IMAGE FILTERING WITH NEURAL NETWORKS

Applications and performance evaluation

Luuk Spreeuwers

ISBN 90-9005555-X

Address: University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands

IMAGE FILTERING WITH NEURAL NETWORKS

APPLICATIONS AND PERFORMANCE EVALUATION

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. Th.J.A. Popma,
volgens besluit van het College van Dekanen
in het openbaar te verdedigen
op vrijdag 6 november 1992 te 13.15 uur.

door
Lieuwe Jan Spreeuwers
geboren op 18 september 1964
te Emmen

Dit proefschrift is goedgekeurd door de promotoren

prof. ir. D. Bosman
ir. Z. Houkes

Acknowledgements

This work could not have come into being without the support of many persons. Especially grateful I am to Dick Bosman, Zweitze Houkes and Ferdi van der Heijden who guided and supported me during my research and my work on the thesis. I would like to thank them and dr.ir. Duin, prof.ir. Mulder and prof.dr.ir. Nijholt for their comments and critical notes on the manuscript. I owe many thanks to the students who did their assignments in the framework of my research. There were a lot of them and they did a lot a work. Their reports have been a great help in writing this thesis. I would like to thank the staff of the measurement laboratory who created an atmosphere in which I always worked with pleasure. Of great importance were Egbert Nijland and Alfred de Vries of our technical staff, who kept the computer systems up running during vital experiments. The cooperation and many discussions with my colleagues Aart-Jan de Graaf, Klamer Schutte, Fred Hugen and Ben Bulsink have improved my insight in image processing and many other things. The latter during the coffee breaks in the morning and the lunches together, where we solved many of the problems of the world. I also owe special thanks to Anneke van Essen and Joan Mollevanger, the secretaries of our group, who solved many problems that were too difficult for me. Finally I would like to thank my parents, to whom I dedicate this work, and God who created a world in which I discover astonishing new aspects every day and on which it is worth living.

Luuk Spreeuwers
Hengelo, October 2, 1992

Contents

1. Introduction

1.1 Neural networks, image filtering and performance evaluation

This work is concerned with the application of neural networks as image filters and with the performance evaluation of these image filters. Neural networks represent a relatively new method for data and information processing. Several approaches exist for image processing and pattern recognition with neural networks, but they are mostly concerned with target tracking, recognition of characters, associative memories that recall images, and modelling of the retina. The subject of the research presented in this work is the application of neural networks for image filtering, i.e. techniques for image restoration, image enhancement and feature enhancement and extraction. All these operations again yield an image, which is in some way better suited for further processing by human or machine vision systems.

The neural networks described in this work are all artificial neural networks, and are studied only for their technical possibilities and not for replication and functioning of biological neural systems. They have however in common with their biological counterparts, the basic idea of a network with many simple processors that are highly interconnected, and the behaviour of the network being determined to a great extent by the architecture and the connection strengths, instead of the function of the individual processors. Furthermore the artificial neural networks are trained by presenting examples. Because of their analogy with biological nervous systems, the properties of biological nervous systems often are also ascribed to neural networks. I.e. they would be able to learn from experience, and be able to find solutions for complex problems.

Neural networks reappeared in the scientific spotlight around 1980 after a period of approximately 10 years, during which the interest for the subject was minimal. In the early days of neural network research (1940-1969), simple networks like Rosenblatt's

perceptron were investigated as a new method for data processing [Rosenblatt 1959]. The research was finally quenched by a lack of mathematical theory and of computing power to perform simulations. The lack of fundamental mathematics and the limitations of these simple networks were illustrated most clearly by Minsky and Papert [Minsky 1969]. In the early 1980's a number of new mathematical developments emerged: Hopfield presented the Hopfield network [Hopfield 1982], Kohonen presented his self organising feature maps [Kohonen 1984], and Rumelhart et al. presented the error backpropagation learning rule [Rumelhart 1986]. Also the available computing power had increased enormously, allowing simulation of networks and training processes. These developments rekindled interest in neural network research. Currently several thousands of researchers are instrumental in the field of neural networks.

An important question that should be asked is why we should use neural networks. E.g. do they result in better solutions? Or do they enable us to find solutions to otherwise unsolvable problems? Of course it is clear that if a process is perfectly understood, an operation on the process can be designed that is in some sense optimal. In this case a neural network cannot yield a better result, at best it can approach equal performance. If, however, a simplified or incomplete model of a process is used to design a solution to a problem, using a neural network that is trained with examples of the desired behaviour, may yield better performance.
Also sometimes, even though a good model of a process is available, it may still be beneficial to use a method that can be trained with example data, if the desired operation is very complex.
Currently most successful applications of neural networks use relatively small networks (<10000 processing elements), and the abstraction levels of the information representations at the input and the output of the network differ not too much. E.g. a network is presented a binary pattern and recognises a character. As yet single networks cannot read and understand texts from printing directly. The scaleability of networks to realise these large transitions in the level of abstraction of information, is uncertain. An approach that is used in the Neocognitron [Fukushima 1988] is to use many subnetworks that each realise a small transition in abstraction level. Since in image filtering the difference in abstraction level between input image and output image generally is not very large, the problem of scaleability and combining a number of subnetworks will not be addressed in this work.

The use of many interconnected simple processing elements to realise or approximate a certain behaviour or function, has much in common with other general function approximation techniques, like e.g. polynomial and spline function approximations. Similarly, neural networks offer a multi-parameter function approximation to prob-

lems. The learning rules are the procedures to find the parameters. Also many of these techniques are, like neural networks, tuned using training data. If this comparison is valid, one would expect neural network solutions to have similar properties. Some of those properties are:

- Increasing the number of parameters, i.e. the number of processing elements and connections of a neural network, generally increases the accuracy of the approximation. However, using too many parameters can result in an under-determined system which can give worse results.
- Increasing the number of data samples generally increases the accuracy of the solution. If too few data samples are used, the solution may lose generalisation properties and become tuned only for the training set. This is called over-training.
- General applicability: these multi-parameter techniques can be used to realise many different types of functions.
- Generally reasonable or good performance: provided that there is a good method to obtain the required parameters, an accurate approximation of the desired behaviour can be obtained.
- There exist classes of problems for which the techniques do not yield acceptable solutions or require an excessive number of parameters for accurate approximation.
- Relatively simple design, because no modelling of the desired function is required. An estimation of the order of the problem, i.e. the required number of parameters to obtain an accurate approximation, suffices.

Focal points in theoretical research concerning neural networks, therefore, are the techniques to obtain the parameters (learning rules), estimation of the required size of networks and training sets and the applicability for different types of problems.

Benefits of the use of neural networks that are mentioned in literature are (see e.g. DARPA 1988]:

- Neural networks are able to learn from data, no modelling of the data is required in order to determine a solution to a problem. Increasing the amount of training data, increases the accuracy or completeness of the solution. Therefore neural networks are easy to design and general applicable. They may very well be able to provide acceptable solutions to problems that were not solved otherwise yet.

- Neural networks naturally utilise massively parallel computation, resulting in high speed data processing (provided that they are implemented in parallel hardware).

A disadvantage of using neural networks is that little is learned about the problem to be solved, only a solution is obtained. If instead much care is taken to model a process, insight of the processes is gained and an optimal solution maybe generated. Of course neural network solutions can be studied, but this is generally not a simple task, since neural network solutions are in someway distributed over the many parameters of the networks. Thus one should avoid to try to solve every problem with neural networks (neural should not mean **ne**ver **u**se a **r**easonable **al**ternative). If a good model can be obtained, this is generally preferable to data modelling using training sets. However, even if a process is well understood, the design of an operator can still be very complex. In these cases it can be beneficial to use function approximation methods that are tuned with training sets, like neural networks.

Image filtering operations often require complex mappings and the underlying models of the imaging processes are not always easily obtained. In many cases examples of the desired filter operations can be obtained or constructed. Therefore neural networks can be used for function approximation and the data modelling process. Also, since neural networks are naturally parallel systems, it should be possible to obtain high speed solutions if the neural networks are implemented in parallel hardware.
The basic idea of designing a neural network image filter is to construct a function in a neural network to map an image on another image in which certain properties of the image are extracted, recovered or enhanced. As an illustration consider the detection of boundaries of objects. To train a neural network with this objective, examples of the mapping must be presented to the network. The process is schematically illustrated in fig.1.1.

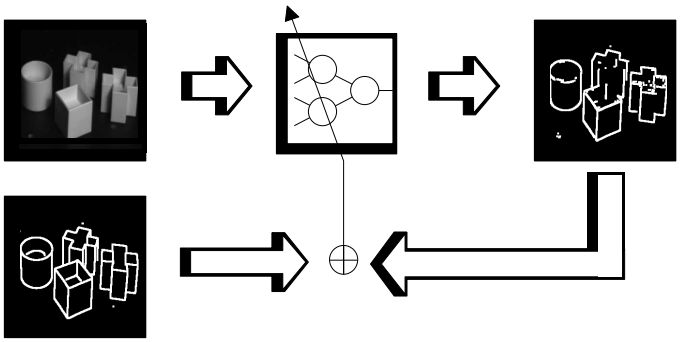


fig.1.1 Training of a neural network for image filtering.

In order to compare the performance of neural network image filters to other types of image filters, a quality measure for the output images is needed. Actually very little has been published on the subject of evaluation of image processing algorithms. In this work we propose to use an evaluation method that is based on a quality measure that is known in literature as the average risk. The average risk takes into account the probabilities on different types of errors and the respective costs of these errors. The lower the average risk, the better the performance of the image filter will be. A method for determining the average risk from test images will be presented. The basic idea is the following: a test image processed by the image filter is compared to an ideal output image. The probabilities on the different types of errors can be estimated from this comparison. This process of performance evaluation is schematically illustrated in fig.1.2.

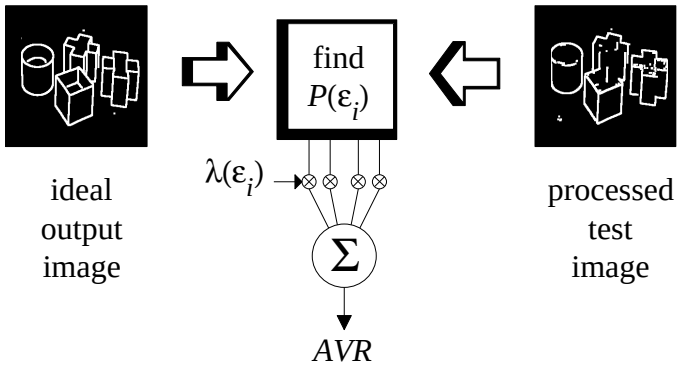


fig.1.2 Evaluation of image filters using average risk.

The correct way to design a neural network for a certain type of image filter operation, would be to train it for minimum average risk. It combines the two proposed techniques: neural network image filtering and performance evaluation with average risk. The result is an operator that is optimal for the application it is trained for (within the limits of the networks capability to approximate the desired behaviour accurately and the accuracy and completeness of the training data). This combination of image filter design with neural networks and evaluation with average risk is illustrated in fig.1.3.

1.2 The scope of this work

The scope of this work is in the first place to investigate and analyze the abilities of neural networks for image filtering. Several examples of neural network image filters are discussed, among which are image filters for edge extraction and image restoration filters for noise suppression and image sharpening. In the second place the performance of the obtained filters is to be evaluated. The average risk is proposed as a quantity to measure the quality, or actually its reverse: the lower the average risk of

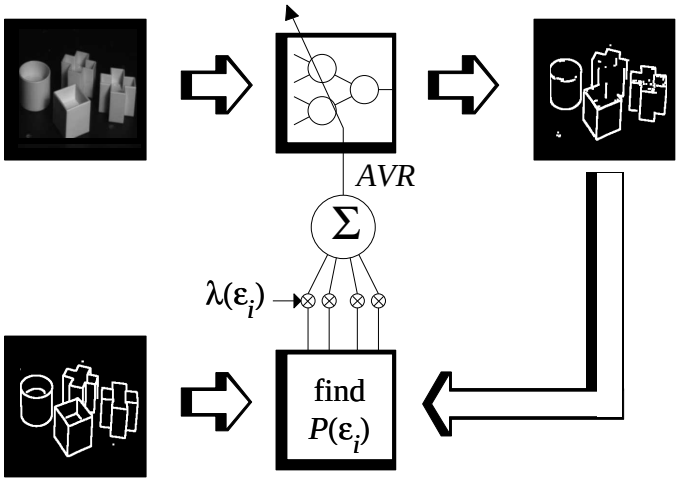


fig.1.3 Training a neural network for minimum average risk.

the operator under test, the better its performance. Quality measures to evaluate the performance of several image filters are worked out. Using the obtained quality measures, the performance of the neural network filters are compared to the performances of other types of image filters. As a conclusion the benefits and disadvantages of neural network image filters are formulated. Also a number of modifications to the used neural networks are proposed to improve the results. In particular a modification is proposed to optimise neural networks for the average risk. It is not the intention of this work to cover all types of neural networks and all types of image filters. Therefore throughout the work the presented methods are illustrated with a few examples, that should give the reader an idea of how to apply the methods. As a complete example, an error backpropagation network for edge detection is presented along with an evaluation method for edge detectors, based on average risk. Actually the concepts and techniques presented in this work originally were developed for edge detection, but they appeared to be applicable in a wider sense.
The concepts of neural networks and average risk evaluation methods developed in this work can be adapted for use in other research. They also create many new possibilities and challenges for further and new research.

1.3 Overview

The contents of this work are as follows. Chapter 2 begins with a short introduction into the fields of image analysis, image processing and image filtering. Specifically image analysis is considered as a measurement technique to reconstruct knowledge from images, i.e. recognition of imaged objects, obtaining parameters associated with the objects (e.g. size, position, colour, shape) or in some cases a description of the complete imaged scene. Image processing includes all possible operations on images. Image processing is broader than image analysis, because it also includes operations

that transform images, without the intention of further analysis, e.g. image compression and image coding. Image filtering is a generic name for image processing techniques that change the levels in images in order to enhance certain features or the appearance of images or the imaged objects. The output of an image filter is again an image that e.g. can be used and is better fit for further processing or analysis.

The next chapter (3) contains an introduction in the field of neural networks. A short general introduction on the subject is given first, followed by a somewhat more detailed description of a number of specific networks, of which the applicability as image filters will be investigated in the consequent chapters.

Chapter 4 is about how to obtain training images, to train neural networks for specific image filtering operations. Several different methods are considered including natural images, hand made training images and synthetic images based on 3D scene models and rendering techniques. These same methods can also be used to obtain test images for the estimation of probabilities on errors that are required to evaluate image filter operations with average risk.

In chapter 5 image filter applications with neural networks are considered. A number of examples for neural networks as image filters are given, including image restoration and edge detection with error backpropagation networks.

The performance evaluation of image filters is addressed in chapter 6. It contains a theoretical derivation of average risk, and describes how this can be used for the evaluation of image filters. An edge detector evaluation method based on average risk is given as a worked out example. A number of different types of edge detectors, among which of course the neural network edge detectors, is compared using this performance measure. Furthermore it is proposed to use the average risk as an optimisation criterion in the learning process of neural networks. For the error backpropagation network it is derived how to modify the learning rule to accomplish this.

Chapter 7 concludes with reconsidering the advantages and disadvantages of using neural networks for image filtering and the average risk evaluation method.

2. Image processing, analysis and filtering

2.1 Introduction

The systems in which image filters are applied can roughly be divided into two groups. The first group concerns image processing systems for image analysis. The object of an image analysis system is to reconstruct knowledge about the imaged scene from the image or images. The second group of image processing systems is concerned with storage, transportation and representation of image data. Some characteristic operations in the latter type of image processing system are image coding, compression and dithering. Image filtering techniques are used in both types of image processing systems. The concepts and examples in this work mostly apply to image analysis systems. However, most of the concepts are generally applicable.
Many image processing systems are designed in a rather adhoc way, without paying much attention to the underlying process models. In this chapter the model based design of image processing systems and subsystems is emphasized.
As an example of an image filter as a subsystem of an image analysis system, edge detection is considered more closely in section 2.4.

2.2 Imaging systems

2.2.1 Measurement with an imaging system

The objective of a measurement system is to reconstruct information about objects or events from measurements. A definition of measurement and information is given in [Finkelstein 1990]:

> *Measurement is the assignment of numbers or other symbols by an objective, empirical process to attributes of objects or events of the real world in such a way as to describe them. Information may be viewed by what is carried by a symbol about a referent by virtue of a defined relation the symbol bears to the referent. Measurement may thus be viewed as an information process.*

The description of objects and events in the real world requires models of these objects and events. Thus measurement always implies referencing a model (prof. Bosman):

> *Measurement is referencing a model*

The information carried by the acquired data (numbers and symbols) may differ depending on the interests of the observers, who define the relations of the numbers and symbols to the models. Information is therefore only defined for an apprehending mind or knowledge based system. My professor (prof. Bosman) defined information as follows:

> *Information is the body of acquired data which serves to enrich the state of knowing of an apprehending mind or knowledge based system. The same set of data can have different meaning dependent of the interests of the observers.*

The obtained measurements can be used e.g. to control a physical process. Generally a measurement system consists of one or more sensors that interface with a physical process, and a processing part. The processing part transforms the raw sensor data into the desired description of the state of the physical process.

The objective of an imaging system is to acquire knowledge about an imaged scene from an image (or images). An image analysis system is therefore a measurement system, that uses an imaging device as a sensor. Imaging devices measure radiation that is reflected or emitted by a physical process. For a detailed description of imaging devices, see e.g. [Ballard 1982]. A great advantage of using an imaging device as a sensor is that there is no direct contact with the physical process which means that the physical process is only minimally disturbed. Also since the sensor is at some distance to the physical process measurements can be obtained from processes that are difficult to approach or even hostile. Some measurements can only be obtained using imaging devices, e.g. images obtained from satellites (remote sensing). Furthermore several different types of measurements can be obtained from images, like position, size, velocity of objects, and spectral properties. Examples of applications are image processing systems for inspection of printed circuit boards, of fruit, of agriculture from

satellite images, and control of robots. Until now almost all image processing systems in industrial applications are rather simple and require extensive control of the illumination. Using carefully conditioned illumination can greatly simplify a number of problems that may occur. E.g. low contrast images can be avoided and structured light can be used to obtain depth information. Often this kind of conditioning is necessary because the extraction of the required information from images tends to be a very complex task.

2.2.2 From scene to image

In image analysis the physical process of which measurements are to be obtained, is called the *scene*. In the imaging device, a projection of this scene is formed. In the most commonly used imaging device, the CCD camera, this projection is on a flat two dimensional plane. The projection is called the image. In the imaging device, the distribution of the incoming luminous flux is measured as a function of the position on the projection plane. Most image processing is done using digital computers. Therefore the image data must be digitised. The positions are discretised (spatial discretisation) and the luminance levels are quantised. A detailed description of the projection and digitation processes can be found in e.g. [Ballard 1982], or [Pratt 1978]. Fig.2.1 illustrates the imaging process.

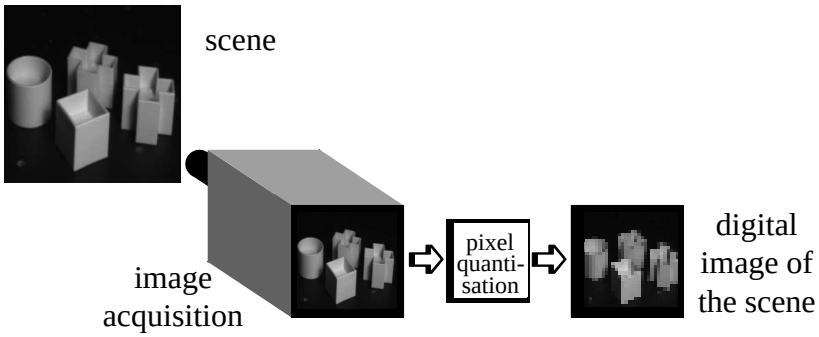


fig.2.1 From scene to digital image. The levels in the digital image represent the spatial luminous flux distribution on the image plane.

2.3 Image analysis

2.3.1 Description of the scene

The objective of an image analysis system is to reconstruct knowledge about the scene from the image data. This knowledge is a (partial) description of the scene. The description can be very simple, e.g. the presence of a rectangle in the image, or the number of red objects. But it can also be very complex, with a complete geometrical description of the scene and its illumination. The required description, of course, depends on the application of the image processing system, i.e. what the description of

the scene is used for. The description of the scene can be divided into three parts: the identification of the objects in the image, the geometric description and the radiometric (in the case of light a photometric) description. The geometric description includes the position, size and shape of objects, while the radiometric description includes the reflection properties of the objects and the light sources.
Thus:

$$\text{scene} = \{ \text{objects, geometry, radiometry} \} \qquad (2.3.1\text{-}1)$$

The above expression is valid for static scenes only. If the scene is dynamic, the descriptions of objects, geometry and radiometry become time dependent. Actually the image itself can be considered as a description of the scene, because it is a photometric description of a projection of the scene. Thus the task of an image analysis system can be considered as a translation of one description of the scene into another description. The latter can e.g. be used to control a physical process that is related to the scene. Figure 2.2 illustrates a measurement system, based on image analysis.

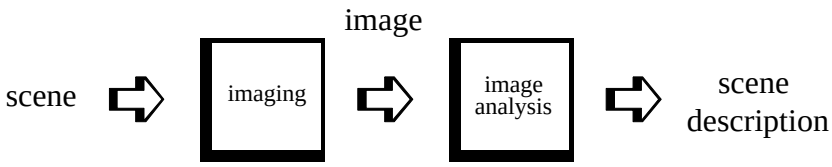


fig.2.2 Measurement system based on image analysis.

The next paragraph provides a closer look at the processing part, i.e. how to generate the desired description of the scene from the image data.

2.3.2 Model based image analysis, function and hypothesis testing approach

In order to design an image analysis system, one has to know how the (digital) image was generated from the scene (imaging model). In principle, a description of the scene could be obtained by a kind of inversion of this generation process. A more accurate description of the scene may be obtained if a more accurate model of the image generation process is used. As stated before, the scene itself can be described using a list of objects, a geometric and a radiometric model. Next, a model of the projection of the scene on the image plane is required and a model of the conversion of the incoming luminance flux distribution into the luminance level signal (generally an electrical cur-

rent or potential). Finally, in the case of digital image processing, a model of the digitisation process is required. Thus, the luminance level $L(x,y)$ at a position (x,y) in an image can be described (rather symbolical) using the following formula:

$$L(x,y) = D,S,P\{\text{objects,geometry,radiometry}\} = D,S,P\{\text{scene}\} \quad (2.3.2\text{-}1)$$

Where: $L(x,y)$ = the spatial luminance distribution in the projection plane
P = model of projection of scene on image plane
S = model of conversion to luminance level signal
D = model of digitisation process

The generation of images from models of objects, geometry, radiometry and projection, is covered by the discipline computer graphics for generating photo realistic images.
In image analysis the inverse operation is desired. To generate the description of the scene from the image data, eq.2.3.2-1 must be inverted:

$$\text{scene} = \{ \text{objects,geometry,radiometry} \} = (D,S,P)^{-1}\{ L(x,y) \} \quad (2.3.2\text{-}2)$$

Unfortunately, in the presented form this inversion generally is not possible. Firstly eq.2.2.3-1 may contain zero's, secondly very often multiple solutions exist, and thirdly because the inversion tends to be extremely complex if the scene is not very simple. In order to solve the inverse problem it must be simplified. There exist two ways to simplify the inverse problem. The first one is to only partly invert eq.2.3.2-1. Very often, a complete, detailed description of the scene is not required. One may be interested merely in a few characteristics of the scene. For example the presence of a brown blob on red objects (rotting spots on tomatoes). Although a complete inversion is generally not possible, a partial inversion of eq.2.3.2-1 may very well be possible. The second solution is that in many cases some knowledge of the scene is available, about its objects, geometry and radiometry, i.e. a model of (part of) the scene is available. This can help to direct the inversion problem to a single, existing solution. Again an example. Suppose it is known that the scene consists of a single cube, of a certain size, and full control over the illumination of the scene is possible. Using this knowledge the position of the cube in space can be estimated. Generally both types of simplifications are required to obtain a solution, i.e. preknowledge of the scene is needed to obtain a partial description of the scene.

The direct way to obtain an image analysis system is to try to design a function that realises or approximates the mentioned inverse. This approach is called the function design approach. The function should contain the (partial) inverse of equation 2.3.2-1, and if appropriate, a partial model of the scene. Although the models of eq.2.3.2-1 are

present in the function, they are generally not directly recognisable, i.e the models have become implicit. A practical problem for scenes that are not simple is the complexity of the required function. The different models are therefore often simplified in order to be able to find a solution to the inverse problem. Since the processing is always from the image (bottom) and directed to the scene description (top) this approach is a bottom-up approach.

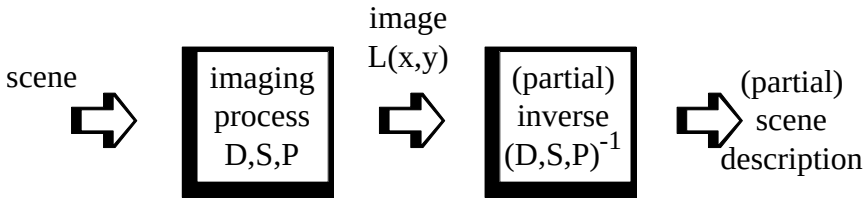


fig.2.3 Function design approach to image analysis.

The second approach to the inverse problem is the (iterative) hypothesis testing approach. Using the available models of scene, projection etc., and an initial guess of the unknown parts of the model, a hypothesis of how the image looks like is generated using eq.2.3.2-1. This hypothesis is compared to the real image, and the differences are used to adjust the initial guess. Using the adjusted guess of the unknown parts of the model, a new hypothesis is generated, tested etc. This iteration process continues until the differences between hypothesis and the real image become acceptably small (or some other stopping criterion becomes valid). The hypothesis testing approach is illustrated in fig.2.4.

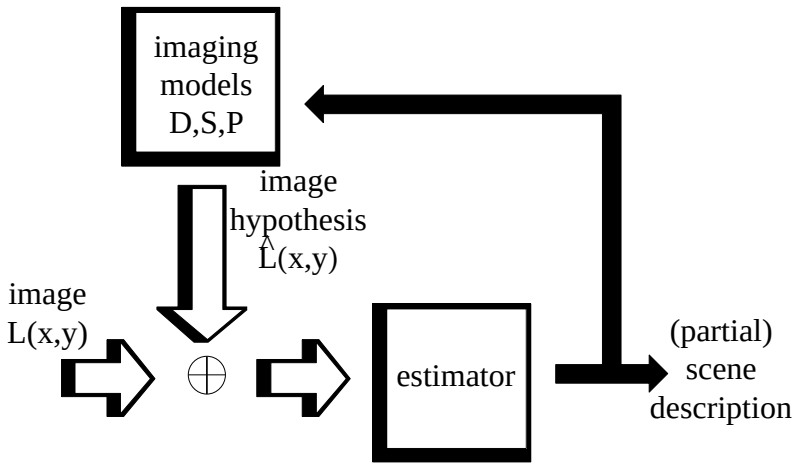


fig.2.4 Hypothesis testing approach to the inverse problem.

The advantage of using an explicit model of the generation of the image, is that improvements of the model are reflected directly in the accuracy of the resulting scene description. Although finding an inverse of the model is not necessary in this approach, a similar problem is present. The problem is to define a difference measure between two images, that can be used to obtain a better estimation in the next iteration. Of course, also in the hypothesis testing approach, the models can be simplified which results in faster processing on the one hand and less accuracy of the generated scene description on the other hand. Unlike in the function design approach, the processing of data is rather from the scene description, directed to the (hypothesised) image. The hypothesis testing method is therefore a top-down approach.

An interesting case, in which both approaches to the inverse problem meet, is a method called template matching. Whith template matching, a number of precalculated image hypotheses (templates) for characteristic scene descriptions, are compared to the actual image data. The scene description corresponding with the image hypothesis that closest matches the real image data, is accepted as an approximation of the required scene description. The set of templates with corresponding scene descriptions can be considered as a rough approximation of the inverse of the imaging model. The template matching method is therefore a function design approach. Since in the template matching method hypotheses are compared to the real image data, the template matching method is also a hypothesis testing approach (although it may not be iterative, since the comparisons to the image data can be done in parallel). In figure 2.5 the template matching method is illustrated.

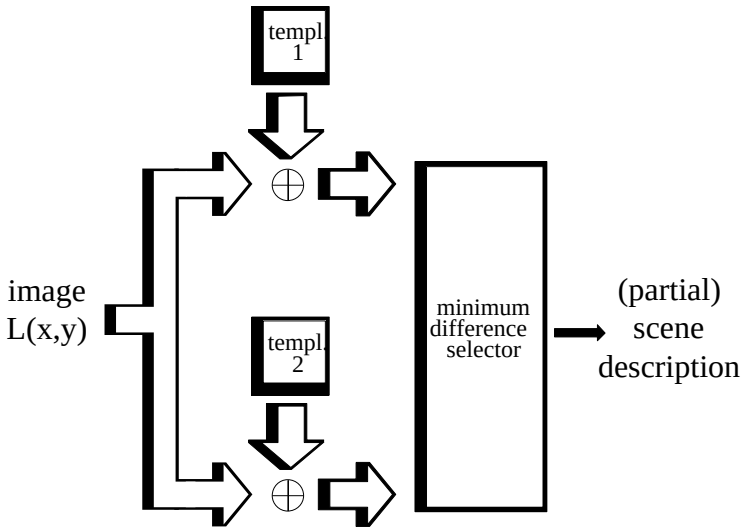


fig.2.5 The template matching method can be regarded as both a function design and as a hypothesis testing approach. The scene description that corresponds with the template most like the input image is selected.

2.3.3 Parametric, non-parametric and multi-parameter models

There are two ways to represent a model: parametric or non-parametric. In a parametric model a process is characterised by a functional description with a number of parameters, that are (more or less) directly related to physical quantities. It is often the objective of a measurement system to estimate these parameters from the measurement data. The advantages of using parametric models are the insight in the physical processes that is gained and the compact representation. The disadvantage is that it may be difficult and laborious to obtain an accurate functional description of the process.

In non-parametric models the behaviour of a process is not characterised by a functional description, but by a sample set of its input-output behaviour. A good example is classification with the nearest neighbour rule, using a labelled reference set. An input vector is assigned the class of the vector in the reference set that is nearest to it, according to the Euclidian distance measure. An advantage of non-parametric models is that they can often be obtained more easily than parametric models. A disadvantage is that little insight in the physical processes is gained and the sample sets tend to be large for accurate modelling of complex processes.

An intermediate form of these two representations is a model with many parameters. E.g. a complex functional description can be approximated by, and a sample set can be described using, general function approximation techniques, like e.g. spline or polynomial approximations or sinus functions as used in Fourier analysis. In nearest neighbour classification often clustering techniques are used. Although strictly speaking these are parametric descriptions, the parameters have no direct relation with physical quantities, and they are generally not regarded as parametric models. Advantages of these multi-parameter models are that they are can generally be obtained easily and provide a more compact description than non-parametric model (but less than a parametric model). A disadvantage still is that little insight is gained in the physical processes.

Similarly solutions can be parametric, non-parametric and multi-parameter solutions. Neural networks are multi-parameter solutions. They are trained either using large sample sets (non-parametric model) or with a limited number of templates (multi-parameter models).

2.3.4 Modular image analysis systems

Although there exist examples of the single step approach to the inverse problem of image analysis presented in the previous section (see e.g. [Korsten 1989], [Korsten 1990], [Graaf 1990] and [Feng 1991]), it is not very often used. Generally the image

processing system is divided into a number of subsystems. Advantages of this approach are that the processing in a subsystem is less complex. Furthermore for low-level processing, that is concerned with the image data directly and therefore with large amounts of data, special purpose hardware could be designed to speed up processing. As mentioned before, the image analysis system actually converts one description of the scene, the image, into another. In a modular image processing system, each subsystem converts (part of) the scene description into another description. Each subsystem therefore has its own level of describing the scene (or part of it). The lowest level is the image itself, the highest level is the desired description of the scene. Sometimes a number of basic levels of description is defined. Examples are Marr's "representational framework for deriving shape information from images" [Marr 1980] and the "four levels of abstraction" of Ballard and Brown [Ballard 1982]. The description schemes are approximately equal. Both schemes incorporate descriptions at the level of images and at the level of scene description, and distinguish raw and segmented images, with segments related to objects in the scene, and object and relational descriptions. Marr's "representational framework for deriving shape information from images" contains the following four levels of description:

- Image(s) - represents intensity
- Primal sketch - makes explicit important information about the two-dimensional image, primarily the intensity changes there and their geometrical distribution and organisation.
- $2^{1/2}$-D sketch - makes explicit the orientation and rough depth of the visible surfaces, and contours of discontinuities in these quantities in a viewer-centred coordinate frame.
- 3-D model representation - describes shapes and their spatial organisation in an object-centred coordinate frame, using a modular hierarchical representation that includes volumetric primitives (i.e. primitives that represent the volume of a space that a shape occupies) as well as surface primitives.

Comment: with "intensity" Marr means luminance.

The four "levels of abstraction" that Ballard and Brown distinguish are:

- Generalised images - images and image-like entities.
- Segmented images - images organised into sub-images that are likely to correspond to "interesting objects."
- Geometric structures - quantitative models of image and world structures.
- Relational structures - complex symbolic descriptions of image and world structures.

Depending on the application and the required type of scene description, these levels of description are more or less appropriate to image processing systems. Many intermediate levels of description may exist in an image analysis system, associated with the different image processing algorithms.
Segmentation of images generally is an important stage in image analysis. Segmentation is the process that partitions an image into segments based on certain features in the image. Since different features in the image are likely to relate to different objects or parts of objects in the scene, the segments are likely to correspond to "interesting objects."
Image filters operate on images and their output results are again images. They must therefore be classified to the first two description levels (for both schemes).

For each subsystem of an image analysis system the description of the scene at the input of the subsystem and the desired description at its output must be available, i.e. each subsystem requires its own model. A good example of such a local model, associated with a subsystem, is a model of unsharpness of the image caused by a defocused lens. A reconstruction algorithm could be designed to compensate the resulting unsharpness, using this model, see fig.2.6.

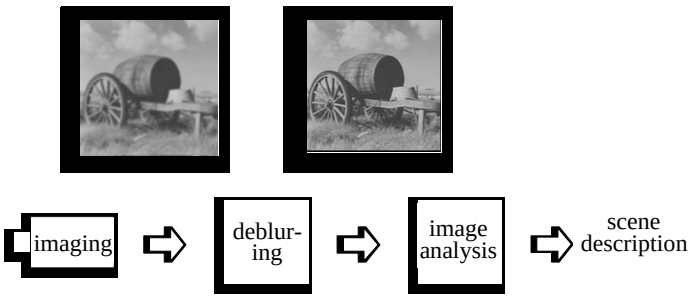


fig.2.6 A subsystem for compensation of image unsharpness in an image analysis system.

Incidently, the operation for compensation of unsharpness is an image filter operation. In subsequent sections the design of these operations is considered in more detail.

Each of the subsystems can be designed using the function design or the hypothesis testing approach. As an example consider an image analysis system, consisting of two subsystems, that is to estimate the position of a cube. It is illustrated in fig.2.7. Firstly, from the image of the cube the boundaries of the imaged cube are detected. Using these boundaries, the position of the cube is estimated. In the example, for the boundary detection, a function is designed using a model of how the boundaries of the cube

are projected on the image. The second stage uses a hypothesis testing approach to estimate the position of the cube, with a geometric model of the cube's boundaries (a kind of wire frame).

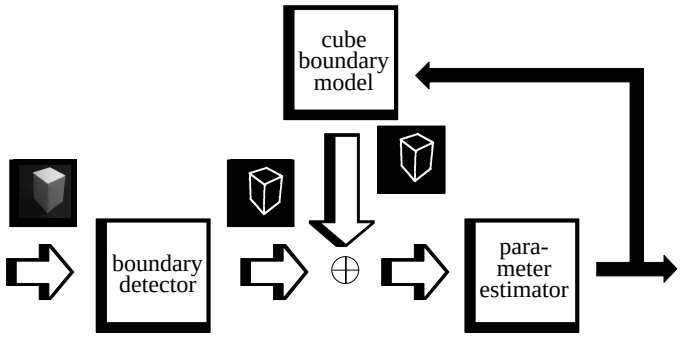


fig.2.7 Example of an image analysis system for estimation of the position of a cube, consisting of 2 subsystems.

The image analysis system uses both bottom up and top down processing, and three levels of description: the image itself, the boundary map, and a geometric description of the cube. Similar approaches to obtaining parameters of objects from images can be found in [Schrap 1989], [Romsom 1989] and [Lowe 1991].

