Nonlinear image processing using artificial neural networks

Dick de Ridder*, Robert P.W. Duin*,
Michael Egmont-Petersen†,
Lucas J. van Vliet* and Piet W. Verbeek*
*Pattern Recognition Group, Dept. of Applied Physics,
Delft University of Technology
Lorentzweg 1, 2628 CJ Delft, The Netherlands
†Decision Support Systems Group,
Institute of Information and Computing Sciences, Utrecht University
PO box 80089, 3508 TB Utrecht, The Netherlands

Contents

Abstract

Artificial neural networks (ANNs) are very general function approximators which can be trained based on a set of examples. Given their general nature, ANNs would seem useful tools for nonlinear image processing. This paper tries to answer the question whether image processing operations can sucessfully be learned by ANNs; and, if so, how prior knowledge can be used in their design and what can be learned about the problem at hand from trained networks. After an introduction to ANN types and a brief literature review, the paper focuses on two cases: supervised classification ANNs for object recognition and feature extraction; and supervised regression ANNs for image pre-processing. A range of experimental results lead to the conclusion that ANNs are mainly applicable to problems requiring a nonlinear solution, for which there is a clear, unequivocal performance criterion, i.e. high-level tasks in the image processing chain (such as object recognition) rather than low-level tasks. The drawbacks are that prior knowledge cannot easily be used, and that interpretation of trained ANNs is hard.

1 Introduction

1.1 Image processing

Image processing is the field of research concerned with the development of computer algorithms working on digitised images (e.g. Pratt, 1991; Gonzalez and Woods, 1992). The range of problems studied in image processing is large, encompassing everything from low-level signal enhancement to high-level image understanding. In general, image processing problems are solved by a chain of tasks. This chain, shown in figure 1, outlines the possible processing needed from the initial sensor data to the outcome (e.g. a classification or a scene description). The pipeline consists of the steps of pre-processing, data reduction, segmentation, object recognition and image understanding. In each step, the input and output data can either be images (pixels), measurements in images (features), decisions made in previous stages of the chain (labels) or even object relation information (graphs).

There are many problems in image processing for which good, theoretically justifiable solutions exists, especially for problems for which linear solutions suffice. For example, for pre-processing operations such as image restoration, methods from signal processing such as the Wiener filter can be shown to be the optimal linear approach. However, these solutions often only work under

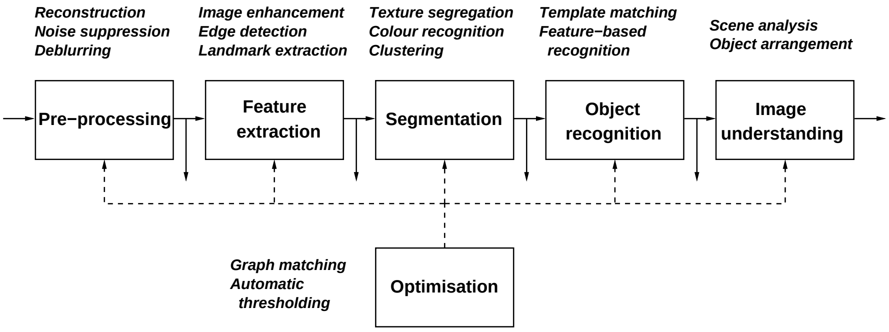


Figure 1: The image processing chain.

ideal circumstances; they may be highly computationally intensive (e.g. when large numbers of linear models have to be applied to approximate a nonlinear model); or they may require careful tuning of parameters. Where linear models are no longer sufficient, nonlinear models will have to be used. This is still an area of active research, as each problem will require specific nonlinearities to be introduced. That is, a designer of an algorithm will have to weigh the different criteria and come to a good choice, based partly on experience. Furthermore, many algorithms quickly become intractable when nonlinearities are introduced. Problems further in the image processing chain, such object recognition and image understanding, cannot even (yet) be solved using standard techniques. For example, the task of recognising any of a number of objects against an arbitrary background calls for human capabilities such as the ability to generalise, associate etc.

All this leads to the idea that nonlinear algorithms that can be trained, rather than designed, might be valuable tools for image processing. To explain why, a brief introduction into artificial neural networks will be given first.

1.2 Artificial neural networks (ANNs)

In the 1940s, psychologists became interested in modelling the human brain. This led to the development of the a model of the neuron as a thresholded summation unit (McCulloch and Pitts, 1943). They were able to prove that (possibly large) collections of interconnected neuron models, neural networks, could in principle perform any computation, if the strengths of the interconnections (or weights) were set to proper values. In the 1950s neural networks were picked up by the growing artificial intelligence community.

In 1962, a method was proposed to train a subset of a specific class of networks, called perceptrons, based on examples (Rosenblatt, 1962). Perceptrons are networks having neurons grouped in layers, with only connections between neurons in subsequent layers. However, Rosenblatt could only prove convergence for single-layer perceptrons. Although some training algorithms for larger neural networks with hard threshold units were proposed (Nilsson, 1965), enthusiasm waned after it was shown that many seemingly simple problems were in fact nonlinear and that perceptrons were incapable of solving

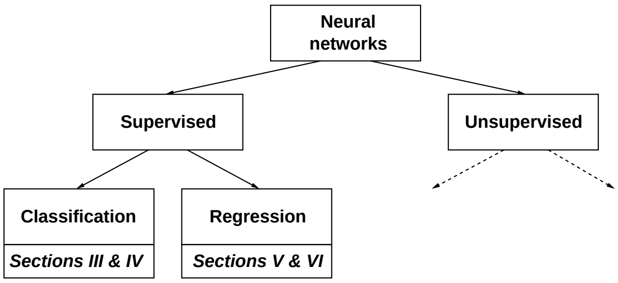


Figure 2: Adaptive method types discussed in this paper.

these (Minsky and Papert, 1969).

Interest in artificial neural networks (henceforth "ANNs") increased again in the 1980s, after a learning algorithm for multi-layer perceptrons was proposed, the back-propagation rule (Rumelhart et al., 1986). This allowed nonlinear multi-layer perceptrons to be trained as well. However, feed-forward networks were not the only type of ANN under research. In the 1970s and 1980s a number of different biologically inspired learning systems were proposed. Among the most influential were the Hopfield network (Hopfield, 1982; Hopfield and Tank, 1985), Kohonen's self-organising map (Kohonen, 1995), the Boltzmann machine (Hinton et al., 1984) and the Neocognitron (Fukushima and Miyake, 1982).

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

- consist of (a large number of) identical, simple processing units;
- have interconnections between these units;
- posess tunable parameters (weights) which define the system's function and
- lack a supervisor which tunes each individual weight.

However, not all systems that are called neural networks fit this description.

There are many possible taxonomies of ANNs. Here, we concentrate on *learning* and *functionality* rather than on biological plausibility, topology etc. Figure 2 shows the main subdivision of interest: *supervised* versus *unsupervised* learning. Although much interesting work has been done in unsupervised learning for image processing (see e.g. Egmont-Petersen et al., 2002), we will restrict ourselves to supervised learning in this paper. In supervised learning, there is a data set $\mathcal{L}$ containing samples in $\mathbf{x} \in \mathbb{R}^d$, where d is the number of dimensions of the data set. For each $\mathbf{x}$ a dependent variable $\mathbf{y} \in \mathbb{R}^m$ has to be supplied as well. The goal of a *regression* method is then to predict this dependent variable based on $\mathbf{x}$. *Classification* can be seen as a special case of regression, in which only a single variable $t \in \mathbb{N}$ is to be predicted, the label of the class to which the sample $\mathbf{x}$ belongs.

In section 2, the application of ANNs to these tasks will be discussed in more detail.

1.3 ANNs for image processing

As was discussed above, dealing with nonlinearity is still a major problem in image processing. ANNs might be very useful tools for nonlinear image processing:

- instead of designing an algorithm, one could construct an example data set and an error criterion, and train ANNs to perform the desired input-output mapping;

- the network input can consist of pixels or measurements in images; the output can contain pixels, decisions, labels, etc., as long as these can be coded numerically – no assumptions are made. This means adaptive methods can perform several steps in the image processing chain at once;

- ANNs can be highly nonlinear; the amount of nonlinearity can be influenced by design, but also depends on the training data (Raudys, 1998a; Raudys, 1998b);

- some types of ANN have been shown to be universal classification or regression techniques (Funahashi, 1989; Hornik et al., 1989).

However, it is not to be expected that application of any ANN to any given problem will give satisfactory results. This paper therefore studies the possibilities and limitations of the ANN approach to image processing. The main questions it tries to answer are:

- *Can image processing operations be learned by ANNs?* To what extent can ANNs solve problems that are hard to solve using standard techniques? Is nonlinearity really a bonus?

- *How can prior knowledge be used*, if available? Can, for example, the fact that neighbouring pixels are highly correlated be used in ANN design or training?

- *What can be learned from ANNs trained to solve image processing problems?* If one finds an ANN to solve a certain problem, can one learn how the problem should be approached using standard techniques? Can one extract knowledge from the solution?

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

It is to be expected that for different ANN types, the answers to these questions will be different. This paper is therefore laid out as follows:

- first, in section 2, a brief literature overview of applications of ANNs to image processing is given;

- in sections 3 and 4, classification ANNs are applied to object recognition and feature extraction;

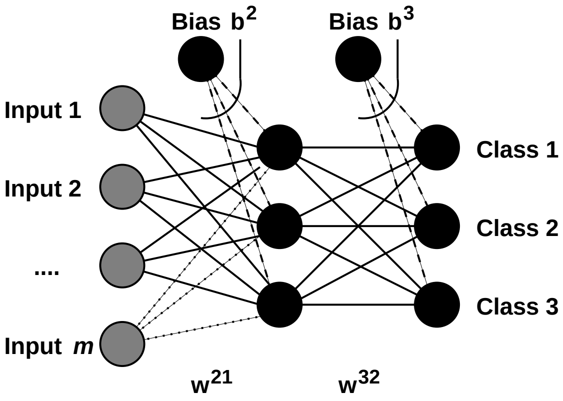


Figure 3: A feed-forward ANN for a three-class classification problem. The center layer is called the hidden layer.

- in sections 5 and 6, regression ANNs are investigated as nonlinear image filters.

These methods are not only applied to real-life problems, but also studied to answer the questions outlined above. In none of the applications the goal is to obtain better performance than using traditional methods; instead, the goal is to find the conditions under which ANNs could be applied.

2 Applications of ANNs in image processing

This section will first discuss the most widely used type of ANN, the feed-forward ANN, and its use as a classifier or regressor. Afterwards, a brief review of applications of ANNs to image processing problems will be given.

