, chapter 6 will inspect the modular architectures discussed in this chapter. The goal is to answer the question whether the use of prior knowledge is necessary for this application.

6

INSPECTION AND IMPROVEMENT OF REGRESSION NETWORKS

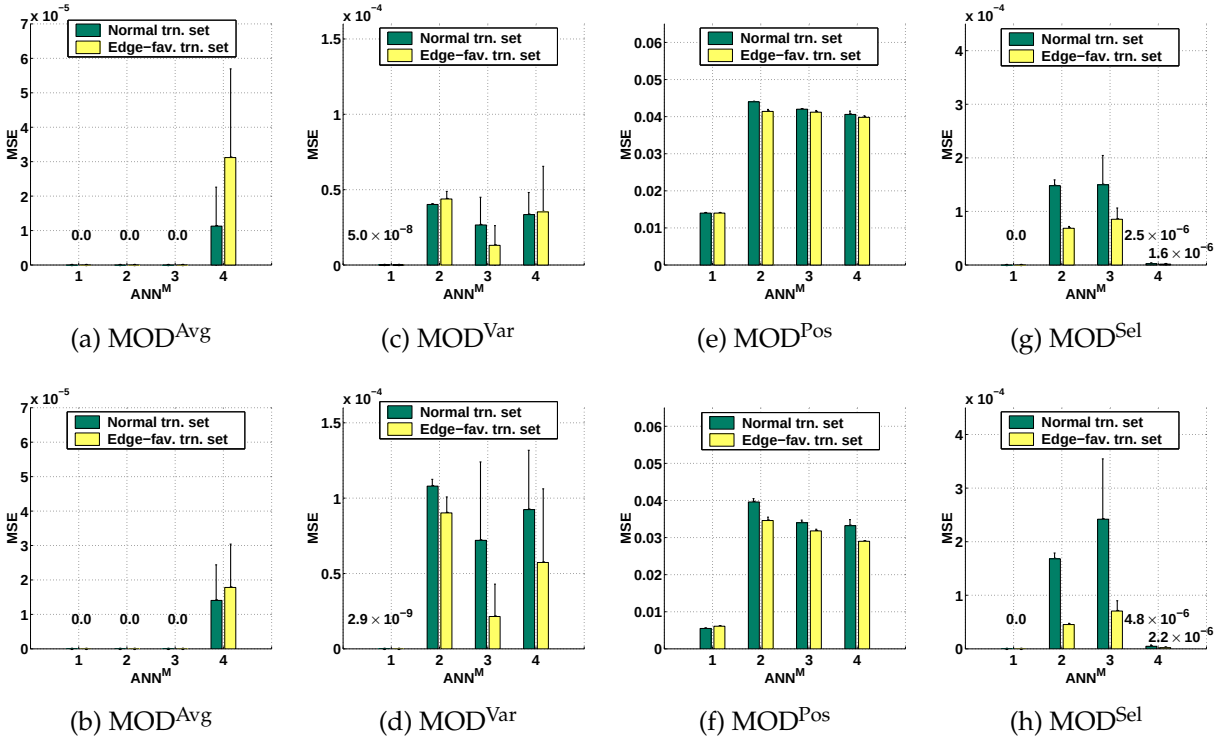
6.1 Introduction

In chapter 5, a number of modular and standard ANNs were trained to perform a non-linear image filtering operation. The results of the experiments raised a number of questions, which were discussed in section 5.6. This chapter will try to answer these questions. First, in section 6.2, the influence of data set sampling is investigated. Next, in section 6.3 the appropriateness of the MSE as a performance measure is discussed and new measures are presented. As it is shown that the MSE is not the optimal error measure for this kind of problem, the use of different ANN error measures in training is investigated in section 6.4. To investigate how ANNs solve the image processing task, sections 6.5 and 6.6 deal with inspection of the standard and modular ANNs, respectively. Finally, in section 6.7 some conclusions are drawn.

6.2 Edge-favouring sampling

Inspection of the ANN outputs and the error bars on those outputs led to the conclusion that the ANNs had learned to emulate the Kuwahara filter well in most places, except in regions near edges (section 5.5). A problem in sampling a training set from an image[1] for this particular application is that such interesting regions, i.e. the regions where the filter is nonlinear, are very poorly represented. Edge pixels constitute only a very small percentage of the total number of pixels in an image (as a rule of thumb, $\mathcal{O}(\sqrt{n})$ edge

[1] From here on, the term *sampling* will be used to denote the process of constructing a data set by extracting windows from an image with coordinates sampled from a certain distribution on the image grid.



(a) MOD$^{\text{Avg}}$ (c) MOD$^{\text{Var}}$ (e) MOD$^{\text{Pos}}$ (g) MOD$^{\text{Sel}}$

(b) MOD$^{\text{Avg}}$ (d) MOD$^{\text{Var}}$ (f) MOD$^{\text{Pos}}$ (h) MOD$^{\text{Sel}}$

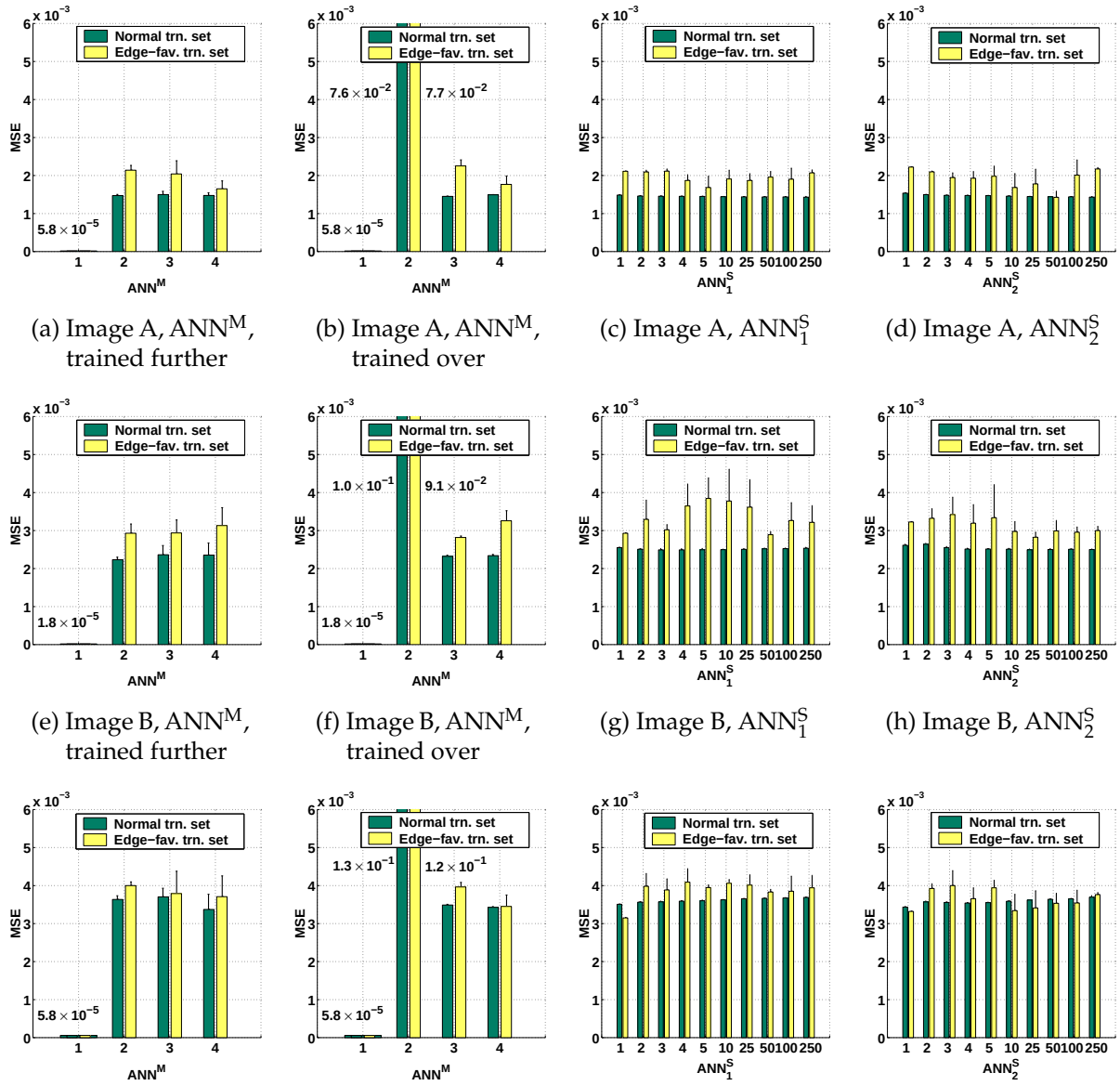
Figure 6.1: Performance of the individual modules in each of the modular ANNs, ANN_1^{M} ... ANN_4^{M} on the normal testing set (top row) and edge-favouring testing set (bottom row).

pixels on $\mathcal{O}(n)$ image pixels) and will therefore not be represented well in the training set when sampling randomly using a uniform distribution.

To learn more about the influence of the training set on performance, a second group of data sets was created by sampling from Image A (figure 5.2 (a)) with a probability density function based on its gradient magnitude image $|\nabla I_A|$ (eqn. 5.3). If $|\nabla I|$ is scaled by a factor c such that $\int_x \int_y c \cdot |\nabla I(x,y)| dydx = 1$, and used as a probability density function when sampling, edge regions have a much higher probability of being included in the data set than pixels from flat regions. This will be called *edge-favouring sampling*, as opposed to *normal sampling*.

6.2.1 Experiments

Performances (in MSE) of ANNs trained on this edge-favouring set are given in figures 6.1 and 6.2. Note that the results obtained on the normal training set (first shown in figure 5.7) are included again to facilitate comparison. The sampling of the data set clearly has an influence on the results. Since the edge-favouring set contains more



(a) Image A, ANN^{M}, trained further

(b) Image A, ANN^{M}, trained over

(c) Image A, ANN^{S}_1

(d) Image A, ANN^{S}_2

(e) Image B, ANN^{M}, trained further

(f) Image B, ANN^{M}, trained over

(g) Image B, ANN^{S}_1

(h) Image B, ANN^{S}_2

(i) Image C, ANN^{M}, trained further

(j) Image C, ANN^{M}, trained over

(k) Image C, ANN^{S}_1

(l) Image C, ANN^{S}_2

Figure 6.2: Performance of all ANN^{M}s and ANN^{S}s on the three images used: (a)-(d) on Image A (fig. 5.2 (a)), (e)-(h) on Image B (fig. 5.2 (b)) and (i)-(l) on Image C (fig. 5.2 (c)).

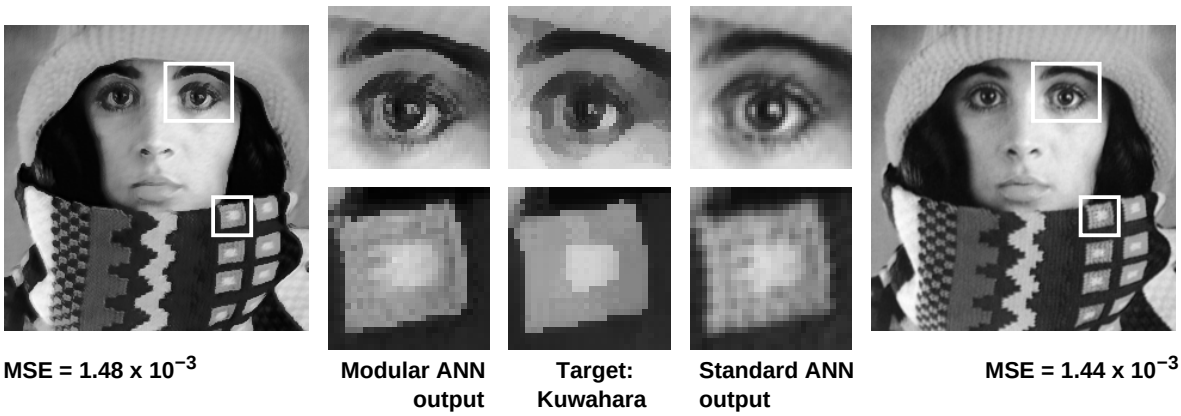


Figure 6.3: Two ANN output images with details. For the left image, output of $\mathrm{ANN}_4^{\mathrm{M}}$ trained on the edge-favouring set, the MSE is 1.48×10^{-3}; for the right image, output of $\mathrm{ANN}_{1\times100}^{\mathrm{S}}$ trained on a normally sampled set, it is 1.44×10^{-3}. The details in the middle show the target output of the Kuwahara filter; the entire target image is shown in figure 5.2 (a).

samples taken from regions around edges, the task of finding the mean is harder to learn due to the larger variation. At the same time, it eases training the position-of-minimum and selection modules. For all tasks except the average, the final MSE on the edge-favouring testing set (figures 6.1 (b), (d), (f) and (h)) is better than that of ANNs trained using a normal training set. The MSE is, in some cases, even lower on the normal testing set (figures 6.1 (e) and (g)).

Overall results for the modular and standard ANNs (figure 6.2) suggest that performance *decreases* when ANNs are trained on a specially selected data set (i.e., the MSE increases). However, when the quality of the filtering operation is judged by looking at the filtered images (see e.g. figure 6.3), one finds that these ANNs give superior results in approximating the Kuwahara filter. Clearly, there is a discrepancy between performance as indicated by the MSE and visual perception of filter quality. Therefore, the next section will investigate the possibility of finding another way of measuring performance for this application.

6.3 Performance measures for edge-preserving smoothing

The results given in section 6.2.1 show that it is very hard to interpret the MSE as a measure of filter performance. Although the MSEs differ only slightly, visually the differences are quite large. If images filtered by various ANNs trained on the normal

and edge-favouring data sets are compared, it seems clear which ANN performs better. As an example, figure 6.3 shows two filtered images. The left image was filtered by $\mathrm{ANN}_4^{\mathrm{M}}$ trained on an edge-favouring training set. The image on the right is the output of $\mathrm{ANN}_{1\times100}^{\mathrm{S}}$ trained on a normal data set. Although the MSEs are nearly equal (1.48×10^{-3} for the left image versus 1.44×10^{-3} for the right one), in the left image the edges seem much crisper and the regions much smoother than in the image on the right; that is, one would judge the filter used to produce the left image to perform better.

One would like to find a measure for filter performance which bears more relation to this qualitative judgement than the MSE. The reason why the MSE is so uninformative is that by far the largest number of pixels do not lie on edges. Figure 6.4 (a) illustrates this: it shows that the histogram of the gradient magnitude image is concentrated near zero, i.e. most pixels lie in flat regions. Since the MSE averages over all pixels, it may be quite low for filters which preserve edges poorly. Vice versa, the visual quality of the images produced by the ANNs trained using the edge-favouring data set may be better while their MSE is worse, due to a large number of small errors made in flat regions.

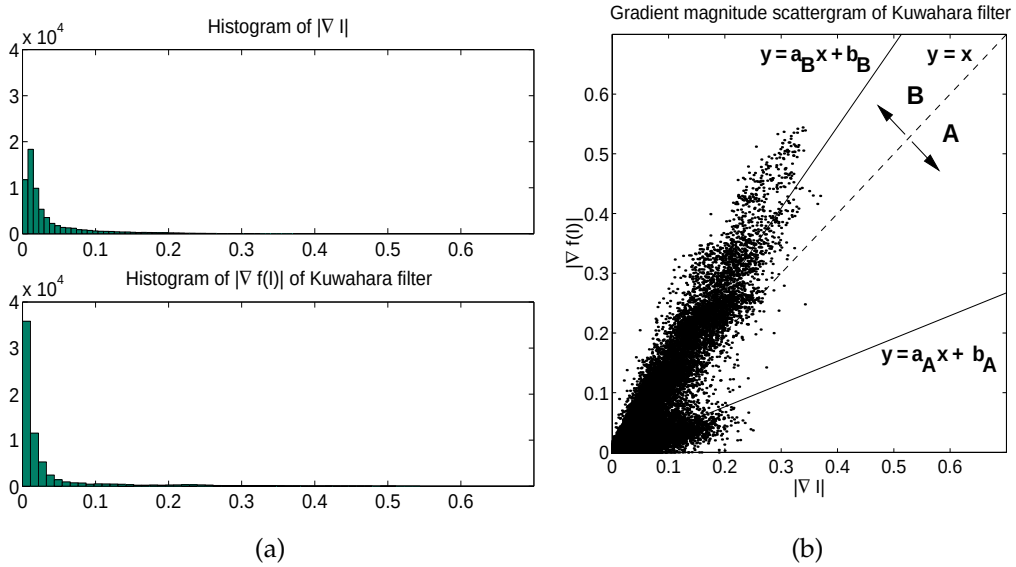
The finding that the MSE does not correlate well with perceptual quality judgement is not a new one. A number of alternatives have been proposed, among which the mean absolute error (MAE) seems to be the most prominent one. There is also a body of work on performance measures for edge detection, e.g. Pratt's Figure of Merit (FOM) [282] or Average Risk [337]. However, none of these capture the dual goals of edge sharpening and region smoothing present in this problem.

6.3.1 Smoothing versus sharpening

In edge-preserving smoothing, two goals are pursued: on the one hand the algorithm should preserve edge sharpness, on the other hand it should smooth the image in regions that do not contain edges. In other words, the gradient of an image should remain the same in places where it is high[2] and decrease where it is low.

If the gradient magnitude $|\nabla I|$ of an image I is plotted versus $|\nabla f(I)|$ of a Kuwahara-filtered version $f(I)$, for each pixel $I_{(i,j)}$, the result will look like figure 6.4 (b). In this figure, the two separate effects can be seen: for a number of points, the gradient is increased by filtering while for another set of points the gradient is decreased. The steeper the upper cloud, the better the sharpening; the flatter the lower cloud, the better the smoothing. Note that the figure gives no indication of the density of both clouds: in general, by far the most points lie in the lower cloud, since more pixels lie in smooth regions than on edges. The graph is reminiscent of the scattergram approach discussed

[2] Or even increase. If the regions divided by the edge become smoother, the gradient of the edge itself may increase, as long as there was no *overshoot* in the original image. Overshoot is defined as the effect of artificially sharp edges, which may be obtained by adding a small value to the top part of an edge and subtracting a small value from the lower part [385].



(a) (b)

Figure 6.4: (a) Histograms of gradient magnitude values $|\nabla I|$ of Image A (figure 5.2 (a)) and a Kuwahara filtered version ($k = 3$). (b) Scattergram of the gradient magnitude image pixel values with estimated lines.

(and denounced) in [189], but here the scattergram of the gradient magnitude images is shown.

To estimate the slope of the trend of both clouds, the point data is first separated into two sets:

$$\mathcal{A} = \left\{ (|\nabla I|_{(i,j)}, |\nabla f(I)|_{(i,j)}) \;\middle|\; |\nabla I|_{(i,j)} \geq |\nabla f(I)|_{(i,j)} \right\} \tag{6.1}$$

$$\mathcal{B} = \left\{ (|\nabla I|_{(i,j)}, |\nabla f(I)|_{(i,j)}) \;\middle|\; |\nabla I|_{(i,j)} < |\nabla f(I)|_{(i,j)} \right\} \tag{6.2}$$

Lines $y = ax + b$ can be fitted through both sets using a robust estimation technique, minimising the absolute deviation [284], to get a *density-independent* estimate of the factors with which edges are sharpened and flat regions are smoothed:

$$(a_{\mathcal{A}}, b_{\mathcal{A}}) = \arg\min_{(a,b)} \sum_{(x,y)\in\mathcal{A}} |y - (ax + b)| \tag{6.3}$$

$$(a_{\mathcal{B}}, b_{\mathcal{B}}) = \arg\min_{(a,b)} \sum_{(x,y)\in\mathcal{B}} |y - (ax + b)| \tag{6.4}$$

The slope of the lower line found, $a_{\mathcal{A}}$, gives an indication of the smoothing induced by the filter f. Likewise, $a_{\mathcal{B}}$ gives an indication of the sharpening effect of the filter. The offsets $b_{\mathcal{A}}$ and $b_{\mathcal{B}}$ are discarded, although it is necessary to estimate them to avoid a bias in the estimates of $a_{\mathcal{A}}$ and $a_{\mathcal{B}}$. Note that a demand is that $a_{\mathcal{A}} \leq 1$ and $a_{\mathcal{B}} \geq 1$, so the values are clipped at 1 if necessary – note that due to the fact that the estimated trends are not forced to go through the origin, this might be the case.

To account for the number of pixels actually used to estimate these values, the slopes found are weighed with the relative number of points in the corresponding cloud. Therefore, the numbers

$$Smoothing(f, I) = \frac{|\mathcal{A}|}{|\mathcal{A}| + |\mathcal{B}|}(a'_{\mathcal{A}} - 1) \quad \text{and} \tag{6.5}$$

$$Sharpening(f, I) = \frac{|\mathcal{B}|}{|\mathcal{A}| + |\mathcal{B}|}(a_{\mathcal{B}} - 1) \tag{6.6}$$

are used, where $a'_{\mathcal{A}} = \frac{1}{a_{\mathcal{A}}}$ was substituted to obtain numbers in the same range $[0, \infty\rangle$. These two values can be considered to be an amplification factor of edges and an attenuation factor of flat regions, respectively.

Note that these measures cannot be used as *absolute* quantitative indications of filter performance, since a higher value does not necessarily mean a better performance; i.e., there is no absolute optimal value. Furthermore, the measures are highly dependent on image content and scaling of $f(I)$ w.r.t. I. The scaling problem can be neglected however, since the ANNs were trained to give output values in the correct range. Thus, for various filters $f(I)$ on a certain image, these measures can now be compared, giving an indication of *relative filter performance on that image*. To get an idea of the range of possible values, smoothing and sharpening values for some standard filters can be calculated, like the Kuwahara filter, a Gaussian filter

$$f_G(I, \sigma) = I \otimes \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{6.7}$$

for[3] $\sigma = 0.0, 0.1, \ldots, 2.0$; and an unsharp masking filter

$$f_U(I, k) = I - k \times \left(I \otimes \begin{pmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{pmatrix}\right) \tag{6.8}$$

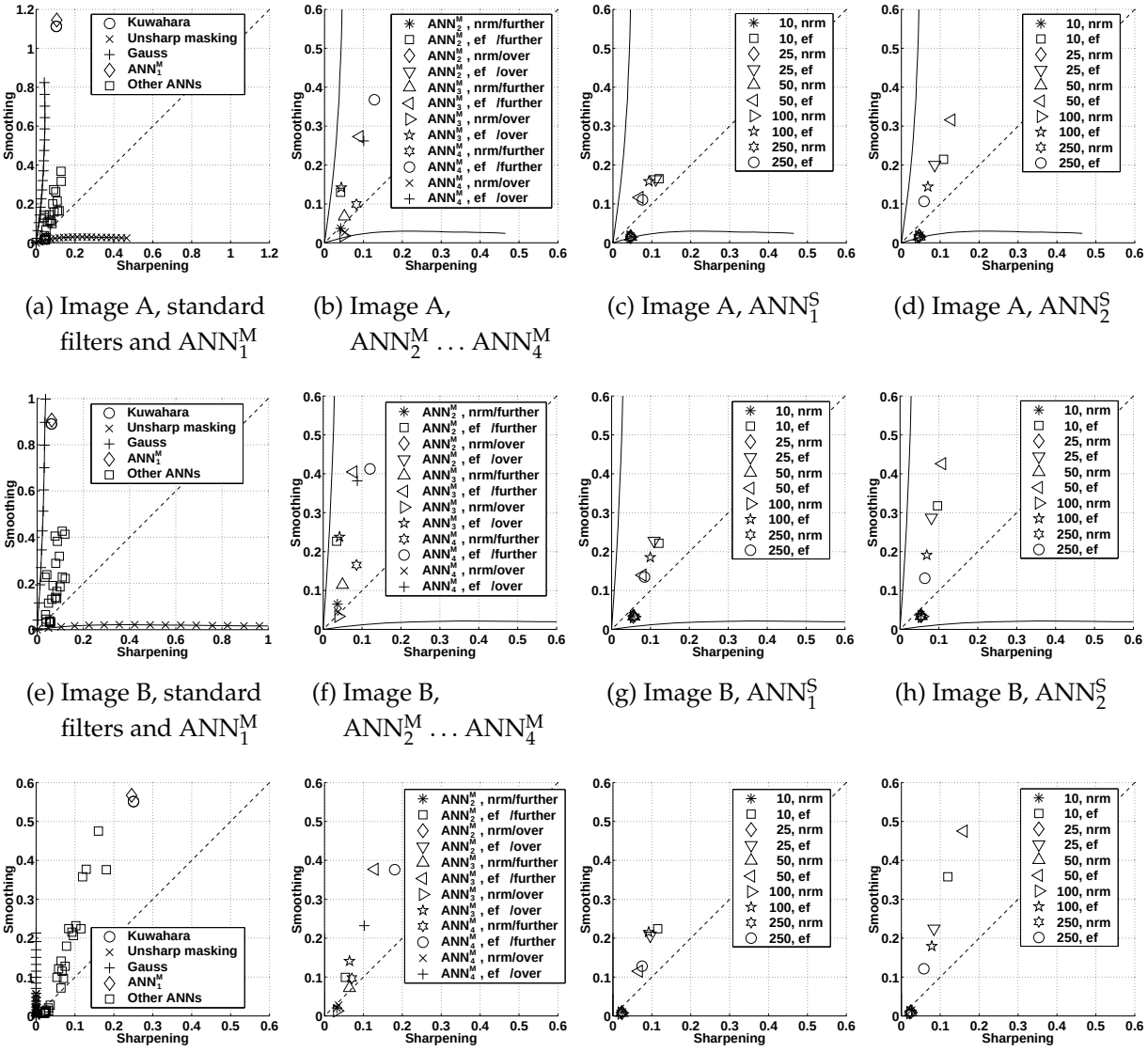
which subtracts k times the Laplacian[4] from an image, $k = 0.0, 0.1, \ldots, 2.0$.

6.3.2 Experiments

Smoothing and sharpening performance values were calculated for all ANNs discussed in section 6.2.1. The results are shown in figure 6.5. First, lines of performance values for the Gaussian and unsharp masking filters give an indication of the range of possible

[3] For $\sigma \leq 0.5$ the Gaussian is ill-sampled; in this case, a discrete approximation is used which is not stricly speaking a Gaussian.

[4] This is an implementation of the continuous Laplacian edge detector mentioned in section 4.2.1, different from the discrete detector shown in figure 4.1.



(a) Image A, standard filters and ANN_1^M

(b) Image A, $ANN_2^M \ldots ANN_4^M$

(c) Image A, ANN_1^S

(d) Image A, ANN_2^S

(e) Image B, standard filters and ANN_1^M

(f) Image B, $ANN_2^M \ldots ANN_4^M$

(g) Image B, ANN_1^S

(h) Image B, ANN_2^S

(i) Image C, standard filters and ANN_1^M

(j) Image C, $ANN_2^M \ldots ANN_4^M$

(k) Image C, ANN_1^S

(l) Image C, ANN_2^S

Figure 6.5: Performance of standard filters, all ANN^Ms and ANN^Ss on the three images used: (a)-(d) on Image A (fig. 5.2 (a)), (e)-(h) on Image B (fig. 5.2 (b)) and (i)-(l) on Image C (fig. 5.2 (c)). In the legends, *ef* stands for ANNs trained on edge-favouring data sets, as opposed to normally sampled data sets (*nrm*); *further* indicates ANNs initialised by training the individual modules as opposed to ANNs trained from scratch (*over*); and 10, 25 and so on denote the number of units per hidden layer.

values. As expected, the Gaussian filter on Images A and B (figures 5.2 (a) and (b)) gives high smoothing values and low sharpening values, while the unsharp masking filter gives low smoothing values and high sharpening values. The Kuwahara filter scores high on smoothing and low on sharpening. This is exactly as it should be: the Kuwahara filter should smooth while preserving the edges, it should not necessarily sharpen them. If ANNs have a higher sharpening value, they are usually producing overshoot around the edges in the output images.

The measures calculated for Image C (figure 5.2 (c)) show the limitations of the method. In this image there is a large number of very sharp edges in an otherwise already rather smooth image. For this image the Gaussian filter gives only very low smoothing values and the unsharp masking filter gives no sharpening value at all. This is due to the fact that for this image, subtracting the Laplacian from an image produces a very small sharpening value, together with a *negative* smoothing value, caused by the Laplacian greatly enhancing the amount of noise in the image. Since the values were clipped at 0, the results are not shown in the figure.

Regarding the ANNs, some things become clear. First, the hand-constructed ANN (ANN_1^{M}) almost perfectly mimics the Kuwahara filter, according to the new measures. However, as soon as the hand-set weights are dropped (ANN_2^{M}), performance drops drastically. Apparently the non-standard transfer functions and special architecture inhibits the ANN too much. ANN_3^{M} and ANN_4^{M} perform better and generalise well to other images. However, besides ANN_1^{M}, no other ANN in this study seems to be able to approximate the Kuwahara filter well. The best *trained* ANN still performs much worse.

Second, edge-favouring sampling has a strong influence. Most of the architectures discussed only perform reasonably when trained on a set with a significantly larger number of edge samples than acquired by random sampling, especially the ANN^{S}s. This indicates that, although the MSE actually indicates ANNs trained on an edge-favouring set perform worse, sampling in critical areas of the image is a prerequisite for obtaining a well-performing, nonlinear approximation to the Kuwahara filter.

Most standard ANNs perform poorly. Only for $\text{ANN}_{2\times10}^{\text{S}}$, $\text{ANN}_{2\times25}^{\text{S}}$ and $\text{ANN}_{2\times50}^{\text{S}}$ performance is reasonable. In retrospect, this concurs with the drop in the MSE that can be seen in figure 6.2 (d), although the differences there are very small. $\text{ANN}_{2\times50}^{\text{S}}$ clearly performs best. A hypothesis is that this depends on the training of the ANNs, since training parameters were not optimised for each ANN. To verify this, the same set of standard ANNs was trained in experiments in which the weights were initialised using random values drawn from a uniform distribution over the range $[-1.0, 1.0]$, using a learning rate of 0.5. Now, the optimal standard ANN was found to be $\text{ANN}_{2\times25}^{\text{S}}$, with all other ANNs performing very poorly.

Generalisation is, for all ANNs, reasonable. Even on Image C (figure 5.2 (c)), which differs substantially from the training image (Image A, figure 5.2 (a)), performance is

quite good. The best standard ANN, $\text{ANN}^{\text{S}}_{2\times 50}$, seems to generalise a little better than the modular ANNs.

6.3.3 Discussion

In [87, 88] it is shown that the smoothing and sharpening performance measures proposed here correlate well with human perception. It should be noted that in this study, subjects had less problems in discerning various levels of smoothing than they had with levels of sharpening. This indicates that the two measures proposed are not equivalently spaced.

The fact that the measures show that edge-favouring sampling in building a training set increases performance considerably, suggests possibilities for extensions. Pugmire et al. [286] claim that learning should be structured, i.e. start with the general problem and then proceed to special cases. This can be easily accomplished in training set construction, by adding a constant to each pixel in the gradient magnitude image before scaling and using it as a probability density function from which window coordinates are sampled. If this constant is gradually lowered, edge-pixels become better represented in the training set. Another, more general possibility would be to train ANNs on normally sampled data first and calculate an error image (such as those shown in the centre column of figure 5.8). Next, the ANN could be trained further – or re-trained – on a data set sampled using the distribution of the errors the ANN made, a new error image can be calculated, and so on. This is similar to boosting and arcing approaches in classification [33, 320]. An advantage is that this does not use the prior knowledge that edges are important, which makes it more generally applicable.

6.4 Training using different criteria

Ideally, the sharpening and smoothing performance measures discussed in the previous section should be used to train ANNs. However, this is infeasible as they are not differentiable. This means they could only be used in learning procedures which do not need the criterion function to be differentiable, such as reinforcement learning [140, 381]. This falls outside the scope of the experiments in this chapter.

However, the previous section showed that ANNs did learn to emulate the Kuwahara filter better when trained using the edge-favouring data set. Note that constructing a data set in this way is equivalent to using a much larger data set and weighing the MSE with the gradient magnitude. Therefore, this approach is comparable to using an adapted error criterion in training the ANN. However, this weighting is quite specific to this problem.

In the literature, several more general alternatives to the MSE (eqn. 2.12) have been proposed [36, 148]. Among these, a very flexible family of error criteria based on the L_p norm is:

$$E_p(\mathbf{W}, \mathbf{B}) = \frac{1}{2|\mathcal{L}|} \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{L}} \sum_{k=1}^{m} |R(\mathbf{x}^i;\ \mathbf{W}, \mathbf{B})_k - y_k^i|^p, \tag{6.9}$$

where $p \in \mathbb{Z}^*$. Note that for $p = 2$, this criterion is equal to the MSE. For $p = 0$, each error is considered equally bad, no matter how small or large it is. For $p = 1$, the resulting error criterion is known as the mean absolute error or MAE. The MAE is more robust to outliers than the MSE, as larger errors are given relatively smaller weights than in the MSE. For $p > 2$, larger errors are given more weight, i.e. the data is considered not to contain outliers. In fact, which p to use should be decided by assuming a noise model for the target data [36]. The L_1 norm (robust to outliers) corresponds to a noise distribution with large tails, a Laplacian distribution [378], under which outliers are probable. At the other extreme, L_∞ corresponds to a uniform noise distribution.

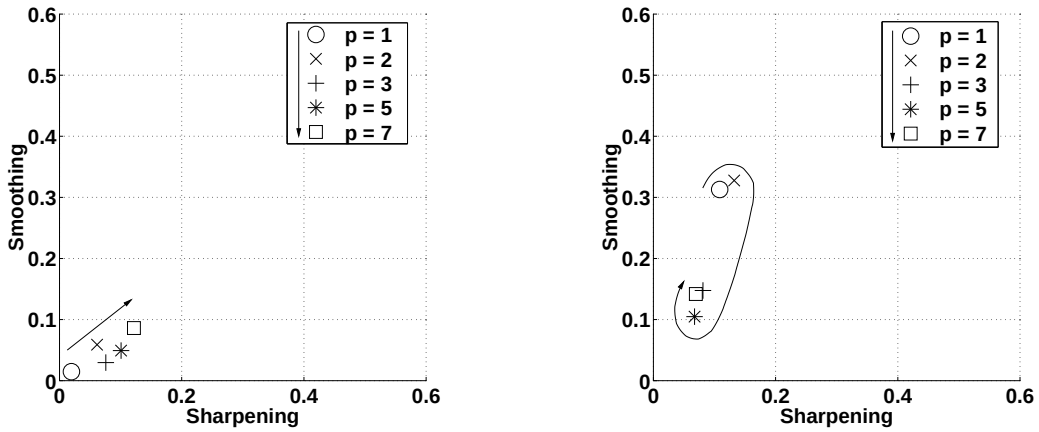
As discussed before, the Kuwahara filter is most interesting around the edges in an image, were the filter behaves nonlinearly. It was also shown that exactly around these edges most ANNs make the largest errors (figure 5.8). Therefore, it makes sense to use an error criterion which puts more emphasis on larger errors, i.e. the L_p norm for $p > 2$. To this end, a number of experiments were run in which different norms were used.

Although implementing these criteria in the back-propagation algorithm is trivial (only the gradient calculation at the output units changes), the modified algorithm does not converge well using standard settings. The learning rate and initialisation have to be adapted for each choice of norm, to avoid divergence. Therefore, the norms were used in the CGD training algorithm (see section 4.2.2), which is less sensitive to initialisation and choice of criterion due to the line minimisation involved.

6.4.1 Experiments

The best performing ANN found in section 6.3, $\text{ANN}^{\text{S}}_{2\times50}$, was trained using CGD with the L_p norm. The parameter p was set to $1, 2, 3, 5$ and 7, and both the normal and the edge-favouring training sets were used. The ANN was trained using the same settings as before (see section 5.4.2); in the CGD algorithm, directions were kept conjugate for 10 iterations.

Figure 6.6 shows the results. Clearly, using the L_p norm helps the ANN trained on the normal set to achieve better performance (figure 6.6 (a)). For increasing p, the sharpening performance becomes higher. However, the smoothing performance still lags behind that of the ANN trained using the MSE on the edge-favouring training set (fig. 6.5 (d)).



(a) (b)

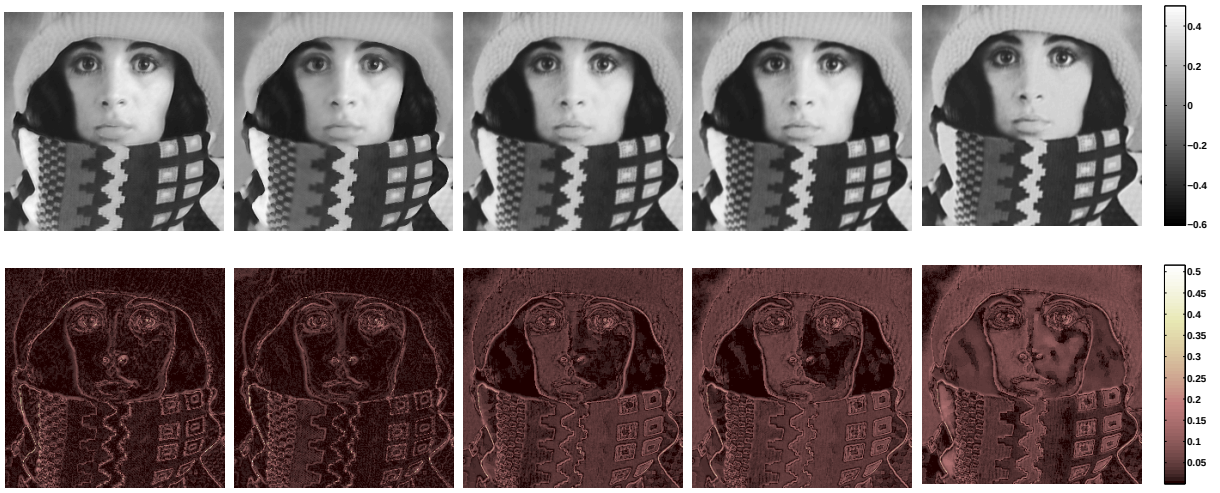
Figure 6.6: Performance of $\text{ANN}^{\text{S}}_{2\times50}$ on Image A (fig. 5.2 (a)), trained using different L_p norm error criteria and (a) the normal training set and (b) the edge-favouring training set.

When $\text{ANN}^{\text{S}}_{2\times50}$ is trained using the L_p norm on the edge-favouring data set, smoothing performance actually *decreases* (figure 6.6 (b)). This is caused by the fact that the training set and error criterion in concert stress errors around edges so much, that the smoothing operation in flat regions suffers. Figure 6.7 illustrates this by showing the output of $\text{ANN}^{\text{S}}_{2\times25}$ as well as the absolute difference between this output and the target image, for various values of p. For increasing p, the errors become less localised around the edges; for $p \geq 3$ the error in flat regions becomes comparable to that around edges.

In conclusion, using different L_p norms instead of the MSE can help in improving performance. However, it does not help as much as edge-favouring sampling from the training set, since only the latter influences the error criterion exactly where it matters, around edges. Furthermore, it requires choosing a value for the parameter p, for which an optimal setting is not clear beforehand. Finally, visual inspection still shows $p = 2$ to be the best choice.

6.5 Inspection of standard networks

To gain insight into the relatively poor performance of most of the standard ANNs according to the performance measure introduced in section 6.3, a very simple architecture was created, containing only a small number of weights (see figure 6.8 (a)). Since the Kuwahara filter should be isotropic, a symmetric weight mask was imposed on the weights (cf. section 10). Furthermore, linear transfer functions were used to avoid the complications introduced in the analysis by the use of sigmoids. No bias was used. This ANN was trained on the normal data set, using a validation set. The learned weight set



(a) $p = 1$ (b) $p = 2$ (c) $p = 3$ (d) $p = 5$ (e) $p = 7$

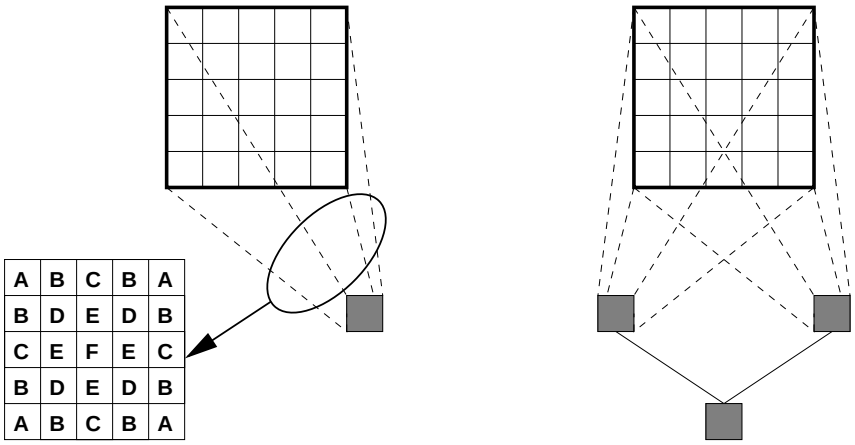
Figure 6.7: Top row: output of $\text{ANN}^{\text{S}}_{2\times50}$ trained using the L_p norm on the edge-favouring data set, for various p. Bottom row: absolute difference between output and target image.

is shown in figure 6.9 (a). In filtering terms, the main component looks like a negative Laplacian-of-Gaussian (i.e. the negative values around the centre and the slightly positive values in the four corners). Further analysis showed that this filter closely resembles a linear combination of a normal Gaussian and a Laplacian-of-Gaussian.

To confirm the hypothesis that standard ANNs learned such linear approximations to the Kuwahara filter, a simple standard ANN was trained in the same way ANN^{K} was, using the DCGD training algorithm (section 4.4). This ANN, $\text{ANN}^{\text{S}}_{1\times2}$, is shown in figure 6.8 (b). All weights were initialised to a fixed value of 0.01, λ was set to 1 and the number of directions to be kept conjugate was set to 10. After training, the MSE on the testing set was 1.43×10^{-3}, i.e. comparable to other standard ANNs (fig. 5.7), and C^2 was 5.1×10^{-3}. The resulting weight sets show that the filter can indeed be decomposed into a Gaussian-like and a negative Laplacian-like filter. Adding more hidden units and training using DCGD, for which results are not shown here, did not cause any new filters to be found.

This decomposition can well be explained by looking at the training objective. The Kuwahara filter smoothes images while preserving the edges. The Gaussian is a smoothing filter, while its second derivative, the Laplacian, emphasises edges when subtracted from the original. Therefore, the following model for the filter found by the ANN was set up:

$$\begin{aligned} f(c_1, \sigma_1, c_2, \sigma_2) &= c_1 f_G(\sigma_1) - c_2 f_L(\sigma_2) \\ &= c_1 \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right) - c_2 \frac{(x^2+y^2)-\sigma_2^2}{2\pi\sigma_2^6} \exp\left(-\frac{x^2+y^2}{2\sigma_2^2}\right) \end{aligned} \quad (6.10)$$



(a) (b)

Figure 6.8: (a) ANN^{K}, the most simple linear ANN to perform a Kuwahara filtering: a 5×5 unit input layer and one output unit without bias. The ANN contains 6 independent weights indicated in the mask by the letters **A** through **F**. (b) $\text{ANN}^{\text{S}}_{1\times 2}$: two hidden units, no mask (i.e., no restrictions).

in which c_1 and σ_1 are parameters to be estimated for the Gaussian and c_2 and σ_2 are parameters for the Laplacian. Figure 6.9 (c) shows these two functions.

A Gauss-Newton fitting procedure [179] was used to find the parameters of $f(c_1, \sigma_1, c_2, \sigma_2)$ given the weights shown in figure 6.9 (a). The resulting model weights are shown in figure 6.9 (b) and a cross-section is shown in figure 6.9 (c). Although the fit ($c_1 = 10.21$, $\sigma_1 = 2.87$, $c_2 = 3.41$, $\sigma_2 = 0.99$) is not perfect with a model fit MSE $\varepsilon_{\text{f}} = 2.5 \times 10^{-3}$, the correlation between the model and the actual weights is quite high ($C = 0.96$).

The hypothesis was that this solution, i.e. applying a Gaussian and a Laplacian, was a local minimum to which the ANN^{S}s had converged. To test this, the model fitting procedure was applied to each of the units in the first hidden layer of each of the ANN^{S}s. This resulted in a model fit error ε_{f} and correlation C between the actual weights and the model weights for each unit.

6.5.1 Experiments

The results, given in figure 6.10 show that, at least for the smaller ANNs, the hypothesis is supported by the data. For the ANNs trained on the normal data set, over a large range of sizes (i.e. 1-5, 10 and 25 hidden units) the model closely fits each hidden unit. Only for larger numbers of hidden units the fit becomes worse. The reason for this is

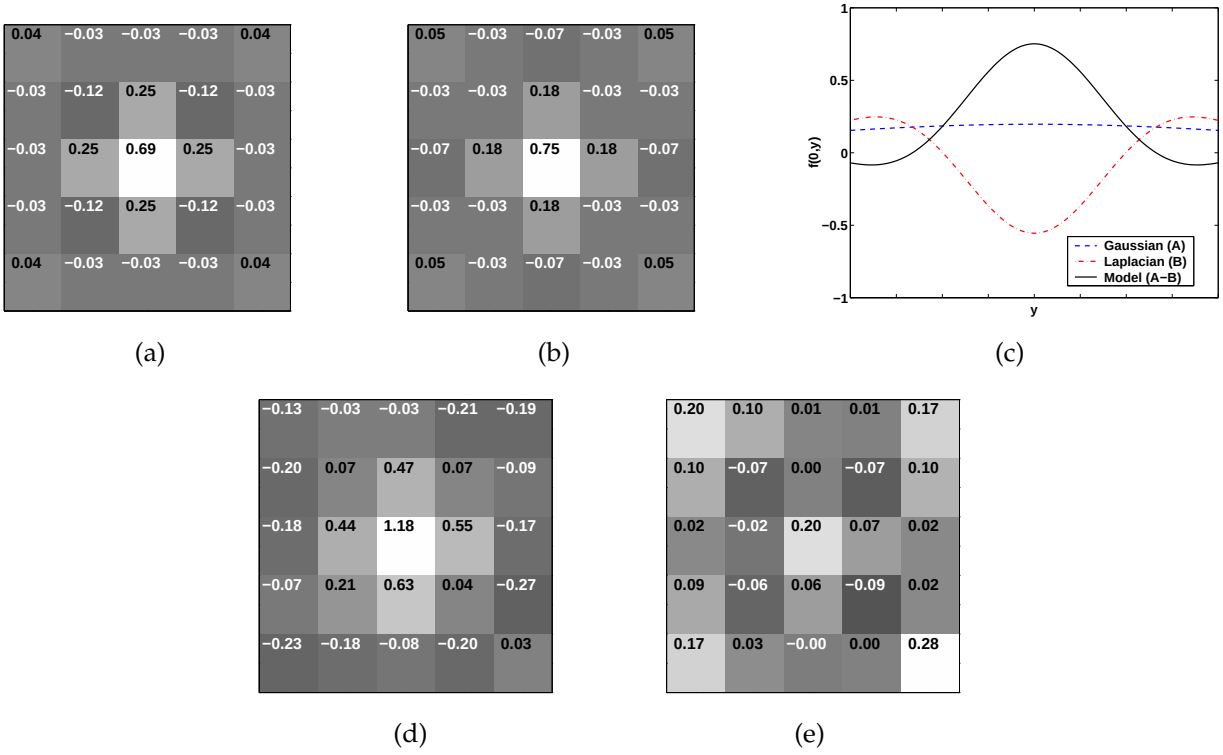
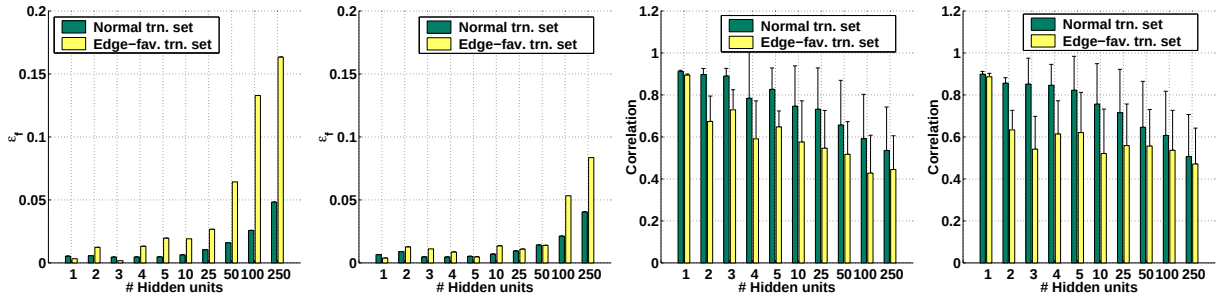


Figure 6.9: (a) Weights found in ANN^{K} (fig. 6.8 (a)). (b) Weights generated by the fitted model (eqn. 6.10: $c_1 = 10.21$, $\sigma_1 = 2.87$, $c_2 = 3.41$, $\sigma_2 = 0.99$). (c) A cross section of this model at $x = 0$. (d), (e) Weight matrices found in $\text{ANN}^{\text{S}}_{1\times 2}$ (fig. 6.8 (b)) trained using DCGD.



(a) ANN^{S}_1, median ε_f (b) ANN^{S}_2, median ε_f (c) ANN^{S}_1, average C (d) ANN^{S}_2, average C

Figure 6.10: A comparison between the actual weights in ANN^{S}s and the fitted models, for both ANN^{S}_1s and ANN^{S}_2s. The median ε_f is shown in (a) and (b) as the average ε_f is rather uninformative due to outliers.

that in these ANNs many units have an input weight distribution which is very hard to interpret. However, these units do not play a large role in the final ANN output, since they are weighted by small weights in the next layer.

For the ANNs trained on the edge-favouring set the fit is less good, but still gives a reasonable correlation. Note however that ANNs which have high performance w.r.t. the smoothing and sharpening measures (section 6.3.2) do not necessarily show the lowest correlation: ANN^{S}s with more hidden units give even lower correlation. An opposite effect is playing a role here: as ANNs become too large, they are harder to train.

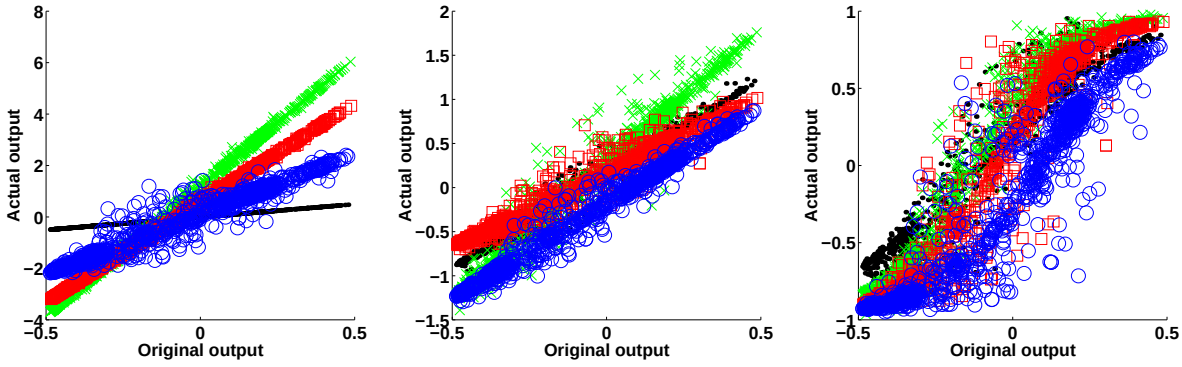
The conclusion is that many of the standard ANNs have learned a linear approximation to the Kuwahara filter. Although this approximation performs well in uniform regions, its output does not correspond to that of the Kuwahara filter near edges.

6.6 Inspection of modular networks

It is interesting to see whether the modular ANNs still use their initialisation. Remember that to obtain good performance, the ANN^{M}s had to either be trained further after the modules were concatenated, or re-initialised and trained over (section 5.4.4). The question is whether the modules are still performing the functions they were initially trained on, or has the ANN – after being trained further for a while – found a better solution? To inspect the ANNs, first the modules were evaluated on the sets they were trained with. Next, the concatenated ANN^{M}s were taken apart and the modules were evaluated on the same sets. Figures 6.11 and 6.12 show some examples of such plots.

Unfortunately, detailed inspection is hard. Ideally, if each module was performing the function it was trained to perform exactly, each plot would show a straight line $y = x$. The plots show that this is, in most cases, not true. However, it is possible to make some general remarks about the differences between the various ways of training the ANNs. These differences are most clear for the mean and selection modules:

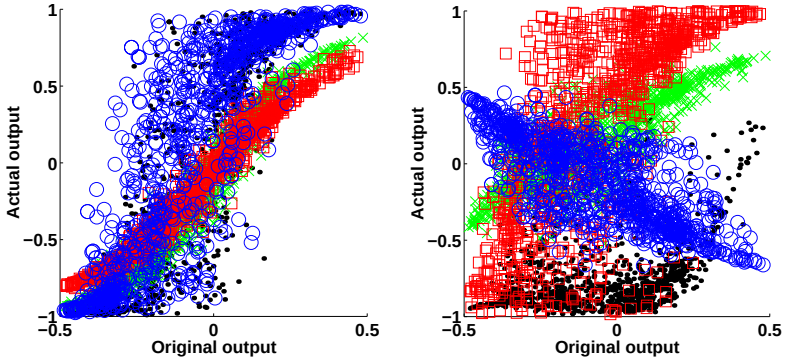
- for well-performing ANNs, the mapping in each module is no longer evident. Instead, it seems these modules make rather good use of their nonlinearity (figure 6.11 (c)). The poorly performing ANNs still show a reasonably linear behaviour (figure 6.12 (a));
- there is a progressive increase in nonlinearity for ANN^{M}_2, ANN^{M}_3 and ANN^{M}_4 (figures 6.11 (a)-(c), 6.12 (a)-(c) and (d)-(f)). The added complexity allows the modules more flexibility when they are trained further. Note however that the basic mapping is still preserved, i.e. the trend is still visible for all units;
- there is an increase in nonlinearity when ANNs are trained on the edge-favouring set instead of the normal set (figures 6.12 (a)-(c) v. (d)-(f));



(a) In ANN_2^M, trained further on normal set

(b) In ANN_3^M, trained further on normal set

(c) In ANN_4^M, trained further on normal set



(d) In ANN_4^M, trained further on edge-favouring set

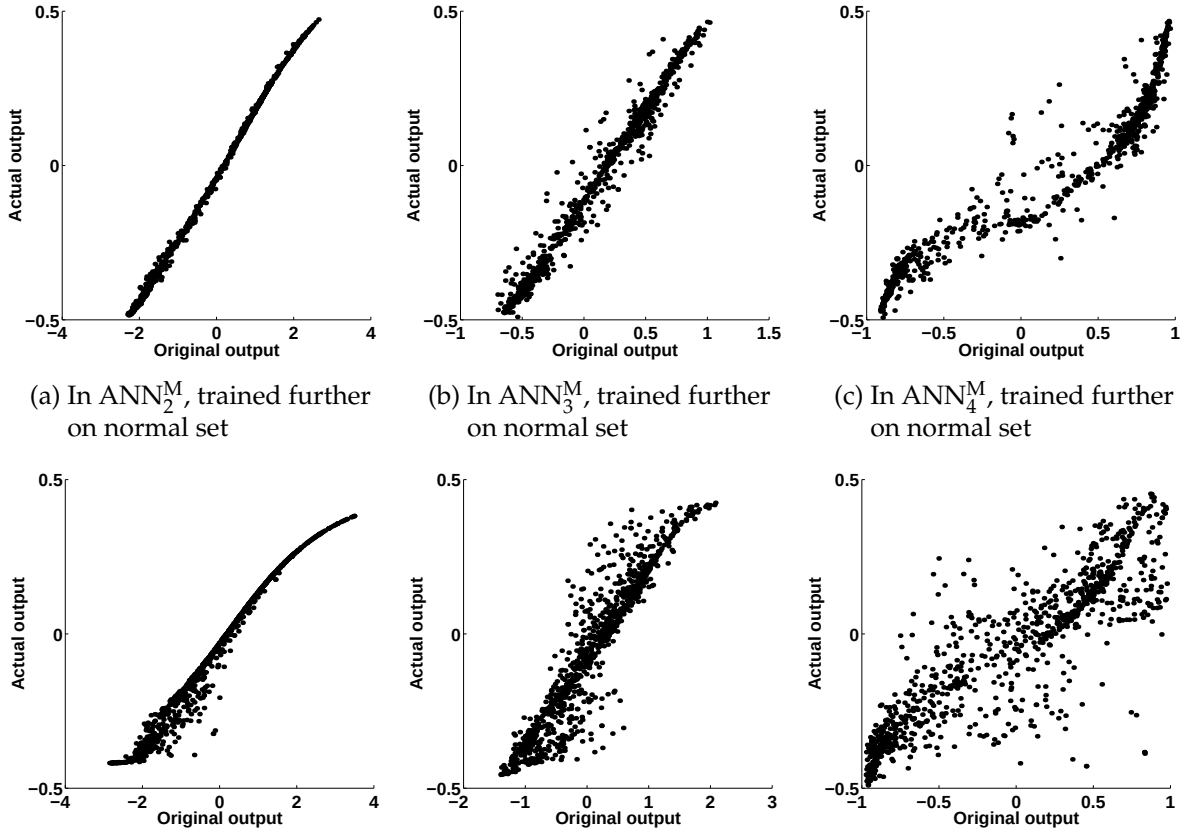
(e) In ANN_4^M, trained over on edge-favouring set

Figure 6.11: Plots of outputs of the four MOD^{Avg}s before concatenation against outputs of the same modules after concatenation and training further or over. Different markers indicate different output units. The plots show progressively more freedom as the modules become less restricted ((a)-(c)) and an increase in nonlinearity when modules are trained on the edge-favouring data set ((a)-(c) vs. (d)-(e)).

- as was to be expected, ANN^Ms trained from scratch generally do not find the modular structure (figures 6.11 (d)-(e)).

This leads to the conclusion that although the initialisation by training models individually was useful, the modules of the more well-performing ANNs are no longer performing their original function. This is likely to be caused by the modules being trained individually on ideal, noiseless data. Therefore, modules have not learned to deal with errors made by other modules. This is corrected when they are trained further together in the concatenated ANNs. The larger the correction, the better the final performance of the concatenated ANN.