2.3.5 A word about performance

It is surprising how little attention is given in literature to performance evaluation of image analysis systems and image processing algorithms. Methods that do exist for evaluation of image processing algorithms often try to define the performance of an algorithm in general, independent of the image and the application. We, however, would like to stress the dependency of the performance on the types of scenes and the application. To evaluate the performance of an image analysis system, first the demands on the scene description that it should generate must be considered. E.g. if only an accurate measure of the position of an object is required, the colour of the object usually is of less importance. It is important to find out what kind of errors the image analysis system makes, and the cost associated with each error. The cost associated with an error is determined by the application (in the above example: an error in the position estimation has a higher cost than an error in the estimation of the colour of the object). Another aspect is the probability (or frequency) of the occurrence of a certain error. If e.g. the image processing system is to detect spots, and the background is always white, it doesn't matter much if the system performs bad if the background is blue. The probability of the occurrence of a certain type of error is therefore related to the type of images the image analysis system should operate on. An error measure that in-

cludes both the cost of errors and the probability of errors is the average risk. The average risk is the sum of all the probabilities on errors, weighted with their costs (see e.g. [Devijver 1982]):

$$AVR = \sum_i P(\varepsilon_i)\,\lambda_i \qquad (2.3.5\text{-}1)$$

Where: AVR = average risk
$P(\varepsilon_i)$ = probability on error type ε_i
λ_i = cost of error type ε_i
the summation is over all error types ε_i

The best system is the system that has the lowest average risk. The average risk as described by Devijver and Kittler is used to evaluate and design optimal classifiers (resulting in the Bayes classifier). We propose to use the average risk in a more general way for evaluation and design of image processing systems and subsystems [Spreeuwers 1992].
For the evaluation of subsystems of image processing systems the same considerations are valid. However, the performance of a subsystem should be defined with respect to the performance of the complete system. Therefore, in addition an analysis should be performed of generation, accumulation and propagation of errors by image processing algorithms and in the remainder of the image processing system. Also, since the input of an image processing algorithm is not always an image, it may be more difficult to find the dependency of the performance on the types of images, i.e. the probability on errors. Performance of image processing algorithms and more specifically image filtering operations is considered in detail in chapter 6.

2.4 Image filtering

An important characteristic of image filter operations is that the result is again an image. A definition of image filtering for grey level images is given in [Ballard 1982]:

> *Image filtering is a generic name for techniques of changing image grey levels to enhance the appearance of objects.*

The level at each position in the filtered image is determined using the levels in the direct neighbourhood of the corresponding position in the original image. The simplest form of image filtering is, when only the level at the position itself is used. In grey level images, this is often called a grey level transformation. Two types of filters can be distinguished: spatially variant and spatially invariant filters. A spatially invariant filter performs the same operation at each position of the image. For a spatially variant

filter, the operation is position dependent. Most filters described in this work are spatially invariant filters. A linear combination of the neighbouring levels results in a linear filter and, in case of a spatially invariant filter, is equivalent with a convolution of the image with a certain convolution mask. A nonlinear combination of the neighbouring levels results in a nonlinear filter. The neural network image filters, described in this work are all nonlinear.

Image filtering has three major applications: image restoration, image enhancement, and image feature enhancement and extraction. Accurate parametric models of the underlying processes often are not available. An advantage of the neural network image filters presented in this work is, that they are designed using non-parametric models (examples of the desired input-output behaviour), which are generally easier to obtain than parametric models. Furthermore, although a parametric model may be available, it can still be difficult to design an appropriate filter operation if the model is complex. Designing image filters with neural networks generally is quite straightforward as is shown in chapter 5. An extensive treatise of image filters using parametric models can be found in [Pratt 1978]. The different image filtering applications are described in the subsequent paragraphs.

2.4.1 Image restoration

The first type of image filter operation is image restoration. The object of image restoration is to compensate for disturbances in the imaging process. A definition is given in [Pratt 1978]:

> *Image restoration may be viewed as an estimation process in which operations are performed on an observed or measured field to estimate the ideal image field that would be observed if no image degradation were present in an imaging system.*

In order to reconstruct from the distorted image data the spatial luminance distribution (the image field in Pratt's definition) of the ideal image, a model of the degradation is required. As an example, consider again the image that is degraded due to a defocused lens. Suppose the projection process can be separated in an ideal projection and a term that represents the distortion:

$$P = B,P_I \tag{2.4.1-1}$$

Where: P = a model of the projection process of the scene
B = a model of the blur caused by a defocused lens
P_I = the ideal projection process

An image restoration filter can be designed using the (parametric) model of the blur as follows (using eq.2.3.2-1):

$$L(x,y) = D,S,P\{scene\} = D,S,B,P_I\{scene\} \Rightarrow$$

$$(D,S,B)^{-1}L(x,y) = P_I\{scene\} \Rightarrow$$

$$L_I(x,y) = D,S,P_I\{scene\} = D,S,(D,S,B)^{-1}L(x,y) \qquad (2.4.1\text{-}2)$$

Where: $L(x,y)$ = the unsharp image
$L_I(x,y)$ = the ideal image

Fig.2.8 illustrates the image restoration operator for unsharp images.

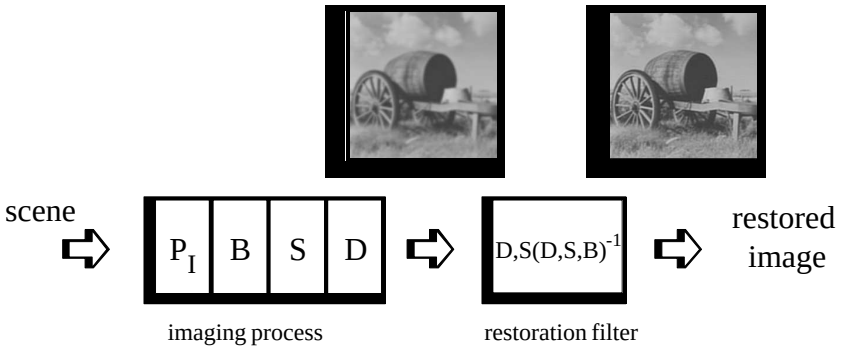


fig.2.8 Design of an image filter for restoration of an unsharp image due to a defocused lens. The model of the blur is denoted by B.

Again it should be noted, that if a model of the scene is available, the inversion of 2.3.2-2, can often be simplified (or first become possible). The disadvantage is that the resulting filter is not generally applicable anymore. Other examples of distortions are noise, due to the thermal noise in the imaging sensor and the digitisation process, and nonlinearities of the conversion of luminance into the luminance level signal. The latter can often be corrected by a grey level transformation. The former is more complicated, because noise is a stochastic process. As a result, the spatial luminance distribution is a stochastic process too. Only statistical properties of the luminance levels can be obtained, like expectance and variance. The easiest way to design a noise filter is to assume that the luminance level at a certain position in the image plane is equal to the expectation of the luminance of its surroundings. This approach however, distorts sharp level transitions. Better results can be expected if higher moments are included in the estimation of the luminance level, but this may result in a rather complex design. Using non-parametric modelling instead (like in neural network image filters) can greatly simplify the design of such image filters, as will be shown in the neural network image filter designs in chapter 5. Spatially invariant filters can of course only restore images with spatially invariant distortions.

2.4.2 Image enhancement

The aim of image enhancement is not, like in image restoration, to reproduce some kind of ideal form of the image. The objective of image enhancement is rather to produce images that are looking nicer, show more detail, or augment certain desired features. A definition of image enhancement is given in [Pratt 1978]:

> *Image enhancement processes consist of a collection of techniques that seek to improve the visual appearance of an image, or to convert the image to a form better suited to human or machine analysis.*

An example of image enhancement to improve visual appearance is creating overshoots at level transitions in images. These overshoots make an image look sharper. Incidently, the human retina uses a similar method for image enhancement. Another example is to improve the contrast of an image by grey level transformations. The effect of a type of grey level transformation that is called histogram equalisation is shown in fig.2.9. Histogram equalisation means a remapping of the grey levels so that approximately equally many pixels are in each grey level range of fixed size. This may considerably improve low contrast images like in fig.2.9.

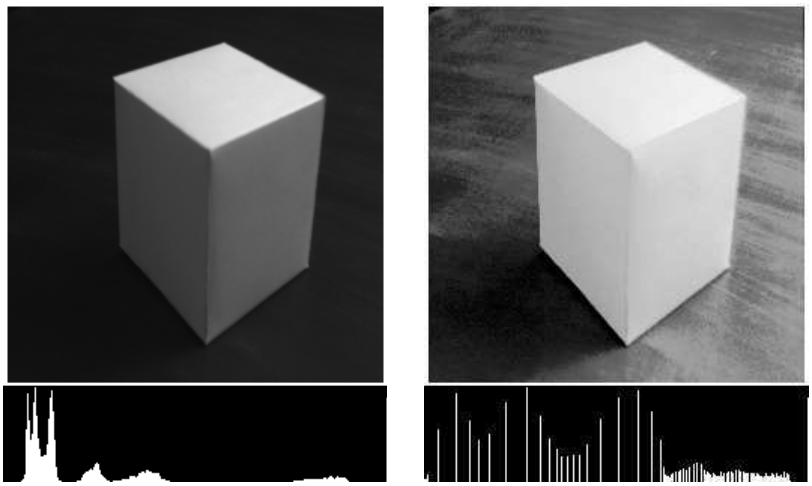


fig.2.9 Image enhancement using histogram equalisation. left: original image; right: enhanced image

In both cases the enhanced images are actually distorted, they only *look* more pleasing. An important application of image enhancement is to improve the readability of displays. Extensive research on this subject was carried out by Bosman et al. [Bosman 1990]. They also developed an engineering model of the human visual system in order to define image quality [Bosman 1989]. Image enhancement with the objective to convert the image into a form better suited for further analysis by machine, is very close

to information processing. An example is enhancement of level transitions, that may simplify an edge detection process. There is a large overlap with the feature enhancement and extraction operations described in the next paragraph.

2.4.3 Feature enhancement and extraction

The objective of feature enhancement and extraction is eventually to obtain primitives that are related to objects in the scene. These primitives can then be used in further analysis to reconstruct the information about the objects in the scene. The following definition of image features is given in [Pratt 1978]:

> *An image feature is a distinguishing primitive characteristic or attribute of an image field.*

Examples of image features are texture, grey level, colour etc. Based on the features an image can be segmented. The idea is that different objects (or parts of objects) induce different image features. The segments based on these features are therefore directly related to the objects. An example of a segmentation of an image based on the grey levels in the image is shown in fig.2.10.

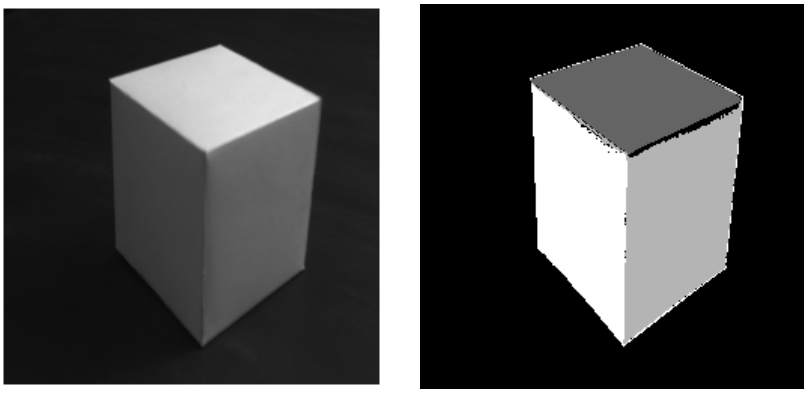


fig.2.10 Image segmentation using ranges of grey levels: left original image; right segmented image.

Simple segmentation methods like in fig.2.10 can only be used if illumination is well controlled and the objects stand out against the background. If this is not the case more sophisticated feature extraction and segmentation techniques are required.

2.5 An example: edge detection

2.5.1 The concept edge

Probably the most explored image features are edges and the corresponding operations edge detection and enhancement. Roughly speaking, edges are the boundaries between areas with different characteristics. The main reason for the interest in edges is that it seems likely, that from these boundaries the extent of imaged objects can be found. Because edge detection is used as an example of image filtering throughout this work, it will be described in somewhat more detail below. Many approaches to edge detection are more or less adhoc and partly or mainly based on heuristics of the phenomenon edge. We will try to remain as close as possible to the model based design strategy for image analysis described in this chapter.
Thus a model has to be found that relates changes in certain local image characteristics with objects in the imaged scene. To quote Pratt [Pratt 1978]:

> *Changes or discontinuities in an image attribute such as luminance (...) are fundamentally important primitive features of an image since they often provide an indication of the physical extent of objects in an image.*

To obtain information about the objects, abrupt changes in luminance are the most general applicable, because they almost always occur as a result of boundaries of objects. Changes in other image attributes like colour and texture are less general. Often an edge is defined as a change in the spatial luminance distribution. However, the changes in luminance do not always correspond to boundaries of objects in the scene. Hildreth, therefore, regards the physical changes that give rise to the luminance changes as edges. From this point of view the edges are not defined in the image plane, but in the scene [Hildreth 1983]:

> *A distinction is drawn between the detection of edges and intensity changes; a change in intensity is the phenomenon that we will detect and describe in the image; edges are physical changes that give rise to these intensity changes (...) in general, there may not exist a one-to-one correspondence between intensity changes detected at a certain resolution, and edges in the physical scene.*

(N.B. Hildreth incorrectly uses the term intensity here, it should read luminance.)
To avoid confusion, we will call the physical changes in the scene that induce luminance changes in the image *physical edges.*
Many "edge detectors" however, are only designed to detect grey level transitions in images, and not to characterise these in the physical changes that induced them. This also means that for the design only a model of grey level transitions in images is used.

This model can be local, i.e. it defines a transition in a small area in an image, or global, i.e. the model defines transitions as borders between regions. In [Ballard 1982], the local grey level transitions are called local edges:

> *A local edge is a small area in the image where the local grey levels are changing rapidly in a simple (e.g. monotonic) way. An edge operator is a mathematical operator (or its computational equivalent) with a small spatial extent designed to detect the presence of a local edge in the image function. It is difficult to specify a priori which local edges correspond to relevant boundaries in the image. Depending on the particular task domain, different local changes will be regarded as likely edges.*

It depends on the application what types of physical changes are required to obtain the desired description of the (objects in the) scene. E.g. if only the outlines of objects are required, it will generally suffice to detect the changes in surface reflectance. Other physical changes that give rise to luminance changes are e.g. illumination changes and changes in surface orientation.
The objective of an edge detector can be defined as follows:

> *The objective of an edge detector, designed for a specific application, is to detect only those luminance changes in an image, that are induced by physical changes of interest in the scene.*

According to this definition the edge detector detects phenomena in the image. We therefore propose the following definition of edges:

> *Edges are the luminance changes in an image that are induced by physical changes in the corresponding scene.*

2.5.2 Design of edge detectors

Not many existing edge detectors are designed according to the above considerations. Most of the literature on edge detectors is concerned only with the detection of luminance changes in images.
To detect luminance changes in an image reliably, a model of the luminance changes is required. As described earlier, the model can be used explicitly in a hypothesis testing approach, or implicitly by designing a function for the detection of intensity changes. The different approaches result in the basic methods, used for detection of level changes.

Hypothesis testing leads to the so called best fit methods. The method is to estimate the parameters of a model of a luminance change that fits best to the image data. Examples of this approach are finding the parameters of the best fitting step edge in a region in the image [Hueckel 1973], and fitting the spatial luminance distribution function by a polynomial [Haralick 1982,1984].
The results of a very clean and straightforward implementation of the hypothesis testing approach, fitting a five parameter model of a luminance transition locally to the spatial luminance distribution are shown in fig.2.11. The parameters used in the experiment are the base luminance level (*b*), the luminance difference (*h*), the steepness (*s*), the angle of the transition in the image plane (φ) and the shift (distance) of the transition relative to the centre of the area to fit to (*d*).

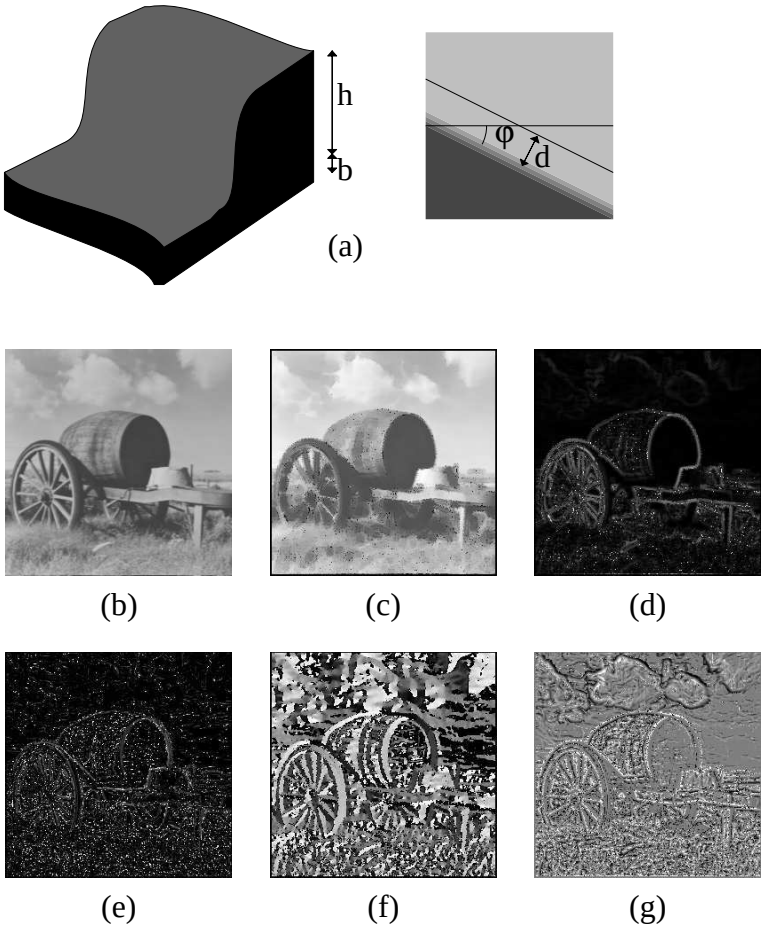


fig.2.11 Edge detection using the hypothesis testing approach. (a) five parameter model of local edge: base level, height, steepness, angle and distance; (b) test image; (c) edge base level map; (d) edge height map; (e) edge steepness map; (f) edge angle map; (g) edge distance map.

The model is used to generate hypothesis subimages of 5x5 pixels, fitting the image data for each pixel position. The least squares criterion was used as a criterion for the best fit.This results in an estimation for the five parameters for each pixel in the image. The resulting parameters are represented as images in fig.2.11. The edges can be detected by a combination of the parameters. E.g. an edge may be defined by a minimum height or steepness. while the position of an edge can be determined very accurately

by finding the zero crossings in the distance map. Thus a possible implementation of an edge detector would be to mask the height map with the zero crossings of the distance map and then threshold the result. Thresholding the height map directly could give thick edges, because level transitions may extent over more than one pixel.

In the function design approach, the objective is to find a filter function that responds to luminance changes in an image. Often the desired response is a local maximum in the output image at the position of the luminance change in the input image. The 'edges' can be detected by detecting the local maxima. The function design approach leads to filter designs, that are optimal with respect to a chosen criterion. Many of these optimal filter designs are based on a one dimensional model of level changes, that are extended later to operate on two dimensional images. An example of detection of level transitions in one dimensional signals is presented in [Canny 1986]. Canny uses a one dimensional step function with additive Gaussian noise as a model for the luminance changes. Detection of the level changes proceeds by filtering the signal with a linear filter that generates local maxima at the positions of level transitions. The local maxima are detected by masking the filtered signal with the zero-crossings of its derivative and thresholding the resulting signal. In Canny's design the filter is optimised with respect to good detection and good localisation, while it is also constrained to have a minimum distance between local maxima. The linear filter can be approximated by the derivative of a Gaussian psf. The Gaussian psf is used to smooth the image and suppress the noise. The width of the Gaussian (σ) depends on the noise in the edge model. The extension to 2 dimensional signals is realised by first finding the edge orientation from the gradient of the smoothed image. The resulting edge detection operator is shown in fig.2.12.

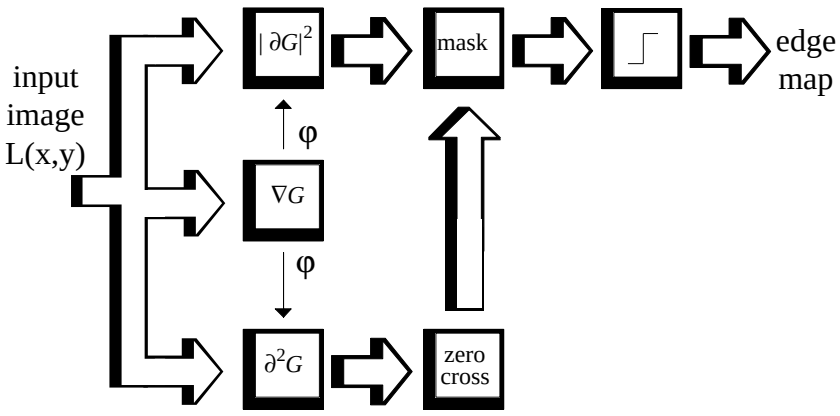


fig.2.12 Approach to edge detection according to Canny, using a linear filter that generates local maxima at the position of level transistions.

A similar approach is used by Marr and Hildreth [Marr 1980], except that no edge directions are determined, but the gradient of the Gaussian is masked with the zero crossings of the, in 2 dimensions symmetrical, Laplacian of a Gaussian.

A critical aspect of the design of these filters is of course the model of the level transition that is used. Van der Heijden [Heijden 1992] correctly points out that in his edge model Canny uses an infinite observation interval for a single level transition. Van der Heijden uses a more realistic model for multiple transitions using a Poisson impulse process [Heijden 1992]. He also points out that the fixed structure of the Canny operator, using a linear filter, constrains the solution. Van der Heijden models the luminance changes in terms of conditional covariance matrices, i.e. luminance changes are regarded as a stochastic process with a Gaussian probability density function. Thus the Bayesian approach for detection problems can be used, by minimisation of expected costs (average risk). The costs are assigned to detection and localization errors. The weak spot in his design is in the assumption of Gaussian probability density functions, as van der Heijden points out himself. The structure of the covariance matrix (cvm) operator is shown in fig.2.13.

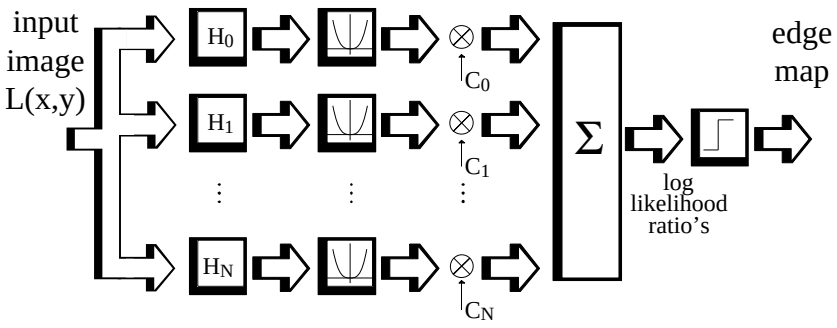


fig.2.13 Computational structure of the covariance matrix operator of van der Heijden.

The cvm-operator consists of a bank of linear filters. The output of the filters is combined, resulting in the log-likelihood ratios of level transition and not a level transition. The local maxima of the log-likelihood ratios correspond to level transitions. Contrary to Canny's approach, the approach of van der Heijden allows direct optimisation of two dimensional image filters.

Template matching is also used in edge detection. The idea is that the template is an ideal case of the feature that is to be detected (in this case a luminance change). From a number of templates, the one that best fits the image data is chosen. Examples of template matching used for local edge detection are the Sobel and Kirsch operators [Pratt 1978]. The Sobel templates are given by:

1	2	1	-1	0	1
0	0	0	-2	0	2
-1	-2	-1	-1	0	-1

The best match for these operators is found by correlation of the templates with the local image data. The result is a local maximum (positive or negative) for respectively primarily horizontal (first template) and vertical (second template) oriented luminance changes. In order to obtain a simple output signal with local maxima at the positions of luminance changes, the results of the correlations of the templates and the image data are usually squared and added together. Note that the result is again a structure that looks like the previous operations: a linear filter operation (correlation) followed by a local maximum detection process. It will be shown later that feed forward neural networks for image filtering have a very similar structure, which can help with the analysis of the neural network image filters.
Only a few of the many existing edge detection operations were presented here. Detailed overviews and evaluations of many different edge operators can be found in [Blicher 1982] and [Heijden 1992].

All of the above approaches to edge detector design are using parametric models of luminance changes. The disadvantage of these models is in the first place that it is rather difficult to obtain accurate models for luminance changes that are induced by physical edges of interest. In the second place it is often difficult to design, based on these models, an operator that can detect these luminance changes. In order to be able to design an operator, the models are often simplified and the solutions are constrained by adopting a simple basic structure for the operator.
On the other hand non-parametric models and solutions could be used for edge detection. This requires a large set of examples of local spatial luminance distributions of images representative for the application. The class, edge or not-edge of the example luminance distributions must be known. The advantages of this approach is that the relation to physical edges is implicitly captured and general multi-parameter function approximation techniques can be used to obtain a solution (e.g. the neural network image filters described in this work). The first advantage is only partly an advantage, because the problem is shifted to obtaining a good quality example set. Still, in our opinion, this is often easier than constructing a complete and accurate model of physical edges and their projection on the image plane. As mentioned earlier, the disadvantage of this approach is that little insight is gained in the problem, only a solution is obtained.

3. Neural networks - an introduction

3.1 Introduction

Artificial neural networks are processing structures that are inspired on the architecture and functioning of the human nervous system (see e.g. [Kuffler 1976], [Rumelhart 1986] and [Kohonen 1988]). Two different groups of researchers are investigating artificial neural networks. The first group are biologists and psychologists who want to study the functioning of the human nervous system. They build artificial neural networks to resemble subsystems of the human nervous system and try to find out what these networks do and how they function. The second group of researchers investigate if and how neural networks can be used for information processing in technical systems. Their objective is to design neural networks for certain applications. These networks generally have only the basic ideas in common with the human nervous system. This work is only concerned with the second approach to neural network research.

Basically neural networks consist of many, highly interconnected processing elements (neurons). Each of these processing elements performs a relatively simple task: collecting the output from other processing elements or sensors, to which it is connected, and producing an output that is in some way related to the total input. This output can be input for other processing elements, or an output of the network. All processing elements operate simultaneously. A processing task is distributed over many processing elements. The function of the network is not so much determined by the processing elements as by the connection structure and how the connections transfer signals. In fact the processing elements perform basic functions on the signals and the connections determine how these basic functions are combined to the total function of the network. Thus a complex function is realised by combining many simple functions. This is very similar to other general function approximation methods like e.g. spline fitting and Taylor series. It can be proven that most complex problems can be solved using combinations of relatively simple functions [Kolmogorov 1957]. Therefore it is likely that

a neural network, consisting of many relatively simple processing elements, can be used to solve complex problems. Since all these elements operate in parallel, information processing in neural networks will be very fast too.
This chapter contains a brief general introduction on neural networks, followed by a more detailed description of the networks that were used as image filters in this work. These networks also happen to be the most popular and successful ones in other neural network research. For a more complete introduction into this field refer to [Lippmann 1987], [Karna 1989] and [Rumelhart 1986]. An overview of the history of neural network research can be found in [Simpson 1987].

3.1.1 Processing elements

A very simple model for a single processing element was proposed by McCulloch and Pitts in 1943 [McCulloch 1943]. In their model the collection of the input signals is a summation of binary output signals. The output of the processing element is also binary: 0 if the sum is below a certain threshold, and 1 if it is above the threshold. The connections have a connection strength with which the input signals are weighted. This results in the following equation for the output of a processing element:

$$y = f\left(\sum_i w_i x_i\right) \qquad (3.1.1\text{-}1)$$

Where: y = the output of the processing element

$f()$ = the threshold function: $f(s) = \begin{cases} 0, & s < \theta \\ 1, & s \geq \theta \end{cases}$

w_i = the weight of connection i to the processing element
x_i = the input signal of connection i
θ = the threshold level
the summation is over all connections i to the processing element

The McCulloch-Pitts model is illustrated in fig.3.1.

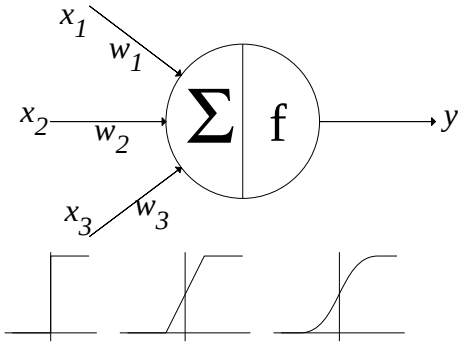


fig.3.1 McCulloch - Pitts model of a neuron, and several different transfer functions that are used.

The basic idea of a weighted summation of inputs with a threshold that determines the output, is still used in most of the neural networks. Many variations on this basic model exist however. Some networks operate on continuous input signals and use a different function to calculate the output from the weighted sum. The best example of such a different *transfer function* is probably the semilinear sigmoid function used in error backpropagation neural networks. A number of different transfer functions is drawn at the bottom of fig.3.1. With linear transfer functions, only linear functions can be realized, because a linear combination of linear functions again results in a linear function. The transfer functions are therefore nonlinear. Another variation is how the input signals are combined. Some networks do not use a summation of the inputs, but a multiplication (PI-units). In Kohonen networks the connection weights are not multiplied with the input signals, but the Euclidian distance of the input vectors to the weight vectors is calculated.

As mentioned in the introduction, the calculating capabilities of a single processing element are rather limited. Consider e.g. a processing element with two inputs that are combined using a weighted summation and a threshold transfer function (this type of units is sometimes called linear threshold elements). The two dimensional input space is separated into two regions by a line.

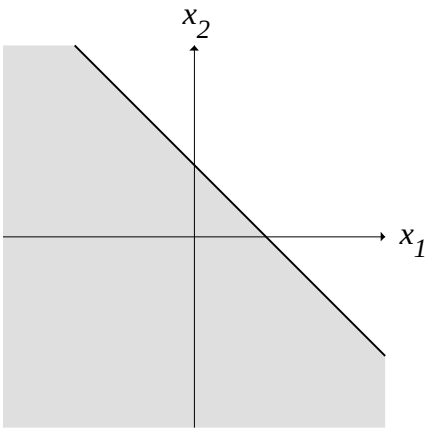


fig.3.2 Linear separation of input space by a linear threshold element. The orientation of the separation line is determined by the weights w_1 and w_2, and the shift by the threshold level θ.

The line is found by setting the summation in eq.3.1.1-1 to zero. This linear separation of the input space is the basic function of this type of processing elements. By combining many of these simple basic functions, complex functions can be realised. This is described in the next paragraph on network architectures.

3.1.2 Network architectures

Neural networks architectures can be divided in architectures that strictly feed forward, network architectures with sparse feedback, and recurrent networks, where all processing elements are connected to all other processing elements. Usually, the pro-

cessing elements in neural networks are ordered in layers. These layers themselves can again be strictly feedforward, have sparse feedback (both from other layers and from the layer itself) or be recurrent. The following types of layers are distinguished:

- Input layer; the input of the network is presented to the processing elements of this layer.
- Hidden layers; these contain processing elements that only connect to other processing elements.
- Output layer; the outputs of the processing elements in this layer forms the output of the network.

In single layer networks the input and output functions are combined into the one layer. The basic network architectures are illustrated in fig.3.3.

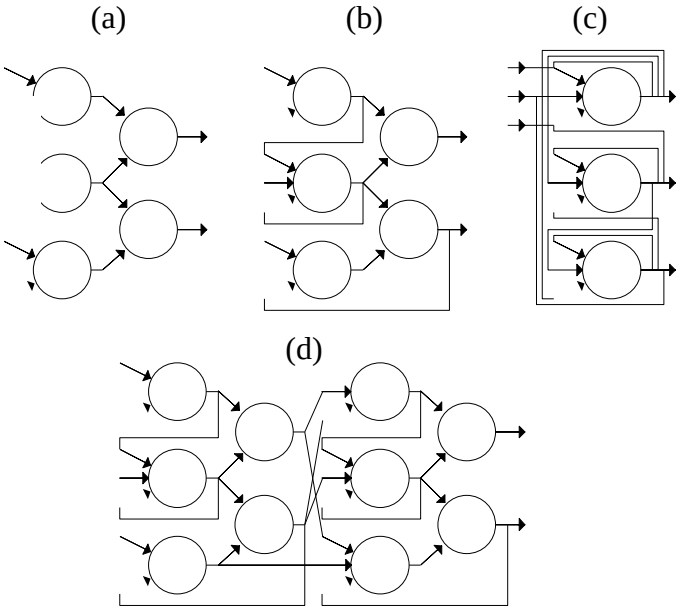


fig.3.3 Network architectures (a) feed forward, (b) sparse feedback, and (c) recurrent network architectures; (d) layered network with input, hidden and output layers.

To illustrate the realisation of complex functions in neural networks, consider feedforward networks consisting of a single, two and three layers and a single output and linear threshold elements. The single layer network in this case is actually a single processing element, which can be used to realise a linear separation. In a two layer network several of these linear separations can be combined into a more complex function (see fig.3.4). Arbitrary partitioning of the input space can be realised with a three layer network. Adding more layers may improve efficiency. In fig.3.4 this partitioning of the input space is illustrated.

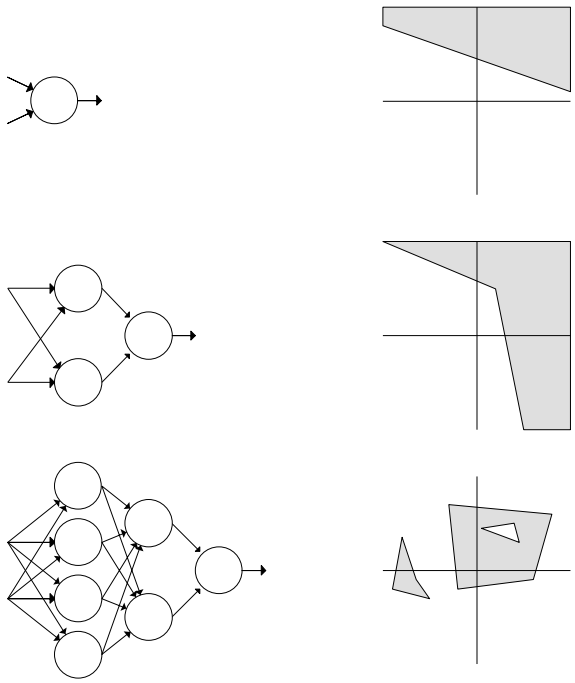


fig.3.4 Partitioning of the input space with one, two and three layer feed forward networks with thresholding elements. A single layer network can only make linear separations, a two layer network partitions the input space in convex regions; a three layer network can realise arbitrary partitioning.

If instead of thresholding elements, processing elements with carefully selected continuous nonlinear transfer functions are used, any continuous nonlinear function of the inputs can be realised using a three layer network. This can be demonstrated using a theorem that was proven by Kolmogorov and is described in [Lorentz 1976]. More recently this was shown for sigmoid transfer functions [Cybenko 1988].
Introducing feedback adds to the complexity of the network. In recurrent networks the outputs of all processing elements are fed back to all processing elements. As a result the only stable output patterns are those that generate the same pattern if used as input. The stable patterns of a network are determined by the weights of the connections. The maximum number of stable patterns is determined by the size of the network (see e.g. [Abu-Mostafa 1985] and [McEliece 1987]).

3.1.3 Learning

In order to realise a certain desired function in a neural network, the network architecture and the weights of the connections must be determined. The first step is to find a network architecture in which the function can be realised or approximated with sufficient accuracy. The next step is to find the appropriate weights of the connections. The procedure for finding the appropriate weights is called the learning rule in neural network research. It actually means finding the parameters of the multi-parameter function approximation method that is called neural network.

A characteristic of neural networks is that they are trained using examples of the desired behaviour. The idea is, that if the network can produce the correct output for the examples, it will also generate the correct output for real data. Two important observations can now be made. Firstly, the training of a neural network to perform a certain behaviour corresponds to the function design approach. Secondly, the model of the desired behaviour of the network, is contained in the example set. It is therefore a non-parametric model. The example set can be either a large set of arbitrary training examples or consist of a small number of ideal examples (templates). It depends on the type of network and learning procedure which types of training sets can be used.

The following two basic types of training can be distinguished:

- supervised training
- non-supervised training

Supervised training means that if an example pattern is presented to the network, the output, generated by the network is in some way evaluated (by the supervisor) and some corrective signal is provided, so that the network can adapt. The corrective signal, that is fed back to the network, can e.g. be the target output of the network or a measure of quality of the network output. In non-supervised learning, no corrective signal is provided. In this case, if a pattern is provided to the network, it is evaluated using the present network state. This network state is either reinforced, if the new pattern fits the present network state (e.g. matches previously presented patterns closely), or, if this is not the case, the network state is altered to incorporate in some way information of the new pattern. The network's internal evaluation of the pattern can be done in a specialised part of the network, or can be integrated in the network. The supervised and non-supervised training processes are illustrated schematically in fig.3.5.

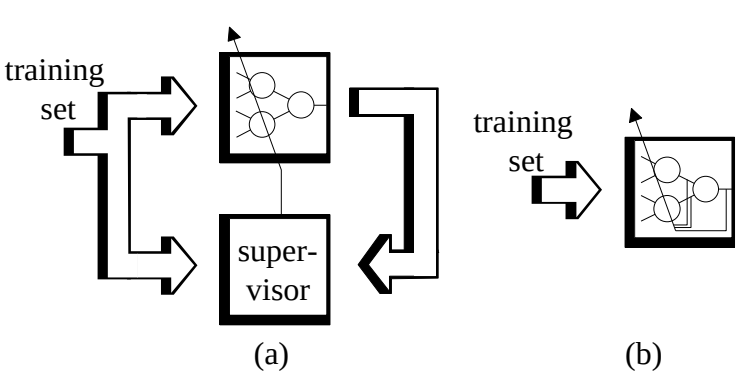


fig.3.5 Supervised (a) and non-supervised training (b) of neural networks.