2.1 Feed-forward ANNs

This paper will deal mostly with feed-forward ANNs (Hertz et al., 1991; Haykin, 1994) (or multi-layer perceptrons, MLPs). They consist of interconnected layers of processing units or *neurons*, see figure 3. In this figure, the notation of weights and biases follows (Hertz et al., 1991): weights of connections between layer p and layer q are indicated by $\mathbf{w}^{qp}$; the bias, input and output vectors of layer p are indicated by $\mathbf{b}^p$, $\mathbf{I}^p$ and $\mathbf{O}^p$, respectively. Basically, a feed-forward ANN is a (highly) parameterised, adaptable vector function, which may be trained to perform classification or regression tasks. A classification feed-forward ANN performs the mapping

$$N : \mathbb{R}^d \rightarrow \langle r_{min}, r_{max} \rangle^m, \qquad (1)$$

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

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

$\mathbf{W}$ is the weight set, containing the weight matrix connecting the input layer with the hidden layer ($\mathbf{w}^{21}$) and the vector connecting the hidden layer with the output layer ($\mathbf{w}^{32}$); $\mathbf{B}$ ($\mathbf{b}^2$ and $\mathbf{b}^3$) contains the bias terms of the hidden and output nodes, respectively. The function $f(a)$ is the nonlinear activation function with range $\langle r_{min}, r_{max}\rangle$, operating on each element of its input vector. Usually, one uses either the sigmoid function, $f(a) = \frac{1}{1+e^{-a}}$, with the range $\langle r_{min} = 0, r_{max} = 1\rangle$; the double sigmoid function $f(a) = \frac{2}{1+e^{-a}} - 1$; or the hyperbolic tangent function $f(a) = \tanh(a)$, both with range $\langle r_{min} = -1, r_{max} = 1\rangle$.

2.1.1 Classification

To perform classification, an ANN should compute the posterior probabilities of given vectors $\mathbf{x}$, $P(\omega_j|\mathbf{x})$, where ω_j is the label of class j, $j = 1,\ldots,m$. Classification is then performed by assigning an incoming sample $\mathbf{x}$ to that class for which this probability is highest. A feed-forward ANN can be trained in a supervised way to perform classification, when presented with a number of training samples $\mathcal{L} = \{(\mathbf{x},\mathbf{t})\}$, with t_l high (e.g. 0.9) indicating the correct class membership and t_k low (e.g. 0.1), $\forall k \neq l$. The training algorithm, for example back-propagation (Rumelhart et al., 1986) or conjugate gradient descent (Shewchuk, 1994), tries to minimise the mean squared error (MSE) function:

$$E(\mathbf{W},\mathbf{B}) = \frac{1}{2|\mathcal{L}|}\sum_{(\mathbf{x}^i,\mathbf{t}^i)\in\mathcal{L}}\sum_{k=1}^{c}(N(\mathbf{x}^i;\ \mathbf{W},\mathbf{B})_k - t_k^i)^2, \tag{3}$$

by adjusting the weights and bias terms. For more details on training feed-forward ANNs, see e.g. (Hertz et al., 1991; Haykin, 1994). (Richard and Lippmann, 1991) showed that feed-forward ANNs, when provided with enough nodes in the hidden layer, an infinitely large training set and 0-1 training targets, approximate the Bayes posterior probabilities

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})},\ j = 1,\ldots,m, \tag{4}$$

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

2.1.2 Regression

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

$$R : \mathbb{R}^d \rightarrow \mathbb{R}^m, \tag{5}$$

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

$$R(\mathbf{x};\ \mathbf{W},\mathbf{B}) = {\mathbf{w}^{32}}^T f({\mathbf{w}^{21}}^T\mathbf{x} - \mathbf{b}^2) - \mathbf{b}^3. \tag{6}$$

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

$$\mathrm{E}(\mathbf{y}|\mathbf{x}) = \int_{\mathbb{R}^m} \mathbf{y} p(\mathbf{y}|\mathbf{x}) d\mathbf{y}. \tag{7}$$

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

$$E(\mathbf{W}, \mathbf{B}) = \frac{1}{2|\mathcal{L}|} \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{L}} \sum_{k=1}^{m} (R(\mathbf{x}^i;\ \mathbf{W}, \mathbf{B})_k - y_k^i)^2, \tag{8}$$

(or the prediction error) is minimised.

Several authors showed that, under some assumptions, regression feed-forward ANNs are universal approximators. If the number of hidden nodes is allowed to increase towards infinity, they can approximate any continuous function with arbitrary precision (Funahashi, 1989; Hornik et al., 1989). When a feed-forward ANN is trained to approximate a discontinuous function, two hidden layers are sufficient for obtaining an arbitrary precision (Sontag, 1992).

However, this does not make feed-forward ANNs perfect classification or regression machines. There are a number of problems:

- there is no theoretically sound way of choosing the optimal ANN architecture or number of parameters. This is called the *bias-variance dilemma* (Geman et al., 1992): for a given data set size, the more parameters an ANN has, the better it can approximate the function to be learned; at the same time, the ANN becomes more and more susceptible to *overtraining*, i.e. adapting itself completely to the available data and losing generalisation;

- for a given architecture, learning algorithms often end up in a local minimum of the error measure E instead of a global minimum[1];

- they are *non-parametric*, i.e. they do not specify a model and are less open to explanation. This is sometimes referred to as the *black box problem*. Although some work has been done in trying to extract rules from trained ANNs (Tickle et al., 1998), in general it is still impossible to specify exactly how an ANN performs its function. For a rather polemic discussion on this topic, see the excellent paper by Green (Green, 1998)).

2.2 Other ANN types

Two other major ANN types are:

[1] Although current evidence suggests this is actually one of the features that makes feed-forward ANNs powerful: the limitations the learning algorithm imposes actually manage the bias-variance problem (Raudys, 1998a; Raudys, 1998b).

- the self-organising map (SOM, Kohonen, 1995; also called topological map) is a kind of vector quantisation method. SOMs are trained in an unsupervised manner with the goal of projecting similar d-dimensional input vectors to neighbouring positions (*nodes*) on an m-dimensional discrete lattice. Training is called *competitive*: at each time step, one winning node gets updated, along with some nodes in its neighbourhood. After training, the input space is subdivided into q regions, corresponding to the q nodes in the map. An important application of SOMs in image processing is therefore unsupervised cluster analysis, e.g. for segmentation.

- the Hopfield ANN (HNN, Hopfield, 1982) consists of a number of fully interconnected binary nodes, which at each given time represent a certain state. Connected to a state is an energy level, the output of the HNN's energy function given the state. The HNN maps binary input sets on binary output sets; it is initialised with a binary pattern and by iterating an update equation, it changes its state until the energy level is minimised. HNNs are not thus trained in the same way that feed-forward ANNs and SOMs are: the weights are usually set manually. Instead, the power of the HNN lies in running it.

  Given a rule for setting the weights based on a training set of binary patterns, the HNN can serve as an auto-associative memory (given a partially completed pattern, it will find the nearest matching pattern in the training set). Another application of HNNs, which is quite interesting in an image processing setting (Poggio and Koch, 1985), is finding the solution to nonlinear optimisation problems. This entails mapping a function to be minimised on the HNN's energy function. However, the application of this approach is limited in the sense that the HNN minimises just one energy function, whereas most problems are more complex in the sense that the minimisation is subject to a number of constraints. Encoding these constraints into the energy function takes away much of the power of the method, by calling for a manual setting of various parameters which again influence the outcome.

2.3 Applications of ANNs

Image processing literature contains numerous applications of the above types of ANNs and various other, more specialised models. Below, we will give a broad overview of these applications, without going into specific ones. Furthermore, we will only discuss application of ANNs directly to pixel data (i.e. not to derived features). For a more detailed overview, see e.g. Egmont-Petersen et al., 2002.

2.3.1 Pre-processing

Pre-processing an image can consist of image reconstruction (building up an image from a number of indirect sensor measurements) and/or image restoration (removing abberations introduced by the sensor, including noise). To perform pre-processing, ANNs have been applied in the following ways:

- optimisation of an objective function specified by a traditional pre-processing approach;

- approximation of a mathematical transformation used in reconstruction, by regression;
- general regression/classification, usually directly on pixel data (neighbourhood input, pixel output).

To solve the first type of problem, HNNs can be used for the optimisation involved in traditional methods. However, mapping the actual problem to the energy function of the HNN can be difficult. Occasionally, the original problem will have to be modified. Having managed to map the problem appropriately, the HNN can be a useful tool in image pre-processing, although convergence to a good result is not guaranteed.

For image reconstruction, regression (feed-forward) ANNs can be applied. Although some succesful applications are reported in literature, it would seem that these applications call for more traditional mathematical techniques, because a guaranteed performance of the reconstruction algorithm is essential.

Regression or classification ANNs can also be trained to perform image restoration directly on pixel data. In literature, for a large number of applications, *non-adaptive* ANNs were used. Where ANNs are adaptive, their architectures usually differ much from those of the standard ANNs: prior knowledge about the problem is used to design them (e.g. in cellular neural networks, CNNs). This indicates that the fast, parallel operation of ANNs, and the ease with which they can be embedded in hardware, can be important factors in choosing for a neural implementation of a certain pre-processing operation. However, their ability to learn from data is apparently of less importance. We will return to this in sections 5 and 6.

2.3.2 Enhancement and feature extraction

After pre-processing, the next step in the image processing chain is extraction of information relevant to later stages (e.g. subsequent segmentation or object recognition). In its most generic form, this step can extract low-level information such as edges, texture characteristics etc. This kind of extraction is also called image *enhancement*, as certain general (perceptual) features are enhanced. As enhancement algorithms operate without a specific application in mind, the goal of using ANNs is to outperform traditional methods either in accuracy or computational speed. The most well-known enhancement problem is edge detection, which can be approached using classification feed-forward ANNs. Some modular approaches, including estimation of edge strength or denoising, have been proposed. Morphological operations have also been implemented on ANNs, which were equipped with shunting mechanisms (neurons acting as switches). Again, as in pre-processing, prior knowledge is often used to restrict the ANNs.

Feature extraction entails finding more application-specific geometric or perceptual features, such as corners, junctions and object boundaries. For particular applications, even more high-level features may have to be extracted, e.g. eyes and lips for face recognition. Feature extraction is usually tightly coupled with classification or regression; what variables are informative depends on the application, e.g. object recognition. Some ANN approaches therefore consist of two stages, possibly coupled, in which features are extracted by the first ANN and object recognition is performed by the second ANN. If the two are completely integrated, it can be hard to label a specific part as a feature extractor (see also section 4).