For the MOD^{Var}s and MOD^{Pos}s, the differences are less clear. Most of these modules



(a) In ANN_2^M, trained further on normal set

(b) In ANN_3^M, trained further on normal set

(c) In ANN_4^M, trained further on normal set

(d) In ANN_2^M, trained further on edge-favouring set

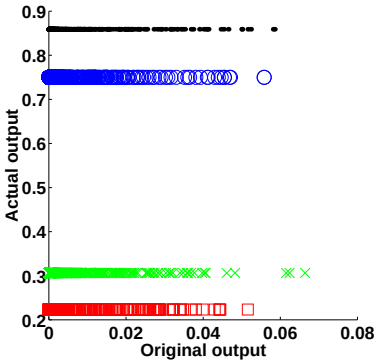
(e) In ANN_3^M, trained further on edge-favouring set

(f) In ANN_4^M, trained further on edge-favouring set

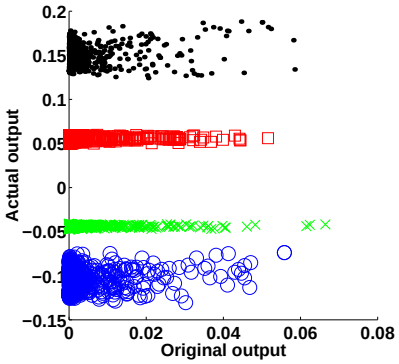
Figure 6.12: Plots of MOD^{Sel} outputs before concatenation against MOD^{Sel} outputs after concatenation and training further or over. The plots show progressively more freedom as the modules become less restricted ((a)-(c), (d)-(f)) and an increase in nonlinearity when modules are trained on the edge-favouring data set ((a)-(c) v. (d)-(f)).

seem to have no function left in the final ANNs: the outputs are clamped at a certain value or vary a little in a small region around a value. For MOD^{Var}, only ANN_4^M modules have enough flexibility. Here too, training on the edge-favouring set increases the nonlinearity of the output (figures 6.13 (a)-(c)). MOD^{Pos}, finally, is clamped in almost all architectures. Only ANN_4^M modules give some variation in output (figures 6.13 (d)-(e)). Networks trained from scratch are always clamped too.

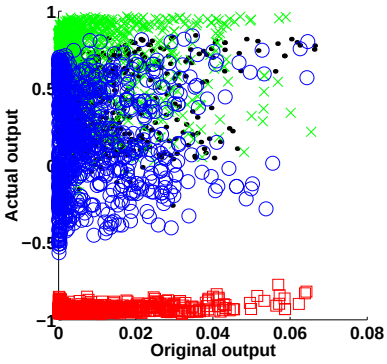
In conclusion, it seems that in most ANNs, the modules on the right hand side (MOD^{Var} and MOD^{Pos}, see figure 5.3) are disabled. However, the ANNs that do show some activity in these modules are the ANNs that perform best, indicating that the modular initialisation to a certain extent is useful. All results indicate that, although the nature of the algorithm can be used to construct and train individual modules, the errors these modules make are such that the concatenated ANNs perform poorly (see section 5.4.4). That



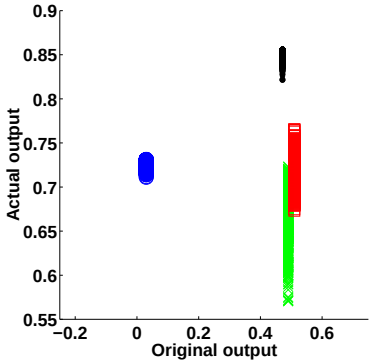
(a) MOD^{Var} in ANN_2^{M}, trained further on normal set



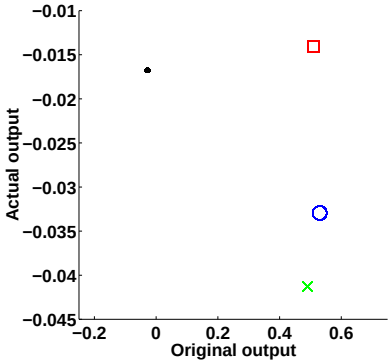
(b) MOD^{Var} in ANN_4^{M}, trained further on normal set



(c) MOD^{Var} in ANN_4^{M}, trained further on edge-favouring set



(d) MOD^{Pos} in ANN_4^{M}, trained further on normal set



(e) MOD^{Pos} in ANN_4^{M}, trained over on normal set

Figure 6.13: Plots of MOD^{Var} and MOD^{Pos} outputs before concatenation against the same outputs after concatenation and training further or over. Different markers indicate different output units. The plots show many module outputs in the concatenated ANNs are clamped at certain values. Note that in the latter two figures, the original output is either 0.0 or 0.5; a small offset has been added for the different units for presentation purposes.

is, modules trained separately on perfect data (e.g. pre-calculated positions of the minimal input) are ill-equipped to handle errors in their input, i.e. the output of preceding modules. For the concatenated ANNs, the training algorithm leaves its modular initialisation to lower the overall MSE; trained as a whole, different weight configurations are optimal. The fact that a trained MOD^{Pos} has a very specific weight configuration (with large weights) to be able to perform its function means it is more susceptible to weight changes than other modules and will easily lose its original functionality. In other words, the final concatenated ANN has "worked around" the errors made by MOD^{Pos} by disabling it.

6.7 Conclusions

The previous chapter discussed a number of experiments, in which modular and standard feed-forward ANNs were trained to mimic the Kuwahara filter. The main result was that all ANNs, from very simple to complex, reached the same MSE. A number of hypotheses was proposed for this phenomenon: that the data set and error measure may not accurately represent the finer points of this particular problem or that all ANNs have reached local minima, simply since the problem is too hard. Testing these hypotheses in this chapter, it was shown that:

- using a different way of constructing training sets, i.e. by mainly sampling from regions around the edges, is of great benefit;
- using performance measures which do not average over all pixels, but take the two goals of edge-preserving smoothing into account, gives better insight into relative filter performance;
- by the proposed smoothing and sharpening performance measures, which correspond better to visual perception, modular ANNs performed better than standard ANNs;
- using the L_p norm to train ANNs, with $p \gg 2$, improves performance, albeit not dramatically;
- the smaller ANN^{S}s have learned a linear approximation of the Kuwahara filter; i.e., they have reached a local minimum;
- in the poorly performing modular ANNs, the modules still perform the functions they were trained on. The better performing modular ANNs retain some of their initialisation, but have adapted further to a point that the function of individual modules is no longer clear. The better the performance of the final ANN (according to the new measure) the less clear the initialisation is retained.

In the attempts to try to understand the operation of an ANN instead of treating it like a black box, the interpretability trade-off (discussed in section 4.2.3) again played a role. For the modular ANNs, as soon as some of the constraints were dropped, ANN performance became much worse: there was no graceful degradation. It was shown too that it is hard to interpret the operation of the modular ANN after training it further; the operation of the ANN is distributed differently than in the original modular initialisation. The one advantage of using the prior knowledge of the modular nature of the problem (for example, as in ANN_4^{M}) is that it helps to avoid pain-staking optimisation of the number of hidden layers and units, which was shown to be quite critical in standard ANNs. Of course, for different problems this prior knowledge may not be available.

The main conclusion is that, in principle, ANNs *can* be put to use as nonlinear image filters. However, careful use of prior knowledge, selection of ANN architecture and

sampling of the training set are prerequisites for good operation. In addition, the standard error measure used, the MSE, will not indicate an ANN performing poorly. Unimportant deviations in the output image may lead to the same MSE as significant ones, if there is a large number of unimportant deviations and a smaller number of important ones. Consequently, standard feed-forward ANNs trained by minimising the traditional MSE are unfit for designing adaptive nonlinear image filtering operations; other criteria should be developed to facilitate easy application of ANNs in this field. Unfortunately, such criteria will have to be specified for each application (see also [337]). In this light it is not surprising to find a large number of non-adaptive, application-specific ANNs in the literature (see section 2.4.1).

Finally, although all performance measures used in this chapter suggest that ANNs perform poorly in edge-preserving smoothing, the perceptual quality of the resulting filtered images is quite good. Perhaps it is the very fact that these ANNs have only partially succeeded in capturing the nonlinearity of the Kuwahara filter that causes this. In some cases this could be considered an advantage: constrained nonlinear parametric approximations to highly nonlinear filtering algorithms may give better perceptual results than the real thing, which is, after all, only a means to an end.

As these chapters 3-6 have demonstrated that using supervised methods introduces a number of problems in choosing the training set, specifying an error criterion etc., the next chapter will move to unsupervised methods. The goal there is to have a method build up a model of the data, based solely on the data itself. Only afterwards will these methods then be used in subsequent classification or regression tasks. The expectation is that this will allow both easier incorporation of prior knowledge and offer better opportunities for interpretation.

7

SUBSPACE MODELS FOR FEATURE EXTRACTION

7.1 Introduction

When techniques from statistical pattern recognition, such as ANNs, are applied to image processing problems, images are often described as vectors in high-dimensional spaces. An image is a function $I(x, y)$ on a rectangular grid of pixel positions. For application of pattern recognition techniques, the function values are usually stored in a vector $\mathbf{x}$. As treating entire images in this way is computationally infeasible, images can be represented as distributions of d-dimensional vectors describing $w \times w$ pixel image patches as well (so $d = w^2$). However, the high-dimensional space in which these vectors reside will never be entirely filled, since images typically do not contain random configurations of pixels. The set of images that arise in practice and make sense to human observers is only a very small subset of all possible images. In well-sampled images, neighbouring pixels will be highly correlated, and coherent regions in images (e.g., textures) can be better described by individual local models in the high-dimensional space. In other words, a representation of images (or image patches) as vectors in a high-dimensional space contains far more parameters than needed.

Furthermore, one would often like to model image information locally invariant to offset, contrast, translation, rotation and scale. However, if an image patch is just slightly brightened, contrast enhanced, translated, rotated or scaled[1], the distribution of the vectors $\mathbf{x}$ will change drastically, whereas to a human observer the image content looks very similar. That is, the representation is not naturally invariant. Figure 7.1 illustrates this by showing an extreme case: when this texture is shifted by one pixel, its vector representation will jump to the exact opposite side of the d-dimensional hypercube on the surface of which all binary images lie.

[1]This latter set of operators – translation, rotation and scaling – will henceforth be referred to as *transformations* in this chapter and the next.

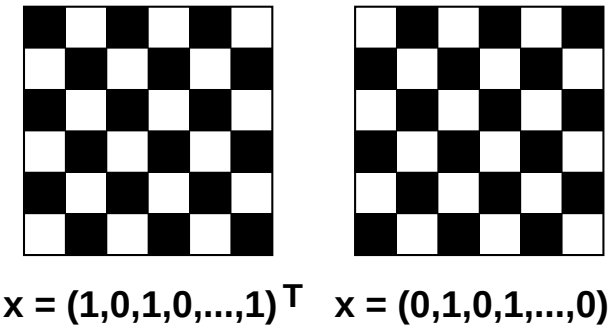


Figure 7.1: An example of the invariance problem in describing image patches by high-dimensional vectors **x**.

It can be shown that transformed versions of an image patch all lie on an m-dimensional manifold in the d-dimensional space spanned by all pixel values, where m is the number of degrees of freedom present in the transformations [224]. Although this manifold may be intrinsically low-dimensional, it is likely to be nonlinear and lie folded up in the d-dimensional space. A good representation would therefore be one which describes this manifold using a small number of parameters, thereby avoiding the estimation problems in high-dimensional spaces and enforcing invariance to elementary transformations. This chapter discusses methods of describing image patches by low-dimensional linear subspace models. Although there are nonlinear subspace models, e.g. auto-associator ANNs (see section 2), nonlinear principal component analysis [229], principal curves [340] and surfaces [145], these are often hard to compute in high-dimensional spaces and algorithms to find them cannot be guaranteed to converge. However, non-linear manifolds can be approximated using mixtures of linear subspace models. The past few years have seen an increase in interest in such mixture models. e.g. mixtures-of-PCA [151, 356] and mixtures-of-ICA [211, 213, 214].

In the remainder of this chapter, various models will be compared based on their application to feature extraction, particularly texture description, e.g. for subsequent segmentation. In chapter 8, mixtures of these models will be discussed and applied to various image processing problems. First, in section 7.2, a short overview of the ideas leading to this work is given and the basic setup is discussed. Section 7.3 describes the data used in texture description experiments later in the chapter. Next, section 7.4 discusses various models that may be used to describe image data: Gaussians, principal component analysis-based (PCA) and independent component analysis-based (ICA). Section 7.5 shows experiments comparing these models in terms of discriminating power and invariance w.r.t. the transformations mentioned above. The role of data normalisation, sample size and model parameters is investigated. Based on observations in these experiments, in section 7.6 the relative merits of ICA for texture description are discussed. Finally, section 7.7 ends with conclusions.

7.2 Overview

7.2.1 Previous work

The Adaptive Subspace SOM

The work presented in this chapter was inspired by a technique proposed by Kohonen in 1995 [197], the adaptive subspace self-organising map or ASSOM. This extension of the ordinary SOM (see section 2.2.2) uses subspaces S_i in each node i rather than just single weight vectors. Training is not done on just single samples but sets $\mathcal{E}$ of slightly translated, rotated and/or scaled signal or image samples, called *episodes*. These episodes are treated as a single entity, that is, samples are assigned as a group to a subspace based on a distance measure between an episode and a subspace, which is the minimum projection error of any sample $\mathbf{x} \in \mathcal{E}$:

$$D(\mathcal{E}, S_i) = \min_{\mathbf{x} \in \mathcal{E}} ||\mathbf{x} - \hat{\mathbf{x}}_j||, \tag{7.1}$$

where $\hat{\mathbf{x}}_j$ is the orthogonal projection of $\mathbf{x}$ onto subspace S_j.

To train the ASSOM, samples drawn from a signal or an image are converted into episodes by creating slightly transformed versions of the original sample. The distance between each node and the episode is then calculated, and the winning node is defined as that node to which the episode has minimum distance. In the adaptation phase, the winning node's subspace, and that of its neighbours, is rotated to better fit the just presented episode.

The ASSOM gives good results (see figure 7.2), but is extremely slow in training. This is caused not only by learning subspaces by rotating them, which demands careful and prudent setting of learning parameters, but also by updating neighbourhoods to obtain a topologically correct map. If these demands are dropped, i.e. by calculating the subspaces in a batch-mode operation (e.g. using PCA) and performing non-topologically correct clustering, the resulting system would be greatly simplified.

Mixtures of principal component analysers

An ASSOM-like system without the use of episodes and without the insistence on topological correctness comes close to the systems described by Bregler and Omohundro [32], Kambhatla and Leen [187], Hinton et al. [151], Roweis [304], Tipping and Bishop [355, 356] and Meinicke and Ritter [237]. These are all mixture-of-PCA algorithms, using different principal component analysis (PCA) formulations and clustering algorithms. Although the ideas have long been known (see e.g. [115]), only recently have they been applied to practical problems. Note that the discussion here is necessarily brief, as PCA will be discussed more in-depth in section 7.4.2.

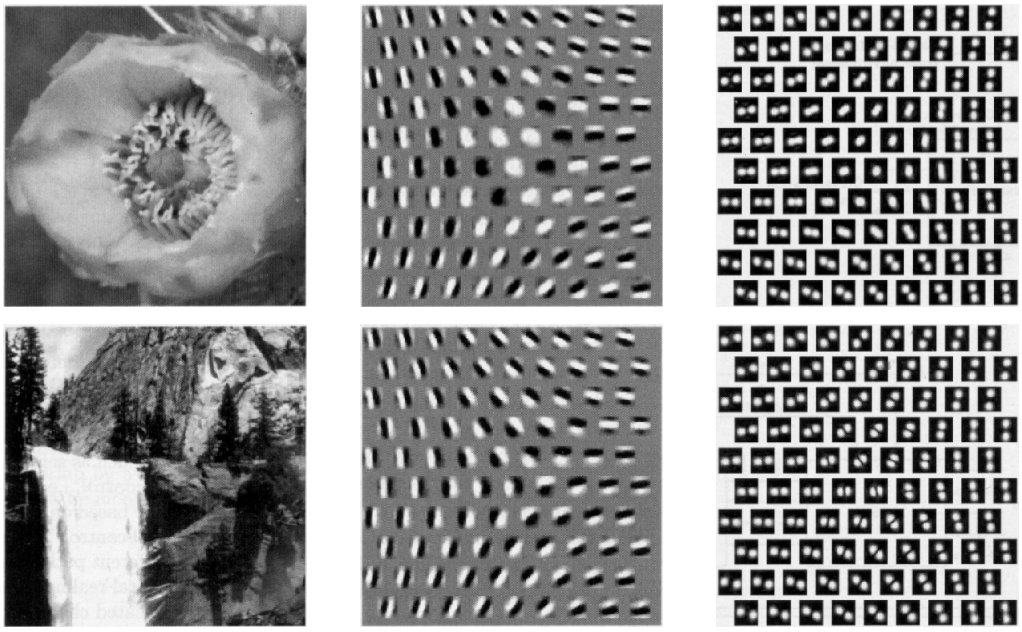


Figure 7.2: An ASSOM trained on the natural images in the left column. The middle column shows the first and second basis vector of each subspace; the right column shows their Fourier transforms. *Reproduced from [197].*

Bregler and Omohundro [32] propose a system in which local PCA bases are trained after clustering vectors using the k-means algorithm. This idea was extended by Kambhatla and Leen in their method [187], which they call VQPCA (for vector quantisation PCA). The algorithm consists of alternating steps of assigning vectors to clusters and re-calculating PCA bases for these clusters. Clustering, as in the ASSOM, is performed based on a subspace distance criterion rather than the Euclidean distance from vectors to cluster centers.

Hinton et al. [151] extend this model by linking PCA to factor analysis (FA). FA is a more general method than PCA [17, 100, 231], in that it assumes noise components outside a subspace[2] to be uncorrelated, but with different variance in each noise direction. FA models covariance (in the subspace) and variance (outside the subspace) independently, whereas PCA only models covariance and assumes noise outside the subspace to have identical variance in all directions. However, there is no closed form solution for finding FA bases; they can be found using, for example, the EM algorithm [28, 84]. By using latent variable methods proposed for FA and restricting the noise variance matrix, PCA models can be fitted in this way as well.

A fully probabilistic version of PCA was proposed by Roweis [304] and Tipping and Bishop [355], based on the same connection to FA. This model was later extended to a mixture model by Tipping and Bishop [356]. The entire mixture model is trained

[2] In this chapter, "outside the subspace" will be used to denote the nullspace [343] of a subspace, i.e. the $(d - m)$ dimensional space perpendicular to that subspace.

using the EM algorithm. All parameters except the number of models (subspaces) and the number of dimensions per model are optimised automatically. In a variation on this algorithm, Meinicke and Ritter [237] allow the method to optimise the number of dimensions per model, by specifying the allowed noise variance.

Note that these probabilistic formulations actually require a few assumptions which might not always be fulfilled in practical applications. The most important of these is the assumption of equal noise variance in all dimensions outside the subspace, which sets PCA apart from FA. Although this is part of the classical PCA model, it is never used in traditional algorithms. For probabilistic formulations of PCA, on the contrary, the noise has to be modelled explicitly.

Mixtures of independent component analysers

In principle, using the EM algorithm, it is quite straightforward to construct methods for training mixtures of arbitrary subspace models, as long as they can be formulated as a maximum likelihood (ML) problem. The EM algorithm [84] for mixture models works by assigning samples to individual models based on current probability density estimates (the E-step) and re-calculating each model's parameters based on the set of samples currently assigned to it (the M-step). The EM algorithm will be discussed in more detail in section 8.2.2 (for mixtures-of-PCA).

Over the last years, independent component analysis or ICA has received much attention. ML-formulations were given by various authors [172, 175, 212, 227, 248] and a mixture-of-ICA method was proposed by Lee et al. [211, 213, 214]. However, the models in this mixture are constrained to have a number of dimensions equal to that of the original space, i.e. $m = d$. In appendix C, a learning rule is derived based on this mixture model which allows finding undercomplete ICA bases. The overcomplete case, where $m > d$, is discussed in [219].

7.2.2 Subspace mixture model elements

A common element of the techniques described above is the fact that the training algorithm necessarily consists of two steps, usually applied iteratively. First, data somehow has to be split into a number of clusters. Approaches so far have used k-means clustering, vector quantisation and the E-step in the EM algorithm. However, in principle any clustering method may be applied, as long as it is based on a given matrix of distances between vectors and clusters (models). In section 8.2, the mean shift clustering algorithm [47, 61] will be discussed, which automatically determines the number of models.

After the data has been clustered, models will have to be fitted to each of the clusters. Using hard clustering methods such as k-means, this is quite straightforward. In the EM

No.	Content	Album
S1	Herringbone weave	D16
S2	French canvas	D20
S3	Netting	D34
S4	Oriental straw cloth	D52
S5	Raffia looped to a high pile	D84
S6	Loose burlap	D103

(a) Structured textures

No.	Content	Album
N1	Grass lawn	D9
N2	Straw	D15
N3	Pressed calf leather	D24
N4	Beach pebbles	D54
N5	Handmade paper, grassy fiber	D110
N6	Plastic bubbles	D111

(b) Natural textures

Table 7.1: The two sets of 6 Brodatz texture images used. In the "Album" columns, the numbers as given in the Brodatz album [34] are shown. Figure 7.3 shows the texture images.

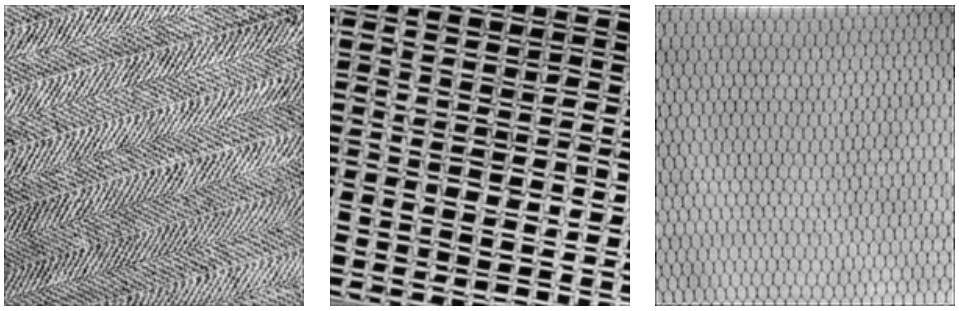
algorithm, the M-step re-calculates the model parameters based on weighted contributions from all data vectors. After the models have been found, distances between all data points and all models can be found again, after which the data can be re-clustered.

As the clustering and model fitting steps in most methods can be decoupled quite easily, they will be studied in isolation. In this chapter, the models will be discussed. First, in section 7.4 a number of possible models will be discussed in more detail. Experiments will then be performed in section 7.5 to learn about (predicted) performance of each model type as texture descriptors and the influence of various implementation choices. This knowledge will be used in chapter 8 to build mixture models for texture segmentation, object recognition and image database retrieval.

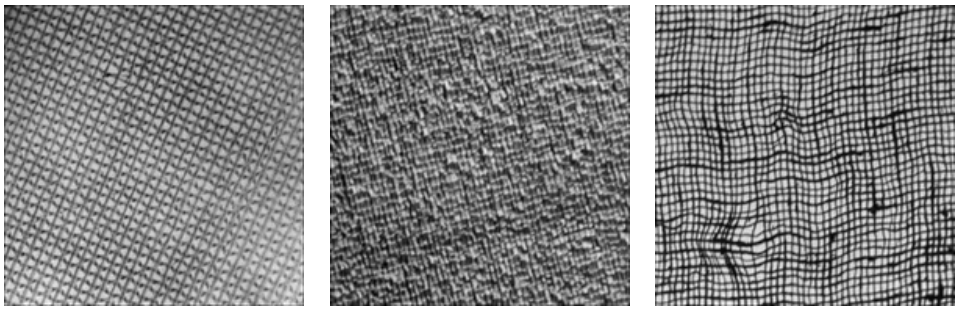
7.3 Texture data

Throughout this chapter, all models will be compared on their ability to describe texture. To this end, two sets of textures were taken from the Brodatz album [34], which is often used for segmentation evaluation. Table 7.1 gives details on the texture sets; figure 7.3 shows the individual texture images. The first set contains 6 structured textures, the second 6 more natural textures. It is to be expected that subspace methods will work better on the former, as these exhibit stronger correlation between pixels.

The original 256×256 pixel images were rescaled to a size of 192×192 pixels using bicubic interpolation. This was done to make sure texel sizes (i.e. the sizes of the "structuring elements") in the structured textures were smaller than 16×16 pixels, the maximum image patch size used in the experiments in this chapter. Furthermore, larger image sizes would have lead to more computation. For each image, the grey value range was rescaled to $[0, 1]$.

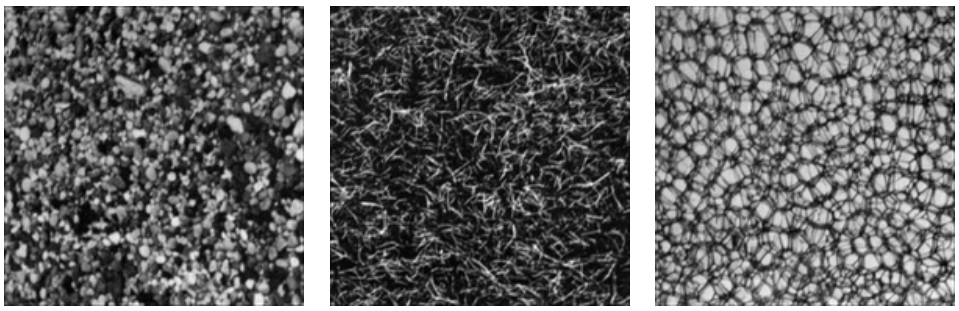


(a) S1 (b) S2 (c) S3



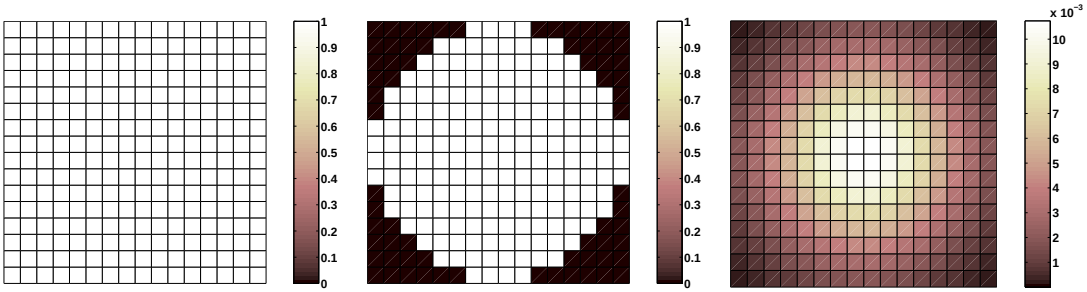
(d) S4 (e) S5 (f) S6

(g) N1 (h) N2 (i) N3



(j) N4 (k) N5 (l) N6

Figure 7.3: The 12 Brodatz textures used in the experiments in this chapter: structured textures (S1–S6) and natural textures (N1–N6). Table 7.1 gives details on both sets.



(a) Rectangular (b) Round (c) Gaussian

Figure 7.4: Three different window shapes used, shown here for a 16×16 pixel window size.

7.3.1 Data collection and episode construction

As the mixture models will be trained on image patches rather than entire images, training sets will have to be sampled from the images. Various choices play a role in this: window size and window shape. In the remainder of this chapter, a window size of 16×16 pixels will be used, although some experiments with smaller sizes (8×8 and 12×12 pixels) will be presented as well. Besides window size, the choice of window shape plays a role. The easiest way of sampling a data set from an image is to extract rectangular windows and place the pixel values row-wise in a vector. However, this introduces directional sensitivity into the training set, as larger diagonal structures can be represented than horizontal or vertical one. Therefore, some authors propose multiplying the windows by a round or Gaussian window shape (where $\sigma = \frac{1}{4}w$) to limit the effect of pixels far away from the window center [166, 196]. The three options are shown in figure 7.4. Note that using round window shapes, less pixels are retained than with the others (e.g. a round window shape in a 16×16 pixel window contains 188 pixels). The experiments in this chapter were performed using rectangular window shapes, unless indicated otherwise.

In all experiments in this chapter, the notion of episodes introduced by Kohonen (section 7.2.1) was used. This artificial enlargement of the data set allows for incorporation of prior knowledge, i.e. invariance over small translation, rotations and scales. This should not be necessary for large sets of images taken from real-life situations, as such sets should already contain all possible transformations of textures and structure that can occur in practice in that application. However, it allows for construction of invariant models based on a limited number of training images. Two kinds of episodes were used:

- *translation-only*: whenever a sample is extracted from the image at position (x_0, y_0), four other samples are extracted at positions $(x_0 + x, y_0 + y)$, where $x \sim U(-5, 5)$ and $y \sim U(-5, 5)$, giving a total of five samples per episode.

- *all-transformation*: next to a sample extracted at position (x_0, y_0), four translated samples are extracted as above; five samples are taken at the same position in images rotated over -45°, -22.5°, 0°, 22.5°and 45°; and five samples are taken at the same position in images scaled to 1.1×, 1.2×, 1.3×, 1.4× and 1.5× the original size. This gives a total of 15 samples per episode.

In most of the experiments, data sets of 1,500 samples *per texture* were created in this way, containing either 300 translation-only episodes or 100 all-transformation episodes, except where indicated otherwise. After data set construction, the notion of episodes is only used in clustering, i.e. episodes are assigned to clusters as a whole. For fitting individual models, this knowledge is not required.

Note that in all-transformation sampling, the invariances are incorporated only by themselves; that is, there are no samples that are translated, rotated as well as scaled. Although this might be beneficial, applying combinations of transformations would lead to a huge increase in episode size.

7.3.2 Normalisation and pre-mapping

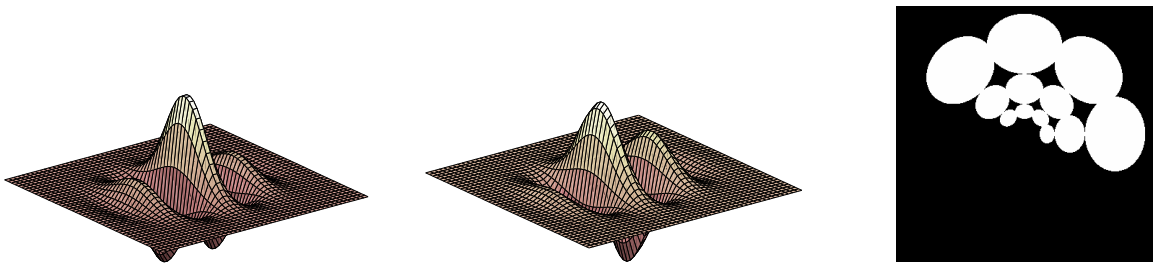
An important question that remains is whether the data should be pre-processed further. In [197], several steps are taken in training the ASSOM. First, samples are high-pass filtered by subtracting local averages. Next, a Gaussian window is applied, which is flattened during training. However, the goal of these experiments was to obtain wavelet-like basis vectors, as shown in figure 7.2.

Some pre-processing can be used to achieve invariance to illumination. In general, illumination differences are modelled by a gain and an offset, i.e. for two image windows I_A and I_B of identical content taken under different illuminations the following should hold:

$$I_B = c_1 I_A + c_2, \tag{7.2}$$

where c_1, c_2 are constants. If each sample is normalised by subtracting the mean grey value from each pixel and dividing each pixel by the standard deviation of grey values in the sampling window (i.e. equalising the length of each sample), an illumination invariant representation can be obtained. A major disadvantage of this approach is that any noise present in the image will be blown up due to the normalisation of standard deviation. Note that normalisation also has an effect on the type of model applicable; this will be discussed in section 7.4.2.

Finally, once entire data sets are created, these can optionally be pre-processed by removing directions in which there is little or no variance. These directions can be assumed to contain noise. This will be called *pre-mapping*. The algorithm used is PCA; the data is pre-mapped by projecting onto the eigenvectors corresponding to the set of largest eigenvalues explaining more than r of the variance, where $r = 90\%$ was used in



(a) (b)

Figure 7.5: (a) An example Gabor pair: the real and imaginary parts of the filter. (b) Coverage of the frequency domain by the 12-filter bank; due to the symmetry of Fourier transforms of real-valued images, only one half of the plane needs to be covered.

the experiments in this chapter. The data is not whitened, however (see section 7.4.2); the variances in the retained dimensions are as they were before. For 16×16 windows taken out of structured textures, this leaves 50 dimensions on average; out of natural textures, on average 70 dimensions remain.

7.3.3 The Gabor filter bank

To compare the model types to be introduced in the next section to a standard approach to texture description, a Gabor filter bank can be used. A Gabor filter is a complex filter in the image domain [31]:

$$g(x,y) = \left(\frac{1}{2\pi\lambda\sigma^2}\right) \exp\left(-\frac{(x'/\lambda)^2 + (y')^2}{2\sigma^2}\right) \exp\left(2\pi j(u_0 x + v_0 y)\right) \tag{7.3}$$

i.e. a Gaussian modulated by a complex sine wave. Here (x', y') is (x, y) rotated over an angle ϕ, i.e. $(x', y') = (x\cos\phi + y\sin\phi, -x\sin\phi + y\cos\phi)$ and (u_0, v_0) is the center frequency. Figure 7.5 (a) shows a pair of Gabor filters, i.e. the real and imaginary parts of $g(x, y)$.

The parameters of the Gabor filter can be found by specifying the filter's frequency f, orientation ϕ, radial frequency bandwidth B_f and orientation bandwidth B_ϕ:

$$\sigma = \sqrt{\ln\sqrt{2}} \Big/ \tan(\frac{1}{2}B_\phi)\pi f \tag{7.4}$$

$$\lambda = \sqrt{\ln\sqrt{2}}\left(2^{B_f}+1\right) \Big/ \pi f\sigma\left(2^{B_f}-1\right) \tag{7.5}$$

$$(u_0, v_0) = (f\cos\phi, f\sin\phi)\,. \tag{7.6}$$

Following [182] and [217], the following parameters were chosen: $f \in \{0.088388, 0.176777, 0.353553\}$, $\phi \in \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$, $B_f = 1$ and $B_\phi = \frac{\pi}{4}$. Before applying the Gabor filter, the local mean in a 7×7 pixel neighbourhood was subtracted from each pixel, to remove the DC component. Figure 7.5 (b) shows how the daisy petal-like frequency responses of this set of 12 filters occupy the frequency domain. Only 12 filters are necessary; because of the symmetry in the Fourier transform of real-valued images filters in the other half-plane do not add information. Note that the filter bank is rather coarse; especially the higher frequencies are covered poorly. For each of the 12 filters, the squared magnitude of its complex output is taken as a texture descriptor for each pixel. These can then be used in the same way that local image windows were used. In the experiments below, Gaussian models were used, both with mean-only and full covariance matrix (see section 7.4.1).

7.4 Models

This section discusses some basic models that can be applied in a mixture algorithm. Starting from the Gaussian model, principal component analysis (PCA) is introduced as a trade-off between modelling power and model complexity. Finally, independent component analysis (ICA) will be discussed as an alternative to PCA.

Throughout this section a data set of d-dimensional vectors $\mathbf{x}$ will be denoted by $\mathcal{L} = \{\mathbf{x}^n\}, n = 1, \ldots, N$. When subspace models are discussed, m will denote the number of dimensions of the subspace.

7.4.1 Gaussian

In a Gaussian model, $\mathbf{x}$ is modelled as follows:

$$p(\mathbf{x}|\Theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\det(\mathbf{C})|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{7.7}$$

where $\Theta = \{\boldsymbol{\mu}, \mathbf{C}\}$ is the set of parameters, $\boldsymbol{\mu}$ is the mean and $\mathbf{C}$ the covariance matrix over all $\mathbf{x} \in \mathcal{L}$. Different models can be constructed by constraining $\mathbf{C}$:

- $\mathbf{C}$ is a full covariance matrix, leading to an elliptic Gaussian;
- $\mathbf{C} = \boldsymbol{\Psi}$, a diagonal matrix, leading to an elliptic Gaussian with axis-aligned major axes, i.e. covariances are not taken into account, only variance in each dimension;
- $\mathbf{C} = \sigma^2\mathbf{I}$, a matrix with equal values on the diagonal, leads to a spherical Gaussian, ignoring covariances and the difference between variance in different dimensions;
- $\mathbf{C} = \mathbf{I}$: only the mean is used.

In the experiments in this chapter, only the two extreme cases are considered, i.e. the full-covariance Gaussian and the mean-only model. To estimate their parameters, the sample mean and covariance matrix can be used:

$$\hat{\boldsymbol{\mu}} = \bar{\mathbf{x}} = \mathrm{E}(\mathbf{X}) = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^n \tag{7.8}$$

$$\hat{\mathbf{C}} = \mathbf{S} = \mathrm{E}((\mathbf{X} - \mathrm{E}(\mathbf{X}))^2) = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}^n - \bar{\mathbf{x}})(\mathbf{x}^n - \bar{\mathbf{x}}). \tag{7.9}$$

Distance to a Gaussian

Distances to these models follow in a straightforward way from eqn. 7.7. The negative log-likelihood $-\ln p(\mathbf{z}|\Theta_G)$, or *normalised Mahalanobis distance* [348] between a vector $\mathbf{z}$ and the full Gaussian model G specified by Θ_G is defined as:

$$D(\mathbf{z}, G) = \frac{d}{2}\ln(2\pi) + \frac{1}{2}\ln|\det(\mathbf{C})| + \frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\mathbf{z}-\boldsymbol{\mu}), \tag{7.10}$$

which is the Mahalanobis distance normalised for the volume introduced by $\mathbf{C}$. For the mean-only model, the distance measure used is simply

$$D(\mathbf{z}, G) = \|\mathbf{z}-\boldsymbol{\mu}\|^2, \tag{7.11}$$

the squared Euclidean distance between $\mathbf{z}$ and $\boldsymbol{\mu}$.

When Gaussian models with full $\mathbf{C}$ are trained on data which does not fill the entire space, $\mathbf{C}$ will be poorly conditioned. That is, $|\det(\mathbf{C})|$ will be very small and $\ln|\det(\mathbf{C})| \to -\infty$. However, the standard Mahalanobis distance can still be used:

$$D_M(\mathbf{z}, G) = (\mathbf{z}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\mathbf{z}-\boldsymbol{\mu}), \tag{7.12}$$

Another possibility is to pre-map the data to retain a certain proportion of variance, say $r = 90\%$ (see section 7.3.2); in other words, the data set can be adapted to fit the model. Note that models trained in this pre-mapped space are not comparable to those trained in the original space.

7.4.2 Principal component analysis

Principal component analysis (PCA, [162, 231]) is a well-known linear technique for reducing the dimensionality of a data set. In short, PCA is a projection from d to m dimensions,

$$\mathbf{u} = \mathbf{W}(\mathbf{x}-\boldsymbol{\mu}), \tag{7.13}$$

which preserves as well as possible the retained variance. Here the $m \times d$ matrix $\mathbf{W}$ contains the PCA *projection vectors* as its rows. It can be shown (see e.g. [377]) that the m projection vectors that maximise the variance of $\mathbf{u}$, i.e. the *principal axes*, are given by the eigenvectors $\mathbf{e}_1 \dots \mathbf{e}_m$ of the sample covariance matrix (eqn. 7.9) corresponding to the largest non-zero eigenvalues $\lambda_1 \dots \lambda_m$. These vectors can be found by solving the set of equations

$$(\mathbf{S} - \lambda_i \mathbf{I})\mathbf{e}_i = 0,\ i = 1, \dots, d \tag{7.14}$$

(see e.g. [343]) and sorting the $\mathbf{e}_i$ by the associated eigenvalues λ_i. The PCA projection matrix is then $\mathbf{W} = \mathbf{E}^T$, where the columns of $\mathbf{E}$ contain the eigenvectors. Usually, the vectors $\mathbf{e}_i$ are first made orthonormal, so that the eigenvalues are proportional to the variance in the eigenvector directions. The proportion of variance retained by mapping down to m dimensions can therefore be found as the normalised sum of these m eigenvalues, i.e.

$$r = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{d} \lambda_i}. \tag{7.15}$$

Note that this can also be used to find the number of dimensions m required to retain at least a proportion r of the variance.

Besides maximising the retained variance, PCA has two other important properties:

- *decorrelation*: the projected data is decorrelated, i.e. $\mathrm{E}(\mathbf{U}\mathbf{U}^T)_{ij} = 0,\ \forall i \neq j$. This can easily be seen by the following observation. Let $\mathbf{W} = \mathbf{E}^T$ be the matrix containing the eigenvectors corresponding to all non-zero eigenvalues as its rows, and $\mathbf{\Lambda}$ be a matrix containing the associated eigenvalues λ on the diagonal. Then the covariance matrix of the projected data $\mathbf{U} = \mathbf{W}\mathbf{X}$, where $\mathbf{X} = \left[\mathbf{x}^1 \mathbf{x}^2 \dots \mathbf{x}^n\right]$, can be found to be:

  $$\mathrm{E}(\mathbf{U}\mathbf{U}^T) = \mathrm{E}((\mathbf{E}^T\mathbf{X})(\mathbf{X}^T\mathbf{E})) = \mathbf{E}^T\mathrm{E}(\mathbf{X}\mathbf{X}^T)\mathbf{E} = \mathbf{E}^T\mathbf{S}\mathbf{E}. \tag{7.16}$$

  But, as the eigenvector equation (eqn. 7.14) gives $\mathbf{S}\mathbf{E} = \mathbf{E}\mathbf{\Lambda}$, this is:

  $$\mathrm{E}(\mathbf{U}\mathbf{U}^T) = \mathbf{E}^T\mathbf{E}\mathbf{\Lambda} = \mathbf{I}\mathbf{\Lambda} = \mathbf{\Lambda}, \tag{7.17}$$

  since $\mathbf{E}$ is orthogonal. In other words, the covariance matrix of the projected data is a diagonal matrix – there are no correlations.

  Using an almost identical derivation, it can be shown that when the data is projected using $\mathbf{W} = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$, the data has equal variance in all dimensions as well:

  $$\mathrm{E}(\mathbf{U}\mathbf{U}^T) = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T\mathbf{E}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{\Lambda} = \mathbf{\Lambda}^{-1}\mathbf{\Lambda} = \mathbf{I}. \tag{7.18}$$

  This process is called *sphering* or *whitening*. PCA is not the only method to do this; any projection which results in $\mathrm{E}(\mathbf{U}\mathbf{U}^T) = \mathbf{I}$ is a sphering projection.

- *least squares reconstruction*: PCA projection minimises the squared reconstruction error. That is, if the projected vector $\mathbf{u} = \mathbf{W}(\mathbf{x} - \boldsymbol{\mu})$ is projected back into the original space as $\hat{\mathbf{x}} = \mathbf{A}\mathbf{W}(\mathbf{x} - \boldsymbol{\mu})$, where $\mathbf{A} = \mathbf{W}^T$ contains the *basis vectors*, then the squared reconstruction error $\|(\mathbf{x} - \boldsymbol{\mu}) - \hat{\mathbf{x}}\|^2$ is minimal. A PCA projection is the optimal projection in the least squares reconstruction sense.

  Note that the backprojection matrix $\mathbf{A}$ equals $\mathbf{W}^T$ only when $\mathbf{W}$ is orthogonal. If $\mathbf{W}$ just contains orthogonal vectors, the more general projection operator $\mathbf{A} = \mathbf{W}(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}^T$ should be used.

Distance to a PCA base

The distance of a vector $\mathbf{z}$ to a PCA subspace P specified by parameters $\Theta_P = \{\boldsymbol{\mu}, \mathbf{W}\}$ can be defined as the reconstruction error, i.e.:

$$D(\mathbf{z}, P)^2 = \|(\mathbf{z} - \boldsymbol{\mu}) - \mathbf{A}\mathbf{W}(\mathbf{z} - \boldsymbol{\mu})\|^2 = \|(\mathbf{z} - \boldsymbol{\mu}) - \hat{\mathbf{z}}\|^2, \tag{7.19}$$

where $\hat{\mathbf{z}}$ is now the result of projecting $(\mathbf{z} - \boldsymbol{\mu})$ onto the subspace. Note that for arbitrary $\mathbf{z}$ and P, this distance measure does not say anything about the distance *in* the subspace; it just says how far vectors lie *from* the subspace.

Here, the difference between working on original and normalised data (see section 7.3.2) comes into play. Normalised data will lie on a hypersphere of radius 1 around the origin, due to the length normalisation of each sample. From the other normalisation step, subtraction of the average pixel intensity in each sample, it does *not* follow that an entire normalised data set will have zero mean in the d-dimensional space. However, assuming all transformed versions of a sample (i.e., all linear combinations of subspace basis vectors with unit vector length) are equally likely, the prior distribution of the data set inside the subspace is uniform on the intersection of the subspace and the hypersphere. As a consequence, the origin of the subspace can be assumed to be zero, i.e. $\boldsymbol{\mu}$ can be fixed at $\mathbf{0}$ in algorithms trained on normalised data.

In this case, there is no need for a model inside the subspace, as new normalised vectors will also lie on the same hypersphere. Therefore, the squared length of a projected vector (inside the subspace) will be negatively proportional to the squared projection error (outside from the subspace), see eqn. 7.19. Measuring distance inside the subspace would not add any information.

For non-normalised data, the simplest approach is to assume a Gaussian model in the subspace and to devise a distance measure combining within-subspace distance and out-of-subspace distance. Attempts at such a distance have been made by, among others, Moghaddam and Pentland [245], Sung and Pong [348], Hinton et al. [151] and Tipping and Bishop [356]. Moghaddam and Pentland start by defining a distance measure as a composite of a distance-from-subspace D_f and distance-in-subspace D_i [245]. The basic idea is shown in figure 7.6.

The squared distance-from-subspace is simply the reconstruction error, eqn. 7.19. For the distance-in-subspace they start with the Mahalanobis distance in the original d-dimensional space (eqn. 7.12, assuming, without loss of generality, zero mean data, i.e. $\boldsymbol{\mu} = \mathbf{0}$):

$$D_M^2(\mathbf{z}) = \mathbf{z}^T \mathbf{C}^{-1} \mathbf{z}. \tag{7.20}$$

Due to the definition of PCA, this can be written as (assuming non-zero eigenvalues):

$$D_M^2(\mathbf{z}) = \mathbf{z}^T (\mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T)^{-1} \mathbf{z} = \hat{\mathbf{z}}^T \boldsymbol{\Lambda}^{-1} \hat{\mathbf{z}} = \sum_{i=1}^{d} \frac{\hat{z}_i^2}{\lambda_i}, \tag{7.21}$$

where $\mathbf{E}$ now is the $d \times d$ orthogonal matrix containing all eigenvectors, so $\mathbf{E}^{-1} = \mathbf{E}^T$. Now when taking only an m-dimensional subspace P into account, only the first m elements of $\hat{\mathbf{z}}$ will lie inside P. Denoting this part by $\hat{\mathbf{z}}_P$, the squared distance-in-subspace simply becomes:

$$D_i^2(\mathbf{z}, P) = D_M^2(\hat{\mathbf{z}}_P) = \sum_{i=1}^{m} \frac{\hat{z}_i^2}{\lambda_i}. \tag{7.22}$$

For the squared distance-from-subspace, note that eqn. 7.19 for P can be written as:

$$D_f^2(\mathbf{z}, P) = \|\mathbf{z} - \hat{\mathbf{z}}_P\|^2 = \|\mathbf{z}\|^2 - \|\hat{\mathbf{z}}_P\|^2 = \sum_{i=1}^{d} z_i^2 - \sum_{i=1}^{m} \hat{z}_i^2 = \sum_{i=m+1}^{d} \hat{z}_i^2. \tag{7.23}$$

So when the data is mapped to subspace P, eqn. 7.21 can be approximated by

$$D_M^2(\mathbf{z}, P) = D_i(\mathbf{z}, P)^2 + \beta D_f(\mathbf{z}, P)^2 = \sum_{i=1}^{m} \frac{\hat{z}_i^2}{\lambda_i} + \beta \sum_{i=m+1}^{d} \hat{z}_i^2. \tag{7.24}$$

The optimal constant β can be shown quite easily to be [245]

$$\beta = \frac{d - m}{\sum_{i=m+1}^{d} \lambda_i}, \tag{7.25}$$

i.e. the inverse of the average of the eigenvalues outside the subspace.

There are two serious problems with eqn. 7.24:

- The distance measure is not normalised like the Mahalanobis distance in eqn. 7.10, as was noted by Hinton et al. [151] and Sung and Poggio [348]. Whereas Hinton et al. add "a constant", Sung and Poggio give the only right solution. As for the Gaussian model (section 7.4.1), the Mahalanobis distance should be normalised to

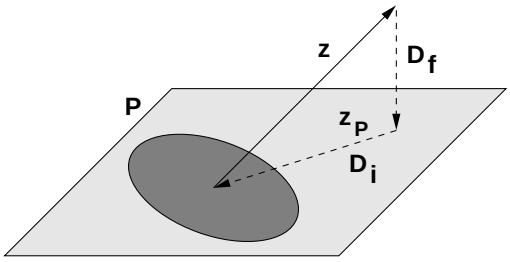


Figure 7.6: A PCA distance measure, composite of distance-from-subspace D_f and distance-in-subspace D_i.

the negative likelihood (dropping constants):

$$\begin{aligned} D_M^2(\mathbf{z}, P) &= \ln|\det(\mathbf{C})| + D_i^2(\mathbf{z}, P) + \beta D_f^2(\mathbf{z}, P) \\ &= \ln|\det(\mathbf{E}\mathbf{\Lambda}\mathbf{E}^T)| + D_i^2(\mathbf{z}, P) + \beta D_f^2(\mathbf{z}, P) \\ &= \ln|\det(\mathbf{\Lambda})| + D_i^2(\mathbf{z}, P) + \beta D_f^2(\mathbf{z}, P) \\ &= \sum_{i=1}^{d} \ln \lambda_i + \sum_{i=1}^{d} \frac{\hat{z}_i^2}{\lambda_i} + \beta \sum_{i=m+1}^{d} \hat{z}_i^2. \end{aligned} \tag{7.26}$$

- More importantly, in order to weight the distance *inside* the subspace (eqn. 7.22), the dimensions *outside* the subspace have to be taken into account (eqn. 7.23). This now imposes not only a model inside the subspace, but also outside: a Gaussian model with equal variance

$$\sigma^2 = \frac{1}{\beta} = \frac{1}{d-m} \sum_{i=m+1}^{d} \lambda_i. \tag{7.27}$$

  If the model does not hold, e.g. when the subspace model fits perfectly and $\lambda_i = 0$ for $i > m$, the distance measure blows up as β will be ∞.

In fact, the resulting model is exactly the same as those introduced by Roweis [304] and Tipping and Bishop [355]. In their approaches, the data set is modelled by a Gaussian with covariance matrix $\mathbf{C}$ constrained to $\mathbf{C} = \mathbf{A}\mathbf{A}^T + \sigma^2\mathbf{I}$, where the subspace basis $\mathbf{A}$ and the noise level σ^2 are learned. This shows how PCA can be seen as a trade-off between a full-covariance Gaussian model and a restricted Gaussian model with $\mathbf{C} = \sigma^2\mathbf{I}$ (section 7.4.1).

Implementation

Summing up, in the experiments performed in this chapter, two different distance measures were used:

- for PCA models trained on normalised data, the distance measure was simply the reconstruction error (eqn. 7.19);
- for PCA models trained on non-normalised data, the distance measure was the normalised distance-in-subspace/distance-from-subspace measure (eqn. 7.26).

A final question is what number of dimensions the subspace should have. A rough approximation would be to set the number of dimensions equal to the number of degrees of freedom that need to be modelled, e.g. 4 for translation in the x-direction and y-direction, rotation and scaling. However, this supposes that transformed textures can be described by a linear subspace, which need not be the case for two reasons:

- the texture itself may require more dimensions to be described well. Consider, for example, a 1D block wave signal of period p; a subspace would need to have p dimensions to describe this in a translation-invariant way;
- the transformations may lead to nonlinear subspaces, which – although low-dimensional – would need to be modelled using more dimensions when using a single linear subspace.

The question as to what number of dimensions is sufficient will be answered by performing experiments (section 7.5.6).

7.4.3 Independent component analysis

Lately, a technique different from PCA has gained considerable interest: independent component analysis or ICA [176]. In its most basic form, this method tries to find not just projection directions in which the data is uncorrelated, but directions in which the data is independent. The model is identical to PCA:

$$\mathbf{u} = \mathbf{W}(\mathbf{x} - \boldsymbol{\mu}). \tag{7.28}$$

However, as ICA was originally applied to blind separation of various signals (or *sources*) $\mathbf{s}$ under an additive model, here the matrix $\mathbf{W}$ is often called the *unmixing matrix* and backprojection matrix $\mathbf{A}$ the *mixing matrix*. This backprojection is defined, cf. PCA, as:

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \boldsymbol{\mu}. \tag{7.29}$$

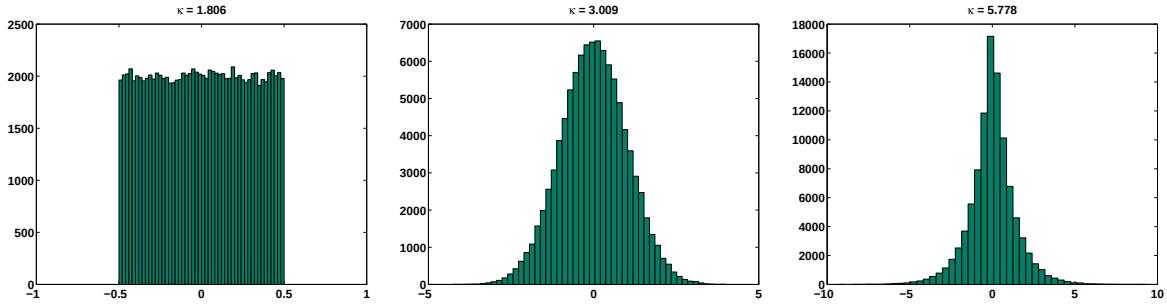
Note that in the notation here, $\mathbf{u}$ is used to denote an estimate of $\mathbf{s}$.

Clearly, independence, i.e.

$$\mathrm{E}(g_1(u_i)g_2(u_j)) = \mathrm{E}(g_1(u_i))\mathrm{E}(g_2(u_j)),\ \forall i \neq j, g_1(.), g_2(.), \tag{7.30}$$

where $g_1(.)$ and $g_2(.)$ are any measurable functions, is a stronger demand than the uncorrelatedness of PCA:

$$\mathrm{E}(u_i u_j) = \mathrm{E}(u_i)\mathrm{E}(u_j),\ \forall i \neq j, \tag{7.31}$$



(a) Uniform (b) Gaussian (c) Laplacian

Figure 7.7: Kurtoses κ calculated for a set of 100,000 points drawn from three distributions: (a) the uniform distribution, which is sub-Gaussian or platykurtotic; (b) the Gaussian distribution, which is mesokurtotic; and (c) the Laplacian distribution, which is super-Gaussian or leptokurtotic.

and cannot be obtained using at most second order information[3]. As such, there is no closed form expression to find an ICA base. Instead, many iterative algorithms have been proposed based on the following observations:

- independent source distributions should have a smaller differential entropy, i.e. $H(\mathbf{u}) = -\int f(\mathbf{u}) \ln f(\mathbf{u}) d\mathbf{u}$, than the Gaussian distribution [173];
- independent sources u_i should have as little mutual information, i.e. $I(u_1, \ldots, u_m) = \sum_{i=1}^{m} H(u_i) - H(\mathbf{u})$, about each other as possible [171];
- the Kullback-Leibler divergence between the factorised density $f_F(\mathbf{u}) = \prod_i f_i(u_i)$ and the true density $f(\mathbf{u})$, $\int f_F(\mathbf{u}) \ln \frac{f_F(\mathbf{u})}{f(\mathbf{u})} d\mathbf{u}$, should be minimal [5];
- independent sources are likely to be found by looking for non-Gaussian distributions in the projection, e.g. by specifying a non-Gaussian distribution and fitting it using maximum likelihood [20, 212].

Most of the observations listed above lead to similar or even identical algorithms [38, 175]. The general idea behind all of them is that distributions of the data projected onto an ICA basis vector should be as non-Gaussian as possible. This links ICA to projection pursuit [110, 111, 186], which often uses non-Gaussianity as a measure of "interestingness" of a projection. An intuitive reasoning is that, due to the central limit theorem which states that a sum of i.i.d. random variables will tend in the limit to have a Gaussian distribution, non-Gaussian projection distributions will indicate that the projection is not a sum of random variables but a single one.

A measure often used to judge the property of non-Gaussianity is the *(Pearson) kurtosis*

[3] Information contained in the covariance matrix of the data.

of a distribution $f(u)$, i.e. the central fourth-order moment:

$$\kappa_{f(u)} = \frac{\mu_4}{\mu_2^2} = \frac{\mathrm{E}((u')^4)}{\mathrm{E}((u')^2)^2} \ , \tag{7.32}$$

where $u' = u - \mathrm{E}(u)$. In this definition of kurtosis, the Gaussian distribution has a kurtosis of 3. More peaked distributions, such as the Laplacian, have a higher kurtosis, whereas more flat distributions, e.g. the uniform distribution, have a smaller kurtosis. Figure 7.7 illustrates this. Some algorithms simply maximise the absolute difference in kurtosis between the projection distribution and the Gaussian.

A limitation of ICA therefore is that Gaussian independent components (ICs) cannot be found. However, for Gaussian ICs simple decorrelation (as performed by PCA) also makes the components independent, as the Gaussian distribution is specified completely by the mean and covariance matrix (cf. section 7.4.1). When it is know that data is distributed according to a Gaussian, whitening will result in i.i.d. Gaussian projections.

Pre-whitening

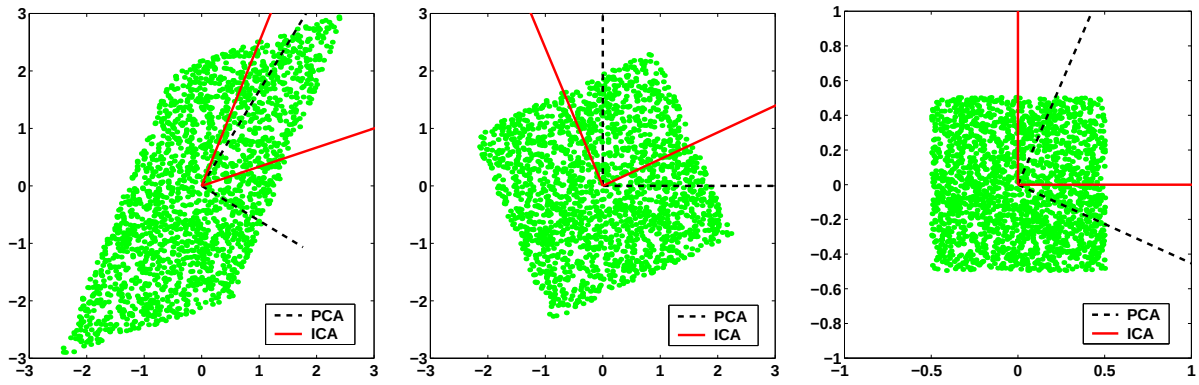
Whitening, e.g. by PCA, is often used as a pre-processing step for ICA. Some algorithms require this, to find independent component irrespective of differences in the variance of projected data. However, even when it is not required, it speeds up algorithms, because:

- the data is already decorrelated, which is a requirement for independence;
- some of the resulting properties of data and parameters can be used to speed up algorithms, such as the mixing matrix $\mathbf{W}$ being orthogonal and the variance being equal in all directions;
- as ICA is very noise sensitive, it is advisable to use PCA pre-processing for discarding dimensions of low variance (assumed to contain noise) as well. This again speeds up algorithms as there are fewer dimensions in the data presented to the ICA algorithm.