A second distinction can be made in learning mechanisms:

- parallel training
- sequential training

Parallel training means that the whole training set is examined at once to determine the weights of the network. In sequential training the training patterns are presented sequentially to the network and the weights are adapted after each presented training pattern.

Each type of network uses a different learning rule. The actual learning rules are therefore treated in the concerning sections about the network type.
The following networks are described in the subsequent sections:

- Perceptron network
- Error backpropagation network
- Hopfield network
- Kohonen self organising feature map

These are the networks that are considered for image filtering in this work. Descriptions of other types of networks (like ART, Boltzmann, Hamming and Brain-state-in-a-box networks) can be found in e.g. [Lippman 1987], [Karna 1989] and [DARPA 1988].

3.2 The perceptron network

3.2.1 Network architecture

Of these networks the perceptron is the simplest, and also the earliest network [Rosenblatt 1959]. The perceptron was originally intended to model and investigate simple visual pattern recognition in the human visual system (the name perceptron is derived from *perception*). The perceptron is a feed forward network, consisting of three layers. The input consists of binary visual patterns. The first layer acts as an input buffer. The processing elements in the second layer act as feature detectors. They are either fully or randomly connected to the input layer. The weights of these connections are fixed and either random or assigned to detect certain desired features. The third (and output) layer consists of *perceptrons* or pattern recognisers that combine the features of the second layer to recognise patterns. These are linear thresholding elements as described in the previous section. The network architecture is illustrated in fig.3.6.

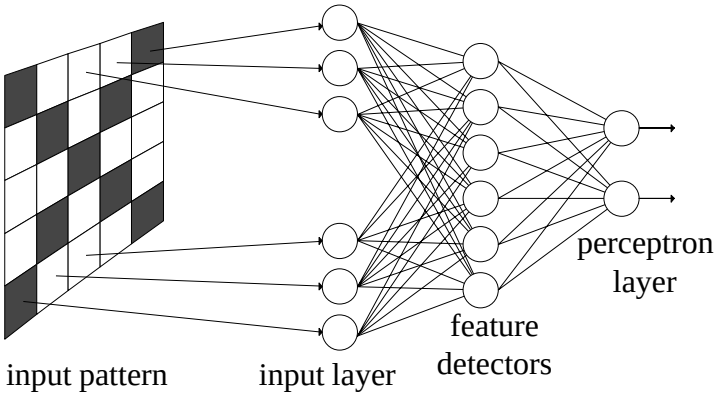


fig.3.6 The perceptron network architecture.

3.2.2 Perceptron learning

During learning only the weights of the perceptrons in the output layer are updated. The perceptron uses a supervised sequential training algorithm. The basic learning rule is as follows:

- If the output is correct, the weights don't change
- If the output is 0 but should be 1, increment the weights of the connections that are active
- If the output is 1 but should be 0, decrement the weights of the active connections

Here active connections are connections that connect to outputs of processing elements that are 1 (high). Inactive connections carry a 0 (low) signal. It can be proven that if a set of weights exists that realises the desired function, it can be found using the above learning rule [Rosenblatt 1962].

3.2.3 Limitations of the perceptron network

A severe limitation of the perceptron network is that the perceptrons in the output layer can only realise linear separations in the feature space of the second layer. Minsky and Papert performed a thorough analysis of the capabilities of perceptron networks and perceptron learning [Minsky 1969]. They showed that many problems cannot be solved using perceptrons, because of their single adaptive layer. The limitations of a single layer feedforward network were already illustrated in fig.3.4.
The solution to these problems is to construct networks with multiple adaptive layers. These networks are therefore often called *multi-layer perceptrons* or *generalised perceptrons*. Unfortunately the learning rule becomes more complex, because only an

error measure for the output layer is directly available. The first learning rule that really handled adaptation of multiple layers, is the error backpropagation learning rule. It is described in the next section.

3.3 Error backpropagation network

The error backpropagation network is a feed forward network like the perceptron. Unlike the perceptron learning rule, the error backpropagation learning rule can be used to adapt multiple layers. By propagating the error backwards through the network, a local error measure is obtained for each processing element. This local error measure is used to adapt the weights.

The error backpropagation learning rule is one of the few learning rules for neural networks that has a mathematical framework. It was published by Rumelhart, Hinton and Williams, in 1986 [Rumelhart 1986]. We are of the opinion that Rumelhart et al. skipped an important step in their mathematical derivation of the error backpropagation learning rule. Therefore, the derivation of the learning rule is given in full in appendix A. In chapter 6 a modification of the error backpropagation is proposed for minimisation of average risk.

3.3.1 Processing elements and network architecture

A processing element in an error backpropagation network calculates a weighted summation of its inputs. The inputs can be either inputs of the network or outputs of other processing elements. Since the error backpropagation network is a feed forward network, these outputs can only be from processing elements (PEs) in lower layers. In the derivation of the learning rule, it is assumed that only connections exist from a layer to the next layer (no connections jumping over layers). The weights are assigned to the connections from outputs of PEs to PEs in the next layer. The output of a PE is a non-linear function, the so-called transfer function, of this weighted sum. For an input pattern x^p of the network the output of PE m in layer k becomes:

$$o_k^{pm} = f(net_k^{pm}) = f\Big(\sum_n w_k^{mn} o_{k-1}^{pm}\Big) \qquad (3.3.1\text{-}1)$$

Where: o_k^{pm} = the output of PE number m in layer k for an input pattern x^p
$f()$ = the transfer function
net_k^{pm} = the weighted sum of the inputs of the PE for input pattern x^p
w_k^{mn} = the weight of the connection between PE n in layer k-1 and PE m in layer k
the summation is over all PEs n of layer k-1

In fig.3.7 the architecture a four layer error backpropagation network are shown.

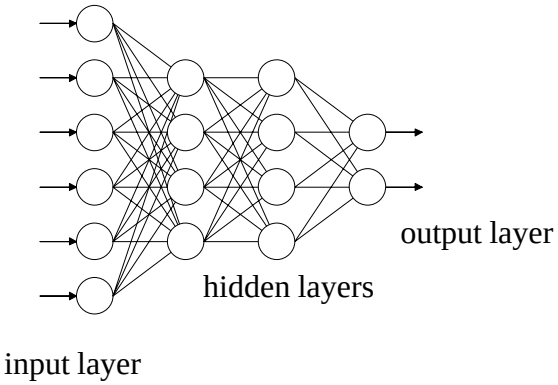


fig.3.7 Error backpropagation network with two hidden layers. The input layer only distributes the inputs to the next layer.

Generally the transfer function of processing elements in an error backpropagation network are sigmoid transfer functions:

$$f(x) = \frac{1}{1+e^{-x+\theta}} \qquad (3.3.1\text{-}2)$$

This function acts as a soft threshold with the centre of the slope at θ. Often θ is implemented by adding an extra connection with weight –θ to a processing element with a fixed output 1.

3.3.2 Optimisation criterion and method

The error backpropagation learning rule attempts to minimise the sum of the square errors of all patterns. The system error is defined as:

$$E = \frac{1}{2}\sum_p E^p = \frac{1}{2}\sum_p \sum_i (t^{pi} - y^{pi})^2 \qquad (3.3.2\text{-}1)$$

Where: E^p = the square error of the output for input pattern x^p
t^{pi} = the target of output i for a training input pattern x^p
y^{pi} = the actual network output i for training input pattern x^p
the summations are over all outputs i and training input patterns p

If the system error is zero, all training patterns are mapped on the correct target output pattern. Since the behaviour a network with a certain architecture is defined by the connection weights, for a given training set E is a function only of the connection weights. The error backpropagation learning rule is an iterative learning rule that attempts to minimise E by adapting the weights. This is done using a gradient descent method, of which derivation is given in appendix A.
For the weights in the output layer (layer L) the weight updates can be calculated directly:

$$\Delta w_L^{ij} = \sum_p \eta \delta_L^{pi} o_{L-1}^{pj} = \sum_p \eta (t^{pi} - y^{pi}) f'(net_L^{pi}) o_{L-1}^{pj} \qquad (3.3.2\text{-}2)$$

Where: w_L^{ij} = the weight of the connection between PE j in layer L-1 and PE i in the output layer L

η = a positive constant that controls the rate of change of the weights (the learning rate)

δ_L^{pi} = a kind of local error measure for PE i of layer L, due to pattern x^p

o_{L-1}^{pj} = the output of PE j in layer L-1 for input pattern x^p

$f'(net_L^{pi})$ = the derivative of the transfer function f in net_L^{pi}

The local error measures (δ's) for a processing element m in a hidden layer k can be determined recursively by:

$$\delta_k^{pm} = \sum_l \delta_{k+1}^{l} w_{k+1}^{lm} f'(net_k^{pm}) \qquad (3.3.2\text{-}3)$$

The expression for the updates of weights in a hidden layer is:

$$\Delta w_k^{mn} = \sum_p \eta \delta_k^{pm} o_{k-1}^{n} \qquad (3.3.2\text{-}4)$$

3.3.3 Sequential training

The weight updates for the error backpropagation learning rule as described in the previous paragraph require the whole training set to be processed for one update of all connection weights in the network (parallel training). This is however not the way the error backpropagation rule is generally used. In most applications the weights are updated after the presentation of *each* training pattern (sequential training). The weight update for a pattern x^p becomes:

$$\Delta w_k^{pmn} = \eta \delta_k^{pm} o_{k-1}^{n} \qquad (3.3.3\text{-}1)$$

If the individual weight updates are small relative to the weights themselves, the effect will be approximately the same as for the cumulative update of eq.3.3.2-4. If they are not, an oscillation effect may occur if a number of successive training patterns result in more or less opposite weight updates. Experiments show that in the general case when the weight updates are small (η sufficiently small), sequential and cumulative weight updates yield comparable solutions.

3.3.4 Advantages and disadvantages of error backpropagation

The error backpropagation learning rule only works correct if the system error is decreased in small steps, using a small learning rate. A disadvantage is that the network often converges very slowly to the solution. Generally the whole training set has to be presented to the network many times to reach a good solution. Attempts have been made to speed up learning with error backpropagation. The first attempt came from Rumelhart et al. themselves [Rumelhart 1986]. They introduced a momentum term that adds a fraction of the previous weight update to the new update. The functionality of this momentum variant has not been proven mathematically. Though it appears to work well in their XOR example, in many problems the introduction of the momentum term even results in slower learning and oscillatory behaviour of the system error during learning. For networks larger than a few processing elements we found that the best choice for the momentum term was 0, while the learning rate should be chosen $\eta < 0.1$. We found that a very simple way to speed up learning is to update the weights only if the output error for a pattern exceeds a certain threshold. This may not reduce the number of iterations required, but reduces calculation times considerably. Many other variations on the error backpropagation learning rule are investigated, but most are only successful for certain problems.
An unsolved problem is the choice of the number of layers and processing elements. No general rule exists to determine the required network size for a certain application. In most applications the network size is chosen in a heuristic or empirical way.
A great advantage of the error backpropagation network is its general applicability. It can be used for both binary and continuous mappings, and it can be trained either with a large set of arbitrary training samples or with a limited number of templates. The learning rule appears to give reasonable upto good results for many different types of problems.

3.4 The Hopfield network

3.4.1 Network architecture and processing

The Hopfield network is a recurrent network that consists of a single layer of processing elements. The output of each processing element is fed back to all other processing elements of the network, see fig.3.8.
The output of a processing element in a recurrent network not only depends on the input of the network, but also on the network state, i.e. the outputs of the processing elements. Recurrent networks are therefore dynamic systems. Most recurrent networks are nonlinear and discrete time systems. For discrete time systems the state at a time

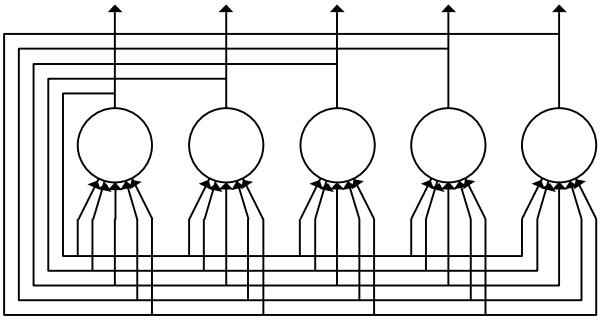


fig.3.8 Network architecture of a Hopfield network: each PE is connected to all other PEs.

t+1 is determined by the state at time t and the system input at time t. In Hopfield networks all outputs of the processing elements are fed back to the inputs and there is no other input. The network state, therefore, depends solely on the previous network state. In the basic Hopfield network the weights and the outputs of the processing elements can only take the values +1 and -1. The discrete time system for the basic Hopfield network is given by:

$$V_i(t+1) = sign\left(\sum_{j=1}^{N} T_{ij} V_j(t) \right) \qquad (3.4.1\text{-}1)$$

Where: $V_i(t)$ = is the state of PE i at time t. It can take the values ±1
T_{ij} = is the weight of the connection between PE i and PE j
N = the number of processing elements

An input pattern is presented to a Hopfield network by setting the network state to the input pattern. The network then iterates until it reaches an equilibrium point, i.e. $V(t+1) = V(t)$. The equilibrium points are determined by the connection weights. The weights should be chosen such that for each input pattern the network state converges to the desired equilibrium point (the output pattern). This means that the equilibrium points must be asymtotically stable, i.e. within a certain distance from the equilibrium point the network state always converges to this equilibrium point. The basin of attraction of a stable state is formed by all input patterns that eventually converge to this stable state. The equilibrium points can be studied using standard techniques of analysis of nonlinear systems (see e.g. [Luenberger 1979]). Creating the desired equilibruim points and basins of attraction is not an easy task. The basins of attractions tend to be irregular and fill only a fraction of the total state space [Keeler 1986]. The rest of the state space is filled with the basins of so-called spurious attractors. Thus input patterns that are very different from the training patterns, may give strange and unwanted

output patterns. If the number of stored stable states increases, also the number of these spurious stable states increases. Beyond a certain limit even the output patterns for training patterns cannot be retrieved reliably anymore.
Variations on the basic Hopfield network include continuous input signals and sigmoid transfer functions and asynchronous discrete, synchronous discrete and continuous time systems. For networks with continuous input signals and sigmoid transfer functions, the basic approach to analysis of equilibrium states does not change, but the creation of the desired equilibrium states may grow even more complicated.

3.4.2 Hopfield network training

To create equilibrium points for training patterns x^p, in the basic Hopfield network the weights T_{ij} are assigned according to:

$$T_{ij} = \begin{cases} \sum_{p=1}^{M} x_i^p x_j^p, & i \neq j \\ 0, & i = j \end{cases} \qquad (3.4.2\text{-}1)$$

Where: x_i^p = element i of training pattern p. It can take the values ±1
M = the number of training patterns
since the training patterns can only take the values ±1, the weights are also limited to these two values

Thus the weights are not found using an iterative process, like for the other networks described in this section, but calculated directly from the training patterns. The training patterns act as exemplars or templates of a class of input patterns.
If the weights are assigned according to eq.3.4.2-1, the network equation for an equilibrium point x^k becomes (substitution of eq.3.4.2-1 into eq.3.4.1-1, remember that weights and outputs can only take the values +1 and -1):

$$x_i^k = sign\left(\sum_{j=1}^{N}\sum_{p=1}^{M} x_i^p x_j^p x_j^k\right) = sign\left(\sum_{j=1}^{N} x_i^k x_j^k x_j^k + \sum_{j=1}^{N}\sum_{p=1,p\neq k}^{M} x_i^p x_j^p x_j^k\right) =$$

$$= sign\left(N x_i^k + \sum_{p=1,p\neq k}^{M} x_i^p \sum_{j=1}^{N} x_j^p x_j^k\right) = sign\left(N x_i^k + \sum_{p=1,p\neq k}^{M} x_i^p \left(x^p \cdot x^k\right)\right) \qquad (3.4.2\text{-}2)$$

Clearly, if all training patterns are orthogonal, eq.3.4.2-2 is true, because then dot product $(x^p \cdot x^k) = 0$. The maximum number of equilibrium points is thus equal to the number of processing elements if all training patterns are orthogonal. If the training

patterns are not orthogonal, the number of equilibrium that can be created points will be less and depends on the differences between the training patterns. Hopfield showed that if the training patterns are generated randomly and the weights are assigned according to eq.3.4.2-1, the training patterns can generally be retrieved reliably if the number of stored patterns is less than 0.15 times the number of processing elements [Hopfield 1982].

3.4.3 Advantages and disadvantages of the Hopfield network

An advantage of the learning rule for the Hopfield network is that it is not iterative, but the weights can be calculated directly from the training set. On the other hand in its basic form it can only handle binary data and training sets that consist of a limited number of templates for different classes and not a large number of arbitrary examples. This limits the applicability of the Hopfield network. There do exist variants that can handle continuous input and output, but the number of stable states is still limited. The network can therefore not be used for arbitrary continuous mappings. Another disadvantage of the Hopfield network is its inefficiency. E.g. in order to store 10 stable patterns that can be retrieved reliably, over 70 nodes and 5000 connections are required.
It has been argued that sequential networks that determine the Hamming distance between binary patterns (Hamming networks) are more efficient and can realise the same mappings as the Hopfield network.
Also the mathematical theory behind Hopfield network and the basins of attraction is very complex especially when the networks are enhanced for continuous signal processing.

3.5 The Kohonen network

3.5.1 Network architecture and processing

The network Kohonen presented in 1984 [Kohonen 1984] consists of a single layer of processing elements, that are geometrically ordered in a two dimensional plane and only interact during learning. The inputs of the network are connected to all processing elements. The output of a processing element is the square of the Euclidian distance between its weight vector, formed by the weights of the connections to the processing element, and the input vector:

$$d_j = \sum_{i=1}^{N} (x_i - w_{ij})^2 \qquad (3.5.1\text{-}1)$$

Where: d_j =output of PE j

x_i = element i of input vector x
w_{ij} = weight of connection from input i to PE j
the summation is over all input elements i

The element with the lowest output (i.e. the minimum Euclidian distance of the weight vector to the input pattern) is selected. Each output is assigned a class label. The output of the network is the class label of the processing element with the minimum Euclidian distance to the input vector. Sometimes other functions are used to calculate the output of a PE, like e.g. a dot product of the weight vector and the input vector. In this case not the minimum, but the maximum output is selected, because the weight vector of the corresponding PE matches the input vector best.
The Kohonen network can be used with continuous inputs. The output is discrete. The maximum number of classes that can be assigned is equal to the number of processing elements in the network. Generally several outputs are assigned the same class label. The Kohonen network is in fact a nearest neighbour classifier. The processing elements represent clusters. The difference with other clustering techniques is the geometrical ordering of the clusters, which is used during the training process. The architecture of a Kohonen network is illustrated in fig.3.9.

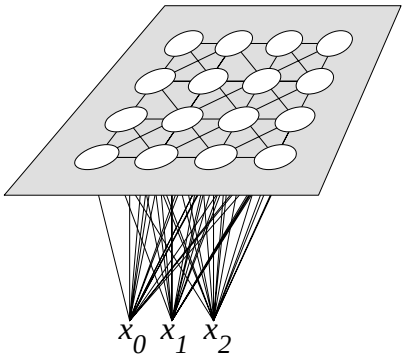


fig.3.9 Architecture of Kohonen's self organising feature map. The processing elements are ordered in a two dimensional plane. Every input is connected to every processing element in the single layer.

3.5.2 Learning in a Kohonen network

Training of the Kohonen network begins with initiating the weights of the connections with small values. Next each training vector is presented sequentially. For each training vector the distances to the weight vectors are calculated, and the processing element with the smallest distance is selected. The weight vector of this processing element are adjusted to make the distance to the training vector smaller:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_i - w_{ij}(t)) \qquad (3.5.2\text{-}1)$$

Where: $w_{ij}(t)$ = the original weight vector
$w_{ij}(t+1)$ = the updated weight vector
η = a gain term, $0 < \eta < 1$

The gain term is decreased slowly during learning. After enough training vectors have been presented, the weight vectors will specify cluster centres that sample the input space such that the point density function of the vector centres tends to approximate the probability density function of the input vectors [Kohonen 1984]. Figure 3.10 shows the position of the weight vectors during the training process for a two dimensional training input vector set with uniform distribution over a square area.

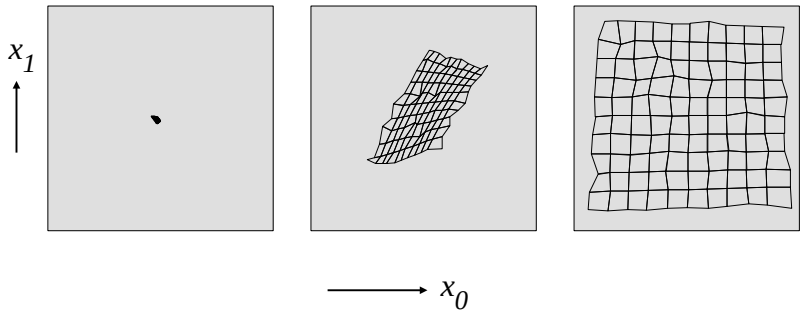


fig.3.10 The weights to 100 PEs in the input space during training with a two dimensional training set with uniform probability distribution over a square area. The weights form a sampled representation of the uniform probability distribution.

Note that this clustering process does not require target outputs to be specified. The clustering process is therefore unsupervised.
In a variation of the above clustering method, not the weight vector of a single PE is updated, but a the weight vectors of all processing elements in a certain neighbourhood of this PE. The neighbourhood of a PE are its geometrical neighbours in the two dimensional plane. The number of weight vectors that are updated (the neighbourhood size), decreases slowly during training.
The second stage of the training process consists of assigning class labels to each processing element. This can be done by presenting a labelled training set to the trained Kohonen network, and assigning the labels to the selected outputs for each of the training vectors.

3.5.3 Properties of the Kohonen network

Like the Hopfield network, the Kohonen network can only generate a limited number of different output patterns. The maximum number of different output patterns is equal to the number of processing elements. The Kohonen network can work with continuous input however. The Kohonen network requires large training sets from which it constructs a sampled approximation of the probability density function, that can be used for classification of the data. If the input data has a high dimensionality, the required training set and the number of processing elements required for an accurate approximation of the probability density function may grow very large. Thus Kohonen networks are applicable mainly for classification tasks with continuous inputs and low dimensional input data.

The advantage of the Kohonen network is that the first stage of the learning process can be unsupervised. Kohonen networks can be used as an alternative to other existing clustering algorithms (see for an overview [Dasarathy 1991]). It depends on the type of application which clustering algorithms performs best. If the Kohonen network is implemented in parallel hardware, a processing speed advantage may be expected relative to other clustering algorithms.

3.6 Recapitulation

Neural networks are networks of simple processing elements that have high connectivity. They can be regarded as multi-parameter function approximation techniques, realising complex functions by combining many simple basic functions. The simple basic functions are the transfer functions of the processing elements. The combination of the basic functions is determined by the connection structure and the connection weights (the parameters) The connections therefore determine the function of the network. Learning rules are methods to find the appropriate connection weights. Learning is done by presenting examples of the desired behaviour to the network. Two types of learning can be distinguished: supervised and non-supervised. In supervised learning the connection weights are updated according to an external evaluation of the network output. In non-supervised learning some kind of internal evaluation takes place. The training sets consist either of a large set of arbitrary examples or of a limited number of carefully selected examples that are templates for classes of input patterns.

The learning-by-example approach has the advantage that a model of the desired behaviour is easily defined: just collect examples of the desired behaviour. However this is only partly true. In many cases it is not easy to determine whether the desired behaviour is sufficiently accurately represented by the example set.

Like other multi-parameter function approximation methods, neural networks can be used to find an approximation of a desired function relatively easy. The accuracy of the approximation can, however, often only be determined experimentally.
If the neural network approach is compared to the model based approach to image processing and analysis described in chapter 2, it can be concluded that the neural network approach is in fact a model based approach, using non-parametric models and multi-parameter solutions. As mentioned before, this has the advantage that no simplifications of the model are required to find a solution, but the disadvantage that little knowledge is obtained about the underlying processes and the solution.

4. Training and test images

4.1 Introduction

Neural networks are trained with examples of the desired input-output behaviour. For image filtering this means that examples of input images and the corresponding desired output images must be presented to the network. The AVR evaluation method, described in this work, uses test images to estimate probabilities on different types of errors. Just like for training neural network image filters, the evaluation requires input test images and the ideal output images for the image filter under test. The input images must be representative for the applications of the image filters.

For non-supervised training the target output image is not required. Hence there should be no problems finding training input images for non-supervised training. However often in a second stage a labelling process takes place, that still does require a target output. For supervised training both input image and a target output image are required. The necessity of a target output image limits the choice of available training input images. In some cases an input image and the corresponding target output image can be obtained easily. Consider e.g. an image obtained under ideal circumstances and an image of the same scene, but with a certain distortions that must be corrected. In many cases the target output images are not available, and must be generated. An example of the latter is the problem of edge detection: no ideal edge maps exist as natural images. There are basically four approaches to obtain an input image and target output image pair fit for training and evaluation:

- natural images
- hand made and hand edited images
- synthetic images, based on 2D image models
- synthetic images, based on 3D scene and imaging models

The four methods will be described in the subsequent sections.

4.2 Natural images

As mentioned in the introduction, only certain types of filtering operations can be designed using natural images. These are primarily filters for image restoration. The advantage of using natural images for training a neural network is that the non-parametric model is extremely close to reality, provided that the images are representative for the application. The resulting filter is also truly optimised for this specific type of images and is tuned for the specific application. An example is compensation of blur caused by moving objects. If the same scene can also be recorded with non moving objects, a target output image can be generated.

4.3 Hand made or hand edited images

The second method to obtain target output images is to make them by hand. The advantage of this approach is that often natural training input images can still be used. The natural images are closest to the images that appear in the actual application. However, sometimes it is a laborious task to construct a reliable target output image for natural images. Consider e.g. drawing an ideal edge map for a natural image. Also the result may not be very accurate. Less laborious and probably more accurate is to edit an imperfect target output image.
These methods for generating target output images can be used for designing neural network image filters for image enhancement and image feature enhancement and extraction. In the next paragraphs two illustrations are given.

4.3.1 Training images for texture classification

To construct a training image and reference image for an image filter that is to classify textures, a cut and paste approach can be used. Examples with the different textures are pasted together to an image. Since for each pixel in this image the texture class is known, a target output image can easily be constructed. In fig.4.2, a handmade test image and reference image for texture classification are shown.

4.3.2 Training images for edge detection

To obtain an "ideal" edge map, the resulting edge map of an existing edge detector can be edited, e.g. to complete broken and missing contours and remove spurious edge pixels. As an example the result of a Sobel edge operator [Pratt 1978] on a grey level image was used as the basis of the edge map. Of the Sobel filtered image, first the local maxima were determined and the result was processed with a low threshold. In

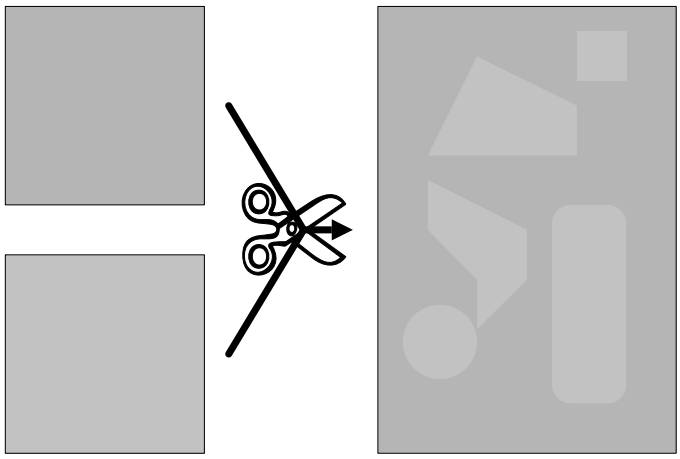


fig.4.1 Cut and paste method to create input image and corresponding target output image for texture classification.

the resulting edge map almost all level transitions are present due to the low threshold. But there are also a many spurious edge pixels. Next this edge map was edited by hand. Spurious edge pixels were removed, and broken contours were completed. Edge pixels caused by shadows were also removed. The original edge map and the result of the editing process are shown in fig.4.2.

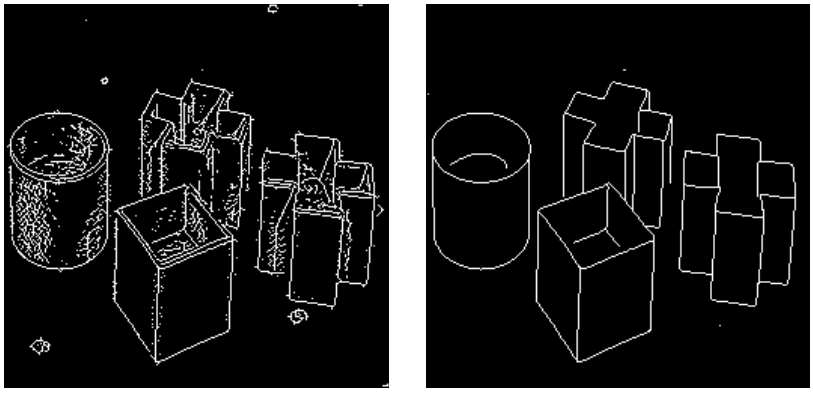


fig.4.2 Generation of an "ideal" edge map by editing the imperfect result of an edge detector.

4.4 Synthetic images, based on 2D image models

In this section the generation of images is based on 2D models of the spatial luminance distributions in images. It is often a good idea to start with the definition of the target output image. In the case of image restoration, this would be the ideal image, without distortions. For feature extraction this is a map with segments with labels that represent the different features. For edge detection the "segments" that represent the feature edge will be lines (one pixel wide). From the target output image, the training input image can be generated using a model of the luminance distribution due to the distortion (for image restoration) or of the features (for feature extraction).

4.4.1 Training images for image restoration

Assume that the imaging process consists of an ideal imaging process and a distortion process, and that this distortion process can be formulated in such a way that the distorted image is obtained after operation of this distortion process on the ideal image. Introducing this in eq.2.3.2-1, this results in:

$$L(x,y) = D,S,P\{scene\} = N,D_I,S_I,P_I\{scene\} \qquad (4.4.1\text{-}1)$$

Where:
- $L(x,y)$ = the luminance at position (x,y) in the image
- D = the non-ideal digitisation process
- S = the non-ideal conversion of luminance to luminance signal
- P = the non-ideal projection of the scene on the image plane
- D_I = the ideal digitisation process
- S_I = the ideal luminance conversion process
- P_I = the ideal projection process
- N = the distortion process

Now the ideal, not-distorted image $L_I(x,y)$ can be obtained by determining N^{-1} and applying it to the distorted image:

$$L_I(x,y) = D_I,S_I,P_I\{scene\} = N^{-1},N,D_I,S_I,P_I\{scene\} = N^{-1}L(x,y) \qquad (4.4.1\text{-}2)$$

As already described in chapter 2, although a model of the distortion process N may be available, it can still be very complicated to find a good approximation for its inverse. From eq.4.4.1-1 it is clear however, that, if a model of the distortion process is available, a training input image can easily be obtained from the ideal target output image. Consider for example designing an image reconstruction filter, that suppresses noise in an image. For most imaging sensors, the associated sensor noise can be modelled either as a zero-mean Gaussian- or Poisson-distributed random process [Pratt 1978]. The Poisson-distributed noise only occurs at very low light intensities. At lower light intensities, the signal to noise ratio (SNR) will decrease. A training input image can be generated by adding zero-mean Gaussian- or Poisson-distributed noise to the ideal target output image. The latter can be a natural image with low noise. An example of a training input image with additive Gaussian noise and corresponding target output image is shown in fig.4.3.

fig.4.3 Training input image with additive Gaussian noise and target output image, for training a neural network for restoration of noisy images.

4.4.2 Training images for edge detection

Secondly an example of generating training input images and corresponding target output images for edge detection is considered. First the assumption is made that the projection of a scene results in an image that consists of segments with different grey levels that are constant within the segments. Next it is assumed that the imaging process is ideal. From these assumptions it follows that all luminance changes are induced by "physical edges". The edges are the borders of the segments. An edge detector operating on this type of images only has to detect all luminance changes. First the target output image, i.e. the ideal edge map, is defined. In fig.4.4, the ideal edge map consists of polygonal segments, obtained by a process that is called Voronoi tessellation [Ahuja 1983]. Voronoi tessellation partitions a two dimensional space into regions around reference points in this space, that are uniformly distributed over the space. All points in a region have the same reference point as their nearest neighbour. This results in an image with polygonal segments. If curved edges are to be detected, circular or arbitrary shaped segments can be used. The next step is to assign grey levels to the segments, in such a way that each segment has a level different from the adjacent segments. A problem that occurs here is, what level should be assigned to the pixels of the border itself. Several solutions are possible. E.g. always the level of the brightest (or darkest) segment could be assigned to the edge pixels between two segments. Another possibility is to choose the levels of the edge pixels so that the number of edge pixels that are assigned the higher level is equal to the number of pixels that are assigned the lower level. Yet another solution is to assign to the edge pixels a level between the levels of the darker and lighter segments. In fig.4.4, the latter is used.

A disadvantage of the described way of generating training images is that they may not be representative for the images that occur in the actual application of the image filter. In order to generate training input images that resemble the actual input images better, the described image generation process can be extended. E.g. in fig.4.4 a sharp

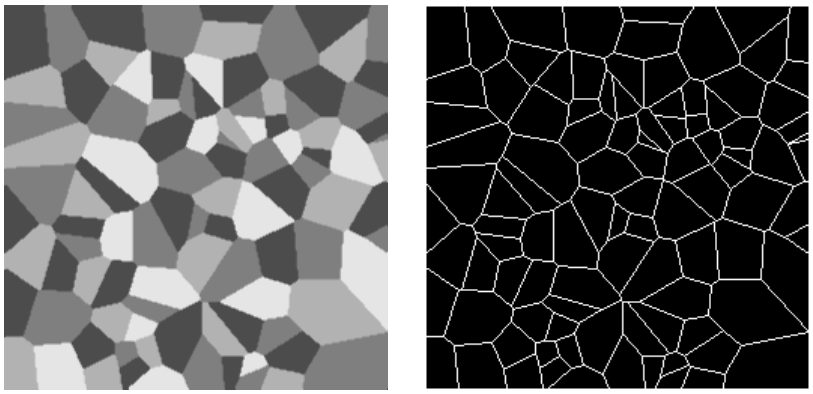


fig.4.4 Generation of a training input image for edge detection, from a target output edge map with polygonal segments. The polygonal segments were obtained by Voronoi tesselation.

edge profile is used. If required the edge profile can be smoothed e.g. by blurring the image. If noise is present in the actual input images, noise can also be added to the training input images.
Nevertheless these images with homogeneous segments still are a poor representation of real input images. Especially for images of 3D scenes this is true. A better, but much more complicated approach is to use scene and imaging models to generate training input and target output images, as is described below.

4.5 Synthetic images, based on 3D scene and imaging models

In this section the generation of training input images and target output images from scene and imaging models, using techniques for generating photo realistic images (ray tracing, rendering) are considered. This approach to obtain training images is generally applicable, but especially of interest for feature extraction and enhancement, because the features are related to their physical origins in the 3D scene. The causes of the features are defined in the 3D scene model, e.g. for edge detection, the physical edges are defined. The target output image is obtained by projecting the 3D features on the image plane. The generation of realistic images requires models of the geometric and the radiometric properties of the scene as well as accurate models for projection and of distortions caused by the imaging system. A good text book on these subjects is [Foley 1989]. As an example generating edge reference maps from 3D scene and imaging models is described (see also [van Rooijen 1992].)

4.5.1 Geometrical and optical scene models

The first step is to define a scene model that is representative for the application of the image filter. To obtain a scene model that is representative for real world scenes, it is often necessary to define the shape of the objects very accurately, including rounded corners, irregularities of object surfaces etc. Therefore advanced CAD tools will generally be indispensable.
In order to calculate the luminance distribution of the resulting image, models of the optical properties of the light sources and the objects in the scene are required. The position and shape of shadows of objects are determined by the position and shape of the light sources. The same applies to highlights on specular reflecting surfaces. Light incident on the surface of an object is partly absorbed and partly reflected. Normally the reflected light is direction dependent. The reflection function peaks in the direction of the specular reflection. Different materials have different reflection functions. Often, for simplicity the reflection function is chosen either completely specular or homogeneously diffuse. This however, results in images that are not very realistic. An illumination model that incorporates the reflection properties of different types of materials is the Torrance-Sparrow model (see e.g. [Foley 1989]). Other items that may have to be considered to accurately model the optical properties of the scene are: colour, polarisation, attenuation and scattering of light by the atmosphere and translucent objects. An extensive description of the modelling of (extended) light sources and the reflection functions of objects can be found in [Foley 1989].