Feed-forward ANNs with bottlenecks (*auto-associative ANNs*) and SOMs are useful for nonlinear feature extraction. They can be used to map high-dimensional image data onto a lower number of dimensions, preserving as well as possible the information contained. A disadvantage of using ANNs for feature extraction is that they are not by default invariant to translation, rotation or scale, so if such invariances are desired they will have to be built in by the ANN designer.

2.3.3 Segmentation

Segmentation is partitioning an image into parts that are coherent according to some criterion: texture, colour or shape. When considered as a classification task, the purpose of segmentation is to assign labels to individual pixels or voxels. Classification feed-forward ANNs and variants can perform segmentation directly on pixels, when pixels are represented by windows extracted around their position. More complicated modular approaches are possible as well, with modules specialising in certain subclasses or invariances. Hierarchical models are sometimes used, even built of different ANN types, e.g. using a SOM to map the image data to a smaller number of dimensions and then using a feed-forward ANN to classify the pixel.

Again, a problem here is that ANNs are not naturally invariant to transformations of the image. Either these transformations will have to be removed beforehand, the training set will have to contain all possible transformations, or invariant features will have to be extracted from the image first. For a more detailed overview of ANNs applied to image segmentation, see (Pal and Pal, 1993).

2.3.4 Object recognition

Object recognition consists of locating the positions and possibly orientations and scales of instances of classes of objects in an image (object detection) and classifying them (object classification). Problems that fall into this category are e.g. optical character recognition, automatic target recognition and industrial inspection. Object recognition is potentially the most fruitful application area of pixel-based ANNs, as using an ANN approach makes it possible to roll several of the preceding stages (feature extraction, segmentation) into one and train it as a single system.

Many feed-forward-like ANNs have been proposed to solve problems. Again, invariance is a problem, leading to the proposal of several ANN architectures in which connections were restricted or shared corresponding to desired invariances (e.g. Fukushima and Miyake, 1982; Le Cun et al., 1989a). More involved ANN approaches include hierarchical ANNs, to tackle the problem of rapidly increasing ANN complexity with increasing image size; and multi-resolution ANNs which include context information.

2.3.5 Image understanding

Image understanding is the final step in the image processing chain, in which the goal is to *interpret* the image content. Therefore, it couples techniques from segmentation or object recognition with the use of prior knowledge of the expected image content (such as image semantics). As a consequence, there are only few applications of ANNs on pixel data. These are usually complicated, modular approaches.

A major problem when applying ANNs for high level image understanding is their black-box character. Although there are proposals for explanation facilities (Egmont-Petersen et al., 1998a) and rule extraction (Tickle et al., 1998), it is usually hard to explain why a particular image interpretation is the most likely one, Another problem in image understanding relates to the amount of input data. When, e.g., seldomly occurring images are provided as input to a neural classifier, a large number of images are required to establish statistically representative training and test sets.

2.3.6 Optimisation

Some image processing (sub)tasks such as stereo matching can best be formulated as optimisation problems, which may be solved by HNNs. HNNs have been applied to optimisation problems in reconstruction and restoration, segmentation, (stereo) matching and recognition. Mainly, HNNs have been applied for tasks that are too difficult to realise with other neural classifiers because the solutions entail partial graph matching or recognition of 3D objects. A disadvantage of HNNs is that training and use are both of high computational complexity.

2.4 Discussion

One of the major advantages of ANNs is that they are applicable to a wide variety of problems. There are, however, still caveats and fundamental problems that require attention. Some problems are caused by using a statistical, data-oriented technique to solve image processing problems; other problems are fundamental to the way ANNs work.

Problems with data-oriented approaches A problem in the application of data-oriented techniques to images is how to incorporate context information and prior knowledge about the expected image content. Prior knowledge could be knowledge about the typical shape of objects one wants to detect, knowledge of the spatial arrangement of textures or objects or of a good approximate solution to an optimisation problem. According to (Perlovsky, 1998), the key to restraining the highly flexible learning algorithms ANNs are, lies in the very combination with prior knowledge. However, most ANN approaches do not even use the prior information that neighbouring pixel values are highly correlated. The latter problem can be circumvented by extracting features from images first, by using distance or error measures on pixel data which do take spatial coherency into account (e.g. Hinton et al., 1997; Simard et al., 1993), or by designing an ANN with spatial coherency (e.g. Le Cun et al., 1989a; Fukushima and Miyake, 1982) or contextual relations beween objects in mind. On a higher level, some methods, such as hierarchical object recognition ANNs can provide context information.

In image processing, classification and regression problems quickly involve a very large number of input dimensions, especially when the algorithms are applied directly on pixel data. This is problematic, as ANNs to solve these problems will also grow, which makes them harder to train. However, the most interesting future applications (e.g. volume imaging) promise to deliver even more input. One way to cope with this problem is to develop feature-based pattern recognition approaches; another way would be to design an architecture that quickly adaptively downsamples the original image.

Finally, there is a clear need for thorough validation of the developed image processing algorithms (Haralick, 1994; De Boer and Smeulders, 1996). Unfortunately, only few of the publications about ANN applications ask the question whether an ANN really is the best way of solving the problem. Often, comparison with traditional methods is neglected.

Problems with ANNs Several theoretical results regarding the approximation capabilities of ANNs have been proven. Although feed-forward ANNs with two hidden layers can approximate any (even discontinuous) function to an arbitrary precision, theoretical results on, e.g., convergence are lacking. The combination of initial parameters, topology and learning algorithm determines the performance of an ANN after its training has been completed. Furthermore, there is always a danger of overtraining an ANN, as minimising the error measure occasionally does not correspond to finding a well-generalising ANN.

Another problem is how to choose the best ANN architecture. Although there is some work on model selection (Fogel, 1991; Murata et al., 1994), no general guidelines exist which guarantee the best trade-off between model bias and variance (see page 8) for a particular size of the training set. Training unconstrained ANNs using standard performance measures such as the mean squared error might even give very unsatisfying results. This, we assume, is the reason why in a number of applications, ANNs were not adaptive at all or heavily constrained by their architecture.

ANNs suffer from what is known as the black-box problem: the ANN, once trained, might perform well but offers no explanation on how it works. That is, given any input a corresponding output is produced, but it cannot be easily explained why this decision was reached, how reliable it is, etc. In some image processing applications, e.g., monitoring of (industrial) processes, electronic surveillance, biometrics, etc. a measure of the reliability is highly necessary to prevent costly false alarms. In such areas, it might be preferable to use other, less well performing methods that do give a statistically profound measure of reliability.

As was mentioned in section 1, this paper will focus both on actual applications of ANNs to image processing tasks and the problems discussed above:

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

In the next section, an ANN architecture developed specifically to address these problems, the shared weight ANN, will be investigated.

3 Shared weight networks for object recognition

In this section, some applications of shared weight neural networks will be discussed. These networks are more commonly known in the literature as

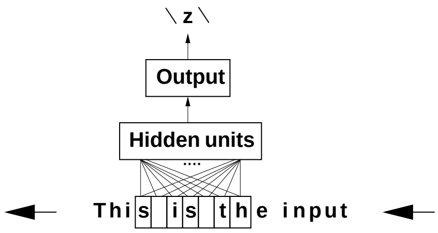


Figure 4: The operation of the ANN used in Sejnowski's NETtalk experiment. The letters (and three punctuation marks) were coded by 29 input units using place coding: that is, the ANN input vector contained all zeroes with one element set to one, giving $7 \times 29 = 203$ input units in total. The hidden layer contained 80 units and the output layer 26 units, coding the phoneme.

TDNNs, *Time Delay Neural Networks* (Bengio, 1996), since the first applications of this type of network were in the field of speech recognition[2]. (Sejnowski and Rosenberg, 1987) used a slightly modified feed-forward ANN in their NETtalk speech synthesis experiment. Its input consisted of an alpha numerical representation of a text; its training target was a representation of the phonetic features necessary to pronounce the text. Sejnowski took the input of the ANN from the "stream" of text with varying time delays, each neuron effectively implementing a convolution function; see figure 4. The window was 7 frames wide and static. The higher layers of the ANN were just of the standard feed-forward type. Two-dimensional TDNNs later developed for image analysis really are a generalisation of Sejnowski's approach: they used the weight-sharing technique not only after the input layer, but for two or three layers. To avoid confusion, the general term "shared weight ANNs" will be used.

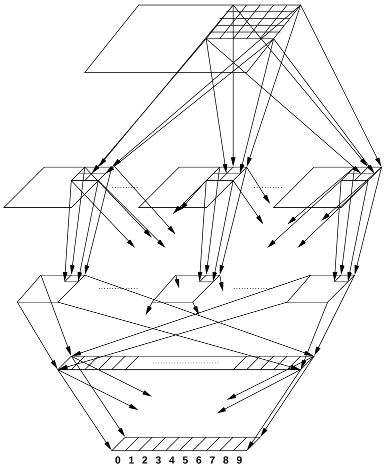
This section will focus on just one implementation of shared weight ANNs, developed by Le Cun et al. (Le Cun et al., 1989a). This ANN architecture is interesting, in that it incorporates prior knowledge of the problem to be solved – object recognition in images – into the structure of the ANN itself. The first few layers act as convolution filters on the image, and the entire ANN can be seen as a nonlinear filter. This also allows us to try to interpret the weights of a trained ANN in terms of image processing operations.

First, the basic shared weight architecture will be introduced, as well as some variations. Next an application to handwritten digit recognition will be shown. The section ends with a discussion on shared weight ANNs and the results obtained.

3.1 Shared weight networks

The ANN architectures introduced by Le Cun et al. (Le Cun et al., 1989a) use the concept of sharing weights, that is, a set of neurons in one layer using the

[2] The basic mechanisms employed in TDNNs, however, were known long before. In 1962, (Hubel and Wiesel, 1962) introduced the notion of receptive fields in mammalian brains. (Rumelhart et al., 1986) proposed the idea of sharing weights for solving the T-C problem, in which the goal is to classify a 3×3 pixel letter T and a 3×2 pixel letter C, independent of translation and rotation (Minsky and Papert, 1969).



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

Figure 5: The LeCun shared weight ANN.

same incoming weight (see figure 5). The use of shared weights leads to all these neurons detecting the same feature, though at different positions in the input image (*receptive fields*); i.e. the image is convolved with a kernel defined by the weights. The detected features are – at a higher level – combined, to obtain shift-invariant feature detection. This is combined with layers implementing a subsampling operation to decrease resolution and sensitivity to distortions. Le Cun et al. actually describe several different architectures (Le Cun et al., 1989b), though all of these use the same basic techniques.