The consequence of pre-whitening is that the ICA projection itself is merely a rotation of the axes. Figure 7.8 gives an example for a 2D data set.

Independent component analysis on image data

The first major application of ICA was to the blind unmixing of signals (also known as *blind source separation* or the cocktail party problem), see section C.4.1. However, the ICA model can be applied to image data as well. In order to discuss its possible merits, the discussion in [22] is summarised here. The starting point for both PCA and ICA is the



(a) Original (b) After PCA pre-whitening (c) After ICA projection

Figure 7.8: A 2D uniform distribution with PCA and ICA bases, in three different spaces.

idea that image patches can be modelled as weighted sums of a number of underlying causes, cf. eqn. 7.29 (in the rest of this discussion, the mean image patch $\boldsymbol{\mu}$ is assumed to be $\mathbf{0}$). If a first demand on the causes is that they are uncorrelated, this means that for a projection matrix $\mathbf{W}$ the following should hold:

$$\mathbf{W}^T\mathbf{W} = \mathbf{C}^{-1}. \tag{7.33}$$

or, equivalently, that $\mathbf{A}\mathbf{A}^T = \mathbf{C}$. However, this demand does not uniquely define a solution; additional demands should be made. One demand is orthogonality of the rows of $\mathbf{W}$, which leads to PCA (eqn. 7.18). PCA projection vectors are global in the image domain and localised in the frequency domain (see figure 7.9); in fact, for stationary signals, PCA projection vectors can be shown to be the basis vectors of the Fourier transform [85, 104]. Another demand could be that $\mathbf{W}$ is a symmetric matrix, which leads to *zero-phase component analysis* (ZCA). In ZCA, $\mathbf{W} = \mathbf{C}^{-\frac{1}{2}}$, giving on-center off-surround projection vectors which are localised in the image domain, but global in the frequency domain (figure 7.9). ICA can be seen as a trade-off between these two opposites. The demand that the projections are not only uncorrelated, but also independent, gives filters which are localised both in frequency and space (figure 7.9). In this light, several authors have suggested links with wavelets [21, 22, 166, 167, 211, 214] and receptive fields found in the mammalian visual cortex [13, 104, 164, 263, 264].

Given the emerging filters, the idea is that ICA might be useful for image data which cannot be described well using the global, low-frequency PCA bases. This would be image data in which certain characteristic high-frequency elements regularly occur against an otherwise irregular background, such as the natural textures shown in figure 7.3 (b). However, although this idea has been expressed by many authors, a successful application of this aspect of ICA to image processing problems has not yet been published.

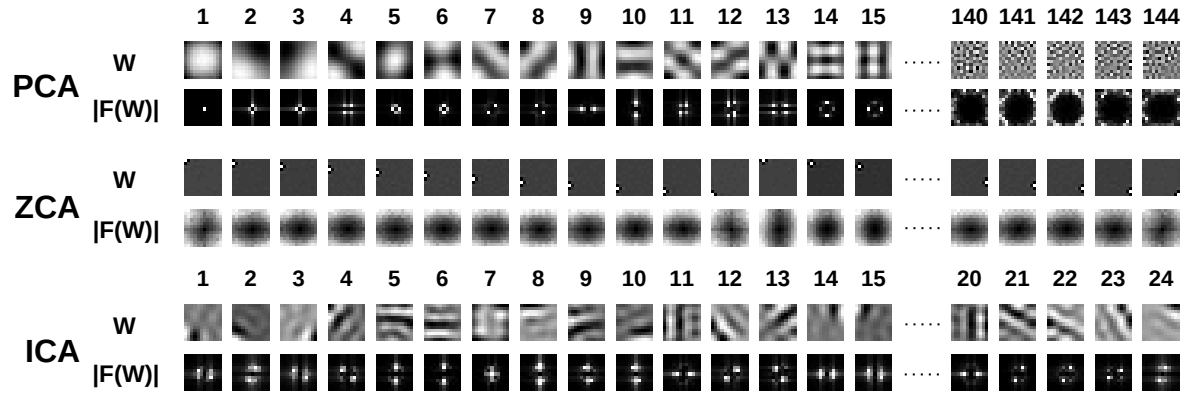


Figure 7.9: PCA, ZCA and ICA filters, or projection vectors, $\mathbf{W}$ found on 10,000 12×12 pixel patches taken from natural images (see figure C.6 on page 239) and the magnitude of their Fourier transforms.

Algorithm

An algorithm was developed for this work as a variation on the extended infomax ICA algorithm of Lee et al. [212]. This algorithm is a maximum likelihood (ML) fit of a set of distributions, in which a *switching matrix* $\mathbf{K}$ (also learned, see section C.1.2) decides whether to fit a sub-Gaussian or a super-Gaussian distribution to each dimension of the projected data $\mathbf{u} = \mathbf{W}\mathbf{x}$. These distributions are given by eqns. C.26 and C.30, in appendix C. An advantage of using an ML algorithm is that it can easily be extended to a mixture model [211, 214].

The main limitation of the original algorithm is the fact that it works only for $m = d$, i.e. to find as many ICs as there are dimensions in the original space. In the case of texture description, it is not to be expected that there are that many ICs. Therefore, a new learning rule was derived for the case where $m < d$, by modelling the remaining dimensions as Gaussian noise. Training is done using a generalised EM algorithm in which the estimate for $\mathbf{W}$ is updated by a gradient descent learning rule. This algorithm is worked out in detail in appendix C.

Distance to an ICA base

As for the other models, the distance of a point $\mathbf{z}$ to an ICA base I defined by $\Theta_I = \{\mathbf{W}, \boldsymbol{\mu}, \mathbf{C}, \mathbf{K}\}$ (see appendix C) is defined as the negative log-likelihood,

$$\begin{aligned} \mathbf{z}' &= \mathbf{C}^{-\frac{1}{2}}\mathbf{z} \\ D(\mathbf{z}, I) &= \frac{d}{2}\ln 2\pi + \frac{1}{2}\ln|\det(\mathbf{C})| + \frac{1}{2}\ln|\det(\mathbf{A}^T\mathbf{A}) \\ &\quad + \frac{1}{2}\left(\mathbf{z}' - \mathbf{A}\mathbf{u}\right)^T\left(\mathbf{z}' - \mathbf{A}\mathbf{u}\right) - \ln p(\mathbf{u}). \end{aligned} \tag{7.34}$$

cf. eqn. C.50 (page 231) in which $\beta = 1$ and $\mathbf{C}^{-\frac{1}{2}}$ is used to pre-whiten the data. The estimate of the sources is found as $\mathbf{u} = \mathbf{W}\mathbf{z}'$ and the matrix $\mathbf{A}$ is the pseudo-inverse of $\mathbf{W}$, i.e. $\mathbf{A} = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$. In this distance measure, the distribution of each dimension of $\mathbf{u}$ is chosen according to $\mathbf{K}$; the possible expressions are given by eqns. C.38 and C.39 on page 229.

7.5 Model experiments

In this section, a number of experiments is performed in which each texture in the Brodatz set (see section 7.3) is modelled by one model. To assess the power of each of the model types, a performance measure is introduced in section 7.5.1. Next, experimental results are shown and the effects of normalisation, the choice of subspace dimensionality and sample size are discussed in sections 7.5.2-7.5.5. Finally, in section 7.5.7 experiments are performed to see to what extent the models are truly invariant representations of the texture data.

Where applicable, the models introduced in the previous section are compared to models trained on features found using a Gabor filter bank (section 7.3.3). As for each pixel a 12D vector is found, the mean-only and full-covariance Gaussian models are applied without any changes to this data as well.

7.5.1 Measures

Texture segmentation quality can be evaluated in a number of ways (see, e.g., [391] and section 2.4.3 on page 25). However, as the focus of this chapter is on investigating the descriptive power of various models trained in an unsupervised way, segmentation quality measures are not really applicable. What is needed is an indication of the quality of each single model. In general, a good model should induce a small intra-class distance, compared to the inter-class distances to other models. That is, samples described by a certain model should have small distances to that model and large distances to all other models. In the case of a set of models M_i trained on sets of samples $\mathcal{L}_i$ taken from individual textures, this means that the distances of samples in $\mathcal{L}_i$ to model M_i should be small, whereas distances of samples in $\mathcal{L}_j$ to model M_i should be large, for $j \neq i$. This idea fits quite naturally with the models introduced in this chapter, as for each model type a distance of a sample $\mathbf{z}$ to a model M_i was defined, most often as the negative log-likelihood of $\mathbf{z}$ belonging to M_i.

Let F_i be the set of distances of samples in $\mathcal{L}_i$ to M_i and F_j the set of distances of samples in $\mathcal{L}_j$ to M_i. If these distances are viewed as features, an often-used feature evaluation measure expressing the requirements above can be applied, the *Fisher distance* between

two feature sets F_1 and F_2:

$$\mathcal{F}(F_1, F_2) = \frac{(\mu_2 - \mu_1)^2}{\sigma_1^2 + \sigma_2^2} \tag{7.35}$$

where μ_i is the average value of F_i and σ_i^2 its variance. Given a number of feature sets F_j (distances of other sets of samples to M_i), a measure for that model M_i could therefore be:

$$\mathcal{F}'(F_i) = \overline{\mathcal{F}}(F_i, F_j) = \frac{1}{|M| - 1} \sum_{j \neq i} \mathcal{F}(F_i, F_j) \tag{7.36}$$

i.e. the average Fisher distance to other models. Note that this measure might give a wrong impression, as distances are not features in the sense that when $\mu_j < \mu_i$, classification will become worse even though the Fisher distance increases. Therefore, $\mathcal{F}(F_i, F_j)$ is set to zero when evaluating model M_i and $\mu_j < \mu_i$.

A problem with this measure is that it is unbounded, i.e. if $\sigma_i^2 + \sigma_j^2 \to 0$ for any M_j, $\mathcal{F}'(M_i) \to \infty$. The Fisher measure can be bounded using a transformation

$$\varepsilon(F_i, F_j) = \frac{1}{2} - \frac{1}{2}\mathrm{erf}\left(\frac{1}{2}\sqrt{\mathcal{F}(F_i, F_j)}\right), \tag{7.37}$$

where $\varepsilon \in [0, 0.5]$ is the *predicted classification error* (or Bayes error) between classes i and j described by feature sets F_i and F_j, assuming normally distributed features with equal variance and equal prior probabilities. This assumption is valid, as the distribution of a likelihood calculated in d dimensions can be approximated by a χ_d^2 distribution, which itself for large d can be approximated by a normal distribution.

For one model M_i, a measure based on this error probability could be the mean predicted error between that model and all other models, i.e. $\varepsilon'(F_i) = \frac{1}{|M|-1} \sum_{j \neq i} \varepsilon(F_i, F_j)$. However, preliminary experiments showed that this gave a poor impression of performance due to outliers. Some textures were very hard to describe well using some models, whereas others could be described perfectly. Therefore, the median value was used:

$$\varepsilon'(F_i) = \operatorname*{med}_{j \neq i} \varepsilon(F_i, F_j). \tag{7.38}$$

This measure is too pessimistic as an approximation of the error a classifier based on distances to a set of models might make, for a number of reasons:

- each model is judged by a single feature, the distance of a set of samples to that model. In a mixture model used for classification, distances to multiple models would be taken into account, i.e. multiple features. If these distances would be uncorrelated, the error estimate would be exact. However, the very nature of

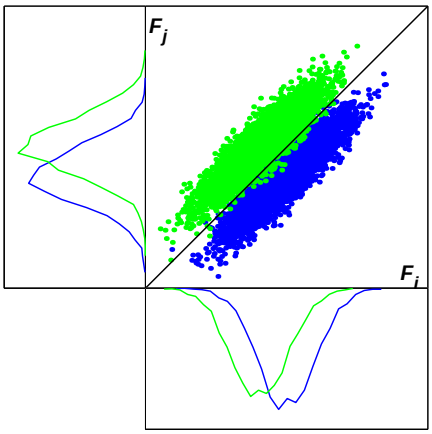


Figure 7.10: The problem of over-estimating the error by evaluating single features: F_i and F_j overlap severely when judged separately, yet are very well separable when used together.

distances makes them highly correlated. Therefore, where distance distributions might overlap in 1D, they might be completely separable in a higher dimensional space. Figure 7.10 illustrates this.

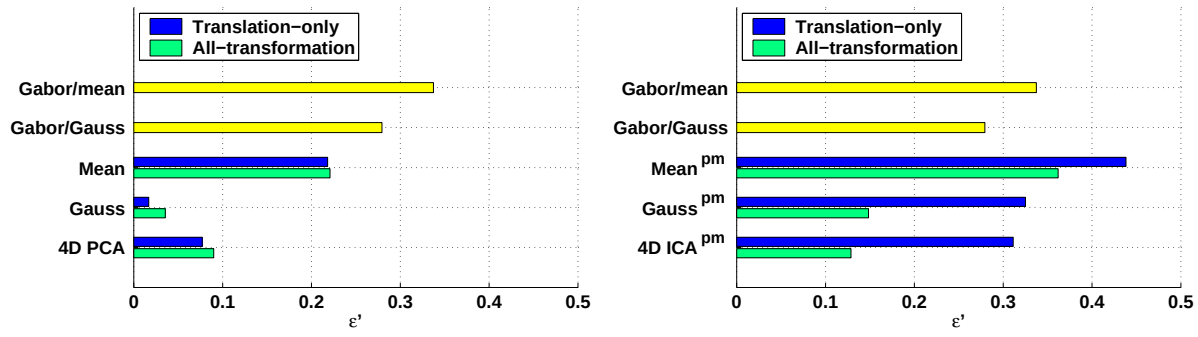
- for the models requiring PCA pre-mapping (see section 7.4.1), the pre-mapping matrix was calculated for each model individually, whereas in a mixture model this would be done on the entire training set before finding any models. This may discard directions which are useful for discriminating between the model under consideration and other models. In fact, it may map two distinct data sets on top of each other.

However, all model types will suffer from the first problem in the same way. Therefore, although the measures cannot be used as indications of absolute performance, they can be used for comparing different model types, provided they use the same pre-mapping.

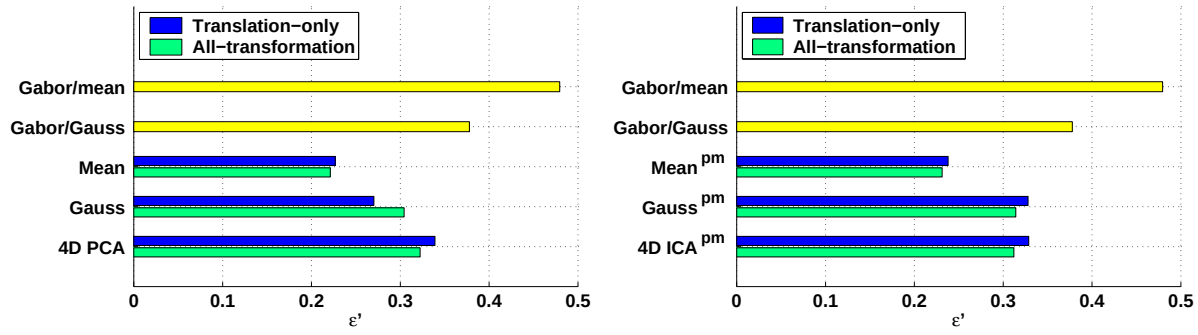
7.5.2 Initial experiments

In the first set of experiments, all models were trained on non-normalised (original) data. For each of the images in the sets of structured textures and natural textures (see section 7.3), one model was trained on 1,500 samples extracted using either the translation-only episode method (300 episodes) or the all-transformation method (100 episodes), as discussed in section 7.3.1. The window size was set to 16×16 pixels, and the window shape used was rectangular. For the subspace models, 4 dimensions were used.

The Gaussian model with full covariance matrix could not be applied in a straight-



(a) Structured textures



(b) Natural textures

Figure 7.11: First experiments: predicted error ε' of various models trained on structured and natural texture sets, using the translation-only and the all-transformation episode construction methods.

forward way to the data sets thus extracted, as the term $\ln|\det(\mathbf{C})|$ in eqn. 7.10 was $-\infty$ for several textures. Therefore, the standard Mahalanobis distance, eqn. 7.12, was used.

As the ICA model needs PCA pre-mapping to reduce the influence of noise (see section 21), and pre-mapping may introduce overlap between classes, the Gaussian models were also trained on pre-mapped data, to allow for comparison. Models trained on pre-mapped data are denoted by "pm" in figures. Unfortunately, this shortcoming of the evaluation measure ε' (eqn. 7.38) means that pre-mapped models (Gaussian, ICA) and non pre-mapped models (Gaussian, PCA) cannot be compared directly. However, they can be compared to what extent they differ from the Gaussian models trained under the same circumstances, i.e. an indirect comparison.

The Gaussian models trained on the Gabor features used the original normalised Mahalanobis distance, as in the 12D space there were no problems in finding a well-conditioned covariance matrix.

After all models were found, the median predicted errors ε' were calculated. These are shown in figure 7.11. For the structured textures, the Gabor filter methods seem to

perform worst, at least according to the measure used. This is not too surprising, as the Gabor filter output space is only 12D, whereas the other models are trained in a 256D space; it is likely there is less room for discrimination. Also, the Gabor filter bank specification chosen may lead to too coarse a discretisation of the frequency domain. Still, as texture descriptions, models in Gabor feature space seem to be less tight than those in the original space. Note that as the Gabor method is not trained on data sets but on entire images, only one result is shown in the figures; no episode construction is necessary.

The mean-only Gaussian model (henceforth "Mean") is only slightly better than the Gabor-based models, which is not surprising given its limited power. The full-covariance Gaussian method and PCA perform well, with the Gaussian clearly the best method. However, the loss in performance is not large considering PCA uses only 4 dimensions, whereas the Gaussian uses all 256.

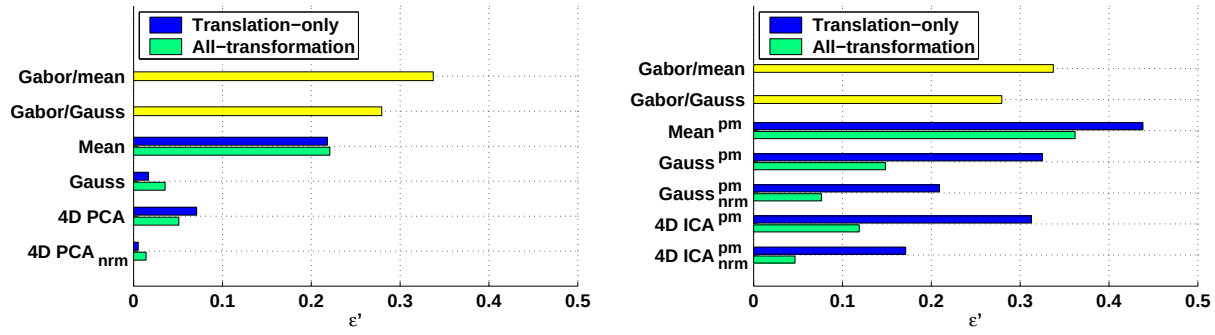
Comparing the left and right graph in figure 7.11 (a), the effect of pre-mapping is obvious. The predicted error of the Gaussian model trained on pre-mapped data is much higher than that of the same model trained on the original data, especially for structured textures. Structured texture data will lose many more dimensions by pre-mapping than natural texture data, as the covariance structure will be limited to a smaller number of dimensions (see section 7.3.2). Therefore, there might be more dimensions lost which might have been discriminating. This also explains why all-transformation sampling is of more use here, as it will (artificially) increase the covariance structure in the data. Still, the figure shows that there is hardly any difference between the Gauss^{pm} and ICA^{pm} models. An experiment to demonstrate why this is the case is discussed below, in section 7.6.

On the structured textures, overall absolute performance is quite good. As the predicted error is a pessimistic estimate of the minimum classification error possible in a mixture model setting, this indicates that these models are applicable to texture description for segmentation purposes. In contrast, on the natural textures, almost all methods perform equally poor (figure 7.11 (b)). The Mean method, the simplest in the experiments, performs best here. Modelling regularity in image patches, the basis for all other methods, is not a good method of describing these textures.

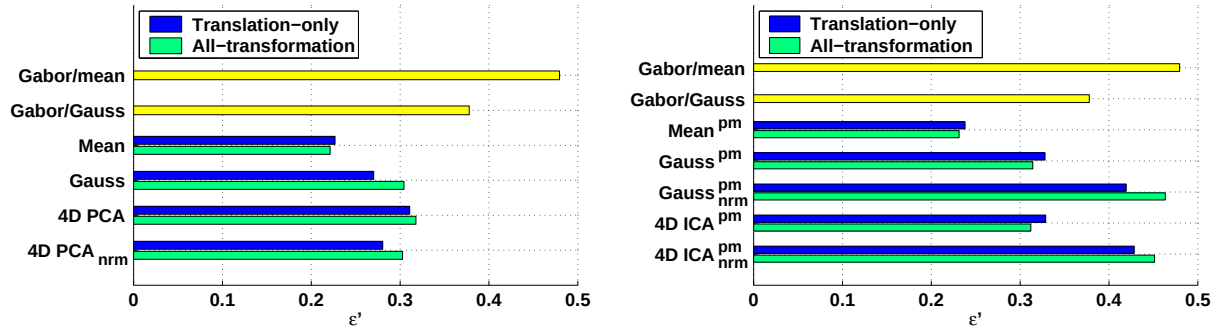
In conclusion, for both types of textures, there is not a very large difference in performance between the Gaussian, PCA and ICA models when trained under identical circumstances. However, several choices made in training the PCA models have been rather arbitrary. The next few sections will discuss the influence of these choices.

7.5.3 Normalisation

Next, the effect of normalisation was investigated. The experiments in the previous section were repeated on normalised data; all other settings were kept identical. For



(a) Structured textures

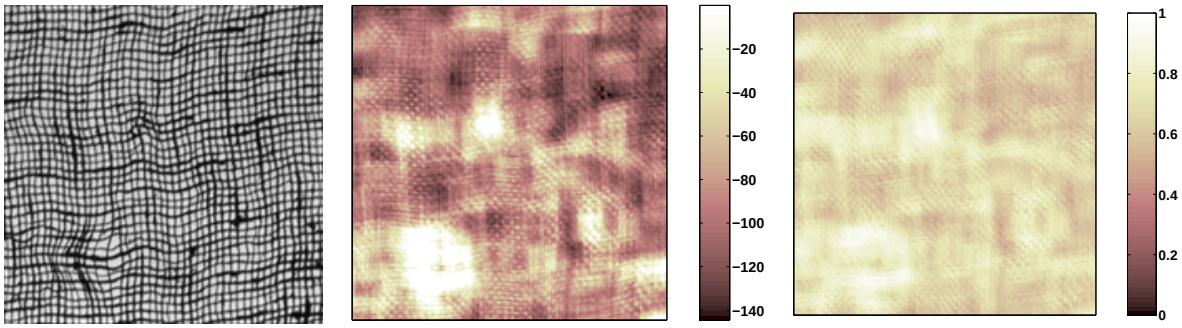


(b) Natural textures

Figure 7.12: The effect of normalisation: predicted error ε' for various models trained on original and normalised samples taken from structured and natural texture sets, using the translation-only and the all-transformation episode construction methods.

the Gabor approaches, normalisation is not applicable, so the results reported here are identical to those presented earlier. The Mean model cannot be used on normalised data, as for all data sets the mean will be $\mathbf{0}$. Finally, the Gaussian model could not be used on normalised data without pre-mapping, as even inversion of $\mathbf{C}$ (used in the Mahalanobis distance) became impossible. Results for all other model types are shown in figure 7.12; from here on, "nrm" indicates models trained on normalised data.

Clearly, results for models trained on normalised structured texture data are better than before; in most cases, the predicted error is nearly halved. For PCA_{nrm}, the drop in predicted error is even larger. This can be explained mostly by the removal of shading, which in some textures is a problem. As an example, figure 7.13 shows the distance of texture S4 to 8D PCA and PCA_{nrm} models. The spots of small and large distance occur where the original texture has a higher-than-average mean grey value and a smaller-than-average mean grey value, respectively. Although both models show the same effect, the absolute value of the coefficient-of-variation $CV = |\sigma/\mu|$ indicates that for the PCA_{nrm} model it has less influence.



(a) Texture S4 (b) 8D PCA: $CV = 0.41$ (c) 8D PCA_{nrm}: $CV = 0.13$

Figure 7.13: Difference in illumination sensitivity between PCA trained on the original data and PCA trained on normalised data (PCA_{nrm}): (a) original texture, (b) and (c) distances to the different models.

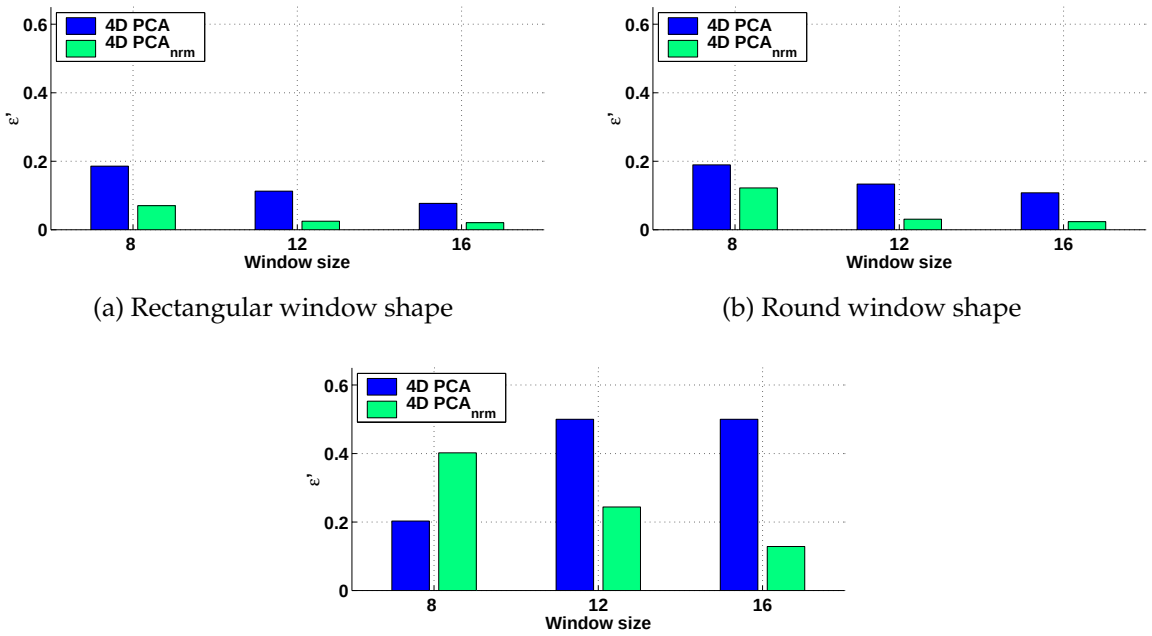
For the natural textures, normalisation worsens performance for some models. This is likely due to the amplification of unimportant details (or noise) which is present in these textures and the removal of the mean grey-value, which is important for describing these textures. However, it may also indicate that the condition of the covariance matrix deteriorates.

Again, the performance of $\text{ICA}^{\text{pm}}_{\text{nrm}}$ is much like that of the $\text{Gauss}^{\text{pm}}_{\text{nrm}}$ model, although the difference here is slightly larger.

7.5.4 Implementation choices

As discussed in section 7.3.1, there are various choices for the window size and shape to be used in sampling the images to create a data set. Until now, the window size used has been 16×16 pixels and the shape was simply rectangular. To verify whether these choices make sense, for PCA and PCA_{nrm} only the experiments were repeated with window sizes of 8×8 and 12×12 pixels, and with round and Gaussian window shapes.

Results are shown in figure 7.14. The rectangular window shape perform best, although there is not a large difference between the rectangular and round shapes. The Gaussian window shapes give much worse results. This is to be expected, as the use of a Gaussian window shape effectively halves the amount of information used, by weighing the pixels near the border of the window with very small values. For the PCA_{nrm} models, the performance using 16×16 pixel Gaussian window shapes is nearly the same as that using 8×8 pixel round window shapes. This problem could be solved by using larger windows (e.g. 24×24 or even 32×32 pixels), but this is computationally infeasible. Remarkable is the fact that for PCA the predicted error increases quickly to 0.5. This is due to the fact that there is too little information outside the subspace, which will cause



(a) Rectangular window shape

(b) Round window shape

(c) Gaussian window shape

Figure 7.14: Predicted error for 4D PCA and PCA_{nrm} models, as a function of window size and shape. Only results for structured textures are shown.

the distance measure used (eqn. 7.26) to blow up. The PCA_{nrm} models do not suffer from this problem, as they do not use this information.

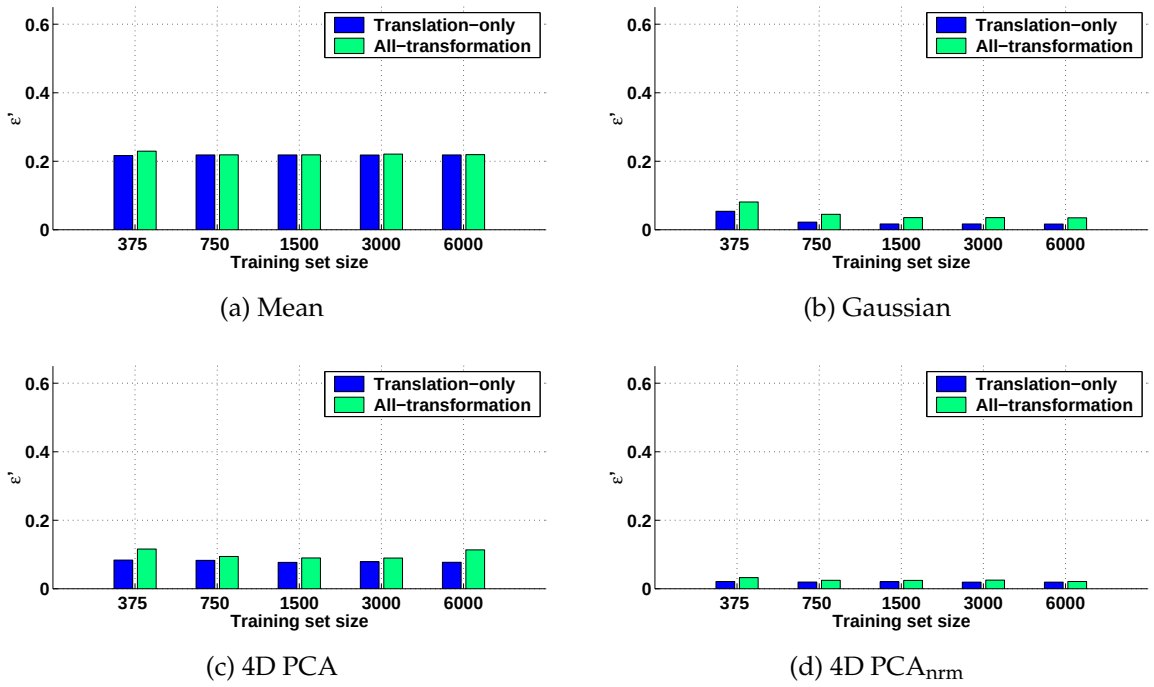
For the natural textures the same conclusions can be drawn, although the differences in performance are much smaller, since they were poor to begin with. These results are not shown here.

Based on these results, for the remaining experiments in this chapter and the experiments on texture segmentation in the next chapter, rectangular 16×16 pixel windows were used in sampling the images.

7.5.5 Sample size

The number of samples used for calculating the models thus far was 1,500, divided over either 300 translation-only episodes or 100 all-transformation episodes. However, a larger sample size might yield better results. A number of experiments were performed for a range of sample sizes. The results are shown in figure 7.15. Results for ICA are not shown, as the results again are very similar to those for the Gaussian model.

The results on the structured textures show that for most models, a sample size of 1,500 is quite sufficient. Only the Gaussian model gives slightly lower errors for larger sample



(a) Mean (b) Gaussian

(c) 4D PCA (d) 4D PCA_{nrm}

Figure 7.15: The effect of training sample size on the results for the various models. Only the results for structured textures are shown, trained on data sets constructed using the translation-only and all-transformation episode construction methods.

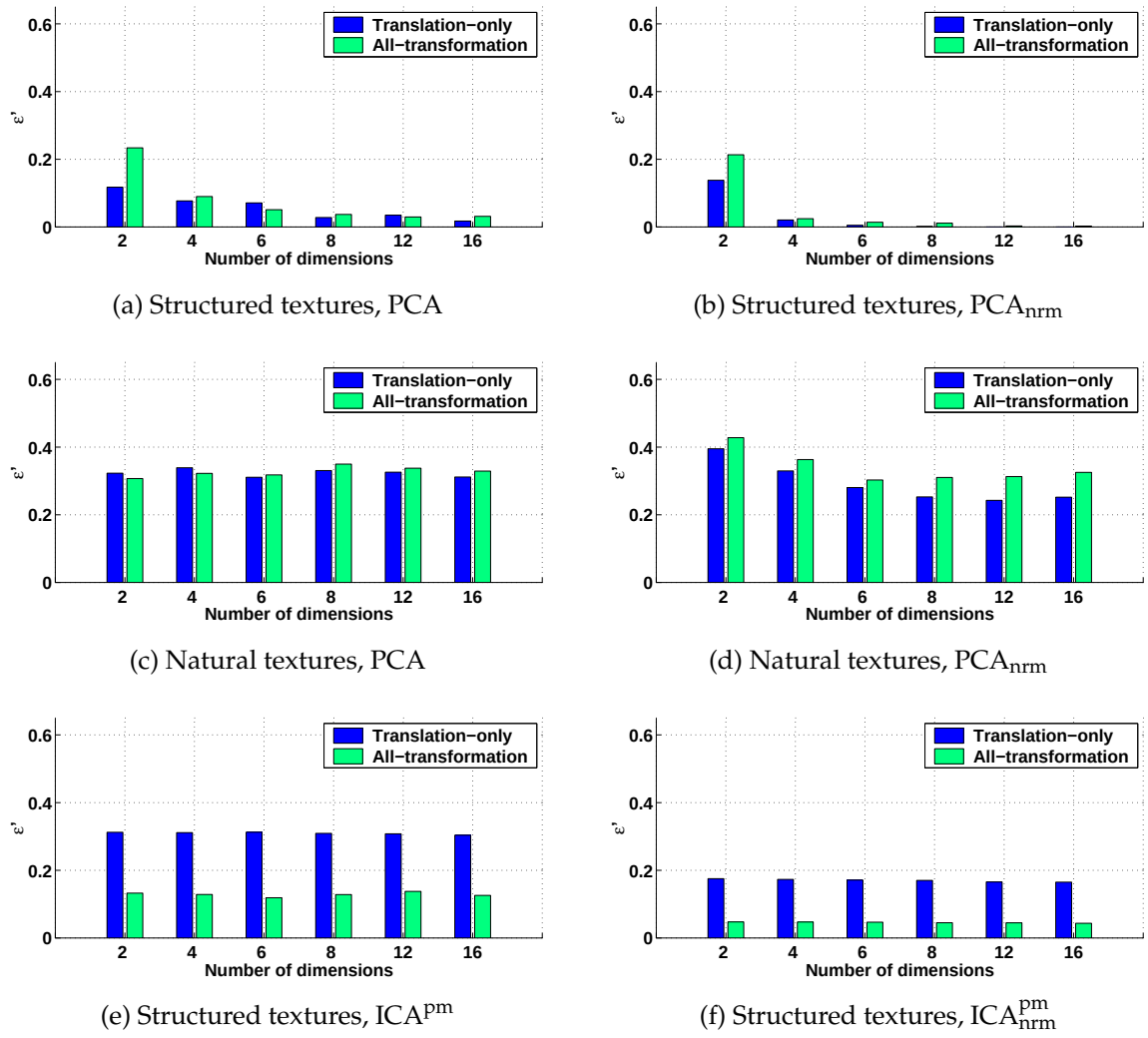
sizes, although the differences are not significant. This is to be expected, as for the Gaussian model a large number of parameters has to be estimated; for pre-mapped data of, say, 60 dimensions, the covariance matrix contains 3,600 parameters.

On the natural texture set, the results (not shown here) are more or less the same, although the drop in error is less pronounced as the error remains quite high for all models. For the remaining texture experiments in this chapter and those using mixture models in the next chapter, a sample size of 1,500 was used. However, note that in principle for PCA and PCA_{nrm} a much lower sample size should be enough; even for only 375 samples, the predicted error is remarkably low.

7.5.6 Subspace dimensionality

Until now, the number of dimensions used in the PCA and PCA_{nrm} experiments was fixed at 4. This was based on the rough approximation that as there are 4 degrees of freedom the subspace needs to cope with, 4 dimensions should suffice (see section 7.4.2). However, for reasons discussed earlier, this need not be the case.

For the PCA, PCA_{nrm}, ICA^{pm} and ICA^{pm}_{nrm} models, trained on both the original and



(a) Structured textures, PCA

(b) Structured textures, PCA_{nrm}

(c) Natural textures, PCA

(d) Natural textures, PCA_{nrm}

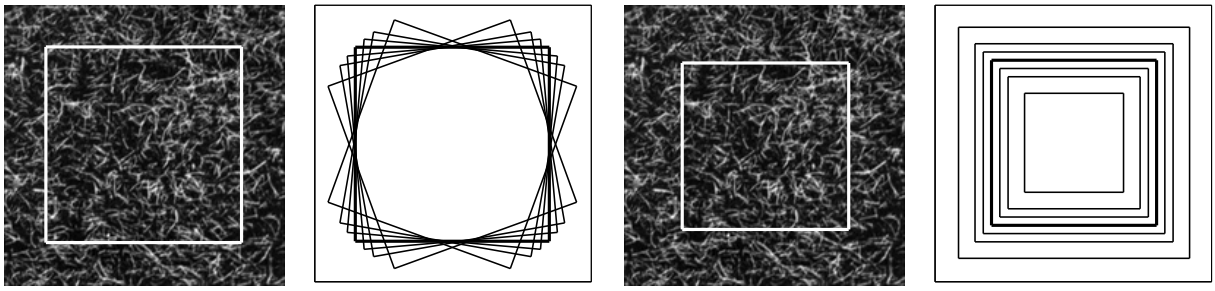
(e) Structured textures, ICA^{pm}

(f) Structured textures, $\text{ICA}^{\text{pm}}_{\text{nrm}}$

Figure 7.16: The effect of varying subspace dimensionality for the PCA and ICA models, trained on both original and normalised data. For ICA, only the results for structured textures are shown.

normalised data, the experiments were repeated for mD subspaces, where $m \in \{2, 4, 6, 8, 12, 16\}$. The resulting predicted error values are shown in figure 7.16. Interestingly, for PCA_{nrm} models, the predicted error drops much faster with increasing m than it does for PCA models. This is caused by the fact that normalisation reduces the number of degrees of freedom present in the data, so that a lower number of dimensions should suffice to describe it. For PCA models, the predicted error levels out at approximately 8 dimensions; for PCA_{nrm}, it stops decreasing significantly at 4 dimensions.

For PCA on natural textures, the effect of increasing dimensionality is not present; for PCA_{nrm}, there is a small effect. However, the predicted error ε' stabilises at a high value. Finally, for ICA^{pm} and $\text{ICA}^{\text{pm}}_{\text{nrm}}$ the dimensionality does not have any influence at all.



(a) Rotation (b) Scaling

Figure 7.17: Sub-images used for measurement of invariance to rotation and scaling. The thick boxes indicate the original sub-image. Note that for scaling the relative is indicated; in all cases, the content was the same.

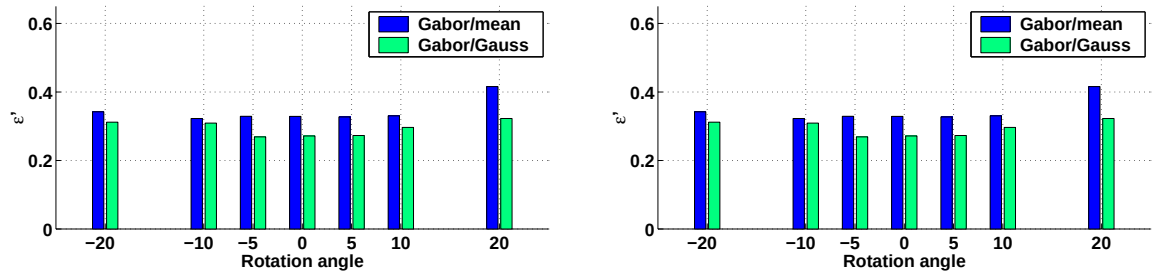
In conclusion, PCA with as little as 8 dimensions or PCA_{nrm} with 4 dimensions can reach performances as good as that of a full-covariance matrix Gaussian model (compare figures 7.16 (a) and (b) to figure 7.11 (a)).

7.5.7 Invariance

A final remaining question is to what extent these methods are truly invariant, and what role the episode construction methods and normalisation (section 7.3.1) and the number of dimensions used in PCA and PCA_{nrm} play. The expectation is that translation-only sampling should have little effect, since in these texture images the content is quite homogeneous; normal, non-episode sampling should already yield translated versions of the same image patches. All-transformation sampling, however, is expected to be useful. With respect to the dimensionality of subspaces, the expectation is that a larger number of dimensions will yield a smaller sensitivity to transformations. To find answers to these questions, a slightly different set of experiments was performed, in which images were rotated or scaled before the predicted error was calculated. Figure 7.17 illustrates the method used for both invariances.

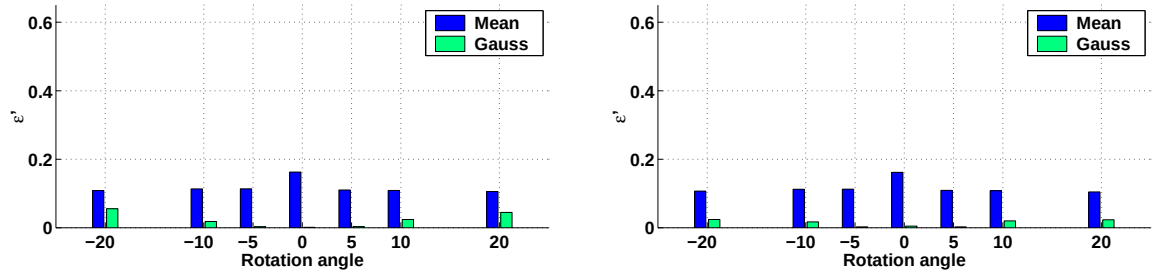
To test invariance to rotation, the measure ε' was calculated on rotated textures. Each texture i described by model M_i was rotated over ϕ degrees, where $\phi \in \{-20°, -10°, -5°, 0°, 5°, 10°, 20°\}$. Rotation was performed by re-sampling using bicubic interpolation. The other textures used in the calculation of ε' were not rotated. After rotation, a sub-image of relative size $\frac{1}{2}\sqrt{2} = 135 \times 135$ pixels around the center of each 192×192 pixel texture image was used. The measures ε' were then calculated based on the distances of the pixels in this sub-image. Note that as only a part of the image is taken into consideration, this will give different results than those found before.

For testing invariance to scaling a similar procedure was followed. Textures were scaled



(a) Translation-only episodes

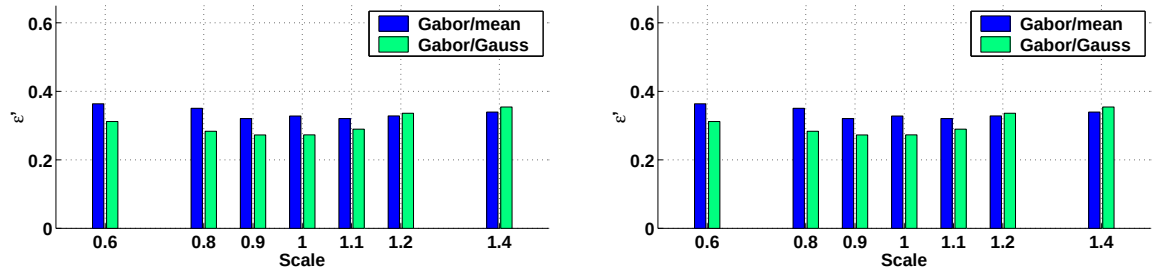
(b) All-transformation episodes



(c) Translation-only episodes

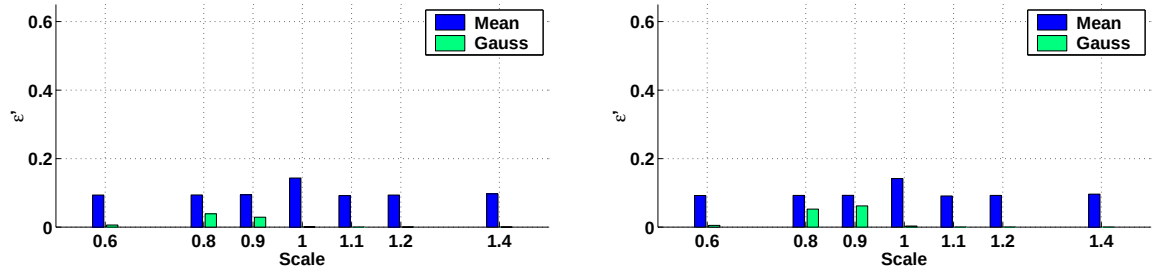
(d) All-transformation episodes

Figure 7.18: Invariance to rotation for the Gabor and Gaussian models trained on structured textures, using different episode-construction methods.



(a) Translation-only episodes

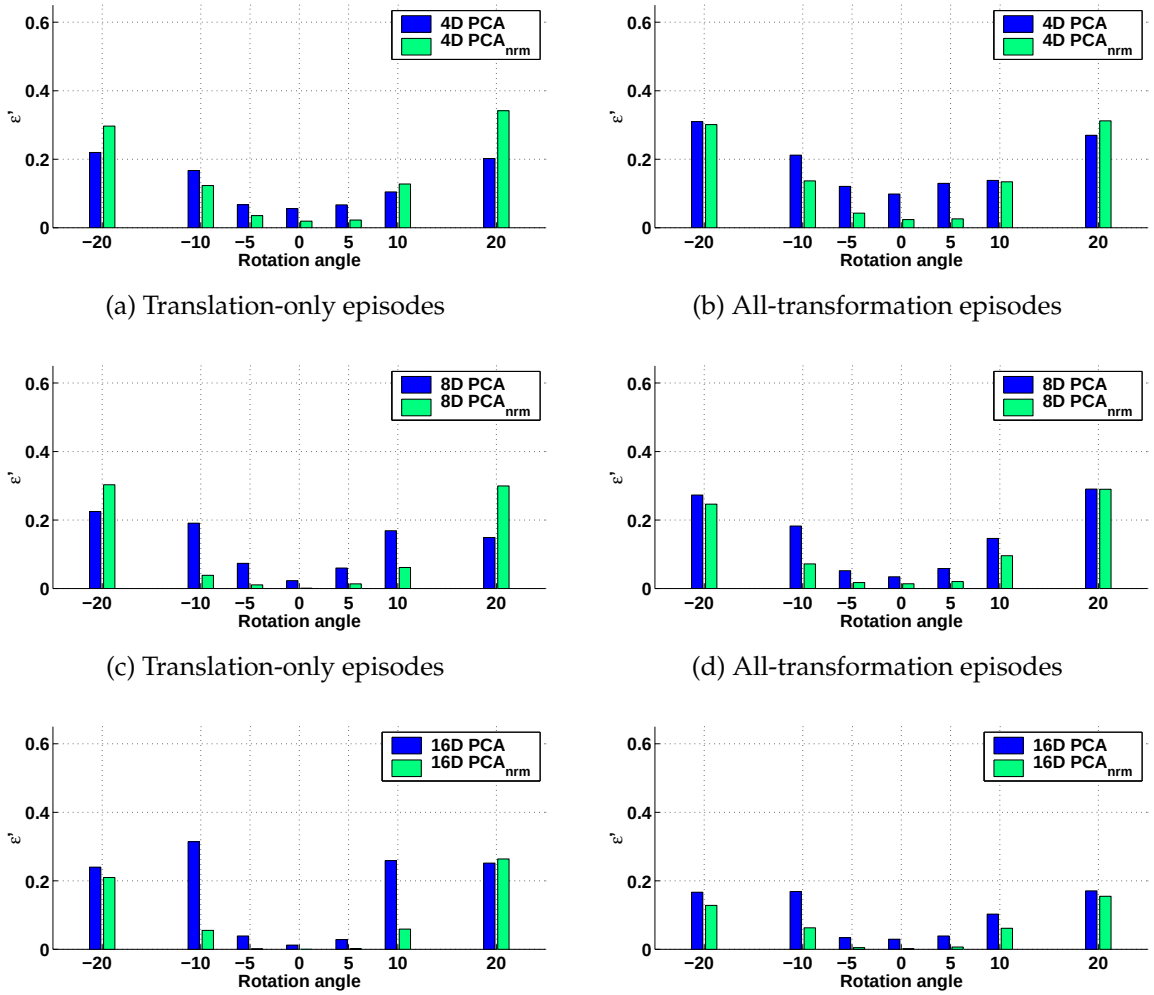
(b) All-transformation episodes



(c) Translation-only episodes

(d) All-transformation episodes

Figure 7.19: Invariance to scale for the Gabor and Gaussian models trained on structured textures, using different episode-construction methods.



(a) Translation-only episodes

(b) All-transformation episodes

(c) Translation-only episodes

(d) All-transformation episodes

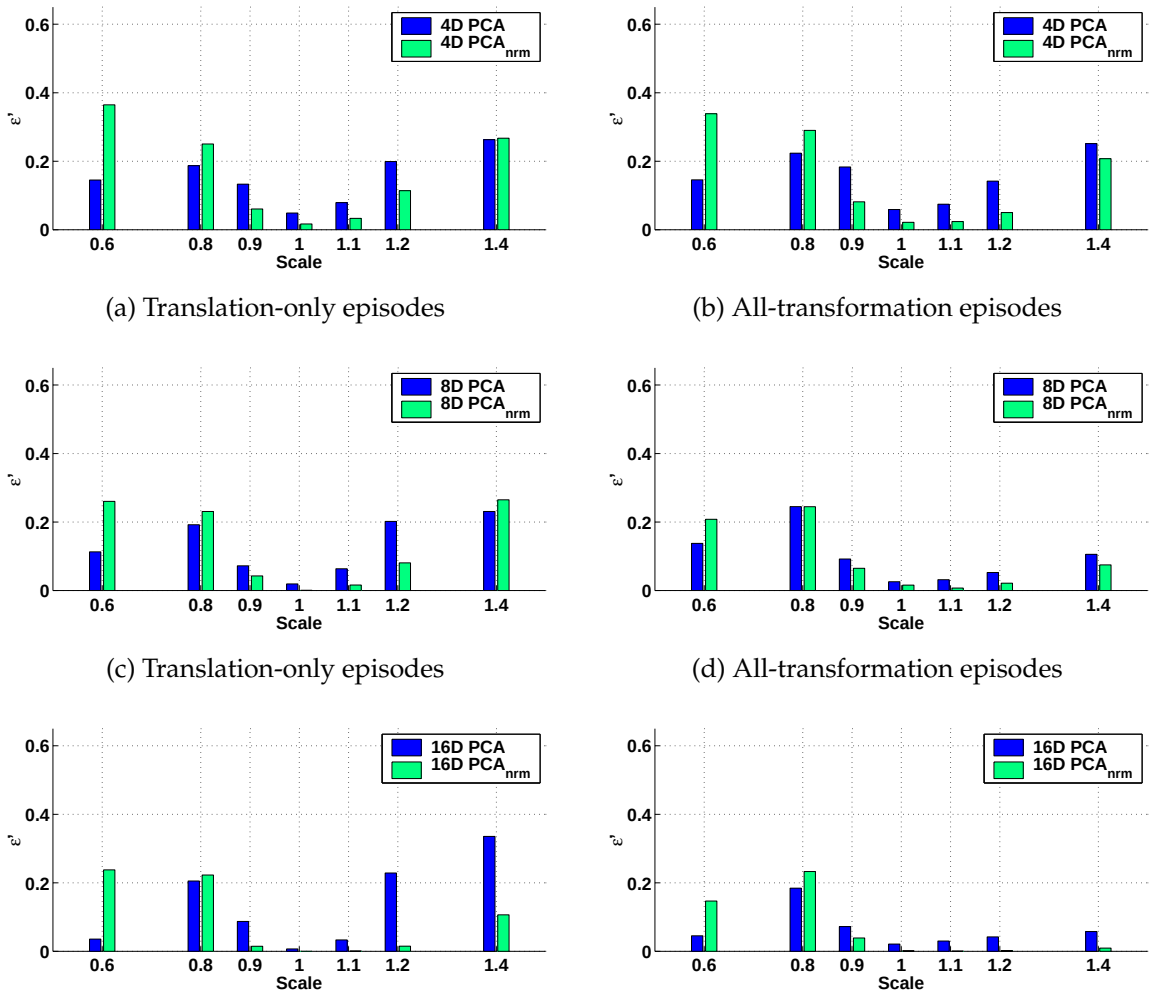
(e) Translation-only episodes

(f) All-transformation episodes

Figure 7.20: Invariance to rotation for PCA and PCA_{nrm} models of various dimensions, trained on structured textures, using different episode-construction methods.

over a range of $s \in \{0.6, 0.8, 0.9, 1.0, 1.1, 1.2, 1.4\}$ times the original scale. Scaling was also performed by re-sampling using bicubic interpolation. For each scale, a sub-image of relative size $0.6 = 115 \times 115$ pixels was extracted around the center of the image. Again, the other texture images were kept the same, and the predicted error ε' was calculated for each scale. The range of scales used will test invariance in a region which was not explicitly used in constructing the episodes for training; recall that only scales in the range $[1.0, 1.5]$ times the original were used (section 7.3.1).

These procedures were applied to the Gabor, Gaussian, PCA and PCA_{nrm} models. Figures 7.18 and 7.20 show the results for rotation; figures 7.19 and 7.21 those for scale. Only results for structured textures are reported; for natural textures, the effects were



(a) Translation-only episodes

(b) All-transformation episodes

(c) Translation-only episodes

(d) All-transformation episodes

(e) Translation-only episodes

(f) All-transformation episodes

Figure 7.21: Invariance to scale for PCA and PCA$_{\text{nrm}}$ models of various dimensions, trained on structured textures, using different episode-construction methods.

the same. Furthermore, results for the ICA models are not shown, as they are similar to those for the Gaussian model.

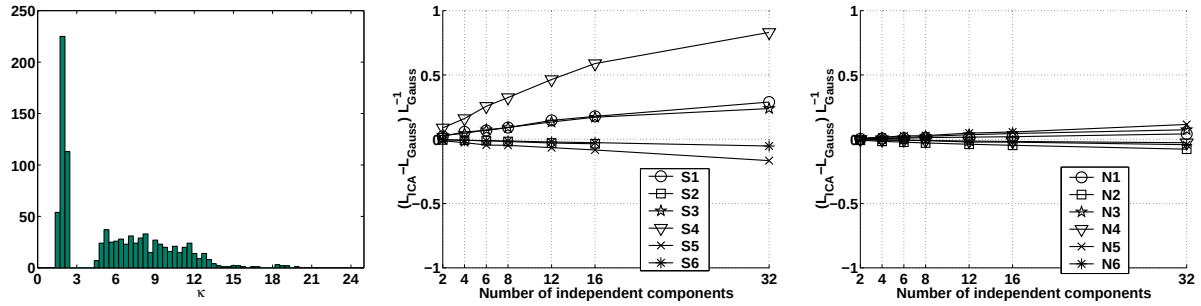
The figures show the following:

- as was to be expected, the Gabor models and the Mean model are invariant over a large range of rotation angles and scales (figures 7.18). However, as performance was poor to start with, these numbers should be interpreted with caution. For these models, there is no significant difference between using translation-only episodes and all-transformation episodes.
- for the Mean model, the predicted error *decreases* slightly when the images are

rotated or scaled. This is due to the smoothing involved in the re-sampling step necessary for rotating or scaling the images. This will decrease variation in the mean over the image, which in turn decreases the variance over the distances of each pixel to that mean and therefore increases the Fisher distance used in calculating the predicted error (eqn. 7.35).

- comparing the remaining methods, the full-covariance Gaussian model gives the best results over a wide range of transformations. This is to be expected, since it models all possible variation in the data.
- in the rotation results, most graphs are symmetric around 0°, which is to be expected as there was no bias towards clockwise or counter-clockwise rotation in the episode construction. In the PCA and PCA_{nrm} graphs, there is slight asymmetry. This is probably due to the fact that some structured textures have a strong dominant orientation, so that rotating a texture might cause it to be described better by another model. As PCA and PCA_{nrm} model the data more tightly than the other models, they may be more sensitive to this effect.
- the scale experiments show large asymmetry around scale 1.0, especially for those models trained on all-transformation episodes. This was already expected, as the range of scales tested here is not the same as that used in episode construction.
- high-dimensional PCA models are more invariant than those with a low number of dimensions, but over a smaller range, especially those trained on translation-only episodes (figures 7.20 (a)-(c)-(e)). This can be explained by looking at the distance measure used (eqn. 7.26). As more dimensions are used in the subspace, there will be less variance outside the subspace and β (eqn. 7.25) will rapidly increase. The model becomes more tight, making it very unlikely for data to fall outside. This means the subspace will describe the data it was trained on better when more dimensions are used, but at the same time accommodates previously unseen samples worse.
- in contrast, PCA_{nrm} models give smaller predicted errors with increasing dimensionality over the entire range of transformations. This effect is most noticeable in the scaling experiments (figures 7.21 (a)-(c)-(e) and (b)-(d)-(f)). The error drops more quickly than for PCA models, as was already shown in section 7.5.6. Also, for PCA_{nrm} the range over which performance is good expands with increasing dimensionality, see e.g. figures 7.20 (a)-(c)-(e). The lack of a model outside the subspace makes these models more invariant.
- the use of all-transformation episodes clearly helps models to become more invariant. This effect is most pronounced in results for the high-dimensional PCA and PCA_{nrm} models.