4.5.2 Model of the image acquisition system

In chapter 2 an imaging system was divided into three processes: projection of the image on the image plane, conversion of the luminance on the image plane to a luminance signal, and digitisation. It may not suffice to use a straightforward pinhole projection and linear conversion model. In order to obtain realistic images that are representative for the input of the image filters, the distortions that can be caused by the image acquisition system must be analyzed.
Distortions, caused by the image acquisition system include focusing effects, lens distortions, diffraction, scatter and vignetting. Focusing effects are caused by the fact that objects are more or less in focus depending on their distance to the lens, an effect known as depth of field. The projections of points that are out of focus will be blurred. The luminance distribution on the image plane of the projection of a point on an object in the scene depends on the lens and the aperture. Below this is briefly described. A more extensive description can be found in [Potmesil 1982]. Lens distortions include spherical and chromatical abberations and absorbtion and scattering. Spherical abbera-

tion is the effect that not all points in the image plane are focused on the same distance to the camera. Chromatic abberation is the effect that different wave lengths are refracted differently by the lens. Note that if grey level images are to be generated, it may still be necessary to model colour, because not doing so may result in unrealistic sharp images. Vignetting is the effect that points on the image plane that are further from the optical axis, receive less light, because they are further from the lens.
Finally if appropriate, models are required of the conversion to a luminance signal and the digitisation process. For an accurate model the size and position of the photodectectors should be known in order to determine the radiant flux on each photodetector. The luminance signal is a function of the radiant flux of a photodetector. The conversion characteristics of radiant flux to luminance signal and luminance signal to digital (grey) level and the conversion characteristic from light radiance to digital grey level should be present in the model. Furthermore a model of the noise properties of the photo detectors may be required.
The above described models primarily apply to scenes in visual light and CCD camera's. For other scenes and imaging devices (e.g. radar and sonar) other items may be of importance.

4.5.3 Rendering

The universal term for generation of realistic images from scene and imaging models is rendering. The basic technique of rendering is ray tracing. The level of a pixel in the image is a function of the radiant flux on the photo detector. To determine the level of the pixel the irradiance [W/m^2] must be integrated over the surface of the photo detector. To determine the radiation transmitted by a infinitesimal small lens area onto a given sensor area, the point of origin of the corresponding light beam must be found. The search for these points of origin, also known as visible surface determination, is done by tracing the imaginary ray from the camera to the objects. If the point is located on an opaque object, the intensity of the light reflected by the point on the surface towards the lens is an integral over the hemisphere above the surface [Cook 84]:

$$I(\Phi_r,\Theta_r) = \int_{\Phi_i}\int_{\Theta_i} L(\Phi_i,\Theta_i)R(\Phi_i,\Theta_i,\Phi_r,\Theta_r)d\Phi_i d\Theta_i \qquad (4.5.3\text{-}1)$$

Where: I = the intensity of the reflected light
L = the illumination function
R = the reflection function
(Φ_i,Θ_i) = the direction of the incident light
(Φ_r,Θ_r) = the direction of the reflected light towards the camera

The illumination function depends on both the light sources and on light reflected by other objects. A general formulation of eq.4.5.3-1, incorporating light sources, transparent and translucent objects is called the *rendering equation* and is discussed in [Kajiya 86]. As the described quantities vary over the spectrum, integration over the spectrum is also required. Summarising one can say that the radiant flux on a photo detector can be described by a number of nested integrals over the area of the photo detector, over the area of the lens and over the hemisphere on the surface of an object of the reflectance multiplied by the illumination and finally over the spectrum. This process is illustrated schematically in fig.4.5.

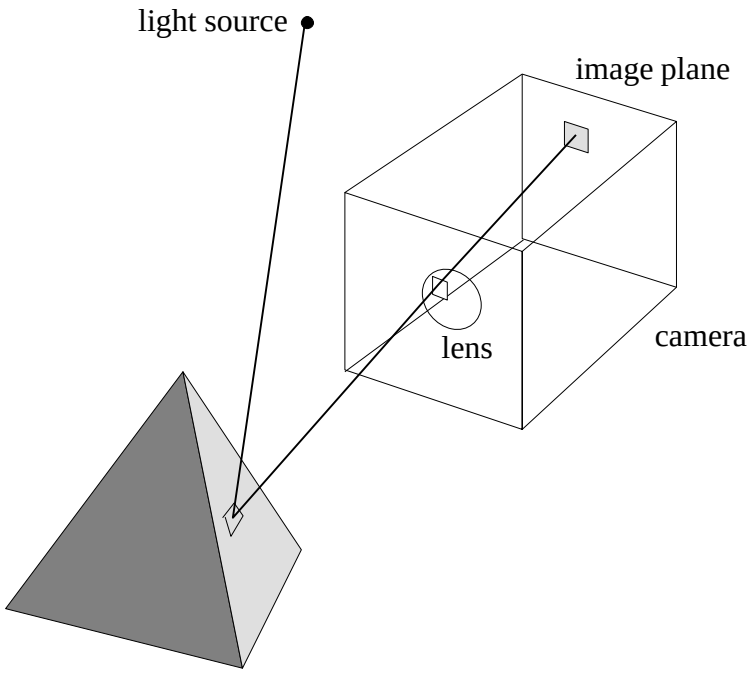


fig.4.5 Detemination of radiant flux on a photo detector by ray tracing. A number of nested integrals have to be calculated: over the surface of the photo detector, over the surface of the lens, over the directions of the incident and refleted light of the product of the illumination and the reflection functions and finally over the spectrum.

Due to the complexity of the rendering equation it is generally impossible to calculate the levels in an image analytically. There are two ways to approximate the expression. The first is to simplify the described rendering process, e.g. by assuming that the light sources are points, and take no reflection of other objects into account for the illumination function. In [Cook 1986] several of these simplifications are described. The disadvantage of this approach is that the simplifications may result in non-realistic effects, like e.g. hard shadows as a result of point shaped light sources. The other method to approximate the expression for the radiant flux, is to point sample the integrals. This means the integrand is calculated for a number of points within the integral boundaries, and the integral is estimated from these points. An unbiased estimation can be obtained if these points are located randomly. This is known as the Monte Carlo method. The resulting ray tracing technique is called *distributed ray tracing* (see [Foley 1989]). The advantage of distributed ray tracing is that, by increasing the number of sample points, the approximation of the integrals can be made arbitrarily good

(at the cost of course of calculation time). It is therefore best suited for generating high quality realistic images. The disadvantage is the huge number of rays that must be traced (for every sample point a ray must be traced).

4.5.4 Generation of the target output image

Up to now only the generation of the realistic training input image was considered. The next step is the generation of the corresponding target output image. In the case of image restoration, this is quite straightforward. It requires only a simplification of the model. E.g. if an image filter is to be designed for compensation of image blur due to a lens that is out of focus, the target output image can be obtained by using an in focus lens in stead of the out of focus lens, used for generating the unsharp training input image. For noise suppression, the target output image can be calculated by removing the noise sources from the model (photo detector noise, quantum effects of light, object surface irregularities).
For feature detection and enhancement, a model is required of the physical process that generates the features in the image. For edge detection, this is a description of the physical edges, i.e. a description of the physical changes of interest that induce luminance changes in an image. This is worked out as an illustration in the next paragraph.

4.5.5 Generation of edge reference maps from 3D scene models

The target output image should contain the locations of the luminance changes that are induced by the physical edges we want to detect. Four different types of physical edges are considered here:

- occluding contours of objects
- abrupt changes in surface orientation of objects
- abrupt changes in surface reflectance of objects
- shadow casting

A method to obtain the occluding contour of objects is detection of discontinuities in the z-coordinate (the z-axis is the axis perpendicular to the image plane). In fig.4.6 a projection of the z-coordinate of a scene on the image plane is shown as a grey level image (darker means further away from the camera). To each position in the image the z-coordinate of the point on the object that reflects light to that position in the image is assigned as a grey level.

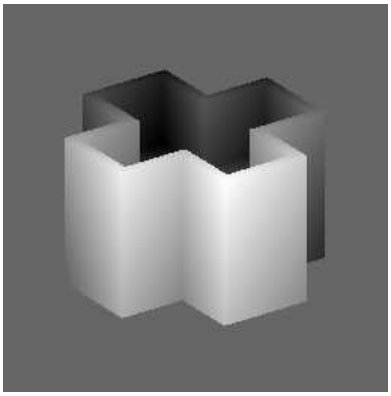


fig.4.6 Projection of the z-coordinate of an object on the image plane as a grey level image (darker means further away). Object boundaries can be found by detecting discontinuities in the z-coordinate.

If objects are not illuminated uniformly around, certain abrupt changes in surface orientation cause luminance changes in images. This is clearest for a point shaped light source. The angle of the incident light from a point shaped light source on a curved plane changes as a function of the position on the plane. This results in a change in the intensity of the reflected light as a function of the position on the plane and therefore in luminance changes in the image. For uniformly around illuminated objects, the incident light on each position is the same, so changes in surface orientation do not result in luminance changes in the image. Finding abrupt changes in surface orientation is more difficult than finding discontinuities in the z-coordinates of objects. A possible solution is to detect changes in the direction of the normal vector on the object surface, and to project these on the image plane. Note that the term abrupt refers not only to the object itself, but also to the imaging system. E.g. a very gradual change in surface orientation may look sharp if it is far away from the camera. A very sharp change in orientation, extending over a very small area, may not be perceptible by a camera, if the resolution of the camera is too low. This effect is illustrated in fig.4.7.

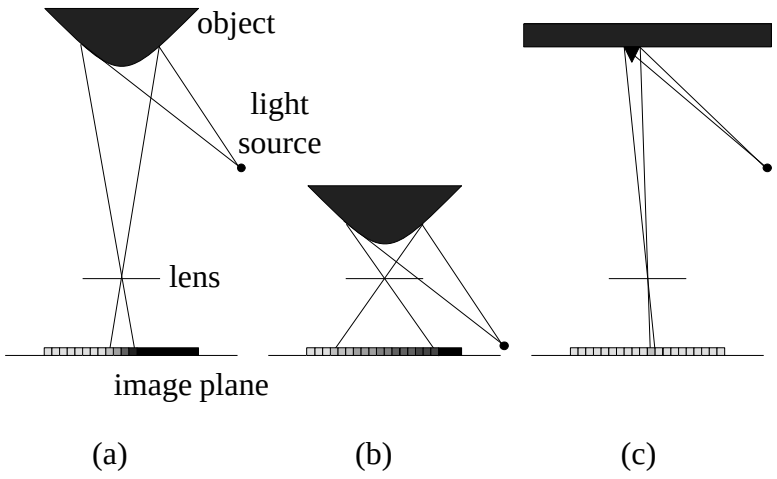


fig.4.7 Detectability of changes in surface orientation depends on camera resolution (a) change in surface orientation causing a sharp lumiance change; (b) same change in surface orientation, but closer to the camera causes a gradual luminance change; (c) undetectable change in surface orientation due to low resolution of camera.

It will depend on the application if these physical edges should all be detected by the edge detector. It is important, however, to bear in mind that some of the physical edges cannot be detected due to the limited resolution of the camera, while others cause low luminance gradients over a large part of the image, which may be difficult to detect. In general luminance changes that can be detected extend over a few pixels, and are thus related to the resolution of the imaging device. It makes sense to choose the physical edges of interest such that they can be detected, or better, to choose an imaging device that is suited for the detection of the physical edges of interest.
The projection on the image plane of the unit normal vectors on the object surfaces, assigns a three dimensional vector to each position in the image plane. The restriction to detection of surface orientations that cause detectable luminance changes, can be realised by averaging the normal vectors over the maximum extent of a luminance change in the image. Next an operator must be found that detects the local maxima of the changes in direction of the normal vectors. In principle the changes in the direction of the normal vector can be obtained by a differential operation of the normal vector image to the image coordinates. This results in the magnitude of the change in the surface orientation or a *measure of curvature* of the projected surfaces. The positions of the corresponding edges in the image plane are defined by the local maxima of the measure of curvature. Finally, because only *abrupt* changes in orientation are physical edges, the measure of curvature must be thresholded. The threshold determines which measure of curvature is still regarded as a physical edge. The above described process of finding abrupt changes in surface orientation is illustrated in fig.4.8.

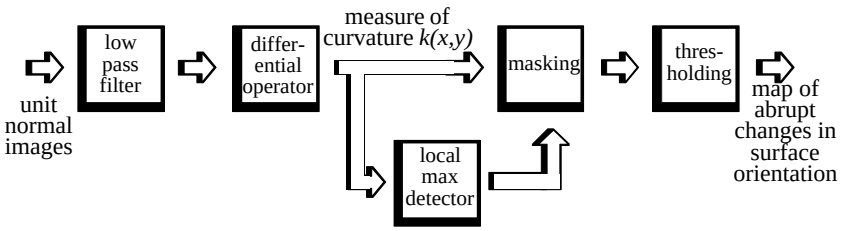


fig.4.8 Detection of the physical edge type abrupt changes in surface.

A problem that remains is to find a suitable differential operator that operates on the unit normal vector image and results in the (scalar) measure of curvature for each position in the image plane. A simple solution would be to take the magnitude of the directional derivative of the normal vector function in the direction where it is maximal:

$$k(x,y) = \max_{\alpha} |f_{\alpha}(x,y)| = \max_{\alpha} \left| \lim_{h \to 0} \frac{f(x+h\cos\alpha, y+h\sin\alpha) - f(x,y)}{h} \right| \qquad (4.5.5\text{-}1)$$

Where: $k(x,y)$ = the measure of curvature at position (x,y) in the image plane

$f(x,y)$ = the unit normal vector function at position (x,y) in the image plane
$f_\alpha(x,y)$ = the directional derivative for an angle α in the image plane
all coordinates are camera coordinates, i.e. x,y in the image plane, z perpendicular to the image plane

The disadvantage of this solution is that curvature due to torsion of a surface also results in a large measure of curvature, although this is often not desired, because torsion does generally not separate different parts of objects (e.g. two perpendicular planes, see fig.4.9). Torsion also often does not induce perceptible luminance changes in images. If torsion curvature should not be detected, not the magnitude but the magnitude of the projection of the directional derivative of the normal vector function on the image plane in the direction α should be taken, because the directional derivative vectors of a torsion curvature are perpendicular on the direction α (see fig.4.9). The measure of curvature in the image plane then becomes:

$$k(x,y) = \max_{\alpha} \left| f_\alpha(x,y) \cdot \begin{pmatrix} \cos\alpha \\ \sin\alpha \\ 0 \end{pmatrix} \right| \qquad (4.5.5\text{-}2)$$

In case of torsion, this results in $k(x,y) = 0$. The processes of projection of the normal vectors on the surface and derivation of the projected normal vectors are illustrated in fig.4.9.

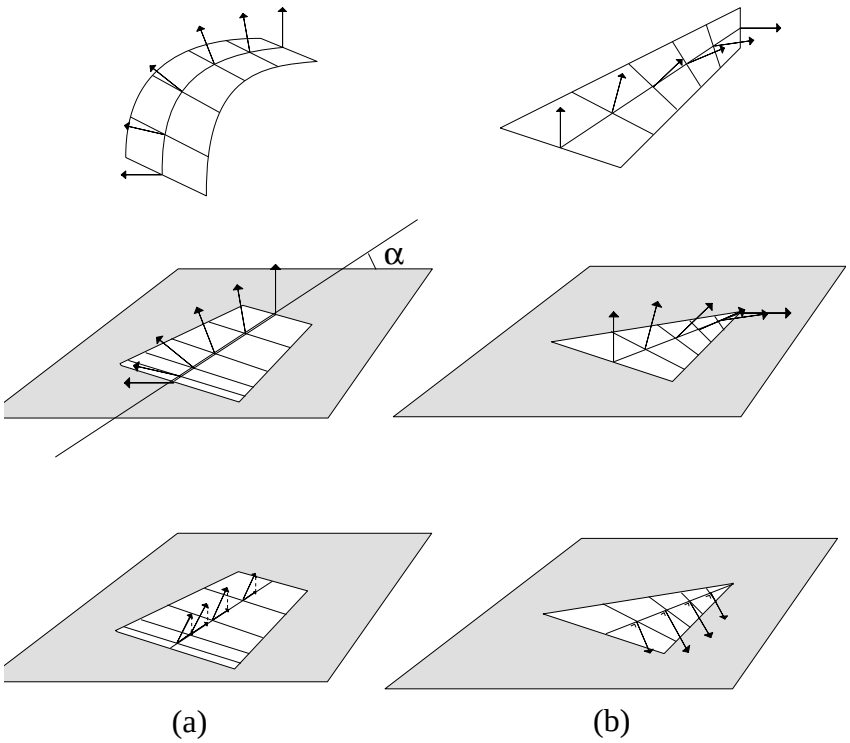


fig.4.9 Detection of surface curvature by projection of the normal vectors on the object surfaces (top) on the image plane (middle) and taking either the magnitude of the directional derivative or the magnitude of its projection in the direction α in the image plane (bottom) as a measure of curvature. (a) A "normal" surface curvature results for both methods in detection; (b) a surface torsion is not detected by the projected magnitude.

Finding the direction α for which the magnitude is maximum is probably easiest using an iterative process.

The physical edge type abrupt changes of surface reflection occurs if there is a change of material (e.g. wood to metal, red ink to blue ink). These changes can generally be obtained directly from the scene model.

Shadow edges are caused by objects that shield parts of other objects from a light source. The parts of objects that are in shadow can easily be determined by ray tracing. The shadow edges are the boundaries between the areas that are in shadow and the areas that are illuminated.
Which of the above described physical edges are to be detected by an edge detector depends on the application. E.g. for some applications shadow edges may disturb recognition of objects.

An example of a training input image and the associated edge reference map, obtained from a simple 3D scene is illustrated in fig.4.10.

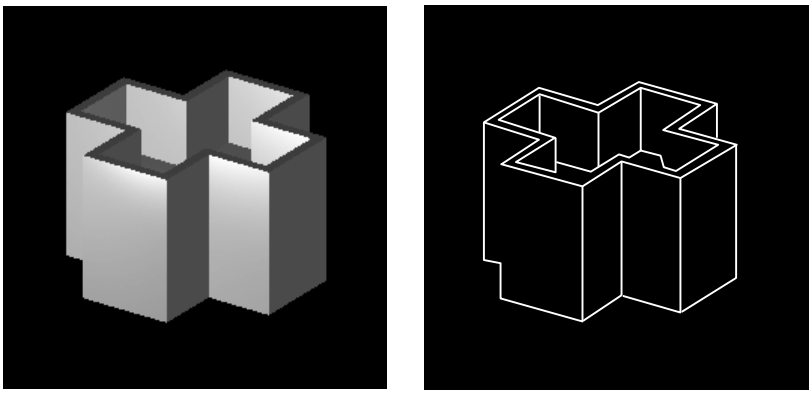


fig.4.10 Training input image and target output image (edge reference map), generated from 3D scene and imaging models.

A few final remarks about generating test and training images from 3D scene and imaging models. Firstly the proposed approach to obtain test and training images is quite new. Secondly it is not completely worked out yet. The concepts are not yet worked out fully and practical realisation and testing should be taken up. Also until now only very simple scenes were considered. Thirdly this kind of accurate modelling of the mapping of physical edges on the image plane may be very useful for other approaches (not based on neural networks) to edge detection and image filtering too.

5. Image filter applications

5.1 Introduction

In this chapter a number of examples of neural network image filter designs are presented. First a number of neural network architectures, based on Hopfield, Kohonen, perceptron and error backpropagation networks, for image filtering are described. It will become clear that the error backpropagation neural networks are most general applicable, because they can be used both for continuous mappings and classification tasks. In order to investigate the applicability and performance of neural network image filters, tools for simulating diverse types of neural networks must be available. The tools must be able to handle efficiently the large amounts of data that are common in image processing. At the time of this research was initiated, no such tool was available. Therefore a toolkit was developed to simulate error backpropagation networks for image filtering in software (see appendix B). Using this toolkit a number of neural network image filters for different applications were designed. The neural network image filters that are presented in this chapter are:

- a noise filter
- an image sharpening filter
- several filters for edge detection

The neural network edge detectors are investigated in more depth. One problem in neural network research is that most application specific knowledge is experimental. There are hardly any theoretically supported rules to determine optimal size and architecture and learning parameters for a certain application. On the other hand, like some other general multi-parameter function approximation methods, neural networks are not too critical about optimal choices.

5.2 Neural network architectures for image filtering

There are two basic approaches to image filtering with neural networks. The first is to use a very large network that uses the whole image as input and produces the whole output image as output. This type of network can be used to realise both position variant and position invariant filtering. For position invariant filtering operations the network has to perform the same operation all over the image. Since the connection architecture and the connection weights determine the operation the network must be highly symmetric.

A disadvantage of this type of networks is that for image sizes common in image processing and analysis, the networks tend to be large. E.g. a 256*256 image requires a network with at least 65536 inputs and the same amount of processing elements in the output layer. If there are multiple layers or there is a high connectivity the number of connections quickly grows to above one million. An advantage of using the whole image as input is that both local and global image properties can be used. A neural network image filter that takes a whole image as input is schematically shown in fig.5.1.

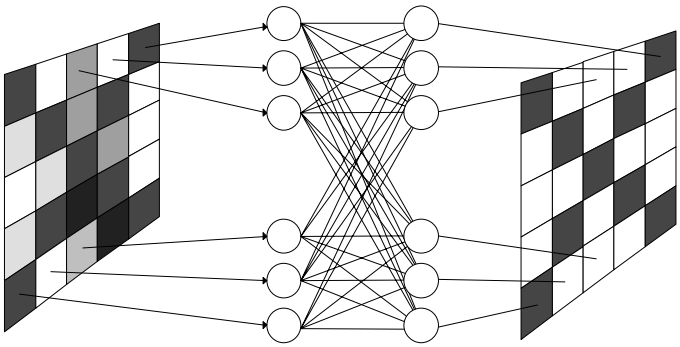


fig.5.1 Example of a neural network image filter using the whole image as input of the network. For clarity a small image is used and not all PEs and connections are drawn. In reality there are 50 PEs and 675 connections.

As described in chapter 3, training of neural networks can be either with a small example set of templates or with a large example set that represents a statistical non-parametric model of the desired network behaviour. It will be clear that the latter is hardly applicable to this type of neural network image filters if all connection weights are to be found independently. If the connection structure is highly symmetric the problem may be manageable. Training using templates can be used both for networks with symmetric and non-symmetric connection structures.

The second approach to image filtering with neural networks is to use small parts of the image as input for the network. If a single network is used for processing the whole image, this approach is restricted to position invariant image filtering. Two variants can be distinguished: filters with a single output, that calculate the output for each pixel separately, and filters with more than one output, that calculate small areas of the output image at once. An image can be filtered by scanning the image, and processing

each local neighbourhood with the neural network image filter. An advantage of this approach is that the network can be relatively small, requiring only as many inputs as the size of the sub-image and one or a small number of outputs. A neural network filtering an image by scanning it is illustrated in fig.5.2.

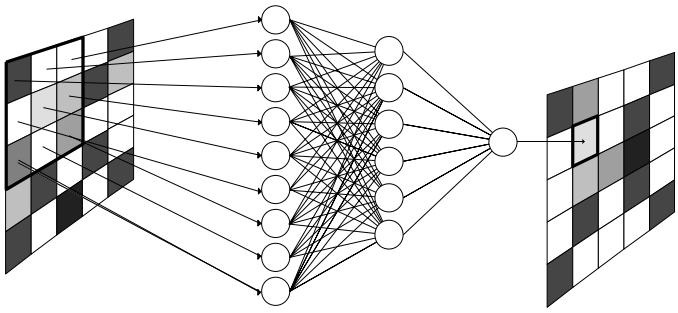


*fig.5.2 Neural network image filtering by scanning the image and processing each local neighbourhoud (here 3*3 pixels) sequentially. The output image is built up pixel by pixel.*

An advantage of using only a small neighbourhood as input of the neural network, is that from a single training image, many training examples can be obtained. Often a single training image suffices to train a neural network image filter. Furthermore, since the networks are small, training times will be relatively short. It is also possible to train position invariant neural network image filters with a template example set. In many cases however, the desired filter operation can be modelled more accurate with a large example set and often it is easier to obtain large example sets from training images, than define good templates.

Since most neural networks are nonlinear, they can be used to realise nonlinear image filter operations. The filter operations, that can be realised with neural networks, depend on the architecture and type of network that is used. The applicability for image filtering of several types of neural networks is considered in the next paragraphs.

5.2.1 Perceptron and error backpropagation networks

The perceptron network was originally designed to detect patterns in binary images. As shown in [Minsky 1969], this type of network can only be used to realise a limited class of functions (see also chapter 3). The error backpropagation neural network has the same layered feed forward architecture, but allows multiple adaptive layers, and can also operate with continuous in- and outputs. Error backpropagation networks can be used to realise arbitrary complex continuous mappings. Since error backpropagation networks can realise all functions that perceptron networks can and more, perceptron networks can be considered as a subset of error backpropagation networks.

Although these networks can be trained using a small set of templates of the desired operation, the strength of the perceptron and the error backpropagation networks is that they can learn a function from arbitrary examples. These networks are therefore generally trained using large example sets. Incidentally also if a small set of templates is used for training these must be presented many times. Therefore these types of networks are best fit for operating on small local neighbourhoods, and less for operating on a whole image. Fig.5.2 illustrates a three layer error backpropagation neural network image filter with a single output operating on a 3*3 local neighbourhood.
Because of the simplicity of the network architecture and the ability to realise both continuous and binary mappings the error backpropagation network may be expected to be suitable for a wide range of different image filter operations.
Although this seems a very simple and promising approach to image filtering with neural networks, not much attention is paid to it in literature.

5.2.2 Hopfield network

Hopfield networks are generally used with binary inputs and outputs, which limits their applicability for image filtering. Training of Hopfield networks is done using templates, that are stored in the network. Hopfield networks have been used for binary pattern recognition. A number of templates (exemplar patterns), e.g. digits, are stored in the network. If a (distorted) pattern is presented to the network, its output converges to the template that best resembles the input pattern. Most of these "image filters" use the whole image as input of the network, resulting in a large network. This architecture however has only limited applicability in image filtering, because only a number of template output images can be generated. Also the networks tend to be rather sensitive for rotation and translation distortions of the patterns. The application of Hopfield networks as image filters is less straightforward and more restricted than the use of error backpropagation networks.

5.2.3 Kohonen network

As described in chapter 3, the Kohonen network uses a non-supervised training strategy, basically a clustering technique, and is trained with a large training set. After enough input vectors have been presented, the weight vectors of the processing elements form a sampled representation of the probability density function of the training data. The output of the Kohonen network for a specific input pattern is the number of the processing element with the minimum (Euclidian) distance to the input vector. Generally a second classification stage is used to assign an output label, value or pattern to each processing element. Since for an input pattern the distance to the weight

vector of each processing element must be calculated, using a whole image as input of the network, is hardly practical. The dimension of the network would be huge, because the weight vector of each processing element has the same dimension as the input image. Also, because the network estimates the probability density function from the input vectors a huge number of training images would have to be available, and training would be extremely slow. Thus Kohonen networks should be used for local neighbourhood processing only. Relatively small neighbourhoods already result in a high dimensional input space for which it may be difficult to obtain an accurate representation of the probability density function. An advantage of the non-supervised training of Kohonen networks over the perceptron and error backpropagation networks, is that no target image is required, so networks can be trained with any natural image. The output of the Kohonen network is not continuous. Only as many output patterns as there are processing elements can be generated. This limits the applicability of Kohonen networks to primarily feature extraction and classification tasks. A first experiment, using a Kohonen network for texture classification in images, based on a 3*3 local neighbourhood proved quite successful.

5.3 Image restoration with an error backpropagation network

5.3.1 Noise filter

In the previous chapter, on training images, a model of noise in an image was described. According to [Pratt 1978], photodetector noise can be modelled with additive zero-mean Gaussian distributed noise. If the variance of the noise is known, an input training image can easily be generated from an ideal target output image. In fig.5.3 the training input image and target output image, used for this experiment, are illustrated. The images are 256*256 pixels in size, and can display 256 different grey levels. The grey level range of both training input and target output images is scaled to fit into the range of 0..1 of the sigmoid transfer. The standard deviation of the noise in this experiment was 15. The variance of the noise may be determined from a model of the noise of the used photodetector, or estimated from test images.

Since the noise is position invariant, it is likely that a position invariant image filter can be used to suppress it. Thus the input of a neural network image filter consists of the local neighbourhood of a pixel. The network has a single output, that represents the estimated grey level of a pixel based on its neighbourhood in the input image, as was illustrated in fig.5.2.

Next the network size and learning parameters must be chosen. To this day there exist no general rules for optimal network size and learning parameters. In most cases they can only be roughly estimated from the complexity of the task and experiments. It may be expected though, that the choice of the size and architecture of the network are not

too critical, since the performance of neural networks tends to degrade gracefully with decreasing size (like other multi-parameter solutions). To suppress noise in a homogeneous area a small local neighbourhood of e.g. 3*3 pixels suffices. To suppress noise near a sharp grey level transition a larger neighbourhood is required. In this experiment a neighbourhood of 7*7 pixels was chosen, resulting in 49 inputs for the neural network image filter. A network with a single adaptive layer (in this case consisting of a single processing element, since there is only one output) is clearly very limited. A network with 2 adaptive layers, 8 processing elements in the first hidden layer, and since there is only a single output, 1 processing element in the second layer (the output layer) appeared to give good results.

From the theory of the backpropagation learning rule, it follows that the learning rate should be chosen small in order to retain the validness of the Taylor series expansions of the system error and also to allow sequential training (see appendix A). Preliminary experiments showed that learning rates of 0.01 to 0.1 give good results with networks of this architecture and size. The connection weights were initialised with random values, uniformly distributed in the interval [-0.1, .. ,0.1].

The training vectors were obtained by scanning the images from top left to bottom right, leaving out the incomplete neighbourhoods at the border. For a 7x7 local neighbourhood size and 256*256 images this results in a total of (256-6)*(256-6) = 62500 training examples. All these examples were presented to the network many times. One cycle through the whole training set is called an *epoch*. During the training of the network the square error was recorded. In this case the squared error is defined as:

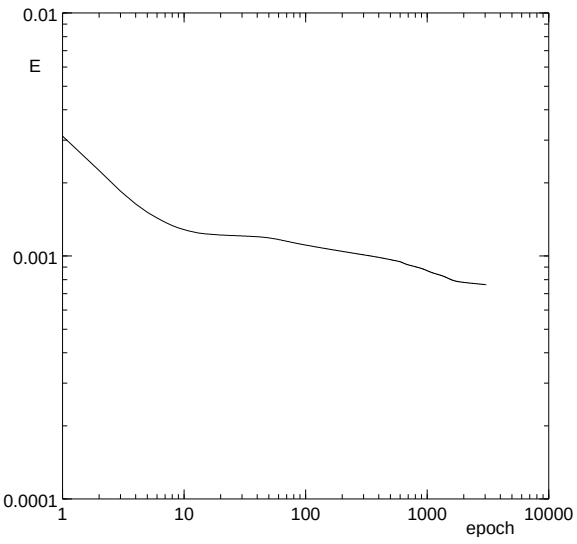
$$E = \sum_{p=1}^{62500} (t^p - y^p)^2 \qquad (5.3.1\text{-}1)$$

Where: t^p = target output for input pattern x^p (7*7 neighbourhood)
y^p = actual output of the network (a single pixel) for input pattern x^p
the summation is over all patterns x^p

In fig.5.3 the squared error during learning is shown as a function of the number of the epoch in the learning curve. In order to be able to compare the learning curves the square error was normalised by dividing it by the number of training vectors and the number of outputs, to give a value between 0 and 1 (the output is restricted to values between 0 and 1 due to the sigmoid transfer functions. From the learning curve it becomes clear that after approximately 3000 epochs the system error still decreases but very slowly. This means that for convergence more than 10^8 training patterns are to be processed by the network. Slow learning is apparently one of the disadvantages of error backpropagation networks. Still searching in a 49 dimensional input space and a more than 200 dimensional weight space will probably always be a bit slow.

Training input image for noise filter. Generated by adding Gaussian noise; $\sigma_{noise} = 15$

Target output image for noise filter.



Learning curve of noise filter: square error (vertical) vs. number of epochs (horizontal).

The resulting, restored image for the noisy input image.

Fig.5.3 Training input images, learning curve and output image of the neural network noise filter. The traininig input image was generated by adding Gaussian noise to a noise free image with $\sigma_{noise} = 15$ *on a grey level range from 0..255. The used network was a three layer error backpropagation network with 49 inputs (7*7), 8 processing elements in the first hidden layer and 1 in the output layer. A learning rate of* $\eta = 0.01$ *was used. The result appears to be quite good. Small details are well preserved.*

Noteworthy is that eq.5.3.1-1 can also be regarded as an estimation of the variance of the noise in the image. The error backpropagation learning rule therefore effectively minimises the variance of the noise in the output image of the neural network noise filter. Eq.5.3.1-1 can be used to estimate the variance in the filtered image. The final system error is about 0.0008, which results in a standard deviation of the noise in the output image of (scaling the output back to 256 grey levels):

$$\sigma_{out} \approx 256 * \sqrt{E} \approx 7 \qquad (5.3.1\text{-}2)$$

The image filter thus improves the SNR of the image by a factor of approx. $\left(\frac{15}{7}\right)^2 \approx 4$.

Note that both high and low frequency noise are suppressed, unlike in simple low pass filters. The latter suppress high frequency noise and sharp transitions, which results in blurred images. Actually they decrease the variance of the image grey levels rather than the variance of the noise.
The result of the neural network noise filter operating on the training image is illustrated in fig.5.3. As can be seen the noise is quite effectively suppressed while sharpness and most small details are preserved well.

5.3.2 Correction of image unsharpness

A similar experiment was set up for correcting unsharpness of images, due to e.g. a lens that is out of focus. The unsharpness due to the optical system of an imaging device is often modelled as a convolution of the image with a two dimensional Gaussian point spread function [Ballard 1982]:

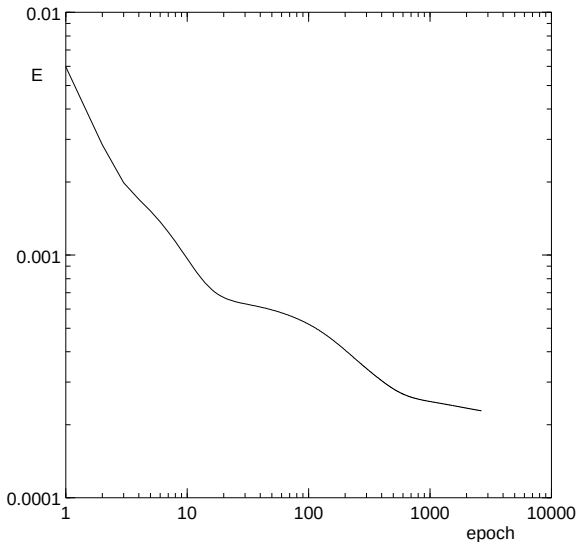
$$G(x,y) = \frac{1}{2\pi\sigma_{psf}^2} \exp\left\{-\frac{x^2+y^2}{2\sigma_{psf}^2}\right\} \qquad (5.3.2\text{-}1)$$

Where: (x,y) = the position in the convolution kernel
σ_{psf} = the width of the point spread function

A training input image was generated by convolving a sharp image with a Gaussian psf with $\sigma_{psf} = 1.5$. The sharp image is the target output image for the blur filter. The convolution results in the grey level being averaged over an area of approximately $4\sigma * 4\sigma$. Thus the size of the local neighbourhood should be chosen at least 6*6 pixels. A network with the same architecture as the noise filter of the previous paragraph was used: 49 inputs, 8 processing elements in the first layer and 1 in the output layer. The

Training input image for blur filter, generated by blurring a sharp image; $\sigma_{psf} = 1.5$

Target output image for blur filter



Learning curve of blur filter. Horizontal: number of epochs; vertical: square error.

Image filtered by the network blur filter.

Fig.5.4 Training input images, learning curve and output image of the blur filter. The traininig input image was generated by blurring the sharp image with $\sigma_{psf} = 1.5$. *The used network was a three layer error backpropagation network with 49 inputs, 8 processing elements in the first hidden layer and 1 in the output layer. A learning rate of* $\eta = 0.01$*was used. The final system error for this network reaches a very low value, which is reflected in the good result of the reconstruction.*

learning rate was taken 0.01 again. The training images and the results are shown in fig.5.4. From the learning curve it can be seen that the system error reaches a very low value of about 0.0002. This low level suggests that the restoration of the blurred image must be quite good. As can be seen in fig.5.4, the filter output image is very close to the target output image.

5.3.3 Discussion

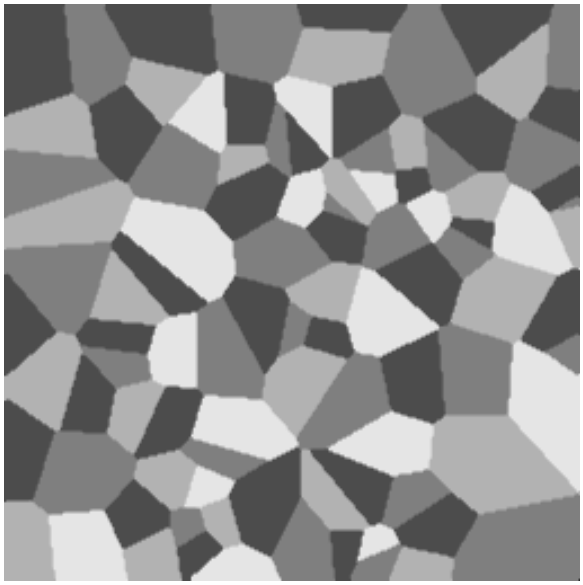
From these two examples of image restoration it already becomes clear that, at least for these applications the design of image filters by training neural networks with training images is a very simple and straightforward procedure. Advantages are that little effort has to be put in modelling and design of an operator and that it is easy to design image filters tuned for a specific application. A disadvantage is the slow learning process. For the noise filter example, the system error of the network also is an estimation of the noise variance in the image. The learning rule therefore effectively minimises the variance of the noise in the output image. For the image sharpening filter the relation between the square error and image sharpness is less obvious. It is clear however that because of the low final error that is reached the result must be quite good.