Shared weight ANNs have been applied to a number of other recognition problems, such as word recognition (Bengio et al., 1994), cursive script recognition (Schenkel et al., 1995), face recognition (Lawrence et al., 1997; Fogelman Soulie et al., 1993; Viennet, 1993), automatic target recognition (Gader et al., 1995) and hand tracking (Nowlan and Platt, 1995). Other architectures employing the same ideas can be found as well. In (Fukushima and Miyake, 1982), an ANN architecture specifically suited to object recognition is proposed; the Neocognitron. It is based on the workings of the visual nervous system, and uses the technique of receptive fields and of combining local features at a higher level to more global features (see also 22.3.4). The ANN can handle positional shifts and geometric distortion of the input image. Others have applied standard feed-forward ANNs in a convolution-like way to large images. Spreeuwers (Spreeuwers, 1992) and Greenhill and Davies (Greenhil and Davies, 1994) trained ANNs to act as filters, using pairs of input-output images.

3.1.1 Architecture

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

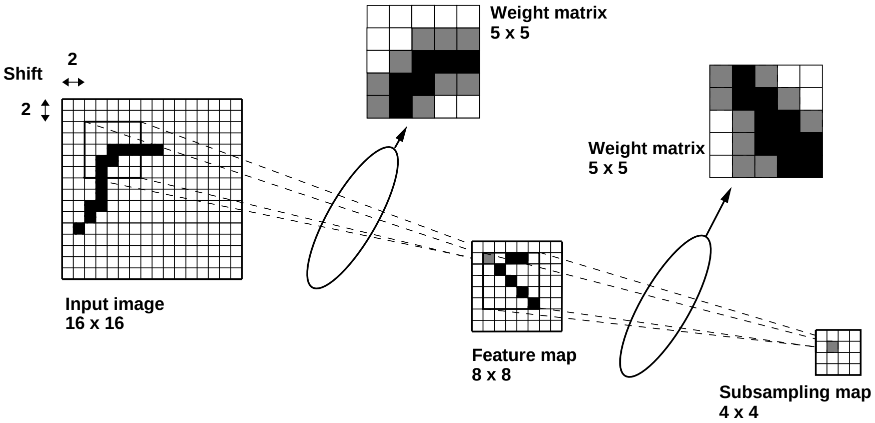


Figure 6: A feature map and a subsampling map.

- The **input layer** consists of a grey-level image of 16 × 16 pixels.
- The second layer contains the so-called **feature maps**; see figure 6. Each neuron in such a feature map has the same 5 × 5 set of incoming weights, but is connected to a square at a unique position in the input image. This set can be viewed as a convolution filter, or template; that is, if a neuron in a feature map has high output, this corresponds to a match with the template. The place of the match in the input image corresponds to the place of the neuron in the feature map. The image is under-sampled, as the receptive field for two neighbouring neurons is shifted two pixels in the input image. The rationale behind this is that, while high resolution is important for detecting a feature, it is not necessary to know its position in the image with the same precision.

  Note that the number of connections between the input and feature map layer is far greater than the number of weights, due to the weight-sharing. However, neurons do *not* share their bias. Figure 5 shows the number of neurons, connections and weights for each layer.

- The third layer consists of **sub-sampling maps** (figure 6). This layer is included mainly to reduce the number of free parameters. The principle is the same as for the feature maps: each neuron in a sub-sampling map is connected to a 5×5 square and all neurons in one sub-sampling map share the same set of 25 weights. Here, too, the feature map is under-sampled, again losing some of the information about the place of detected features.

  The main difference however, is that each neuron in a sub-sampling map is connected to more than one feature map. This mapping of feature maps onto sub-sampling maps is not trivial; Le Cun et al. use different approaches in their articles. In (Le Cun et al., 1989a), only the number of feature maps connected to each sub-sampling map, 8, is mentioned; it is not clear which feature maps are linked to which sub-sampling maps. In (Le Cun et al., 1989b) however, table 1 is given. Again, due to the

	Subsampling map											
	1	2	3	4	5	6	7	8	9	10	11	12
Feature map 1	●	●	●							●	●	●
2	●	●	●							●	●	●
3	●	●	●	●	●	●						
4	●	●	●	●	●	●						
5				●	●	●	●	●	●			
6				●	●	●	●	●	●			
7							●	●	●	●	●	●
8							●	●	●	●	●	●
9	●	●	●	●	●	●	●	●	●	●	●	●
10	●	●	●	●	●	●	●	●	●	●	●	●
11	●	●	●	●	●	●	●	●	●	●	●	●
12	●	●	●	●	●	●	●	●	●	●	●	●

Table 1: Connections between the feature map layer and subsampling map layer in the LeCun architecture.

use of shared weights, there are significantly less weights than connections (although biases are not shared). See figure 5 for an overview.

- The output of the sub-sampling map is propagated to a **hidden layer**. This layer is fully connected to the sub-sampling layer. The number of neurons is 30.

- The **output layer** is fully connected to the hidden layer. It contains 10 neurons, and uses place coding for classification; the neurons are numbered $0 \ldots 9$, and the neuron with the highest activation is chosen. The digit recognised is equal to the neuron number.

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

Shared weight ANNs can be trained by any standard training algorithm for feed-forward ANNs (Hertz et al., 1991; Haykin, 1994), provided that the derivative of the cost function with respect to a shared weight is defined as the sum of the derivatives with respect to the non-shared weights (Viennet, 1993). The individual weight updates are used to update the bias for each neuron, since biases are not shared.

Clearly, the architecture presented uses prior knowledge (recognising local features, combining them at a higher level) about the task to solve (i.e., object recognition), thus addressing the problem discussed in section 22.4 In (Solla and Le Cun, 1991), the authors show that this approach indeed gives better performance. They compare three simple architectures: a standard back-propagation ANN, an ANN with one feature map and one sub-sampling map and an ANN with two feature maps, each mapped onto one sub-sampling map. It is shown that the more prior knowledge is put into the ANN, the higher its generalisation ability[3].

3.1.2 Other implementations

Although the basics of other ANN architectures proposed by Le Cun et al. and others are the same, there are some differences to the one discussed above (Le Cun et al., 1989a). In (Le Cun et al., 1990), an extension of the architecture is proposed with a larger number of connections, but a number of unique parameters even lower than that of the LeCun ANN. The "LeNotre" architecture is a proposal by Fogelman Soulie et al. in (Fogelman Soulie et al., 1993) and, under the name Quick, in (Viennet, 1993). It was used to show that the ideas that resulted in the construction of the ANNs described above can be used to make very small ANNs that still perform reasonably well. In this architecture, there are only two feature map layers of two maps each; the first layer contains two differently sized feature maps.

3.2 Handwritten digit recognition

This section describes some experiments using the LeCun ANNs in a handwritten digit recognition problem. For a more extensive treatment, see (de Ridder, 2001). The ANNs are compared to various traditional classifiers, and their effectiveness as feature extraction mechanisms is investigated.

3.2.1 The data set

The data set used in the experiments was taken from Special Database 3 distributed on CD-ROM by the U.S. National Institute for Standards and Technology (NIST) (Wilson and Garris, 1992). Currently, this database is discontinued; it is now distributed together with Database 7 as Database 19. Of each digit, 2,500 samples were used. After randomising the order per class, the set was split into three parts: a training set of 1,000 images per class, a testing set of 1,000 images per class and a validation set of 500 images per class. The latter set was used in the ANN experiments for early stopping: if the error on the validation set increased for more than 50 cycles continuously, training was stopped and the ANN with minimum error on the validation set was used. This early stopping is known to prevent overtraining.

The binary digit images were then pre-processed in the following steps (de Ridder, 1996):

- shearing, to put the digit upright;
- scaling of line width, to normalise the number of pixels present in the image;
- segmenting the digit by finding the bounding box, preserving the aspect ratio;

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

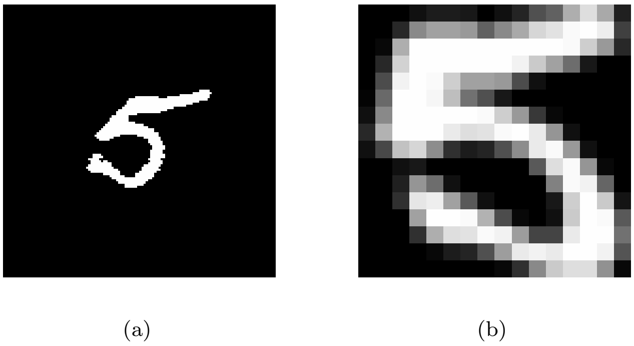


Figure 7: A digit before (a) and after (b) pre-processing.

- converting to floating point and scaling down to 16 × 16 using low-pass filtering and linear interpolation.

Figure 7 shows an example.

3.2.2 Experiments

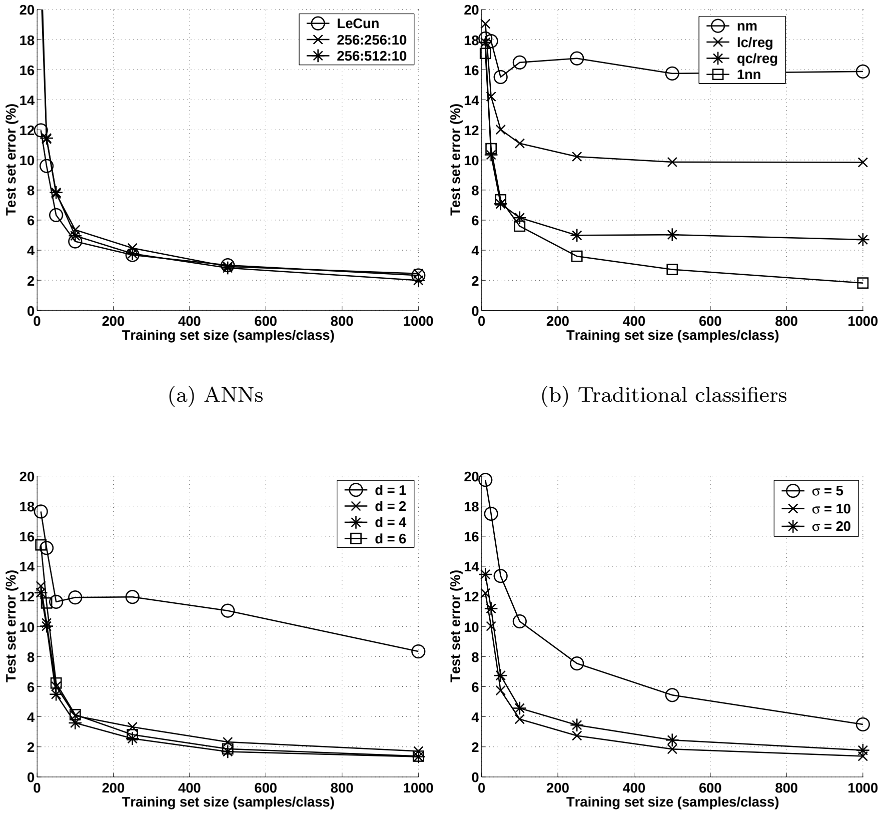
Instances of the LeCun ANN were trained on subsets of the training set containing 10, 25, 50, 100, 250, 500 and 1000 samples per class. Following (Le Cun et al., 1989a), weights and biases were initialised randomly using a uniform distribution in the range $\left[-\frac{2.4}{F}, \frac{2.4}{F}\right]$, where F was the total fan-in of a unit (i.e. the number of incoming weights). Back-propagation was used for training, with a learning rate of 0.5 and no momentum. Training targets were set to 0.9 for the output neuron coding the right digit class, and 0.1 for the other output neurons. After training, the testing set was used to find the error.