In conclusion, the Gaussian models give the best overall results. Still, PCA and PCA_{nrm} models yield nearly similar results using only a fraction of the number of parameters



(a) (b) (c)

Figure 7.22: (a) Histogram of kurtoses of projected distributions, over all textures and all ICA models. (b) Normalised difference in likelihood between the ICA and Gaussian models trained on structured textures S1-S6. (c) Same, for natural textures N1-N6.

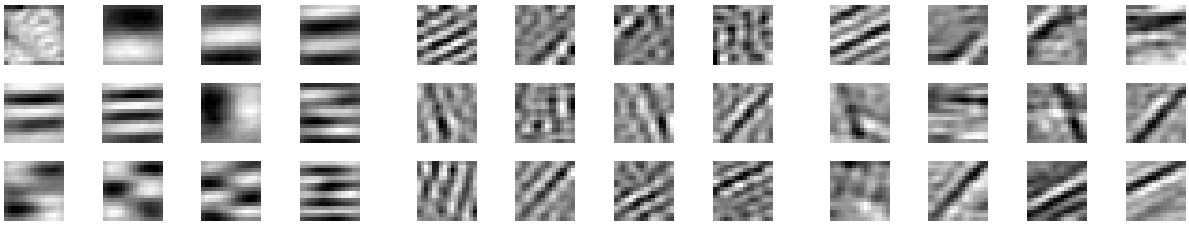
needed for the Gaussian model, although their ranges of invariance seem to be somewhat smaller. Of the subspace models, those trained on normalised data (PCA_{nrm}) perform best. Finally, there is evidence that the all-transformation episode extraction method helps in achieving invariance to rotation and scale.

7.6 Applicability of independent component analysis

The experiments discussed in sections 7.5.2 and 7.5.3 showed that the performance measures found for ICA models were very close to those found for Gaussian models. Furthermore, for an increase in the number of independent components used (see section 7.5.6), performance did not increase at all. This leads to the question what directions the ICA model actually finds.

Investigation of the kurtoses of the distributions of data projected onto the extracted independent components shows that non-Gaussian directions have indeed been found. Figure 7.22 (a) shows a histogram of the kurtoses κ found using the independent components of all ICA models of all dimensions, trained on all textures. Clearly, sub-Gaussian ($\kappa < 3$) and super-Gaussian ($\kappa > 3$) distributions have been found, whereas none of the projected distributions resemble a Gaussian (for which $\kappa = 3$). This indicates that there are independent components present in the data, most of which correspond to super-Gaussian distributions.

To investigate the difference between the Gaussian and ICA models, inspection of the likelihood of the data set after training is instructive. If the ICA model really fits much better than the Gaussian, its likelihood $\mathfrak{L}_{\text{ICA}}$ (see appendix C) should be significantly larger than that of the Gaussian, $\mathfrak{L}_{\text{Gauss}}$. However, for nearly all textures, the increase in likelihood was negligible. Figures 7.22 (b) and (c) show the relative change in likelihood,



(a) PCA (b) ICA, **W** (c) ICA, **A**

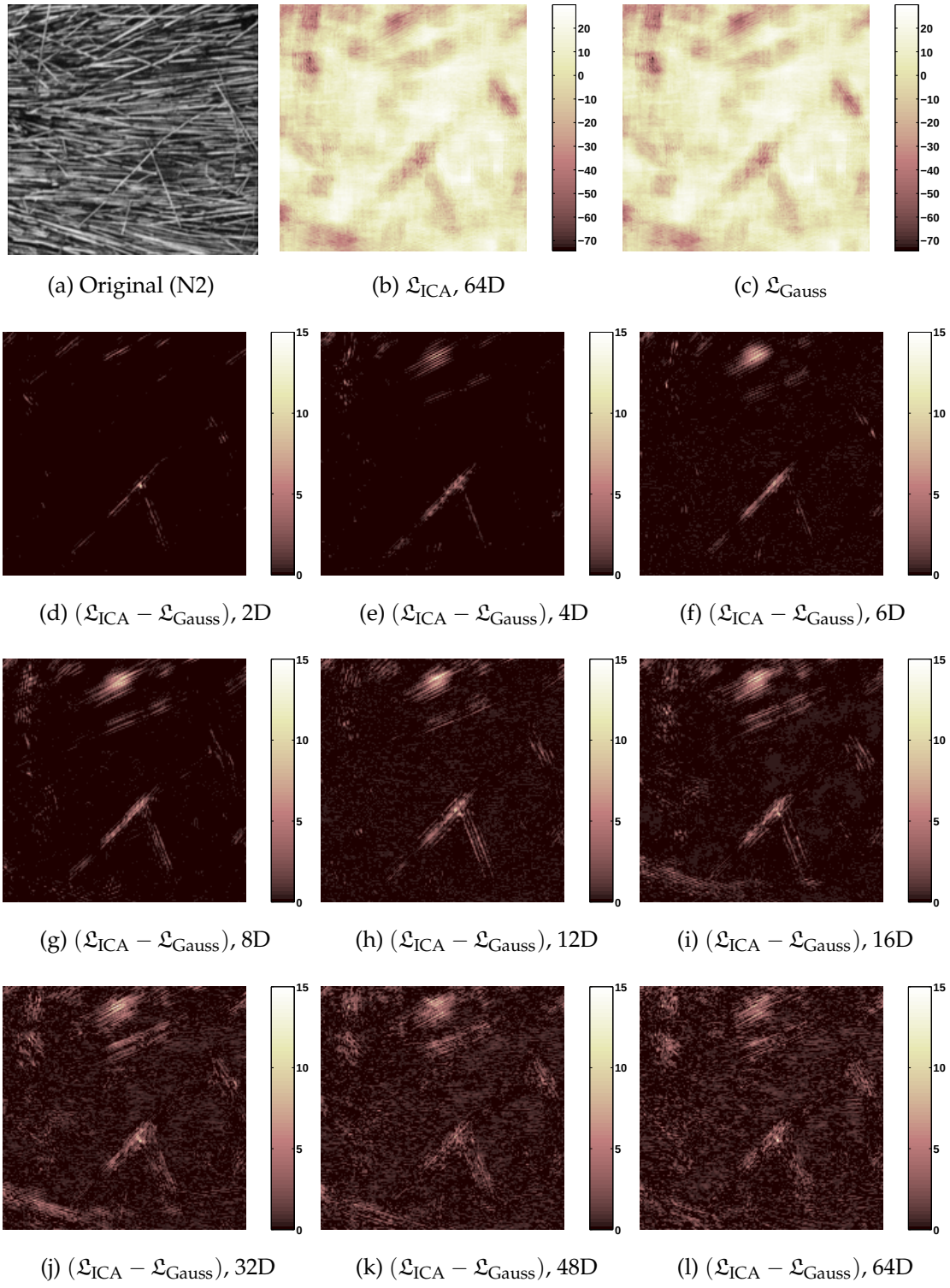
Figure 7.23: (a) The first 12 basis vectors of PCA. (b),(c) Filters and basis vectors of 12D ICA on texture N2.

$\frac{\mathfrak{L}_{\text{ICA}}-\mathfrak{L}_{\text{Gauss}}}{\mathfrak{L}_{\text{Gauss}}}$, for ICA models of different dimensionality, trained on different textures. Clearly, for most models the likelihood increases only slightly or not at all. For some textures the ICA model even seems to be less likely than the Gaussian model.

To get a better idea of why these increases are so low, the "Straw" natural texture N2 (see figure 7.24 (a)) is considered. Figure 7.23 shows the first 12 PCA basis vectors and the ICA filters and basis vectors found after training a 12D ICA model. Clearly, the ICA filters are directed edge detectors, whereas the PCA basis vectors correspond to global content. One would expect these filters to be of use in segmenting the straw image. To see their effect, for each pixel in the original image the likelihood of the window of which it is the center can be plotted, again as an image. Figures 7.24 (b) and (c) show these likelihood images for both the 64D ICA model and the Gaussian model. At first glance, there is no difference between the two. However, there is a difference, albeit small; figure 7.24 (d)-(l) shows this difference, for an increasing number of independent components in the ICA model. For presentation purposes, negative differences (which fell in the range $[-2, 0]$) have not been shown.

It now becomes obvious why the ICA model shows no improvement over the Gaussian model. In general, the independent components correspond to characteristic but more or less unique high-frequency events in images. For this image, these are the few straws that have a different orientation than the majority. The first few independent components (2D-4D models) correspond to the straws below the center of the image; as more dimensions are added, the straws at the top of the image get modelled better as well. Finally, as more and more independent components are found, all straws with non-standard orientations are singled out (relative to the Gaussian model), whereas the main structure of the texture becomes slightly more likely, but in a noise-like fashion.

In fact, this happens for all textures. The reason that some of the structured textures show a large increase in relative likelihood (S4, S1 and S3 in figure 7.22 (b)) is that for structured textures, these high-frequency events are plentiful in the images, so that the overall likelihood increases faster. The increase in likelihood per pixel, though, is of the same order as for N2. Moreover, as these events are not described in a translation



(a) Original (N2) (b) $\mathfrak{L}_{\text{ICA}}$, 64D (c) $\mathfrak{L}_{\text{Gauss}}$

(d) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 2D (e) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 4D (f) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 6D

(g) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 8D (h) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 12D (i) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 16D

(j) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 32D (k) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 48D (l) $(\mathfrak{L}_{\text{ICA}} - \mathfrak{L}_{\text{Gauss}})$, 64D

Figure 7.24: Difference between ICA and Gaussian models, for various dimensions of the ICA subspace. The top row shows (a) the original image, N2; (b) $\mathfrak{L}_{\text{ICA}}$, the likelihood of each pixel belonging to a 64D ICA model; (c) $\mathfrak{L}_{\text{Gauss}}$, the same for a Gaussian model and (d)-(l) the difference between $\mathfrak{L}_{\text{ICA}}$ of various dimensionalities and $\mathfrak{L}_{\text{Gauss}}$.

invariant way, very large numbers of independent components are needed to come to a significant increase in overall likelihood. Given the algorithmic and storage complexity of ICA, this is not worth the effort.

These observations lead to the following conclusions:

- the increase in likelihood is generally so low, that the full-covariance Gaussian model may be considered a good model for a data set consisting of texture image patches.
- ICA is not useful as a texture description tool. For segmentation, the goal is to find models which describe textures in a shift-invariant way. To this end, non-standard regions should be *ignored* rather than modelled. The Gaussian model does this by modelling the data by (co)variance only, ignoring outliers. Under the ICA model, as it focuses on non-Gaussianity alone, outliers are more probable (for super-Gaussian sources); in fact, the sources are indirectly optimised to make outliers more likely.
- it also means that ICA is potentially useful for *detection* and *coding*. For example, given a set images containing objects, ICA might find features corresponding to unique properties of the objects. However, there is no guarantee these will be useful properties. In coding, one of the goals is to obtain a sparse code, in which each code word specifically describes the occurrence of one event. Following the outlier argument above, ICA might be appropriate for this goal.

The main conclusion is that, where many authors have suggested ICA might be useful for image processing (see section 21), its application area is limited to problems in which the detection of unique, characteristic events is of importance. Texture description is not one of these.

7.7 Conclusions

This chapter introduced the idea of using subspace models for feature extraction, applied to texture description. In section 7.2, an overview was given of work published on subspace models. The ASSOM was discussed as an interesting model, yet cumbersome to use in practice. It was argued that the self-organising capabilities of the SOM might not be useful in many applications, and that simplifying the ASSOM by dropping these capabilities leads to a mixture-of-PCA algorithm. The basic elements of these algorithms are clustering and the calculation of subspaces. This chapter then continued to focus on the latter; possible clustering schemes will be discussed in the next chapter.

Next, two sets of Brodatz texture images were presented in section 7.3, one containing structured textures and the other containing natural textures. These sets were used as a benchmark in subsequent texture description experiments. It was shown how data sets

can be constructed by sampling from these images, how the notion of episodes used in the ASSOM can be used, and how normalisation and pre-mapping might help in achieving illumination invariance and removing noise.

Section 7.4 then introduced a number of models and discussed them theoretically: the Gaussian, principal component analysis (PCA) and independent component analysis (ICA) models. It was shown how distance measures to each of these models can be expressed, and what effect normalisation has on the applicable PCA distance measure. An algorithm for finding undercomplete ICA bases was briefly presented; a complete derivation can be found in appendix C. Based on several publications, expectations with respect to the usefulness of ICA in image description were formulated, i.e. that the method might be useful to describe textures containing characteristic high-frequency elements.

In section 7.5, these models were put to the test. A performance measure was introduced, the median predicted classification error. The models discussed in section 7.4 were then trained on the Brodatz texture sets and performances were compared. On structured textures, most models performed well, indicating that they are applicable for description of such data. Of the different models, the Gaussian performed best. However, the differences with PCA were small, whereas PCA uses only a fraction of the number of parameters the Gaussian model needs. Although not all models could be compared directly as some of them needed a PCA pre-mapping, ICA was shown to give nearly the same performance as a Gaussian model trained under the same circumstances. On natural textures, finally, none of the models seem particularly applicable; the mean-only Gaussian model performed best. The underlying idea of using subspaces, i.e. modelling certain regularity in an invariant way, is not applicable to these textures; they are probably best described statistically at the level of individual pixels.

Further experiments showed that on the texture data used, normalisation of the data improves PCA results; that the use of rectangular, 16×16 windows gave the best performance; that small sample sizes suffice to train PCA models, whereas the Gaussian model needs larger data sets; and that for PCA models, 4-8 dimensions are enough to give good performance, depending on whether the original data or normalised data is used to train them. Finally, experiments were performed on the invariance of models over a range of rotation angles and scaling factors. These showed the Gaussian model to be the most invariant. However, PCA models with 8-16 dimensions, especially those trained on normalised data, again come very close over a limited range of transformations. These experiments also demonstrated the drawback of training on non-normalised data, that is that the need for estimating a model inside the subspace introduces a noise model outside the subspace, which might not always be applicable. Although regularisation might help to overcome this, this introduces several new problems. Interestingly, this contradicts of some authors that a problem of PCA is that it does not define a probability model [355]; in some applications we find that such a model is simply not needed. The all-transformation episode construction method, in which ro-

tated and scaled versions of samples are added to the data set, was found to be useful when the training data does not yet contain these transformations.

The experimental finding that the Gaussian and ICA models gave nearly the same results was investigated in section 7.6. It was shown that, although the ICA method does find non-Gaussian directions in the data, they mainly describe unique events in the image in a shift-dependent way. Therefore, for any reasonable number of independent components, the likelihood does not increase significantly. The conclusion is that ICA, although it might be useful for detection of rare occurrences of high-frequency structure in images, is not applicable to texture segmentation. Given its high computational cost and lack of use, ICA will not be further considered in chapter 8 as a mixture model element.

This chapter has laid the basis for application of subspace models to image processing problems. PCA, especially when trained on normalised data, has been shown to be able to give well-performing, invariant descriptions of structured textures. In the next chapter, mixtures of these models will be applied to texture segmentation, object recognition and image database retrieval problems.

8

IMAGE DESCRIPTION USING MIXTURE-OF-SUBSPACE MODELS

8.1 Introduction

In the previous chapter, a number of subspace models was examined in the context of texture description. It was shown how PCA can be used to describe single textures invariant to simple transformations. In practical applications however, multiple textures will have to be modelled, e.g. for segmenting an image. Section 7.2 introduced the idea of using mixtures-of-subspaces to approach this kind of application. This chapter discusses various ways of formulating mixture-of-subspaces models (henceforth "MoS models") and apply the resulting algorithms to a number of problems.

First, section 8.2 introduces a number of clustering algorithms which can be used to train MoSs. Next, in section 8.3, the resulting models will be applied to texture segmentation. Section 8.4 shows how MoS models can be used to describe the content of single images, and how distances between these mixtures can be used as distance measures between images. A natural application for this method is image database retrieval. Next, section 8.5 introduces the idea of using MoS assignments as image class descriptions, i.e. sets of features characterising image content for a range of images. This technique will be applied to the problem of object recognition. As a last application, section 8.6 returns to the problem discussed in chapter 3, handwritten digit recognition. It is shown how the MoS approach reaches performances quite near those of the heavily optimised supervised methods used before. Furthermore, the ease of interpretation of these models will be demonstrated. The chapter ends with some conclusions in section 8.7.

The application of MoSs to various problems in this chapter is meant to be illustrative rather than proving that such models are the best methods available; they probably are not. As such, little attempts have been made to pre-process data or post-process results to increase performance. However, when possible the methods are compared to alternative approaches.

8.2 Clustering

8.2.1 The k-subspaces algorithm

The simplest way of clustering data in subspaces is to use a variation on the k-means algorithm [85, 92], one of the simplest clustering algorithms available. The main difference with the k-means algorithm is that, where k-means only recalculates the cluster mean $\boldsymbol{\mu}$ in each iteration, the k-subspaces algorithm recalculates the PCA projection matrix $\mathbf{W}$ as well. Furthermore, the distance measure $D(\mathbf{x}, P)$ used is a distance to a subspace P, either eqn. 7.19 or 7.26, rather than a distance to a cluster centre $\boldsymbol{\mu}$ only.

Say a data set $\mathcal{L} = \{\mathbf{x}^n\}, n = 1 \ldots N$ is given and a set of k subspaces $\{P_1, \ldots, P_k\}$ with m dimensions each is to be trained, where each P_i is characterised by its set of parameters $\Theta_i = \{\boldsymbol{\mu}_i, \mathbf{W}_i\}$. The basic algorithm then consists of the following iteration (starting with $t = 0$):

1. assign each sample $\mathbf{x}^n \in \mathcal{L}$ to that subspace P_i to which it has the smallest distance $D(\mathbf{x}^n, P_i)$, i.e. $\mathcal{S}_i = \left\{\mathbf{x}^n | i = \arg\min_j D(\mathbf{x}^n, P_j)\right\}$;
2. for each subspace P_i, re-calculate the parameters $\mathbf{W}_i$ and $\boldsymbol{\mu}_i$ by performing PCA on the set of samples $\mathcal{S}_i$;
3. set $t = t + 1$; while the average change in distance is larger than τ and $t < t_{max}$, go to 1.

Here τ and t_{max} are a stopping criterion and maximum number of iterations, respectively. In the experiments in this chapter, τ was set to 1.0×10^{-6}, and t_{max} to 1,000. Note that the number of subspaces k has to be specified beforehand.

An advantage of the k-subspaces algorithm is that it allows for easy incorporation of the episode idea introduced in section 7.3.1. As samples are assigned to a single subspace, groups of samples can be treated in the same way. Recall that Kohonen proposed to use episodes $\mathcal{E}$ in an adapted distance formulation (eqn 7.1). However, where he used the minimum of the distances over all samples in the episode, in this chapter the average distance is used:

$$D(\mathcal{E}, P_i) = \frac{1}{|\mathcal{E}|} \sum_{\mathbf{x}^n \in \mathcal{E}} ||\mathbf{x}^n - \hat{\mathbf{x}}_i^n||, \tag{8.1}$$

where $\hat{\mathbf{x}}_i^n$ is the projection of $\mathbf{x}^n$ onto subspace P_i. This distance measure was found to give more stable convergence. Step 1 of the algorithm above then becomes:

1. for each episode $\mathcal{E}$, assign all samples $\mathbf{x}^n \in \mathcal{E}$ to that subspace P_i to which it has the lowest distance $D(\mathcal{E}, P_i)$, i.e. $\mathcal{S}_i = \left\{\mathbf{x}^n : \mathbf{x}^n \in \mathcal{E} \wedge i = \arg\min_j D(\mathcal{E}, P_j)\right\}$;

MoS models trained using the k-subspaces algorithm will from here on be called *adaptive subspace maps* or ASMs.

8.2.2 Maximum likelihood

The expectation-maximisation (EM) algorithm can be used for clustering probability density models to find the maximum likelihood (ML) solution. Unlike in the k-subspaces algorithm, samples are given a soft assignment to each model in the mixture, according to the relative probability of belonging to it. The EM algorithm will be discussed briefly here, based on the work by Tipping and Bishop [355, 356], which can be consulted for further detail.

Mixtures of probabilistic PCA

In Tipping and Bishop's probabilistic PCA model (henceforth PPCA), a d-dimensional *observed variable* $\mathbf{x}$ is supposed to originate from an m-dimensional *latent variable* $\mathbf{u}$ ($m \leq d$):

$$\mathbf{x} = \mathbf{A}\mathbf{u} + \boldsymbol{\mu} + \epsilon \tag{8.2}$$

where $\mathbf{A} = \mathbf{W}^T$ (cf. section 7.4.2) and $\epsilon \sim N(\mathbf{0}, \sigma^2\mathbf{I})$ is noise. The latent variables are assumed to have a standard normal distribution, i.e.

$$p(\mathbf{u}) = \frac{1}{(2\pi)^{\frac{m}{2}}} \exp\left(-\frac{\mathbf{u}^T\mathbf{u}}{2}\right) \tag{8.3}$$

and the conditional distribution of the observed variables is modelled by a Gaussian:

$$p(\mathbf{x}|\mathbf{u}) = \frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{A}\mathbf{u} - \boldsymbol{\mu})^T(\mathbf{x} - \mathbf{A}\mathbf{u} - \boldsymbol{\mu})\right), \tag{8.4}$$

so that the distribution of $\mathbf{x}$ can be written as:

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{u})p(\mathbf{u})d\mathbf{u} = \frac{1}{(2\pi)^{\frac{d}{2}}} |\det(\mathbf{C})|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T\mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{8.5}$$

in which $\mathbf{C} = \sigma^2\mathbf{I} + \mathbf{A}\mathbf{A}^T$ is the model covariance matrix. The likelihood of observing the entire data set $\mathcal{L}$ is

$$\begin{aligned} \mathfrak{L}_{\text{PCA}} = \sum_{n=1}^{N} \ln p(\mathbf{x}^n) &= N\left(-\frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\det(\mathbf{C})| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})\mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \\ &= -\frac{N}{2}\left(d\ln 2\pi + \ln|\det(\mathbf{C})| + \operatorname{tr}(\mathbf{C}^{-1}\mathbf{S})\right), \end{aligned} \tag{8.6}$$

where $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^n - \boldsymbol{\mu})(\mathbf{x}^n - \boldsymbol{\mu})^T$, i.e. the sample covariance matrix of $\mathcal{L}$.

In a mixture model setting with k subspaces $P_1, \ldots, P_k$, this likelihood becomes

$$\mathfrak{L}_{\text{PCA}} = \sum_{n=1}^{N} \ln\left(\sum_{i=1}^{k} \pi_i p(\mathbf{x}^n|P_i)\right), \tag{8.7}$$

where π_i is the mixing weight ($\pi_i \geq 0$, $\forall i$; $\sum \pi_i = 1$). The *responsibility* of model P_i for generating point $\mathbf{x}^n$ is found in the E-step as

$$R_i^n = \frac{\pi_i p(\mathbf{x}^n | P_i)}{p(\mathbf{x}^n)}. \tag{8.8}$$

The maximum-likelihood solution can be found by taking derivatives of $\mathfrak{L}_{\text{PCA}}$ w.r.t. $\boldsymbol{\mu}$, $\mathbf{A}$ and σ^2 [355]. This gives the update equations for the M-step, for $i = 1, \ldots, k$:

$$\boldsymbol{\mu}_i = \frac{1}{N} \sum_{n=1}^{N} R_i^n \tag{8.9}$$

$$\pi_i = \frac{\sum_{n=1}^{N} R_i^n \mathbf{x}^n}{\sum_{n=1}^{N} R_i^n} \tag{8.10}$$

$$\mathbf{S}_i = \frac{1}{\pi_i N} \sum_{n=1}^{N} R_i^n (\mathbf{x}^n - \boldsymbol{\mu}_i)(\mathbf{x}^n - \boldsymbol{\mu}_i)^T, \tag{8.11}$$

after which $\mathbf{W}_i$ and σ_i^2 can be found by applying standard PCA (eqns. 7.14 and 7.27) based on $\mathbf{S}_i$, and $\mathbf{A}_i = \mathbf{W}_i^T$. Note that the k-subspaces algorithm discussed in the previous section is a limit case of this algorithm, in which $\pi_i = 1$ and $\pi_j = 0$, $\forall j \neq i$.

The soft assignment of samples to subspaces through R_i^n unfortunately makes incorporation of the idea of episodes difficult. Therefore, in experiments in this chapter using the EM algorithm, episode-wise assignment of samples to subspaces was not used. Furthermore, the method is quite a bit slower than the k-subspaces algorithm, especially for a large number of subspaces. In the EM algorithm, all samples participate in the parameter estimation of each model, whereas in the k-subspaces algorithm the estimates are based on just the set of samples assigned to each single model.

A last problem is that the EM algorithm can become unstable when one of the models shrinks to only one point [28]. In that case, $\sigma \to 0$ and $\mathfrak{L}_{\text{PCA}} \to \infty$ (eqn. 8.7). Although there are heuristic ways of circumventing this problem, such as regularising σ or re-initialising collapsed models, these were not used here; the algorithm was simply re-started a maximum of four times if it did not converge.

Mixtures of Gaussians

Eqns. 8.2–8.11 are identical to the updates necessary to train a mixture of Gaussians [28], in which $m = d$, $\mathbf{S}$ is an estimate of the Gaussian covariance matrix $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ and there is thus no need to estimate σ^2. The likelihood $\mathfrak{L}_{\text{Gauss}}$ of a vector $\mathbf{x}^n$ is given by $-D(\mathbf{x}^n, G)$ (eqn. 7.10).

8.2.3 The subspace shift algorithm

The k-subspaces and EM algorithms have in common that they fix the number of models in the mixture and then minimise the average sample-cluster distance. Another approach would be to fix the maximum sample-cluster distance (or *radius*) and attempt to minimise the number of clusters necessary to represent the data. This is how the subspace shift algorithm works, a variation on the mean shift clustering algorithm. In the original mean shift algorithm [47, 60, 61], the algorithm tries to find k clusters by mode seeking. A greedy version of the algorithm introduced for color image segmentation [59] iteratively finds clusters by seeking a single mode and removing neighbouring samples until the data set is exhausted, automatically finding k in the process. The subspace shift algorithm is based on this greedy algorithm.

As for the k-subspaces algorithm, the difference between the mean shift and subspace shift algorithms are that the latter re-calculates subspace projection matrices $\mathbf{W}$ at each step as well as $\boldsymbol{\mu}$, and that the distance measure used is $D(\mathbf{x}, P)$ instead of $D(\mathbf{x}, \boldsymbol{\mu})$. The algorithm consists of two nested iterations:

1. set the working set $\mathcal{W}$ to $\mathcal{L}$;
2. set k, the number of subspaces found thus far, to 1;
3. initialise P_k by calculating $\Theta_k = \{\mathbf{W}_k, \boldsymbol{\mu}_k\}$ on $d+1$ randomly selected samples $\mathbf{x}^n \in \mathcal{W}$, where d is the number of dimensions of $\mathbf{x}^n$, and set $t = 0$;
   (a) find the set $\mathcal{V} \subset \mathcal{W}$ of samples $\mathbf{x}^n$ for which $D(\mathbf{x}^n, P_k) \leq \rho$;
   (b) re-calculate $\Theta_k = \{\mathbf{W}_k, \boldsymbol{\mu}_k\}$ on the samples in $\mathcal{V}$;
   (c) set $t = t + 1$; while the average change in distance is larger than τ and $t < t_{max}$, go to (a).
4. set $\mathcal{W} = \mathcal{W} \setminus \mathcal{V}$;
5. while $|\mathcal{W}| > d$, set $k = k + 1$ and go to 3.

This algorithm has three parameters: the radius of each subspace, ρ; the stopping criterion τ and the maximum number of iterations t_{max}. The latter two were always set to 1.0×10^{-6} and 1,000, respectively. In the remainder of this chapter, subspace-shift trained MoS models will be denoted by ASM$^{\text{ss}}$.

The subspace shift method is fast in training, as subspace parameters are only calculated on the set of points $\mathcal{V}$ assigned to the current subspace P_k. Furthermore, as the working set $\mathcal{S}$ shrinks with each subspace found, less distances will have to be calculated after each iteration of the outer loop.

Method	Model	Clustering algorithm	Data normalisation	Distance equation	Parameter(s)
Mean	mean-only Gaussian	k-means		7.11	k
Gauss	full-cov. Gaussian	EM		7.10	k
$\text{Gauss}_{\text{nrm}}$	full-cov. Gaussian	EM	•	7.10	k
ASM	PCA	k-subspaces		7.26	k, m
ASM_{nrm}	PCA	k-subspaces	•	7.19	k, m
ASM^{ss}	PCA	subspace shift		7.26	ρ, m
$\text{ASM}^{\text{ss}}_{\text{nrm}}$	PCA	subspace shift	•	7.19	ρ, m
PPCA	probabilistic PCA	EM		7.26	k, m
PPCA_{nrm}	probabilistic PCA	EM	•	7.26	k, m

Table 8.1: The mixture models used in the experiments in this chapter. In the "Parameters" column, k indicates the number of models to fit, m the number of dimensions per model and ρ the subspace radius.

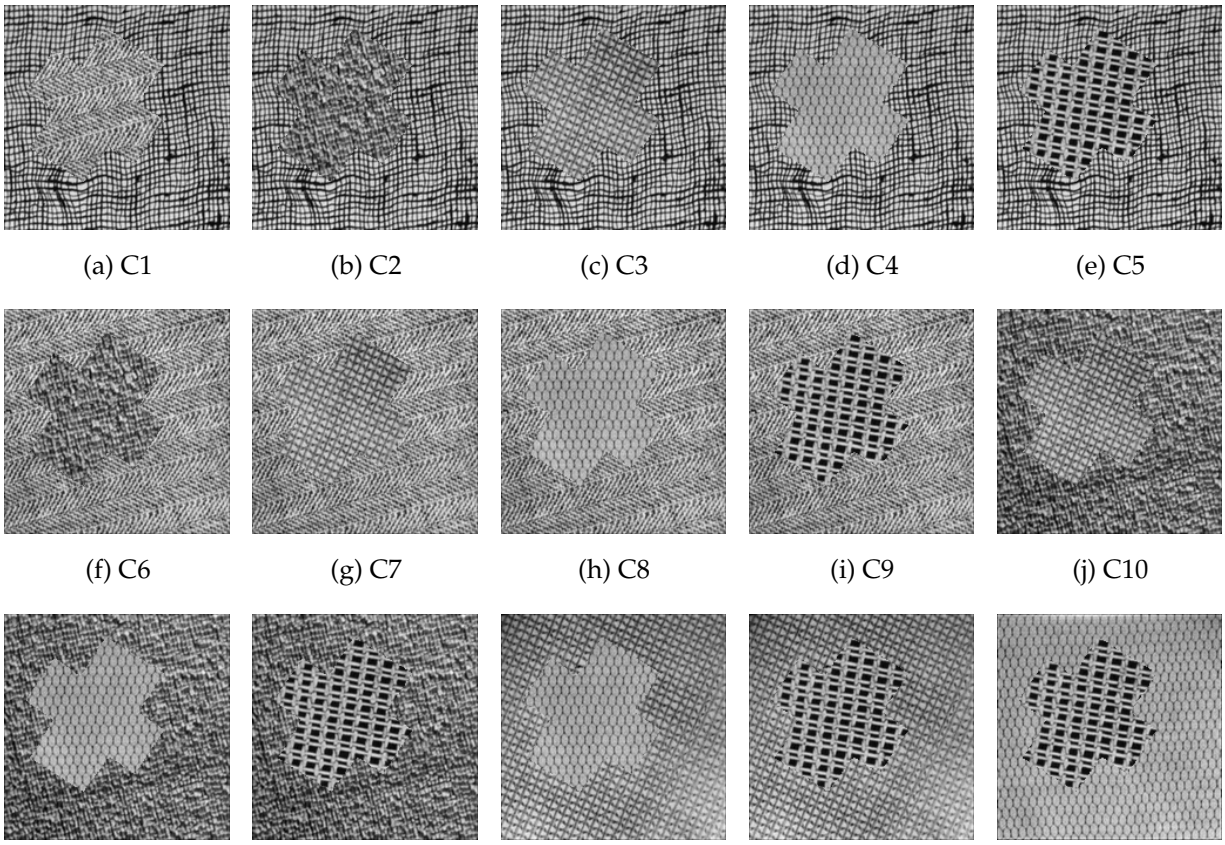
8.2.4 Model overview

The various models discussed in section 7.4 can be combined with the clustering algorithms above in a number of ways. As the goal was to use the original distance formulations (i.e., the negative likelihood) where possible, for all models pre-mapping was applied (retaining $r = 90\%$ of the variance, see section 7.3.2) to avoid degeneration of these measures. Besides being necessary for some models, this pre-mapping also speeds up the training process considerably by lowering d, the number of dimensions of the training set. Furthermore, the k-subspaces and EM algorithms were initialised by choosing random orthogonal subspace basis vectors, but setting the origins $\boldsymbol{\mu}$ to vectors containing the average grey value of cluster centres found by simple k-means clustering on individual image pixels. For the EM algorithm, finally, σ was initialised to 1. Table 8.1 gives an overview of the methods and their parameters. As algorithms trained on normalised data (see section 7.3.2) have different characteristics, i.e. $\boldsymbol{\mu}_i = \mathbf{0}\ \forall i$, these are treated as different models.

There is no reason to favour one model a priori. On the one hand, the k-subspaces and subspace shift algorithms allow the idea of episodes to be used during training. On the other hand, in real segmentation problems there are border effects (i.e. windows in which more than one texture occurs), to which the EM algorithm may be more robust due to its soft assignment procedure.

8.3 Texture segmentation

The mixture models listed in 8.1 can be tested as texture segmentation methods. To this end, of the set of 6 structured Brodatz textures used in chapter 7 (see figure 7.3) all 15



(a) C1 (b) C2 (c) C3 (d) C4 (e) C5

(f) C6 (g) C7 (h) C8 (i) C9 (j) C10

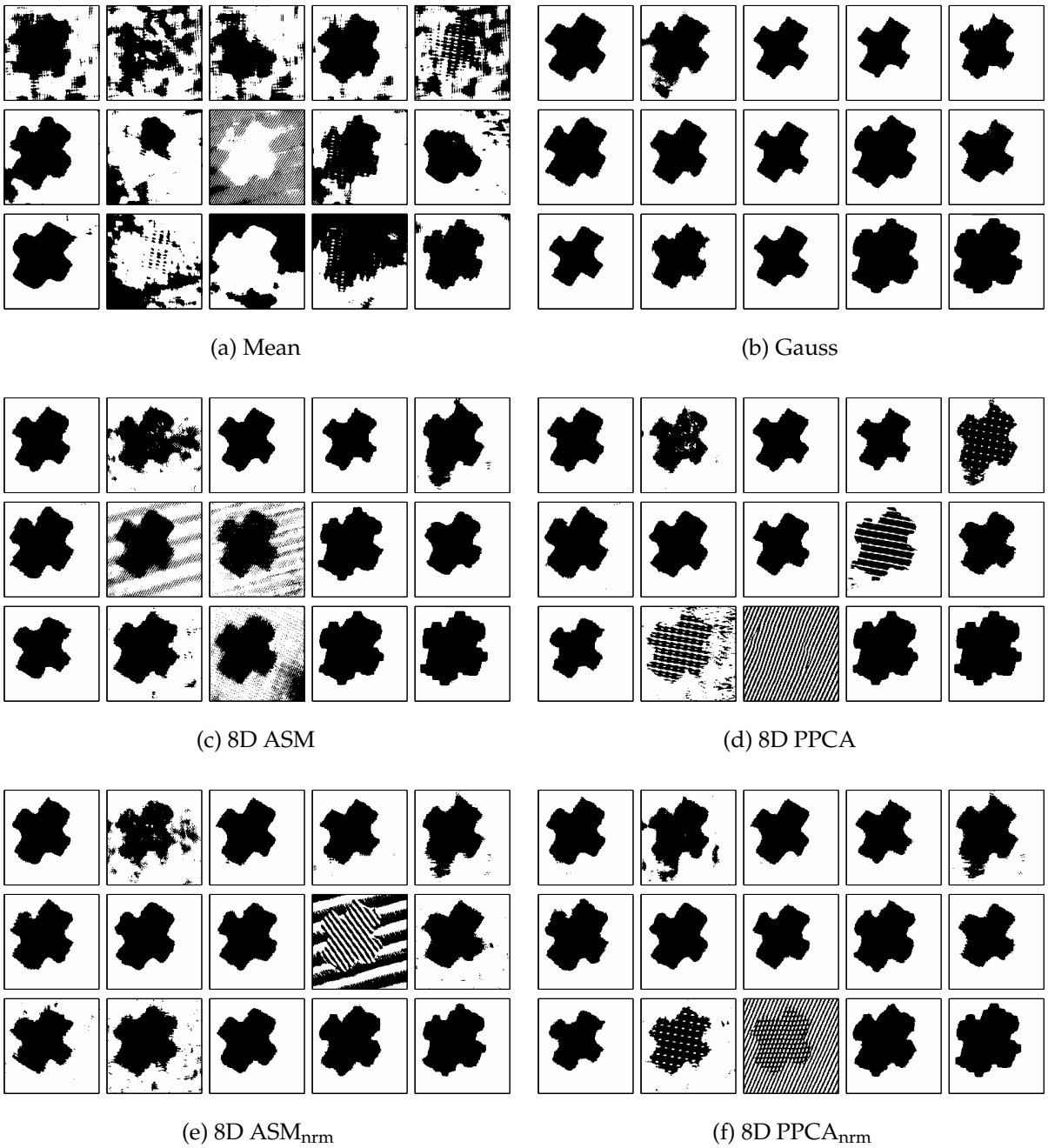
(k) C11 (l) C12 (m) C13 (n) C14 (o) C15

Figure 8.1: The 15 structured Brodatz texture combinations.

possible combinations of two textures were created using a cross-shaped mask image, following [103]. Figure 8.1 shows these combinations. The methods were not tested on combinations of natural textures, as they were already shown to perform poorly on these textures in chapter 7. Furthermore, no rotated or scaled versions of the textures were used, as the effect of these transformations on the descriptive power of the models was already studied in section 7.5.7.

8.3.1 Segmentation

All models were trained on the combination images. From each image, 3,000 samples were extracted using either the translation-only or the all-transformation episode method (see section 7.3). Rectangular $w \times w = 16 \times 16$ pixel masks were used; pre-mapping typically left 40-70 of the original 256 dimensions. The number of subspaces per model, k, was set to 2; the number of dimensions per subspace, m, was chosen as 4, 8, 12 or 16. Figure 8.2 shows segmentation results for 8D models trained on the tex-



(a) Mean (b) Gauss

(c) 8D ASM (d) 8D PPCA

(e) 8D ASM_{nrm} (f) 8D PPCA_{nrm}

Figure 8.2: Structured texture combination segmentations using translation-only episodes. In each figure the ordering is the same as in figure 8.1 (a).

ture combinations. These images were created by *mapping* each pixel; i.e. finding the subspace P_i to which the window around that pixel is closest, and assigning it label i. Note that the segmented texture combinations are not strictly test images, as training sets were extracted from the same images.

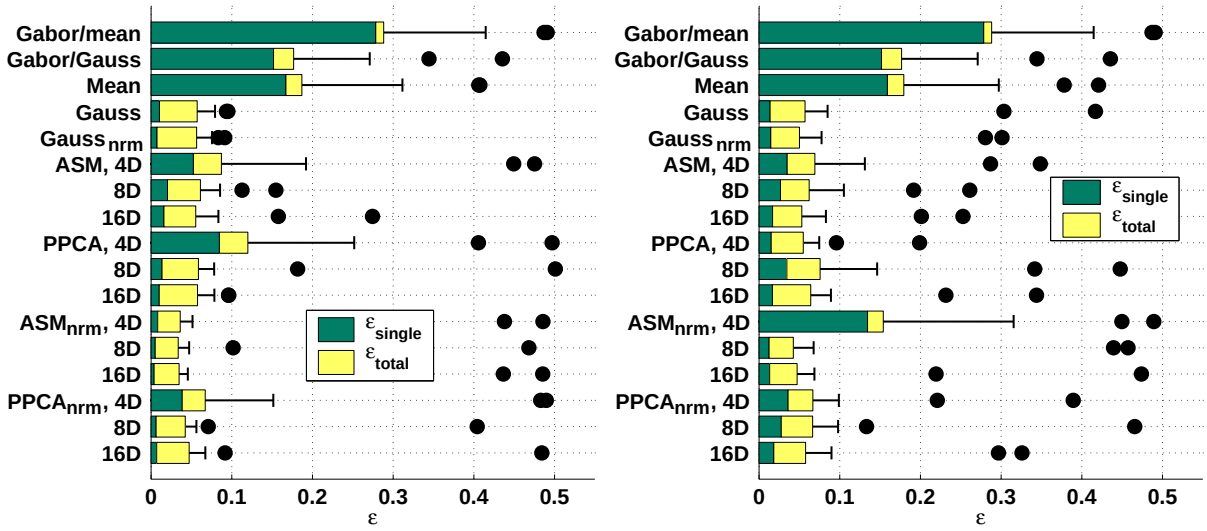
The segmentations show that the mean-only Gaussian model (figure 8.2 (a)) is ill-equipped to handle illumination differences. Textures are segregated on average local grey value only, so some texture combinations are segmented incorrectly. On the contrary, the full-covariance Gaussian model gives near-perfect segmentation results (figure 8.2 (b)). The only problems occur in combination C2, where texture S5 (see figure 7.3 on page 133) is a little irregular and at the border between textures: sometimes the segmented cross is a little too large (e.g. in C14 and C15). The first problem indicates that segmentation result depends not only on the descriptive power of the models used, but also on the texture combinations actually present. Although texture S5 occurs in combinations C1–C5, its description only overlaps with that of the other texture in the second one.

The MoS models (ASM, PPCA, ASM_{nrm} and PPCA_{nrm}, figures 8.2 (c)-(f)) all give reasonably similar results. Note that only segmentation results for 8D subspaces are shown; other results will be given as segmentation errors only. Each of the models performs less well on one or two textures, and most fail to some extent on combination C13. The ASMs seem to have problems mostly with modelling the two directions in the herringbone weave texture (S1). For all models, normalisation seems to help, as overall the segmentations seem to be slightly better. However, for some combinations normalisation also introduces problems; note for example the difference for the fourth texture in the second row between ASM and ASM_{nrm} (figures 8.2 (c) and (e)).

8.3.2 Segmentation errors

Segmentation errors were also calculated for all methods, as the relative number of pixels segmented incorrectly. As the results in figure 8.2 show that many errors occur around the border, two error percentage were calculated: ε_{total}, the error percentage over all pixels; and ε_{single}, the error percentage over all pixels in windows with just a single texture present, i.e. with a shortest distance of more than $\lceil \sqrt{2}\frac{w}{2} \rceil = \lceil \sqrt{2}\frac{16}{2} \rceil = 12$ pixels to the texture border. Figure 8.3 shows the results. As the average error percentages thus calculated are heavily influenced by the one or two texture combinations that are segmented poorly, the 10% largest errors for each method were not included in the average error, but plotted separately as dots. The lines indicate the standard deviation in the ε_{total} results.

The results confirm that the full-covariance Gaussian model and the subspace models (ASM, PPCA, ASM_{nrm} and PPCA_{nrm}) perform best; much better than the mean-only Gaussian model and the Gabor-filter based models. Some of the lower-dimensional



(a) Translation-only episodes

(b) All-transformation episodes

Figure 8.3: Texture combination segmentation errors, in fraction of pixels segmented incorrectly. All bars show two levels: the highest level is the overall segmentation error ε_{total}, the lowest level is the segmentation error when just pixels outside a range of 12 pixels from the border are considered, ε_{single}. Lines indicate the standard deviation of ε_{total}; dots show the 10% largest errors, which were not included in ε_{total} and ε_{single}.

subspace models (4D ASM, 4D PPCA_{nrm} on translation-only episodes; 4D ASM_{nrm} on all-invariance episodes) give relatively large errors, indicating that at least 8 dimensions are needed to get reliable results. Indeed, these higher-dimensional subspace models either perform as well as the Gaussian model, or (in one or two cases) fail completely. Models trained on normalised data perform marginally better than those trained on the original data. As was already noticed, most of the errors occur near the border: ε_{total} is much larger than ε_{single} for all models.

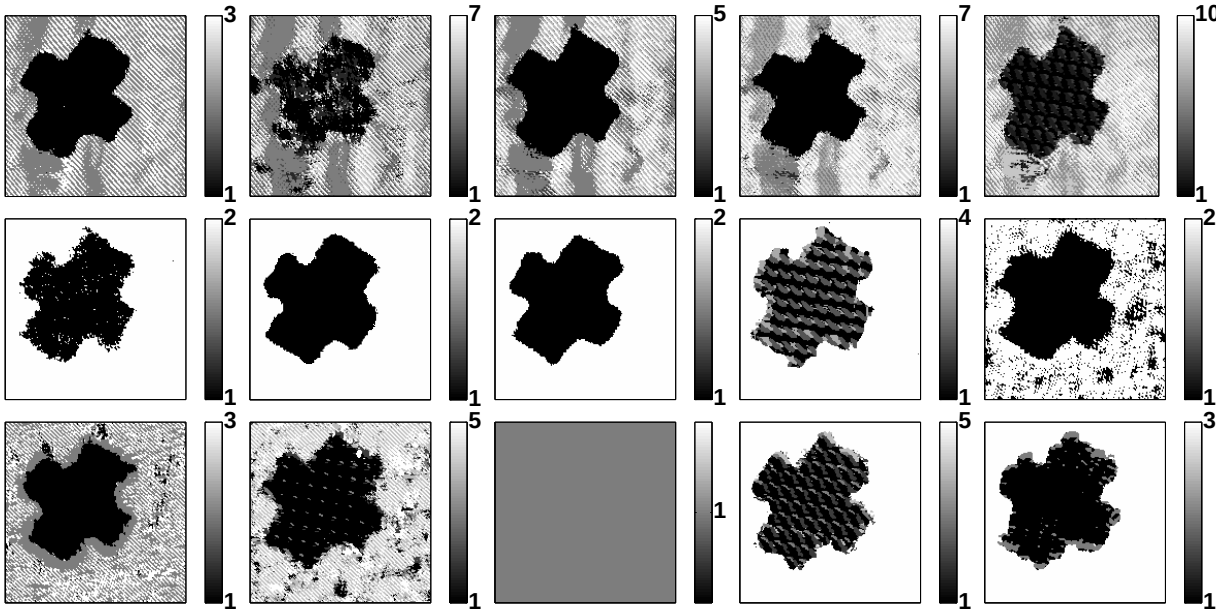
Training on all-transformation episodes increases the error only slightly for most models, while it has already been shown to make them invariant over a larger range of rotations and scale (cf. section 7.5.7). At the same time, the texture combinations for which segmentation failed using translation-only episodes (the dots in the figures) seem to give even more problems; compare, for example, the results for the Gaussian model between figures 8.3 (a) and (b). On all-transformation episodes, the 8D and 16D ASM_{nrm} models seem to perform slightly better than PPCA_{nrm}, which indicates that the ASM_{nrm} benefits from the use of episodes.

8.3.3 Subspace shift-trained models

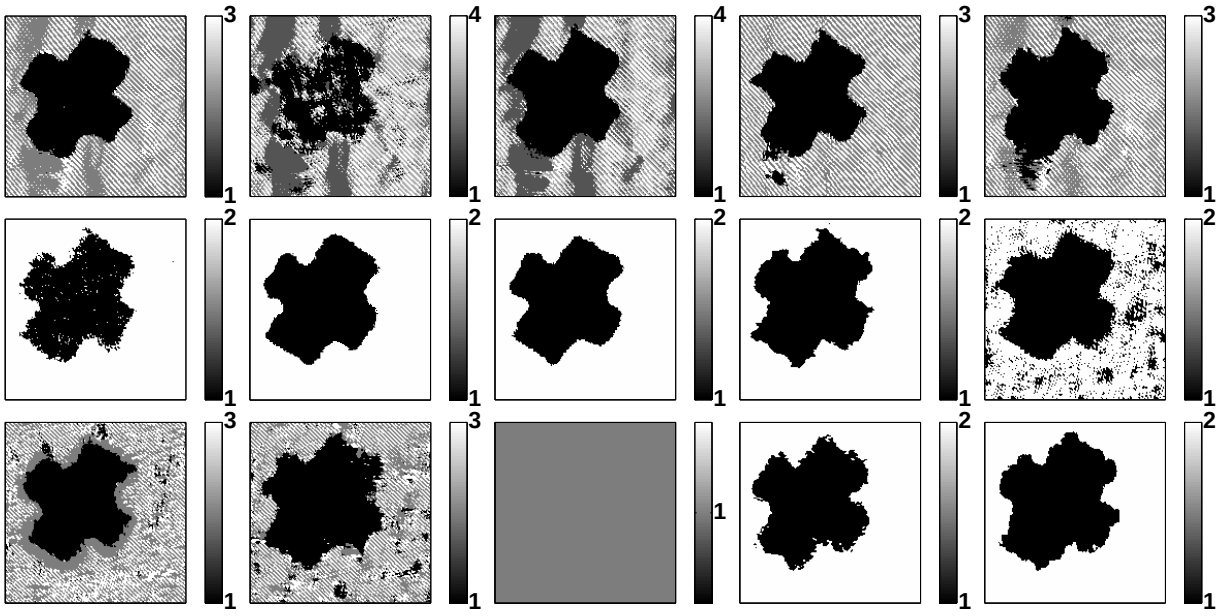
The subspace shift algorithm was not included in the results presented above, as it did not always give useful results. An obvious problem with this algorithm is that the radius of the subspaces will have to be specified beforehand, which might be problematic. Moreover, it is identical for each subspace, which in the case of textures is far from optimal. However, since episodes are used (identically to their use in the k-subspaces algorithm), some idea of the right setting of this parameter can be deduced from the episode distance. If each episode is supposed to be represented by a single subspace, that subspace should have a radius of at least the distance of that episode to it (eqn. 8.1). Formally, this can be achieved by calculating the initial subspace P_k in step 3 of the algorithm on page 173 on a single, randomly selected episode $\mathcal{E}$. The radius ρ for that subspace can then be set to $\alpha \cdot D(\mathcal{E}, P_k)$, where α is some multiplier to prevent the subspace fit from becoming too tight. This multiplier will still have to be specified, but allows different subspaces to have different radii. Note that fixing ρ in this way is rather sensitive to the exact choice of the randomly selected episode, so repeated application of the algorithm might give different results.

Experiments using this method showed that it only works when there is sufficient subspace structure in the data. If the subspace contains more dimensions than are necessary to describe the data (which is often the case for single translation-only episodes) a reliable estimate of ρ is hard to find for each texture combination. Only for the all-transformation episodes the method works reasonably well; here, α was set to 3. Figure 8.4 (a) shows the results. Most images have been over-segmented, i.e. they contain more than two segments. Some of these images were segmented poorly before, e.g. C12; in this image, texture S1 (herringbone weave) is represented by two subspaces, each of which represents one dominant direction of the texture. In C1–C5, additional subspaces model the illumination difference in the background texture. On C13, only one subspace was found, as ρ was too large due to the choice of α; however, this was also the single combination on which most subspace models failed completely.

The segmentations show that some of the subspaces represent only very small parts of the image. A simple post-processing step would therefore be to discard those subspaces representing too small a portion of the training data. When all subspaces to which less than 15% of the episodes in the training set were assigned are removed, there is less over-segmentation; see figure 8.4 (b). Still, the over-segmentation left makes it hard to compare this algorithm to the others in terms of segmentation error. Furthermore, using a threshold on the size of the set of pixels assigned to each subspace introduces another parameter, of which the optimal setting depends on image scale and size.



(a) Original segmentation



(b) After subspace pruning

Figure 8.4: 8D $\text{ASM}^{\text{ss}}_{\text{nrm}}$ segmentation results on the 15 texture combinations using all-transformation episodes. In each figure the ordering is the same as in figure 8.1 (a). The bars indicate the number of subspaces.

8.3.4 Discussion

The experiments in this section showed how MoS models can be used for texture segmentation. Normalisation of the data before training was shown to improve performance. There is no significant difference in results between the k-subspaces and EM algorithms; however, the former is faster and allows the use of episodes in training. Although the training sets used by the EM algorithm also contained these episodes, there is no guarantee they are actually used as such; there is no mechanism in the EM algorithm to force it to use episodes as single entities. Finally, the subspace shift algorithm was shown to be less applicable to texture segmentation as it has a tendency to over-segment.

In this section, ASMs were solely considered as texture segmentation tools. However, an ASM trained on samples (or episodes) collected from images, or even classes of images, can also be seen as a descriptor of that image. This opens up a variety of possible applications based on image description. In the next section, ASMs will be used for image database retrieval, in which it is often a problem to define feature sets such that image content can be described in a compact way.

8.4 Image database retrieval

A large body of literature exists on indexing images based on their texture content – see e.g. [9, 331] for reviews. As Antani et al. [9] note, it is impossible to define a good set of features a priori for a wide variety of images; therefore, the best approach is to be adaptive. However, some mechanism should be present to shield the end user from the actual implementation. One approach is to find an optimal subset of standard features, as Alexandrov et al. [4] did. They used a large number (120) of Gabor filters, from which indices were created using feature selection. Of course, one can use any of a number of texture representation methods. Pentland et al. [275] used Wold components for textures; Kelly et al. [190] used histograms of texture masks evaluated at each pixel in an image and developed a method of approximating and comparing these histograms. Ramesh and Sethi [289] looked at edges, or more precisely, wavelet features extracted at points of high curvature in edge descriptions.

In the ASM[1] approach, the feature extraction and feature selection stages are rolled into one and performed automatically. All that remains to be able to apply ASMs to image database retrieval is to define a way of using the ASMs to measure distances between (classes of) images, say I_A and I_B. There are two possible strategies:

[1] In this section and section 8.5, only ASMs trained on normalised data will be used. For brevity, the term ASM will be used instead of the name ASM_{nrm} used in the previous section.

1. train ASMs A and B on images (or classes of images) I_A and I_B and use these as a descriptor;
2. train an ASM A on image (or class of images) I_A and use the histogram of pixels in image I_B assigned to subspaces in ASM A as a descriptor.

The second option seems to be preferable, as it defines distances between images I_A and I_B in terms of the content of both images and the size of the regions of similar content. However, it requires assigning all pixels in each new image (or query image) to all maps found so far, which is a computation-intensive task. The first option is computationally much lighter, but discards information on the size of the regions responsible for each subspace in the map. For the image database retrieval application, the first option will be used, and a way of taking the image region sizes at least partially into consideration will be incorporated. For object recognition, the second option is explored, in section 8.5.

8.4.1 ASM distance measures

A distance measure between ASMs A, with k^A subspaces P_i^A defined by parameters $\Theta_i^A = \{\mathbf{W}_i^A, \boldsymbol{\mu}_i^A\}$, and B, with k^B subspaces P_j^B can be defined as follows:

$$D(A,B) = \max(D'(A,B), D'(B,A)) \tag{8.12}$$

$$D'(A,B) = \frac{1}{k^A}\sum_{i=1}^{k^A} \min_{j=1,\ldots,k^B} D''(P_i^A, P_j^B) \tag{8.13}$$

$$D''(P_i^A, P_j^B) = \frac{1}{m}\sum_{l=1}^{m} ||\mathbf{e}_l^{A_i} - \hat{\mathbf{e}}_l^{A_i}||^2 \tag{8.14}$$

where each m-dimensional subspace P is spanned by basis vectors $\mathbf{W} = [\mathbf{e}_1 \ldots \mathbf{e}_m]^T$ and $\hat{\mathbf{e}}_l^{A_i}$ is the projection of basis vector l of subspace P_i^A onto subspace P_j^B. For convenience, in the rest of this discussion all subspaces are assumed to have zero mean (as is the case for the ASM_{nrm} model). The idea behind this measure is to find, for each subspace in ASM A, the closest subspace[2] in B and average this distance over all subspaces in A. The same is done for B and the maximum is taken, like in the Haussdorf distance (see e.g. [169]).

The problem with this distance measure is that all information about the size of the regions responsible for the subspaces is discarded. A possible solution for this is to

[2]Experiments were also performed in which a more principled distance measure between subspaces called the *gap* [341] was used: $D''(P_i^A, P_j^B) = ||\mathbf{A}_i^A - \mathbf{A}_j^B||_2$, where $\mathbf{A} = \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ is the backprojection matrix onto the subspace spanned by $\mathbf{W}$. However, the computational burden of this method was much higher due to the singular value decomposition needed for the calculation of the norm, and results were nearly identical.

weigh the distance between a subspace P_i^A and an ASM B with the relative importance ζ_i^A of P_i^A in ASM A:

$$D'(A,B) = \frac{1}{k^A} \sum_{i=1}^{k^A} \zeta_i^A \min_{j=1,\ldots,k^B} D''(P_i^A, P_j^B) \tag{8.15}$$

There are various ways of expressing the importance ζ_i^A of subspace P_i^A. First of all, subspaces can be weighted equally:

$$\zeta_{i,0}^A = 1, \ \forall i. \tag{8.16}$$

Two other possible weight measures are:

$$\zeta_{i,1}^A = \frac{h_i^A}{\sum_{j=1}^{k^A} h_j^A}, \tag{8.17}$$

where h_i^A is the i^{th} bin of the histogram $\mathbf{h}^A$ of the assignment of pixels in the training image to ASM A, and

$$\zeta_{i,2}^A = \frac{h_i^A(1-\bar{\epsilon}_i^A)}{\sum_{j=1}^{k^A} h_j^A}, \tag{8.18}$$

where $\bar{\epsilon}_i^A$ is the average projection error of the pixels assigned to subspace P_i^A (eqn. 7.19). These measures express the ideas that subspace distances should be weighted according to their importance (using $\mathbf{h}$) and descriptiveness (using $1-\bar{\epsilon}$).