5.4 Edge detection using an error backpropagation network

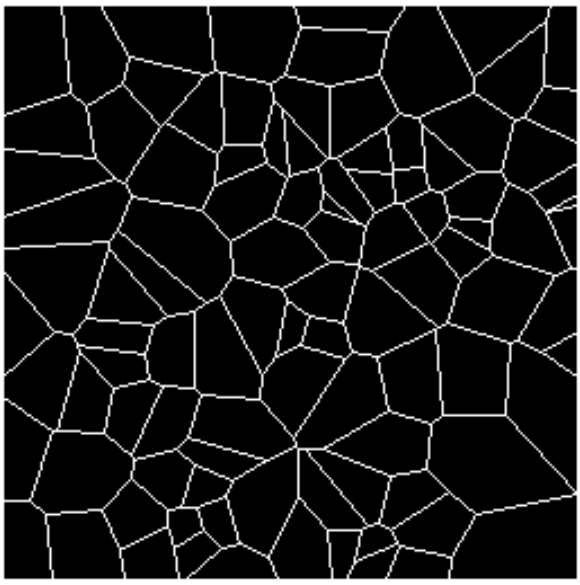
A number of experiments were carried out to investigate the application of neural networks for edge detection. For most experiments the same network architecture as for the two neural network image restoration filters was used: 49 inputs (7*7 neighbourhood), one hidden layer with 8 processing elements and an output layer with one processing element. Neural network were trained with the following training images (see chapter 4):

- Voronoi training image
- Blurred Voronoi training image
- Voronoi training image with additive Gaussian noise
- Natural image with hand edited edge reference map

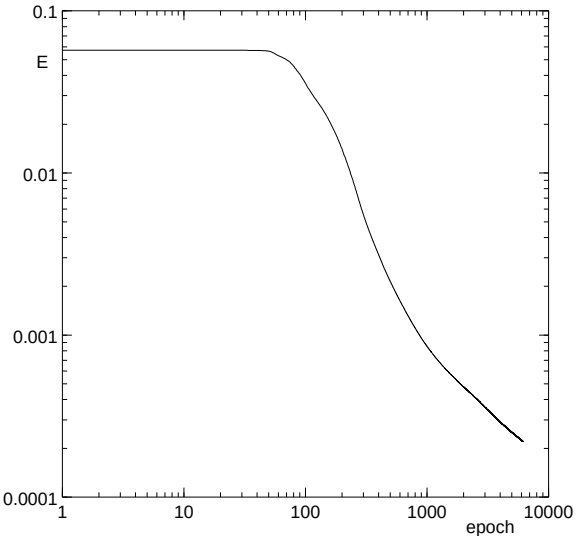
The first experiment is carried out to demonstrate the applicability of neural networks for edge detection. However edge detection in an image with clearly distinguishable segments, as there are in the Voronoi images, is not a very difficult task. Therefore networks were also trained to detect edges in noisy and blurred images and finally with a natural image with a hand edited edge reference map.



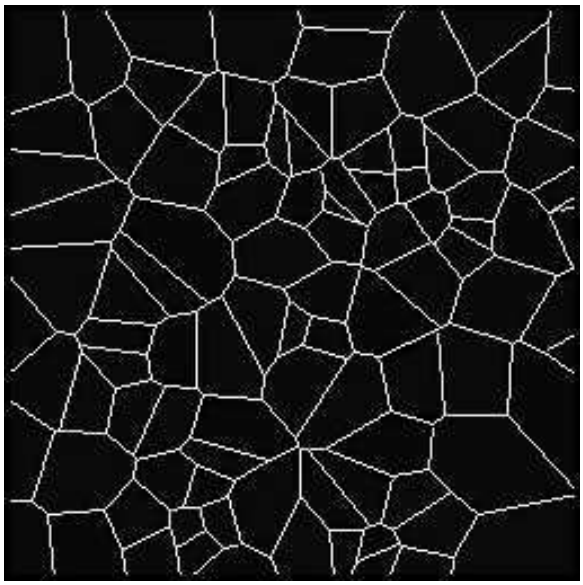
Training input image for edge detection, generated by Voronoi tessellation.



Target output image for edge detector filter.



Learning curve: square error (vert.) vs. number of epochs (hor.)



Result of neural network edge detector on Voronoi input image.

Fig.5.5 Training input images, learning curve and output image for a neural network edge detector. The traininig images were generated by Voronoi tesselation. The used network was a three layer error backpropagation network with 49 inputs, 8 processing elements in the first hidden layer and 1 in the output layer. A learning rate of $\eta = 0.01$ *was used. On this image the edge detector performs almost perfectly. A very low system error is obtained. Edge detection in this type of perfectly clear images is actually not a very difficult task. Many other simple edge detectors can match this result.*

The three layer network architecture used in these networks shows a close resemblance with the architecture of the cvm operator of van der Heijden [Heijden 1992]. In the cvm operator the first stage consists of a number of linear filters of which the output is combined in a log likelihood ratio which is thresholded to detect the edges (see fig.2.13 in chapter 2). The weighted summations of the processing elements of the hidden layer in a three layer error backpropagation network can also be regarded as linear image filters. The outputs of these filters are transformed by a sigmoid function and combined in the output layer by the weighted sum of the output processing element. Van der Heijden obtained a number of orthogonal image filters. Because of the similarity in the architecture, one might expect similar convolution kernels for the cvm and neural network edge operators. The "filters" of the neural network edge detectors are compared to the filters of the cvm operator in paragraph 5.4.4.

5.4.1 Training with clean Voronoi image

The aim of this first experiment was to find out if error backpropagation neural networks can be trained for feature detection tasks like edge detection. In fig.5.5 training input and target output images are shown that were obtained by Voronoi tessellation (see chapter 4). The resulting output image after training, shown in fig.5.5, is nearly perfect. Like for the noise and blur filter the system error in the learning curve gives an indication of the performance of the network. The final system error of approximately 0.0002 results in an average error of $\sqrt{0.0002} \approx 0.014$ per pixel on a scale of 0..1. This can mean that all pixels are classified correctly, the target value of edge (1) and not edge (0) being approximated with an accuracy of 0.014, but it can also mean that e.g. 62500*0.014=875 pixels are classified incorrectly. Both give the same system error, but the latter is much worse (and clearly not the case in fig.5.5). This shows that the squared error measure is not very suitable for performance evaluation of neural network edge detectors. Also a number of important errors that can occur in edge detection, e.g. localisation errors, are not well accounted for in the squared error measure. Thus the performance of the edge detectors actually should be determined in a different way. In chapter 6 an edge detector evaluation method based on the average risk is described. Also the relation between the squared error and the average risk is analyzed.

5.4.2 Training with blurred and noisy Voronoi images

To train neural network edge detectors for detection of edges in blurred and noisy images, blurred and noisy training input images were used. A blurred test image was generated by convolving the original Voronoi image with a Gaussian psf with

$\sigma_{psf} = 1.5$ and a noisy test image was generated by adding zero mean Gaussian distributed noise to the original Voronoi image with $\sigma_{noise} = 20$ on a total grey level range of 255. The local SNR near a grey level transition varies from about 5 to 100, depending on the difference in grey levels. The blurred and noisy Voronoi images are shown in fig.5.6.

The three different neural network edge detectors, trained on clean, blurred and noisy Voronoi images, were tested on clean, blurred and noisy test images. It is to be expected that the network that is specifically trained for a certain type of image also performs best on this type of images. Fig.5.7 shows that this is indeed the case. It also shows that the performance can be very bad for a type of image that the network is not trained for, like e.g. the results of the network trained on the clean Voronoi image for the blurred and noisy Voronoi images. Hence to train a neural network image filter the training images should be chosen very carefully. A neural network edge detector that performs good on all three test images (although less than the specialised networks) may be obtained by training with a composite image or several images.

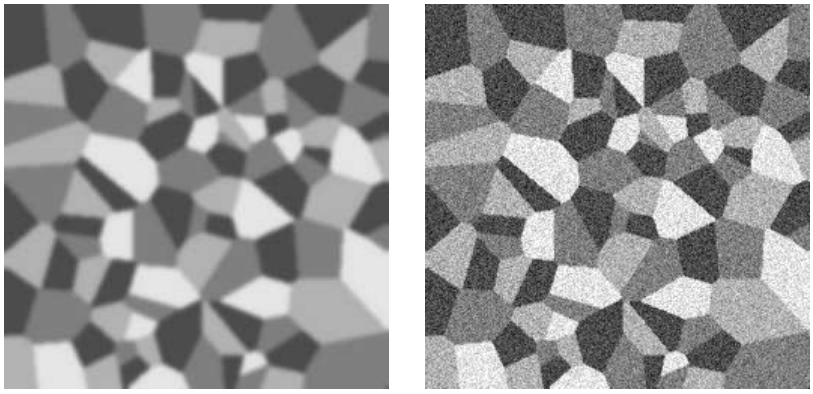


fig.5.6 Blurred and noisy Voronoi images used to train neural networks edge detectors for detection of edges in blurred and noisy images.

5.4.3 Training with natural image and hand edited edge reference map

As a final experiment a network was trained with the image of the blocks and hand edited edge map of fig.4.2. The results are shown in fig.5.8. There are two things that should be noted in the result of fig.5.8. The first is that in the hand edited edge reference map the shadow edges were deliberately removed. The neural edge detector did not detect the shadow edges either, conform its training. This shows again how easy neural networks can be tuned to detect a certain type of feature. The second important observation is that the neural network edge detector appears to have missed a substantial number of edges. One reason for this may be the inaccuracy of the edge reference map. As will be shown in chapter 6, the error backpropagation learning rule may fail if the edges in the edge reference map are not localised accurately.

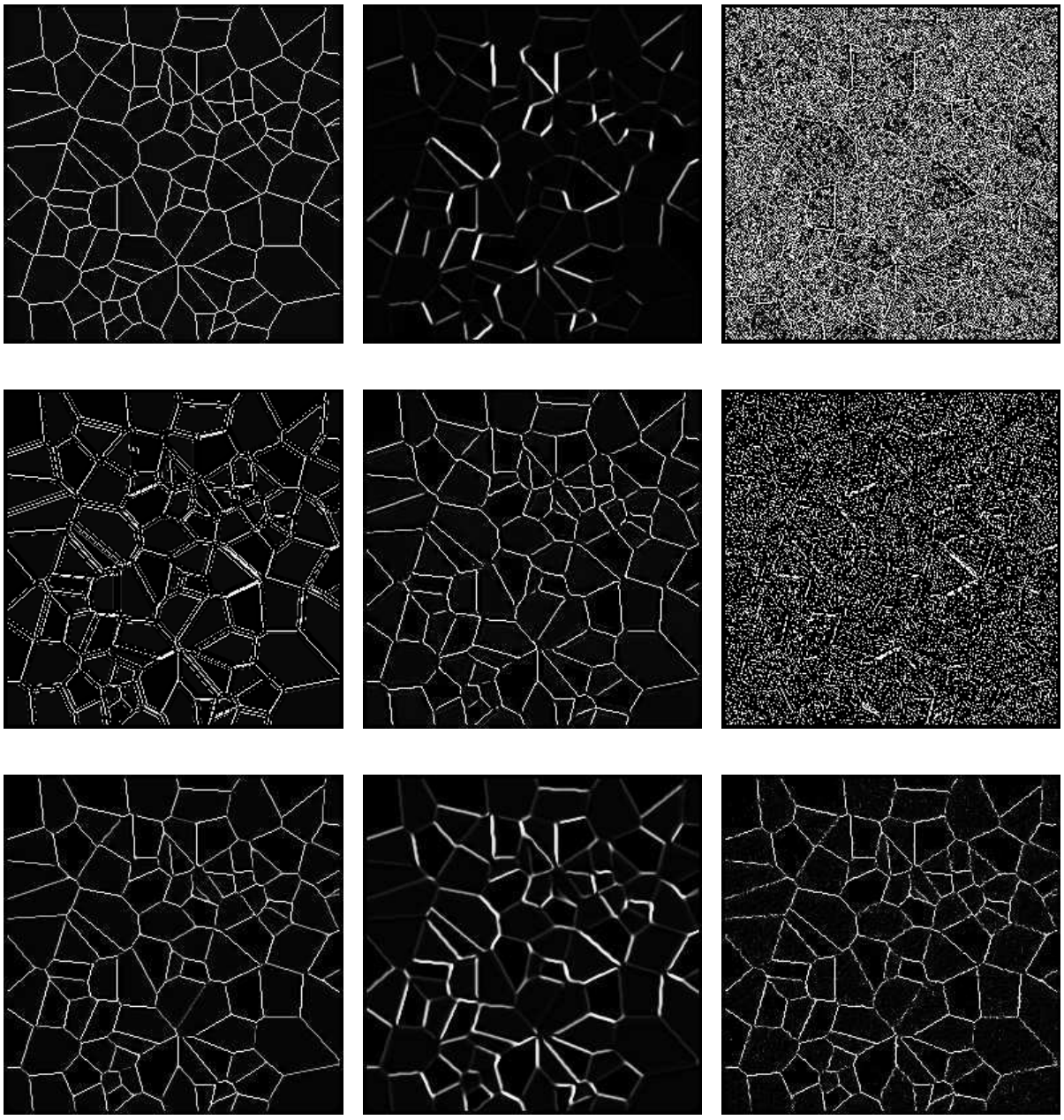
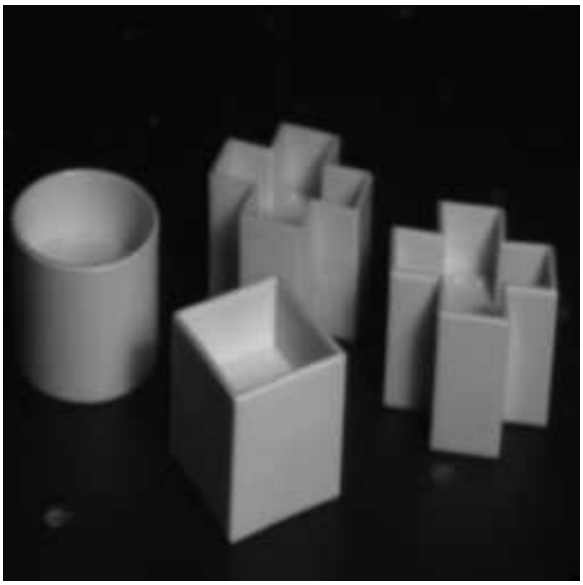
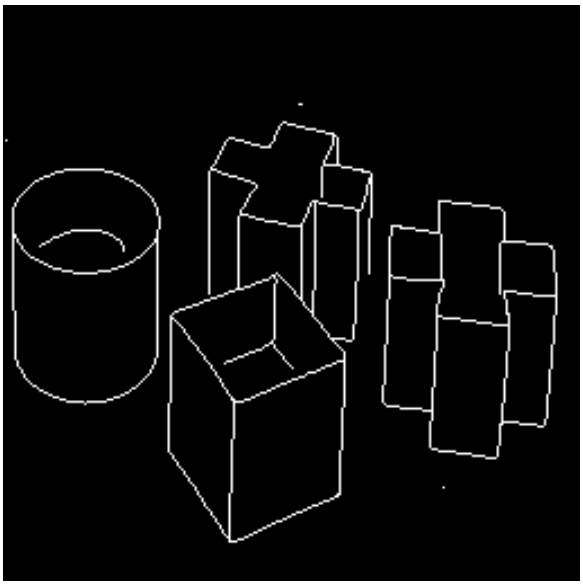


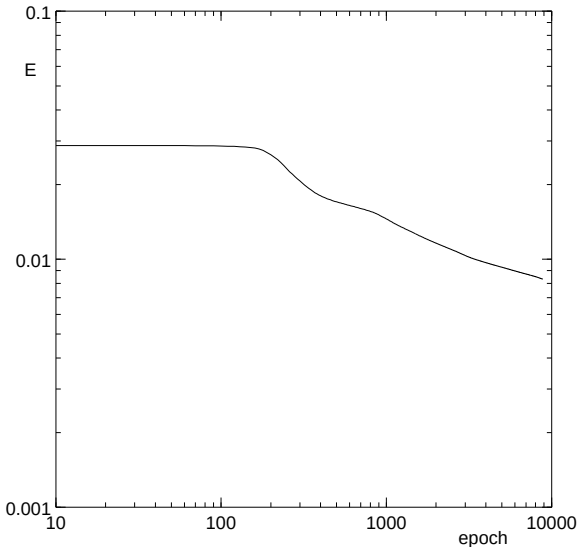
fig.5.7 Output images for three different test images of three neural network edge detectors, trained for different types of images. From top to bottom the output images of neural networks trained with resp. clean, blurred and noisy images; from left to right the output images of the respective networks for clean, blurred and noisy input images. Clearly the network that was trained for a specific type of image operates best on that type of image (images on the diagonal from top left to bottom right). In other words networks can be trained for a specific image filtering application by using the right training images.



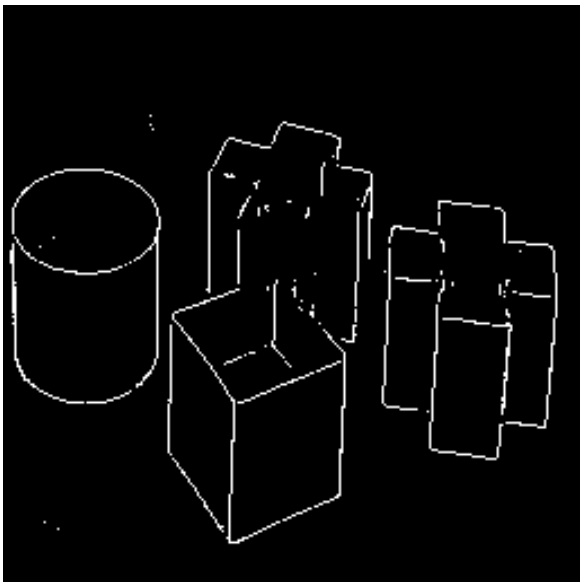
Natural training input image for edge detection.



Target output image obtained by editing the imperfect output of a Sobel edge detector.



Learning curve: squared error (vert.) vs. number of epochs (hor.)



Output of the resulting neural edge detector for the training input image.

fig.5.8 Natural image and hand edited target output image for training a neural network for edge detection (top) and learning curve and actual output image of the neural network after training (bottom). The network architecture was 49 inputs, 16 PE's in the second and 1 in the output layer. Learning rate was 0.01. It appears that hand edited target output images can be used for training neural networks. Note that in the target output image the shadow edges were deliberately left out and that the neural network edge detector does indeed not detect them. Unfortunately some other edges are also not detected.

5.4.4 Interpretation of the weights in the hidden layer

As described in the introduction of this section, the weighted sum of the processing elements in the hidden layer of a three layer network can be regarded as a convolution. The connection weights are the elements of the convolution kernels. Each processing element in the hidden layer can be regarded as an image filter that extracts certain (edge) features. It is often suggested that a spontaneous ordering of information takes place in neural networks, resulting in a kind of principle components of the signals being stored in the weights of the network.
In this case the architecture of the network very closely matches the architecture of the cvm operator of van der Heijden [Heijden 1992]. Van der Heijden obtained a number of orthogonal convolution kernels that can be regarded as a sort of principle components of edges. If the assumption about the spontaneous ordering is correct, one would expect similar types of convolution kernels for the neural network edge detectors.
The convolution kernels can be represented visually by showing the weight values as grey levels. An example of the kernels a rotation invariant cvm operator and the kernels of the three networks that were trained with Voronoi images are shown in fig.5.8.

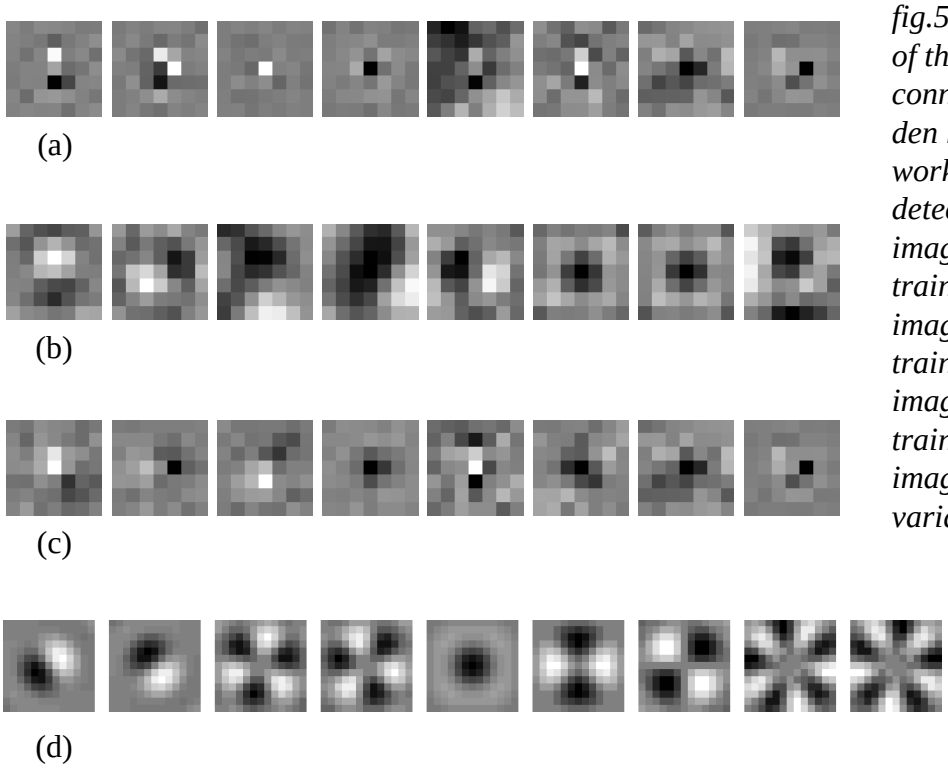


fig.5.9 Representation of the weights of the connections to the hidden layers of the networks trained for edge detection with Voronoi images. (a) network trained with clean image; (b) network trained with blurred image; (c) network trained with noisy image; (d) rotation invariant cvm kernels

Clearly the kernels of the three different networks differ. The kernels for edge detection in clean images show sharp changes in the centre, while the kernels for blurred images show slow gradients. The kernels for noisy images are rather chaotic.

Although some "principle components" can be recognised, compared to the kernels of the cvm operator the kernels of the neural networks look rather chaotic. An explanation could be that the "principle components" are not separated, but combined in the kernels of the neural networks. The neural networks for edge detection apparently do not spontaneously order the information, but distribute it in a rather chaotic way over all kernels. This could be analyzed by expanding the kernels into principle components. For the principle components the cvm kernels could be used. A more general approach is to expand the kernels in 0th, 1st, 2nd etc. order components using a Taylor series expansion of the two dimensional Fourier transform of the kernels. This results for each kernel in a description of its function in directional dependent 0th, 1st, 2nd etc. order behaviour.

5.4.5 Network architecture and size

The accuracy of a function approximation by an error backpropagation network depends on both the number of layers and on the number of processing elements in the different layers (very much like the accuracy of a polynomial function approximation depends on the order of the polynome). Theoretically with 3 adaptive layers every continuous function can be approximated arbitrary accurate, provided that there are enough processing elements. Theoretically, with more layers fewer processing elements are required to obtain the same accuracy of approximation. A drawback of using many layers is that learning speed may become very low, and convergence of the training process may become uncertain. The learning curves for a number of networks with different sizes are shown in fig.5.10. The networks were trained for edge detection with the noisy Voronoi images. It is clear that more complex networks learn slower, but reach a lower system error. Choosing a larger local neighbourhood results in increased learning speed and a lower system error. Apparently the complexity of the network only increases marginally, while the larger local neighbourhood allows better noise suppression.
The network with lowest system error (the 49-16-1 network) did not give better edge detection results than the other networks. This again shows the inadequacy of the least squares criterion for this types of problems.

5.5 Discussion

The approach to designing image filters with neural networks, that was taken in this section, was a very straight forward approach. Nevertheless the performance of the designed image filters seems to be quite acceptable.

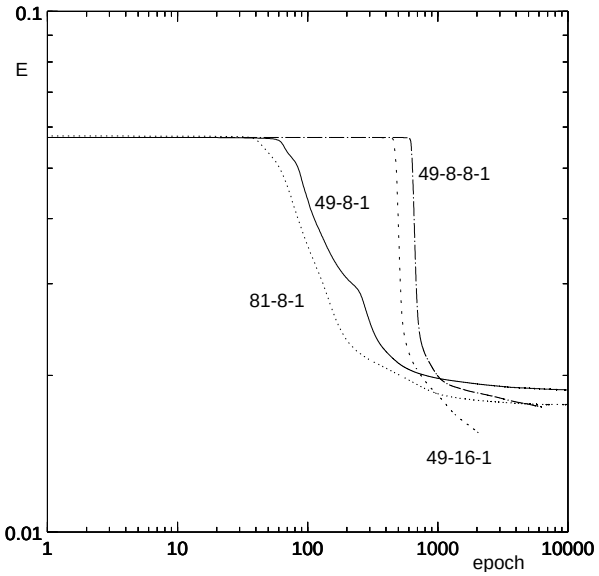


fig.5.10 Learning curves for error back-propagation networks with different network sizes. More complex networks appear to learn slower but also reach a lower system error.

Three important conclusions can be drawn:

- The ease of design appears to be one of the greatest advantages of using neural networks.
- The optimisation criteria that are used in neural network learning often do not correspond with the desired optimisation criteria.
- Learning times can be very long

6. Performance evaluation

6.1 Overview of this chapter

In this chapter performance evaluation of image filters is considered. A performance criterion based on the *average risk* (AVR) is proposed. The average risk is a performance criterion that incorporates both the probability on errors (or the frequency of errors) and the costs of the different types of errors. For an arbitrary input vector it expresses the expected total cost of the generated output. The optimal image filter is the one with the lowest expected total cost. This is often called the *minimum cost* solution for short. Evaluation of image filters consists of finding the filter with minimum AVR.

In section 6.2 first expressions for the AVR will be derived for both discrete and continuous output. In section 6.3 cost functions are discussed and how to obtain the probabilities on (or frequencies of) errors from images. An example of an evaluation method for edge detectors, based on AVR is worked out in section 6.4. The best approach to design image filters would be if they were optimised directly for minimum AVR. The relation between the optimisation criterion of backpropagation and training a neural network for minimum average risk are considered in section 6.5. Section 6.6 contains a summary and discussion.

6.2 Average risk - in depth

The definition of average risk as a weighted sum of probabilities on errors in chapter 2, was presented in a rather ad hoc way. In this section it is defined more precisely and expressions for the AVR in known quantities are derived. Two different cases are distinguished: AVR for continuous and for discrete output. The latter results in a countable number of errors. To each different type of error a cost can be assigned, and the formula for average risk of 2.3.5-1 is obtained. For continuous output, the domain of possible errors is also continuous, hence the summation of the weighted probabilities

on errors becomes an integral. First an expression for the average risk for discrete output will be derived in paragraph 6.2.1. This derivation is comparable to the derivation given by Devijver and Kittler for the average risk of classificators [Devijver 1982], but it is somewhat more general. In paragraph 6.2.2 the derived expressions are generalised for continuous output.

6.2.1 Average risk for operators with discrete output

Consider a function F (e.g. a neural network), that maps an input vector x in the input domain I on an output vector $y = \omega_i$, in the output domain Ω. The probability that the output should have been ω_j for this input vector is denoted $P(\omega_j|x)$. With this particular erroneous mapping a cost $\lambda(\omega_i|\omega_j)$ is associated. The expectation of the loss or risk of the mapping of x on $y = \omega_i$ is then equal to:

$$l_i(x) = \sum_j \lambda(\omega_i|\omega_j) P(\omega_j|x) \qquad (6.2.1\text{-}1)$$

Where: $l_i(x)$ = the expected loss or risk for the mapping of x on ω_i
$\lambda(\omega_i|\omega_j)$ = the loss entailed by an output ω_i where it should have been ω_j
$P(\omega_j|x)$ = the probability that the correct output for x is ω_j
the summation is over all possible output vectors ω_j

To obtain the average of the expected loss or average risk for the mapping ω_i the integral over all input vectors that are mapped on ω_i must be taken:

$$L_i = \int_{x|\omega_i} l_i(x) p(x) dx \qquad (6.2.1\text{-}2)$$

Where: L_i = the average risk for the mapping ω_i
$p(x)$ = the unconditional probability distribution function of x
the integral is over all input vectors x that are mapped on the output vector ω_i

Substituting (6.2.1-1) into (6.2.1-2) results in:

$$L_i = \int_{x|\omega_i} \sum_j \lambda(\omega_i|\omega_j) P(\omega_j|x) p(x) dx = \sum_j \lambda(\omega_i|\omega_j) \int_{x|\omega_i} P(\omega_j|x) p(x) dx \qquad (6.2.1\text{-}3)$$

And since:

$$\int_{x|\omega_i} P(\omega_j|x) p(x) dx = P(\omega_j, \omega_i) \qquad (6.2.1\text{-}4)$$

This reduces to a simple formula for the average risk of the mapping ω_i:

$$L_i = \sum_j \lambda(\omega_i|\omega_j)P(\omega_j,\omega_i) \qquad (6.2.1\text{-}5)$$

Where: $P(\omega_j,\omega_i)$= the probability that an input vector should be mapped on ω_j and is actually mapped on ω_i

The total average risk of the function *F* is the sum of all the average risks of the individual mappings:

$$L(F) = \sum_i L_i = \sum_i \sum_j \lambda(\omega_i|\omega_j)P(\omega_j,\omega_i) \qquad (6.2.1\text{-}6)$$

Where: $L(F)$ = the average expected loss or average risk of the function *F*
$\lambda(\omega_i|\omega_j)$ = the loss entailed by an output ω_i where it should have been ω_j
$P(\omega_j,\omega_i)$= the probability that an input vector should be mapped on ω_j and is actually mapped on ω_i
the summations are over all possible output vectors ω_i and ω_j

The probabilities $P(\omega_j,\omega_i)$ are the probabilities on the errors that may occur. Each ω represents an vector in the output domain Ω. The errors that may occur can be ordered in different types of errors. The probability on a certain error type is the sum of the probabilities on the different realisations of the error type. If the error type is called ε_i, and the associated cost λ_i, equation 2.3.5-1 is obtained:

$$AVR = \sum_i P(\varepsilon_i)\lambda_i \qquad (6.2.1\text{-}7)$$

6.2.2 Average risk for operators with continuous valued output

The derivation of the average risk for continuous valued output is almost identical to the case of discrete valued output. For continuous valued output, the summations over the output vectors become integrals. The risk of the mapping of an input vector on an output vector ω_i is then equal to:

$$l_i(x) = \int_\Omega \lambda(\omega_i|\omega_j)p(\omega_j|x)d\omega_j \qquad (6.2.2\text{-}1)$$

Where: $l_i(x)$ = the expected loss or risk for the mapping of *x* on ω_i
$\lambda(\omega_i|\omega_j)$ = the loss entailed by an output ω_i where it should have been ω_j
$p(\omega_j|x)$ = probability distribution function of the correct output for given *x*
the integration is over the whole output domain Ω

To obtain the average risk for the mapping ω_i integration over all input vectors that are mapped on ω_i is required:

$$L_i = \int_{x|\omega_i} l_i(x)p(x)dx \qquad (6.2.2\text{-}2)$$

Where: L_i = the average risk for the mapping ω_i
$p(x)$ = the unconditional probability distribution function of x
the integral is over all input vectors x that are mapped on the output vector ω_i

Substituting (6.2.2-1) into (6.2.2-2) results in:

$$L_i = \int_{x|\omega_i} \int_{\Omega} \lambda(\omega_i|\omega_j)p(\omega_j|x)p(x)d\omega_j dx = \int_{\Omega} \lambda(\omega_i|\omega_j) \int_{x|\omega_i} p(\omega_j|x)p(x)d\omega_j dx \qquad (6.2.2\text{-}3)$$

Next substituting:

$$\int_{x|\omega_i} p(\omega_j|x)p(x)dx = p(\omega_j,\omega_i) \qquad (6.2.2\text{-}4)$$

yields the formula for the average risk of the mapping ω_i:

$$L_i = \int_{\Omega} \lambda(\omega_i|\omega_j)p(\omega_j,\omega_i)d\omega_j \qquad (6.2.2\text{-}5)$$

Where: $p(\omega_j,\omega_i)$ = the probability that an input vector should be mapped on ω_j and is actually mapped on ω_i

The total average risk of the function F is the integral over the whole output domain of eq.6.2.2-5:

$$L(F) = \int_{\Omega} L_i d\omega_i = \int_{\Omega\Omega} \lambda(\omega_i|\omega_j)p(\omega_j,\omega_i)d\omega_i d\omega_j \qquad (6.2.2\text{-}6)$$

Where: $L(F)$ = the average expected loss or average risk of the function F
$\lambda(\omega_i|\omega_j)$ = the loss entailed by an output ω_i where it should have been ω_j
$p(\omega_j,\omega_i)$ = the probability that an input vector should be mapped on ω_j and is actually mapped on ω_i

The last formula expresses the average risk of the operator F as a function of a cost function $\lambda(\omega_i|\omega_j)$ and a probability density function $p(\omega_j,\omega_i)$. The cost function λ is a function of the two continuous vector variables ω_i and ω_j. The probability distribution function $p(\omega_j,\omega_i)$ expresses the probability on the simultaneous occurrence of actual output ω_i and target output ω_j. For continuous output signals it is often more convenient to express an error as a distance between target and actual output. Here distance

not necessarily means Euclidian distance, but can be any distance measure. If the distance between target output ω_j and actual output ω_i is denoted $\Delta(\omega_i,\omega_j)$, the expression for the AVR of an operator becomes:

$$AVR = \int_{\omega_i}\int_{\omega_j} \lambda(\Delta(\omega_i,\omega_j))p(\Delta(\omega_i,\omega_j))d\omega_i d\omega_j = \int_{\Delta} \lambda(\Delta)p(\Delta)d\Delta \qquad (6.2.2\text{-}7)$$

Where: Δ = the distance between target and actual output
$p(\Delta)$ = the probability density function of the distance Δ
$\lambda(\Delta)$ = the cost of a distance Δ between target and actual output

Eq.6.2.2-7 has the same form again as eq.2.3.5-1.

6.2.3 Average risk and the squared error measure

Without losing generality, in eq.6.2.2-7 a distance measure can be chosen that is equal to the cost function. The AVR then simplifies to the expectation of the distance:

$$AVR = \int_{\Delta} \Delta p(\Delta)d\Delta \qquad (6.2.3\text{-}1)$$

A popular distance measure is the quadratic distance measure or squared error:

$$\Delta = |\ \omega_j - \omega_i\ |^2 \qquad (6.2.3\text{-}2)$$

The system error of the error backpropagation learning rule actually is, except for a multiplication factor, an estimation of the expectation of the squared error (mean square error):

$$E = \frac{1}{2}\sum_p \sum_i (t^{pi}-y^{pi})^2 = \frac{1}{2}\sum_p |\ t^p-y^p\ |^2 \approx \frac{N}{2}E\{\ |\ t^p-y^p\ |\ \} \qquad (6.2.3\text{-}3)$$

Where: E = the system error of an error backpropagation network
t^p = the target output vector for input vector x^p
y^p = the actual output vector for input vector x^p
$E\{\}$ = the expectation

Thus the error backpropagation learning rule is an optimisation method for minimisation of the AVR with a quadratic cost criterion. The resulting solution is often called the *least squares solution*.

6.3 Average risk for image filters

6.3.1 Obtaining the probabilities on errors

It is generally impossible to calculate the probabilities on errors for an image filter. The alternative to calculation is estimation from test images. For the estimation of the probabilities on errors both the actual and the ideal output images must be available. The image generation methods described in chapter 4 can be used to obtain the test images.
There are three possibilities to evaluate the output image of an image filter:

- Evaluation of the output image as a whole
- Evaluation on a per pixel basis
- Evaluation on basis of local neighbourhoods

In the first approach the distance between two whole images is to be measured. The approach has two major problems. Firstly defining a meaningful distance measures for whole images maybe difficult. Secondly the expectation of the distances must be determined, which requires many realisations. It is unlikely that this can be accomplished in reasonable time.
The second approach compares each pixel in the filter output image with the corresponding pixel in the ideal output image. Since there are many pixels in an image, it is likely that the expectation of the distance between the pixel levels can be estimated reliably. This approach was used in the examples of the error backpropagation network filters in chapter 5. As already mentioned there, for the noise filter this seems to be a good evaluation measure, because the mean square error is an estimation of the variance of the noise in the image. This approach is however only valid if the filter output image can be regarded as the ideal output image plus an additive Gaussian noise term, like the noisy input image. As remarked in the edge detector example evaluation on a per pixel basis is rather limited. If e.g. an image filter must detect features that extent over an area in the image, the coherence of the pixels in the corresponding area in the output image should be included in the distance measure. The third approach, evaluation on basis of local neighbourhoods, allows both the incorporation of this local coherence in the distance measure and reliable estimation of the expectation of the distance, since an image contains many local neighbourhoods. For evaluation of most image filters the evaluation based on local neighbourhoods is therefore the preferable approach. A worked out example of such an evaluation method for edge detectors is described in section 6.4.