For comparison, a number of traditional classifiers were trained as well: the nearest mean linear classifier (which is denoted nm in the figures), the linear and quadratic *Bayes plug-in classifiers*[4] (lc and qc) and the 1-nearest neighbour classifier (1nn) (see e.g. (Devijver and Kittler, 1982; Fukunaga, 1990) for a discussion on these statistical pattern classifiers). For the Bayes plug-in classifiers, regularisation was used in calculating the 256 × 256 element covariance matrix $\mathbf{C}$:

$$\mathbf{C}' = (1 - r - s)\,\mathbf{C} + r\,\mathrm{diag}(\mathbf{C}) + \frac{s}{256}\,\mathrm{tr}(\mathbf{C})\mathbf{I} \tag{9}$$

where $\mathrm{diag}(\mathbf{C})$ is the matrix containing only the diagonal elements of $\mathbf{C}$, $\mathrm{tr}(\mathbf{C})$ is the trace of matrix $\mathbf{C}$, and using $r = s = 0.1$. Furthermore, two standard feed-forward ANNs were trained, containing one hidden layer of 256 and 512 hidden units, respectively. Finally, support vector classifiers (SVMs, (Vapnik, 1995)) were trained with polynomial kernels of various degrees and with radial basis kernels, for various values of σ.

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



(a) ANNs

(b) Traditional classifiers

(c) Polynomial SVMs

(d) Radial basis SVMs

Figure 8: Classification errors on the testing set, for (a) the LeCun and standard ANNs; (b) the nearest mean classifier (nm), linear and quadratic Bayes plug-in rules (lc, qc) and the 1-nearest neighbour classifier (1nn); (c) SVMs with a polynomial kernel function of degrees 1, 2, 4 and 6; (d) SVMs with a radial basis kernel function, $\sigma = 5$, 10, 20.

Results are shown in figures 8 (a)-(d). The LeCun ANN performs well, better than most traditional classifiers. For small sample sizes the LeCun ANN performs better than the standard feed-forward ANNs. The 1-nearest neighbour classifier and the standard feed-forward ANNs perform as well as the LeCun ANN or slightly better, as do the SVMs.

In general, classifiers performing better also have many more parameters and require more calculation in the testing phase. For example, when trained on 1,000 samples per class the LeCun ANN (2.3% error) performs slightly worse than the 1-nearest neighbour classifier (1.8% error) and the best performing SVMs (e.g. radial basis kernels, $\sigma = 10$: 1.4% error), but slightly better than the 256 hidden unit feed-forward ANN (2.4% error). The LeCun ANN has 64,660 parameters, requiring as many FLOPs (floating point operations) to

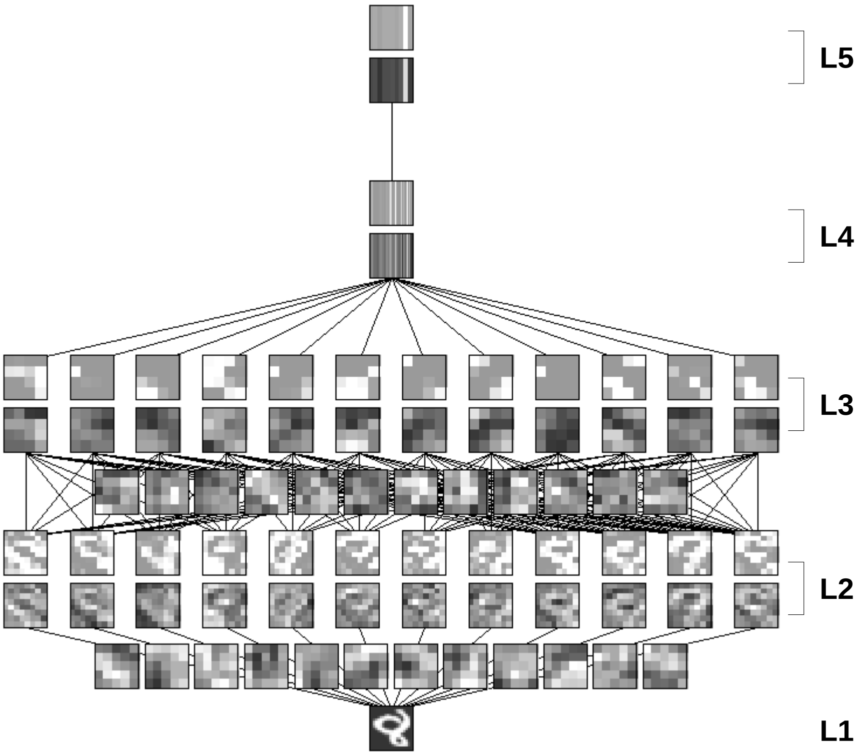


Figure 9: The LeCun ANN trained on the handwritten digit set, 1,000 samples/class. Note: for each map in the third layer, only the first set of weights (the first filter) is depicted. Bias is not shown in the figure. In this representation, the bottom layer is the input layer.

test one sample. In contrast, the 1-nearest neighbour rule, trained on 1,000 samples per class, requires 10,000 distance calculations in 256 dimensions, i.e. roughly 5,120,000 FLOPs. Similarly, the SVM uses a total of 8,076 support vectors in its 10 classifiers, requiring 4,134,912 FLOPs. However, the fully connected feed-forward ANN with 256 hidden units requires $256 \times 256 + 256 \times 10$ = 68,096 FLOPs, a number comparable to the LeCun ANN. In conclusion, the LeCun ANN seems to perform well given its limited number of parameters, but a standard feed-forward ANN performs equally well using the same amount of computation. This indicates that the restrictions placed on the shared weight ANNs are not quite necessary to obtain a good performance. It also contradicts the finding in (Solla and Le Cun, 1991) that the use of shared weights leads to better performance.

3.2.3 Feature extraction

In figure 9, an image of the LeCun ANN trained on the entire training set is shown. Some feature maps seem to perform operations similar to low-level

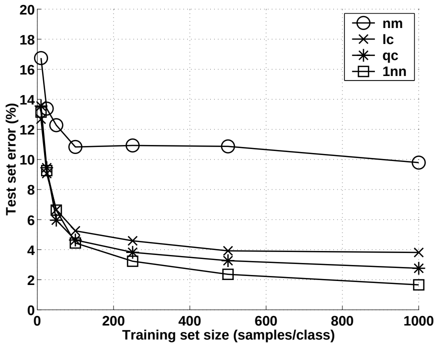


Figure 10: Performance of various classifiers trained on data sets extracted from the feature extraction parts of the LeCun ANN.

image processing operators such as edge detection. It is also noteworthy that the extracted features, the outputs of the last subsampling layer, are nearly binary (either high or low). However, visual inspection of the feature and subsampling masks in the trained shared weight ANNs in general does not give much insight into the features extracted. Gader et al. (Gader et al., 1995), in their work on automatic target recognition, inspected trained feature maps and claimed they were "... suggestive of a diagonal edge detector with a somewhat weak response" and "... of a strong horizontal edge detector with some ability to detect corners as well"; however, in our opinion these maps can be interpreted to perform any of a number of image processing primitives. In the next section, a number of simpler problems will be studied in order to learn about the feature extraction process in shared weight ANNs.

Here, another approach is taken to investigate whether the shared weight ANNs extract useful features: the features were used to train other classifiers. First, the architecture was cut halfway, after the last layer of subsampling maps, so that the first part could be viewed to perform feature extraction only. The original training, testing and validation sets were then mapped onto the new feature space by using each sample as input and finding the output of this first part of the ANN. This reduced the number of features to 192. In experiments, a number of classifiers were trained on this data set: the nearest mean linear classifier (nm), the Bayes plug-in linear and quadratic classifier (lc and qc) and the 1-nearest neighbour classifier (1nn). For the Bayes plug-in classifiers, the estimate of the covariance matrix was regularised in the same way as before (9), using $r = s = 0.1$. Figure 10 shows the results.

In all cases the 1-nearest neighbour classifier performed better than the classification parts of the ANNs themselves. The Bayes plug-in quadratic classifier performed nearly as well as the ANN (compare figure 8 (a) to figure 10. Interestingly, the LeCun ANN does not seem to use its 30 unit hidden layer to implement a highly nonlinear classifier, as the difference between this ANN's performance and that of the Bayes plug-in quadratic classifier is very small. Clearly, for all shared weight ANNs, most of the work is performed in the shared weight layers; after the feature extraction stage, a quadratic classifier suffices to give good classification performance.

Most traditional classifiers trained on the features extracted by the shared weight ANNs perform better than those trained on the original feature set (figure 8 (b)). This shows that the feature extraction process has been useful. In all cases, the 1-nearest neighbour classifier performs best, even better than on the original data set (1.7% vs. 1.8% error for 1,000 samples/class).

3.3 Discussion

A shared weight ANN architecture was implemented and applied to a handwritten digit recognition problem. Although some non-neural classifiers (such as the 1-nearest neighbour classifier and some support vector classifiers) perform better, they do so at a larger computational cost. However, standard feed-forward ANNs seem to perform as well as the shared weight ANNs and require the same amount of computation. The LeCun ANN results obtained are comparable to those found in the literature.