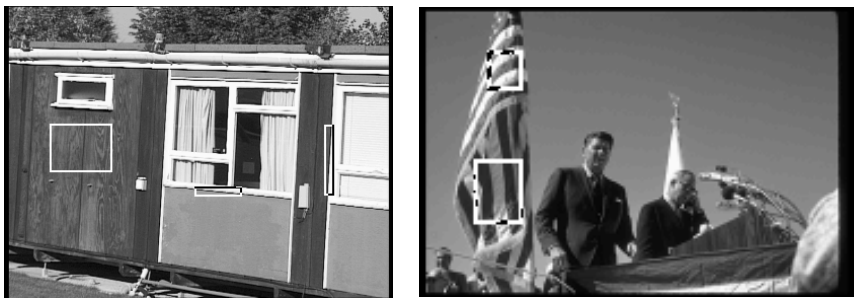
8.4.2 The data set

A small database of 200 images was created, containing images taken randomly from the MPEG7 database and stills from an hour-long video sequence of Sky news. Besides these images, five sequences of similar images of a news reader ("news reader"), Her Royal Highness the Queen ("queen"), Guildford cathedral ("cathedral"), a hut ("hut"), and a US flag ("flag") were added. The goal of the experiment was, given an image in a sequence, to find the other images in that sequence. Examples of the images in these sequences are given in figure 8.5. All 200 database images, originally 24-bit colour, were converted to grey values and histogram stretched before samples were taken.

8.4.3 Measures

On each individual image, ASMs were trained for various settings of the parameters. To learn about the influence of the episodes construction methods on this type of application, experiments were run using both the translation-only and all-transformation

(a) (b) (c)



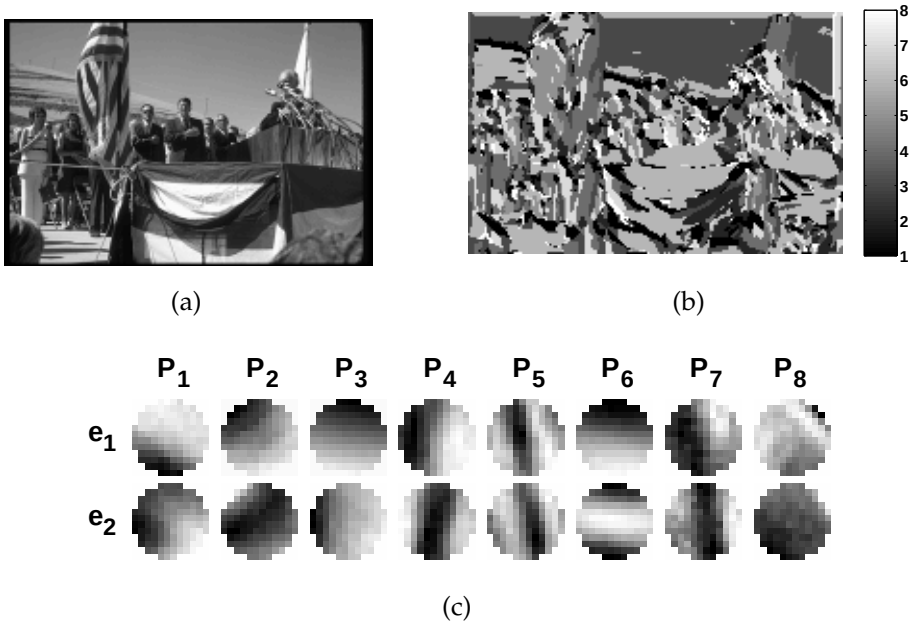
(d) (e)

Figure 8.5: Example images from the five sequences, from left to right: "news reader", "queen", "cathedral", "hut" and "flag". The regions indicated were used only in the KIDS system.

episode extraction methods (see section 7.3) to extract 200 episodes from each image. Pre-mapping was not applied, as this could remove detail from the data set which may be important. To reduce the number of dimensions the method has to deal with, a round sampling mask (see section 7.3) was used, with a diameter of $w \in \{8, 12, 16\}$ pixels. The experiments were run using ASMs with $k \in \{4, 8, 12, 16, 20\}$ subspaces of $m = 2$ and $m = 4$ dimensions, respectively. An example of a trained ASM and the corresponding segmented database image is given in figure 8.6.

After training, for each n-image sequence $\mathcal{Q}$, one image $I_{\mathcal{Q}}^{tst}$ was used as a test image. The distance $D(A_{\mathcal{Q}}^{tst}, A_i), i = 1, \ldots, 199$ between the ASM $A_{\mathcal{Q}}^{tst}$ trained on this image and all other ASMs was calculated using equations 8.12-8.18 and the images were ordered by increasing distance of their ASM to that of the test image. Finally, the ranks $r_1 \ldots r_{n-1}$ (starting from 0) of the other images in this ordered sequence $\mathcal{Q}$, $I_{\mathcal{Q},i}^{trn}$ were noted.

It is not easy to find a single quality measure for image database retrieval. Perhaps one of the simplest and most intuitive is the *normalised recall* measure proposed by Faloutsos et al. [102] in their discussion of IBM's QBIC (Query By Image Content, [11, 107, 256]) system. Given a number of ranks $r_i, i = 1, \ldots, n-1$, the average retrieval rate (ARR) is



(a) (b) (c)

Figure 8.6: (a) One of the images in the "flag" sequence; (b) segmentation by a $k = 8, w = 12$ ASM trained on translation-only episodes taken from that image and (c) the basis vectors of that ASM. Note the vertical and horizontal textures in the flag found by P_5, P_4 and P_6, respectively.

defined as:

$$ARR = \frac{1}{n-1} \sum_{i=1}^{n-1} r_i. \tag{8.19}$$

for any $n \geq 2$. The *ideal* average retrieval rate is:

$$IARR = \frac{1}{n-1} \sum_{i=1}^{n-1} i - 1, \tag{8.20}$$

allowing a definition of normalised recall as:

$$R = \frac{IARR}{ARR} = \frac{\sum_{i=1}^{n-1} i - 1}{\sum_{i=1}^{n-1} r_i}. \tag{8.21}$$

This value will be $R_{max} = 1$ for query results in which the images sought occupy the first ranks, and will be close to zero in the worst case, in which the images sought are found last. Say there are N images in the database, then for a sequence of $n - 1$ images the worst possible normalised recall value is:

$$R_{min} = \frac{\sum_{i=1}^{n-1} i - 1}{\sum_{i=1}^{n-1} N - i} \approx \frac{n-2}{2N} \tag{8.22}$$

which will quickly go to zero for increasing N. For $N = 200$ and $n = 5$, $R_{min} = 7.6 \times 10^{-3}$. If the ranks are drawn from a uniform random distribution over the range $[0, 198]$, simulation shows that $R_{rnd} \approx 1.7 \times 10^{-2}$. Note that the measure drops quickly; e.g. if the 4 test images in $\mathcal{Q}$ are found at ranks 10-13, $R = 0.14$; if they are found at ranks 20-23, $R = 0.07$.

8.4.4 The KIDS system

As a comparison, the same queries were performed using a state-of-the-art system called KIDS (or Kieron's Image Database System, [239, 241]). This system can also handle colour features, but for the comparison only texture features (local variance in 9 DCT filtered versions, 8 Gabor filtered versions and 4 wavelet filtered versions of the original image) were used. Basically, KIDS segments each database image using colour and texture information, and stores segment locations, sizes and median feature values. Retrieval is based on training a two-output unit feed-forward neural network to distinguish between a region the user specified in the query image (output 1) and a randomly drawn set of regions in the database (output 2). Any region in the database for which network output 1 is larger than a pre-set threshold T is then labelled as a match. Finally, a similarity measure for entire images is based on a scoring function incorporating both the network output and the size of any matched regions.

The regions specified for KIDS are shown in figure 8.5. A network output threshold of $T = 0.5$ was used which gave optimal overall results in terms of ranking.

8.4.5 Experiments

Using subspace weight measure $\zeta^A_{i,2}$ (cf. equation 8.18) gives for most queries by far the best results. Therefore, all results reported from here on have been found using this subspace weight measure. Figure 8.7 shows normalised recall values calculated for each query and the settings described earlier, using weight measure $\zeta^A_{i,2}$. In each of the images, the recall values can be compared with that obtained using KIDS. Table 8.2 summarises the graphs by showing normalised recall values and rankings for optimal settings for each query.

The results show that the ASM method gives promising results, even compared to KIDS. For most of the query images the other images in the sequence are ranked high. The results show no clear preference for training on all-transformation episodes. Also, the optimal settings vary quite a bit for the different queries. For most, a window size of $w = 8$ and a large number of subspaces (12-20) seems to be optimal. This is likely to be due to the relatively small images in the database, most of which had a size of 144×192 pixels and depicted large scenes. As a consequence, they are likely to contain a large number of regions of fine texture.

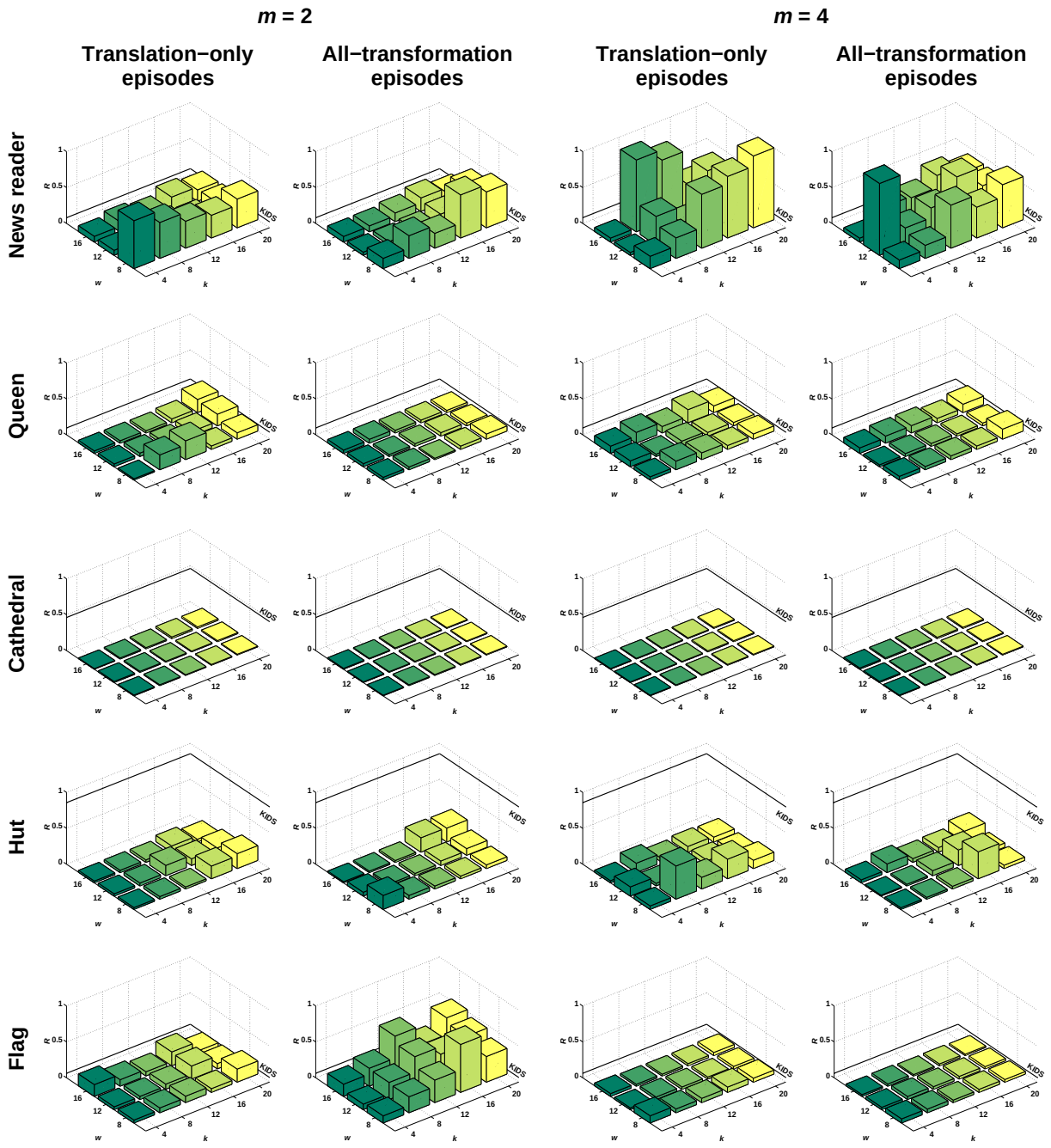


Figure 8.7: Image database query results: normalised recall R for the five queries and various ASM settings.

Sequence	ASM						KIDS	
	Parameter settings				R	Ranking	R	Ranking
	w	k	m	episodes				
"news reader"	8	20	4	translation-only	1.000	[0, 1, 2, 3]	0.080	[17, 19, 20, 23]
"queen"	8	12	2	translation-only	0.269	[2, 4, 9, 12]	0.095	[14, 15, 17, 21]
"cathedral"	16	16	2	translation-only	0.022	[8, 25, 65, 183]	0.462	[2, 4, 5, 6]
"hut"	8	8	4	translation-only	0.462	[1, 2, 4, 10]	0.857	[1, 2, 3, 5]
"flag"	8	16	2	all-transformation	0.667	[1, 2, 3, 4, 10]	0.063	[24, 25, 30, 35, 50]
"cathedral" (2)	8	12	4	all-transformation	0.048	[4, 5, 23, 97]	-	

Table 8.2: Optimal results for the image database queries for both the ASM and KIDS.

Although on most queries results are quite good, a notable exception is the "cathedral" sequence. This is the only sequence consisting of high-resolution images, with areas of fine texture; it was shot using a digital video camera[3]. A hypothesis is that it is this high-frequency content that causes problems, or that the camera might have introduced artifacts into the texture due to aliasing. Figure 8.8 corroborates this; it shows the images in the "cathedral" sequence with some of the ASM basis vectors corresponding to specific textured regions. Due to the aliasing, the episodes constructed by rotating and scaling do not enforce these invariances well. Therefore, the ASMs represent the texture quite precisely at a certain orientation and scale, and different views of the cathedral lead to quite different ASMs, resulting in large distances. Interestingly, KIDS performs best on this query, indicating that the two techniques might to a certain extent be complementary.

To verify that it was indeed the high frequencies in the "cathedral" images that played a role, the "cathedral" images were sub-sampled to one half their original size, using bi-cubic interpolation. The query was then re-run, excluding the original "cathedral" images. Performances, did indeed improve to a more reasonable level; in table 8.2, the new optimal results are shown as "cathedral" (2).

8.4.6 Discussion

The problems with the "cathedral" query, and inspection of other ASMs (not shown here) indicate that the ASMs are more useful when they are coding large-scale image structure, e.g. edges, than they are when coding small-scale texture. If such textures are modelled precisely, as they were for the original "cathedral" query, the same texture under different viewing angles will create two subspaces with a relatively large distance. The idea that ASMs are useful when coding structure rather than fine texture is also supported by some of the query returns, such as the ones shown in figure 8.9.

[3]The "hut" sequence was shot using the same camera, but does not contain texture as fine as the "cathedral" sequence.

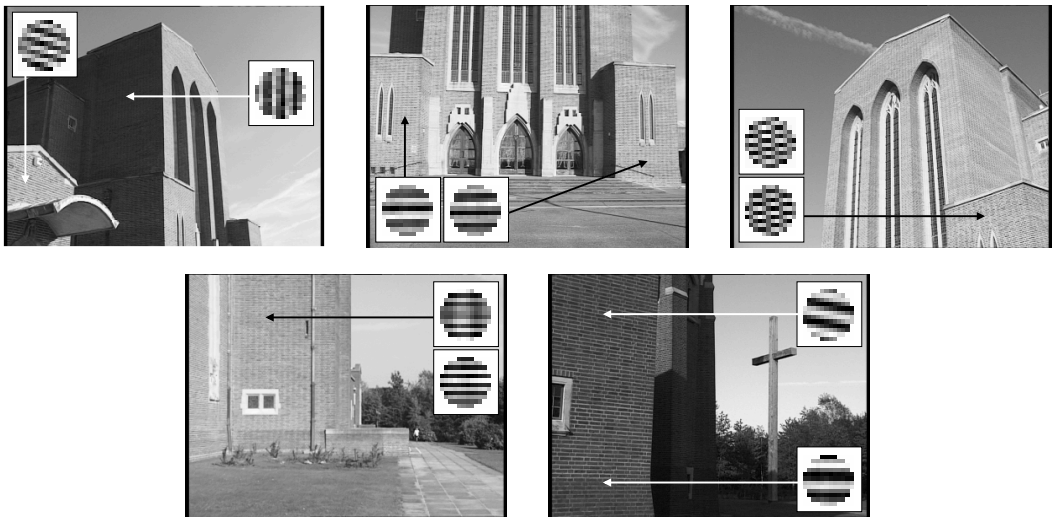


Figure 8.8: The 5 "cathedral" query images. The inserts show some basis vectors tuned to specific textures and specific orientations.

Note how for the "hut" query, which contains a large number of horizontal and vertical edges, images containing quite different textures but identical edges are found.

While it was possible to find reasonable overall ASM settings (w, k and m), the actual optimal settings were different for each query, which would pose a problem in real-world application. Of course, this is to be expected, as discriminative features will be found on different scales and in different quantities in each image. For easy applicability, though, the settings would have to be found automatically. For example, m could be optimised, per subspace, by demanding that a certain percentage r of the variance in the original data be explained by the subspace (e.g. 80%); or the subspace shift algorithm could be applied, for which the radius ρ may be easier to specify than the k for the ASMs used here. The only parameter that cannot be optimised well is the window size w, as it needs to be fixed throughout the database to allow ASM comparison. However, ASMs with different window sizes could be used and the resulting distance measures could be combined.

8.5 Object recognition

In the previous section, ASMs were used as for description of single images. However, the algorithms can trivially be trained on a *class of images*, simply by sampling the data set from images representative of that class. As discussed in section 8.4 on page 182, the histograms of image pixels assigned to such a class ASM can then be used for classification. In this section, this idea wil be illustrated on a simple object recognition problem.

Figure 8.9: Returns for the "hut" query, using an all-transformation ASM ($w = 8$, $k = 12$ and $m = 4$). The original query image is shown in the top-left.

The idea of using assignment histograms has been explored earlier by e.g. Idris and Panchanathan [178], who use vector quantisation on an entire set of images and use histograms of images assigned to the code book vectors as descriptors. Another example is the work of Lampinen and Oja [202], in which clusters are found in a space spanned by Gabor filters at different resolutions, and a supervised layer is applied for classification.

8.5.1 Experiments

A small data set of images of 6 different classes (book cases, chess pieces, mugs, workstations, a tea flask and some bridges) was created [216]. All images were acquired using a Sony digital camera, re-sized to 320×240 pixels, converted to grey values and histogram stretched. Per class, 9 training images and 6 test images were used. The intra-class variation between objects was quite high, since objects were photographed at different distances and against different backgrounds. Also, the inter-class distance was kept low for the object images (chess pieces, mugs and the tea flask) by taking photographs of each of these against three different backgrounds. For some examples, see figure 8.10.

Figure 8.10: Some training images in the object image data set: book cases, workstations, bridges, chess pieces (3×), mugs (3×) and tea flasks (3×).

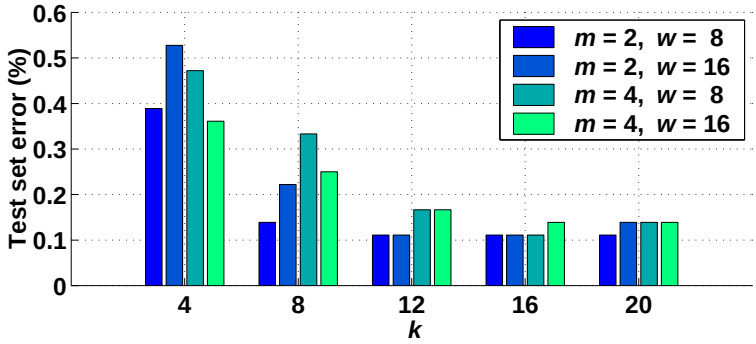


Figure 8.11: Test error for various settings of the number of subspaces k, the sample window size w and subspace dimensionalities m.

Figure 8.12: The four incorrectly classified images.

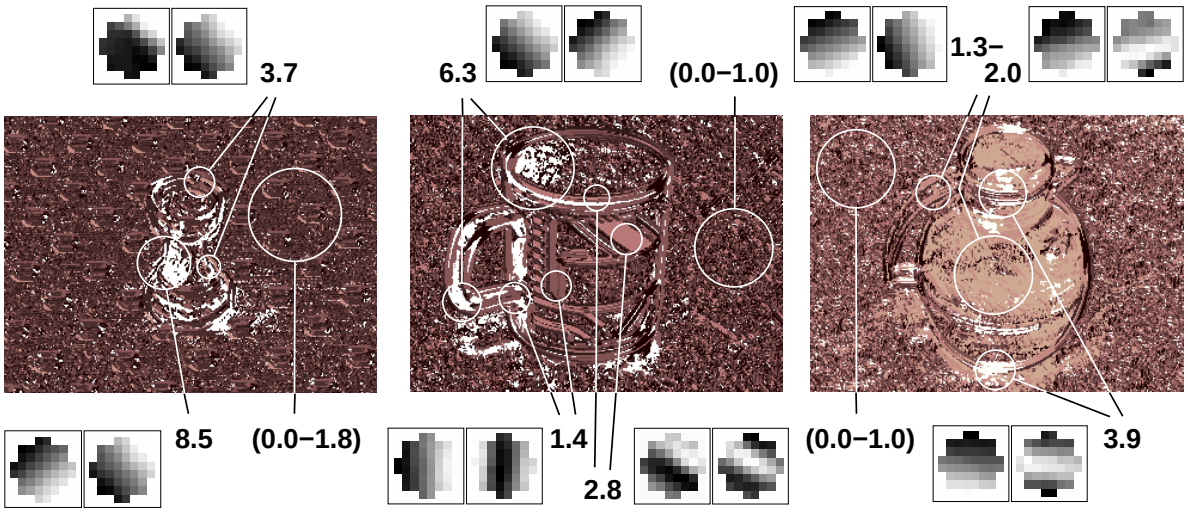


Figure 8.13: Feature evaluation measures for an image containing a chess piece (left), a mug (middle) and a tea flask (right). For some regions, the corresponding subspace basis vectors are shown, together with the feature rank.

An ASM A_c with k subspaces of m dimensions each was trained for each class $c = 1, \ldots, 6$, on 900 translation-only episodes taken from 9 training images I_c^{trn}, using a round window with a diameter of w pixels. No data pre-mapping was applied. After training, for each class c all training images $I_{c,i}^{trn}$ were pixel-wise assigned to their ASM A_c, and a k-bin class histogram $\mathbf{h}_{c,i}^{trn}$ was created by counting the relative number of pixels assigned to each subspace (i.e. the number of pixels assigned to a subspace divided by the total number of pixels in the image). The mean $\boldsymbol{\mu}_c$ and covariance matrix $\mathbf{C}_c$ of these histograms were then used as class descriptions.

Each test image $I_{l,j}^{tst}$ (class $l = 1, \ldots, 6$; image $j = 1, \ldots, 6$) was pixel-wise assigned to each of the class ASMs A_c. The histograms $\mathbf{h}_{l,j,c}^{tst}$ of these assignments were then used to calculate the Mahalanobis distance to each of the classes:

$$D_M^2(I_{l,j}^{tst}, A_c) = (\mathbf{h}_{l,j,c}^{tst} - \boldsymbol{\mu}_c)^T \mathbf{C}_c^{-1} (\mathbf{h}_{l,j,c}^{tst} - \boldsymbol{\mu}_c). \tag{8.23}$$

Due to the small number of training images, some regularisation was necessary: $\mathbf{C}_c = \mathbf{C}_c + 10^{-4}\,\mathbf{I}$. Each test image $I_{l,j}^{tst}$ was then assigned to that class c which gave the lowest Mahalanobis distance:

$$c = \arg\min_{c'} D_M^2(I_{l,j}^{tst}, A_{c'}). \tag{8.24}$$

The results of these experiments, performed for $k \in \{8, 12, 16, 20\}$ subspaces, subspace dimensionality $m \in \{2, 4\}$, and a sample window size of $w \in \{8, 16\}$, are shown in figure 8.11. The best result obtained is a test error of 11% (4 out of 36 images classified incorrectly), which is quite reasonable given the difficulty of the problem. The window

size does not seem to be too important in this application, as for both $w = 8$ and $w = 16$ the optimum is reached. There is an optimal number of subspaces, but again the exact choice is not too critical. Furthermore, $m = 2$ is sufficient; 4-dimensional subspaces lack the distinctiveness needed to describe class-specific image information well.

8.5.2 Discussion

Mappings of images onto ASMs were shown to yield useful features for object recognition. Although performance is not impressive, it is certainly reasonable. To gain insight, it is interesting to look at the misclassifications of the best performing ASMs. Mostly the same test images are misclassified for a variety of settings: three images in the mugs class and one workstation image. These are shown in figure 8.12. The three images of mugs are the only three images in which the ears of the mug are not visible; they appeared only in the testing set. These images are classified as chess pieces. The workstations image is the only one in which a row of books is also visible, and is labelled as a book case. In all these cases, the difference in Mahalanobis distance to the true class and to the class the image was labelled as, was small.

To investigate what features are found by the ASMs, and whether the ASMs do not merely describe the background, the subspaces can be ranked as follows. Only the classes of images containing chess pieces, mugs and teaflasks were considered, as these shared the same backgrounds. Each subspace $P_i^{A_c}$, $i = 1, \ldots, k$ representing class c was evaluated as a feature by calculating the average Mahalanobis distance of the training images of the other classes to that class c, using only assignment histogram bin h_i. This distance was then used to label the image: the brighter the colour of a region, the higher the Mahalanobis distance due to the feature describing that region, the more useful the feature. Figure 8.13 shows three examples of these rankings. The background is assigned to just a few subspaces, more or less randomly. For some regions, the fact that they are *not* textured is important (e.g., the region with feature rank 8.5 in the chess piece image). However, some informative and discriminating features corresponding to structure have been found for each of the classes as well: curved edges for the chess pieces, the curved ears of the mugs and curves and horizontal edges for the tea flask. Of course, it is the Mahalanobis distance that makes use of these features – the ASM is not trained to specifically find discriminating features, it just tries to describe the image as well as possible.

8.6 Handwritten digit recognition

Finally, the original ASM model and the subspace shift trained ASM^{ss} were applied to the problem of handwritten digit recognition. This has been a popular benchmark

application in various publications on MoSs [151, 356]. Most of the results reported were obtained on the CEDAR database [165]; however, as section 3.3.2 contains a large number of results obtained on the NIST database, the same training and testing sets (see section 3.3.1) were used here, to allow easy comparison. Note that only results obtained on the entire training set, containing 1,000 samples per class, are given. The testing set again contained 1,000 samples per class as well.

8.6.1 Experiments

Normalisation of the data is not applicable here, as the average grey-value and standard deviation may be important cues for judging (dis)similarity between digits. Therefore, only the ASM, PPCA and ASM^{ss} algorithms were considered. The use of episodes it not necessary either; the data was already pre-processed to remove the major degrees of freedom (slant and line width, cf. section 3.3.1). The entire training set was pre-mapped to retain $r = 90\%$ of the variance; this left 51 dimensions. For each of the 10 classes, an MoS was then trained on the 1,000 samples available of that class in the training set. The number of subspaces and number of dimensions per subspace were varied: $k \in \{2, 4, 8, 12, 16\}$, $m \in \{4, 8, 12, 16, 20\}$. Of course, the ASM^{ss} do not need k to be set, but the radius ρ; after some experimentation, it was set to 80. After training the ASM^{ss} models, subspaces representing less than 20 samples were removed to reduce over-fitting.

Table 8.3 shows the results, in % error on the testing set. For larger m and k (numbers of dimensions and subspaces), in fact for all $k \geq 12$, the PPCA methods could no longer be trained, as for some of the models σ went to zero and the models collapsed. Note that the algorithm was restarted four times before giving up. Apparently, hard-assignment clustering algorithms are more robust on this data set for these settings of m and k.

8.6.2 Discussion

Overall results are very good, especially when considering that the models are trained on individual classes without using the classification error as a criterion. The best ASM reaches a testing set error of 2.2%, which is better than the LeCun ANN (see figure 3.5 (a)). Clearly, the MoS models also fit the data better than single Gaussian densities (the "lc" and "qc" curves in figure 3.5 (b)). The ASM^{ss} algorithm reaches nearly the same performance as the ASM, but at a fraction of the computational cost; as was already discussed in section 8.2.3, it is much faster to train than the k-subspaces algorithm. Per digit, it finds the appropriate number of subspaces. This number seems to correspond roughly to the amount of variation possible in writing each digit; e.g., there are less ways of writing a "1" (modelled by 2-3 subspaces) than there are of writing a "2" (modelled by 6-10 subspaces). Only for $m = 2$ does the ASM^{ss} find too little subspaces,

	ASM					PPCA			ASM^{ss}											
	$k=2$	4	8	12	16	$k=2$	4	8	error	k, for "0" ... "9" and total										
$m=2$	4.1	3.5	3.0	3.0	*2.6*	6.8	5.6	4.5	4.1	6	3	8	8	7	12	7	5	8	6	70
4	3.1	2.5	*2.4*	2.7	2.6	5.5	4.2	4.1	2.5	5	2	8	7	6	9	4	4	9	3	57
8	2.6	2.5	*2.4*	3.0	3.2	5.3	3.2	-	2.4	3	3	10	8	5	8	4	3	8	4	56
12	2.4	*2.2*	2.6	3.4	3.6	4.7	-	-	2.2	5	3	8	6	6	7	4	3	7	3	52
16	*2.3*	2.3	3.0	3.7	4.3	-	-	-	2.3	3	2	6	6	4	6	3	3	6	3	42

Table 8.3: Handwritten digit recognition results, in % error on the testing set, for various numbers of subspaces k and subspace dimensionalities m. For ASM^{ss}, the number of subspaces found is also shown, per digit.

resulting in a larger error than that of the ASM. This is caused by the fixed setting of ρ for all experiments; it is probably too large for the $m = 2$ experiments, resulting in too few subspaces being found.

In general the ASM^{ss} algorithm finds somewhat more subspaces than ASMs giving similar performance. Still, both algorithms seem to perform well using, say, between 40 and 60 12D subspaces. In total, they use one 51×256 pre-mapping matrix, and 40 to 60 subspaces of $(12 + 1) \times 51$ elements, including the origin of each subspace. This amounts to between 39,576 and 52,836 parameters, which is even lower than the LeCun ANN in section 3.3.2.

An added advantage of the MoS models is the ease with which they can be interpreted. Whereas ANNs are to a large extent "black boxes", the subspace models can easily be visualised. An example is shown in figure 8.14. Of each of the 9 subspaces in the description of the digit "5" in the 4D ASM^{ss}, the origin $\boldsymbol{\mu}$ is plotted, together with the variation allowed by the first basis vector $\mathbf{e}_1$ of the subspace. For most subspaces, it is clear which variations they model: P_1 allows for slight slant differences, P_2, P_5, P_7 and P_9 for horizontal size differences, P_3, P_4 and P_6 for differences in writing style. P_8 even seems to account for mis-segmented digits in which the top stroke is missing. These plots make it possible to investigate the variation in the data set and, where necessary, remove erroneous training samples or even superfluous subspaces.

8.7 Conclusions

This chapter introduced three clustering algorithms useful for training MoS models: k-subspaces, EM and subspace shift. For the first two, next to the dimensionality of each subspace, the number of subspaces to be found has to be given beforehand. The subspace shift algorithm, an adaptation of the mean shift algorithm, requires a radius parameter ρ which might be easier to specify. The k-subspaces and subspace shift al-

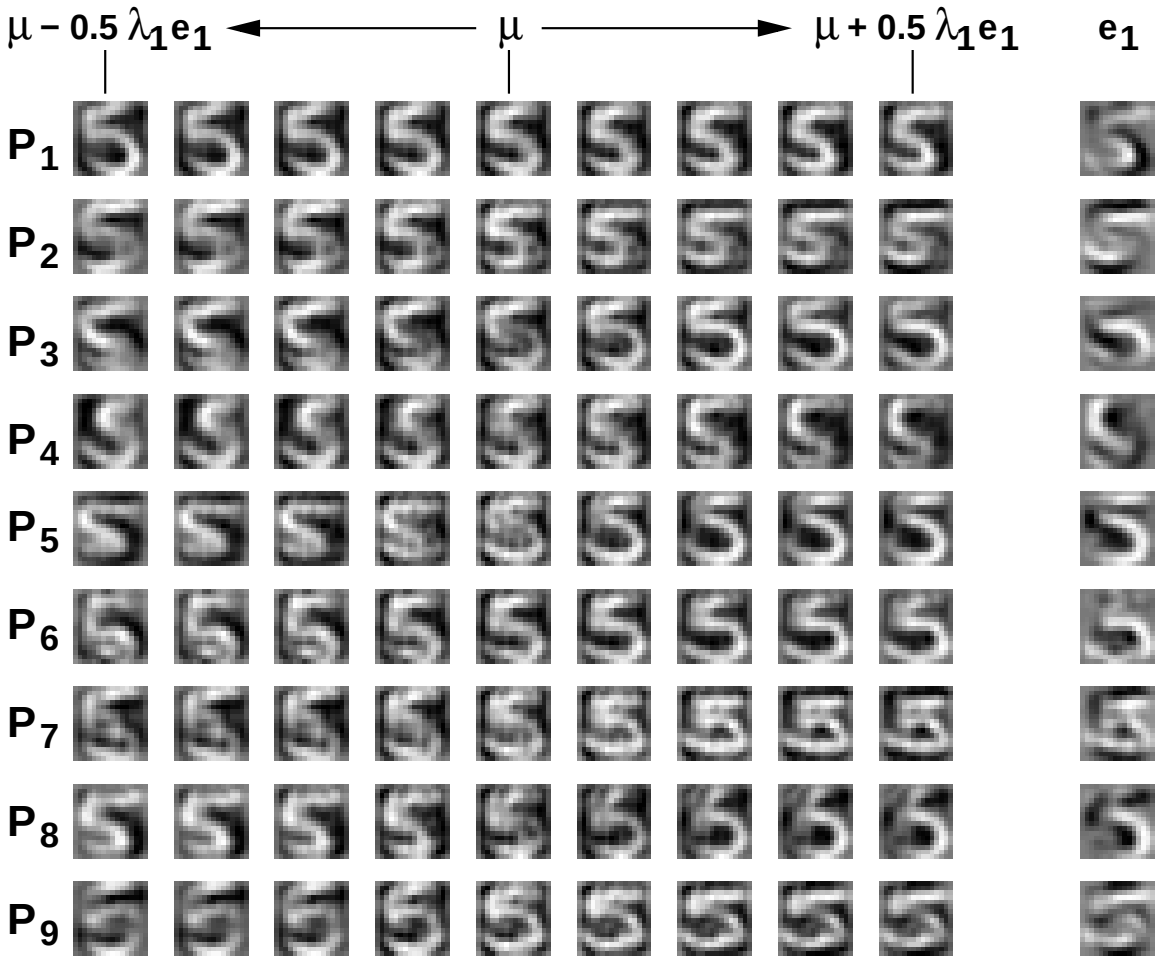


Figure 8.14: The 9-subspace, 4D ASM^{SS} trained on digit 5. For each subspace, the origin $\boldsymbol{\mu}$ and the variation due to the first basis vector $\mathbf{e}_1$ is shown. The range of variation is set using the corresponding eigenvalue λ_1.

gorithms have the advantage that the use of episodes can easily be incorporated into the algorithm; however, the EM algorithm might deal better with data overlap (i.e. sampled windows containing multiple textures).

The resulting models were then applied to the problem of texture segmentation; experiments were performed on a number of artificially combined structured textures. The segmented images and an overview of segmentation errors showed that:

- the Gaussian mixture model, 8D/16D k-subspaces-trained PCA mixture models (ASM) and 8D/16D EM-trained PCA mixture models (PPCA) perform well in segmenting textures;
- normalisation even improves this performance somewhat;

- for most models, segmentation errors occur mainly in the region around the border between two textures;
- sometimes the models fail to find the right segmentation completely, when one of the textures is hard to describe using a single subspace;
- it is difficult to assess the subspace shift algorithm, as it tends to oversegment images;
- errors remain low when models are trained on episodes constructed according to the all-transformation method, which in chapter 7 was demonstrated to increase invariance to rotation and scaling.

The ASM model can be used for more than just texture segmentation, as a trained ASM can be seen as a description or "signature" of an image, or even for a class of images. This idea was demonstrated on two applications: image database retrieval and object recognition. For image database retrieval, a distance measure was proposed between two sets of subspaces. Each image in an image database can then be represented by an ASM, and queries can be performed by sorting images based on distances between their representative ASMs. Experimental results, although the database used was small, were quite good compared to a state-of-the-art system, given the fact that only texture and edge information is used (since the average grey value is removed from the samples). Some experiments on one of the queries on which performance was poor, together with inspection of ASMs and query returns, lead to the conclusion that on this image database the ASMs code large-scale structure rather than fine texture. This is due to the fact that less texture is present, but also because texture usually has a much smaller scale than in the Brodatz texture images and is harder to model.

Besides using the ASMs themselves, histograms of image pixels assigned to the ASMs subspaces may be used as well. In an object recognition experiment, classes of objects were each represented by one ASM. Histograms of images pixel-wise assigned to the ASMs were then used as features in a simple classification scheme based on the Mahalanobis distance. This method gave a reasonably low test error. A feature ranking method was applied to learn what subspaces are important for classification; it was shown that the ASMs had picked up on structures (edges etc.) specific to each of the classes.

In a last application, the handwritten digit recognition problem of chapter 3 was revisited. On this problem, MoS models trained using the hard clustering algorithms (i.e. k-subspaces, subspace shift) were shown to be more robust than the EM-trained ones. Overall, performance was quite good, considering that the method is not trained with the goal of minimising classification error and that the number of parameters used is quite low compared to classifiers applied earlier. Furthermore, the subspace shift algorithm was shown to automatically find an optimal number of subspaces, with errors as low as the best-performing ASM.

In most of the experiments in this chapter some important settings, such as the number of subspaces to use within each ASM, the number of dimensions per subspace and the sampling window size, were optimised by hand. For easier applicability, it would be interesting to investigate optimising these automatically, or perhaps to use a multi-scale approach. Although some authors propose methods to find optimal parameter settings (e.g. by a Bayesian approach, [29, 30, 123]), these do not seem quite applicable to the applications discussed here due to their high computational complexity and the fact that results depend on the size of the training set. Another possible extension is the use of colour information. This would be quite desirable, as the literature on image databases clearly shows color information to be the most important to use in image database retrieval [9].

An important finding is that unsupervised methods not only perform well on the applications discussed in this chapter, but also allow for easier interpretation than supervised methods. In image segmentation and description, textured regions and structures in an image are characterised by (sets of) subspace basis vectors. These can easily be seen to respond to certain specific elements in images, e.g. edges of various slopes at various rotations. In the handwritten digit recognition application, subspace origins correspond to typical ways of writing digits, while basis vectors indicate which variations in writing occur (e.g. size, writing style etc.). In view of the problems encountered in interpreting supervised feed-forward neural networks (see chapters 4 and 6), this is quite an advantage.

9

CONCLUSIONS

This thesis discussed the application of adaptive methods in image processing. Three main questions were formulated in chapter 1:

- *Applicability*: can (nonlinear) image processing operations be learned by adaptive methods?
- *Prior knowledge*: how can prior knowledge be used in the construction and training of adaptive methods?
- *Interpretability*: what can be learned from adaptive methods trained to solve image processing problems?

These three questions return throughout this thesis, for three different approaches: supervised classification (chapters 3-4), supervised regression (chapters 5-6) and unsupervised clustering (chapters 7-8). Below, answers will be formulated to each of the questions for the different approaches.

9.1 Applicability

The review in chapter 2 showed that many researchers have attempted to apply artificial neural networks (ANNs) to image processing problems. To a large extent, it is an overview of what can now perhaps be called the "neural network hype" in image processing: the approximately 15-year period following the publications of Kohonen, Hopfield and Rumelhart et al. Their work inspired many researchers to apply ANNs to their own problem in any of the stages of the image processing chain. In some cases, the reason was biological plausibility; however, in most cases the goal was simply to obtain well-performing classification, regression or clustering methods.

In some of these applications the most interesting aspect of ANNs, the fact that they can be trained, was not (or only partly) used. This held especially for applications to the first few tasks in the image processing chain: pre-processing and feature extraction. Another

advantage of ANNs often used to justify their use is the ease of hardware implementation; however, in most publications this did not seem to be the reason for application. These observations, and the fact that often researchers did not compare their results to established techniques, casted some doubt on the actual advantage of using ANNs. In the remainder of the thesis, ANNs were therefore trained on three tasks in the image processing chain: object recognition (supervised classification), pre-processing (supervised regression) and feature extraction (unsupervised) and, where possible, compared to traditional approaches.

The experiments on supervised classification, in handwritten digit recognition and automatic target recognition, showed that ANNs are quite capable of solving difficult object recognition problems. Training was not without problems. This was shown by the LeNet ANN, for which detailed inspection revealed that parts were not functioning at all after training. Training ANNs to perform automatic target recognition was also difficult. Of a number of trained instances of an ANN architecture, only some performed well. Still, in both object recognition problems the ANNs performed well. On handwritten digit recognition, ANNs performed nearly as well as some traditional pattern recognition methods, such as the nearest neighbour rule and support vector classifiers, but at a fraction of the computational cost. This corroborates the finding in the review that object recognition was one of the most popular application areas of ANNs.

As supervised regressors, a number of ANN architectures was trained to mimic the Kuwahara filter, a nonlinear edge-preserving smoothing filter used in pre-processing. The experiments showed that careful construction of the training set is very important. If filter behaviour is critical only in parts of the image represented by a small subset of the training set, this behaviour will not be learned. Constructing training sets using the knowledge that the Kuwahara filter is at its most nonlinear around edges improved performance considerably. This problem is also due to the use of the mean squared error (MSE) as a training criterion, which will allow poor performance if it only occurs for a small number of samples. Another problem connected with the use of the MSE is that it is insufficiently discriminative for model choice; in first attempts, almost all ANN architectures showed identical MSEs on test images. Criteria which were proposed to measure smoothing and sharpening performance showed larger differences. An attempt at using another criterion for training (the L_p norm) was not successful, as for most p smoothing and sharpening performance was worse than for $p = 2$, i.e. the MSE. It did however illustrate the large influence of the choice of error criterion. Unfortunately, the results indicate that the training set and performance measure will have to be tailored for each specific application, with which ANNs lose much of their attractiveness as all-round methods. The findings also explain why in many of the reviewed papers in chapter 2, ANNs applied to pre-processing were non-adaptive.

Finally, mixture-of-subspace (MoS) models were used as unsupervised clustering methods for feature extraction. These models were inspired by an ANN, the adaptive subspace self-organising map (ASSOM). One can argue about whether MoS models are

neural, as they do not fit the description given in section 1.1.2 and they are not as generally applicable as the feed-forward ANNs used before. However, the only difference with the self-organising map is the lack of topological correctness, which in most applications is never used. Various subspace models were discussed. Principal component analysis (PCA) was first examined as a texture description method and proved to perform well compared to Gaussian models and Gabor filter banks. In contrast, independent component analysers (ICA, for which a new training algorithm was derived), were shown not to be useful for texture description, as they focused on rarely occurring, high-frequency elements in images. Next, mixtures of PCA subspaces were considered. As these MoS models themselves do not yet perform a task, but simply describe the data, various post-processing steps can be added. Simply using the distance to the MoS gave good results in segmentation (of Brodatz textures) and object recognition (of handwritten digits). In the recognition of handwritten digits, MoS models gave results as good as the shared weight ANNs used earlier, using less parameters. Histograms of pixel assignments were useful for object recognition; and using inter-model distance measures lead to a well-performing image database retrieval method. An open problem with MoS models is the setting of a number of model parameters, which should ideally be optimised automatically.

In conclusion, supervised methods seem to be most applicable for problems requiring a nonlinear solution, for which there is a clear, unequivocal performance criterion. This means ANNs are more suitable for high-level tasks in the image processing chain, such as object recognition, rather than low-level tasks. For both classification and regression, the choice of architecture, the performance criterion and data set construction play a large role and will have to be optimised for each application. Unsupervised methods are suitable as feature extraction mechanisms and can easily be coupled to post-processing steps to perform segmentation and object recognition. Here, parameter settings rather than architectural choices are important.

9.2 Prior knowledge

In a large number of the publications reviewed in chapter 2, prior knowledge was used to constrain ANNs. This is to be expected; unconstrained ANNs contain large numbers of parameters and run a high risk of being overtrained. Prior knowledge can be used to lower the number of parameters in a way which does not restrict the ANN to such an extent that it can no longer perform the desired function. One way to do this is to construct modular architectures, in which use is made of the knowledge that an operation is best performed as a number of individual sub-operations. Another way is to use the knowledge that neighbouring pixels are related and should be treated in the same way, e.g. by using receptive fields in shared weights ANN.

The latter idea was tested in supervised classification, i.e. object recognition. The shared

weight ANNs used contain several layers of feature maps (detecting features in a shift-invariant way) and subsampling maps (combining information gathered in previous layers). The question is to what extent this prior knowledge was truly useful. Visual inspection of trained ANNs revealed little. Standard feed-forward ANNs comparable in the number of *connections* (and therefore the amount of computation involved), but with a much larger number of *weights*, performed as well as the shared weight ANNs. This proves that the prior knowledge was indeed useful in lowering the number of parameters without affecting performance. However, it also indicates that training a standard ANN with more weights than required does not necessarily lead to overtraining.

For supervised regression, a number of modular ANNs was constructed. Each module was trained on a specific subtask in the nonlinear filtering problem the ANN was applied to. Furthermore, of each module different versions were created, ranging from architectures specifically designed to solve the problem (using hand-set weights and tailored transfer functions) to standard feed-forward ANNs. According to the proposed smoothing and sharpening performance measures, the fully hand-constructed ANN performed best. However, when the hand-constructed ANNs were (gradually) replaced by more standard ANNs, performance quickly decreased and became level with that of some of the standard feed-forward ANNs. Furthermore, in the modular ANNs that performed well the modular initialisation was no longer visible (see also the next section). The only remaining advantage of a modular approach is that careful optimisation of the number of hidden layers and units, as for the standard ANNs, is not necessary.

In the MoS models finally, prior knowledge is rather easy to incorporate. Normalisation of the data uses the knowledge that image data should be modelled invariant to illumination. The very use of subspace mixture models itself is a powerful use of prior knowledge, i.e. that the data can be locally modelled by linear subspaces, invariant to certain transformations. Although there is a large difference in the number of parameters, the MoS models perform nearly as well as mixtures-of-Gaussians. The model parameters (the number of subspaces, the number of dimensions per subspace) too have a certain physical meaning which gives insight into their optimal setting. The only application in which such prior knowledge was not available due to the large range of image content, the image database retrieval problem, gave most problems in choosing optimal parameter settings. Finally, the fact that MoS models describe data rather than a function on it can be used to find simple yet powerful post-processing steps using prior knowledge. Examples were the use of the histogram of pixels of an image assigned to an MoS as a "signature" of that image, and the measurement of distances between images by inter-MoS distances.

These observations lead to the conclusion that prior knowledge can be used to restrict adaptive methods in a useful way. However, various experiments showed that feed-forward ANNs are not natural vehicles for doing so, as this prior knowledge will have to be translated into a choice for ANN size, connectivity, transfer functions etc., para-

meters which do not have any physical meaning related to the problem. Therefore, such a translation does not necessarily result in an optimal ANN. It is easier, as the MoS applications showed, to construct a (rough) model of the data and allow model variation by allowing freedom in a number of well-defined parameters. Prior knowledge should be used in constructing models rather than in molding general approaches.

9.3 Interpretability

Throughout this thesis, strong emphasis was placed on the question whether ANN operation could be inspected after training. Rather than just applying ANNs, the goal was to learn from the way in which they solved a problem. In almost none of the publications discussed in the review (chapter 2) this played a large role, although it would seem to be an important issue when ANNs are applied in mission-critical systems, e.g. in medicine, process control or defensive systems.

Supervised classification ANNs were inspected w.r.t. their feature extraction capabilities. As feature extractors, shared weight ANNs were shown to perform well, since standard pattern recognition algorithms trained on extracted features performed better than on the original images. Unfortunately, visual inspection of trained shared weight ANNs revealed nothing. The danger here is of over-interpretation, i.e. reading image processing operations into the ANN which are not really there. To be able to find out what features are extracted, two smaller problems were investigated: edge recognition and two-class handwritten digit recognition. A range of ANNs was built, which showed that ANNs need not comply with our ideas of how such applications should be solved. The ANNs took many "short cuts", using biases and hidden layer-output layer weights. Only after severely restricting the ANN did it make sense in terms of image processing primitives. Furthermore, in experiments on an ANN with two feature maps the ANN was shown to distribute its functionality over these maps in an unclear way. An interpretation tool, the decorrelating conjugate gradient algorithm (DCGD), can help in distributing functionality more clearly over different ANN parts. The findings lead to the formulation of the interpretability trade-off, between realistic yet hard-to-interpret experiments on the one hand and easily interpreted yet non-representative experiments on the other.

This interpretability trade-off returned in the supervised regression problem. Modular ANNs constructed using prior knowledge of the filtering algorithm performed well, but could not be interpreted anymore in terms of the individual sub-operations. In fact, retention of the modular initialisation was negatively correlated to performance. ANN error evaluation was shown to be a useful tool in gaining understanding of where the ANN fails; it showed that filter operation was most poor around edges. The DCGD algorithm was then used to find out why: most of the standard feed-forward ANNs found a sub-optimal linear approximation to the Kuwahara filter. The conclusion of the

experiments on supervised classification and regression is that as long as a distributed system such as an ANN is trained on single goal, i.e. minimisation of prediction error, the operation of sub-systems cannot be expected to make sense in terms of traditional image processing operations. This held for both the receptive fields in the shared weight ANNs and the modular setup of the regression ANNs: although they are there, they are not necessarily used as such. This also supports the conclusion of the previous section, that the use of prior knowledge in ANNs is not straightforward.

Finally, the MoS models offer very natural ways of interpretation. First, the models themselves allow immediate inspection of subspace basis vectors as possible variations on prototypical image patches; the model parameters are expressed in terms of image data itself. It is rather easy to point out specific invariances, as was shown for the handwritten digit recognition problem. This even does not lead to a degradation in performance; there is no interpretability trade-off. Second, the parameters (number of subspaces, number of dimensions) have a clear interpretation. Third, as most applications of the MoS models just used the notion of distance to the model, inspection of feature importance is simple. This was demonstrated for segmentation, i.e. the problem with the cathedral image in the image database retrieval problem; but also for the object recognition problem, in which features were ranked by their discriminative powers.

This thesis showed that interpretation of supervised ANNs is hazardous. As large distributed systems, they can solve problems in a number of ways, not all of which necessarily correspond to human approaches to these problems. Simply opening the black box at some location one expects the ANN to exhibit certain behaviour does not give insight into the overall operation. Furthermore, knowledge obtained from the ANNs cannot be used in any other systems, as it only makes sense in the precise setting of the ANN itself. In contrast, unsupervised methods allow easy interpretation, as they do not depend on any derived performance criteria.

9.4 Conclusions

We believe that the last few years have seen an attitude change towards ANNs, in which ANNs are not anymore automatically seen as the best solution to any problem. The field of ANNs has to a large extent been re-incorporated in the various disciplines that inspired it: machine learning, psychology and neurophysiology. In machine learning, researchers are now turning towards other, non-neural adaptive methods, such as the support vector classifier. For them the ANN has become a tool, rather than *the* tool it was originally thought to be.

So when are ANNs useful in image processing? First, they are interesting tools when there is a real need for a fast parallel solution. Second, biological plausibility may be a factor for some researchers. But most importantly, ANNs trained based on examples

can be valuable when a problem is too complex to construct an overall model based on knowledge only. Often, real applications consist of several individual modules performing tasks in various steps of the image processing chain. A neural approach can combine these modules, control each of them and provide feedback from the highest level to change operations at the lowest level. The price one pays for this power is the black-box character, which makes interpretation difficult, and the problematic use of prior knowledge. If prior knowledge is available, it is better to use this to construct a model-based adaptive method and learn its parameters; performance can be as good, and interpretation comes natural.

APPENDICES

A

SHARED WEIGHT NETWORK ARCHITECTURES

A.1 LeCun

This was the original proposal, as found in [207, 209, 233]. It was described in section 3.2.1. The only difference is that a border of 1 pixel has been added on the top and the left and a border of 2 pixels on the bottom and on the right, to take care of the problems encountered in places where the filter extended beyond the borders of the 16×16 image. Therefore, instead of 1256 cells, this network has $1256 - 16^2 + 19^2 = 1361$ cells, as indicated in table A.1. Table A.2 shows how subsampling maps are connected to feature maps. In figure A.1 the LeCun ANN architecture is shown; figure A.2 depicts a trained LeCun ANN.

A.2 LeNet

This network is described in section 3.2.2 and was found in [208]. Note that although this network has much more connections than the previous one, the number of weights is far less (table A.1). Table A.3 details how feature maps are connected to subsampling maps. Figure A.3 shows the LeNet architecture and figure A.4 a trained LeNet ANN.

A.3 LeNotre

The LeNotre-architecture is a proposal by Fogelman Soullie et al. in [109] and, under the name Quick, by Viennet [364]. It was used to show that the ideas that resulted in the construction of the networks described above can be used to make very small networks

No. of...	LeCun	LeNet	LeNotre
Units	1361	4634	394
Connections	63660	94952	2210
Weights	8760	2584	626
Biases	1000	3850	138

Table A.1: Number of elements in the LeCun, LeNet and LeNotre architectures.

		Subsampling map											
		1	2	3	4	5	6	7	8	9	10	11	12
Feature map	1	•	•	•							•	•	•
	2	•	•	•							•	•	•
	3	•	•	•	•	•	•						
	4	•	•	•	•	•	•						
	5				•	•	•	•	•	•			
	6				•	•	•	•	•	•			
	7							•	•	•	•	•	•
	8							•	•	•	•	•	•
	9	•	•	•	•	•	•	•	•	•	•	•	•
	10	•	•	•	•	•	•	•	•	•	•	•	•
	11	•	•	•	•	•	•	•	•	•	•	•	•
	12	•	•	•	•	•	•	•	•	•	•	•	•

Table A.2: Connections between the feature map layer and subsampling map layer in the LeCun architecture.

		Subs. map						Subs. map			
		1	2	3	4			1	2	3	4
Feature map	1	•				Feature map	7			•	
	2	•	•				8			•	•
	3	•	•				9			•	•
	4		•				10				•
	5	•	•				11			•	•
	6	•	•				12			•	•

Table A.3: Connections between the feature map layer and subsampling map layer in the LeNet architecture.

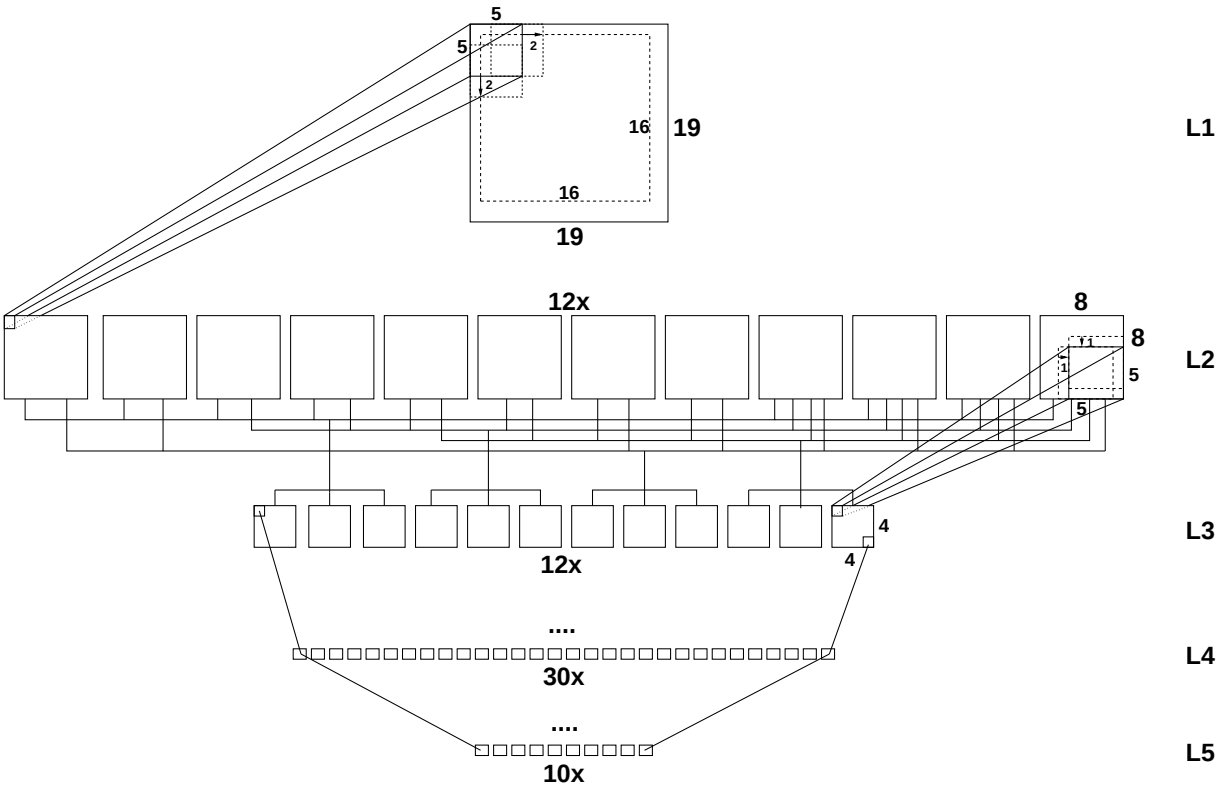


Figure A.1: The LeCun architecture. The top layer is the input layer.

that still perform reasonably well. Notice that the second layer contains two differently sized feature maps. Also, this network does not enlarge the input image to compensate for border effects. Table A.1 shows the number of components of the ANN. Figure A.5 shows the layout of the ANN, figure A.6 a trained instance of the LeNotre ANN.