6.3.2 Choice of cost function

The costs that should be assigned to the types of errors that may occur in the filter output image depend on the further processing of the output image. For image enhancement this may require an accurate model of the human visual system [Bosman 1990]. For an image analysis system this may require evaluation of the whole image analysis system, because the performance of an image processing subsystem is only defined in relation with performance of the whole system. For the design of image filters however this may become very cumbersome. A better approach is therefore to analyze the errors that can occur in the filtered output image and their effect on the output of the image analysis system. In fig.6.1 a possible setup to find out the relation between the errors of an edge detector and the errors of the output of an image analysis system is given. In this case the dependency of the accuracy of the measurement of the position of a cube on the localisation and detection errors of an edge detector are investigated. Based on this relation, a cost function can be chosen for the different types of errors.

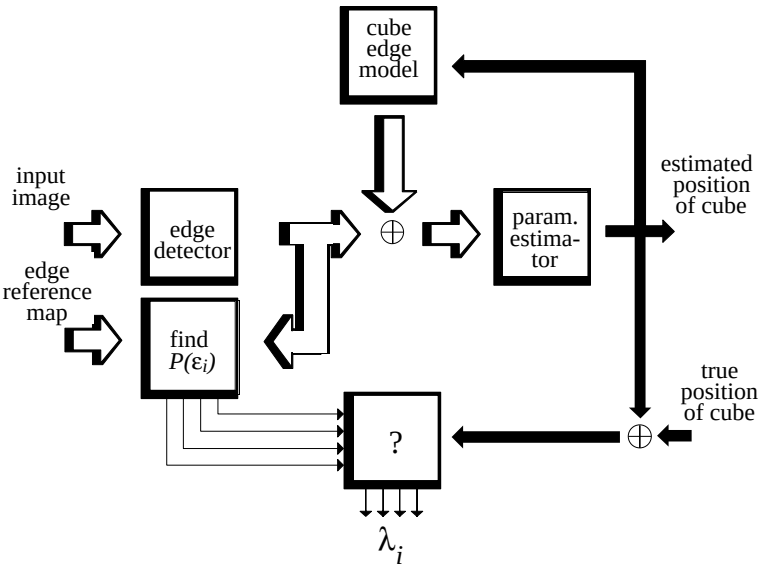


fig.6.1 Experimental setup to determine the relation between errors of an edge detector and the accuracy of the position estimation of a cube. Based on this relation a cost function for evaluation of edge detectors can be chosen that matches this problem.

The edge detector with the minimum average risk with this cost function, is the optimal edge detector for this application. This approach to finding cost functions for the AVR evaluation method is not worked out further yet.

6.4 Edge detector performance evaluation with AVR

The edge detector evaluation method proposed in this section builds on evaluation methods proposed by Besuijen and van der Heijden [Heijden 1989] and van der Heijden [Heijden 1989] and [Heijden 1992].

For the evaluation method proposed in this section, an edge detector is considered as an operator that classifies pixels in a digital image as edge or not-edge pixels. For evaluation one has to find out which pixels are classified incorrectly and in what way.

6.4.1 Error types

Consider an edge detector that can classify pixels in an image as edge or not-edge. Then there are two basic types of errors:

- pixels that are incorrectly labelled edge (spurious edges pixels)
- pixels that are incorrectly labelled not-edge (missed edges pixels)

A pixel is incorrectly assigned the label edge by an edge detector, if this label is not the result of any of the physical edges that are to be detected. If a physical edge does not cause the edge detector to label pixels as edges, the pixels that should have been labelled are incorrectly labelled as not-edge pixels. For the further processing of the edge map the nature of the spurious and missed edge pixels is important. If they appear isolated, the spurious edge pixels can generally be removed easily and edge segments with isolated missed edge pixels can easily be completed (an edge segment is a number of linked edge pixels). Spuriously detected edge pixels that appear in clusters or missing a larger part of an edge segment are much harder to cope with.
If an edge label is induced by a physical edge, the detection is in principle correct. However, the detection process may still be imperfect. E.g. the pixel labelled as edge can be at a location that does not correspond exactly with the location of the projection of the physical edge on the image plane (localisation error). An other error that may occur is that multiple responses are induced by a single physical edge, resulting e.g. in an edge segment that is more than a single pixel wide.
Summarising the following error types can be distinguished for edge detection:

- isolated spuriously detected edge pixels
- clustered spuriously detected edge pixels
- isolated missed edge pixels
- clustered missed edge pixels (missed edge segments)
- localisation errors
- edge segments more that 1 pixel wide

Depending on the application, other types of errors may have to be added to the list. E.g. some edge detectors also generate an estimation of the angle of the edge in the image plane. In that case the error in the angle can be added to the list. It may also be necessary to subdivide the individual types of errors further, e.g. in pixels at corners and in the middle of edge segments.

6.4.2 Estimation of the probabilities on errors

One method to obtain the probabilities on the different types of errors is to estimate these probabilities from test images. The probability on a certain type of error can be estimated by counting the number of pixels that are classified incorrectly according to the concerned type of error and dividing this number by the total number of pixels in the image:

$$P(\varepsilon_i) \approx \frac{N_{\varepsilon_i}}{N} \qquad (6.4.2\text{-}1)$$

Where: N_{ε_i} = the number of pixels classified incorrectly according to error type ε_i
N = the total number of pixels in the image

To estimate the probabilities on errors a test image, processed by the edge detector under test (detected edge map), is needed and a method to detect the different types of errors in the detected edge map. The latter can be done by comparing the detected edge map with an ideal edge map: the edge reference map. Hence for evaluation an image is required for which the edge reference map is available. The choice of the test image depends on the type of images that may be expected in the application of the edge detector. In fig.6.2 a scheme of edge detector evaluation with AVR using test images is shown.
Next assume that two binary images are available: the edge reference map with the exact positions of the edges, and the detected edge map, the output of the edge detector under test. The method to detect the different types of errors involves the following stages:

- finding correctly detected edge pixels
- finding spuriously detected edges pixels
- finding missed edge pixels
- determining the displacements of the correctly detected edge pixels
- determining the thickness of the edge segments
- determining the clustering of spuriously detected edge pixels
- determining the size of missed edge segments

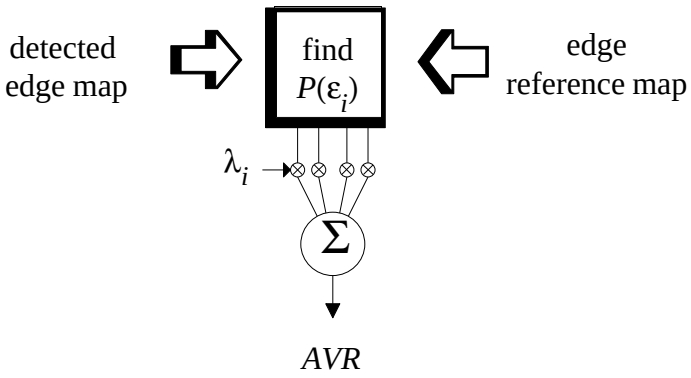


fig.6.2 Edge detector evaluation using AVR: a test image is processed by the edge operator under test. The resulting detected edge map is compared to the corresponding edge reference map. From this comparison the probabilities on the different types of errors are estimated. The sum of the probabilities, weighted with their individual costs yields the average risk.

In the first step, finding the correctly detected edge pixels, the pixels labelled edge in the detected edge map must be related with the edge pixels in the edge reference map. One approach would be to analyze the edge detector in depth, and for each edge pixel in the test image determine its response. However, this is only possible for very simple edge detectors and test images (such an attempt was done by Abdou and Pratt, see [Abdou 1979]). In order to be able to evaluate arbitrary edge detectors with arbitrary test images, the edge detector under test should be considered as a black box. The only thing that can therefore be said about detected edge pixels, is that they are probably detected correctly if they are localised close to a true edge, i.e. at or close to the corresponding position in the edge reference map there is an edge pixel. Thus correctly detected edge pixels are defined as:

> *A pixel in the detected edge map is correctly labelled edge, if it is within a certain distance from an edge pixel in the edge reference map.*

The distance mentioned in the above definition defines the maximum displacement that is accepted. If a pixel in the detected edge map is labelled edge, and it is not within a certain distance from an edge pixel in the edge reference map, it is considered a spuriously detected edge pixel. Hence the following definition for spuriously detected edges:

> *A pixel in the detected edge map is incorrectly labelled edge, if it is not within a certain distance from an edge pixel in the edge reference map.*

A missed edge segment occurs if there are no pixels labelled edge in the detected edge map within a certain distance from a segment in the edge reference map. Thus the pixels at the corresponding position in the detected edge map of this edge segment are missed edge pixels. However isolated missed edge pixels cannot be found in this way, because there are pixels labelled edge in the detected edge map close to the relevant edge pixel in the edge reference map. Both the missed edge segments and the isolated missed edge pixels can however be found by first completing the detected edge map, using the edge reference map, to make it look as closely as possible like the edge reference map. This includes adding the missed segments and filling up gaps in edge segments that result from isolated missed edge pixels. If the detected edge map is subtracted from the completed detected edge map, the missed edge pixels are obtained. Thus missed edge pixels can be defined as:

> *Missed edge pixels are those pixels that must be added to the detected edge map to make it look as closely as possible like the edge reference map with the exception of displacements and spurious edges.*

The displacement of edge segments can be determined by considering the displacements of the individual detected edge pixels. The displacement of a pixel that is correctly labelled edge in the detected edge map, is the distance to the closest edge pixel in the edge reference map. If the detected edge pixel is part of an edge segment that is more than 1 pixel wide, only the displacements of the pixels on the heartline of the detected edge segment should be considered.

> *Edge segments in the detected edge map displaced over a certain distance, can be found by measuring the distance of the pixels on the heartline of the segments to the closest edge pixel in the edge reference map.*

The width of an edge segment in the detected edge map at a certain position in the edge segment is the width of the edge segment measured in an intersection perpendicular to the segment. In this way each pixel of an edge segment can be assigned a local width of the edge segment.

> *The width of edge segments in the detected edge map can be found by determining the width at the position of each edge pixel in the segment of an intersection perpendicular to the segment.*

Isolated spuriously detected edge pixels are easily found, because they have no neighbouring spuriously detected edge pixels. Clustered spuriously detected edge pixels are those that are not isolated. It is possible to distinguish different types of clustering, e.g.

line segments or blobs. The former may be more difficult to handle in further processing. In the evaluation method proposed here, this distinction will not be made. Hence isolated and clustered spuriously detected edges are defined as:

> *Isolated spuriously detected edge pixels are spuriously detected edge pixels that have no neighbouring spuriously detected edges. Clustered spuriously detected edges are not isolated.*

Likewise only two different sizes for missed edge segments are distinguished: 1 pixel and larger than one pixel. For convenience these will be called isolated and clustered missed edge pixels.

> *Isolated missed edge pixels are missed edge segments with a size of one pixel. Clustered missed edge pixels are parts of missed edge segments larger than one pixel.*

In the next paragraph methods are proposed to determine the errors in the detected edge map using the definitions given in this paragraph.

6.4.3 Implementation

In order to keep the implementation as simple as is possible, only binary logical and morphological operations were used whenever possible. Also all distance measures are rounded to pixels, which means only displacements and widths of 1,2,3 etc. pixels can be distinguished.

The correct detected edge pixels in the detected edge map are localised within a distance of *n* pixels of the edge pixels in the edge reference map can be found by dilating the edge reference map *n* times and then performing a logical and function of the dilated edge reference map with the detected edge map. If an edge segment in the detected edge map is more than one pixel wide, part of the edge segment may be cut off in this way. Therefore all detected edge pixels that are connected to the detected edge pixels that are within a distance of *n* pixels of edge pixels in the edge reference map are added. This process is called propagation. The resulting map of correctly detected edge pixels will be called the *correct detected edge map*. Fig.6.3 shows schematically how the correct detected edge map can be obtained using dilation, logical and propagation functions.

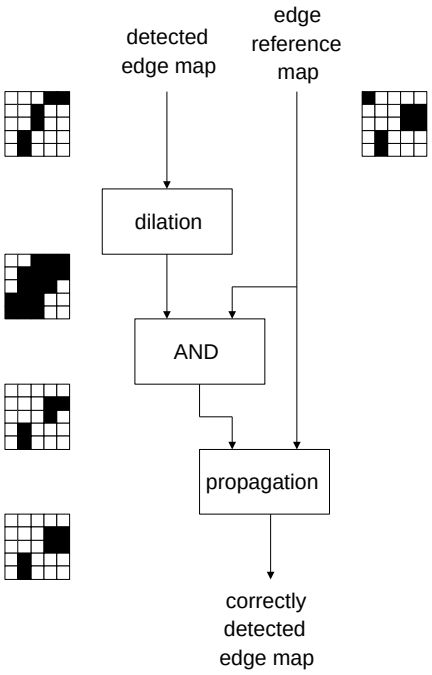


fig.6.3 Determining the correcly detected edge pixels. The detected edge pixels within a distance n of edge pixels in the edge reference map are found by a logical and of the n times delated edge reference map and the detected edge map. The result is propagated to add parts of thick segments that are cut of by the logical and.

The spuriously detected edge pixels are the detected edge pixels that are in the detected edge map, but not in the correct detected edge map. These pixels can therefore be found using a logical and of the detected edge map with the inverted correct detected edge map, see fig. 6.4.

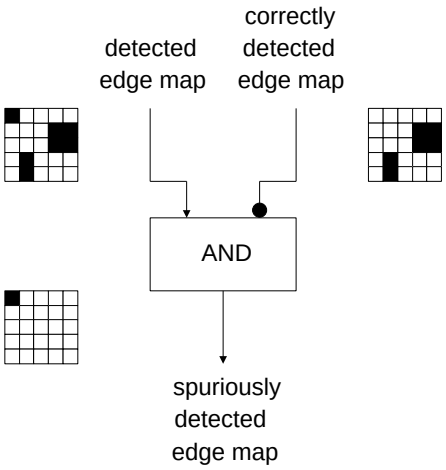


fig.6.4 Spuriously detected edges pixels are the detected edge pixels that are in the detected edge map but not in the correct detected edge map.

The missed edge pixels are those pixels that must be added to the detected edge map to fill up gaps in edge segments and add missing edge segments. The approach taken here is first to fill up the gaps and add the missing edge segments in the detected edge map, resulting in the *completed edge map.* Then the detected edge map is subtracted from the completed edge map to obtain the missed edge pixels. The completed edge map can be obtained from the detected edge map using the edge reference map. First the edge reference map is dilated *n* times, where *n* is again the maximum accepted dis-

placement. The dilated edge reference map is combined using a logical or with the correct detected edge map. The result is a super set of the completed edge map. The completed edge map can be obtained by skeletonising this map, removing all superfluous, but none of the edge pixels of the correct detected edge map. The missing edge pixels are the edge pixels that are in the completed edge map, but not in the correct detected edge map. Fig.6.5 shows the process of finding the missed edge pixels.

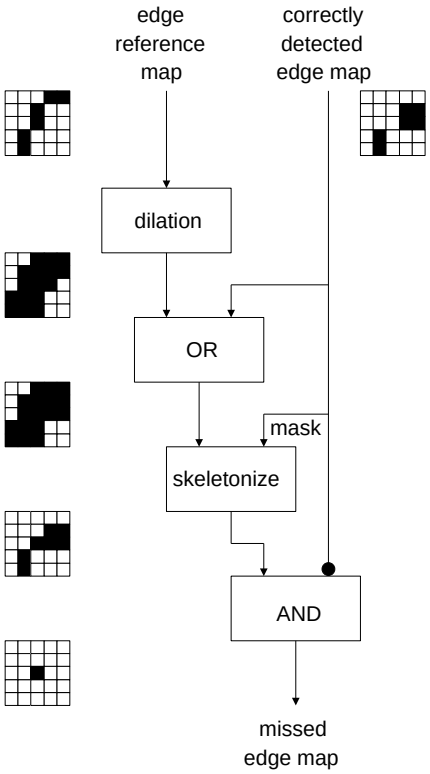


fig.6.5 Finding missed edges by first completing the detected edge map and then subtracting the detected edgemap from the completed detected ege map.

As described in the previous paragraph, only the displacements of pixels that are on the heartline of an edge segment should be counted. Therefore the correct detected edge map is first skeletonised, leaving a map with edge segments all one pixel wide at the heartline of the originally detected edge segments. The pixels that are at a distance of *d* pixels from the edge segments in the edge reference map, can be found by dilating the edge reference map *d* times and subtracting a *d*-1 times dilated edge reference map. This ring of pixels at a distance *d* of the edge segments is then used to mask out the edge pixels that have a displacement of *d* pixels in the skeletonised correct detected edge map. The process in illustrated in fig.6.6.

To determine the width of an edge segment at a certain position, the orientation of the edge segment at the considered position is required. This would involve an extra step of determining the orientation of the edge segment for each pixel in the edge segment. Since the intersection perpendicular to the edge segment is also the smallest intersection, the perpendicular intersection can be approximated by choosing the intersection

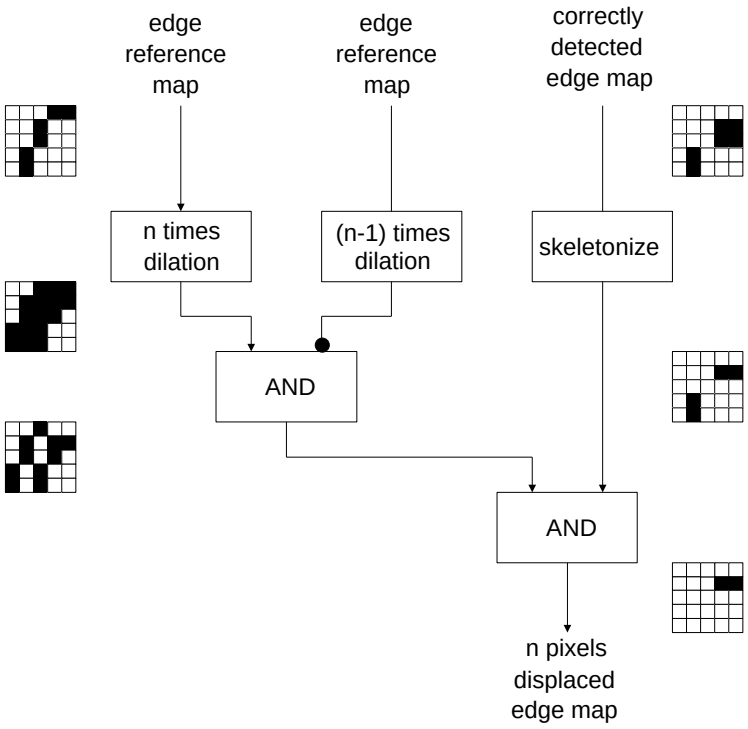


fig.6.6 Determination of the displacement of detected edge pixels at the heartline of edge segments.

with minimum width of a number of intersections with different angles. In this implementation four directions for the intersections were used: horizontal, vertical and two diagonal intersections (see fig.6.7). For the diagonal intersections $\sqrt{2}$ times the number of pixels rounded up is taken as the width.

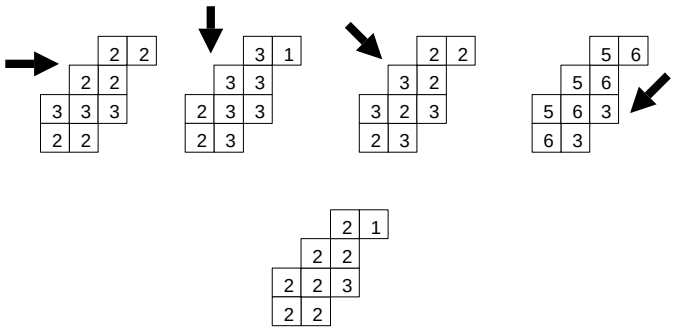


fig.6.7 Determination of edge thickness by taking the minimum width of four different intersections. The result is the figure at the bottom.

As described in the previous section, isolated spuriously detected edge pixels are spuriously detected edge pixels with no neighbouring edge pixels. A morphological operator that filters out isolated white pixels from a binary black and white image is called a salt operation. The detected edge pixels that remain are clustered edge pixels. The same method can be used to determine if missed edge pixels are isolated or clustered. Fig.6.7 illustrates finding clustered and isolated pixels using a salt operation.

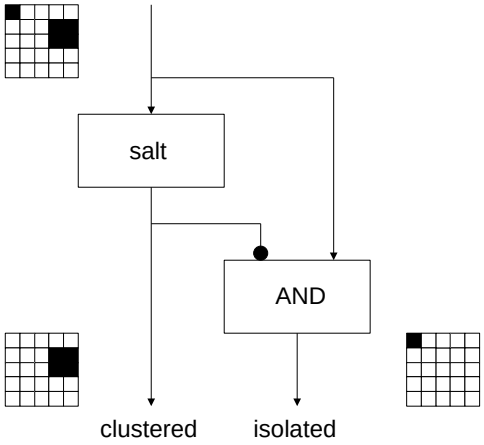


fig.6.8 Determination of clustered and isolated spuriously detected edge pixels and missed edge pixels using the salt operator.

6.4.4 Cost functions

As we are of the opinion that the performance of edge detectors cannot be regarded separately from the whole image processing system, no general cost function should be used. In paragraph 6.3.2 an experimental approach to finding a cost function for a specific application was proposed. Since we have no such a cost function available yet, a cost function based on heuristics about some general applications is proposed here to be able to do some evaluations. The results of the evaluations are therefore of limited value.

The most serious errors an edge detector can make are generally the clustered missed and spuriously detected edge pixels. They are therefore assigned the highest cost. Isolated missed and spuriously detected edge pixels are assigned half this maximum cost. The cost of displacements increase with the displacements. We assigned a linear increasing cost up to half the maximum cost at a displacement of 4 pixels. Likewise the cost of thickened edge segments increases with the width. The costs are assigned such that a 2 pixel thick edge segment has a cost of 1/8 of the maximum cost, a three pixel wide edge segment 1/4 etc. The costs for the different types of errors are summarised in the table below.

type of error		cost
clustered spurious edge		1
isolated spurious edge		1/2
clustered missed edge		1
isolated missed edge		1/2
displacement of:	1 pixel	1/8
	2 pixels	1/4
	3 pixels	3/8
	4 pixels	1/2
edge width:	2 pixels	1/16
	3 pixels	1/12
	4 pixels	3/32

table 6.1 Cost function used in the evaluations in paragraph 6.4.6

6.4.5 Comparison to other evaluation schemes

Most methods for performance evaluation of edge detectors (e.g. [Fram 1975], [Peli 1982], [Pratt 1978]) are more or less arbitrary combinations of different types of errors without. The average risk approach on the other hand is a structured approach to combining different types of errors into one error measure.
One of the first evaluation methods, and still the most widely used, is the method proposed by Pratt [Pratt 1978]. It measures edge location accuracy by a figure of merit rating factor defined by:

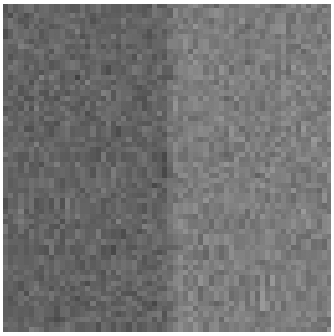
$$FOM = \frac{1}{I_N} \sum_{i=1}^{I_A} \frac{1}{1+\alpha d_i^2} \qquad (6.4.5\text{-}1)$$

Where: I_I = the number of ideal edge map pixels
I_A = the number of detected edge pixels (actual edge map pixels)
I_N = $\max(I_I, I_A)$
α = a scaling constant
d_i = the distance of an detected edge pixel to the closest ideal edge pixel

The FOM can take values between 0 and 1. For a perfectly detected edge map FOM=1. The scaling factor α may be adjusted to penalise edges that are localised but offset from the true position. In evaluations generally the value $\alpha = 1/9$ is used.
The FOM of Pratt requires a test image and an edge reference map in order to calculate the distances of the detected edge pixels to the closest ideal edge pixels. The test images generally used with Pratt's Fom are 64*64 pixel arrays with 256 grey levels and a single straight vertical, horizontal or diagonal edge, with a slope placed at its centre. Pratt uses an edge height of h=25 grey levels and an linear edge slope with width of w=1 pixel. Zero-mean Gaussian noise with a standard deviation of σ_n is added to the image. The signal to noise ratio is defined as:

$$SNR = \frac{h^2}{\sigma_n^2} \qquad (6.4.5\text{-}2)$$

A test image with noise $\sigma_n = 10$ and the corresponding edge reference map are shown in fig.6.9.



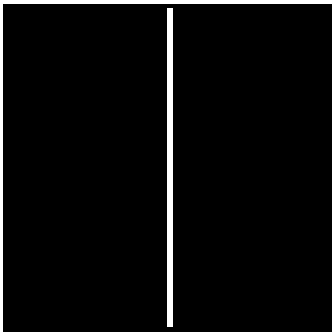


fig.6.9 Test images used for Pratt's FOM. Left: step edge with height 25,width 1 pixel and additive Gaussian noise with $\sigma_n = 10$*; right: the corresponding edge reference map.*

The edge detectors are tested for different SNR's. It is clear that the FOM of Pratt is far more restricted than the AVR evaluation method proposed here. The FOM of Pratt only measures the localisation errors (displacements) of edge pixels. The measure itself has no fundamental meaning like the average risk. And the test images mostly used with Pratt's FOM are not representative for most real life scenes. Therefore the average risk evaluation method proposed in this section is to be preferred if the performance of edge detectors for specific applications is to be measured. Incidently, the FOM of Pratt can be constructed from the AVR method, by choosing an appropriate cost function. Eq.6.4.5-1 can be rewritten as:

$$FOM = \frac{1}{I_N}\sum_d \frac{I_d}{1+\alpha d^2} = \frac{N}{I_N}\sum_d \frac{1}{1+\alpha d^2}\frac{I_d}{N} \approx \sum_d \lambda_d P(d) \qquad (6.4.5\text{-}3)$$

with:

$$\lambda_d = \frac{N}{I_N}\frac{1}{1+\alpha d^2} \qquad (6.4.5\text{-}4)$$

Where: I_d = the number of detected edge pixels with displacement d
N = the total number of pixels in the image
λ_d = the cost of a displacement of a detected edge pixel over d pixels
$P(d)$ = the probability on a displacement error of d pixels

In fig.6.10 the FOM of Pratt for the Marr-Hildreth edge detector [Marr 1980] is given for several operator widths as a function of SNR. Large operator widths improve the results on noisy images. It is well known, however, that the location errors for larger

operator widths increase for curved edges. Due to the nature of the test image this is not reflected by Pratt's FOM with the usual test images. From fig.6.10 one would conclude that increasing the width of the operator also increases its accuracy.

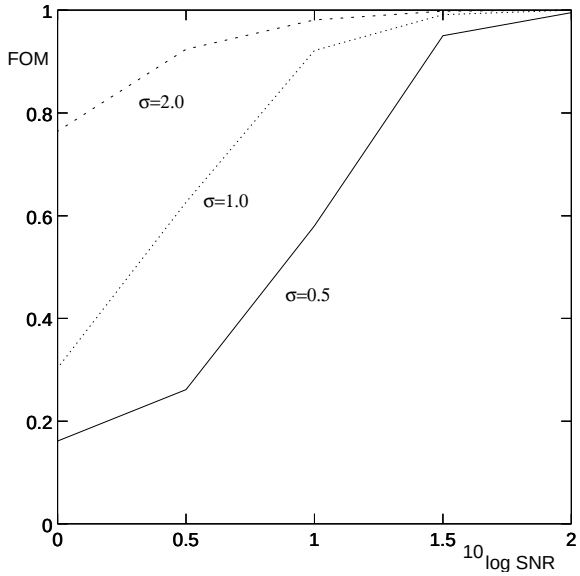


fig.6.10 FOM of Pratt for Marr-Hildreth edge detector for three different operator widths as a function of SNR. The curves suggest that larger operator widths give more accurate localisation. This however is not correct for curved edges.

In the next section a number of edge detectors is evaluated using the AVR performance measure. It is also shown that the accuracy of a Marr-Hildreth edge detector becomes worse for larger operator widths.

6.4.6 Comparison of several edge detectors using AVR

A number of edge detectors is compared here using the proposed AVR performance evaluation method and the cost function of paragraph 6.4.4. First the ability of the AVR evaluation method to find the strengths and weaknesses of edge detectors will be demonstrated by analyzing the localisation error as a function of operator width of the Marr-Hildreth edge detector. Next a number of edge detectors are compared using AVR. The considered edge detectors are:

- Sobel edge detector [Pratt 1978]
- Marr-Hidreth edge detector [Marr 1980]
- Canny edge detector [Canny 1986]
- Several neural network edge detectors

Two test images are used: the Voronoi test images with additive zero-mean Gaussian distributed noise. The AVR is plotted as a function of the standard deviation of the noise. And second the image with blocks and the hand edited edge reference map of fig.4.2. For all edge detectors a threshold was chosen that resulted in the lowest AVR.

In fig.6.11 the AVR curves of the Marr-Hildreth edge detector for three different operator widths is shown. From these curves it is clear that larger operator width results in worse performance for high SNR, but relatively better performance for low SNR.

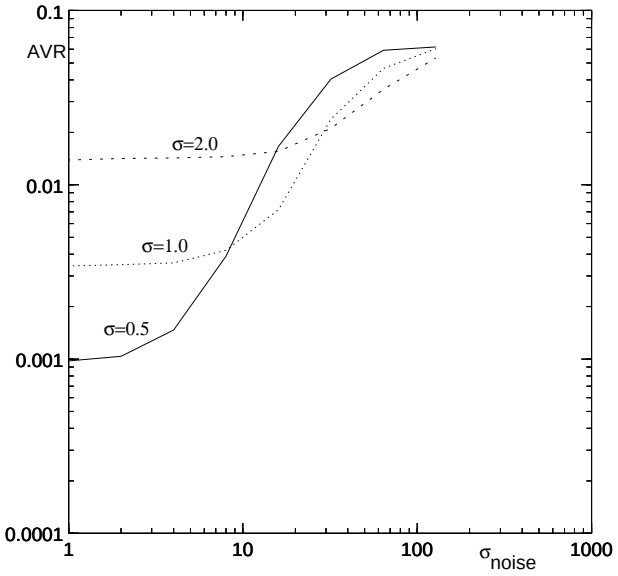


fig.6.11 AVR of Marr-Hildreth edge detector for three different operator widths. In contrary to the FOM of Pratt, the curves show clearly that increasing operator width decreases performance for high SNR. At low SNR the larger operator widths give better performance.

As already mentioned in the previous paragraph, the worse performance is primarily due to localisation errors. Since the probabilities on the different types of errors are obtained separately, this can be shown by plotting the AVR and the localisation error component of the AVR as a function of the operator width for a test image with fixed SNR (fig.6.12). It is clear that the displacement error increases with operator width, while at the same time the detection errors decrease. This uncertainty principle was formulated earlier by Canny [Canny 1986]. It has now been demonstrated experimentally (see also [Spreeuwers 1992].)

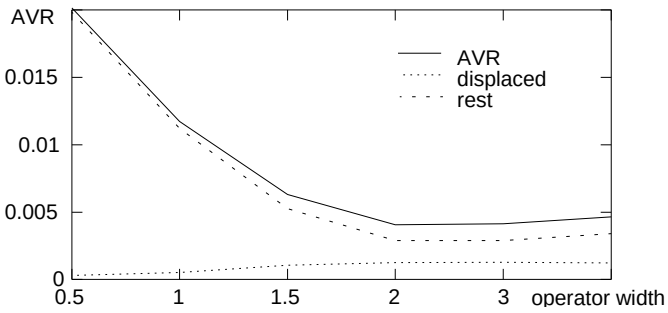


fig.6.12 AVR versus operator width. Clearly the displacement errors increase as a function of the operator width, while at the same time the detection errors decrease.

In fig.6.13 the AVR as a function of the standard deviation of the noise of the listed edge detectors is shown. The test images were the Voronoi images (but another realisation than the one the neural networks were trained with). The Canny and Marr-Hildreth operators were designed using edge models that are very close to the edges in these Voronoi test images: step edges with additive Gaussian noise. Therefore they

give near optimal results for these images for small values of the operator width: $\sigma_w \leq 1$, see fig.6.12 (see also the comparison of edge detectors in [Heijden 1992]). The Canny and Marr-Hildreth operators were tested for three different operator widths: σ=0.5, 1.0 and 2.0. The Sobel edge detector operates poorest at high SNR and is also very sensitive to noise. The neural network edge detector trained on a noise free image outperforms all other edge detectors at high SNR, but is also extremely sensitive to noise. For $\sigma_{noise} = 20$, the Canny operator with operator width 1 performs best, but the Marr-Hildreth operator and the neural network trained on a noisy Voronoi image perform almost as good. This shows that straightforward training of neural networks for image filtering is likely to result in near optimal image filters.

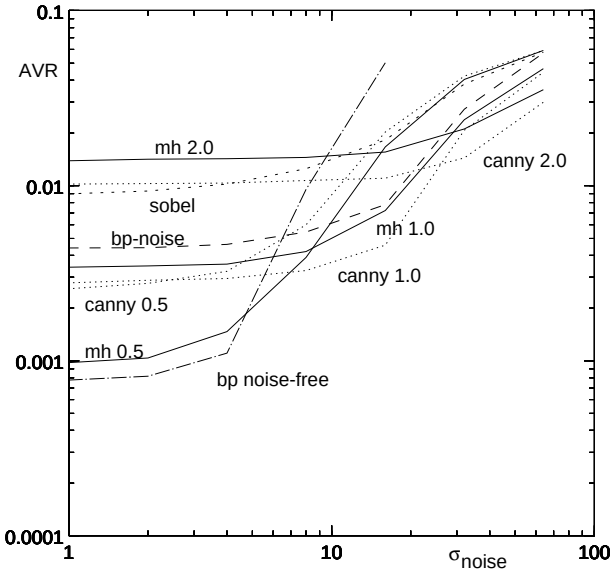


fig.6.13 AVR of a number of different edge detectors as a function of the standard deviation of the noise in the Voronoi test images.

In table 6.2 the AVRs for the mentioned edge detectors for the test image with the blocks and the hand edited edge reference map are given. Clearly the neural network that was trained specifically for this task outperforms all other edge detectors. This also shows that the models used for designing the Canny, Marr-Hidreth and Sobel edge detectors and also the Voronoi images are not very representative for real world scenes.

edge detector		AVR
neural net	blocks	0.00634
Canny	0.5	0.01351
Canny	1.0	0.01504
Canny	2.0	0.01596
Marr-Hildreth	0.5	0.01556
Marr-Hildreth	1.0	0.01588
Marr-Hildreth	2.0	0.01671
Sobel		0.01961
neural net	Voronoi	0.01851
neural net	Voronoi+blur	0.02121
neural net	Voronoi+noise	0.02722

table 6.2 AVR for several different edge detectors for the image with the blocks in fig.4.2. For the neural networks the training images and for the other edge detectors the operator widths are given in the second column.

The output images of the two edge detectors that gave the best results are shown in fig.6.14. Although this result seems already quite satisfactory, still the neural edge detectors can be improved, as is shown in the next section.

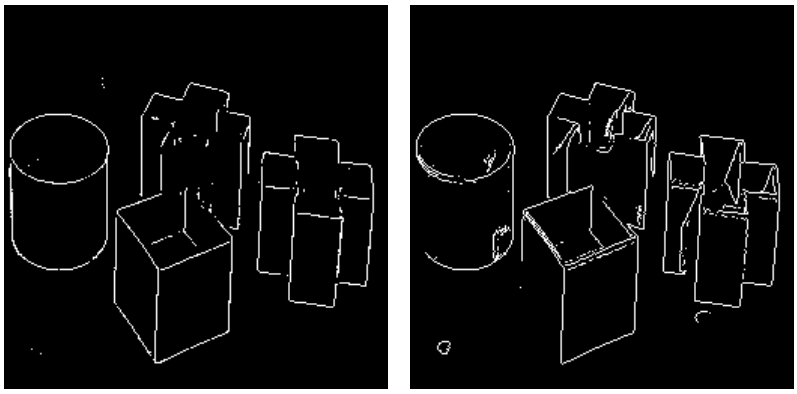


fig.6.14 Output image of the neural network (left) and Canny edge detector (right) that performed best on the image with the blocks.

6.5 Training neural networks for minimum average risk

We have defined the performance of an image processing algorithm using the average risk. The algorithm that has the lowest average risk for a certain application is the optimal algorithm. Ideally one should therefore try to minimise the average risk for the neural network image filters. In this paragraph it is investigated to what extent this is done for the error backpropagation network, and how the networks and learning rules can be adapted to obtain a minimum average risk solution.

6.5.1 Error backpropagation as unity cost criterion

It has already been shown in paragraph 6.2.3 that for continuous output the system error of the error backpropagation learning rule is equal to an AVR error measure with a quadratic cost criterion. In this section it is shown that for classification tasks (binary

outputs) the squared error is equal to the AVR with a unity cost criterion. This has been shown earlier by Devijver [Devijver 1973] and more recently in [Wan 1990] and [Ruck 1990].