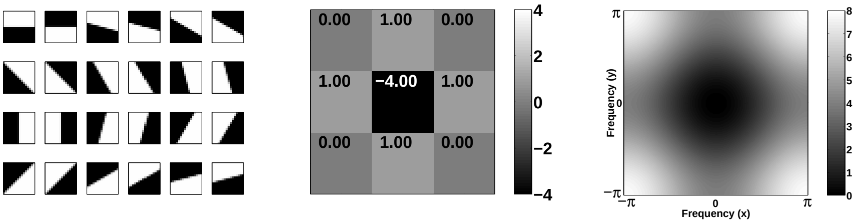
Unfortunately, it is very hard to judge visually what features the LeCun ANN extracts. Therefore, it was tested on its feature extraction behaviour, by using the output of the last subsampling map layer as a new data set in training a number of traditional classifiers. The LeCun ANN indeed acts well as a feature extractor, as these classifiers performed well; however, performance was in at best only marginally better than that of the original ANN.

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

4 Feature extraction in shared weight networks

This section investigates whether ANNs, in particular shared weight ANNs, are capable of extracting "good" features from training data. In the previous section the criterion for deciding whether features were good was whether traditional classifiers performed better on features extracted by ANNs. Here, the question is whether sense can be made of the extracted features by interpretation of the weight sets found. There is not much literature on this subject, as authors tend to research the way in which ANNs work from their own point of view, as tools to solve specific problems. Gorman and Sejnowski (Gorman and Sejnowski, 1988) inspect what kind of features are extracted in an ANN trained to recognise sonar profiles. Various other authors have inspected the use of ANNs as feature extraction and selection tools, e.g. (Egmont-Petersen et al., 1998b; Setiono and Liu, 1997), compared ANN performance to known image processing techniques (Ciesielski et al., 1992) or examined decision regions (Melnik and Pollack, 1998). Some effort has also been invested in extracting (symbolic) rules from trained ANNs (Setiono, 1997; Tickle et al., 1998) and in investigating the biological plausibility of ANNs (e.g. Verschure, 1996).

An important subject in the experiments presented in this section will be the influence of various design and training choices on the performance and feature



(a)

(b)

(c)

Figure 11: (a) The edge samples in the edge data set. (b) The Laplacian edge detector. (c) The magnitude of the frequency response of the Laplacian edge detector.

extraction capabilities of shared weight ANNs. The handwritten digit experiment showed that, although the LeCun ANN performed well, its complexity and that of the data set made visual inspection of a trained ANN impossible. For interpretation therefore it is necessary to bring both data set and ANN complexity down to a bare minimum. Of course, many simple problems can be created (de Ridder, 1996); here, two classification problems will be discussed: edge recognition and simple two-class handwritten digit recognition.

4.1 Edge recognition

The problem of edge recognition is treated here as a classification problem: the goal is to train an ANN to give high output for image samples containing edges and low output for samples containing uniform regions. This makes it different from edge *detection*, in which localisation of the edge in the sample is important as well. A data set was constructed by drawing edges at $0°, 15°, \ldots, 345°$ angles in a 256×256 pixel binary image. These images were rescaled to 16×16 pixels using bilinear interpolation. The pixel values were -1 for background and +1 for the foreground pixels; near the edges, intermediate values occurred due to the interpolation. In total, 24 edge images were created. An equal number of images just containing uniform regions of background (-1) or foreground ($+1$) pixels were then added, giving a total of 48 samples. Figure 11 (a) shows the edge samples in the data set.

The goal of this experiment is not to build an edge recogniser performing better than traditional methods; it is to study how an ANN performs edge recognition. Therefore, first a theoretically optimal ANN architecture and weight set will be derived, based on a traditional image processing approach. Next, starting from this architecture, a series of ANNs with an increasing number of restrictions will be trained, based on experimental observations. In each trained ANN, the weights will be inspected and compared to the calculated optimal set.

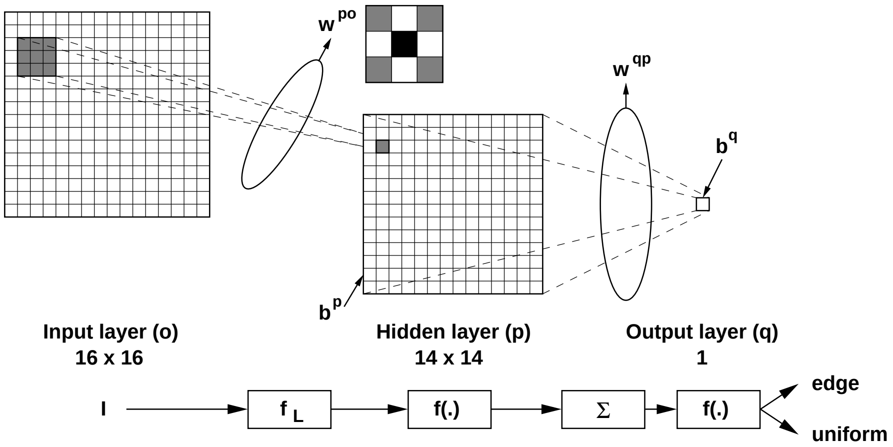


Figure 12: A sufficient ANN architecture for edge recognition. Weights and biases for hidden units are indicated by $\mathbf{w}^{po}$ and $\mathbf{b}^{p}$ respectively. These are the same for each unit. Each connection between the hidden layer and the output layer has the same weight $\mathbf{w}^{qp}$ and the output unit has a bias $\mathbf{b}^{q}$. Below the ANN, the image processing operation is shown: convolution with the Laplacian template f_L, pixel-wise application of the sigmoid $f(.)$, (weighted) summation and another application of the sigmoid.

4.1.1 A sufficient network architecture

To implement edge recognition in a shared weight ANN, it should consist of at least 3 layers (including the input layer). The input layer contains 16×16 units. The 14×14 unit hidden layer will be connected to the input layer through a 3×3 weight receptive field, which should function as an edge recognition template. The hidden layer should then, using bias, shift the high output of a detected edge into the nonlinear part of the transfer function, as a means of thresholding. Finally, a single output unit is needed to sum all outputs of the hidden layer and rescale to the desired training targets. The architecture described here is depicted in figure 12.

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

As the goal is to detect edges irrespective of their orientation, a rotation-invariant edge detector template is needed. The first order edge detectors known from image processing literature (Pratt, 1991; Young et al., 1998) cannot be combined into one linear rotation-invariant detector. However, the second order Laplacian edge detector can be. The continuous Laplacian,

$$f_L(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{10}$$

can be approximated by the discrete linear detector shown in figure 11 (b). It is a high-pass filter with a frequency response as shown in figure 11 (c). Note that in well-sampled images only frequencies between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ can be expected to occur, so the filters behaviour outside this range is not critical. The resulting image processing operation is shown below the ANN in figure 12.

Using the Laplacian template, it is possible to calculate an optimal set of weights for this ANN. Suppose the architecture just described is used, with double sigmoid transfer functions. Reasonable choices for the training targets then are $t = 0.5$ for samples containing an edge and $t = -0.5$ for samples containing uniform regions. Let the 3×3 weight matrix ($\mathbf{w}^{po}$ in figure 12) be set to the values specified by the Laplacian filter in figure 11 (b). Each element of the bias vector of the units in the hidden layer, $\mathbf{b}^p$, can be set to e.g. $b^p_{opt} = 1.0$.

Given these weight settings, optimal values for the remaining weights can be calculated. Note that since the DC component[5] of the Laplacian filter is zero, the input to the hidden units for samples containing uniform regions will be just the bias, 1.0. As there are 14×14 units in the hidden layer, each having an output of $f(1) \approx 0.4621$, the sum of all outputs O^p will be approximately $196 \cdot 0.4621 = 90.5750$. Here $f(\cdot)$ is the double sigmoid transfer function introduced earlier.

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

$$\begin{array}{|c|c|c|c|c|c|}\hline -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 2 & 2 & 2 & 2 \\ \hline -2 & -2 & -2 & -2 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \tag{11}$$

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

$$\begin{aligned} \sum_i O^p_i &= 14f(2 + b^p_{opt}) + 14f(-2 + b^p_{opt}) + 168f(b^p_{opt}) \\ &= 14f(3) + 14f(-1) + 168f(1) \\ &\approx 14 \cdot 0.9051 + 14 \cdot (-0.4621) + 168 \cdot 0.4621 = 82.0278 \end{aligned} \tag{12}$$

These values can be used to find the w^{qp}_{opt} and b^q_{opt} necessary to reach the targets. Using the inverse of the transfer function,

$$f(x) = \frac{2}{1 + e^{-x}} - 1 = a \Rightarrow f^{-1}(a) = \ln\left(\frac{1+a}{1-a}\right) = x, a \in \langle -1, 1 \rangle \tag{13}$$

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

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

$$\begin{aligned} \text{edge:} \quad t &= 0.5 \Rightarrow \mathbf{I}^q = 1.0986 \\ \text{uniform:} \quad t &= -0.5 \Rightarrow \mathbf{I}^q = -1.0986 \end{aligned} \tag{14}$$

This gives:

$$\begin{aligned} \text{edge:} \quad 82.0278\, w_{opt}^{qp} + b_{opt}^q &= 1.0986 \\ \text{uniform:} \quad 90.5750\, w_{opt}^{qp} + b_{opt}^q &= -1.0986 \end{aligned} \tag{15}$$

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

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

4.1.2 Training

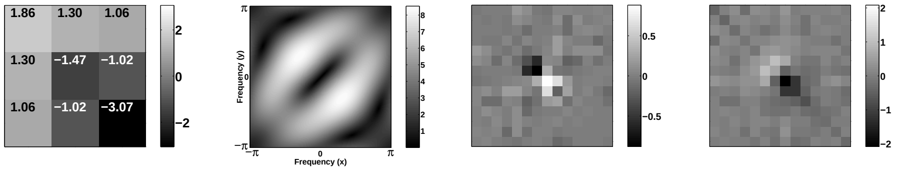
Starting from the sufficient architecture described above, a number of ANNs were trained on the edge data set. The weights and biases of each of these ANNs can be compared to the optimal set of parameters calculated above.

An important observation in all experiments was that as more restrictions were placed on the architecture, it became harder to train. Therefore, in all experiments the conjugate gradient descent (*CGD*, Shewchuk, 1994; Hertz et al., 1991; Press et al., 1992) training algorithm was used. This algorithm is less prone to finding local minima or diverging than back-propagation, as it uses a line minimisation technique to find the optimal step size in each iteration. The method has only one parameter, the number of iterations for which the directions should be kept conjugate to the previous ones. In all experiments, this was set to 10.