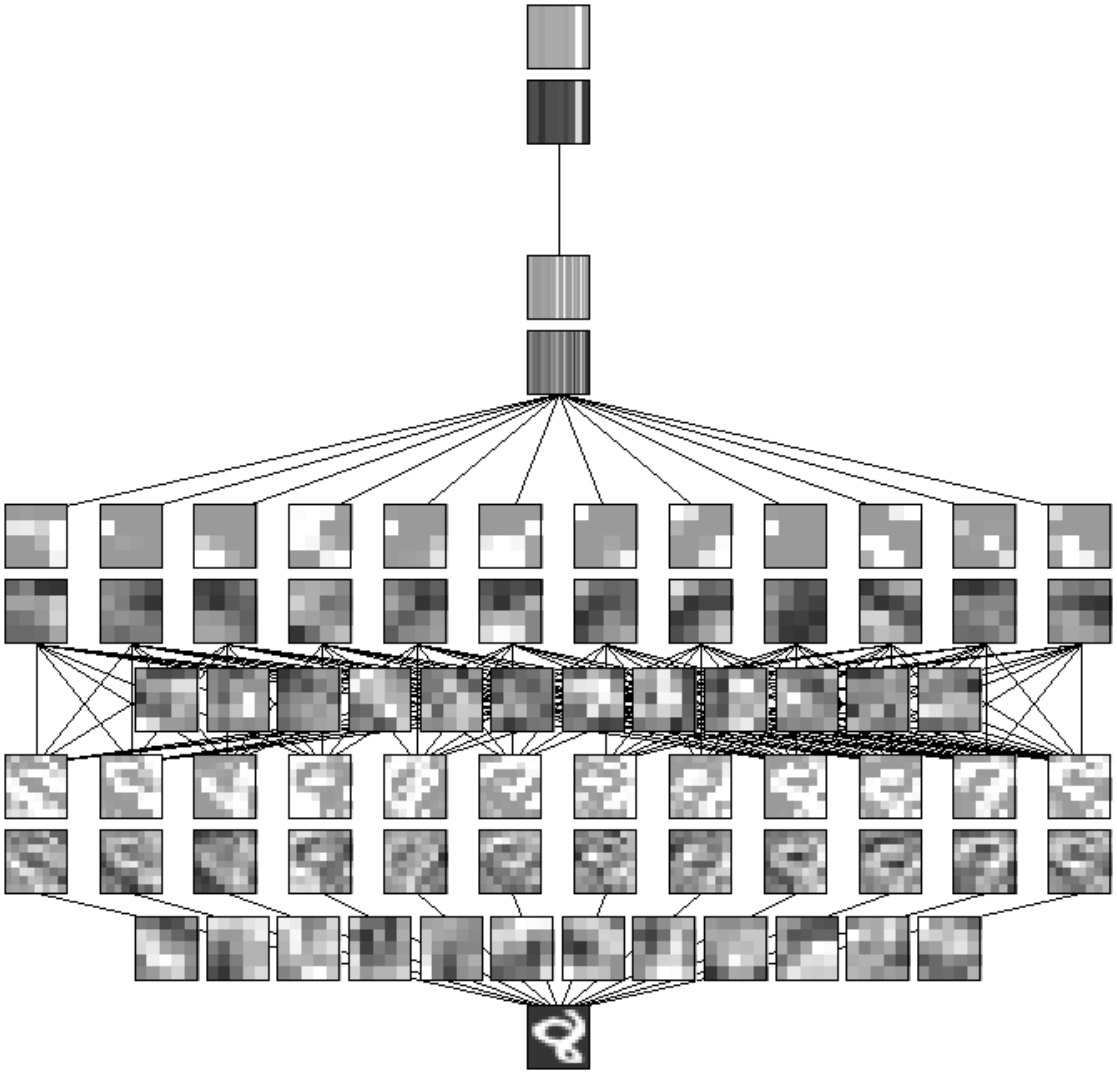


Figure A.2: The LeCun ANN trained on the handwritten digit set, 1,000 samples/class. Note: for each map in the third layer, only the first set of weights (the first filter) is depicted. Bias is not shown in the figure. In this representation, the bottom layer is the input layer.

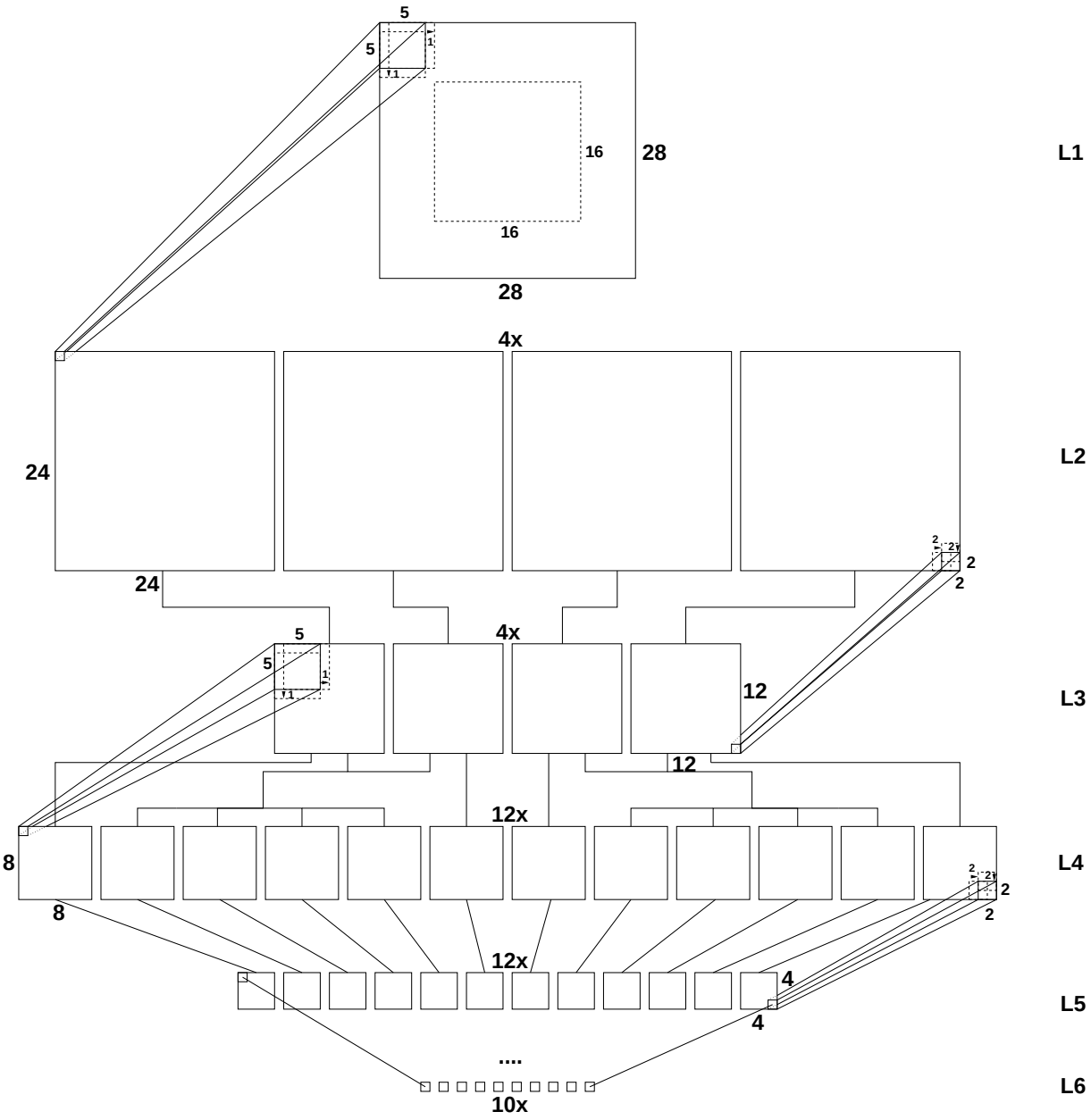


Figure A.3: The LeNet architecture. The top layer is the input layer.

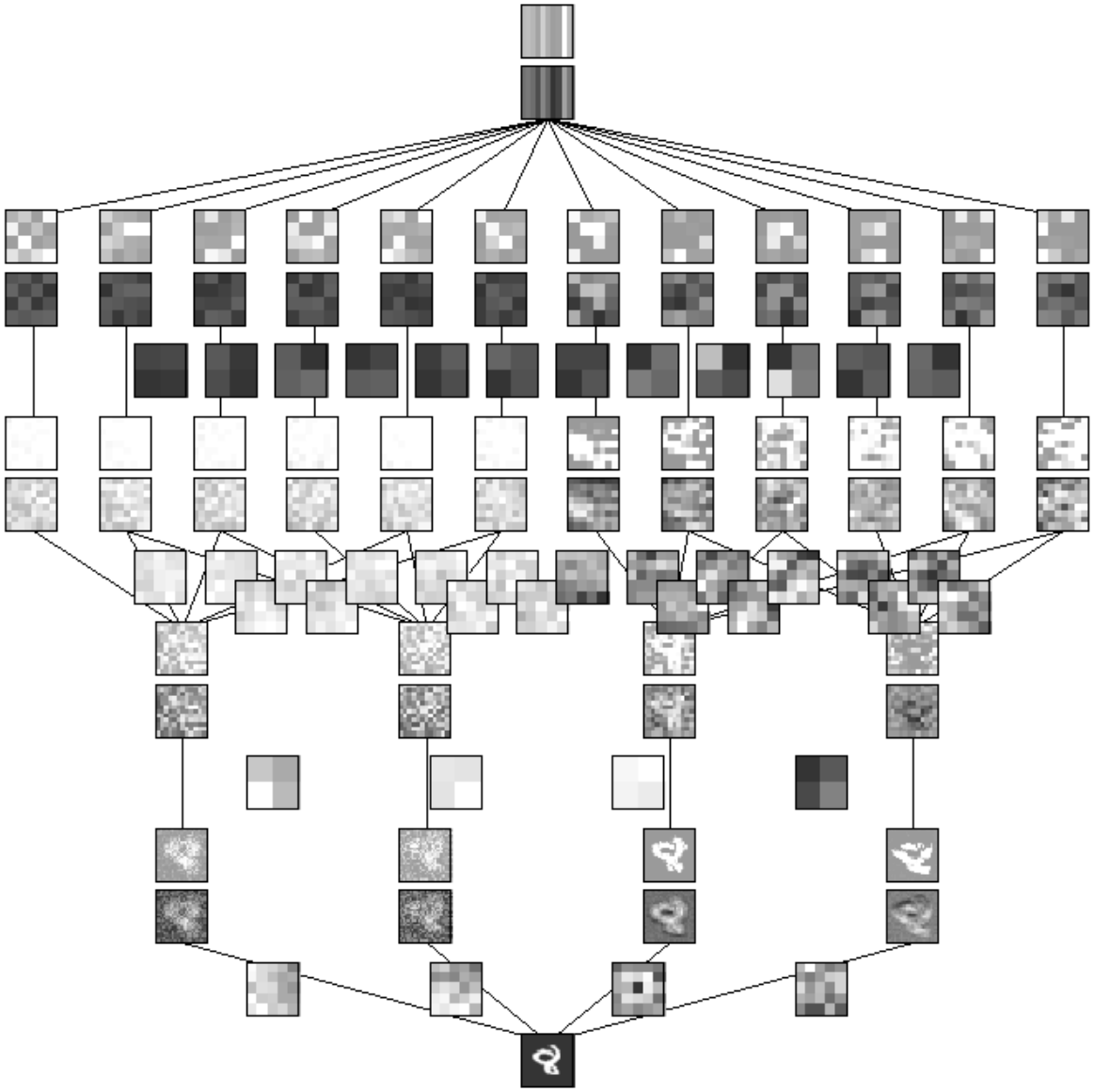


Figure A.4: The LeNet ANN trained on the handwritten digit set, 1,000 samples/class. Bias is not shown in the figure. In this representation, the bottom layer is the input layer.

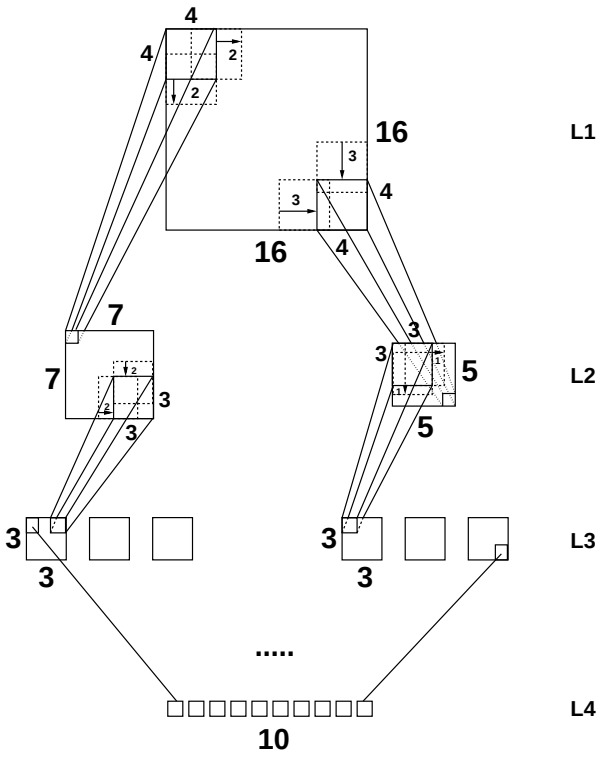


Figure A.5: The LeNotre architecture. The top layer is the input layer.

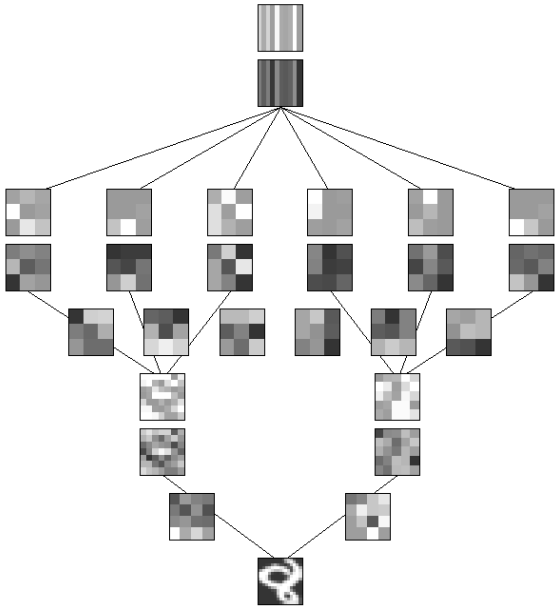


Figure A.6: The LeNotre ANN trained on the handwritten digit set, 1,000 samples/class. Bias is not shown in the figure. In this representation, the bottom layer is the input layer.

B

ARTIFICIAL NEURAL NETWORK ERROR EVALUATION

In this appendix, an expression will be derived for calculation of the variance of a single neural network output given a training set and one input vector. It is an abridged version of [28], pp. 385-401. The basic idea is to view the output of a trained ANN as generated by a probability density function. When certain assumptions are made regarding the distribution of the weights of the ANN, this variance can be calculated and can serve as an indication of ANN confidence for a certain input.

Let the ANN function be denoted by $R(\mathbf{x};\ \mathbf{w})$ (cf. section 2.2.1), where $\mathbf{x}$ is an input vector (with corresponding scalar target z) and $\mathbf{w}$ is now considered to be a vector of length W containing all weights and biases (note that this is slightly different from the notation used in chapter 2). The ANN has been trained on a set $\mathcal{L}$ with samples $(\mathbf{x}^i, y^i)$, $i = 1, \ldots, |\mathcal{L}|$.

The distribution of a single ANN output z given an input vector $\mathbf{x}$ can be written as

$$p(z|\mathbf{x}, \mathcal{L}) = \int p(z|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{L}) d\mathbf{w}, \tag{B.1}$$

where $\int d\mathbf{w}$ is shorthand for $\int_{\mathbb{R}} \cdots \int_{\mathbb{R}} dw_1 \cdots dw_W$. In the integral, the first term corresponds to the noise on the ANN targets, which is modelled by a Gaussian with a known and fixed variance σ_N^2:

$$p(z|\mathbf{x}, \mathbf{w}) \sim \exp\left(\frac{1}{2\sigma_N^2}(R(\mathbf{x};\ \mathbf{w}) - z)^2\right). \tag{B.2}$$

The second term is the posterior distribution of the weights, i.e. the distribution of the weight values after the ANN has been trained. It can be found using Bayes' theorem, which states that

$$p(\mathbf{w}|\mathcal{L}) = \frac{p(\mathcal{L}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{L})} = \frac{p(\mathcal{L}|\mathbf{w})p(\mathbf{w})}{\int p(\mathcal{L}|\mathbf{w})p(\mathbf{w})d\mathbf{w}}. \tag{B.3}$$

This means a prior distribution $p(\mathbf{w})$ will have to be found for the weights. Usually, a Gaussian distribution is assumed, as it corresponds to using a weight regularisation term $\frac{r}{2}||\mathbf{w}||^2$ during training (where r is the regularisation parameter). However, in the experiments in chapter 5 no regularisation has been used. To simplify calculations, a *non-informative prior* is chosen, in which $p(\mathbf{w})$ is considered to be uniformly distributed over the entire space[1]. This simplifies eqn. B.3 to:

$$p(\mathbf{w}|\mathcal{L}) = c\frac{p(\mathcal{L}|\mathbf{w})}{\int p(\mathcal{L}|\mathbf{w})d\mathbf{w}}, \tag{B.4}$$

with c a suitable normalisation constant.

Now, as the input vectors $\mathbf{x}$ are drawn independently, the likelihood $p(\mathcal{L}|\mathbf{w})$ in eqn. B.4 can be written as

$$p(\mathcal{L}|\mathbf{w}) = \prod_{(\mathbf{x}^i,y^i\in\mathcal{L})} p(y^i|\mathbf{x}^i,\mathbf{w}) \tag{B.5}$$

$$= \frac{1}{(2\pi\sigma_N^2)^{\frac{|\mathcal{L}|}{2}}} \exp\left(-\frac{1}{2\sigma_N^2}\sum_{(\mathbf{x}^i,y^i)\in\mathcal{L}}(R(\mathbf{x}^i;\ \mathbf{w})-y^i)^2\right) \tag{B.6}$$

$$= c'\exp(-S(\mathbf{w})), \tag{B.7}$$

in which the mean squared error criterion of eqn. 2.12 can be recognised:

$$S(\mathbf{w}) = \frac{|\mathcal{L}|}{\sigma_N^2}E(\mathbf{w}). \tag{B.8}$$

This gives as posterior distribution of the weights

$$p(\mathbf{w}|\mathcal{L}) = c''\frac{\exp(-S(\mathbf{w}))}{\int\exp(-S(\mathbf{w}))} \tag{B.9}$$

After training, this distribution can be approximated by a Gaussian around the minimum of the MSE, or equivalently, the maximum of the likelihood (eqn. B.9). If the weights at this minimum are denoted by $\mathbf{w}_{MP}$, the second-order approximation of $S(\mathbf{w})$ becomes:

$$S(\mathbf{w}) \approx S(\mathbf{w}_{MP}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^T\mathbf{H}(\mathbf{w}-\mathbf{w}_{MP}) \tag{B.10}$$

in which $\mathbf{H}$ is the Hessian of the ANN output w.r.t. its weights, i.e.

$$H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}. \tag{B.11}$$

[1]Strictly speaking this is not allowed, as this is an *improper prior*: the integrals can no longer be evaluated. However, it shortens the following derivation considerably and corresponds exactly to the end result of the derivation in [28] with the regularisation constant r set to 0.

The overall expression for the distribution of the ANN outputs now becomes, putting together eqns. B.2 and B.9, using eqn. B.10 and dropping terms not involving z or $\mathbf{w}$:

$$p(z|\mathbf{x},\mathcal{L}) \sim \int \exp\left(-\frac{1}{2\sigma_N^2}(R(\mathbf{x};\ \mathbf{w})-z)^2\right)\exp\left(-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^T\mathbf{H}(\mathbf{w}-\mathbf{w}_{MP})\right)d\mathbf{w}. \tag{B.12}$$

If the width of the posterior distribution of the weights is small enough, the ANN function can be approximated by a linear expansion around $\mathbf{w}_{MP}$:

$$R(\mathbf{x};\ \mathbf{w}) \approx R(\mathbf{x};\ \mathbf{w}_{MP}) + \nabla_{\mathbf{w}} R(\mathbf{x};\ \mathbf{w})|_{\mathbf{w}_{MP}}, \tag{B.13}$$

where the last term is the derivative of the ANN w.r.t. its weights, evaluated at $\mathbf{w}_{MP}$. Then eqn. B.12 can be written as ([28], p. 399):

$$p(z|\mathbf{x},\mathcal{L}) \sim \frac{1}{(2\pi\sigma_{tot}^2)^{\frac{1}{2}}}\exp\left(-\frac{1}{2\sigma_{tot}^2}(z-R(\mathbf{x};\ \mathbf{w}_{MP}))^2\right), \tag{B.14}$$

i.e. a Gaussian distribution with variance

$$\sigma_{tot}^2 = \sigma_N^2 + \left(\nabla_{\mathbf{w}} R(\mathbf{x};\ \mathbf{w})|_{\mathbf{w}_{MP}}\right)^T \mathbf{H}^{-1}\left(\nabla_{\mathbf{w}} R(\mathbf{x};\ \mathbf{w})|_{\mathbf{w}_{MP}}\right). \tag{B.15}$$

If the contribution due to the variance of the ANN outputs is negligible, i.e. when σ_N^2 is small, the second term dominates the variance σ_{tot}^2. The values of σ_{tot}^2 can then be calculated using just the derivative and Hessian of the ANN w.r.t. its weights. This Hessian can be found using any of a number of (rather involved) techniques discussed in [27, 28]. In chapter 5, simple finite differencing approximation has been used.

C

A MAXIMUM LIKELIHOOD ALGORITHM FOR UNDERCOMPLETE ICA BASES

In this appendix, the extended infomax ICA algorithm proposed by Lee et al. is discussed, in section C.1. It will go into more technical detail than Lee's publication [212] and the various assumptions made by the authors will be discussed. A fundamental problem of this algorithm is that it can only be used to find as many independent components (m) as there are dimensions in the data set (d). A new formulation is given in section C.2 to find undercomplete ICA bases, or ICA subspaces, the case where $m < d$. In section C.3 some properties of the algorithm are discussed, i.e. sensitivity to variance, implementation and sample size requirements. Finally, as a benchmark, in section C.4 the algorithm is applied to some simple 1D and 2D data sets and a data set of natural image patches.

C.1 Extended infomax ICA

The starting point is the basic ICA model, a latent variable model without noise [212, 227]:

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \boldsymbol{\mu}, \tag{C.1}$$

where $\mathbf{x}$ is a d-dimensional vector, $\mathbf{s}$ is the d-dimensional vector containing the independent sources and $\mathbf{A}$ is a full rank $d \times d$ mixing matrix. The goal of ICA is to find both matrix $\mathbf{A}$ and the sources $\mathbf{s}$ from a data set $\mathcal{L} = \{\mathbf{x}^n\}, n = 1, \ldots, N$.

First, maximum likelihood ICA will be discussed in section C.1.1. Next, the extensions to the basic ML approach introduced by Lee et al. [212] will be given in subsection C.1.2. In the derivations, without loss of generality the mean vector $\boldsymbol{\mu}$ is assumed to be zero, i.e. $\mathbf{x} = \mathbf{A}\mathbf{s}$.

C.1.1 Maximum likelihood ICA

Likelihood

Given the model in eqn. C.1, the probability of observing a data vector $\mathbf{x}$ given the latent variable $\mathbf{s}$ is not a true distribution, since there is no noise. The distribution can therefore be modelled by a Kronecker delta function [227]:

$$p(\mathbf{x}|\mathbf{s},\mathbf{A}) = \delta(\mathbf{x}-\mathbf{A}\mathbf{s}) = \prod_{l=1}^{d}\delta\left(x_l - \sum_{k=1}^{d} A_{lk}s_k\right), \tag{C.2}$$

where the factorisation is possible due to the independence assumption. The likelihood of observing a data vector $\mathbf{x}$ is

$$p(\mathbf{x}|\mathbf{A}) = \int p(\mathbf{x}|\mathbf{s},\mathbf{A})p(\mathbf{s})d\mathbf{s} = \int \delta(\mathbf{x}-\mathbf{A}\mathbf{s})p(\mathbf{s})d\mathbf{s}. \tag{C.3}$$

The following identity can be used to work out the integral, using $y = as$:

$$\int \delta(x-as)f(s)ds = \int \delta(x-y)f\left(\frac{y}{a}\right)\frac{1}{a}dy = \frac{1}{a}f\left(\frac{x}{a}\right) \tag{C.4}$$

which, for vectors, translates to:

$$\int \delta(\mathbf{x}-\mathbf{A}\mathbf{s})f(\mathbf{s})d\mathbf{s} = \int \delta(\mathbf{x}-\mathbf{y})f(\mathbf{A}^{-1}\mathbf{y})\frac{1}{|\det(\mathbf{A})|}d\mathbf{y} = |\det(\mathbf{A})|^{-1}f(\mathbf{A}^{-1}\mathbf{x}). \tag{C.5}$$

Here, the factor $|\det(\mathbf{A})|^{-1}$ should be seen as a normalisation factor. The linear transformation $\mathbf{A}$ causes a unit hypercube with volume 1 to have a volume $|\det(\mathbf{A})|$ after the transform – since $|\det(\mathbf{A})|$ is the volume of the parallepepid spanned by the columns of $\mathbf{A}$, i.e. the basis vectors of the transformation.

This allows equation C.3 to be solved:

$$p(\mathbf{x}|\mathbf{A}) = \int \delta(\mathbf{x}-\mathbf{A}\mathbf{s})p(\mathbf{s})d\mathbf{s} = |\det(\mathbf{A})|^{-1}p(\mathbf{A}^{-1}\mathbf{x}), \tag{C.6}$$

so the log-likelihood becomes, again using the fact that the elements of $\mathbf{s}$ are independent,

$$\begin{aligned}
\ln p(\mathbf{x}|\mathbf{A}) &= \ln\left[|\det(\mathbf{A})|^{-1}p(\mathbf{A}^{-1}\mathbf{x})\right] \\
&= -\ln|\det(\mathbf{A})| + \ln\left[\prod_{l=1}^{d} p\left(\sum_{k=1}^{d} A_{lk}^{-1}x_k\right)\right] \\
&= -\ln|\det(\mathbf{A})| + \sum_{l=1}^{d}\ln p\left(\sum_{k=1}^{d} A_{lk}^{-1}x_k\right).
\end{aligned} \tag{C.7}$$

Here the summation over $A^{-1}_{lk} x_k$ simply is the multiplication of row l of $\mathbf{A}^{-1}$ with $\mathbf{x}$.

Letting $\mathbf{W} = \mathbf{A}^{-1}$ and making use of $\det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})}$, this can also be written as

$$\ln p(\mathbf{x}|\mathbf{A}) = \ln|\det(\mathbf{W})| + \sum_{l=1}^{d} \ln p \left(\sum_{k=1}^{d} W_{lk} x_k \right), \tag{C.8}$$

where $\mathbf{W}$ is called the unmixing matrix. The log-likelihood for the entire data set is:

$$\mathfrak{L} = \sum_{n=1}^{N} \ln p(\mathbf{x}^n|\mathbf{A}) = \sum_{n=1}^{N} \left[\ln|\det(\mathbf{W})| + \sum_{l=1}^{d} \ln p \left(\sum_{k=1}^{d} W_{lk} x_k^n \right) \right]. \tag{C.9}$$

A generalised EM algorithm

The log-likelihood found in eqn. C.7 can now be maximised to find a maximum likelihood (ML) solution. This is achieved using a *generalised expectation-maximisation algorithm* (GEM): in each step the expectation of the log-likelihood is calculated (E-step) and gradient ascent is performed in a direction which maximises this likelihood (M-step). For more information on the GEM algorithm, see [84].

Rules for taking derivatives of matrix functions w.r.t. matrix elements can be found in [85]. Using the following basic equations:

$$\frac{\partial}{\partial M_{ij}} f(\mathbf{M}) = \mathbf{M}_{(i,j)} \tag{C.10}$$

$$\frac{\partial}{\partial M_{ij}} f(\mathbf{M}^T) = \mathbf{M}_{(j,i)}, \tag{C.11}$$

where $\mathbf{M}_{(i,j)}$ is a matrix filled with zeros and just $M_{ij} = 1$, the following rules are given or derived:

$$\frac{\partial}{\partial M_{ij}} [f(\mathbf{M}) + g(\mathbf{M})] = \frac{\partial}{\partial M_{ij}} f(\mathbf{M}) + \frac{\partial}{\partial M_{ij}} g(\mathbf{M}) \tag{C.12}$$

$$\frac{\partial}{\partial M_{ij}} [f(\mathbf{M}) g(\mathbf{M})] = \left[\frac{\partial}{\partial M_{ij}} f(\mathbf{M}) \right] g(\mathbf{M}) + f(\mathbf{M}) \left[\frac{\partial}{\partial M_{ij}} g(\mathbf{M}) \right] \tag{C.13}$$

$$\frac{\partial}{\partial M_{ij}} f^{-1}(\mathbf{M}) = -f^{-1}(\mathbf{M}) \left[\frac{\partial}{\partial M_{ij}} f(\mathbf{M}) \right] f^{-1}(\mathbf{M}) \tag{C.14}$$

$$\frac{\partial}{\partial M_{ij}} \det(\mathbf{M}) = \det(\mathbf{M}) M^{-1}_{ji}. \tag{C.15}$$

Note that the expressions obtained in this way can be gathered for entire matrices by using the identity

$$\mathbf{P}\mathbf{M}_{(i,j)}\mathbf{Q} = (\mathbf{P}^T \mathbf{Q}^T)_{ij}. \tag{C.16}$$

Taking the derivative with respect to $\mathbf{W} = \mathbf{A}^{-1}$ instead of the derivative with respect to $\mathbf{A}$ simplifies the final expression. The derivative of eqn. C.8 with respect to a single element W_{ij} is:

$$\frac{\partial}{\partial W_{ij}} \ln p(\mathbf{x}|\mathbf{A}) = \frac{\partial}{\partial W_{ij}} \left[\underbrace{\ln |\det(\mathbf{W})|}_{(I)} + \underbrace{\sum_{l=1}^{d} \ln p \left(\sum_{k=1}^{d} W_{lk} x_k \right)}_{(II)} \right]. \tag{C.17}$$

The derivative of (I) is quite easy to find using eqn. C.15:

$$\begin{aligned} \frac{\partial}{\partial W_{ij}} \ln |\det(\mathbf{W})| &= |\det(\mathbf{W})|^{-1} \frac{\partial}{\partial W_{ij}} |\det(\mathbf{W})| \\ &= |\det(\mathbf{W})|^{-1} |\det(\mathbf{W})| W_{ji}^{-1} = A_{ji}. \end{aligned} \tag{C.18}$$

For (II), it is useful to denote the estimate of the source s_i by

$$u_i = \sum_{k=1}^{d} W_{ik} x_k, \tag{C.19}$$

since $\mathbf{u} = \mathbf{A}^{-1}\mathbf{x} = \mathbf{W}\mathbf{x}$. Then (II) can be written as:

$$\begin{aligned} \frac{\partial}{\partial W_{ij}} \sum_{l=1}^{d} \ln p \left(\sum_{k=1}^{d} W_{lk} x_k \right) &= \sum_{l=1}^{d} \left[\frac{\partial \ln p \left(\sum_{k=1}^{d} W_{lk} x_k \right)}{\partial \sum_{k=1}^{d} W_{lk} x_k} \right] \left[\frac{\partial \sum_{k=1}^{d} W_{lk} x_k}{\partial W_{ij}} \right] \\ &= \phi_i(u_i) x_j, \end{aligned} \tag{C.20}$$

in which the nonlinearity $\phi_i(u_i)$ is the derivative of the log-likelihood of one source u_i w.r.t. u_i itself:

$$\phi_i(u_i) = \frac{\partial \ln p(u_i)}{\partial u_i} = \frac{1}{p(u_i)} \frac{\partial p(u_i)}{\partial u_i} \tag{C.21}$$

Choosing a function $\phi_i(u_i)$ amounts to choosing a model distribution for s_i. For example, using $\phi_i(u_i) = -cu_i$, i.e. no nonlinearity, is equivalent to assuming s_i is normally distributed. Using $\phi_i(u_i) = -\tanh(u_i)$ is equivalent to assuming $p(s_i) = \cosh^{-1}(s_i)$, i.e. a heavier-tailed distribution than the Gaussian [227].

The total gradient (eqn. C.17) therefore becomes:

$$\frac{\partial}{\partial W_{ij}} \ln p(\mathbf{x}|\mathbf{A}) = A_{ji} + \phi_i(u_i) x_j \tag{C.22}$$

or, for matrices,

$$\frac{\partial}{\partial \mathbf{W}} \ln p(\mathbf{x}|\mathbf{A}) = \mathbf{A}^T + [\phi_1(u_1) \dots \phi_d(u_d)]^T \mathbf{x}^T = \mathbf{A}^T + \phi(\mathbf{u})\mathbf{x}^T. \tag{C.23}$$

The learning rule is

$$\Delta \mathbf{W} = \eta \left[\mathbf{A}^T + \phi(\mathbf{u})\mathbf{x}^T\right] \tag{C.24}$$

where η is a learning rate. The batch learning rule, for the entire N-sample dataset (where $\mathbf{U}$ is now the matrix containing the source approximations $\mathbf{u}$ as its columns and $\mathbf{X}$ likewise), is:

$$\Delta \mathbf{W} = \eta \left[N\mathbf{A}^T + \phi(\mathbf{U})\mathbf{X}^T\right] \tag{C.25}$$

This rule was found by various authors [20, 212, 227]. The assumptions made to find it were:

- there is no noise;
- matrix $\mathbf{A}$ is full rank, i.e. there are as many sources as there are mixtures.

C.1.2 Extended infomax model distributions

The only remaining choice is now how to model the distributions of the sources u_i. In [212], Lee et al. propose using two different models, one for sub-Gaussian sources and one for super-Gaussian sources. A switching matrix $\mathbf{K}$ is then used to decide per source which distribution is most likely.

A sub-Gaussian distribution

A sub-Gaussian model is found by noting that the Pearson distribution function,

$$p(u_i) = f_{sub}(u_i;\ \mu, \sigma^2) = \frac{1}{2}\left(f_N(u_i;\ \mu, \sigma^2) + f_N(u_i;\ -\mu, \sigma^2)\right), \tag{C.26}$$

where $f_N(u_i;\ \mu, \sigma^2)$ is a Gaussian distribution function, can model any distribution between a Gaussian ($\mu = 0$) and a bimodal, sub-Gaussian distribution ($\mu \gtrapprox 1.5$). Calcu-

lating $\phi_i(u_i)$ using eqn. C.21, writing c' for $\frac{1}{\sqrt{2\pi\sigma^2}}$, gives:

$$\begin{aligned}
\phi_i(u_i) &= \frac{1}{p(u_i)}\frac{\partial p(u_i)}{\partial u_i} \\
&= \frac{\frac{1}{2}c'\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)\cdot-\left(\frac{u_i-\mu}{\sigma}\right)\frac{1}{\sigma}+\frac{1}{2}c'\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)\cdot-\left(\frac{u_i+\mu}{\sigma}\right)\frac{1}{\sigma}}{\frac{1}{2}c'\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)+\frac{1}{2}c'\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)} \\
&= -\frac{u_i}{\sigma^2}\left[\frac{\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)+\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)}{\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)+\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)}\right]+ \\
&\qquad \frac{\mu}{\sigma^2}\left[\frac{\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)-\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)}{\exp\left(-\frac{1}{2}\left(\frac{u_i-\mu}{\sigma}\right)^2\right)+\exp\left(-\frac{1}{2}\left(\frac{u_i+\mu}{\sigma}\right)^2\right)}\right] \\
&= -\frac{u_i}{\sigma^2}+\frac{\mu}{\sigma^2}\left[\frac{\exp\left(\frac{u_i\mu}{\sigma^2}\right)\exp\left(-\frac{1}{2}\frac{u_i^2+\mu^2}{\sigma^2}\right)-\exp\left(-\frac{u_i\mu}{\sigma^2}\right)\exp\left(-\frac{1}{2}\frac{u_i^2+\mu^2}{\sigma^2}\right)}{\exp\left(\frac{u_i\mu}{\sigma^2}\right)\exp\left(-\frac{1}{2}\frac{u_i^2+\mu^2}{\sigma^2}\right)+\exp\left(-\frac{u_i\mu}{\sigma^2}\right)\exp\left(-\frac{1}{2}\frac{u_i^2+\mu^2}{\sigma^2}\right)}\right] \\
&= -\frac{u_i}{\sigma^2}+\frac{\mu}{\sigma^2}\left[\frac{\exp\left(\frac{\mu}{\sigma^2}u_i\right)-\exp\left(-\frac{\mu}{\sigma^2}u_i\right)}{\exp\left(\frac{\mu}{\sigma^2}u_i\right)+\exp\left(-\frac{\mu}{\sigma^2}u_i\right)}\right] \\
&= -\frac{u_i}{\sigma^2}+\frac{\mu}{\sigma^2}\tanh\left(\frac{\mu}{\sigma^2}u_i\right), \qquad \text{(C.27)}
\end{aligned}$$

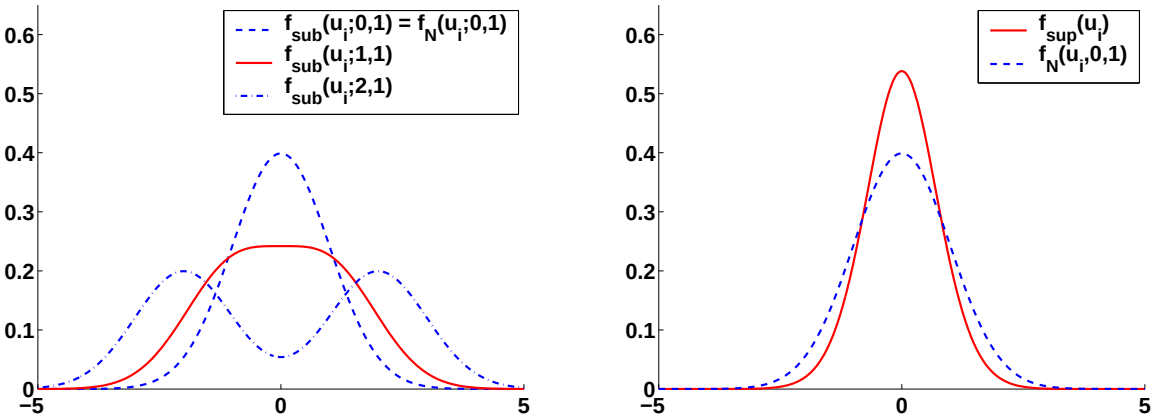
which, setting $\mu = \sigma^2 = 1$, reduces to:

$$\phi_i(u_i) = \tanh(u_i) - u_i, \qquad \text{(C.28)}$$

giving for the learning rule in eqn. C.24:

$$\Delta\mathbf{W} = \eta\left[\mathbf{A}^T + (\tanh(\mathbf{u}) - \mathbf{u})\mathbf{x}^T\right]. \qquad \text{(C.29)}$$

Although the distribution given by eqn. C.26 is not yet bimodal for $\mu = 1$, it is sub-Gaussian; see figure C.1 (a). From here on, $f_{sub}(u_i)$ will be used to denote $f_{sub}(u_i;\ 1, 1)$.



(a) (b)

Figure C.1: (a) The sub-Gaussian function given by eqn. C.26, for $\sigma = 1$ and various values of μ. (b) The super-Gaussian function given by eqn. C.30, compared to a normal Gaussian.

A super-Gaussian distribution

Following more or less the same sort of derivation, starting with

$$p(u_i) = f_{sup}(u_i) = c f_N(u_i;\, 0,1)\,\mathrm{sech}(u_i) = \frac{\frac{c}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}u_i^2\right)}{\cosh(u_i)}$$
$$= \left[c'' \exp\left(-\frac{1}{2}u_i^2\right)\right]\left[2\left(\exp(u_i)+\exp(-u_i)\right)^{-1}\right], \qquad \text{(C.30)}$$

with $f_N(u_i;\, 0,1)$ the standard normal distribution, one gets:

$$\begin{aligned}\phi_i(u_i) &= \frac{1}{p(u_i)}\frac{\partial p(u_i)}{\partial u_i} \\ &= \frac{1}{c'' \exp\left(-\frac{1}{2}u_i^2\right) 2\left(\exp(u_i)+\exp(-u_i)\right)^{-1}} \\ &\quad \left[c'' \exp\left(-\frac{1}{2}u_i^2\right)\cdot(-u_i)\cdot 2\left(\exp(u_i)+\exp(-u_i)\right)^{-1} + \right. \\ &\quad \left. c'' \exp\left(-\frac{1}{2}u_i^2\right)\cdot -2\left(\exp(u_i)+\exp(-u_i)\right)^{-2}\left(\exp(u_i)-\exp(-u_i)\right)\right] \\ &= -u_i - \frac{\exp(u_i)-\exp(-u_i)}{\exp(u_i)+\exp(-u_i)} = -u_i - \tanh(u_i)\,. \end{aligned} \qquad \text{(C.31)}$$

In eqn. C.30, c is a constant needed to make f_{sup} a true density function. Numerical integration shows that $c \approx \frac{1}{0.741264}$ makes the integral of f_{sup} equal to 1.

This gives as a learning rule:

$$\Delta \mathbf{W} = \eta \left[\mathbf{A}^T + (-\tanh(\mathbf{u}) - \mathbf{u})\mathbf{x}^T \right]. \tag{C.32}$$

The nonlinearity used makes for a peaked distribution, but not much more peaked than a Gaussian – see figure C.1 (b).

Switching matrix

Noting the similarity between eqns. C.29 and C.32, Lee proposes to use a *switching matrix* $\mathbf{K}$, a diagonal matrix in which $K_{ii} = -1$ if u_i has a sub-Gaussian distribution and $K_{ii} = 1$ if u_i has a super-Gaussian distribution. This allows the single sample learning rule to be written as

$$\Delta \mathbf{W} = \eta \left[\mathbf{A}^T - (\mathbf{K}\tanh(\mathbf{u}) + \mathbf{u})\,\mathbf{x}^T \right]. \tag{C.33}$$

The switching matrix can be found by, for example, setting K_{ii} according to the kurtosis of the distribution of u_i. However, Lee uses a stability criterion [212] to find the following expression for K_{ii}:

$$K_{ii} = \text{sign}\left[E(\text{sech}^2(u_i))E(u_i^2) - E(u_i \tanh(u_i)) \right]. \tag{C.34}$$

Approximation of likelihood

Equations C.26 and C.30 can also be used to approximate the distribution of $\mathbf{s}$ using $\mathbf{u}$, to use in the log-likelihood $\mathfrak{L}$ (eqn. C.9). For sub-Gaussian sources ($K_{ii} = -1$), using the settings $\mu = 1$ and $\sigma = 1$ as before, the likelihood for the entire data set is:

$$\begin{aligned}
\ln p(u_i) &= \ln \prod_{n=1}^{N} \frac{1}{2}\left[f_N(u_i;\, 1, 1) + f_N(u_i;\, -1, 1) \right] \\
&= \ln \frac{1}{(2\sqrt{2\pi})^N} + \sum_{n=1}^{N} \ln \left[\exp\left(-\frac{1}{2}(u_i^n - 1)^2 \right) + \exp\left(-\frac{1}{2}(u_i^n + 1)^2 \right) \right] \\
&= -N\ln(2\sqrt{2\pi}) + \sum_{n=1}^{N} \ln \left[\exp\left(-\frac{1}{2}\left((u_i^n)^2 + 1 \right) \right) \left(\exp(u_i^n) + \exp(-u_i^n) \right) \right] \\
&= -N\ln(2\sqrt{2\pi}) + \sum_{n=1}^{N} \left[-\frac{1}{2}\left((u_i^n)^2 + 1 \right) + \ln\left(\exp(u_i^n) + \exp(-u_i^n) \right) \right] \\
&= -N\ln(2\sqrt{2\pi}) + \sum_{n=1}^{N} \left[\ln(2\cosh(u_i^n)) - \frac{1}{2}(u_i^n)^2 - \frac{1}{2} \right] \\
&= -\frac{1}{2}N(1 + \ln 2\pi) + \sum_{n=1}^{N} \left[\ln\cosh(u_i^n) - \frac{1}{2}(u_i^n)^2 \right],
\end{aligned} \tag{C.35}$$

and for super-Gaussian sources ($K_{ii} = 1$),

$$\begin{aligned}\ln p(u_i) &= \ln \prod_{n=1}^{N} c f_N(u_i;\ 0,1)\,\mathrm{sech}\,(u_i^n) = \ln \prod_{n=1}^{N} \frac{\frac{c}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(u_i^n\right)^2\right)}{\cosh\left(u_i^n\right)} \\ &= \ln\left[\left(\frac{c}{\sqrt{2\pi}}\right)^N\right] + \sum_{n=1}^{N} \ln\left[\cosh^{-1}(u_i^n) \exp\left(-\frac{1}{2}(u_i^n)^2\right)\right] \\ &= -\frac{1}{2}N(-2\ln c + \ln 2\pi) + \sum_{n=1}^{N}\left[-\ln\cosh(u_i^n) - \frac{1}{2}(u_i^n)^2\right]. \end{aligned} \tag{C.36}$$

Lee [212] leaves out the constants in eqns. C.35 and C.36 and arrives at a single expression:

$$\ln p(u_i) = \sum_{n=1}^{N}\left[-\mathbf{K}\ln\cosh(u_i^n) - \frac{1}{2}(u_i^n)^2\right]. \tag{C.37}$$

Although simple to use, this equation would give wrong likelihood values.

Note that there is a problem when eqns. C.35 and C.36 are used for comparing likelihoods, e.g. to see which model is more applicable: they do not have equal variance. The variance of the sub-Gaussian distribution, a sum of two Gaussians, is simply the sum of the individual variances, i.e. 2. Numerical integration shows that the variance of the super-Gaussian distribution is approximately 0.591833. Therefore, the following expression can be used to calculate the likelihood of the data belonging to a unit variance sub-Gaussian distribution:

$$\ln p(u_i) = -\frac{1}{2}N(1 + \ln 2\pi - \ln 2) + \sum_{n=1}^{N}\left[\ln\left(\cosh\left(\sqrt{2}\,u_i^n\right)\right) - \frac{1}{2}\left(\sqrt{2}\,u_i^n\right)^2\right] \tag{C.38}$$

and the expression for the unit variance super-Gaussian distribution becomes:

$$\begin{aligned}\ln p(u_i) = &-\frac{1}{2}N(2\ln 0.741264 + \ln 2\pi - \ln 0.591833) \\ &+ \sum_{n=1}^{N}\left[-\ln\cosh\left(\sqrt{0.591833}\,u_i^n\right) - \frac{1}{2}\left(\sqrt{0.591833}\,u_i^n\right)^2\right]. \end{aligned} \tag{C.39}$$

C.2 Undercomplete ICA bases

C.2.1 Learning rule

The model for finding undercomplete ICA bases, in which the number of sources m is smaller than the number of dimensions in the original data set d, is similar to eqn. C.1,

but now contains a noise term:

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \boldsymbol{\mu} + \boldsymbol{\epsilon}, \tag{C.40}$$

where $\mathbf{s}$ is now an m-dimensional vector containing the independent sources, $\mathbf{A}$ is a $d \times m$ mixing matrix and $\boldsymbol{\epsilon}$ is Gaussian distributed noise. Again, the mean is assumed to have been removed ($\boldsymbol{\mu} = \mathbf{0}$). As parts of the derivation of the learning rule for this model are identical in the derivation of the learning rule in the previous section, the focus here will be on the differences.

The probability of observing a data vector $\mathbf{x}$ given the latent vector $\mathbf{s}$ now becomes:

$$p(\mathbf{x}|\mathbf{s}, \mathbf{A}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\det(\mathbf{C})|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{A}\mathbf{s})^T \mathbf{C}^{-1} (\mathbf{x} - \mathbf{A}\mathbf{s})\right). \tag{C.41}$$

The assumption on the noise is that it is i.i.d. and small:

$$\begin{aligned} C_{ij} &= 0, \quad \forall i \neq j \\ C_{ii} &= \sigma^2 = \frac{1}{\beta}. \end{aligned} \tag{C.42}$$

This simplifies eqn. C.41 to:

$$p(\mathbf{x}|\mathbf{s}, \mathbf{A}) = \frac{1}{(2\pi)^{\frac{d}{2}} \beta^{-\frac{d}{2}}} \exp\left(-\frac{\beta}{2}(\mathbf{x} - \mathbf{A}\mathbf{s})^T (\mathbf{x} - \mathbf{A}\mathbf{s})\right). \tag{C.43}$$

In the following derivation, $\mathbf{W}$ can no longer be expressed as the inverse of $\mathbf{A}$, as the matrices are no longer square. The Moore-Penrose pseudo-inverse can be used, however:

$$\mathbf{W} = \mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T. \tag{C.44}$$

This implies the following identities, which will be used below:

$$\mathbf{A} = \mathbf{W}^+ = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1} \tag{C.45}$$
$$\mathbf{A}^T\mathbf{A} = (\mathbf{W}\mathbf{W}^T)^{-1} \tag{C.46}$$
$$\mathbf{W}^T\mathbf{A}^T = (\mathbf{A}\mathbf{W})^T = \mathbf{A}\mathbf{W} \tag{C.47}$$
$$\mathbf{A}\mathbf{W}\mathbf{A}\mathbf{W} = \mathbf{A}\mathbf{W}. \tag{C.48}$$

The likelihood for one vector $\mathbf{x}$ again is:

$$p(\mathbf{x}|\mathbf{A}) = \int p(\mathbf{x}|\mathbf{s}, \mathbf{A}) p(\mathbf{s}) d\mathbf{s}. \tag{C.49}$$

Now if β is large enough (i.e. the noise variance σ^2 is small enough), the term $p(\mathbf{x}|\mathbf{s},\mathbf{A})$ has a single peak at, say, $\mathbf{s}_{MP}$ that dominates the integral in each dimension. In other words, the distribution is such that the optimal value $\mathbf{s}_{MP}$ can be used instead of the entire distribution, just as the Kronecker function was used in eqn. C.6. As before, the change in volume due to the matrix $\mathbf{A}$ will have to be normalised for. The log-likelihood then becomes

$$\begin{aligned}\ln p(\mathbf{x}|\mathbf{A}) &\approx \ln\left[p(\mathbf{x}|\mathbf{s}_{MP},\mathbf{A})p(\mathbf{s}_{MP})|\det(\mathbf{A}^T\mathbf{A})|^{-\frac{1}{2}}\right] \\ &= \ln\left[\frac{1}{(2\pi)^{\frac{d}{2}}\beta^{-\frac{d}{2}}}\exp\left(-\frac{\beta}{2}(\mathbf{x}-\mathbf{A}\mathbf{s}_{MP})^T(\mathbf{x}-\mathbf{A}\mathbf{s}_{MP})\right)|\det(\mathbf{A}^T\mathbf{A})|^{-\frac{1}{2}}p(\mathbf{s}_{MP})\right] \\ &= \frac{d}{2}\ln\frac{\beta}{2\pi}-\frac{\beta}{2}\left(\mathbf{x}-\mathbf{A}\mathbf{s}_{MP}\right)^T\left(\mathbf{x}-\mathbf{A}\mathbf{s}_{MP}\right)-\frac{1}{2}\ln|\det(\mathbf{A}^T\mathbf{A})|+\ln p(\mathbf{s}_{MP}),\end{aligned} \tag{C.50}$$

with

$$\mathbf{s}_{MP}=\mathbf{W}\mathbf{x}. \tag{C.51}$$

In this formula, the previous normalisation by $|\det(\mathbf{A})|^{-1}$ has been replaced by $|\det(\mathbf{A}^T\mathbf{A})|^{-\frac{1}{2}}$, as $\mathbf{A}$ is no longer a square matrix.

Taking the derivative is slightly more complicated than before. Starting with the second term, denoting the estimate of $\mathbf{s}_{MP}$ by $\mathbf{u}$ and using equations C.12-C.15 and C.45-C.48, one obtains

$$\begin{aligned}\frac{\partial}{\partial W_{ij}}\left[-\frac{\beta}{2}\left(\mathbf{x}-\mathbf{A}\mathbf{u}\right)^T\left(\mathbf{x}-\mathbf{A}\mathbf{u}\right)\right] &= -\frac{\beta}{2}\frac{\partial}{\partial W_{ij}}\left[\mathbf{x}^T\mathbf{x}-\mathbf{u}^T\mathbf{A}^T\mathbf{x}-\mathbf{x}^T\mathbf{A}\mathbf{u}-\mathbf{u}^T\mathbf{A}^T\mathbf{A}\mathbf{u}\right] \\ &= -\frac{\beta}{2}\frac{\partial}{\partial W_{ij}}\left[\mathbf{x}^T\mathbf{x}-\mathbf{x}^T\mathbf{W}^T\mathbf{A}^T\mathbf{x}-\mathbf{x}^T\mathbf{A}\mathbf{W}\mathbf{x}+\mathbf{x}^T\mathbf{A}\mathbf{W}\mathbf{x}\right] \\ &= -\frac{\beta}{2}\frac{\partial}{\partial W_{ij}}\left[-\mathbf{x}^T\mathbf{A}\mathbf{W}\mathbf{x}\right] \\ &= -\frac{\beta}{2}\frac{\partial}{\partial W_{ij}}\left[-\mathbf{x}^T\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}\mathbf{x}\right] \\ &= -\frac{\beta}{2}\Big[-\mathbf{x}^T\mathbf{W}_{(j,i)}\left[(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}\mathbf{x}\right] \\ &\qquad -\mathbf{x}^T\mathbf{W}^T\left[-(\mathbf{W}\mathbf{W}^T)^{-1}\frac{\partial\mathbf{W}\mathbf{W}^T}{\partial\mathbf{W}_{ij}}(\mathbf{W}\mathbf{W}^T)^{-1}\right]\mathbf{W}\mathbf{x} \\ &\qquad -\mathbf{x}^T\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}_{(i,j)}\mathbf{x}\Big]\end{aligned} \tag{C.52}$$

$$
\begin{aligned}
&= -\frac{\beta}{2}\Big[-\mathbf{x}^T\mathbf{W}_{(j,i)}\left[(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}\mathbf{x}\right] \\
&\quad -\mathbf{x}^T\mathbf{W}^T\left[-(\mathbf{W}\mathbf{W}^T)^{-1}\left[\mathbf{W}_{(i,j)}\mathbf{W}^T+\mathbf{W}\mathbf{W}_{(j,i)}\right](\mathbf{W}\mathbf{W}^T)^{-1}\right]\mathbf{W}\mathbf{x} \\
&\quad -\mathbf{x}^T\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}_{(i,j)}\mathbf{x}\Big] \\
&= -\frac{\beta}{2}\left[-\mathbf{x}^T\mathbf{W}_{(j,i)}\mathbf{A}^T\mathbf{x}+\mathbf{x}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{A}\mathbf{W}\mathbf{x}+\mathbf{x}^T\mathbf{W}^T\mathbf{A}^T\mathbf{W}_{(j,i)}\mathbf{A}^T\mathbf{x}-\mathbf{x}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{x}\right].
\end{aligned}
\tag{C.53}
$$

Now since the first and fourth term result in scalars and they are each others transposed, they are identical. This also holds for the second term and third term. This simplifies the equation to:

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}}\left[-\frac{\beta}{2}(\mathbf{x}-\mathbf{A}\mathbf{u})^T(\mathbf{x}-\mathbf{A}\mathbf{u})\right] &= -\frac{\beta}{2}\left[-2\mathbf{x}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{x}+2\mathbf{x}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{A}\mathbf{W}\mathbf{x}\right] \\
&= \beta\left[\mathbf{A}^T\mathbf{x}\mathbf{x}^T+\mathbf{A}^T\mathbf{x}\mathbf{x}^T\mathbf{W}^T\mathbf{A}^T\right].
\end{aligned}
\tag{C.54}
$$

Although the derivatives of the last two terms in eqn. C.50 are also slightly harder to compute, the results are the same as those found before in eqn. C.33. The single sample learning rule therefore becomes:

$$
\Delta\mathbf{W} = \eta\left[\beta\mathbf{A}^T\mathbf{x}\mathbf{x}^T(\mathbf{I}-\mathbf{A}\mathbf{W})+\mathbf{A}^T-(\mathbf{K}\tanh(\mathbf{u})+\mathbf{u})\,\mathbf{x}^T\right]. \tag{C.55}
$$

It is obvious that in the case where $m = d$ and therefore $\mathbf{W} = \mathbf{A}^{-1}$, the first term of the derivative is zero and the learning rule is identical to the one found earlier (eqn. C.33). Furthermore, in the first term the covariance matrix $\mathbf{C} = E(\mathbf{x}\mathbf{x}^T)$ can be recognised; the batch update rule can be written as:

$$
\Delta\mathbf{W} = \eta\left[\beta N\mathbf{A}^T\mathbf{C}(\mathbf{I}-\mathbf{A}\mathbf{W})+N\mathbf{A}^T-(\mathbf{K}\tanh(\mathbf{U})+\mathbf{U})\,\mathbf{X}^T\right]. \tag{C.56}
$$

The first term is an orthogonalisation term which works in the space rotated and scaled by $\mathbf{C}$.

C.3 The algorithm

C.3.1 Pre-whitening

Equation C.55 gives the basic learning rule for $\mathbf{W}$. However, the noise parameter β has to be estimated as well. This can easily be done using the GEM algorithm. In each step,

estimate the current parameters in the E-step by:

$$\mathbf{A} = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1} \tag{C.57}$$

$$\mathbf{u}^n = \mathbf{W}\mathbf{x}^n,\ i = 1,\ldots,N \tag{C.58}$$

$$\mathbf{K}_{ii} = \text{sign}\left[E(\text{sech}^2(u_i))E(u_i^2) - E(u_i\tanh(u_i))\right],\ \forall i = 1,\ldots,m \tag{C.59}$$

$$\mathbf{v}^n = \mathbf{W}^\top\mathbf{x}^n,\ i = 1,\ldots,N \tag{C.60}$$

$$\beta^{-1} = \frac{1}{d-m}\sum_{i=1}^{d-m}\text{var}(v_i), \tag{C.61}$$

where $\mathbf{W}^\top$ is the nullspace of $\mathbf{W}$, i.e. β corresponds to the inverse of the average noise outside the subspace (cf. probabilistic PCA, eqn. 7.27 on page 142). In the M-step, the log-likelihood is then maximised by applying the learning rule (eqn. C.56) to $\mathbf{W}$.