The squared error that is minimised by the error backpropagation learning rule is:

$$E = \frac{1}{2} \sum_{p} \sum_{i} (t^{pi} - y^{pi})^2 \qquad (6.5.1\text{-}1)$$

Where: t^{pi} = element i of the target output for input pattern x^p
y^{pi} = element i of the actual output for input pattern x^p
The summations are over all patterns p and outputs i

Now consider a network that is to be used for a classification task (e.g. feature classification). Each output y^{pi} represents a class and should be 1 if the input pattern is of the corresponding class and 0 otherwise. Thus t^{pi} can only be either 0 or 1. Therefore rewrite eq.6.5.1-1 can be written as:

$$E = \frac{1}{2} \sum_{i} \left[\sum_{p|t^{pi}=0} (y^{pi})^2 + \sum_{p|t^{pi}=1} (1-y^{pi})^2 \right] \qquad (6.5.1\text{-}2)$$

Where the summation over $p|t^{pi}= 0$ means over all patterns for which the target output element i is 0.

The output of a error backpropagation network is limited by the sigmoid in the range <0,1>. If the network converges, then the outputs y^{pi} will be close to 0 or 1. The following approximation will then be valid:

$$\sum_{p|t^{pi}=0} (y^{pi})^2 \approx \sum_{p|t^{pi}=0} (y^{pi}) \approx N(y^i=1,t^i=0) \qquad (6.5.1\text{-}3)$$

Where $N(y^i=1,t^i=0)$ is the number of patterns in the training set, for which the actual output y^i is equal to 1, while the target output is zero. Likewise the second term in eq.6.5.1-2 results in $N(y^i=0,t^i=1)$.
Eq.6.5.1-3 can also be expressed in the combined probability $P(y^i=1 ,t^i=0)$:

$$N(y^i=1,t^i=0) = NP(y^i=1,t^i=0) \qquad (6.5.1\text{-}4)$$

Where: N = the total number of training samples

The combined probabilities $P(y^i=1,t^i=0)$ and $P(y^i=1,t^i=0)$ are the probabilities on the occurrence of a certain type of error. Eq.6.5.1-1 can now be written as a sum of different types of errors that can occur in the classification process:

$$E \approx \frac{N}{2} \sum_i P(y^i=1,t^i=0) + P(y^i=0,y^i=1) \qquad (6.5.1\text{-}5)$$

This is equal to the average risk of the classifier if the costs for all types of errors is chosen equal to $N/2$. Thus the error backpropagation learning rule results in a minimum average risk classifier, if the outputs are close to 0 and 1 and the costs for all errors are $N/2$.

Many applications of the error backpropagation network more or less satisfy this condition of equal costs for all types of errors. Also if the final error comes very close to zero, the result will always be close to the optimal (provided that the training set is representative). However, there exist also applications that require a different cost function, and of course not all applications are classification applications. For these problems it would be interesting to find out if the optimisation criterion for the error backpropagation learning rule can be adapted, so that it does minimise the average risk.

6.5.2 Adapting error backpropagation for other optimisation criteria

The derivation of the backpropagation learning rule by Rumelhart et al. [Rumelhart 1986], starts with the introduction of the least squares optimisation criterion (eq.6.5.1-1).

For a feed forward neural network they derived the following weight update rule for a gradient descent optimisation method:

$$\Delta w_k^{pij} = \eta \delta_k^{pi} o_{k-1}^{pi} \qquad (6.5.2\text{-}1)$$

Where:
- Δw_k^{pij} = the weight update of a connection between processing element j in layer k-1 to processing element i in layer k for an input pattern x^p
- η = the learning rate
- δ_k^{pi} = an error measure local to the processing element i in layer k for input pattern x^p
- o_{k-1}^{pj} = output value of processing element j in layer k-1 for input pattern x^p

The δ's in eq.4.6 are calculated backwards through the network (hence the name backpropagation). For the output layer (layer L) the δ's can be calculated directly from the error:

$$\delta_L^{pi} = (t^{pi} - y^{pi})\, f'(net_L^{pi}) \qquad (6.5.2\text{-}2)$$

For each lower layer, the δ's can be calculated from the δ's in the higher layers:

$$\delta_k^{pi} = \sum_h \delta_{k+1}^{pi} w_{k+1}^{hi} f'(net_k^{pi}) \qquad (6.5.2\text{-}3)$$

Where: $f'()$ = the derivative of the transfer function of the processing element
net_k^{pi} = the weighted sum of the inputs of processing element *i* in layer *k* for input pattern x^p
w_{k+1}^{hi} = the connection weight of the connection between processing element *i* in layer *k* to processing element *h* in layer *k*+1
the summation is over all processing elements *h* in layer *k*+1

Note that since *L* is the last (output) layer, $y^{pi} = o_L^{pi}$.

The question now is how do these equations change if another error measure than the one of eq.6.5.1-1 is used. Since the δ's in the lower layers are calculated from the δ's in the higher layers, it follows that the error measure is only explicitly present in the calculation of the δ's of the output layer. If the squared error measure is not substituted into the calculation of the δ's of the output layer, in stead of eq.6.5.2-2 the following is obtained:

$$\delta_L^{pi} = -\frac{\partial E}{\partial o_L^{pi}} f'(net_L^{pi}) \qquad (6.5.2\text{-}4)$$

This is a general formula for the calculation of the δ's of the output layer, for any error measure *E*. Thus other error measures can be applied with only a slight modification to the standard error backpropagation learning rule.
Of course *E* must be continuous and differentiable. Desirable properties of *E* are that it has a single global minimum and it descends smoothly to this minimum.

6.5.3 Example of training an error backpropagation network for minimum AVR

The usefulness of training an error backpropagation network for a minimum AVR will be demonstrated with a pulse detection problem in a one dimensional signal. Incidentally this problem has a lot in common with edge detection. Consider an error backpropagation neural network that is to be trained to detect pulses in a one dimensional digital signal that is corrupted by low and high frequency noise. In order to determine

if a sample of the digital signal is a pulse or not, the sample itself and the neighbouring sample are presented as input to the network, see fig.6.8.The single output of the network should be 1 of the sample is a pulse and 0 otherwise.

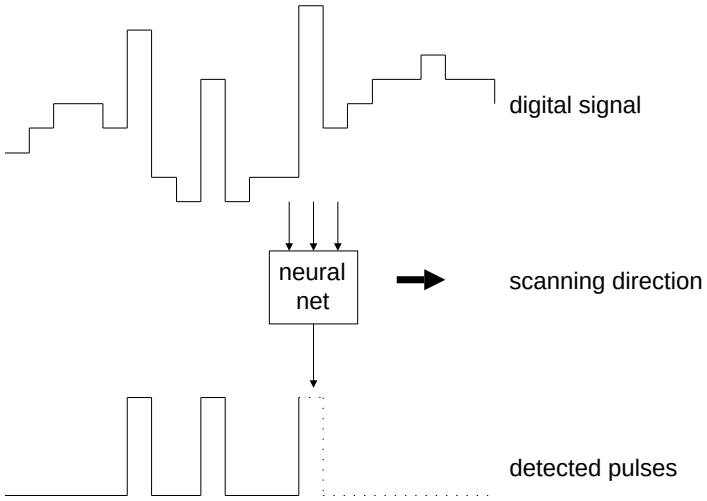


fig.6.15 Pulse detection in a one dimensional signal with an error backpropagation network.

For training the error backpropagation network a training set is required consisting of an example input signal and a target output signal. Now suppose training data is available in which the pulses in the target output signal are localized with an accuracy of 1 sample period. This means that if e.g. in the training set the target is 1 (a pulse is present), it might have been the previous or the next sample that actually is the pulse. Now consider what happens if the squared error is used in the case of a displaced pulse in the target output signal. Suppose the target output of sample p is 1, but actually the pulse is at position p-1. Next suppose that the network correctly detects the pulse at position p-1, i.e. its output for sample p-1 is 1 and for sample p it is 0. The squared error due to these two samples according to eq.6.5.1-1 will be 2, because according to the target output signal, both samples are classified incorrectly, see fig.6.16.

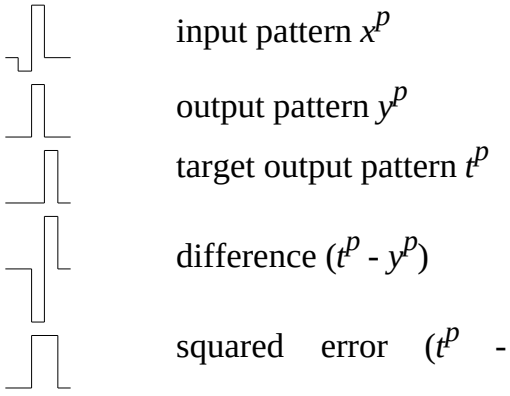


fig.6.16 Double counting of displacement errors: left from top to bottom: input signal, output signal and target output; right: difference between target and output, and squared error.

Thus the error due to a one sample period displacement of the pulse is counted double. Since the squared error due to e.g. simply missing the pulse, is only 1, the network will probably tend to suppress the pulse. The conclusion is that for this problem the least squares error measure yields a network that will fail to detect a lot of pulses. Instead the network should be optimised for the average risk.
In this problem three types of errors can be distinguished:

- the network detects a pulse, although there is no pulse
- the network fails to detect a pulse, but there is a pulse
- the network detects a pulse, but displaced over one period

The first error, a spurious pulse, occurs if the output of the network $y^p = 1$, and all target outputs within one sample period distance are zero:

$$(t^{p-1} = 0 \cap t^p = 0 \cap t^{p+1} = 0) \cap y^p = 1 \qquad (6.5.3\text{-}1)$$

The second error, a missed pulse, occurs if the target output $t^p = 1$, and all network outputs within one sample period distance are zero:

$$t^p = 1 \cap (y^{p-1} = 0 \cap y^p = 0 \cap y^{p+1} = 0) \qquad (6.5.3\text{-}2)$$

The third error, displacement over one period, occurs if the output of the network $y^p = 1$ and one of the neighbouring target outputs is also 1:

$$(t^{p-1} = 1 \cup t^{p+1} = 1) \cap y^p = 1 \qquad (6.5.3\text{-}3)$$

If the distance between the pulses is at least 2 periods, the unions and the intersections may be substituted by additions. The average risk becomes:

$$\begin{aligned} AVR = {} & \lambda_1 P(t^{p-1}+t^p+t^{p+1} = 0 \cap y^p = 0) + \\ & + \lambda_2 P(t^p = 0 \cap y^{p-1}+y^p+y^{p+1} = 0) + \\ & + \lambda_3 P(t^{p-1}+t^{p+1} = 1 \cap y^p = 1) \end{aligned} \qquad (6.5.3\text{-}4)$$

Where: $\lambda_1,\lambda_2,\lambda_3$= the respective costs of the three different types of errors

The probabilities one the error types can be estimated from the training set, by counting the number of errors and dividing this by the total number of samples in the training set. The following is then obtained:

$$AVR \approx \lambda_1 \frac{N_{t^{p-1}+t^p+t^{p+1}=0 \cap y^p=0}}{N} + \lambda_2 \frac{N_{t^p=0 \cap y^{p-1}+y^p+y^{p+1}=0}}{N} + \lambda_3 \frac{N_{t^{p-1}+t^{p+1}=1 \cap y^p=1}}{N} \quad (6.5.3\text{-}5)$$

Since the error backpropagation learning rule requires an error measure that is continuous and differentiable in y_p, eq.6.5.3-5 is not fit for direct application. In paragraph 6.5.1, it was shown that the continuous and differentiable squared error measure actually is an approximation of a sum of the probabilities on different types of errors, if it is assumed that the output of the network is always close to 1 or 0. A similar method, but in the reversed order, can be used here to transcribe eq.6.5.3-5 into a form that is continuous and differentiable. For the number of misclassified samples according to the three different types of errors in eq.6.5.3-5, the following approximations can be made:

$$N_{t^{p-1}+t^p+t^{p+1}=0 \cap y^p=0} \approx \sum_{p|t^{p-1}+t^p+t^{p+1}=0} (y^p)^2 \quad (6.5.3\text{-}6)$$

$$N_{t^p=0 \cap y^{p-1}+y^p+y^{p+1}=0} \approx \sum_{p|t^p=1} (1-y^{p-1}+y^p+y^{p+1})^2 \quad (6.5.3\text{-}7)$$

$$N_{t^{p-1}+t^{p+1}=1 \cap y^p=1} \approx \sum_{p|t^{p-1}+t^{p+1}=1} (y^p)^2 \quad (6.5.3\text{-}8)$$

Now an approximation for the average risk can be formulated, that is continuous and differentiable in y^p:

$$AVR \approx \lambda_1 \sum_{p|t^{p-1}+t^p+t^{p+1}=0} (y^p)^2 + \lambda_2 \sum_{p|t^p=1} (1-y^{p-1}+y^p+y^{p+1})^2 + \lambda_3 \sum_{p|t^{p-1}+t^{p+1}=1} (y^p)^2 \quad (6.5.3\text{-}9)$$

An experiment for pulse detection was set up with artificially generated training data to test the validity of the proposed solution. As a training set a signal with pulses corrupted with low frequency noise was used. In the target output signal, random shifts of 1 sample period were applied to the pulses. The training set contained 1000 samples. Two networks were trained with the same architecture: 3 layers, 7 inputs, 2 processing elements in the hidden layer and 1 in the output layer. The first network was trained

using the squared error measure, the second was trained using the average risk approximation of eq.6.5.3-9. The cost function was chosen to ignore the displacement errors and to weight the spuriously detected and missed pulses equally:

- $\lambda_1 = \lambda_2 = \frac{N}{2}$
- $\lambda_3 = 0$

Both networks were trained by presenting the whole training set 1000 times. The error measures of the networks as a function of the number of epochs are shown in fig.6.17.

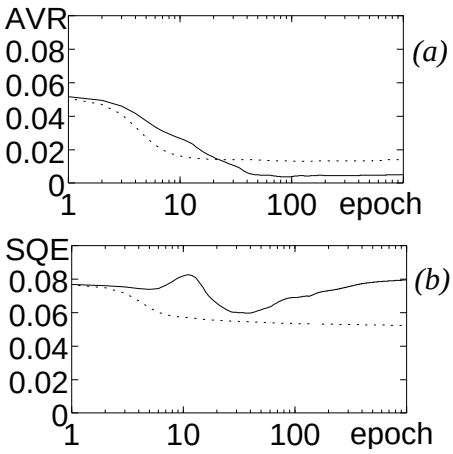


fig.6.17 AVR and SQE, during learning of two networks. Solid curve: network minimising AVR ; dotted curve: network minimising SQE. (a) AVR during learning of both networks; (b) SQE during learning of both networks.

In fig.6.17a, the average risk for both networks during the learning process is given. It is clear that the network trained to minimise the AVR (solid curve) results in a lower AVR. In fig.6.17b the squared error measure for both networks during the learning process is given. As may be expected, the network trained to minimise SQE (dotted curve) reaches a lower SQE. From these curves it becomes very clear that the squared error measure is not equal to the average risk for this application. If network is required that is optimal in the sense of a minimum AVR, it should not be trained with the standard backpropagation learning rule.

The output of the networks was evaluated with an independent evaluation set, with the same statistics as the training set. The network trained with the SQE error measure failed to detect many of the pulses. The network trained for minimum AVR made almost no errors. Fig.6.18 shows a small part of the output signals for both networks and the corresponding input and target signals.

Clearly the output of the AVR-trained network (fig.6.18d) shows much stronger peaks at the positions of the pulses than the SQE-trained network. Actually, the SQE trained network is presented contradictory information, because at one time an input vector should, according to the target output, be classified as a pulse, while at another time a similar output, with a shifted target output, should, according to the target output, not be classified as a pulse. The result is that the network tends to suppress the pulses with

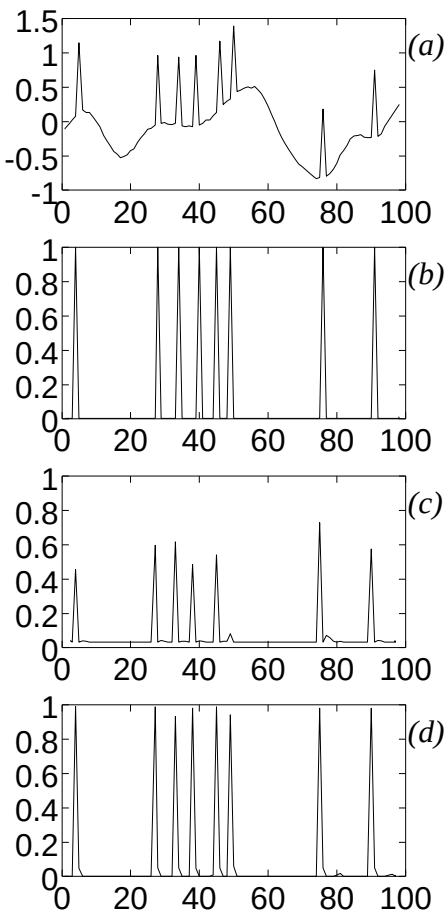


fig.6.18 Signals of the pulse detection network: (a) training signal; (b) target output; (c) output of SQE-trained network; (d) output signal of AVR trained network.

shifted target outputs. This is best illustrated by the response on the pulse at position 46, of which the target output is shifted one period to the left. The SQE-trained network suppresses this pulse almost completely, but the pulse is cleanly detected by the AVR-trained network.

6.6 Discussion

The AVR performance evaluation measure appears to be applicable for a wide range of image filters. But its use can even be extended further. There is no reason why it should not be used as a general performance measure for image processing subsystems. The cost function, required for the evaluation of an image processing subsystem, depends on the image processing system and its required output. For each application, the cost function should therefore be worked out separately. Image processing operations should be optimised for the average risk. In the previous section it has been shown that neural networks can be optimised for the AVR. Optimisation of an image filter for AVR is illustrated schematically in fig.6.19.

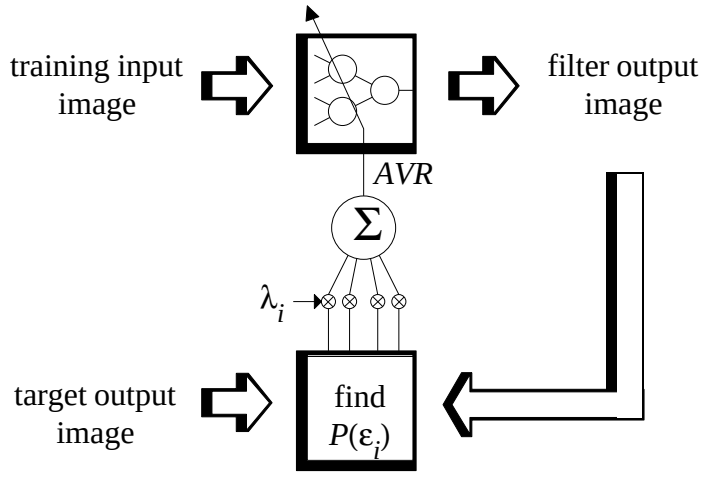


fig.6.19 Designing a neural network image filter using minimum AVR as an optimisation criterion.

7. Summary, conclusions and prospects

7.1 Image filtering with neural networks

To investigate the application of neural networks to image filtering, two fundamental problems had to be addressed:

- Designing a tool for image filtering with neural networks
- Developing an evaluation method for image filtering

A tool for error backpropagation neural networks for image filtering was developed and the average risk evaluation method. During research a third fundamental problem was addressed:

- Training neural networks for minimum average risk.

In this work the emphasis was on model based design of image filters with neural networks. With this model based design approach is not meant modelling the human visual system, but using scene models and models of imaging systems in order to design a certain desired filter operation.
The models used to design neural networks are non parametric models consisting of examples of the desired behaviour of the network. In the case of image filtering the examples are the input images and the corresponding target output images. Natural, hand edited and synthetic test and training images using 2D and 3D scene and imaging models are described in chapter 4.
Compared to parametric models, advantages of using non-parametric models are:

- they are relatively easy obtained

- they can be very accurate

Disadvantages of using non-parametric models are:

- accurate models often exist of a lot of data
- it is difficult to determine the quality of the model
- little knowledge is gained of the underlying processes

Parametric models on the other hand can generally be formulated compactly and provide insight in the underlying processes. But often more effort is required to obtain accurate parametric models.

Neural networks can be regarded as multi-parameter solutions, based on general function approximation by combining many simple basic functions. The learning rules are methods to obtain the parameters. The advantages and disadvantages of multi-parameter solutions relative to parametric solutions are similar to those of non-parametric relative to parametric.
Advantages of using multi-parameter solutions are:

- solutions are relatively easily obtained, using general methods to obtain the parameters
- no simplifications of the models are required

Disadvantages of multi-parameter solutions are:

- solutions maybe inefficient, often more parameters are used than is necessary
- little knowledge is gained about the solution itself
- the quality of the solution can often only be determined experimentally

The behaviour of neural networks is determined to great extent by the connection weights. It is often suggested that a spontaneous ordering of information takes place in neural networks. For the neural networks for edge detection that were used in chapter 5, this was the case only to some extent. No really clear "principle components" could be discovered in the networks. It is probable that they are mixed and distributed over the network.

Neural networks image filters are multi-parameter approaches to image filtering that can be trained using training images. The advantage is that they can easily be tuned for a specific types of scene if the training images are representative for this type of scene.

Neural networks may outperform other solutions if the other solutions are based on inaccurate or too much simplified models or the architecture of the solutions themselves restricts the accuracy. Thus there maybe three major reasons to use neural networks:

- If obtaining a solution is more important than gaining insight in the processes
- If obtaining an accurate model of the underlying processes is hard
- If designing an operator, based on the process models is difficult

Generally, neural networks do not necessarily give better solutions, but rather provide a quick and easy way to obtain near optimal solutions.

7.2 Performance evaluation with AVR

Not much literature exists on performance evaluation of image filters (and image processing (sub)systems in general). In some performance evaluation methods performance of an image filter is defined in general, separately from its application and the image operating system of which it is a part. In our opinion this is incorrect. The performance of an image processing (sub)system should only be defined in connection with its application and the total image processing system.
In this work a performance evaluation method for image filters based on the average risk is proposed. The average risk performance measure has a solid basis in statistical theory. The average risk is the summation of the probabilities or frequencies of errors, weighted with a cost function. The cost function is determined by the application. The probabilities of errors depend on the types of images that occur and on the image filter operation.
For performance evaluation of image filters a method is proposed to estimate the probabilities on errors from test images by comparing the output of an image filter for a test image with the corresponding target output image. The test image should be representative for the types of images the image filter operates on. An average risk performance evaluation method for edge detectors is worked out as an example. Several edge detectors, among which neural network edge detectors, are compared using this performance measure. The results show that the neural network edge detectors reach near optimal performance. When trained for specific types of edges and images, they outperform other edge detectors, not specifically designed for these images.

The basic approach of using average risk for performance evaluation is applicable in a wider sense. In principle average risk can be used as a performance measure for all types of image processing systems and subsystems. However cost functions and test data must be worked out for each case separately (and for each application).

7.3 Optimisation for AVR

One disadvantage of the used neural networks is that they are actually not optimised for average risk but yield a least squares solution. In some cases the minimum average risk and the least squares solution approximately, but often they don't. It is shown that for a quadratic cost function and a distance measure that is equal to the cost function the average risk reduces to the mean squared error. Also for a unity cost function in classification problems it is shown that average risk can be approximated by the mean squared error. If these cost functions are appropriate for the application, the least squares criterion can be used for optimisation of neural networks. If not the learning rules should be adapted to minimise the average risk. For the error backpropagation learning rule this is worked out.

7.4 Further research

A number of subjects, which should be investigated further, were only shortly addressed in this work. They include items concerned with fast and reliable learning in neural networks like efficiency of learning rules, problems with local minima and stopping criteria, quality of training sets, optimal network architectures and clever initialisation of connection weights. Initial work on quality of training sets was carried out by Siteur [Siteur 1991] (similar research was carried out at another location by [Kraaijveld 1990]).
Several architectures for image filters were only very briefly addressed. An interesting architecture for image filtering that was mentioned, is an architecture with symmetric weights that takes the whole image as input. The advantages of such an architecture are that the storage requirements are not excessive, while in contrary to the local neighbourhood based image filtering approach, it is able to use global image features. A disadvantage is the increased complexity. A first attempt to use a Boltzmann network with this kind of architecture was not very successful [Morselt 1991]. A first experiment using a Kohonen network for texture classification, using local neighbourhoods, appeared quite successful.

The research presented in this work also creates several possibilities and challenges for further and new research. Subjects especially of interest are:

- Applying AVR for design and evaluation of image filters and other image processing operations
- Determining cost functions for specific applications by analyzing the effect of errors in the output of an image processing operation on the output of the total image processing system
- Generation of test and training images using 3D scene and imaging models
- Training neural networks for minimum average risk
- Analysis of internal representations of neural networks using decomposition in basic components
- Using neural network image filters in image processing systems
- Investigating applicability of neural networks for other image processing operations.

References

[Abdou 1979] Abdou, I.E., Pratt, W.K., Quantitive design and evaluation of enhancement/thresholding edge detectors, *Proc. of the IEEE*, vol.67, no.5, pp.753-763, 1979

[Abu-Mostafa 1985] Abu-Mostafa, Y., St. Jacques, J., Information capacity of the Hopfield model, *IEEE Trans. Inf. Theory*, vol.7, pp.1-11, 1985

[Ahuja 1983] Ahuja, N., Schachter, *B.J., Pattern models*, John Wiley & Sons, New York, 1983

[Ballard 1982] Ballard, D.H., Brown, C.M., *Computer vision*, Prentice-Hall Inc., Eaglewood Cliffs, New Jersey, 1982

[Besuijen 1989] Besuijen, J., Heijden, F. van der, An edge detector performance measure incorporating structural errors, *Proc. of the Douzieme colloque Gretsi*, Juan-le-Pins, June 12-16, 1989

[Blicher 1984] Blicher, P., *Edge detection and geometric methods in computer vision*, dissertation, UMI, Michigan, 1987

[Bosman 1989] Bosman, D., *Display engineering*, Elsevier Science Publishers, 1989

[Bosman 1990] Bosman D., White, T.W., Simulation of the perception of computer generated images, in: *Digest International Symposium for Information Display*, Las Vegas, vol.XXI, pp.21-24, 1990

[Canny 1986] Canny, J., A computational approach to edge detection, *IEEE Trans. Pattern Recogn. and Mach. Int.*, PAMI-8, pp.679-698, Nov. 1986

[Cook 1984] Cook, R.L., Porter, T., Carpenter, L., Distributed raytracing, *Computer graphics (SIGGRAM 84 Proceedings)*, vol.18, no.3, pp.137-145

[Cook 1986] Cook, R.L., Stochastic sampling in computer graphics, *ACM Trans. on Graphics*, vol.5, no.1, pp.51-72, 1986

[Cybenko 1988] Cybenko, G., Continuous valued neural networks with two hidden layers are sufficient, *Tech. report*, Department of Cumputer Science, Tufts University, March 1988

[DARPA 1988] *DARPA neural network study,* AFCEA Press, Fairfax Verginia, 1988

[Dasarathy 1991] Dasarathy, B.V., *Nearest neighbour (nn) norms: NN pattern classification techniques*, IEEE Computer Soc. Press, Las Alamitos, California, 1990

[Devijver 1982] Devijver, P.A., Kittler, J., *Pattern recognition: a statistical approach*, Prentice Hall, Eaglewood Cliffs, N.J., 1982

[Devijver 1973] Devijver, P.A., Relationships between risks and the least-mean-square design criterion in pattern recognition, *Proc. of the 1st Int. Conference on Pattern Recognition*, Washington, D.C., pp.139-148, 1973

[Feng 1991] Feng, T., Houkes, Z., Korsten, M.J., Spreeuwers, L.J., A study on backpropagation networks for parameter estimation from grey-scale images, *Proceedings of the 1991 IEEE International Joint Conference on Neural Networks IJCNN'91*, Singapore, pp.331-336, 1991

[Feng 1992] Feng, T., Houkes, Z., Korsten, M.J., Spreeuwers, L.J., Internal measuring models in trained neural networks for parameter estimation from images, *Proceedings of the IEE 4th International Conference on Image Processing and its Applications*, Maastricht, Netherlands, pp.230-233, 1992

[Foley 1989] Foley, J.D., Dam, A. van, Feiner, S.K., Hughes, J.F., *Computer graphics, principles and practice*, Adison-Wesley, Massachusets, 1989

[Finkelstein 1990] Finkelstein, L., Measurement and instrumentation as a discipline in the framework of information technology, *VDI Berichte*, nr.856, pp.275-283,1990

[Fram 1975] Fram, J.R., Deutsch, E.S., On the Quantitative evaluation of edge detection schemes and their comparison with human performance, *IEEE Trans. on Computers*, vol.C-24, pp.616-628, June 1975

[Fukushima 1988] Fukushima, K., A hierarchical neural network capable of visual pattern recognition, *Neural Networks*, vol.1, pp.119-130, 1988

[Graaf 1990] Graaf, A.J. de, Korsten, M.J., Houkes, Z., Estimation of position and orientation of objects from stereo images, *Proceedings of the 12th DAGM Symposium on "Mustererkennung ’90", Informatik-Fachberichte 254*, Oberkochen-Aalen, Germany, 1990

[Haralick 1982] Haralick, R.M., Zero-crossing of second directional derivative egde operator, *Proc. of SPIE Symposium on Robot Vision*, Arlington, Verginia, May 1982

[Haralick 1984] Haralick, R.M., Digital step edges from zero crossing of second directional derivatives, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no.1, pp.58-68, 1984

[Heijden 1989] Heijden, F. van der, Quality of edge detectors, *Proc. CSN89*, Sion, Utrecht, Netherlands, Nov. 1989

[Heijden 1992] Heijden, F. van der, *A statistical approach to edge and line detection in digital images*, dissertation, Febodruk, Enschede, Netherlands, 1992

[Hildreth 1983] Hildreth, E.C., The detection of intensity changes by computer and biological vision systems, *CVGIP*, vol.22, pp.1-27, Jan. 1983

[Hopfield 1982] Hopfield, J.J., Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl., Acad. Sci. USA*, vol.79, pp.2554-2558, April 1982

[Houkes 1989] Houkes, Z., Korsten, M.J., A comparison of two nonlinear motion estimation methods, Proceedings of the 11th DAGM Symposium "Mustererkennung ’89", Hamburg, Germany, pp.302-309, 1989

[Houkes 1990] Houkes, Z., Korsten, M.J.,Considering shape from shading as an estimation problem, *Proceedings of the SPIE/SPSE Symposium on Electronic Imaging Science & Technology, Image Processing Algorithms and Techniques* Santa Clara, CA, USA, vol.1244, pp.56-67, 1990

[Houkes 1990] Houkes, Z., Korsten, M.J., Heijden, F. van der, Pattern recognition and parameter estimation: a tool for automated measurement, *In: VDI Berichte 856, Knowledge Based Measurement - Application, Research and Education, no. 1990*, pp.285-294, 1990

[Hueckel 1973] Hueckel, M.H., A local visual operator which recognizes edges and lines, *JACM*, vol.20, no.1, pp.634-647, Oct. 1973

[Hugen 1990] Hugen, F.M., Korsten, M.J., Houkes, Z., A reconfigurable architecture for real-time 3D-parameter estimation from image sequences, Proceedings of the SPIE/SPSE Symposium on Visual Communications and Image Processing, Lausanne (Switserland), pp.304-315, 1990

[Hugen 1992] Hugen, F.M., Houkes, Z., Systolic arrays for recursive linear least squares parameter estimation, *Proceedings of the IEEE Benelux ProRisk Workshop on Circuits, Systems and Signal Processing*, Houthalen, Belgium, pp.283-289, 1992

[Kajiya 1986] Kajiya, J.T., The rendering equation, Computer graphics *(SIGGRAM 86 Proceedings)*, vol.20, no.4, pp.143-149

[Karna 1989] Karna, K.N., Breen, D.M., An artificial neural network tutorial: part 1 - basics, *Neural Networks*, vol.1, no.1, pp.4-23

[Keeler 1986] Keeler, J.D., Basins of attraction of neural network models, in: *Neural networks for computing, AIP Conference 151* (J.S. Denker ed.), 1986

[Kohonen 1984] Kohonen, T., *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984

[Kohonen 1988] Kohonen, T., An introduction to neural computing, Neural Networks, vol.1. no.2, pp.3-16, 1988

[Kolmogorov 1957] Kolmogorov, A.N., On the representation of continuous functions of many variables by superposition of cuntinuous functions of one variable and addition, *Dokl. Acad. Nauk USSR*, vol.114, pp.953-956, 1957

[Korsten 1989] Korsten, M.J., *Three-dimensional body parameter estimation from digital images*, dissertation, Febodruk Enschede, Netherlands, 1989

[Korsten 1989] Korsten, M.J., Houkes, Z., Parametric descriptions and estimation, a synergetic approach to resolving shape from shading and motion problems, *Proceedings of the IEE 3rd International Conference on Image Processing and Applications*, Warwick, UK, pp.5-9, 1989

[Korsten 1990] Korsten, M.J., Houkes, Z., The estimation of geometry and motion of a surface from image sequences by means of linearisation of a parametric model, *Computer Vision, Graphics and Image Processing*, vol.50, pp.1-28, 1990

[Kraaijveld] Kraaijveld, M.A., Duin, R.P.W., On backpropagation of edited data sets, *Proc. of the Int. Neural Networks Conference*, Paris, July 9-13, pp.741-744, 1990

[Kuffler 1976] Kuffler, S.W., Nichols, J.G., *From neuron to brain*, Sunderland Sinauer ass. inc., 1976

[Lippmann 1987] Lippmann, R.P., An introduction to computing with neural networks, *IEEE ASSP Magasine*, April 1987, pp.4-22, 1987

[Lorentz 1976] Lorentz, G.G., The 13th problem of Hilbert, in: *Mathematical developments arising from Hilbert problems*, (F.E. Browder, ed.), Provedence, R.I.: American Mathematical Society, 1976

[Lowe 1991] Lowe, L., Fitting parameterized three-dimensional models to images, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.13, no.5, pp.441-450, 1991

[Marr 1980] Marr, D., Hildreth, E.C., Theory of edge detection, *Proc. Royal Soc. London B*, vol.207, pp.187-217, 1980

[McCulloch 1943] McCulluch, W.S., Pitts, W., A logical calculus of the ideas imminent in nervous activity, *Bulletin of Mathematical Biophysics*, vol.5, pp.115-133, 1943

[McEliece 1987] McEliece, R. et al., The capacity of the Hopfield associative memory, *IEEE Trans. Inf. Theory*, vol.1, pp.33-45, 1987

[Minsky 1969] Minsky M., Papert, S., *Perceptrons: an introduction to computational geometry*, MIT Press, 1969

[Morselt 1991] Morselt, F., Edge detection with a Boltzmann machine?, *Master's Thesis*, Report 91M033, Department of Electrical Engineering, University of Twente, Netherlands, 1991

[Peli 1982] Peli, T., Malah, D., A study of edge detection algorithms, *CVGIP*, vol.20, pp.1-21, 1982

[Potmesil 1982] Potmesil, M., Chakravarty, I., Synthetic image generation with a lens and aperture camera model, *ACM Trans. on Graphics*, vol.1, no.2, pp.85-108, 1982

[Pratt 1978] Pratt, W.K., *Digital image processing*, John Wiley & Sons, New York, 1978

[Romson 1989] Romson, E.P., Duin, R.P.W., Model based recognition of 3D objects from single 2D images, Proc. of Intelligent Autonomous Systems 2, Amsterdam, Dec. 11-14, pp.853-863,1989

[Rooijen 1992] Rooijen, J.A. van Generation of realistic images and associated edge reference maps using 3 dimensional scene and imaging models, *Master's Thesis*, Report 92M081, Department of Electrical Engineering, University of Twente, Netherlands, 1992

[Rosenblatt 1959] Rosenblatt, F., *Principles of neurodynamics*, Spartan Books, New York, 1959

[Rosenblatt 1962] Rosenblatt, F., *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*, Spartan Books, New York, 1962

[Ruck 1990] Ruck, D.W., Rogers, S.K., Kabrinsky, M., Oxley, M.G., Suter, B.W., The multilayer perceptron as an approximation to a Bayes optimal discriminatino funciton, *IEEE Trans. Neural Networks*, vol.1, no.4, pp.296-298, 1990

[Rumelhart 1986] Rumelhart, D.E., McClelland, J.L., *Parallel distributed processing: explorations of the microstructure of cognition*, MIT Press, 1986

[Rumelhart 1986] Rumelhart, D.E., Hinton, G.E., Williams, R.J., Learning internal representations by error propagation, in: *Parallel distributed processing: explorations of the microstructure of cognition, vol.1: Foundations* (D.E. Rumelhart and J.L. McClelland eds.), MIT Press, 1986

[Schrap 1990] Schrap, P., Parameter estimation of 3-D bodies using detected edges in digital images, *Master's Thesis*, Report 90M046, Department of Electrical Engineering, University of Twente, Netherlands, 1990

[Simpson 1987] Simpson, P.K., A survey of artificial neural systems, *Unisys*, San Diego, 1987

[Siteur 1991] Siteur, W., Neural networks for classification problems, preprocessing of training sets, *Master's Thesis*, Report 91M007, Department of Electrical Engineering, University of Twente, Netherlands, 1991

[Spreeuwers 1991] Spreeuwers, L.J., A neural network edge detector, *Proc. SPIE Conf. on Nonlinear Image Processing II*, SPIE vol.1451, pp.205-215, 1991

[Spreeuwers 1992] Spreeuwers, L.J., Heijden, F. van der, An edge detector evaluation method based on average risk, in: *Robust computer vision* (Forstner and Ruwiedel eds.), Wichmann, Karlsruhe, 1992

[Spreeuwers 1992] Spreeuwers, L.J., Heijden, F. van der, Evaluation of edge detectors using average risk, *Proc. of the 11th APR International Conference on Pattern Recognition*, The Hague, 1992

[Spreeuwers 1992] Spreeuwers, L.J., Backpropagation with matched error measures, to be published

[Wan 1990] Wan, E.A., Neural network classification: a Bayesian interpretation, *IEEE Trans. Neural Networks*, vol.1, no.4, pp.303-305, 1990

Appendix A

Derivation of the error backpropagation learning rule

A.1 Processing elements and network architecture

A processing element in an error backpropagation network calculates a weighted summation of its inputs. The inputs can be either inputs of the network or outputs of other processing elements. Since the error backpropagation network is a feed forward network, these outputs can only be from processing elements (PEs) in lower layers. In the derivation of the learning rule, it is assumed that only connections exist from a layer to the next layer (no connections jumping over layers). The weights are assigned to the connections from outputs of PEs to PEs in the next layer. The output of a PE is a nonlinear function, the socalled transfer function, of this weighted sum. For an input pattern x^p of the network the output of PE m in layer k becomes:

$$o_k^{pm} = f(net_k^{pm}) = f(\sum_n w_k^{mn} o_{k-1}^{pm}) \qquad \text{(A.1-1)}$$

Where: o_k^{pm} = the output of PE number m in layer k for an input pattern x^p

$f()$ = the transfer function

net_k^{pm} = the weighted sum of the inputs of the PE for input pattern x^p

w_k^{mn} = the weight of the connection between PE n in layer k-1 and PE m in layer k

the summation is over all PEs n of layer k-1

In fig.A.1 the architecture and the indices for PEs and signals, as used in the formulas, of an error backpropagation network are shown.