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

All weights and biases were initialised by setting them to a fixed value of 0.01, except where indicated otherwise[6]. Although one could argue that random

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



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

Figure 13: (a) The template and (b) the magnitude of its frequency response, (c) hidden layer bias weights and (c) weights between the hidden layer and output layer, as found in ANN$_1$.

initialisation might lead to better results, for interpretation purposes it is best to initialise the weights with small, equal values.

ANN$_1$*: *The sufficient architecture The first ANN used the shared weight mechanism to find $\mathbf{w}^{po}$. The biases of the hidden layer, $\mathbf{b}^p$, and the weights between hidden and output layer, $\mathbf{w}^{qp}$, were not shared. Note that this ANN already is restricted, as receptive fields are used for the hidden layer instead of full connectivity. However, interpreting weight sets of unrestricted, fully connected ANNs is quite hard due to the excessive number of weights – there would be a total of 50,569 weights and biases in such an ANN.

Training this first ANN did not present any problem; the MSE quickly dropped, to 1×10^{-7} after 200 training cycles. However, the template weight set found – shown in figures 13 (a) and (b) – does not correspond to a Laplacian filter, but rather to a directed edge detector. The detector does have a zero DC component. Noticeable is the information stored in the bias weights of the hidden layer $\mathbf{b}^p$ (figure 13 (c)) and the weights between the hidden layer and the output layer, $\mathbf{w}^{qp}$ (figure 13 (d)). Note that in figure 13 and other figures in this section, individual weight values are plotted as grey values. This facilitates interpretation of weight sets as feature detectors. Presentation using grey values is similar to the use of Hinton diagrams (Hinton et al., 1984).

Inspection showed how this ANN solved the problem. In figure 14, the different processing steps in ANN classification are shown in detail for three input samples (figure 14 (a)). First, the input sample is convolved with the template (figure 14 (b)). This gives pixels on and around edges high values, i.e. highly negative (-10.0) or highly positive (+10.0). After addition of the hidden layer bias (figure 14 (c)), these values dominate the output. In contrast, for uniform regions the bias itself is the only input of the hidden hidden layer units, with values approximately in the range $[-1, 1]$. The result of application of the transfer function (figure 14 (d)) is that edges are widened, i.e. they become bars of pixels with values +1.0 or -1.0. For uniform regions, the output contains just the two pixels diagonally opposite at the centre, with significantly smaller values.

The most important region in these outputs is the centre. Multiplying this region by the diagonal +/- weights in the centre and summing gives a very small input to the output unit (figure 14 (e)); in other words, the weights cancel the input. In contrast, as the diagonal -/+ pair of pixels obtained for uniform

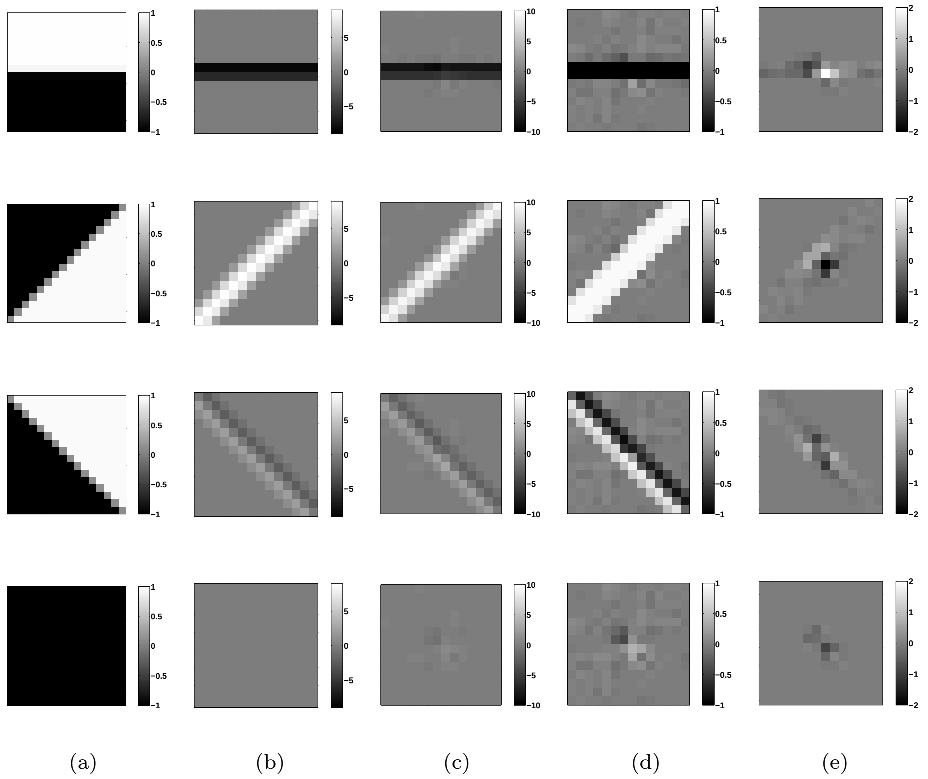
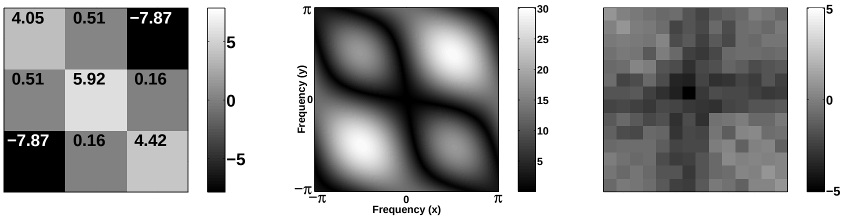


Figure 14: Stages in ANN_1 processing, for three different input samples: (a) the input sample; (b) the input convolved with the template; (c) the total input to the hidden layer, including bias; (d) the output of the hidden layer and (e) the output of the hidden layer multiplied by the weights between hidden and output layer.

samples is multiplied by a diagonal pair of weights of the opposite sign, the input to the output unit will be negative. Finally, the bias of the output unit (not shown) shifts the input in order to obtain the desired target values $t = 0.5$ and $t = -0.5$.

This analysis shows that the weight set found is quite different from the optimal one calculated in section 4.1.1. As all edges pass through the centre of the image, the edge detector need not be translation-invariant: information on where edges occur is coded in both the hidden layer bias and the weights between the hidden layer and the output layer.

ANN*$_2$*: Sharing more weights To prevent the ANN from coding place-specific information in biases and weights, the architecture will have to be simplified further. As a restriction, in the next ANN architecture the weights between the hidden layer and output layer were shared. That is, there was one single weight shared among all 196 connections between the hidden units and the output unit. Training took more time, but converged to a 1×10^{-6} MSE



(a) (b) (c)

Figure 15: (a) The template, (b) the magnitude of its frequency response and (b) hidden layer bias weights as found in ANN$_2$.

after 2,400 cycles. Still, the network does not find a Laplacian; however, the template found (figure 15 (a) and (b)) has a more clear function than the one found before. It is a strong detector for edges with slope $-45°$, and a weak detector for edges with slope $45°$.

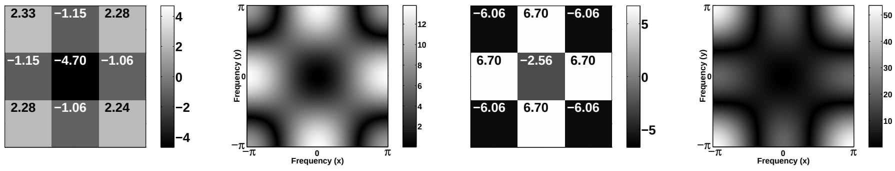
In the bias weights of the hidden layer (figure 15 (c)), place-specific information is now stored for edges which are not amplified well by this detector. Bias weight values are also significantly higher than before (an average of -1.2144). This allows the ANN to use the transfer function as a threshold operation, by scaling large positive pixel values differently from large negative pixel values. In conclusion, responsibility for edge recognition is now shared between the template and the bias weights of the hidden layer.

ANN*$_3$*: Sharing bias As the biases of hidden layer units are still used for storing place-dependent information, in the next architecture these biases were shared too[7]. Training became even harder; the ANN would not converge using the initialisation used before, so weights were initialised to a fixed value of 0.1. After 1,000 episodes, the MSE reached 8×10^{-4}, just slightly higher than the minimal error possible (at 3×10^{-4}, larger than zero due to the interpolation used in scaling the edge samples). The template found is shown in figures 16 (a) and (b).

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

ANN*$_4$*: Enforcing symmetry In the last ANN, the prior knowledge that the goal is to obtain a rotation-invariant filter was used as well, by sharing

[7] Sharing biases would have required a major rewrite of the simulation package used, SPRLIB/ANNLIB (Hoekstra et al., 1996). Therefore, biases were shared by replacing all biases by their average after each training cycle.



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

Figure 16: (a) The template found in ANN_3 and (b) the magnitude of its frequency response. (c) The template found in ANN_4 and (d) the magnitude of its frequency response.

weights in the filter itself. The *mask* used for this purpose was:

$$\begin{array}{|c|c|c|}\hline \text{A} & \text{B} & \text{A} \\ \hline \text{B} & \text{C} & \text{B} \\ \hline \text{A} & \text{B} & \text{A} \\ \hline \end{array} \tag{16}$$

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

Training was again more cumbersome, but after initialising weights with a fixed value of 0.1 the ANN converged after 1,000 episodes to an MSE of 3×10^{-4}. The filter found is shown in figures 16 (c) and (d). Finally, a solution similar to the optimal one was found: its frequency response is like that of the Laplacian in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and the weights are symmetric.

4.1.3 Discussion

The experiments described in this section show that ANNs can be used as edge detectors. However, the presence of receptive fields in the architecture in itself does not guarantee that shift-invariant feature detectors will be found, as claimed by some (Le Cun et al., 1990; Le Cun et al., 1989b; Viennet, 1993). Also, the mere fact that performance is good (i.e., the MSE is low) does not imply that such a feature extraction process is used. An important observation in ANN_1 and ANN_2 was that the ANN will use weights and biases in later layers to store place-dependent information. In such a network, where edge positions are stored, in principle any template will suffice. Obviously, this makes interpretation of these templates dubious: different observers may find the ANN has learned different templates. One reason for the ease with which ANNs store place-dependent information might be the relative simplicity of the dataset: the fact that edges all passed through the centre of the image makes this possible. Therefore, in the next section similar ANNs will be trained on a real-world dataset.