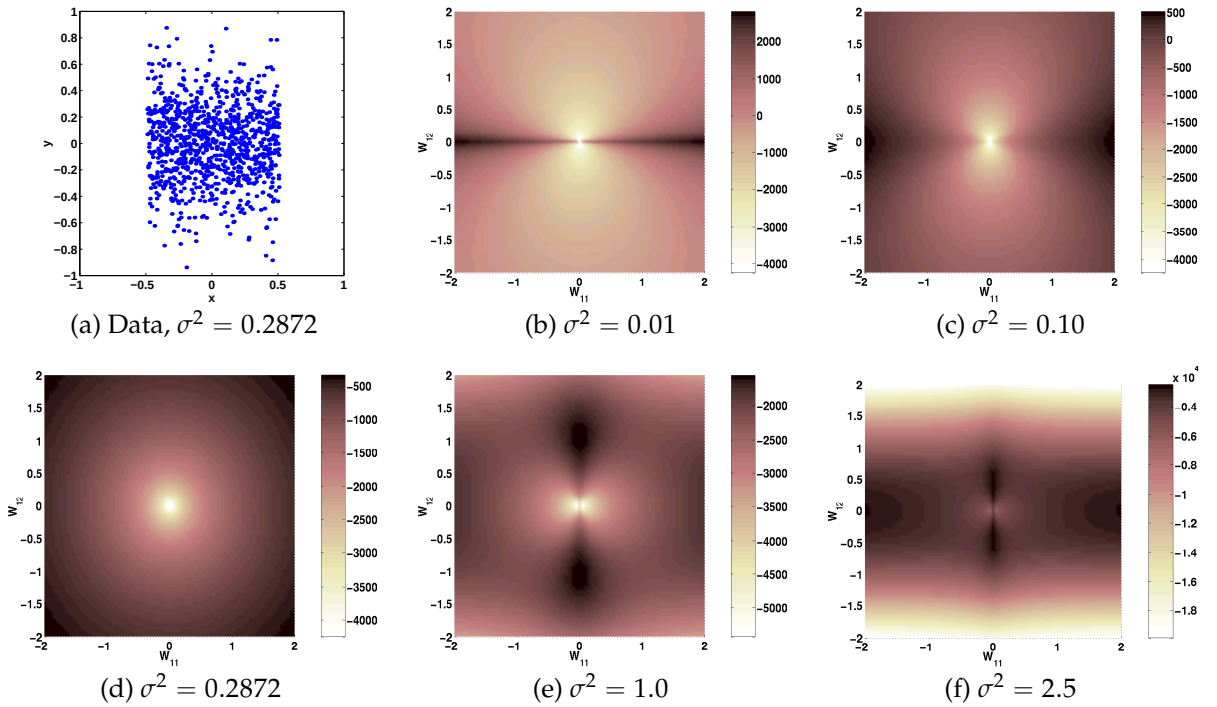
However, the new algorithm does not only find independent components: it finds subspaces as well. In the log-likelihood, eqn. C.50, there is an inherent trade-off between finding a subspace (the second term) and finding independent components (the fourth term). This trade-off is controlled by β, which decides how much weight the subspace term has. As the estimate of β is based on the current orientation of the subspace, it might be estimated incorrectly and the algorithm might converge to a local maximum. Moreover, the data might not fit the model in which the noise has equal variance in all directions, which would also lead to incorrect estimates.

This is illustrated in figure C.2. The data set consists of 2D samples of which the x-coordinate is drawn from a uniform distribution in the range $[-0.5, 0.5]$ with variance ≈ 0.2872 and the y-coordinate is drawn from a Gaussian distribution with zero mean and variance σ^2. If β is fixed at σ^{-2}, the algorithm converges for any setting of σ. However, if β is learned, this is not always the case. For various settings of σ^2, figure C.2 indicates the likelihood $\mathfrak{L}$ of the model for each setting of $\mathbf{W}$. Clearly, as σ increases, finding the independent component becomes less likely than finding the Gaussian component, depending on initialisation.

To find only independent components, the role variance plays will have to be eliminated. A solution would be to use β as a parameter instead of a variable. It can be fixed at a small value, e.g. 0.1, to stress finding independent components rather than subspaces. However, in the experiments in chapter 7 and this appendix, another method was chosen: the data was whitened before applying the ICA subspace algorithm. This means using $\mathbf{x}' = \mathbf{C}^{-\frac{1}{2}}\mathbf{x}$, which is often done in ICA algorithms. This will make the data variance 1 in all directions.

Pre-whitening also simplifies the algorithm in a number of ways:

- β can be fixed at 1;
- the rows of $\mathbf{W}$ and columns of $\mathbf{A}$ can be constrained to have unit length;



(a) Data, $\sigma^2 = 0.2872$ (b) $\sigma^2 = 0.01$ (c) $\sigma^2 = 0.10$

(d) $\sigma^2 = 0.2872$ (e) $\sigma^2 = 1.0$ (f) $\sigma^2 = 2.5$

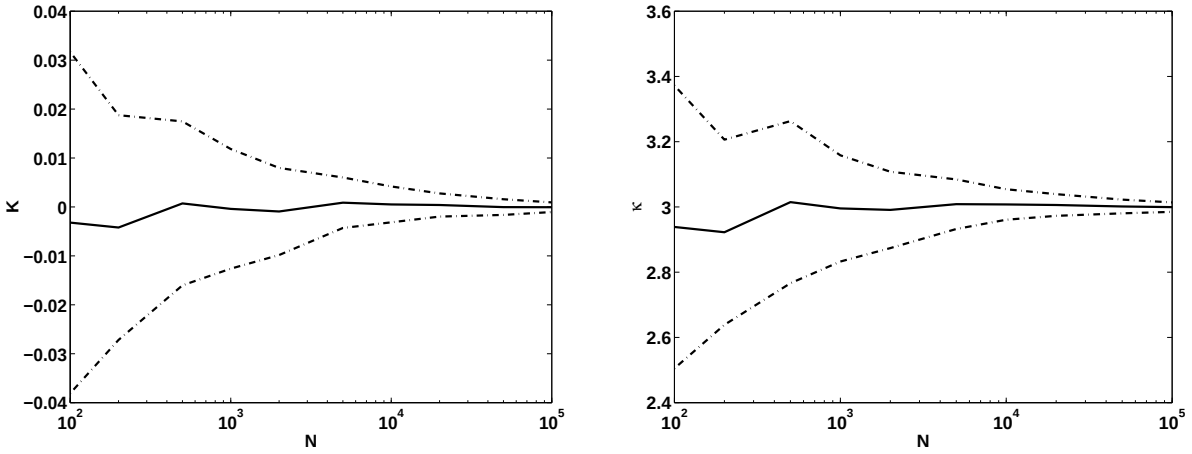
Figure C.2: (a) A simple 2D data set. (b)-(f) The likelihood $\mathfrak{L}$ as a function of $\mathbf{W}$. For small σ^2, the local maxima correspond to the independent component; for large σ^2, they correspond to the Gaussian component.

- in the batch learning rule only, the first term of eqn. C.56 drops out, as for the whitened data $\mathbf{C} = \mathbf{I}$:

$$N\mathbf{A}^T\mathbf{C}(\mathbf{I} - \mathbf{AW}) = N\left(\mathbf{A}^T - \mathbf{A}^T\mathbf{AW}\right)$$
$$= N\left(\mathbf{A}^T - \mathbf{A}^T\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\right) = \mathbf{0}. \tag{C.62}$$

  so the learning rule is identical to the one originally proposed by Lee at al. (eqn. C.33).

This discussion exposes a problem of most ICA algorithms: the high sensitivity to variance estimates. Although for simple 1D or 2D problems the learning rule in eqn. C.55 can be used, the difficulty in estimating β means that for high-dimensional problems the only working solution is to use prior knowledge of β or to pre-whiten the data. As by pre-whitening the subspace nature of the model no longer plays a role (i.e. there is no more subspace structure in the data), the rule is simplified to eqn. C.33, the one proposed originally by Lee.



(a) (b)

Figure C.3: Mean $\pm$ standard deviation over 100 repetitions of the estimate of (a) the switching matrix $\mathbf{K}$ and (b) the kurtosis of a 1D Gaussian data set, as a function of the sample size N.

C.3.2 Implementation

The algorithm outlined in section C.3 still needs a few settings. The learning rate, η, was experimentally found to be optimal when it is set to approximately $\frac{0.1}{N}$. If it is set larger, the algorithm can diverge. Second, the algorithm needs to be initialised. $\mathbf{W}$ was initialised simply to a matrix containing random elements drawn from a uniform distribution in the range $[-0.5, 0.5]$, after which the rows of $\mathbf{W}$ were made orthonormal.

To avoid fitting noise, which is a problem in ICA (see below), image data was usually pre-mapped using PCA to retain $r = 90\%$ of the variance. Note that this automatically whitens the data in the remaining dimensions too, so the whitening step discussed in the previous section does not need to be applied.

C.3.3 Sample size requirements

ICA algorithms are notoriously sensitive to noise [177] due to their use of higher order moments. In the ML algorithm discussed here, this holds especially true for the calculation of $\mathbf{K}$, the switching matrix. Once a wrong $\mathbf{K}_{ii}$ is found for a certain source u_i, the algorithm may diverge from the right solution. Figure C.3 illustrates this by plotting the mean and standard deviation of the estimates of $\mathbf{K}$ and the kurtosis κ over 100 realisations of a 1D Gaussian data set, as a function of the sample size. For a Gaussian, $\mathbf{K}$ should be zero and κ should be 3. Clearly, for small sample sizes the estimate has a very large error. Therefore, in the experiments on images, data sets of 10,000 samples were used.

C.4 Application

To illustrate the working of the ML-ICA algorithm, this section discusses some experiments on simple signal and 2D data sets and a data set containing patches taken from a set of natural images. The outcome of the ML algorithm is compared to that of a well-known and often applied ICA algorithm, fastICA [170, 174] (see also chapter 21).

C.4.1 Blind source separation

In the first toy problem, three simple signals were mixed: a sine wave, a sawtooth wave and a square wave with slightly different frequencies. Each signal contained 2,000 samples. The mixing matrix elements were drawn from a $U(0,1)$ random distribution. ML-ICA and fastICA were then applied to find two independent components. The results are shown in figure C.4. Clearly, ML-ICA and fastICA give similar results, although they may pick two different ICs depending on initialisation.

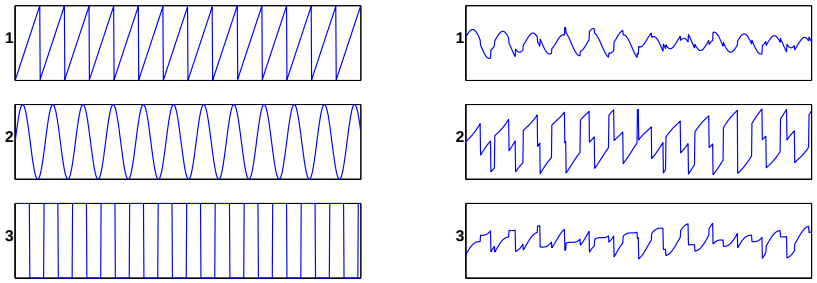
C.4.2 Density estimation

In this problem, a density estimate is found on a simple 2D dataset. This data set is a mixture of a Laplacian and a uniformly distributed component. Figure C.5 shows the components found, density estimates using both the ML-ICA model and a Gaussian model (cf. section 7.4.1), and the projected data distributions with their kurtoses. The likelihood $\mathfrak{L}$ of the data belonging to the ML-ICA model is higher than it belonging to the Gaussian model, but not much higher. The model found fits the data well and gives projections with a high kurtosis (for Laplacian distribution) and a low kurtosis (the uniform distribution).

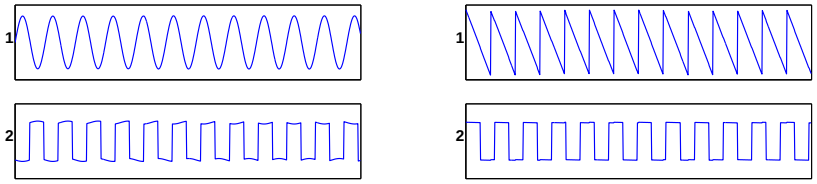
C.4.3 Natural image data

Finally, ML-ICA was applied to a data set of 10,000 12×12 pixel patches taken from natural image data. The four natural images, shown in figure C.6, have been used in publications before [21, 22, 166, 167, 214, 215, 262] and are known to lead to wavelet-like independent components.

The data set was drawn randomly from the four images and normalised by removing the mean of each sample and scaling it to unit standard deviation. It was then whitened using a PCA mapping $\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$ to retain $r = 90\%$ of the variance, leaving 59 dimensions. The fastICA algorithm was used to extract as many independent components as possible, and ML-ICA was applied with the number of components to extract, m, set to

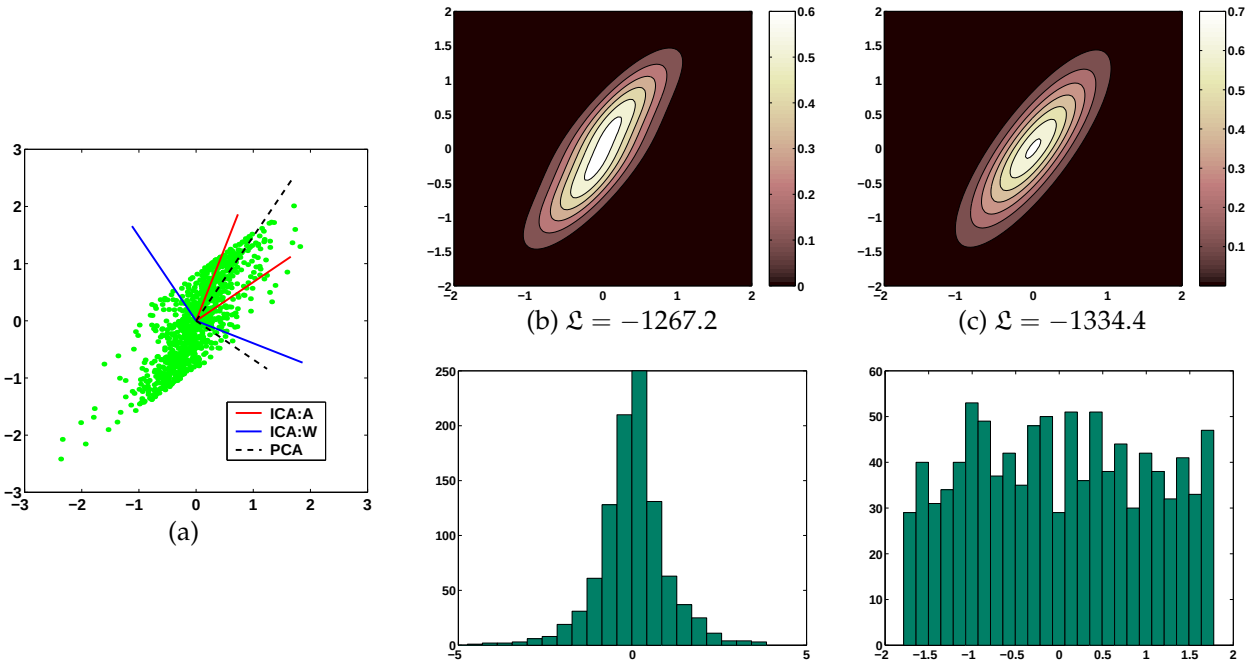


(a) Original (b) Mixed



(c) fastICA unmixed (d) ML-ICA unmixed

Figure C.4: Blind source separation: (a) original signals, (b) mixtures, (c) unmixed signals found using fastICA and (d) same, using ML-ICA.



(a) (b) $\mathfrak{L} = -1267.2$ (c) $\mathfrak{L} = -1334.4$

(d) $\kappa = 7.23$ (e) $\kappa = 1.87$

Figure C.5: ML-ICA density estimation: (a) original data set with principal and independent components; (b) density estimate using ML-ICA; (c) density estimate using a Gaussian; (d)-(e) histograms of projected data.

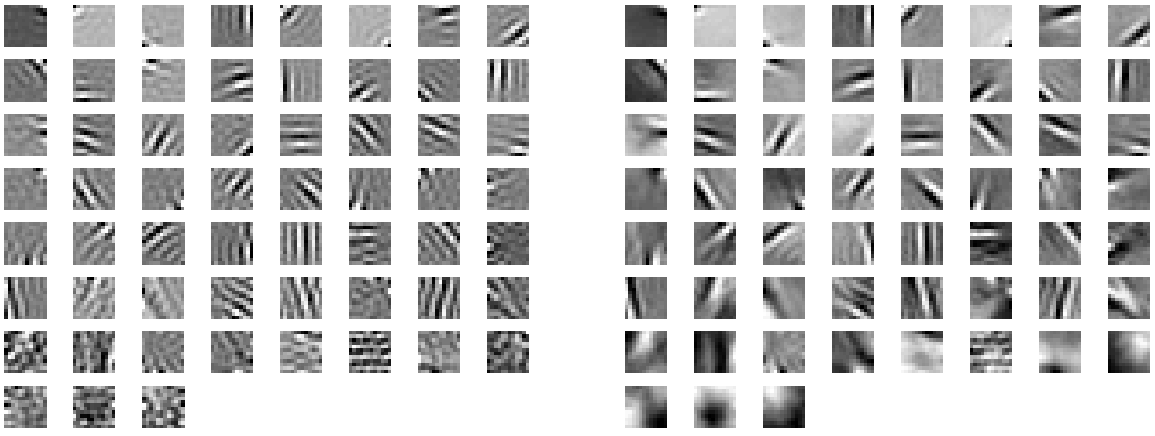
8, 16 and 32. The ICA filters and basis vectors thus found, projected back into the original space, are shown in figure C.7. To facilitate comparison, the resulting independent components were ordered by their duration (or spread) D [269]

$$D = \int\int (x^2 + y^2)|I(x,y)|^2 dxdy, \tag{C.63}$$

where (x, y) are pixel coordinates in image I.

Clearly, both fastICA and ML-ICA find similar components, often the same. They are also similar to those found in literature. The components result in super-Gaussian projection distributions with high kurtoses, roughly in the range $[7, 10]$. However, the ordering by bandwidth obscures the fact that fastICA finds these components in a more or less random order, i.e. the first components do not necessarily correspond to those giving the highest kurtosis. The least wavelet-like components, leading to lower kurtoses, are found by fastICA only. This, and the fact that many of the components are identical for $m = 8, 16$ and 32, indicates that ML-ICA succeeds in finding the most kurtotic distributions present in the data.

Figure C.6: Four natural images.



(a) fastICA: $\mathbf{W}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$

(b) fastICA: $\mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{A}$

(c) ML-ICA, $m = 8$: $\mathbf{W}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$

(d) ML-ICA, $m = 8$: $\mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{A}$

(e) ML-ICA, $m = 16$: $\mathbf{W}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$

(f) ML-ICA, $m = 16$: $\mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{A}$

(g) ML-ICA, $m = 32$: $\mathbf{W}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{E}^T$

(h) ML-ICA, $m = 32$: $\mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{A}$

Figure C.7: Filters (left) and base vectors (right) found by fastICA and ML-ICA.

BIBLIOGRAPHY

[1] A. Adler and R. Guardo. A neural network image reconstruction technique for electrical impedance tomography. *IEEE Transactions on Medical Imaging*, 13(4):594–600, 1994. Pages: 18

[2] A.K. Agrawala. *Machine recognition of patterns*. IEEE Press Selected Reprint Series. IEEE Press, New York, NY, 1977. Pages: 2

[3] M.N. Ahmed and A.A. Farag. Two-stage neural network for volume segmentation of medical images. *Pattern Recognition Letters*, 18(11-13):1143–1151, 1997. http://www.cvip.uofl.edu/Papers.html. Pages: 24

[4] A.D. Alexandrov, W.Y. Ma, A. El Abbadi, and B.S. Manjunath. Adaptive filtering and indexing for image databases. In W. Niblack and R.C. Jain, editors, *Storage and retrieval for image and video databases III*, volume 2420 of *Proc. of SPIE*, Bellingham, WA, 1995. SPIE, SPIE. http://www.cs.ucsb.edu/~berto/papers/. Pages: 181

[5] S. Amari, A. Cichocki, and H.H. Yang. A new learning algorithm for blind signal separation. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 757–763. MIT Press, Cambridge, MA, 1995. http://www.islab.brain.riken.go.jp/~amari/pub_j.html. Pages: 144

[6] R. Anand, K. Mehrotra, C.K. Mohan, and S. Ranka. Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, 6(1):117–124, 1995. Pages: 89

[7] D. Anguita, G. Parodi, and R. Zunino. Associative structures for vision. *Multidimensional Systems and Signal Processing*, 5(1):75–96, 1994. Pages: 28

[8] N. Ansari and Z.Z. Zhang. Generalised adaptive neural filters. *IEE Electronics Letters*, 29 (4):342–343, 1993. http://www-ec.njit.edu/~ang/papers/NN96.pdf. Pages: 19

[9] S. Antani, R. Kasturi, and R. Jain. Pattern recognition methods in image and video databases: past, present and future. In A. Amin, D. Dori, P. Pudil, and H. Freeman, editors, *Advances in Pattern Recognition, Proc. IAPR workshops on Structural and Statistical Pattern Recognition '98 (SSPR'98) and Statistical Techniques in Pattern Recognition '98 (SPR'98)*, pages 31–53, Berlin, 1998. IAPR, Springer-Verlag. http://vision.cse.psu.edu/onlinepubs.html. Pages: 181, 198

[10] M. Antonucci, B. Tirozzi, N.D. Yarunin, and V.S. Dotsenko. Numerical simulation of neural networks with translation and rotation invariant pattern recognition. *International Journal of Modern Physics B*, 8(11-12):1529–1541, 1994. Pages: 26

[11] J. Ashley, R. Barber, M. Flickner, J. Hafner, D. Lee, W. Niblack, and D. Petkovic. Automatic and semi-automatic methods for image annotation and retrieval in QBIC. In W. Niblack and R.C. Jain, editors, *Storage and retrieval for image and video databases III*, volume 2420 of

Proc. of SPIE, pages 24–35, Bellingham, WA, 1995. SPIE, SPIE. Pages: 184

[12] P. Baldi and J. Hornik. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2:53–58, 1989. Pages: 22

[13] H.B. Barlow. Unsupervised learning. *Neural Computation*, 1:295–311, 1989. Pages: 146

[14] J. Basak, B. Chanda, and D. Dutta Majumder. On edge and line linking in graylevel images with connectionist models. *IEEE Transactions on Systems, Man and Cybernetics*, 24:413–428, 1994. Pages: 24

[15] J. Basak and S.K. Pal. A connectionist system for learning and recognition of structures - application to handwritten characters. *Neural Networks*, 8:643–657, 1995. Pages: 26

[16] J. Basak and S.K. Pal. Psycop - a psychologically motivated connectionist system for object perception. *IEEE Transactions on Neural Networks*, 6(6):1337–1354, 1995. Pages: 26, 27

[17] A. Basilevsky. *Statistical factor analysis and related methods*. John Wiley & Sons, New York, NY, 1994. Pages: 130

[18] L. Bedini and A. Tonazzini. Image restoration preserving discontinuities: the Bayesian approach and neural networks. *Image and Vision Computing*, 10(2):108–118, 1992. Pages: 18, 19

[19] R. Beer. *Intelligence as adaptive behavior. An experiment in computational neuroethology*. Academic Press, San Diego, CA, 1990. Pages: 1

[20] A.J. Bell and T.J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995. `http://sloan.salk.edu/~tony/ica.html`. Pages: 144, 225

[21] A.J. Bell and T.J. Sejnowski. Edges are the "independent components" of natural scenes. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, page 831. MIT Press, Cambridge, MA, 1997. `http://sloan.salk.edu/~tony/ica.html`. Pages: 146, 236

[22] A.J. Bell and T.J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997. `http://sloan.salk.edu/~tony/ica.html`. Pages: 145, 146, 236

[23] Y. Bengio. *Neural networks for speech and sequence recognition*. International Thompson Computer Press, Boston, MA, 1996. Pages: 37

[24] Y. Bengio, Y. Le Cun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, space displacement neural networks and hidden Markov models. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, Cambridge, MA, 1994. `http://www.research.att.com/~yann/publis/`. Pages: 42

[25] B. Bhanu. Automatic target recognition: state of the art survey. *IEEE Transactions on Aerospace and Electronic Systems*, 22(4):364–379, 1986. Pages: 51

[26] B. Bhanu, D. Dudgeon, E. Zelnio, A. Rosenfeld, D. Casasent, and I. Reed. Introduction to the special issue on automatic target detection and recognition. *IEEE Transactions on Image Processing*, 6(1):1–3, 1997. Pages: 51

[27] C.M. Bishop. Exact calculation of the Hessian matrix for the multi-layer perceptron. *Neural Computation*, 4(4):494–501, 1992. `http://www.ncrg.aston.ac.uk/`. Pages: 219

[28] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, Oxford, 1995. Pages: 2, 11, 81, 102, 130, 172, 217, 218, 219

[29] C.M. Bishop. Bayesian PCA. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in*

Neural Information Processing Systems 12, pages 382–388. MIT Press, Cambridge, MA, 1999. Pages: 198

[30] C.M. Bishop and J. Winn. Non-linear Bayesian image modelling. In D. Vernon, editor, *Proc. 6th European Conference on Computer Vision (ECCV 2000), Part I*, volume 1842 of *Lecture Notes in Computer Science*, pages 3–17, Berlin, 2000. ECCV, Springer. http://www-sigproc.eng.cam.ac.uk/~jmw39/Papers/Papers.htm. Pages: 198

[31] A.C. Bovik, M. Clark, and W.S. Geisler. Multichannel texture analysis using localized spatial filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):55–73, 1990. Pages: 136

[32] C. Bregler and S.M. Omohundro. Surface learning with applications to lipreading. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 43–50. Morgan Kaufmann, San Mateo, CA, 1994. http://mambo.ucsc.edu/psl/bregler.html. Pages: 129, 130

[33] L. Breiman. Bias, variance, and arcing classifiers. Technical Report Report 460, UC Berkeley, Berkely, CA, 1996. http://www.stat.berkeley.edu/tech-reports/. Pages: 114

[34] P. Brodatz. *Textures, a photographic album for artists and designers*. Dover Publications, New York, NY, 1966. http://www.ux.his.no/~tranden/brodatz.html. Pages: 132

[35] R. Brooks, C. Breazeal, M. Marjanovic, B. Scassellati, and M. Williamson. *Computation for metaphors, analogy, and agents*, volume 1562 of *Lecture notes in artificial intelligence*, chapter The Cog project: building a humanoid robot, pages 52–87. Springer-Verlag, New York, NY, 1998. http://www.ai.mit.edu/people/scaz/pubs.html. Pages: 1

[36] P. Burrascano. A norm selection criterion for the generalized delta rule. *IEEE Transactions on Neural Networks*, 2(1):125–130, 1991. Pages: 115

[37] J.G. Carbonell. *Machine learning: paradigms and methods*. MIT Press, Cambridge, MA, 1990. Pages: 2

[38] J.-F. Cardoso. Infomax and maximum likelihood for blind source separation. *IEEE Letters on Signal Processing*, 4:112–114, 1997. http://www-sig.enst.fr/~cardoso/jfbib.html. Pages: 144

[39] G.A. Carpenter, S. Grossberg, and G.W. Lesher. The what-and-where filter - a spatial mapping neural network for object recognition and image understanding. *Computer Vision and Image Understanding*, 69(1):1–22, 1998. Pages: 26, 27

[40] G.A. Carpenter, S. Grossberg, and D.B. Rosen. ART 2-A: an adaptive resonance algorithm for rapid annote learning and recognition. *Neural Networks*, 4(4):493–504, 1991. Pages: 54

[41] G.A. Carpenter and W.D. Ross. ART-EMAP: a neural network architecture for object recognition by evidence accumulation. *IEEE Transactions on Neural Networks*, 6(4):805–818, 1995. Pages: 26

[42] K.R. Castleman. *Digital image processing*. Prentice-Hall, Englewood Cliffs, NJ, 1979. Pages: 3

[43] J.L. Castro, C.J. Mantas, and J.M. Benítez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, 13(6):561–563, 2000. Pages: 12

[44] H-.P. Chan, S-.C.B. Lo, B. Sahiner, K.L. Lam, and M.A. Helvie. Computer-aided detection of mammographic microcalcifications: Pattern recognition with an artificial neural network. *Medical Physics*, 22(10):1555–1567, 1995. Pages: 26

[45] V. Chandrasekaran, M. Palaniswami, and T.M. Caelli. Range image segmentation by dynamic neural network architecture. *Pattern Recognition*, 29(2):315–329, 1996. Pages: 21

[46] K.S. Cheng, J.S. Lin, and C.W. Mao. The application of competitive Hopfield neural network to medical image segmentation. *IEEE Transactions on Medical Imaging*, 15(4):560–567, 1996. Pages: 24

[47] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995. Pages: 131, 173

[48] D. Chester. Why two hidden layers are better than one. In *Proc. International Joint Conference on Neural Networks*, Los Alamitos, CA, 1990. IEEE, IEEE Press. Pages: 97

[49] G.I. Chiou and J.N. Hwang. A neural network based stochastic active contour model (NNS-SNAKE) for contour finding of distinct features. *IEEE Transactions on Image Processing*, 4(10):1407–1416, 1995. Pages: 24

[50] C. Chong and J. Jia. Assessments of neural network classifier output codings using variability of Hamming distance. *Pattern Recognition Letters*, 17(8):811–818, 1996. Pages: 24

[51] C.K Chow. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, EC-6:247–254, 1957. Pages: 2

[52] S.S. Christensen, A.W. Andersen, T.M. Jorgensen, and C. Liisberg. Visual guidance of a pig evisceration robot using neural networks. *Pattern Recognition Letters*, 17(4):345–355, 1996. Pages: 26, 27

[53] W.J. Christmas, J. Kittler, and M. Petrou. Analytical approaches to the neural net architecture design. In E.S. Gelsema and L.N. Kanal, editors, *Multiple paradigms, comparative studies and hybrid systems*. North-Holland, The Netherlands, 1994. `http://www.ee.surrey.ac.uk/Research/VSSP/publications/1994.html`. Pages: 31

[54] W. Chua and L. Yang. Cellular neural networks: applications. *IEEE Transactions on Circuits and Systems*, 35(10):1273–1290, 1988. Pages: 18

[55] W. Chua and L. Yang. Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272, 1988. Pages: 18

[56] P.C. Chung, C.T. Tsai, E.L. Chen, and Y.N. Sun. Polygonal approximation using a competitive Hopfield neural network. *Pattern Recognition*, 27(11):1505–1512, 1994. Pages: 28, 29

[57] V. Ciesielski, J. Zhu, J. Spicer, and C. Franklin. A comparison of image processing techniques and neural networks for an automated visual inspection problem. In A. Adams and L. Sterling, editors, *Proc. 5th Joint Australian Conference on Artificial Intelligence*, pages 147–152, Singapore, 1992. World Scientific. `http://goanna.cs.rmit.edu.au/~vc/nn.html`. Pages: 63

[58] A.C. Clarke and S. Kubrick. *2001: A space odyssey*. Buccaneer Books, Laguna, CA, 1968. Pages: 1

[59] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 750–755, 1997. `http://www.caip.rutgers.edu/~comanici/`. Pages: 173

[60] D. Comaniciu and P. Meer. Distribution free decomposition of multivariate data. *Pattern Analysis and Applications*, 2(1):22–30, 1999. `http://www.caip.rutgers.edu/~comanici/`. Pages: 173

[61] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *Proc. International Conference on Computer Vision (ICCV'99)*, Corfu, Greece, 1999. `http://www.caip.rutgers.`

edu/~comanici/. Pages: 131, 173

[62] C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995. http://www.research.att.com/info/corinna. Pages: 46

[63] R. Crabbe and A. Dubey. Artificial intelligence FAQ. http://www.faqs.org/faqs/ai-faq/general/part1/. 2000. Pages: 1

[64] J.J. Craig. *Introduction to robotics*. Addison-Wesley, Reading, MA, 1989. Pages: 1

[65] J.M. Cruz, G. Pajares, and J. Aranda. A neural network model in stereovision matching. *Neural Networks*, 8(5):805–813, 1995. Pages: 21, 22

[66] J.M. Cruz, G. Pajares, J. Aranda, and J.L.F. Vindel. Stereo matching technique based on the perceptron criterion function. *Pattern Recognition Letters*, 16(9):933–944, 1995. Pages: 21, 22

[67] C. De Boer and A.W.M. Smeulders. Bessi: an experimentation system for vision module evaluation. In *Proc. 13th IAPR International Conference on Pattern Recognition (ICPR'96)*, volume C, pages 109–113, Los Alamitos, CA, 1996. IAPR, IEEE Computer Society Press. http://carol.wins.uva.nl/~cdeboer/isis/abstracts/bessi_abstract.html. Pages: 31

[68] D. de Ridder. Shared weights neural networks in image analysis. Master's thesis, Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology, march 1996. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8, 42, 43, 63

[69] D. de Ridder, R.P.W. Duin, P.W. Verbeek, and L.J. van Vliet. On the application of neural networks to non-linear image processing tasks. In S. Usui and T. Omori, editors, *Proc. International Conference on Neural Information Processing 1998 (ICONIP'98), Vol. I*, pages 161–165, Tokyo, 1998. JNNS, Ohmsha Ltd. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[70] D. de Ridder, R.P.W. Duin, P.W. Verbeek, and L.J. van Vliet. The applicability of neural networks to non-linear image processing. *Pattern Analysis and Applications*, 2(2):111–128, 1999. Pages: 8

[71] D. de Ridder, R.P.W. Duin, P.W. Verbeek, and L.J. van Vliet. A weight set decorrelating training algorithm for neural network interpretation and symmetry breaking. In B.J. Ersbøll and P. Johansen, editors, *Proc. 11th Scandinavian Conference on Image Analysis (SCIA'99), Vol. 2*, pages 739–746, Copenhagen, Denmark, 1999. DSAGM (The Pattern Recognition Society of Denmark), DSAGM. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8, 82

[72] D. de Ridder, A. Hoekstra, and R. P. W. Duin. Feature extraction in shared weights neural networks. In E.J.H. Kerckhoffs, P.M.A. Sloot, J.F.M. Tonino, and A.M. Vossepoel, editors, *Proc. 2nd Annual Conference of the Advanced School for Computing and Imaging (ASCI'96)*, pages 289–294, Delft, The Netherlands, 1996. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[73] D. de Ridder, J. Kittler, and R.P.W. Duin. Probabilistic PCA and ICA subspace mixture models for image segmentation. In M. Mirmehdi and B. Thomas, editors, *Proc. 11th British Machine Vision Conference (BMVC 2000)*, pages 112–121, Bristol, UK, 2000. BMVA, BMVA. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[74] D. de Ridder, J. Kittler, O. Lemmers, and R.P.W. Duin. The adaptive subspace map for texture segmentation. In A. Sanfeliu, J.J. Villanueva, M. Vanrell, R. Alquézar, J.-O. Eklundh,

and Y. Aloimonos, editors, *Proc. 15th IAPR International Conference on Pattern Recognition (ICPR 2000)*, pages 216–220, Los Alamitos, CA, 2000. IAPR, IEEE Computer Society Press. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[75] D. de Ridder, O. Lemmers, R.P.W. Duin, and J. Kittler. The adaptive subspace map for image description and image database retrieval. In F.J. Ferri, J.M. Iñesta, A. Amin, and P. Pudil, editors, *Proc. IAPR International Workshops on Syntactical and Structural Pattern Recognition (S+SSPR 2000)*, pages 94–103, Berlin, 2000. IAPR, Springer-Verlag. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[76] D. de Ridder, O. Lemmers, R.P.W. Duin, and J. Kittler. Image database retrieval using adaptive non-linear image descriptions. In L.J. van Vliet, J.W.J. Heijnsdijk, T. Kielmann, and P.M.W. Knijnenburg, editors, *Proc. 6th Annual Conference of the Advanced School for Computing and Imaging (ASCI 2000)*, pages 145–152, Delft, The Netherlands, 2000. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[77] D. de Ridder, K. Schutte, and P. Schwering. Detection of vehicles in infrared imagery using shared weights neural networks. In I. Kadar, editor, *Signal processing, sensor fusion and target recognition VIII, Proc. Aerosense '98 (Orlando, FL, April 13-15), Vol. 3374*. SPIE, SPIE, 1998. Pages: 8

[78] D. de Ridder, K. Schutte, and P. Schwering. Vehicle recognition in infrared images using shared weights neural networks. *Optical Engineering*, 37(3):847–857, March 1998. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[79] D. de Ridder, D.M.J. Tax, and R.P.W. Duin. An experimental comparison of one-class classification methods. In B.M. Ter Haar Romeny, D.H.J. Epema, J.F.M. Tonino, and A.A. Wolters, editors, *Proc. 4th Annual Conference of the Advanced School for Computing and Imaging (ASCI'98)*, pages 213–218, Delft, The Netherlands, 1998. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8, 51

[80] J. de Villiers and E. Barnard. Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, 4(1):136–141, 1993. Pages: 12, 97

[81] T.L. Dean, J. Allen, and J. Aloimonos. *Artificial intelligence: Theory and practice*. Benjamin/Cummings, San Francisco, CA, 1994. Pages: 1

[82] A. Delopoulos, A. Tirakis, and S. Kollias. Invariant image classification using triple-correlation-based neural networks. *IEEE Transactions on Neural Networks*, 5(3):392–408, 1994. Pages: 26, 27

[83] D. DeMers and G.W. Cottrell. Non-linear dimensionality reduction. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, San Mateo, CA, 1993. ftp://ftp.cs.ucsd.edu/pub/guest/demers/demers.nips92-nldr.ps.Z. Pages: 22

[84] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977. http://www-eksl.cs.umass.edu/library/library/statistics/expectation-maximization/dempster.pdf. Pages: 130, 131, 223

[85] P.A. Devijver and J. Kittler. *Pattern recognition, a statistical approach*. Prentice-Hall, London, 1982. Pages: 2, 21, 45, 54, 146, 170, 223

[86] W.P. Dewaard. Neural techniques and postal code detection. *Pattern Recognition Letters*, 15(2):199–205, 1994. Pages: 26, 27

[87] J. Dijk, D. de Ridder, P.W. Verbeek, J. Walraven, I.T. Young, and L.J. van Vliet. A new

measure for the effect of sharpening and smoothing filters on images. In B.J. Ersbøll and P. Johansen, editors, *Proc. 11th Scandinavian Conference on Image Analysis (SCIA'99), Vol. 1*, pages 213–220, Copenhagen, Denmark, 1999. DSAGM (The Pattern Recognition Society of Denmark), DSAGM. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8, 114

[88] J. Dijk, D. de Ridder, P.W. Verbeek, J. Walraven, I.T. Young, and L.J. van Vliet. A quantitative measure for the perception of sharpening and smoothing in images. In M. Boasson, J.A. Kaandorp, J.F.M. Tonino, and M.G. Vosselman, editors, *Proc. 5th Annual Conference of the Advanced School for Computing and Imaging (ASCI'99)*, pages 291–298, Delft, The Netherlands, 1999. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8, 114

[89] E. do Valle Simões, L.F. Uebel, and D.A.C. Barone. Hardware implementation of RAM neural networks. *Pattern Recognition Letters*, 17(4):421–429, 1996. http://eleceng.ukc.ac.uk/~edvs1/web/articles/smc97.ps.zip. Pages: 26, 27

[90] R.D. Dony and S. Haykin. Neural network approaches to image compression. *Proceedings of the IEEE*, 83(2):288–303, 1995. Pages: 16

[91] J.M.H. Du Buf, M. Kardan, and M. Spann. Texture feature performance for image segmentation. *Pattern Recognition*, 23(3/4):291–309, 1990. Pages: 25

[92] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis*. John Wiley & Sons, New York, NY, 1973. Pages: 2, 170

[93] R.P.W. Duin and D. de Ridder. Neural network experiences between perceptrons and support vectors. In A.F. Clark, editor, *Proc. 8th British Machine Vision Conference (BMVC'97)*, pages 590–599. BMVA, September 1997. http://www.ph.tn.tudelft.nl/People/bob/papers/bmvc_97_perceptron.ps. Pages: 8

[94] M. Egmont-Petersen and T. Arts. Recognition of radiopaque markers in X-ray images using a neural network as nonlinear filter. *Pattern Recognition Letters*, 20 (5):521–533, 1999. http://www.cs.uu.nl/people/michael/Journal-papers/Egmont-prl-1999a.pdf. Pages: 26, 28

[95] M. Egmont-Petersen, W.R.M. Dassen, C.J.H.J. Kirchhof, J. Heijmeriks, and A.W. Ambergen. An explanation facility for a neural network trained to predict arterial fibrillation directly after cardiac surgery. In *Computers in Cardiology 1998*, pages 489–492, Cleveland, 1998. IEEE. http://www.cs.uu.nl/people/michael/Abstract-P-CIC98.htm. Pages: 28

[96] M. Egmont-Petersen, D. De Ridder, and H. Handels. Image processing using neural networks – a review. *Pattern Recognition*. http://www.cs.uu.nl/people/michael/index.html. To appear. Pages: 8, 16, 17

[97] M. Egmont-Petersen and E Pelikan. Detection of bone tumours in radiographic images using neural networks. *Pattern Analysis and Applications*, 2(2):172–183, 1999. http://www.cs.uu.nl/people/michael/Journal-papers/Egmont-Paa-1999.pdf. Pages: 25

[98] M. Egmont-Petersen, U. Schreiner, S.C. Tromp, T.M. Lehmann, D.W. Slaaf, and T. Arts. Detection of leukocytes in contact with the vessel wall from in vivo microscope recordings using a neural network. *IEEE Transactions on Biomedical Engineering*, 47 (7):941–951, 2000. http://www.cs.uu.nl/people/michael/Journal-papers/Egmont-BME-2000.pdf. Pages: 26

[99] M. Egmont-Petersen, J.L. Talmon, A. Hasman, and A.W. Ambergen. Assessing the importance of features for multi-layer perceptrons. *Neural Networks*, 11 (4):623–635, 1998. http://www.cs.uu.nl/people/michael/Journal-papers/Egmont-Petersen-NN-1998.pdf. Pages: 30, 63

[100] B.S Everitt. *An introduction to latent variable methods*. Chapman and Hall, London, UK, 1984. Pages: 130

[101] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, Los Altos, CA, 1990. http://www-2.cs.cmu.edu/afs/cs/project/connect/tr/cascor-tr.ps.Z. Pages: 81

[102] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3 (3-4):231–262, 1994. http://www.cs.cmu.edu/~christos/PUBLICATIONS.OLDER/qbic.ps.gz. Pages: 184

[103] N. Fatemi-Ghomi. *Performance measures for wavelet-based segmentation algorithms*. PhD thesis, University of Surrey, Guildford, Surrey, United Kingdom, September 1997. ftp://ftp.ee.surrey.ac.uk/pub/vision/papers/ghomi-phd97.ps.Z. Pages: 175

[104] D.J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America A*, 4(12):2370–2393, 1987. Pages: 146

[105] M.A.T. Figueiredo and J.M.N. Leitao. Sequential and parallel image restoration: Neural network implementations. *IEEE Transactions on Image Processing*, 3(6):789–801, 1994. Pages: 18, 19

[106] E. Fix and J.L. Hodges Jr. Nonparametric discrimination: small sample performance. Technical Report Project number 21-49-004, Report number 11, USAF School of Aviation Medicine, Randolph Air Force Base, Texax, August 1951. Pages: 2

[107] M.D. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, and D. Lee. Query by image and video content: the QBIC system. *IEEE Computer*, 28(9):23–31, 1995. Pages: 184

[108] D.B. Fogel. An information criterion for optimal neural network selection. *IEEE Transactions on Neural Networks*, 2(5):490–497, 1991. Pages: 32

[109] F. Fogelman Soulie, E. Viennet, and B. Lamy. Multi-modular neural network architectures: applications in optical character and human face recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):721–755, 1993. Pages: 26, 37, 41, 42, 47, 209

[110] J.H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987. Pages: 144

[111] J.H. Friedman and J.W Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9):881–890, 1974. Pages: 144

[112] M. Fukumi, S. Omatu, and Y. Nishikawa. Rotation-invariant neural pattern recognition system estimating a rotation angle. *IEEE Transactions on Neural Networks*, 8(3):568–581, 1997. Pages: 21, 22, 26

[113] M. Fukumi, S. Omatu, F. Takeda, and T. Kosaka. Rotation-invariant neural pattern-recognition system with application to coin recognition. *IEEE Transactions on Neural Networks*, 3(2):272–279, 1992. Pages: 21, 22, 26

[114] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, New York, NY, 2^{nd} edition, 1990. Pages: 21, 45, 54

[115] K. Fukunaga and D.R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 20(2):176–183, 1971. Pages: 129

[116] K. Fukushima. Neocognitron: a hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988. Pages: 2, 26

[117] K. Fukushima and S. Miyake. Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982. Pages: 2, 42

[118] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: a neural model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):826–834, 1983. Pages: 2, 26, 31

[119] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989. Pages: 5, 12, 97

[120] P.D. Gader, J.R. Miramonti, Y. Won, and P. Coffield. Segmentation free shared weight networks for automatic vehicle detection. *Neural Networks*, 8(9):1457–1473, 1995. Pages: 26, 42, 51, 60

[121] M.D. Garris, C.L. Wilson, and J.L. Blue. Neural network-based systems for handprint OCR applications. *IEEE Transactions on Image Processing*, 7(8):1097–1112, 1998. `ftp://sequoyah.nist.gov/pub/papers_preprints/nnbsh_ocr.ps.Z`. Pages: 26

[122] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias-variance dilemma. *Neural Computation*, 4(1):1–58, 1992. Pages: 12

[123] Z. Ghahramani and M. J. Beal. Variational inference for Bayesian mixtures of factor analysers. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 449–455. MIT Press, Cambridge, MA, 1999. `http://www.gatsby.ucl.ac.uk/publications/papers/06-2000.pdf`. Pages: 198

[124] A. Ghosh. Use of fuzziness measures in layered networks for object extraction: a generalization. *Fuzzy Sets and Systems*, 72:331–348, 1995. Pages: 24

[125] A. Ghosh, N.R. Pal, and S.K. Pal. Image segmentation using a neural network. *Biological Cybernetics*, 66:151–158, 1991. Pages: 24

[126] A. Ghosh, N.R. Pal, and S.K. Pal. Object background classification using Hopfield type neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 6:989–1008, 1992. Pages: 24

[127] A. Ghosh, N.R. Pal, and S.K. Pal. Self-organization for object extraction using a multilayer neural network and fuzziness measures. *IEEE Transactions on Fuzzy Systems*, 1(1):54–68, 1993. Pages: 24

[128] A. Ghosh and S.K. Pal. Neural network, self-organization and object extraction. *Pattern Recognition Letters*, 13(5):387–397, 1992. Pages: 24

[129] J.O. Glass and W.E. Reddick. Hybrid artificial neural network segmentation and classification of dynamic contrast-enhanced MR imaging (DEMRI) of osteosarcoma. *Magnetic Resonance Imaging*, 16(9):1075–1083, 1998. Pages: 21, 22, 24

[130] C. Goerick, D. Noll, and M. Werner. Artificial neural networks in real-time car detection and tracking applications. *Pattern Recognition Letters*, 17(4):335–343, 1996. `ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/manuscripts/articles/prl96.ps.gz`. Pages: 26

[131] R.C. Gonzalez and R.E. Woods. *Digital image processing*. Addison-Wesley, Reading, MA, 1992. Pages: 3

[132] L.S. Goodenday, K.J. Cios, and I. Shin. Identifying coronary stenosis using an image-recognition neural network. *IEEE Engineering in Medicine and Biology*, 16(5):139–144, 1997. Pages: 22, 26

[133] R.P. Gorman and T.J. Sejnowski. Analysis of the hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1(1):75–89, 1988. Pages: 63

[134] C.D. Green. Are connectionist models theories of cognition? *Psycoloquy*, 9(4), 1998. `http://www.cogsci.soton.ac.uk/cgi/psyc/newpsy?9.04`. Pages: 92

[135] D. Greenhil and E.R. Davies. Relative effectiveness of neural networks for image noise suppression. In E.S. Gelsema and L.N. Kanal, editors, *Pattern Recognition in Practice IV*, pages 367–378, Vlieland, 1994. North-Holland. Pages: 18, 42

[136] M. Groß and F. Seibert. Visualization of multidimensional data sets using a neural network. *The Visual Computer*, 10(3):145–159, 1993. Pages: 24

[137] S. Grossberg and N.P. Mcloughlin. Cortical dynamics of three-dimensional surface perception - binocular and half-occluded scenic images. *Neural Networks*, 10(9):1583–1605, 1997. Pages: 24

[138] S. Grossberg and E. Mingolla. Neural dynamics of surface perception: boundary webs, illuminants, and shape from shading. *Computer Vision, Graphics, and Image Processing*, 37 (1):116–165, 1987. Pages: 24

[139] L. Guan, J.A. Anderson, and J.P. Sutton. A network of networks processing model for image regularization. *IEEE Transactions on Neural Networks*, 8(1):169–174, 1997. Pages: 18, 19

[140] V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990. Pages: 114

[141] L.O. Hall, A.M. Bensaid, L.P. Clarke, R.P. Velthuizen, M.S. Sibiger, and J.C. Bezdek. A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain. *IEEE Transactions on Neural Networks*, 3(5):672–682, 1992. Pages: 24

[142] E.R. Hancock and J. Kittler. A Bayesian interpretation for the Hopfield network. In *Proc. 13th IEEE Conference on Neural Networks, Vol. I*, pages 341–346, Los Alamitos, CA, 1993. IEEE, IEEE Computer Society Press. Pages: 29

[143] H. Hanek and N. Ansari. Speeding up the generalized adaptive neural filters. *IEEE Transactions on Image Processing*, 5(5):705–712, 1996. `http://www-ec.njit.edu/~ang/papers/IP96.pdf`. Pages: 18, 19

[144] R.M. Haralick. Performance characterization in computer vision. *Computer Vision, Graphics and Image Processing: Image understanding*, 60(2):245–249, 1994. Pages: 31

[145] T. Hastie. *Principal curves and surfaces*. PhD thesis, Stanford University, 1984. Pages: 128

[146] S. Haykin. *Neural networks: a comprehensive foundation*. Macmillan College Publishing Co., New York, NY, 1994. Pages: 2, 9, 15

[147] S. Haykin, P. Yee, and E. Derbez. Optimum nonlinear filtering. *IEEE Transactions on Signal Processing*, 45(11):2774–2786, 1997. `http://citeseer.nj.nec.com/haykin97optimum.html`. Pages: 9, 11, 27, 41

[148] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, Reading, MA, 1991. Pages: 2, 9, 11, 15, 16, 27, 41, 48, 54, 68, 115

[149] F. S. Hiller and G. J. Lieberman. *Introduction to operations research*. McGraw-Hill, New York, NY, 6th edition, 1995. Pages: 29

[150] G. E. Hinton and T.J. Sejnowski. Learning and relearning in Boltzmann machines. In Rumelhart, D.E. and McClelland, J.L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I. MIT Press, Cambridge, MA, 1986. Pages: 2

[151] G.E. Hinton, P. Dayan, and M. Revow. Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997. http://www.gatsby.ucl.ac.uk/Hinton/absps/manifold.pdf. Pages: 22, 31, 128, 129, 130, 140, 141, 194

[152] G.E. Hinton and T.J. Sejnowski. Optimal perceptual inference. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'83)*, pages 448–453, New York, NY, 1983. IEEE, IEEE. Pages: 2

[153] G.E. Hinton, T.J. Sejnowski, and D.H. Ackley. Boltzmann machines: constraint satisfaction networks that learn. Technical Report Report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh, PA, 1984. Pages: 70

[154] A. Hoekstra. *Generalisation in feed-forward neural classifiers*. PhD thesis, Delft University of Technology, Delft, 1998. http://www.ph.tn.tudelft.nl/PHDTheses/AHoekstra/thesis_hoekstra.html. Pages: 43

[155] A. Hoekstra, M.A. Kraaijveld, D. de Ridder, and W.F. Schmidt. *The Complete SPRLIB & ANNLIB*. Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology, 1996. Pages: 55, 71, 81, 96

[156] A. Hoekstra, H. Netten, and D. de Ridder. A neural network applied to spot counting. In E.J.H. Kerckhoffs, Sloot. P.M.A., J.F.M. Tonino, and A.M. Vossepoel, editors, *Proc. 2nd Annual Conference of the Advanced School for Computing and Imaging (ASCI'96)*, pages 224–229, Delft, The Netherlands, 1996. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[157] A. Hoekstra, H. Netten, and D. de Ridder. Cell nuclei analysis using neural networks. In *Neural Networks: Best Practice in Europe, Proc. SNN'97, Progress in Neural Processing Vol. 8*, pages 165–168, Singapore, 1997. SNN, World Scientific. Pages: 8

[158] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.*, 81:3088–3092, 1982. Pages: 2, 14

[159] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, 1985. Pages: 2, 9, 15

[160] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. Pages: 5, 12, 92, 97

[161] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5): 551–560, 1990. Pages: 5

[162] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933. Pages: 138

[163] Q. Huang and B. Dom. Quantitative methods of evaluating image segmentation. In *Proc. International Conference on Image Processing (ICIP'95)*, volume 3, pages 53–56. IEEE Computer Society Press, Los Alamitos, CA, 1995. Pages: 25

[164] D.H. Hubel and T.N. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962. Pages: 37,

146

[165] J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994. http://www.cedar.buffalo.edu/Databases/CDROM1/. Pages: 194

[166] J. Hurri. Independent component analysis of image data. Master's thesis, Dept. of Computer Science and Engineering, Helsinki University of Technology, Espoo, Finland, March 1997. http://www.cis.hut.fi/jarmo/. Pages: 134, 146, 236

[167] J. Hurri, A. Hyvärinen, and E. Oja. Image feature extraction using independent component analysis. In *Proc. IEEE Nordic Signal Processing Symposium*, Espoo, Finland, 1996. NORSIG. http://www.cis.hut.fi/~aapo/. Pages: 146, 236

[168] R.E. Hurst, R.B. Bonner, K. Ashenayi, R.W. Veltri, and G.P. Hemstreet. Neural net-based identification of cells expressing the p300 tumor-related antigen using fluorescence image analysis. *Cytometry*, 27(1):36–42, 1997. Pages: 26

[169] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993. Pages: 182

[170] A. Hyvärinen. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997. http://www.cis.hut.fi/~aapo/. Pages: 236

[171] A. Hyvärinen. Independent component analysis by minimization of mutual information. Technical Report A46, Laboratory of Computer and Information Science, Dept. of Computer Science and Engineering, Helsinki University of Technology, Rakentajanaukio 2 C, FIN-02150 Espoo, Finland, August 1997. http://www.cis.hut.fi/~aapo/. Pages: 144

[172] A. Hyvärinen. Independent component analysis in the presence of noise: a maximum likelihood approach. In *Proc. International ICSC Workshop on Independence and Artificial Neural Networks (I&ANN'98)*, pages 32–38, Tenerife, Spain, 1998. http://www.cis.hut.fi/~aapo/. Pages: 131

[173] A. Hyvärinen. New approximations of differential entropy for independent component analysis and projection pursuit. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*. MIT Press, Cambridge, MA, 1998. http://www.cis.hut.fi/~aapo/. Pages: 144

[174] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999. http://www.cis.hut.fi/~aapo/. Pages: 236

[175] A. Hyvärinen. The fixed-point algorithm and maximum likelihood estimation for independent component analysis. *Neural Processing Letters*, 10:1–5, 1999. http://www.cis.hut.fi/~aapo/. Pages: 131, 144

[176] A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 1 (2):94–128, 1999. http://www.cis.hut.fi/~aapo/. Pages: 143

[177] A. Hyvärinen, J. Särelä, and R. Vigário. Spikes and bumps: artefacts generated by independent component analysis with insufficient sample size. In *Proc. First International Workshop on Independent Component Analysis (ICA'99)*, pages 425–429, Aussois, France, Jan. 11-15 1999. http://www.cis.hut.fi/~aapo/. Pages: 235

[178] F. Idris and S. Panchanathan. Image indexing using vector quantization. In W. Niblack and R.C. Jain, editors, *Storage and retrieval for image and video databases III*, volume 2420 of

Proc. of SPIE, Bellingham, WA, 1995. SPIE, SPIE. Pages: 190

[179] The Mathworks Inc. MATLAB release 11.1. http://www.mathworks.com/. 2000. Pages: 35, 118

[180] R.A. Jacobs, M.I. Jordan, and A.G. Barto. Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks. *Cognitive Science*, 15:219–250, 1991. ftp://ftp.gmd.de/documents/neuroprose/jacobs.ps.gz. Pages: 80

[181] B. Jähne. *Digital image processing. Concepts, algorithms and scientific applications*. Springer-Verlag, Berlin, 1995. Pages: 27

[182] A.K. Jain and F. Farrokhnia. Unsupervised texture segmentation using Gabor filters. *Pattern Recognition*, 24(12):1167–1186, 1991. Pages: 137

[183] A.K. Jain and K. Karu. Automatic filter design for texture discrimination. In *Proc. 12th IAPR International Conference on Pattern Recognition (ICPR'94)*, volume 1, pages 454–458, Piscataway, NJ, 1994. IEEE. Pages: 24

[184] A.K. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997. Pages: 30

[185] B. Javidi and Q. Tang. Optical implementation of neural networks by the use of nonlinear joint transform correlators. *Applied Optics*, 34(20):3950–3962, 1995. Pages: 26

[186] M.C. Jones and R. Sibson. What is projection pursuit? *Journal of the Royal Statistical Society A*, 150:1–36, 1987. Pages: 144

[187] N. Kambhatla and T.K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, 1997. ftp://cse.ogi.edu/pub/neural/papers/kambhLeen95.DimRed.ps.Z. Pages: 129, 130

[188] N.K. Kasabov, S.I. Israel, and B.J. Woodford. Adaptive, evolving, hybrid connectionist systems for image pattern recognition. In S.K. Pal, A. Ghosh, and M.K. Kundu, editors, *Soft computing for image processing*. Physica-Verlag, Heidelberg, 2000. http://kel.otago.ac.nz/moorloch/scfip99.zip. Pages: 26

[189] H. Katsulai and N. Arimizu. Evaluation of image fidelity by means of the fidelogram and level mean-square error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3 (3):337–347, 1981. Pages: 110

[190] P.M. Kelly and M. Cannon. Query by image example: the CANDID approach. In W. Niblack and R.C. Jain, editors, *Storage and retrieval for image and video databases III*, volume 2420 of *Proc. of SPIE*, pages 239–248, Bellingham, WA, 1995. SPIE, SPIE. http://www.c3.lanl.gov/~kelly/pubs/aipr.94/aipr.94.ps.gz. Pages: 181

[191] V.Z. Këpuska and S.O. Mason. A hierarchical neural network system for signalized point recognition in aerial photographs. *Photogrammetric Engineering & Remote Sensing*, 61(7): 917–925, 1995. Pages: 21, 22, 26

[192] G.J. Klir and T.A. Folger. *Fuzzy sets, uncertainty, and information*. Prentice Hall, Englewood Cliffs, NJ, 1988. Pages: 2

[193] J. Koh, M.S. Suk, and S.M. Bhandarkar. A multilayer self organizing feature map for range image segmentation. *Neural Networks*, 8(1):67–86, 1995. Pages: 24