Generally the transfer function of processing elements in an error backpropagation network are sigmoid transfer functions:

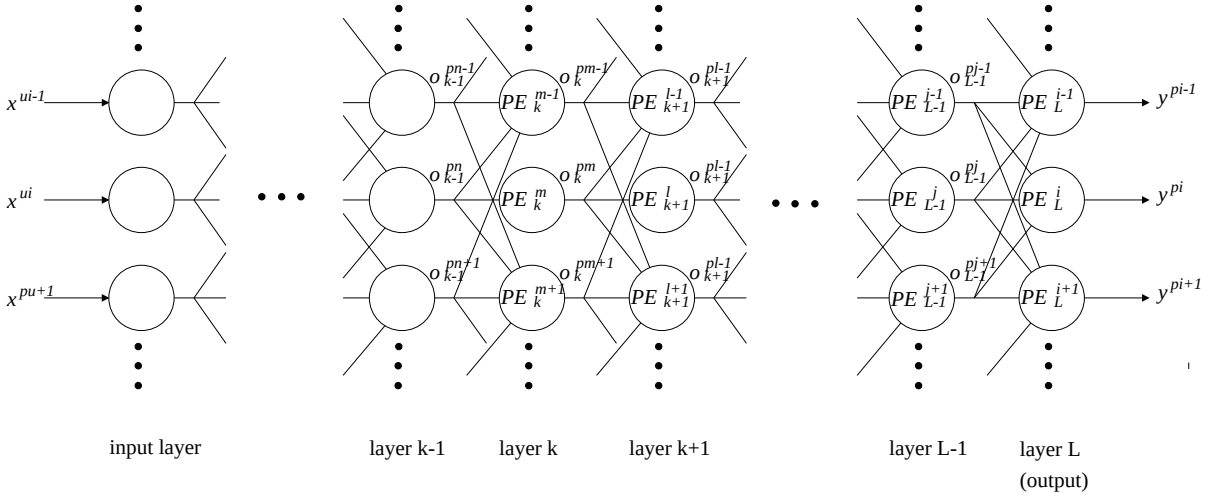


fig.A.1 Error backpropagation network. The input layer only distributes the inputs to the next layer. The layers k-1, k, k+1 and L-1 are hidden layers, and layer L is the output layer.

$$f(x) = \frac{1}{1+e^{-x+\theta}} \qquad \text{(A.1-2)}$$

This function acts as a soft threshold with the centre of the slope at θ. Often θ is implemented by adding an extra connection with weight −θ to a processing element with a fixed output 1.

A.2 Optimisation criterion and method

The error backpropagation learning rule attempts to minimise the sum of the squared errors of all patterns. The system error is defined as:

$$E = \frac{1}{2}\sum_p E^p = \frac{1}{2}\sum_p \sum_i (t^{pi} - y^{pi})^2 \qquad \text{(A.2-1)}$$

Where: E^p = the square error of the output for input pattern x^p
t^{pi} = the target of output i for a training input pattern x^p
y^{pi} = the actual network output i for training input pattern x^p
the summations are over all outputs i and training input patterns p

If the system error is zero, all training patterns are mapped on the correct target output pattern.

Since the behaviour a network with a certain architecture is defined by the connection weights, for a given training set E is a function only of the connection weights. The error backpropagation learning rule is an iterative learning rule that attempts to minimise E by adapting the weights. After each weight update, E should should become smaller.
Thus:

$$E(..,w_k^{mn}+\Delta w_k^{mn},..) < E(..,w_k^{mn},..) \qquad \text{(A.2-2)}$$

Where: w_k^{mn} = the weight of the connection between PE n in layer k-1 and PE m in layer k
Δw_k^{mn} = the update of weightw_k^{mn}

A first order Taylor series expansion of eq.A.2-2 yields:

$$E(...,w_k^{mn}+\Delta w_k^{mn},...) = E(...,w_k^{mn},...) + \ldots + \Delta w_k^{mn}\frac{\partial E(...,w_k^{mn},...)}{\partial w_k^{mn}} + \ldots \text{(A.2-3)}$$

If all weights are adjusted, the change in the system error E becomes:

$$\Delta E = \sum_k \sum_m \sum_n \Delta w_k^{mn} \frac{\partial E}{\partial w_k^{mn}} \qquad \text{(A.2-4)}$$

Where: the summations are over all layers k, PEs m of layer k and connections from PEs
n in layer k-1 to PE m

The change in the system error should be negative to make E smaller. This is guaranteed if each weight update is chosen according to:

$$\Delta w_k^{mn} = -\eta \frac{\partial E}{\partial w_k^{mn}} \qquad \text{(A.2-5)}$$

Where: η = a positive constant that controls the rate of change of the weights (learning rate)

The disadvantage of this gradient optimisation method is that it will get stuck in local minima if they exist. The advantage is that the method is simple.

In the following it is assumed that for each input pattern x^p of the training set the network output and also the weighted sum and output of each of the processing elements is calculated and known.
First rewrite the weight adjustment of eq.A.2-5 as a sum of adjustments due to the individual errors for all patterns in the trainingset:

$$\Delta w_k^{nm} = -\eta \frac{\partial \left\{ \sum_p E^p \right\}}{\partial w_k^{nm}} = \sum_p \left\{ -\eta \frac{\partial E^p}{\partial w_k^{mn}} \right\} \qquad \text{(A.2-6)}$$

Using the chain rule one can write:

$$\frac{\partial E^p}{\partial w_k^{mn}} = \frac{\partial E^p}{\partial net_k^{pm}} \frac{\partial net_k^{pm}}{\partial w_k^{mn}} = -\delta_k^{pm} \frac{\partial net_k^{pm}}{\partial w_k^{mn}} \qquad \text{(A.2-7)}$$

Where: $\delta_k^{pm} = -\frac{\partial E^p}{\partial net_k^{pm}}$, a kind of local error measure for PE *m* of layer *k* due to pattern x^p

A.3 Weight updates for the output layer

For the weights in the output layer (layer *L*) the weight updates can be calculated directly. Again using the chain rule results in:

$$\delta_L^{pi} = -\frac{\partial E^p}{\partial net_L^{pi}} = -\frac{\partial E^p}{\partial y^{pi}} \frac{\partial y^{pi}}{\partial net_L^{pi}} \qquad \text{(A.3-1)}$$

Where: net_L^{pi} = the weighted sum of the inputs of PE *i* in the output layer for pattern x^p
y^{pi} = output *i* of the ouput layer (and network) for pattern x^p

Substituting E^p gives:

$$\frac{\partial E^p}{\partial net_L^{pi}} = \frac{\partial (\frac{1}{2} \sum_i (t^{pi} - y^{pi})}{\partial y^{pi}} = -(t^{pi} - y^{pi}) \qquad \text{(A.3-2)}$$

Substituting eq.A.1-1 for y^{pi} gives:

$$\frac{\partial y^{pi}}{\partial net_L^{pi}} = \frac{\partial f(net_L^{pi})}{\partial net_L^{pi}} = f'(net_L^{pi}) \qquad \text{(A.3-3)}$$

Where: $f'(net_L^{pi})$ = the derivation of the transfer function *f* in net_L^{pi}

Now δ_L^{pi} can be expressed in known quantities:

$$\delta_L^{pi} = (t^{pi} - y^{pi}) f'(net_L^{pi}) \qquad \text{(A.3-4)}$$

The second term of the right hand part of eq.A.2-7 can be calculated as follows:

$$\frac{\partial net_L^{pi}}{\partial w_L^{ij}} = \frac{\partial \left\{ \sum_j w_L^{ij} o_{L-1}^{pj} \right\}}{\partial w_L^{ij}} = o_{L-1}^{pj} \qquad \text{(A.3-5)}$$

Thus for the updates of the weights of the connections to the PEs of the ouput layer, the following expression is obtained (substitute A.3-4, A.3-5 and A.2-7 in A.2-6):

$$\Delta w_L^{ij} = \sum_p \eta \delta_L^{pi} o_{L-1}^{pj} = \sum_p \eta (t^{pi} - y^{pi}) f'(net_L^{pi}) o_{L-1}^{pj} \qquad \text{(A.3-6)}$$

A.4 Weight updates for the hidden layers

The calculation of the updates for the weights of the connections to the other layers in the network, is somewhat more laborious. We start by calculating δ_k^{pi}.
Rumelhart et al. proceed by writing:

$$\delta_k^{pm} = -\frac{\partial E^p}{\partial net_k^{pm}} = -\frac{\partial E^p}{\partial o_k^{pn}} \frac{\partial o_k^{pn}}{\partial net_k^{pm}} \qquad \text{(A.4-1)}$$

and calculate the first term of the right part in the following way:

$$-\frac{\partial E^p}{\partial o_k^{pm}} = -\sum_l \frac{\partial E^p}{\partial net_{k+1}^{pl}} \frac{\partial net_{k+1}^{pl}}{\partial o_k^{pm}} =$$

$$= \sum_h \delta_{k+1}^{pl} \frac{\partial \left\{ \sum_m w_{k+1}^{lm} o_k^{pm} \right\}}{\partial o_k^{pm}} = \sum_l \delta_{k+1}^{pl} w_{k+1}^{lm} \qquad \text{(A.4-2)}$$

The first step in eq.A.4-2 is, although correct, not so obvious. Therefore we propose a more thorough derivation:

$$\delta_k^{pm} = -\frac{\partial E^p}{\partial net_k^{pm}} = -\frac{\partial \left\{ \frac{1}{2} \sum_i (t^{pi} - y^{pi})^2 \right\}}{\partial net_k^{pm}} = \sum_i (t^{pi} - o^{pi}) \frac{\partial y^{pi}}{\partial net_k^{pm}} =$$

$$= \sum_i (t^{pi} - y^{pi}) \frac{\partial f(net_L^{pi})}{\partial net_k^{pm}} = \sum_i (t^{pi} - y^{pi}) f'(net_L^{pi}) \frac{\partial net_L^{pi}}{\partial net_k^{pm}} =$$

$$= \sum_i (t^{pi} - y^{pi}) f'(net_L^{pi}) \frac{\partial \left\{ \sum_j w_L^{ij} o_{L-1}^{pj} \right\}}{\partial net_k^{pm}} = \sum_i (t^{pi} - y^{pi}) f'(net_L^{pi}) \sum_j w_L^{pi} \frac{\partial o_{L-1}^{pj}}{\partial net_k^{pm}} =$$

$$= \sum_i \sum_j (t^{pi} - y^{pi}) f'(net_L^{pi}) \, w_L^{pi} \frac{\partial o_{L-1}^{pj}}{\partial net_k^{pm}} = \ldots$$

The process of substituting the output functions continues until layer *l*, the layer next to layer *k* is reached. We then have the expression:

$$\delta_k^{pm} = \sum_i \sum_j \ldots \sum_l (t^{pi} - y^{pi}) f'(net_L^{pi}) \, w_L^{pi} \ldots f'(net_{k+1}^{pl}) \frac{\partial net_{k+1}^{pl}}{\partial net_k^{pm}} =$$

$$= \sum_i \sum_j \ldots \sum_l \left(t^{pi} - y^{pi} \right) f'(net_L^{pi}) \, w_L^{pi} \ldots f'(net_{k+1}^{pl}) \frac{\partial \left\{ \sum_m w_{k+1}^{lm} o_k^{pm} \right\}}{\partial net_k^{pm}} =$$

$$= \sum_i \sum_j \ldots \sum_l (t^{pi} - y^{pi}) f'(net_L^{pi}) \, w_L^{pi} \ldots f'(net_{k+1}^{pl}) \frac{\partial \left\{ \sum_m w_{k+1}^{lm} f(net_k^{pm}) \right\}}{\partial net_k^{pm}} =$$

$$= \sum_i \sum_j \ldots \sum_l (t^{pi} - y^{pi}) f'(net_L^{pi}) \, w_L^{pi} \ldots f'(net_{k+1}^{pl}) w_{k+1}^{lm} f'(net_k^{pm}) \quad \text{(A.4-3)}$$

The last equation enables us to calculate the δ's of the PEs in the hidden layers from known quantities. There is however a more convenient way to calculate the δ's, by using the repetitions of terms in eq.A.4-3. By rearanging eq.A.4-3 this will become clear:

$$\delta_k^{pm} = \sum_l \left[\sum_i \sum_j \ldots (t^{pi} - y^{pi}) f'(net_L^{pi}) \, w_L^{pi} \ldots f'(net_{k+1}^{pl}) \right] w_{k+1}^{lm} f'(net_k^{pm}) \quad \text{(A.4-4)}$$

The term between the square brackets is actually δ_{k+1}^{pl}. Hence δ_k^{pm} can be expressed as a sum of the δ's of the PEs in the next layer:

$$\delta_k^{pm} = \sum_l \delta_{k+1}^{l} w_{k+1}^{lm} f'(net_k^{pm}) \qquad \text{(A.4-5)}$$

If eq.A.4-5 is compared to eq.A.1-1, it becomes clear why this learning rule is called error backpropagation: the δ's are propagated backward through the network in a similar way as the input patterns *x* are propagated forward: by a weighted summation of the inputs of a processing element.

The second term of the right hand part of eq.A.2-7 becomes for a PE in a hidden layer:

$$\frac{\partial net_k^{pm}}{\partial w_k^{mn}} = \frac{\partial\left\{\sum_n w_k^{mn} o_{k-1}^{pn}\right\}}{\partial w_k^{mn}} = o_{k-1}^{pn} \qquad \text{(A.4-6)}$$

The expression for the updates of weights in a hidden layer becomes:

$$\Delta w_k^{mn} = \sum_p \eta \delta_k^{pm} o_{k-1}^{n} \qquad \text{(A.4-7)}$$

With δ_k^{pm} given by eq.A.4-5.

A.5 Sequential training

The weight updates for the error backpropagation learning rule, derived in the previous paragraph, require the whole training set to be processed for one update of all connection weights in the network (parallel training). This is however not the way the error backpropagation rule is generally used. In most applications the weights are updated after the presentation of *each* training pattern (sequential training). The weight update for a pattern x^p becomes:

$$\Delta w_k^{pmn} = \eta \delta_k^{pm} o_{k-1}^{n} \qquad \text{(A.5-1)}$$

If the individual weight updates are small relative to the weights themselves, the effect will be approximately the same as for the cumulative update of eq.A.4-7. If they are not, an oscillation effect may occur if a number of successive training patterns result in more or less opposite weight updates. Experiments show that in the general case when the weight updates are small (η sufficiently small), sequential and cumulative weight updates yield comparable solutions.

Appendix B

BPLIB software package

B.1 Short description

The bplib software package was specially written for image filtering with error back-propagation neural networks. The software package, that is written in C, consists of three main parts:

- bplib - function library for error backpropagation
- bpima - program for image filtering with error backpropagation networks
- mknet - program for creation of feedforward networks

The bplib function library contains function for allocating data structures for feed forward networks, reading and writing these structures to files and performing the forward and error backpropagation calculations. The program bpima scans an image either from top left to bottom right or randomly to obtain local neighbourhoods of pixels. The grey levels of these sub-images are the inputs for the neural network. From the outputs of the neural network an image is built up in the same way.

B.2 Manual pages

The UNIX style manual pages of the bplib function library, the program bpima for image filtering with error backpropagation neural networks, and the program mknet for creation of networks, are given on the subsequent pages.

BPLIB(3) C LIBRARY FUNCTIONS BPLIB(3)

NAME

bplib - standard backpropagation neural network function library

SYNOPSIS

```
#include.h

NET *readnet(f)
FILE *f;

void writenet(f,netp)
FILE *f;
NET *netp;

void recall(netp,in,out)
NET *netp;
double *in,*out;

void backprop(netp,err)
NET *netp;
double *err;

void updateweights(netp)
NET *netp;

void cumulate(netp)
NET *netp;

void adapt(netp,err)
NET *netp;
double *err;

void learn(netp,in,outd,out)
NET *netp;
double *in,*outd,*out;
```

DESCRIPTION

readnet() reads a neural network according to ndf(5) (network description file) format from file f. If an error is encountered in the file format an error message is generated and an exit(1) call is done. The memory allocation for the network structure is done in the function readnet.

writenet() writes a neural network according to ndf(5) format to file f.

recall() calculates the output of the network using the network pointed to by netp. The input vector is an array of n doubles, with n the number of inputs of the network. The outputvector is an array of m doubles, with m the number of

outputs of the network. Recall calculates the weighted sum and output of each pe and puts them in the corresponding fields of the pe structure.

backprop() propagates the error in *err* back through the network according to the standard backpropagation learning rule. *err* is of the same dimension as the output vector and should normally contain the difference between the desired (or target) output and the actual network output:

err = outd - out

Where *out* is the actual output of the network (produced by recall()) and *outd* is the desired or target output.
The error is propagated back through the network by calculating the socalled delta's for all pe's. These are stored in the delta fields of the pe's.

cumulate() calculate the weight changes from the delta's and cumulate them in the deltaw fields of the connections. The momentum term is not used in this function.

*deltaw += learnrate*delta*input*

The *learnrate* and *delta* are taken from the destination pe of the connection;
The *input* is the output of the source pe of the connection.

updateweights() calculates the new weights using the weight updates in the deltaw fields of the connections:

weight += deltaw

adapt() adapts the weights of the neural network pointed to by netp according to the standard backpropagation learning rule, using the vector *err*. adapt() does both backpropagation and weight updates, for sequential learning. It also uses the momentum term to incorporate the previous weight update in the new update.
The function first calls the function backprop() and then updates the weights, according to:

*deltaw = learnrate*delta*input + momentum*deltaw*
weight += deltaw

learn() does first a recall() and then an adapt(). The input pattern is stored in *in*, *outd* is the target output and *out* is the output of the network. The error for adapt() is calculated internal in the function according to

err = outd - out.

NETWORK DATA STRUCTURE

The network datastructure contains elements of 4 different types: connections, processing elements (pe's), layers:

```
typedef struct _con     CON;
typedef struct _pe      PE;
typedef struct _layer   LAYER;
typedef struct _net     NET;
```

The top level of the structure is NET. A network structure consists of a pointer to its layers, pe's and connections:

```
struct _net
{
     LAYER *layerp;
     int nrlayers;

     PE *pep;
     int nrpes;

     CON *conp;
     int nrcons;
};
```

layerp[*0*] is to the first layer, *layerp*[*1*] the second etc.
layer[*0*] is the input buffer layer. The input of the network is copied to the outputs of these pe's, at the beginning of a recall.
pep and *conp* are arrays of the processing elements and the connections of the network.
pep[*0*] is the *bias* pe, of which the output is always 1.

A layer consists of a pointers to its processing elements and its connections:

```
struct _layer
{
     PE *pep;
     int nrpes;

     CON *conp;
     int nrcons;
};
```

The next level of the network are the pe's of the layer. The pe structure contains a pointer to the connections that lead to the pe, and a number of data fields:

```
struct _pe
{
     CON *conp;
     int nrcons;
```

```
        double sum;
        double output;
        double delta;
        double learnrate;
        double momentum;
        int transfer;
};
```

The data fields of the pe structure have the following meaning:
sum - the weighted sum of its inputs
output - the output of the pe
delta - the delta of the pe calculated during the error backpropagation
learnrate - the pe's learnrate used in updating the weights of its connections
momentum - the pe's momentum used in updating the weights of its connections
transfer - the pe's transfer function to calculate its output from the sum

The available transfer functions are:

TRF_IDENTITY
TRF_PARABOLA
TRF_SIGMOID
TRF_EXPONENT
TRF_LOG

The call to the transfer function is done via an array of pointers to functions. There are 8 entries in this array. Entries 0-4 are used by the above given transfer functions, so 5..7 are available to the programmer. If you want to use a costum transfer function, you must initialize the appropiate entries in the function arrays transf and dtransf (for the differentiated transfer function) and write the functions. For example:

```
double my_transfer(x)
double x;
{
     return calculation(x);
}

double my_dtransfer(x,y)
double x,y;
{
     return calculation_d(x,y);
}

transf[5] = my_transfer;
dtransd[5] = my_dtransfer;
```

To the transfer function the weighted sum of the pe is passed in *x*.
To the differentiated transfer functions both the sum and the output of the pe are passed in *x* and *y*, because in some cases this simplifies the calculations (e.g. for the sigmoid).

The pe's are connected through the connections. A connection structure consists of a pointer to its source pe and three data fields:

```
struct _con
{
     double weight;
     double deltaw;
     int variable;
     PE *srcpep;
};
```

In the *weight* field the weight of the connection, that is used in calculating the weighted sum, is stored.
The *deltaw* field is used to store the calculated weight update.
If the flag *variable* is set, the weight will be updated in updateweights(), adapt() or learn(). If not the weight will not be updated, although deltaw will be calculated. The macro's VAR_WEIGHT and FIX_WEIGHT are available to test and set the flag.

MACRO'S

A number of macro's for initialization of the structures and access to the structures and their fields are provided. In the description below, the parameters of the macro's have the following meanings:

netp - pointer to a network structure
layerp - pointer to a layer
pep - pointer to a processing element
srcpep - pointer to a processing element
conp - pointer to a connection

The layers, processing elements and connections are stored in arrays. Thus layerp[3] is layer no. 3 of a network; pep[2] is processing element 2 of a network or a layer etc.

layer - number of layer
pe - number of pe relative to the given object (network or layer)
con - connection number relative to network, layer or pe

The rest of the arguments are numbers to put in the data

fields.

```
/*
 *      macro's for network creation and initialization
 */
```

CREATE_NET - create a network structure; return a pointer to the network structure
CREATE_LAYERS(n) - create n layers and return a pointer to the array of layers
CREATE_PES(n) - create n pe's and return a pointer to the array of pe's
CREATE_CONS(n) - create n connections and return a pointer to the array of connections

```
INIT_NET(netp,nrlayers,layerp,nrpes,pep,nrcons,conp)
INIT_LAYER(layerp,nrpes,pep,nrcons,conp)
INIT_PE(pep,nrcons,conp,sum,output,delta,learnrate,momen
tum,transfer)
INIT_CON(conp,srcpep,weight,deltaw,variable)
```

```
/*
 *      macro's to access networks
 */
```

CON_WEIGHT(conp) - weight of connection pointed to by conp
CON_DELTAW(conp) - deltaw of connection pointed to by conp
CON_SRCPEP(conp) - source pe of connection pointed to by conp
CON_VARIABLE(conp) - variable flag of connection

PE_NRCONS(pep) - number of connections to pe pointed to by pep
PE_CONP(pep) - pointer to array of connections of pe
PE_SUM(pep) - weighted sum of pe
PE_OUTPUT(pep) - output of pe
PE_DELTA(pep) - delta calculated by backprop() of pe
PE_LEARNRATE(pep) - pe's learnrate
PE_MOMENTUM(pep) - pe's momentum
PE_TRANSFER(pep) - pe's transfer function

LAYER_NRPES(layerp) - number of pe's of layer pointed to by layerp
LAYER_PEP(layerp) - array of pe's of layer
LAYER_NRCONS(layerp) - number of connections to layer
LAYER_CONP(layerp) - array of connections to layer

NET_NRLAYERS(netp) - number of layers of network pointed to by netp
NET_LAYERP(netp) array of layers of network
NET_NRPES(netp) - number of pe's of network
NET_PEP(netp) - array of pe's of network
NET_NRCONS(netp) - number of connections in network

NET_CONP(netp) - array of connections in network

CONP(netp,layer,pe,con) - pointer to connection *con* to pe *pe* in layer *layer* of network pointed to by *netp*
PEP(netp,layer,pe) - pointer to pe *pe* of layer *layer*
LAYERP(netp,layer) - pointer to layer *layer*
BIASP(netp) - pointer to the bias pe
LAST_LAYERP(netp) - pointer to the output layer of the network

Especially the last 5 macro's will be usefull, because they provide easy access to the network.

EXAMPLES

To get a pointer to connection 2 of processing element 3 of layer 4 of a network, use:

```
conp = CONP(netp,4,3,2);
```

To set the weight of a connection pointed to by conp to 0.5, the following code can be used:

```
CON_WEIGHT(conp) = 0.5;
```

To get the output of pe 3 in layer 2, use:

```
output23 = PEP_OUTPUT(PEP(netp,2,3));
```

For other examples see the source of the related programs.

SEE ALSO

mknet(1), bpnet(1), prtnet(1), ndf(5)

DIAGNOSTICS

The ndf format version is checked (currently 3.10) in readnet(). Readnet() reads all previous versions; however writenet() always writes the latest version.

BUGS

There may be some, however none have been found yet. Please report!

AUTHOR

L.J. Spreeuwers

BPIMA(1) USER COMMANDS BPIMA(1)

NAME

bpima - back propagation neural network for pattern recognition in images

SYNOPSIS

bpima [-trbcdm] [-s *low high*] [-l *epoch*] [-p *layer pe*] [-a *accept*] *ndf-file* [*ima-in* [*ima-out/ref*]]

DESCRIPTION

bpima features neural network simulation for feed forward neural networks that use the standard backpropagation learning rule. Bpima is to be used for neural network image filtering. All images are 1 byte/pixel deep. Bpnet uses stdin and stdout. The used network architecture is in the *ndf-file* (network description file, see ndf(5)). A feedforward network can be generated using mknet(1).

For learning, an input image and a reference image must be specified. The input image can be aither read from stdin or from the file *ima-in*. The reference image is read from the file *ima-ref*. If no reference image is specified, the input image will also serve as reference image. Input measurement (training) vectors are formed from a local neighbourhood of a pixel. The size of the local neighbourhood is determined by the number of inputs of the neural network, and must be square. Target vectors are obtained by taking a local neighbourhood of the corresponding pixel from the reference image. The size of the local neighbourhood of the target and output vectors is determined by the number of outputs of the neural network. For simple image filtering this is 1, so the target is just the pixel in the reference image corresponding to the pixel in the input image. During learning the error is dumped to stdout. The error is the sum of the squared distances from the target and actual output vectors over all training vectors, devided by the number of training vectors and the number of outputs. The number of training vectors is equal to the number of pixels in the input image minus a border of half the local neighbourhood size (except when using block scan mode). The error is written after each cycle through the image for the learning process (epoch). The number of cycles is determined by the parameter *epoch* in the -l option. Also the network definition file is written after each epoch. It is written over the specified ndf-file.

In the recall operating mode bpima acts as an image filter. It reads the input image from stdin or the file *ima-in*, processes the data using the specified network, and writes the output to either stdout or *ima-out*. During learning and recall the image can be scanned in several different ways. Available are random and line (default) scanning and block scan mode. Random scan is only available during learning;

training vectors are selected randomly from the input image. The number of vectors still depends on the size of the image as described earlier. Block scan mode means that non overlapping neighbourhoods are selected. The block size is equal to the maximum of input and output neighbourhood sizes. Block mode is available in both learning and recall.
There is a possibility to send the output of any processing element in the network to the output image for a recall. This is done using the p-option.
Sometimes a considerable speedup in the learning process can be obtained by skipping addaption of the weight if the error for a particular training vector is smaller than a certain acceptation level. This level can be set using the a-option.
In order to make the grey-level range fit into the input and output range of the neural network, the scaling option is supplied. It is by default set so that the maximum output range of the network is used (scaling to 0..1).

OPTIONS

-r select random scan mode
-b select block scan mode
-c send line count to stderr
-t cumulate error and update after processing the whole training set (only in learning mode)
-m print whole error, inclusive accepted errors (learning mode)
-d output an image in doubles instead of bytes
-s *low high* scaling for image grey levels
default: low=0.0, high=1.0
-l *epoch* use network in learn mode; epoch cycles through the image
-p *layer pe* send output of pe pe in layer *layer* to output image
-a *accept* set acceptation level for error (default 0.01)

SEE ALSO

mknet(1), prtnet(1), bpnet(1), bplib(3), ima(3), ima(5), ndf(5)

BUGS

There may be some; if you find any, please report them as soon as possible.

AUTHOR

L.J. Spreeuwers

MKNET(1) USER COMMANDS MKNET(1)

NAME

mknet - build a simple feed forward backpropagation network in .ndf format

SYNOPSIS

mknet [-vl] [-s *seed*] [-f *net-in*] [*ndf-out*]

DESCRIPTION

mknet builds a simple feed forward backpropagation neural network in ndf file format. The network will be fully connected between the layers with only connections between succesive layers, and all processing elements connected to the bias. The network information is read from *net-in* (or stdin if this file is not given) and the ndf file is written to the file *ndf-out* or stdout if the file is not given.
The easy way to use mknet is in interactive mode (option -v). The program will ask a number of questions that must be answered. The first question it asks is about using short or long description. In short definition the momentum term is set to 0, and the transfer function to sigmoid. In the long definition these parameters can be set (per layer).

In non interactive mode mknet expects the following input from *net-in*, (or stdin):

<Inumber *of layers* <Inumber *of inputs* <Iweight *initialisation limit*

and then for each layer:

<Inumber *of pe's* <Ilearnrate { <Imomentum <Itransfer *function* }

The part between curly brackets {} should only be given for long definition (-l option).

The number of layers includes the input buffer layer. Since the input layer only buffers the input, it is defined by a single parameter, i.e. the number of inputs of the network. Therefor it should not be defined in the layer descriptions.

The weight initialisation limit determines the range of the uniform distribution for the random initialisation of the weights. The range is:

-weightinit .. +weightinit

The following transfer functions are available:

0 identity

1 parabola
2 sigmoid
3 exponential
4 logarithmic

OPTIONS

-v verbose - use interactive mode
-l use long definition
-s *seed* define seed of random number generator for weight initialisation
-f *net-in* read network input from file *net-in*

EXAMPLES

To generate a network with 3 layers (1 input buffer, 1 hidden layer and 1 output layer), 4 inputs, 2 pe's in the hidden layer and 1 in the output layer, and the learning rates of the hidden and output layer 0.9 and 0.1 respectively. The following input should be used for mknet in short definition mode:

```
3 4 0.5
2 0.9
1 0.1
```

The weight initialisation is uniform between -0.5 .. 0.5

To create a network with the same architecture, however with a momentum term of 0.3 for hidden and output layers, and a parabola transfer function in the hidden layer and a sigmoid for the output layer, the following input should be used in long definition mode (-l option):

```
3 4 0.5
2 0.9 0.3 1
1 0.1 0.3 2
```

SEE ALSO

bpnet(1), bplib(3), prtnet(1), ndf(5), bpima(1)

AUTHOR

L.J. Spreeuwers

Samenvatting

Beeldfiltering met neurale netwerken

Toepassingen en prestatie evaluatie

Neurale netwerken vertegenwoordigen een relatief nieuwe methode voor data- en informatieverwerking. Er bestaan verschillende benaderingen voor beeldverwerking en patroonherkenning met behulp van neurale netwerken, maar deze hebben over het algemeen betrekking op doel-zoek systemen, karakterherkenning, associatieve geheugens voor het opslaan en oproepen van beelden en het modelleren van de retina. In dit proefschrift staat de toepassing van neurale netwerken als beeldfilters centraal. Beeldfiltering technieken worden gebruikt voor o.a. beeld restauratie, beeld verbetering en kenmerkversterking en -extractie uit beelden. Het resultaat van deze operaties is opnieuw een beeld, dat op een bepaalde manier beter geschikt is voor verdere verwerking door mensen dan wel door machinale beeldverwerkingssystemen.
Een aantal belangrijke vragen die aan de orde komen zijn:

- wanneer is het nuttig om neurale netwerken te gebruiken voor beeldfiltering
- hoe kunnen neurale netwerken worden gebruikt als beeldfilters
- hoe kan de qualiteit van beeldfilters worden gemeten
- wat is de qualiteit van de beeldfilters op basis van neurale netwerken

Neurale netwerken worden getraind met behulp van voorbeelden van het gewenste gedrag. Dit betekent dat in het algemeen slechts weinig aandacht besteed hoeft te worden aan de modellering van de onderliggende processen. Het gevolg is dat het gebruik van

neurale netwerken wel een oplossing voor het probleem oplevert, maar dat bij het ontwerp weinig kennis over de processen wordt verkregen. Toepassing van neurale netwerken heeft dan ook voornamelijk zin als het verkrijgen van een werkende oplossing belangrijker is dan het verkrijgen van inzicht in de onderliggende processen.

Beeldfiltering met neurale netwerken kan zeer grote, moeilijk beheersbare netwerken opleveren. In dit werk wordt voorgesteld kleine neurale netwerken te gebruiken die werken op lokale omgevingen van pixels in een beeld. Het niveau van elk pixel in het uitgangsbeeld wordt bepaald door de niveaus van de pixels in een lokale omgeving van het corresponderende pixel in het ingangsbeeld. Een nadeel van deze benadering is dat slechts positie invariante filters kunnen worden gerealiseerd. Als voorbeelden van beeldfilters op basis van neurale netwerken worden filters voor beeldopscherping, ruisonderdrukking en randdetectie (edge detection) beschreven. De filters zijn gerealiseerd met behulp van een software pakket, bplib, voor simulatie van error backpropagation netwerken, dat speciaal voor dit doel is geschreven.

Het is verbazingwekkend hoe weinig er gepubliceerd is over het evalueren van de qualiteit van beeldverwerkingssystemen en -operaties. Vaak wordt geprobeerd de qualiteit te definiëren los van hun toepassingsgebieden. Een belangrijk uitgangspunt in dit werk is dat de qualiteit van beeldverwerkingssystemen en -operaties alleen maar gedefinieerd mag worden in relatie met hun toepassingen. De methode voor de evaluatie die in dit werk wordt voorgesteld is gebaseerd op het "gemiddelde risico" (average risk, AVR). In de AVR worden de kansen op fouten (of de frequenties van fouten) gewogen met de kosten van de fouten en door sommatie gecombineerd. In het geval van edge detectie betekent dit bijvoorbeeld dat voor een bepaalde edge detector de kans dat een edge pixel niet gedetecteerd wordt met kosten 1 wordt gestraft, terwijl de kans dat een edge pixel wel gedetecteerd wordt, maar niet correct gelokaliseerd in lagere kosten resulteert. Het voorbeeld van evaluatie van edge detectoren wordt ook verder uitgewerkt. De kosten van de verschillende typen fouten hangen af van de toepassing van het beeldverwerkingssysteem. Vanuit de statistische patroonherkenning is er een goede theoretische onderbouwing voor de AVR. Deze wordt in dit werk uitgebreid en aangepast aan de toepassing voor evaluatie van beeldverwerkingssystemen en -operaties. Hiermee wordt een theoretisch raamwerk geschapen voor de evaluatie van beeldverwerkingssystemen en -operaties in het algemeen.

Vergelijking van de prestaties van beeldfilters op basis van neurale netwerken met andere beeldfilters laat zien dat i.h.a. vergelijkbare resultaten behaald kunnen worden. Voordeel van de toepassing van neurale netwerken is vooral dat op eenvoudige wijze een beeldfilter voor een specifieke toepassing ontworpen kan worden (door de juiste voorbeelden aan te bieden), zonder dat daarvoor diepgaande analyse van het gewenste gedrag nodig is. Voor een specifieke toepassing kunnen filters worden verkregen die superieur zijn aan oplossingen op basis van slechte of onvolledige modellering (bijvoorbeeld omdat modellering erg complex is).

Index