[194] T. Kohonen. Clustering, taxonomy and topological maps of patterns. In *Proc. 6th IAPR International Conference on Pattern Recognition (ICPR'82)*, pages 114–128, München, 1982. IAPR, IEEE Computer Society Press. Pages: 13, 14

[195] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982. Pages: 2, 9, 13

[196] T. Kohonen. *Self-organizing maps*. Springer Series in Information Sciences. Springer-Verlag, Berlin, 1995. Pages: 2, 13, 14, 134

[197] T. Kohonen, S. Kaski, and H. Lappalainen. Self-organized formation of various invariant-feature filters in the Adaptive-Subspace SOM. *Neural Computation*, 9(6):1321–1344, 1997. `http://www.cis.hut.fi/~sami/publications.html`. Pages: 129, 130, 135

[198] E. Körber. Self organising maps in image classification. Master's thesis, Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology, december 1997. Pages: 8

[199] B. Kosko. Adaptive bidirectional associative memories. *Applied Optics*, 26(23):4947–4960, 1987. Pages: 26

[200] M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *American Institute of Chemical Engineers Journal*, 37:223–243, 1991. Pages: 22

[201] M. Kuwahara, K. Hachimura, S. Eiho, and M. Kinoshita. *Digital processing of biomedical images*, pages 187–203. Plenum Press, New York, NY, 1976. Pages: 89, 90

[202] J. Lampinen and E. Oja. Distortion tolerant pattern recognition based on self-organizing feature extraction. *IEEE Transactions on Neural Networks*, 6(3):539–547, 1995. `http://www.lce.hut.fi/~jlampine/jl_bib.html`. Pages: 21, 22, 23, 26, 190

[203] J. Lampinen and S. Smolander. Self-organizing feature extraction in recognition of wood surface defects and color images. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(2):97–113, 1996. `http://www.lce.hut.fi/~jlampine/jl_bib.html`. Pages: 21, 22, 23, 26

[204] C.G. Langton. *Artificial Life (Proceedings of the First International Conference '87)*. Addison-Wesley, Reading, MA, 1989. Pages: 1

[205] S. Lawrence, C.L. Giles, A.C. Tsoi, and A.D. Back. Face recognition - a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997. `http://www.neci.nj.nec.com/homepages/lawrence/papers.html`. Pages: 42

[206] D.X. Le, G.R. Thoma, and H. Wechsler. Classification of binary document images into textual or nontextual data blocks using neural network models. *Machine Vision and Applications*, 8(5):289–304, 1995. `http://archive.nlm.nih.gov/pubs/doc_class/mv.html`. Pages: 24

[207] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551, 1989. `http://www.research.att.com/~yann/publis/`. Pages: 25, 26, 31, 37, 38, 40, 41, 45, 47, 209

[208] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, 1990. `http://www.research.att.com/~yann/publis/`. Pages: 26, 37, 38, 41, 47, 72, 209

[209] Y. Le Cun, L. J. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communication*, 27(11):41–46, november 1989. `http://www.research.att.com/~yann/publis/`. Pages: 26, 37, 38, 40, 47, 72, 209

[210] C.C. Lee and J.P. Degyvez. Color image processing in a cellular neural-network environment. *IEEE Transactions on Neural Networks*, 7(5):1086–1098, 1996. Pages: 18

[211] D. Lee, R. Barber, W. Niblack, M. Flickner, J. Hafner, and D. Petkovic. Indexing for complex queries on a query-by-content image database. In *Proc. 12th International Conference on Pattern Recognition (ICPR'94)*, pages 142–146, Jerusalem, Israel, 1994. Pages: 128, 131, 146, 147

[212] T.-W. Lee, M. Girolami, and T.J. Sejnowski. Independent component analysis using an extended infomax algorithm for mixed sub-Gaussian and super-Gaussian sources. *Neural Computation*, 11(2):417–441, 1999. `http://www.cnl.salk.edu/~tewon/pubs.html`. Pages: 131, 144, 147, 221, 225, 228, 229

[213] T.-W. Lee, M.S. Lewicki, and T.J. Sejnowski. ICA mixture models for unsupervised classification and automatic context switching. In *Proc. First International Workshop on Independent Component Analysis (ICA'99)*, pages 209–214, Aussois, France, Jan. 11-15 1999. `http://www.cnl.salk.edu/~tewon/pubs.html`. Pages: 128, 131

[214] T.-W. Lee, M.S. Lewicki, and T.J. Sejnowski. Unsupervised classification with non-Gaussian mixture models using ICA. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, Cambridge, MA, 1999. `http://www.cnl.salk.edu/~tewon/pubs.html`. Pages: 128, 131, 146, 147, 236

[215] T.-W. Lee, M.S. Lewicki, and T.J. Sejnowski. ICA mixture models for unsupervised classification and automatic context switching in blind signal separation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(16):1078–1089, 2000. `http://www.cnl.salk.edu/~tewon/pubs.html`. Pages: 236

[216] O. Lemmers. Invariant-feature extraction using adaptive linear subspaces. Master's thesis, Pattern Recognition Group, Dept. of Applied Physics, Delft University of Technology, may 1999. Pages: 8, 190

[217] B. Levienaise-Obadia, J. Kittler, and W. Christmas. Comparative study of strategies for illumination-invariant texture representations. In M.M. Yeung, B. Yeo, and C.A. Bouman, editors, *Storage and retrieval for image and video databases VII*, volume 3656 of *Proc. of SPIE*, Bellingham, WA, 1999. SPIE, SPIE. Pages: 137

[218] S. Levy. *Artificial life*. Pantheon, New York, NY, 1992. Pages: 1

[219] M.S. Lewicki and T.J. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12(2):337–365, 2000. `http://www.cs.cmu.edu/~lewicki/`. Pages: 131

[220] J.S. Lin, S.C.B. Lo, A. Hasegawa, M.T. Freedman, and S.K. Mun. Reduction of false positives in lung nodule detection using a two-level neural classification. *IEEE Transactions on Medical Imaging*, 15(2):206–217, 1996. Pages: 26

[221] W.-C. Lin, E. C.-K. Tsao, and C.-T. Chen. Constraint satisfaction neural networks for image segmentation. *Pattern Recognition*, 25(7):679–693, 1992. Pages: 24

[222] T. Lindeberg. *Scale-space theory in computer vision*. Kluwer Academic, Dordrecht, 1994. Pages: 27

[223] S.C.B. Lo, H.P. Chan, J.S. Lin, H. Li, M.T. Freedman, and S.K. Mun. Artificial convolution neural network for medical image pattern recognition. *Neural Networks*, 8(7-8):1201–1214, 1995. Pages: 26

[224] H. Lu, Y. Fainman, and R. Hecht-Nielsen. Image manifolds. In N.M. Nasrabadi and A.K. Katsaggelos, editors, *Applications of Artificial Neural Networks in Image Processing III, Proceedings of SPIE*, volume 3307 of *SPIE Proceedings Series*, pages 52–63, Bellingham, WA,

1998. SPIE, SPIE. http://citeseer.nj.nec.com/lu98image.html. Pages: 128

[225] S. Lu and A. Szeto. Hierarchical artificial neural networks for edge enhancement. *Pattern Recognition*, 26(8):1149–1163, 1993. Pages: 21

[226] S.W. Lu and H. Xu. Textured image segmentation using autoregressive model and artificial neural network. *Pattern Recognition*, 28(12):1807–1817, 1995. Pages: 24

[227] D. MacKay. Maximum likelihood and covariant algorithms for independent component analysis. http://wol.ra.phy.cam.ac.uk/mackay/. Draft 3.7, 1996. Pages: 131, 221, 222, 224, 225

[228] P. Maes. *Designing autonomous agents: theory and practice from biology to engineering and back*. MIT Press, Cambridge, MA, 1991. Pages: 1

[229] E.C. Malthouse. Some theoretical results on nonlinear principal component analysis. Technical report, Integrated Marketing Communications, Medill School of Journalism, Kellogg Graduate School of Management, Northwestern University, 1908 Sheridan Road, Evanston, IL 60208-1290, September 1996. ftp://ftp.cis.ohio-state.edu/pub/neuroprose/malthouse.nlpca.ps.Z. Pages: 128

[230] B.S. Manjunath, T. Simchony, and R. Chellappa. Stochastic and deterministic networks for texture segmentation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38:1039–1049, 1990. http://www-iplab.ece.ucsb.edu/publications/90ASSPTrans.pdf. Pages: 24

[231] B.F.J. Manly. *Multivariate statistical methods*. Chapman & Hall, London, 2 edition, 1994. Pages: 130, 138

[232] J.A. Marshall. Self-organizing neural networks for perception of visual motion. *Neural Networks*, 3:45–74, 1990. Pages: 24

[233] O. Matan, R. K. Kiang, C. E. Stenard, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and Y. Le Cun. Handwritten character recognition using neural network architectures. In *Proc. 4th USPS Advanced Technology Conference*, pages 1003–1011, Washington D.C., 1990. U.S. Postal Services. http://www.research.att.com/~yann/publis/. Pages: 209

[234] T. Matsumoto, H. Kobayashi, and Y. Togawa. Spatial versus temporal stability issues in image processing neuro chips. *IEEE Transactions on Neural Networks*, 3(4):540–569, 1992. Pages: 18, 19

[235] W.S. McCulloch and W. Pitts. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. Pages: 2

[236] M.R.J. McQuoid. Neural ensembles: Simultaneous recognition of multiple 2-D visual objects. *Neural Networks*, 6(7):970–917, 1993. Pages: 26

[237] P. Meinicke and H. Ritter. Local PCA learning with resolution-dependent mixtures of Gaussians. In *Proc. International Conference on Artificial Neural Networks (ICANN'99)*, pages 497–502, Edinburgh, UK, 1999. http://www.techfak.uni-bielefeld.de/techfak/ags/ni/publications/papers/MeinickeRitter1999-LPL.ps.gz. Pages: 129, 131

[238] O. Melnik and J.B. Pollack. Exact representations from feed-forward networks. Technical Report CS-99-205, Dept. of Computer Science, Volen National Center for Complex Systems, Brandeis University, Waltham, MA, 1998. http://www.demo.cs.brandeis.edu/papers/nc1.ps.gz. Pages: 63

[239] K. Messer. *Automatic image database retrieval system using adaptive colour and texture*

descriptors. PhD thesis, University of Surrey, Guildford, Surrey, United Kingdom, 1999. Pages: 186

[240] K. Messer, D. de Ridder, and J. Kittler. Adaptive texture representation for automatic target recognition. In T. Pridmore and D. Elliman, editors, *Proc. 10th British Machine Vision Conference (BMVC'99)*, pages 443–452, Nottingham, UK, 1999. BMVA, BMVA. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 8

[241] K. Messer and J. Kittler. A region-based image database system using colour and texture. *Pattern Recognition Letters*, 20(11-13):1323–1330, 1999. Pages: 186

[242] R.R. Meyer and E. Heindl. Reconstruction of off-axis electron holograms using a neural net. *Journal of Microscopy*, 191(1):52–59, 1998. Pages: 18

[243] M. L. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969. Pages: 2, 37

[244] T. Mitchell. *Machine learning*. McGraw-Hill, New York, NY, 1997. Pages: 2

[245] B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):696–710, 1997. ftp://whitechapel.media.mit.edu/pub/OUP/chapter.ps.Z. Pages: 140, 141

[246] J. Moh and F.Y. Shih. A general purpose model for image operations based on multilayer perceptrons. *Pattern Recognition*, 28(7):1083–1090, 1995. Pages: 21

[247] R.J.T. Morris, L.D. Rubin, and H. Tirri. Neural network techniques for object orientation detection: solution by optimal feedforward network and learning vector quantization approaches. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11):1107–1115, 1990. Pages: 22

[248] E. Moulines, J.-F. Cardoso, and E. Gassiat. Maximum likelihood for blind separation and deconvolution of noisy signals using mixture models. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP'97)*, pages 3617–3620. ICASSP, 1997. http://www-sig.enst.fr/~cardoso/jfbib.html. Pages: 131

[249] M.R. Moya and D.R. Hush. Network constraints and multi-objective optimization for one-class classification. *Neural Networks*, 9(3):463–474, 1996. Pages: 54

[250] M.R. Moya, M.W. Koch, and L.D. Hostetler. One-class classifier networks for target recognition applications. In *Proc. World Congress on Neural Networks*, pages 797–801, Portland, OR, 1993. International Neural Network Society, INNS. Pages: 54

[251] N. Murata, S. Yoshizawa, and S. Amari. Network information criterion - determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, 5(6):865–872, 1994. http://www.islab.brain.riken.go.jp/~mura/paper/mura94_nic.ps.gz. Pages: 32

[252] M. Nagao and T. Matsuyama. *A structural analysis of complex aerial photographs*. Advanced applications in pattern recognition. Plenum Press, New York, NY, 1980. Pages: 90

[253] N.M. Nasrabadi and C.Y. Choo. Hopfield network for stereo vision correspondence. *IEEE Transactions on Neural Networks*, 3(1):5–13, 1992. Pages: 28, 29

[254] C. Neubauer. Evaluation of convolutional neural networks for visual recognition. *IEEE Transactions on Neural Networks*, 9(4):685–696, 1998. Pages: 26

[255] S.C. Ngan and X. Hu. Analysis of functional magnetic resonance imaging data using self-organizing mapping with spatial connectivity. *Magnetic Resonance in Medicine*, 41(5): 939–946, 1999. Pages: 24

[256] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: querying images by content using color, tex-

ture and shape. In W. Niblack, editor, *Storage and retrieval for image and video databases*, volume 1908 of *Proc. of SPIE*, pages 173–181, Bellingham, WA, 1993. SPIE, SPIE. http://wwwqbic.almaden.ibm.com. Pages: 184

[257] N.J. Nilsson. *Learning machines*. McGraw-Hill, New York, NY, 1965. Pages: 2

[258] J.A. Nossek and T. Roska. Special issue on cellular neural networks. *IEEE Transactions on Circuits and Systems*, 40(3), 1993. Pages: 18

[259] S. Nowlan and J. Platt. A convolutional neural network hand tracker. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 901–908. MIT Press, Cambridge, MA, 1995. http://research.microsoft.com/~jplatt/hands.ps. Pages: 42

[260] C. Obellianne, F. Fogelman Soulie, and G. Galibourg. Connectionist models for image processing. In Simon, J.C., editor, *From pixels to features, a workshop held at Bonas, France 22-27 august 1988*, pages 185–196, Amsterdam, 1988. North-Holland. Pages: 42

[261] E. Oja. Data compression, feature extraction, and autoassociation in feed-forward neural networks. In O. Simula, T. Kohonen, K. Makisara, and J. Kangas, editors, *International Conference on Artificial Neural Networks*, pages 737–745, Helsinki, Finland, 1991. Elsevier, Amsterdam. Pages: 22

[262] E. Oja, J. Karhunen, A. Hÿvarinen, R. Vigário, and J. Hurri. Neural independent component analysis - approaches and applications. In S. Amari and N. Kasabov, editors, *Brain-like computing and intelligent information systems*, chapter 8, pages 167–188. Springer-Verlag, Singapore, 1998. Pages: 236

[263] B.A. Olshausen and D.J. Field. Emergence of simple-cell receptive field properties by learning sparse codes for natural images. *Nature*, 381:607–609, 1996. ftp://redwood.psych.cornell.edu/pub/papers/sparse-coding.ps.Z. Pages: 146

[264] B.A. Olshausen and D.J. Field. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research*, 37:3311–3325, 1997. ftp://redwood.psych.cornell.edu/pub/papers/VR.ps.Z. Pages: 146

[265] R. Opara and F. Worgotter. Using visual latencies to improve image segmentation. *Neural Computation*, 8(7):1493–1520, 1996. Pages: 24

[266] M. Ozkan, B.M. Dawant, and R.J. Maciunas. Neural-network-based segmentation of multi-modal medical images: a comparative and prospective study. *IEEE Transactions on Medical Imaging*, 12(3):534–544, 1993. Pages: 24

[267] J.K. Paik and A.K. Katsaggelos. Image restoration using a modified Hopfield network. *IEEE Transactions on Image Processing*, 1(1):49–63, 1992. http://ivpl.ece.nwu.edu/Publications/Journals/1992/IEEETransIP92a.pdf. Pages: 18, 19

[268] N.R. Pal and S.K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993. Pages: 24

[269] A. Papoulis. *The Fourier integral and its applications*. McGraw-Hill, New York, NY, 1962. Pages: 238

[270] T. N. Pappas. An adaptive clustering algorithm for image segmentation. *IEEE Transactions on Signal Processing*, 40(4):901–914, 1992. Pages: 24

[271] D.B. Parker. Learning logic, invention report s81-64, file 1. Technical report, Office of Technology Licensing, Stanford University, Stanford, California, 1982. Pages: 2

[272] G. Pasquariello, G. Satalino, V. la Forgia, and F. Spilotros. Automatic target recognition for naval traffic control using neural networks. *Image and Vision Computing*, 16(2):67–73, 1998.

Pages: 28

[273] D. Patel, E.R. Davies, and I. Hannah. The use of convolution operators for detecting contaminants in food images. *Pattern Recognition*, 29(6):1019–1029, 1996. Pages: 21, 22

[274] M.G. Penedo, M.J. Carreira, A. Mosquera, and D. Cabello. Computer-aided diagnosis: A neural-network-based approach to lung nodule detection. *IEEE Transactions on Medical Imaging*, 17(6):872–880, 1998. Pages: 26, 28

[275] A. Pentland, R.W. Picard, and S. Sclaroff. Photobook: tools for content-based manipulation of image databases. Technical report, Perceptual Computing Section, The Media Laboratory, MIT, Cambridge, MA, 1994. `ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-255.ps.Z`. Pages: 181

[276] L.I. Perlovsky. Conundrum of combinatorial complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):666–670, 1998. Pages: 31

[277] L.I. Perlovsky, W.H. Schoendorf, B.J. Burdick, and D.M. Tye. Model-based neural network for target detection in SAR images. *IEEE Transactions on Image Processing*, 6(1):203–216, 1997. Pages: 26

[278] D.T. Pham and E.J. Bayro-Corrochano. Neural computing for noise filtering, edge detection and signature extraction. *Journal of Systems Engineering*, 2:111–222, 1992. Pages: 21

[279] V.V. Phoha and W.J.B. Oldham. Image recovery and segmentation using competitive learning in a layered network. *IEEE Transactions on Neural Networks*, 7(4):843–856, 1996. Pages: 18, 19

[280] T. Poggio and C. Koch. Ill-posed problems in early vision: from computational theory to analogue networks. *Proceedings of the Royal Society London*, B(226):303–323, 1985. Pages: 15

[281] I. Pratt. *Artificial intelligence*. Macmillan, London, 1994. Pages: 1

[282] W. K. Pratt. *Digital image processing*. John Wiley & Sons, New York, NY, 2^{nd} edition, 1991. Pages: 3, 27, 66, 109

[283] L. Prechelt. Early stopping - but when? In G.B. Orr and K.-R. Müller, editors, *Neural networks: tricks of the trade*, pages 55–69. Springer-Verlag, Berlin, 1998. `http://wwwipd.ira.uka.de/~prechelt/Biblio/Biblio/stop_tricks1997.ps.gz`. Pages: 43

[284] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, MA, 2^{nd} edition, 1992. `http://www.nr.com/`. Pages: 81, 110

[285] P. Pudil, J. Novovicová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994. Pages: 30

[286] R.H. Pugmire, R.M. Hodgson, and R.I. Chaplin. The properties and training of a neural network based universal window filter developed for image processing tasks. In S. Amari and N. Kasabov, editors, *Brain-like computing and intelligent information systems*, chapter 3, pages 49–77. Springer-Verlag, Singapore, 1998. Pages: 21, 89, 100, 114

[287] W. Qian, M. Kallergi, and L.P. Clarke. Order statistic-neural network hybrid filters for gamma-camera-bremsstrahlung image restoration. *IEEE Transactions on Medical Imaging*, 12(1):58–64, 1993. Pages: 18, 19

[288] J.R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1992. Pages: 2

[289] N. Ramesh and I. Sethi. Feature identification as an aid to content-based image retrieval. In W. Niblack and R.C. Jain, editors, *Storage and retrieval for image and video databases III*,

volume 2420 of *Proc. of SPIE*, Bellingham, WA, 1995. SPIE, SPIE. Pages: 181

[290] H.S. Ranganath, D.E. Kerstetter, and R.F. Sims. Self partitioning neural networks for target recognition. *Neural Networks*, 8(9):1475–1486, 1995. Pages: 51

[291] S. Raudys. Evolution and generalization of a single neurone: I. Single-layer perceptron as seven statistical classifiers. *Neural Networks*, 11(2):283–296, 1998. Pages: 5, 12

[292] S. Raudys. Evolution and generalization of a single neurone: II. Complexity of statistical classifiers and sample size considerations. *Neural Networks*, 11(2):297–313, 1998. Pages: 5, 12

[293] E.S. Raymond. The new hacker's dictionary. `http://www.tuxedo.org/~esr/jargon/`. 2000. Pages: 35

[294] W.E. Reddick, J.O. Glass, E.N. Cook, T.D. Elkin, and R.J. Deaton. Automated segmentation and classification of multispectral magnetic resonance images of brain using artificial neural networks. *IEEE Transactions on Medical Imaging*, 16(6):911–918, 1997. Pages: 24

[295] E. Rich and K. Knight. *Artificial intelligence*. McGraw-Hill, New York, NY, 1991. Pages: 1

[296] M. D. Richard and R. P. Lippmann. Neural network classifiers estimate Bayesian posterior probabilities. *Neural Computation*, 3(4):461–483, 1991. Pages: 11

[297] B.D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, Cambridge, 1996. Pages: 11

[298] H. Ritter, T. Martinez, and K. Schulten. *Neuronale Netze*. Addison-Wesley, Bonn, 1991. Pages: 13

[299] S.K. Rogers, J.M. Colombi, C.E. Martin, J.C. Gainey, K.H. Fielding, T.J. Burns, D.W. Ruck, M. Kabrisky, and M. Oxley. Neural networks for automatic target recognition. *Neural Networks*, 8(7/8):1153–1184, 1995. Pages: 51

[300] R.G. Rosandich. Havnet - a new neural network architecture for pattern recognition. *Neural Networks*, 10(1):139–151, 1997. Pages: 27

[301] F. Rosenblatt. *Principles of neurodynamics*. Spartan Books, New York, NY, 1962. Pages: 2

[302] M.W. Roth. Survey of neural network technology for automatic target recognition. *IEEE Transactions on Neural Networks*, 1(1):28–43, 1990. Pages: 26, 51

[303] S. Rout, S. P. Srivastava, and J. Majumdar. Multi-modal image segmentation using a modified Hopfield neural network. *Pattern Recognition*, 31(6):743–750, 1998. Pages: 24, 28, 29

[304] S. Roweis. EM algorithms for PCA and SPCA. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*. MIT Press, Cambridge, MA, 1997. `http://www.gatsby.ucl.ac.uk/~roweis/publications.html`. Pages: 129, 130, 142

[305] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998. `http://www.cs.cmu.edu/afs/cs.cmu.edu/user/har/Web/pami98.ps.gz`. Pages: 26, 27, 42

[306] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, pages 319–362. MIT Press, Cambridge, MA, 1986. Pages: 2, 9, 11, 37, 98

[307] F. Russo. Hybrid neuro-fuzzy filter for impulse noise removal. *Pattern Recognition*, 32(11): 1843–1855, 1999. Pages: 19

[308] F. Russo. Image filtering using evolutionary neural fuzzy systems. In S.K. Pal, A. Ghosh,

and M.K. Kundu, editors, *Soft computing for image processing*, pages 23–43. Physica-Verlag, Heidelberg, 2000. Pages: 19

[309] B. Sahiner, H.P. Chan, N. Petrick, D.T. Wei, M.A. Helvie, D.D. Adler, and M.M. Goodsitt. Classification of mass and normal breast tissue - a convolution neural network classifier with spatial domain and texture images. *IEEE Transactions on Medical Imaging*, 15(5):598–610, 1996. Pages: 26

[310] H. Sako, M. Whitehouse, A. Smith, and A. Sutherland. Real-time facial-feature tracking based on matching techniques and its applications. In *Proc. 12th IAPR International Conference on Pattern Recognition (ICPR'94), Vol. II*, pages 320–324, Piscataway, NJ, 1994. IEEE. Pages: 21, 22

[311] T. Sams and J.L. Hansen. Implications of physical symmetries in adaptive image classifiers. *Neural Networks*, 13(6):565–570, 2000. Pages: 25

[312] J. Scharcanski, J.K. Hovis, and H.C. Shen. Representing the color aspect of texture images. *Pattern Recognition Letters*, 15(2):191–197, 1994. Pages: 21, 22, 23

[313] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time delay neural networks and hidden Markov models. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP'95)*, volume 2, page 637, 1995. `http://www.isi.ee.ethz.ch/~schenkel/publications/MVA-paper.ps`. Pages: 42

[314] A.J. Schofield, P.A. Mehta, and T.J. Stonham. A system for counting people in video images using neural networks to identify the background scene. *Pattern Recognition*, 29(8):1421–1428, 1996. Pages: 24

[315] A. Schuler, P. Nachbar, J.A. Nossek, and L.O. Chua. Learning state space trajectories in cellular neural networks. In *IEEE International Workshop on Cellular Neural Networks and their applications (CNNA '92)*, pages 68–73, München, 1992. Pages: 18

[316] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, I:145–168, 1987. Pages: 37

[317] S.B. Serpico, L. Bruzzone, and F. Roli. An experimental comparison of neural and statistical non-parametric algorithms for supervised classification of remote-sensing images. *Pattern Recognition Letters*, 17(13):1331–1341, 1996. Pages: 24

[318] R. Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1):205–225, 1997. `http://www.comp.nus.edu.sg/~rudys/p1072.ps`. Pages: 63

[319] R. Setiono and H. Liu. Neural network feature selector. *IEEE Transactions on Neural Networks*, 8(3):645–662, 1997. `http://www.comp.nus.edu.sg/~rudys/newfs.ps`. Pages: 63

[320] R. Shapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990. Pages: 114

[321] D. Shen and H.H.S. Ip. A Hopfield neural network for adaptive image segmentation: an active surface paradigm. *Pattern Recognition Letters*, 18(1):37–48, 1997. `http://cbic1.rad.jhu.edu/~dgshen/papers/PRL97.pdf.gz`. Pages: 28, 29

[322] J-.Y. Shen, Y-.X. Zhang, and G-.G. Mu. Optical pattern recognition system based on a winner-take-all model of a neural network. *Optical Engineering*, 32(5):1053–1056, 1992. Pages: 26

[323] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mel-

lon University, Pittsburgh, Philadelphia, March 1994. http://www-2.cs.cmu.edu/~quake-papers/painless-conjugate-gradient-figs.ps. Pages: 11, 55, 68

[324] F.Y. Shih, J. Moh, and F.-C. Chang. A new ART-based neural architecture for pattern classification and image enhancement without prior knowledge. *Pattern Recognition*, 25 (5):533–542, 1992. Pages: 21, 26

[325] A. Shustorovich. A subspace projection approach to feature extraction - the 2-D Gabor transform for character recognition. *Neural Networks*, 7(8):1295–1301, 1994. Pages: 21, 22

[326] R. H. Silverman and A. S. Noetzel. Image processing and pattern recognition in ultrasonograms by backpropagation. *Neural Networks*, 3(5):593–604, 1990. Pages: 24

[327] R.H. Silverman. Segmentation of ultrasonic images with neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 5:619–628, 1991. Pages: 24

[328] P. Simard, Y. Le Cun, J. Denker, and B. Victorii. An efficient algorithm for learning invariances in adaptive classifiers. In *Proc. 11th IAPR International Conference on Pattern Recognition (ICPR'92), Volume II, Conference B: Pattern Recognition Methodology and Systems*, pages 651–665, Los Alamitos, CA, september 1992. IAPR, IEEE Computer Society Press. Pages: 47

[329] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA, 1993. http://www.research.att.com/~yann/publis/. Pages: 31

[330] J. Sklansky and M. Vriesenga. Genetic selection and neural modelling of piecewise-linear classifiers. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(5):587–612, 1996. Pages: 26

[331] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1379, 2000. http://columbo.ucsd.edu/work/CBIR/PAMI-01-review.pdf. Pages: 181

[332] S. A. Solla and Y. Le Cun. Constrained neural networks for pattern recognition. In P. Antognetti and V. Milutinovic, editors, *Neural networks: concepts, applications and implementations*. Prentice Hall, 1991. http://www.research.att.com/~yann/publis/. Pages: 41, 46

[333] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis and machine vision*. Chapman & Hall, London, 1993. Pages: 4

[334] E. D. Sontag. Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, 3(6):981–990, 1992. http://www.math.rutgers.edu/~sontag/FTP_DIR/two-layer.pdf. Pages: 12

[335] L. Spirkovska and M.B. Reid. Coarse-coded higher-order neural networks for PSRI object recognition. *IEEE Transactions on Neural Networks*, 4(2):276–283, 1993. Pages: 26, 27

[336] L. Spirkovska and M.B. Reid. Higher-order neural networks applied to 2D and 3D object recognition. *Machine Learning*, 15(2):169–199, 1994. Pages: 26, 27

[337] L. J. Spreeuwers. *Image filtering with neural networks, applications and performance evaluation*. PhD thesis, Universiteit Twente, Enschede, 1992. Pages: 18, 20, 21, 42, 109, 125

[338] V. Srinivasan, P. Bhatia, and S.H. Ong. Edge detection using a neural network. *Pattern Recognition*, 27(12):1653–1662, 1994. Pages: 21

[339] V. Srinivasan, Y.K. Han, and S.H. Ong. Image reconstruction by a Hopfield neural net-

work. *Image and Vision Computing*, 11(5):278–282, 1993. Pages: 18

[340] D. Stanford and A.E. Raftery. Principal curve clustering with noise. Technical Report 317, Dept. of Statistics, University of Washington, Box 354322, Seattle, WA, February 1997. http://www.stat.washington.edu/raftery. Pages: 128

[341] G.W. Stewart. Error and perturbation bounds for subspaces associated with certain eigenvalue problems. *SIAM Review*, 15:727–764, 1973. Pages: 182

[342] D.G. Stork and A.C. Clarke. *Hal's legacy: 2001's computer as dream and reality*. MIT Press, Cambridge, MA, 1999. Pages: 1

[343] G. Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich, San Diego, CA, 3rd edition, 1989. Pages: 130, 139

[344] P.N. Suganthan, E.K. Teoh, and D.P. Mital. Pattern recognition by graph matching using the Potts MFT neural networks. *Pattern Recognition*, 28(7):997–1009, 1995. Pages: 28, 29

[345] P.N. Suganthan, E.K. Teoh, and D.P. Mital. Pattern recognition by homomorphic graph matching using Hopfield neural networks. *Image and Vision Computing*, 13(1):45–60, 1995. Pages: 15

[346] P.N. Suganthan and H. Yan. Recognition of handprinted chinese characters by constrained graph matching. *Image and Vision Computing*, 16(3):191–201, 1998. Pages: 21, 22, 28, 29

[347] Y. Sun and S.-Y. Yu. An eliminating highest error IEHE criterion in Hopfield neural networks for bilevel image restoration. *Pattern Recognition Letters*, 14(6):471–474, 1993. Pages: 18, 19

[348] K.-K. Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998. ftp://publications.ai.mit.edu/ai-publications/1500-1999/AIM-1521.ps.Z. Pages: 138, 140, 141

[349] S. Tamura, H. Kawai, and H. Mitsumoto. Male female identification from 8x6 very low resolution face images by neural network. *Pattern Recognition*, 29(2):331–335, 1996. Pages: 26

[350] D.M.J. Tax. *One-class classification*. PhD thesis, Delft University of Technology, Delft, 2001. http://www.ph.tn.tudelft.nl/PHDTheses/DMJTax/thesis_tax.html. Pages: 31, 51, 53

[351] D.M.J. Tax, D. de Ridder, and R.P.W. Duin. Support vector classifiers: a first look. In H.E. Bal, H. Corporaal, P.P. Jonker, and J.F.M. Tonino, editors, *Proc. 3rd Annual Conference of the Advanced School for Computing and Imaging (ASCI'97)*, pages 253–258, Delft, The Netherlands, 1997. ASCI, ASCI. http://www.ph.tn.tudelft.nl/~dick/publications.html. Pages: 46

[352] D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11-13):1191–1199, 1999. Pages: 31

[353] S. Thirumalai and N. Ahuja. Parallel distributed detection of feature trajectories in multiple discontinuous motion image sequences. *IEEE Transactions on Neural Networks*, 7(3): 594–603, 1996. Pages: 28, 29

[354] A.B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–1068, 1998. Pages: 13, 28, 63

[355] M. Tipping and C. Bishop. Probabilistic principal component analysis. Technical Re-

port Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, Birmingham, UK, September 1997. http://www.gatsby.ucl.ac.uk/~quaid/course/readings/ppca.ps. Pages: 129, 130, 142, 167, 171, 172

[356] M.E. Tipping and C.M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999. ftp://ftp.research.microsoft.com/users/mtipping/mppca.ps.gz. Pages: 22, 128, 129, 130, 140, 171, 194

[357] F. Tomita and S. Tsuji. Extraction of multiple regions by smoothing in selected neighbourhoods. *IEEE Transactions on Systems, Man and Cybernetics*, 7:107–109, 1977. Pages: 90

[358] C.-T. Tsai, Y.-N. Sun, P.-C. Chung, and J.-S. Lee. Endocardial boundary detection using a neural network. *Pattern Recognition*, 26(7):1057–1068, 1993. Pages: 21

[359] M. Turner, J. Austin, N.M. Allinson, and P. Thompson. Chromosome location and feature extraction using neural networks. *Image and Vision Computing*, 11(4):235–239, 1993. Pages: 22, 26, 27

[360] S. Usui, S. Nakauchi, and M. Nakano. Internal color representation acquired by a five-layer neural network. In O. Simula, T. Kohonen, K. Makisara, and J. Kangas, editors, *International Conference on Artificial Neural Networks*, Helsinki, Finland, 1991. Elsevier, Amsterdam. Pages: 22

[361] M.M. van Hulle and T. Tollenaere. A modular artificial neural network for texture processing. *Neural Networks*, 6(1):7–32, 1993. Pages: 24

[362] V.N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, Berlin, 1995. Pages: 46

[363] P.F.M.J. Verschure. *Neural networks and a new AI*, chapter Connectionist explanation: taking positions in the mind-brain dilemma, pages 133–188. Thompson, London, 1996. ftp://ftp.ini.unizh.ch/pub/pfmjv/pub/ConnExpl.ps.gz. Pages: 63

[364] E. Viennet. *Architectures Connexionistes Multi-Modulaires, Application à l'Analyse de Scène*. PhD thesis, Université de Paris-Sud, Centre d'Orsay, 1993. Pages: 26, 37, 41, 42, 47, 72, 209

[365] D.L. Vilarino, V.M. Brea, D. Cabello, and J.M. Pardo. Discrete-time CNN for image segmentation by active contours. *Pattern Recognition Letters*, 19(8):721–734, 1998. Pages: 24

[366] V.V. Vinod, S. Chaudhury, J. Mukherjee, and S. Ghose. A connectionist approach for clustering with applications in image analysis. *IEEE Transactions on Systems Man and Cybernetics*, 24(3):365–383, 1994. Pages: 24

[367] M.A.J. Wachters. Non-linear convolution networks for texture discrimination. Master's thesis, Pattern Recognition Group, Dept. of Applied Physics, Delft University of Technology, august 1996. Pages: 8

[368] J. Waldemark. An automated procedure for cluster analysis of multivariate satellite data. *International Journal of Neural Systems*, 8(1):3–15, 1997. Pages: 24

[369] W.G. Waller and A.K. Jain. On the monotonicity of the performance of a Bayesian classifier. *IEEE Transactions on Information Theory*, 24(3):392–394, 1978. Pages: 21

[370] W.H. Walton. Automatic counting of microscopic particles. *Nature*, 169:518–520, 1952. Pages: 4

[371] L.C. Wang, S.Z. Der, and N.M. Nasrabadi. Automatic target recognition using a feature-decomposition and data-decomposition modular neural network. *IEEE Transactions on Image Processing*, 7(8):1113–1121, 1998. Pages: 21, 22, 26, 27

[372] L.C. Wang, S.Z. Der, and N.M. Nasrabadi. Composite classifiers for automatic target recognition. *Optical Engineering*, 37(3):858–868, 1998. Pages: 21, 22, 26, 27

[373] T. Wang, X. Zhuang, and X. Xing. Robust segmentation of noisy images using a neural network model. *Image and Vision Computing*, 10(4):233 – 240, 1992. Pages: 24

[374] Y. Wang, T. Adali, S.Y. Kung, and Z. Szabo. Quantification and segmentation of brain tissues from MR images - a probabilistic neural network approach. *IEEE Transactions on Image Processing*, 7(8):1165–1181, 1998. http://www.engr.umbc.edu/~adali/papers/IP98.pdf. Pages: 24

[375] Y.M. Wang and F.M. Wahl. Vector-entropy optimization-based neural-network approach to image reconstruction from projections. *IEEE Transactions on Neural Networks*, 8(5):1008–1014, 1997. Pages: 18

[376] A.M. Waxman, M.C. Seibert, A. Gove, D.A. Fay, A.M. Bernardon, C. Lazott, W.R. Steele, and R.K. Cunningham. Neural processing of targets in visible multispectral IR and SAR imagery. *Neural Networks*, 8(7-8):1029–1051, 1995. Pages: 19, 21

[377] A. Webb. *Statistical pattern recognition*. Arnold, London, 1999. Pages: 45, 139

[378] E.W. Weisstein. *The CRC concise encyclopedia of mathematics*. CRC Press, Boca Raton, FL, 1998. http://mathworld.wolfram.com/. Pages: 115

[379] P. J. Werbos. Beyond regression: new tools for prediction and analysis in the behavorial sciences. Master's thesis, Harvard University, 1974. Pages: 2

[380] C.K.I. Williams, M. Revow, and G.E. Hinton. Instantiating deformable models with a neural net. *Computer Vision and Image Understanding*, 68(1):120–126, 1997. http://www.cs.toronto.edu/~revow/papers/NCRG_96_025.ps.Z. Pages: 21, 22

[381] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. ftp://ftp.ccs.neu.edu/pub/people/rjw/conn-reinf-ml-92.ps. Pages: 114

[382] C. L. Wilson and M. D. Garris. Handprinted character database 3, february 1992. http://www.nist.gov/srd/nistsd19.htm. National Institute of Standards and Technology; Advanced Systems division. Pages: 43

[383] A.J. Worth and D.N. Kennedy. Segmentation of magnetic resonance brain images using analogue constraint satisfaction neural networks. *Image and Vision Computing*, 12(6):345–354, 1994. Pages: 24

[384] H. Yan and J. Wu. Character and line extraction from color map images using a multi-layer neural network. *Pattern Recognition Letters*, 15(1):97–103, 1994. Pages: 24

[385] I.T. Young, J.J. Gerbrands, and L.J. Van Vliet. *The digital signal processing handbook*, chapter Image processing fundamentals, pages 51/1 – 51/81. CRC Press/IEEE Press, Boca Raton, FL, 1998. http://www.ph.tn.tudelft.nl/Courses/FIP. Pages: 5, 66, 91, 103, 109

[386] S.S. Young, P.D. Scott, and C. Bandera. Foveal automatic target recognition using a multiresolution neural network. *IEEE Transactions on Image Processing*, 7(8):1122–1135, 1998. Pages: 26, 27

[387] S.S. Young, P.D. Scott, and N.M. Nasrabadi. Object recognition using multilayer Hopfield neural network. *IEEE Transactions on Image Processing*, 6(3):357–372, 1997. Pages: 27

[388] S.-S. Yu and W.-H. Tsai. Relaxation by the Hopfield neural network. *Pattern Recognition*, 25(2):197–210, 1992. Pages: 28, 29

[389] M. Zamparelli. Genetically trained cellular neural networks. *Neural Networks*, 10(6):1143–1151, 1997. Pages: 18

[390] A. Zell, G. Mamier, M. Vogt, N. Mache, R. Hübner, S. Döring, K.-U. Herrman, T. Soyez, M. Schmalzl, T. Sommer, A. Hatzigeorgiou, D. Posselt, T. Schreiner, B. Kett, G. Clemente,

and J. Wieland. SNNS (Stuttgart Neural Network Simulator) user manual, version 4.1. http://www-ra.informatik.uni-tuebingen.de/SNNS/. 2000. Pages: 35

[391] Y.J. Zhang. A survey on evaluation methods for image segmentation. *Pattern Recognition*, 29(8):1335–1346, 1996. Pages: 25, 148

[392] Z.Z. Zhang and N. Ansari. Structure and properties of generalized adaptive neural filters for signal enhancement. *IEEE Transactions on Neural Networks*, 7(4):857–868, 1996. http://www-ec.njit.edu/~ang/papers/NN96.pdf. Pages: 18, 19

[393] Y. Zheng, J.F. Greenleaf, and J.J. Gisvold. Reduction of breast biopsies with a modified self-organizing map. *IEEE Transactions on Neural Networks*, 8(6):1386–1396, 1997. Pages: 22, 26

[394] Y.T. Zhou, R. Chellappa, A. Vaid, and B.K. Jenkins. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(7):1141–1151, 1988. Pages: 18

[395] Z.G. Zhu, S.Q. Yang, G.Y. Xu, X.Y. Lin, and D.J. Shi. Fast road classification and orientation estimation using omni-view images and neural networks. *IEEE Transactions on Image Processing*, 7(8):1182–1197, 1998. Pages: 26

[396] T. Ziemke. Radar image segmentation using recurrent artificial neural networks. *Pattern Recognition Letters*, 17(4):319–334, 1996. http://www.his.se/ida/~tom/HS-IDA-TR-96-001.ghostview.ps.Z. Pages: 26, 27

SUMMARY

This thesis is concerned with the application of adaptive methods in image processing. Adaptive methods, among which artificial neural networks, have since the 1980s become popular all-purpose tools. In chapter 1, three basic questions are formulated:

- Can (nonlinear) image processing operations be learned by adaptive methods?
- How can prior knowledge be used in the construction and training of adaptive methods?
- What can be learned from adaptive methods trained to solve image processing problems?

The remainder of the thesis answers these questions for three types of adaptive methods:

- supervised classification by shared weight feed-forward neural networks;
- supervised regression by modular feed-forward networks;
- unsupervised clustering by mixture-of-subspace models.

First, chapter 2 reviews publications on applications of artificial neural networks to image processing problems. To this end, the image processing chain is discussed: preprocessing, feature extraction, segmentation, object recognition and image understanding. As many problems can be seen as optimisation problems, optimisation methods are viewed as a separate, auxiliary category. In the review, it is noted that many of the more low-level applications do not use the adaptive capabilities of artificial neural networks; here, the possibility of hardware implementation seems to be a reason to choose for an artificial neural network. Artificial neural networks have been mostly applied for feature extraction (the self-organising map) and segmentation and object recognition (feed-forward neural networks). A number of problems in application of artificial neural networks are discussed, among which their black-box character and the question whether prior knowledge on the nature of image processing problems can be used sensibly.

In chapter 3, shared weight neural networks are introduced and applied to the problems of handwritten digit recognition and automatic target recognition. These networks, con-

taining receptive field-like weight sets and aggregating layers, have been constructed with the goal of image processing in mind. They perform well, although some traditional classification methods are still slightly better. The networks' feature extraction capabilities are investigated and found to be quite good. Unfortunately, it is hard to make sense of the extracted features. To learn more, in chapter 4 two simpler problems are treated: edge recognition and two-class handwritten digit recognition. By constructing a number of more and more restricted networks, it is shown that the networks have a tendency to use all their degrees of freedom. Standard feed-forward networks internally do not conform to the traditional image processing approach, and functionality is distributed over the network in an unclear way. A learning algorithm, decorrelating conjugate gradient descent, is proposed to obtain separate feature detectors. This helps in understanding the networks behaviour after training. However, the restrictions on the network needed for interpretation make the situation non-representative of everyday neural network use. This is called the interpretability trade-off.

Next, chapter 5 moves to another problem, that of pre-processing. A set of modular and standard feed-forward regression networks is trained to emulate the Kuwahara edge-preserving smoothing filter. The modular networks have been constructed using the fact that the Kuwahara filter can be split up into four sub-systems. Judging by the mean squared error (the criterion used in training the networks) the use of prior knowledge has no influence: all networks seem to perform equally well. In chapter 6, a number of hypotheses are tested as to why this should be the case. Using a novel performance criterion for edge-preserving smoothing, it is shown that in fact the modular network do perform better. Furthermore, evaluation of network errors indicates most errors occur around the edges in the image. A data set containing relatively more samples taken around edges improves performance. The mean squared error is a poor criterion both for learning this nonlinear filter and for selecting between different trained ANNs. To learn more on how the networks approximate the filter, they are inspected. The modular networks, which have been trained further after the modules have been concatenated, are shown to perform better the more they have lost their modular initialisation; modular initialisation is not useful. Using the weight-decorrelating training algorithm, the standard networks are shown to have learned a linear approximation of the Kuwahara filter.

As a last method, in chapter 7 subspace models are introduced as feature extraction mechanisms, applied to texture description. Many images, high-dimensional vectors when described as collections of pixels, can be adequately described using a small number of subspace basis vectors. These subspaces can also be learned using episodes, collections of samples which have been translated, rotated or scaled. In this way, the subspaces become descriptions invariant to these transformations. Two subspace models are compared, principal component analysis and independent component analysis. Principal component analysis is shown to give results nearly as good as those obtain using full Gaussian models, and better than Gabor filters, yet at a fraction of the number of parameters needed. A number of experiments proves the descriptive power of prin-

cipal component subspaces. Independent component analysis, for which a learning rule is derived in appendix C, is not fit for description of textures. It focuses on rarely occurring, high-frequency image elements. In chapter 8, mixtures of principal subspaces are then applied successfully to image segmentation and handwritten digit recognition. On the digit recognition problem, performance is identical to that of the shared weight neural networks, but using less parameters. Extending these results, histograms of image pixels assigned to subspaces are used for object recognition and distances between mixture models are used as distances between images themselves in an image database retrieval application. In all these applications, interpretation of the working of the method is straightforward.

The conclusion is that artificial neural networks can well be applied to high-level, complex image processing problems for which it is too hard to construct a model, but for which a clearly measurable criterion function is available. The black-box problem remains: after training, it is hard to say what solution the network has arrived at. If prior knowledge is available, it should be used to construct a (simple) adaptive model and find the remaining parameters by learning. Such models can perform as well as artificial neural networks, at the same time offering more insight.

Dick de Ridder

SAMENVATTING

Dit proefschrift onderzoekt de toepassing van adaptieve methoden in de beeldbewerking. Adaptieve methoden, waaronder kunstmatige neurale netwerken, zijn sinds de jaren tachtig populaire gereedschappen geworden met een breed toepassingsgebied. In hoofdstuk 1 worden drie kernvragen geformuleerd:

- Kunnen (niet-lineaire) beeldbewerkingsoperaties door adaptieve methoden worden geleerd?
- Hoe kan voorkennis bij het construeren en leren van adaptieve methoden worden gebruikt?
- Wat kan men leren van adaptieve methoden die geleerd hebben een beeldbewerkingsprobleem op te lossen?

De rest van het proefschrift beantwoordt deze vragen voor drie types van adaptieve methoden:

- classificatie met leraar (*supervised*) door *feed-forward* neurale netwerken met gedeelde gewichten;
- regressie met leraar door modulaire feed-forward neurale netwerken;
- clustering zonder leraar (*unsupervised*) door combinatie-van-deelruimten modellen (*subspace mixture models*).

In hoofdstuk 2 wordt een overzicht gegeven van publicaties over toepassingen van kunstmatige neurale netwerken op beeldbewerkingsproblemen. Hiertoe wordt eerst de beeldbewerkingspijplijn geïntroduceerd: voorbewerking, kenmerk-extractie, segmentatie, objectherkenning en interpretatie. Aangezien veel problemen kunnen worden beschouwd als optimalisatieproblemen, worden optimalisatietechnieken als aparte, ondersteunende categorie gedefinieerd. Uit het overzicht blijkt dat veel van de toepassingen op laag niveau geen gebruik maken van de adaptiviteit van kunstmatige neurale netwerken; de mogelijkheid om de netwerken in silicon te implementeren lijkt hier een reden te zijn om voor een kunstmatig neuraal netwerk te kiezen. Kunstmatige neurale netwerken zijn voornamelijk toegepast op kenmerk-extractie (de *self-organising map*) en segmentatie en objectherkenning (feed-forward neurale netwerken). Een aantal proble-

men bij de toepassing van kunstmatige neurale netwerken wordt besproken, waaronder hun zwarte-doos-karakter en de vraag of voorkennis over eigenschappen van beeldbewerkingsproblemen op een zinvolle manier gebruikt kan worden.

Hoofdstuk 3 behandelt neurale netwerken met gedeelde gewichten, die worden toegepast op herkenning van handgeschreven cijfers en herkenning van voertuigen. Deze netwerken, die *receptive field*-achtige sets van gewichten en middelende lagen hebben, zijn gebouwd met een toepassing op beeldmateriaal in gedachten. Ze presteren goed, hoewel sommige traditionele methoden nog iets beter werken. De kenmerk-extractie capaciteiten worden onderzocht en blijken vrij goed te zijn. Helaas is het onmogelijk om de gevonden kenmerkdetectoren te interpreteren. Om hierover meer te leren, worden in hoofdstuk 4 twee simpelere problemen behandeld: randherkenning en herkenning van twee klassen van handgeschreven cijfers. Een reeks gebouwde netwerken, steeds meer beperkt in hun vrijheid, laat zien dat neurale netwerken de neiging hebben al hun vrijheidsgraden te gebruiken. Standaard feed-forward netwerken werken intern niet als traditionele beeldbewerkingsoperaties, en functionaliteit wordt op onduidelijke wijze verdeeld binnen het netwerk. Een leeralgoritme, de decorrelerende geconjugeerde gradiënt methode, wordt geformuleerd om gesepareerde kenmerkdetectoren te verkrijgen. Dit helpt in het begrijpen van het gedrag van netwerken na het leren. De restricties die nodig zijn om het netwerk te begrijpen zorgen ervoor dat de situatie niet representatief is voor het alledaags gebruik van neurale netwerken. Dit wordt de interpretatieafweging genoemd.

In hoofdstuk 5 wordt een ander probleem behandeld, dat van voorbewerking. Een verzameling modulaire en standaard feed-forward regressie netwerken wordt geleerd het Kuwahara filter voor randbehoudende effening te emuleren. De modulaire netwerken zijn gebouwd gebruikmakend van het feit dat het Kuwahara filter algoritme uit vier deeltaken bestaat. Afgaand op de gemiddelde kwadratische fout (het criterium zoals gebruikt tijdens het leren van de netwerken) lijkt dit gebruik van voorkennis geen invloed te hebben: alle netwerken lijken even goed te functioneren. In hoofdstuk 6 wordt een aantal hypothesen getoetst waarom dit het geval zou zijn. Gebruikmaking van een nieuw prestatiecriterium voor randbehoudende effening laat zien dat de modulair netwerken wèl beter presteren. Daarnaast laat evaluatie van de fouten die het netwerk maakt, zien dat de meeste fouten optreden nabij de randen in een beeld. Als in de data set relatief meer elementen opgenomen worden die verkregen zijn nabij randen, verbeteren de prestaties. De gemiddelde kwadratische fout is een slecht criterium voor zowel het leren van dit niet-lineaire filter, als voor het selecteren van de beste van een aantal geleerde netwerken. Tenslotte worden alle netwerken geïnspecteerd om meer kennis te verkrijgen over hoe zij het filter benaderen. De modulaire netwerken, die doorgeleerd hebben nadat de modules geconcateneerd zijn, blijken beter te presteren naarmate zij hun modulaire initialisatie verloren hebben; modulaire initialisatie is niet zinvol. Gebruik van het decorrelerende geconjugeerde gradiënten algoritme laat zien dat de standaard netwerken het Kuwahara filter lineair benaderd hebben.

Een laatste methode, combinatie-van-deelruimten modellen, wordt geïntroduceerd in hoofdstuk 7 voor kenmerk-extractie en toegepast op textuurbeschrijving. Veel beelden, hoogdimensionale vectoren wanneer ze beschreven worden als verzamelingen pixels, kunnen goed beschreven worden door gebruik te maken van slechts een klein aantal deelruimtebasisvectoren. Deze deelruimten kunnen tevens geleerd worden gebruikmakend van episodes, verzamelingen van getransleerde, geroteerde en geschaalde beelden. Op deze manier worden deelruimten beschrijvingen van beelden die invariant zijn met betrekking tot deze transformaties. Twee deelruimtemodellen worden vergeleken, principale componenten analyse en onafhankelijke componenten analyse. Principale componenten analyse geeft resultaten die vrijwel even goed zijn als volledige Gaussische modellen, en beter dan Gabor filters, maar gebruikmakend van slechts een fractie van het aantal benodigde parameters. Een aantal experimenten bewijst dat principale componenten analyse een krachtige beschrijvende methode is. Onafhankelijke componenten analyse, waarvoor een leerregel wordt afgeleid in appendix C, is niet toepasbaar op textuurbeschrijving. Het richt zich op weinig voorkomende, hoogfrequente elementen in beelden. In hoofdstuk 8 worden combinaties van principale deelruimten succesvol toegepast op beeldsegmentatie en herkenning van handgeschreven cijfers. De prestaties op het cijferherkenningsprobleem zijn gelijk aan die van de neurale netwerken met gedeelde gewichten, ofschoon zij gebruikmaken van minder parameters. Als uitbreiding op deze methode worden histogrammen van aan deelruimten toegekende beeldelementen (*pixels*) gebruikt voor objectherkenning, en worden afstanden tussen combinatiemodellen gebruikt als afstanden tussen beelden in een beeld-databank (*image database*) toepassing. In al deze toepassingen is interpretatie van het functioneren van de methode eenvoudig.

De conclusie is dat kunstmatige neurale netwerken goed toegepast kunnen worden op complexe, hoog niveau beeldbewerkingsproblemen waarvoor het te lastig is een model op te stellen, maar waarvoor wel een éénduidig meetbaar criterium voorhanden is. Het zwarte-doos-probleem blijft bestaan; na het leren is het lastig te zeggen wat voor oplossing het netwerk gevonden heeft. Als voorkennis aanwezig is, zou deze gebruikt moeten worden om een (eenvoudig) adaptief model op te zetten, waarvoor de parameters door leren gevonden kunnen worden. Zulke modellen kunnen even goed presteren als kunstmatige neurale netwerken, maar bieden meer inzicht.

Dick de Ridder

Curriculum vitae

Dick de Ridder was born in Rotterdam on December 15, 1971. In 1990 he obtained the VWO diploma at the Comenius College in Capelle aan den IJssel and started a study in Computer Science at the Delft University of Technology. After graduating in the Pattern Recognition Group of the Department of Applied Physics, he received his M.Sc. degree with honours in 1996. The subject of the thesis was "Shared weight neural networks in image analysis", a preliminary study on the subject of this thesis. During his studies, he visited the Universitat de Barcelona on a two month exchange program, and worked for five years as a part-time programmer and systems analyst.

In 1996 he started a Ph.D. at the Pattern Recognition Group, in a NWO/SION project called "Nonlinear-feature extraction from image data", under supervision of Dr. ir. Robert P.W. Duin. During his five years in the group he participated in various scientific, educational and applied projects in collaboration with TNO/FEL, Shell, Parsytec GmbH, DERA (UK) and Hewlett-Packard. In 1999 and 2000, he visited the Centre for Vision, Speech and Signal Processing at the University of Surrey in the United Kingdom, as a guest of Prof. dr. Josef Kittler. This visit was partly sponsored by the EPSRC.

In 2001, he began working as a teacher and postdoctoral researcher in the Pattern Recognition Group, in a project called "Tools for non-linear data analysis".

Acknowledgements

This thesis is long enough as it is, so I will try to keep these acknowledgements brief. Let me therefore start by apologising to anyone I have not named below; be assured your help was valued!

First and foremost, I would like to thank Bob Duin, who supervised me during the last six years. After a careful start as an MSc student (of what use could a computer scientist be in the physics department?), during my Ph.D. he has become my mentor rather than merely a supervisor. I learned a lot from him, not only on pattern recognition but – perhaps more importantly – on the scientific process itself. Thanks also go to Lucas van Vliet, my *promotor*, for his dedication in reading the thesis, providing ideas and being open-minded enough to have good debates with; and to Piet Verbeek, who donated some very good ideas (although it sometimes took a while for me to understand they were good). Finally, Ted Young made sure my stay at the Pattern Recognition Group was a good and enjoyable one.

During my Ph.D., I spent some time at Centre for Vision, Speech and Signal Processing at the University of Surrey, under Josef Kittler. I would like to thank him for not only finding a place for me, but also for finding time in his hectic schedule to have discussions as if I were one of his own students. Many thanks also go to the other people I've worked with in the Centre.

As the bibliography shows, I have co-operated with many people, which is one of the opportunities science offers that I value much. Without Aarnoud Hoekstra, David Tax, Judith Dijk, Michael Egmont-Petersen, Klamer Schutte, Kieron Messer, Eric Körber, Olaf Lemmers, André van der Graaf, Ela Pękalska and Mohamed Musa this thesis would have been a lot less voluminous. Ela Pękalska offered to read my thesis once it was finished and, although I did not expect a kind of Spanish Inquisition, managed to find a couple of important errors and inconsistencies. Thank you all!

Next to direct co-operation, many people have helped me along by having discussions. I would particularly like to mention David Tax, Alexander Ypma, Ela Pękalska, Pavel Paclík and Marina Skurichina for many good discussions on pattern recognition and more philosophical issues. The same goes for the Emergentia discussion group, which showed me how varied the ways in which people approach artificial intelligence are.

Michael van Ginkel was always there to help me get my head around some image processing (or, as it often turned out, image processing software) problems.

The diversity of people in the Pattern Recognition Group made sure I had a good time outside office hours as well. Thanks for all the pub visits, BBQs, ASCI parties, dinners and so on! I would also like to thank the people in the University of Surrey, who made my stay there not just fruitful but enjoyable as well (among other things by showing me a completely new way of playing Monopoly).

Finally I would like to thank my mother Sonja, for making me realise that receiving an education is a gift, not a right; and my wife, Barbara. I realise I asked a lot of you whenever I was moody when experiments did not go as I had planned, during the nine months you spent alone in Rotterdam when I was in Guildford and in the time I wrote my thesis. Thank you for being there for me.

After 25 years it would seem that my education is finished. I hope I can go on learning and discovering new things.