When the ANNs were further restricted by sharing biases and other weights (ANN_3), convergence became a problem. The explanation for this is that the

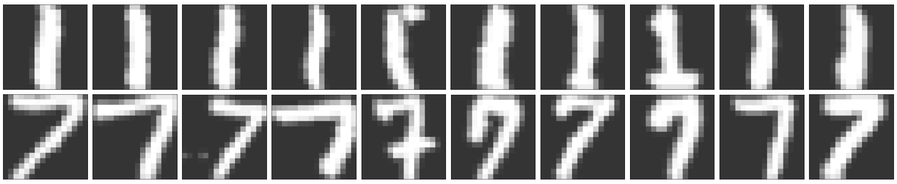


Figure 17: The two-class handwritten digit data set.

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

The last ANN, ANN_4, was able to find an edge detector very similar to the Laplacian. However, this architecture was restricted to such an extent that it can hardly be seen as representative for practical application of ANNs. This indicates there is a trade-off between complexity and the extent to which experiments are true-to-life on the one hand, and the possibility of interpretation on the other. This effect might be referred to as a kind of ANN *interpretability trade-off*[8]. If an unrestricted ANN is trained on a real-world data set, the setup most closely resembles the application of ANNs in everyday practice. However, the subtleties of the data set and the many degrees of freedom in the ANN prevent gaining a deeper insight into the operation of the ANN. On the other side, once an ANN is restrained, e.g. by sharing or removing weights, lowering the number of degrees of freedom or constructing architectures only specifically applicable to the problem at hand, the situation is no longer a typical one. The ANN may even become too constrained to learn the task at hand. The same holds for editing a data set to influence its statistics or to enhance more preferable features with regard to ANN training, which will be discussed in section 6.

4.2 Two-class handwritten digit classification

To construct a more real-life dataset while still maintaining the expectation that weights can be interpreted, experiments with a small NIST subset were performed. This subset consisted of 10 samples each of the classes "1" and "7", shown in figure 17. The 16×16 pixel values were scaled linearly between -1.0 (background) and 1.0 (foreground). Training targets were set to $t = 0.5$ for class "1" and $t = -0.5$ for class "7".

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

[8] Note that this is not precisely the same issue as addressed by the bias-variance trade-off (see page 8), which is concerned with the relation between model complexity and error. The concern here is with the specificity of the model with respect to interpretation which, in principle, is unrelated to complexity: making a model more specific need not introduce a bias.

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

The starting point therefore will be the ANN used for edge recognition, shown in figure 12. However, three different architectures will be used. The first has a 3×3 pixel receptive field and $14\times14=196$ units in the hidden layer, the second contains a 5×5 pixel receptive field and $12\times12=144$ hidden units and the last contains a 7×7 pixel receptive field and $10\times10=100$ hidden units. As for this data set it is to be expected that using more than one feature map will increase performance, architectures using two feature maps were trained as well. In this case, the number of hidden units doubles.

4.2.1 Training

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

ANN*$_1$*: Unrestricted The first ANNs were identical to the one shown in figure 12, except for the fact that three different ANNs were trained with 3×3 ($\text{ANN}_1^{3\times3}$), 5×5 ($\text{ANN}_1^{5\times5}$) and 7×7 ($\text{ANN}_1^{7\times7}$) pixel receptive fields, respectively. These ANNs quickly converged to a nearly zero MSE: after 250 training cycles, the MSE was in the order of 1×10^{-10}. The feature detectors found, shown in figure 18 (a), are not very clear however. The frequency responses (figure 18 (b)) give more information. The filters most closely resemble horizontal edge detectors: note the basic shape returning for the three sizes of feature detector.

As was the case in the edge recognition ANNs, the weights between the hidden layer and the output unit have been used to store positions of the digits. Figure 18 (c) illustrates this. Positive weights indicate pixel positions where typically only class “7” samples have high values; negative weights indicate positions where class “1” is present. Although noisy, these same basic shapes are present for each size of the receptive field.

In contrast to what was found for the edge recognition ANNs, the bias weights in the second layer were not used heavily. Bias values fell roughly in the range $\left[-2\times10^{-2}, 2\times10^{-2}\right]$, i.e. negligible in comparison to feature detector weight values.

ANN*$_2$*: Fully restricted In the next architecture, the number of weights was restricted by sharing weights between hidden layer and output layer and by sharing the bias weights in the second layer (i.e., the basic architecture was the same as ANN_3 for edge recognition, on page 30). As a consequence, there were far fewer parameters left in the ANNs: the number of weights in the feature detector plus two biases and one weight between hidden and output layer.

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

2000 cycles for the 5×5 pixel receptive field ANN and 1450 cycles for the 7×7 pixel receptive field ANN.

The feature detectors found are shown in figure 19. Note that since the 3×3 receptive field ANN did not converge, the resulting filter cannot be interpreted. Since the weights between hidden layer and output layer can no longer be used, the feature detectors of the other two look rather different. The 5×5 pixel feature detector is the most pronounced: it is a detector of 3-pixel wide bars with a slope of 45°. Evidence for this can also be found by inspecting the output of the hidden layer for various inputs, as shown in figure 20. In the location of the stem of the "7"s, output values are much higher than those in the location of the stem of the "1"s. Finally, the function of the 7×7 pixel feature detector is unclear.

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

ANN*$_3$*: Two feature maps Although the frequency response of the 5×5 pixel feature detector is clearer than the others, the filter itself is still noisy, i.e. neighbouring weights have quite different values. There is no clear global feature (within a 5×5 pixel region) that corresponds to this detector. The reason for this might be that in fact several features are detected (either amplified or attenuated) using this one set of weights. Therefore, ANN$_3$ contained two feature maps instead of one. In all other respects, the ANN was the same as ANN$_2^{5\times5}$, as shown in figure 21.

If this ANN is initialised using a fixed value, the two feature detectors will always remain identical, as each corresponding weight in the two detectors is equally responsible for the error the ANN makes. Therefore, random initialisation is necessary. This frustrates interpretation, as different initialisations will lead to different final weight sets. To illustrate this, four ANNs were trained in which weights were initialised using values drawn from a uniform distribution with range $[-0.01, 0.01]$. Figure 22 shows four resulting template pairs. The feature detector found before in ANN$_2^{5\times5}$ (figure 19) often returns as one of the two feature maps. The other feature detector however shows far more variation. The instantiation in the second row of figure 22 (b) looks like the horizontal edge detector found in ANN$_1$ (figures 18 (a), (b)), especially when looking at its frequency response (in the fourth column). However, in other ANNs this shape does not return. The first and fourth ANN indicate that actually multiple feature detectors may be distributed over the two feature maps.

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

4.2.2 Decorrelating conjugate gradient descent

The last experiment on the NIST subset showed that interpretation of ANNs with multiple feature detectors is difficult. The main causes are the random weight initialisation required and a tendency of the ANNs to distribute features to be detected in a non-obvious way over receptive fields. To address the latter problem, hidden units learning identical functions, a modular approach has been suggested (Jacobs et al., 1991). However, this is not applicable in cases in which there is no clear decomposition of a task's input space into several domains.

To allow fixed value weight initialisation and still obtain succinct feature detectors, a new training algorithm will be proposed. The algorithm is based on CGD, but has as a soft constraint the minimisation of the squared covariance between receptive fields. In this way, the symmetry between feature detectors due to fixed value initialisation can be broken, and receptive field weight sets are forced to become orthogonal while still minimising the ANN's MSE.

Decorrelation Note that, in trained ANNs, weight sets belonging to different receptive fields need not be exactly the same for the feature maps to perform the same function. This is because weights are interdependent, as was already noted in section 4.1.1. As an example, consider the weight vectors $\mathbf{w}^{po,A}$ and $\mathbf{w}^{po,B}$ (from here on, $\mathbf{w}^A$ and $\mathbf{w}^B$) in ANN$_3$ (figure 21). As long as $\mathbf{w}^A = c_1 \mathbf{w}^B + c_2$, biases in the hidden and output layer and the weights between these layers can correct the differences between the two weight sets, and their functionality can be approximately identical[9]. The conclusion is that to compare weight sets, one has to look at their correlation.

Suppose that for a certain layer in an ANN (as in figure 21) there are two incoming weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$, both with $K > 2$ elements and $\mathrm{var}(\mathbf{w}^A) > 0$ and $\mathrm{var}(\mathbf{w}^B) > 0$. The correlation coefficient C between these vectors can be calculated as:

$$C(\mathbf{w}^A, \mathbf{w}^B) = \frac{\mathrm{cov}(\mathbf{w}^A, \mathbf{w}^B)}{\sqrt{\mathrm{var}(\mathbf{w}^A)\mathrm{var}(\mathbf{w}^B)}} \tag{17}$$

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

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

$$\begin{aligned} \frac{\partial \mathrm{cov}(\mathbf{w}^A, \mathbf{w}^B)^2}{\partial \mathbf{w}_i^A} &= \frac{\partial}{\partial \mathbf{w}_i^A} \left(\frac{1}{K} \textstyle\sum_{k=1}^{K} (\mathbf{w}_k^A - \overline{\mathbf{w}}^A)(\mathbf{w}_k^B - \overline{\mathbf{w}}^B) \right)^2 \\ &= \frac{2}{K} \mathrm{cov}(\mathbf{w}^A, \mathbf{w}^B)(\mathbf{w}_i^B - \overline{\mathbf{w}}^B) \end{aligned} \tag{18}$$

This derivative can then be used in combination with the derivative of the MSE w.r.t. the weights to obtain a training algorithm minimising both MSE and squared covariance (and therefore squared correlation, because the variance of

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

the weight vectors will remain bounded since the ANN still has to minimise the MSE).

Correlation has been used before in neural network training. In the cascade correlation algorithm (Fahlman and Lebiere, 1990), it is used as a tool to find an optimal number of hidden units by taking the correlation between a hidden unit's output and the error criterion into account. However, it has not yet been applied on weights themselves, to force hidden units to learn different functions during training.

A decorrelating training algorithm Squared covariance minimisation was incorporated into the CGD method used before. Basically, CGD iteratively applies three stages:

- calculation of the derivative of the error w.r.t. the weights, $\mathbf{d}^E = \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w})$;
- deriving a direction $\mathbf{h}$ from $\mathbf{d}^E$ which is conjugate to previously taken directions;
- a line minimisation of E from $\mathbf{w}$ along $\mathbf{h}$ to find a new weight vector $\mathbf{w}'$.

The squared covariance term was integrated into the derivative of the error function as an additive criterion, as in weight regularisation (Bishop, 1995). A problem is how the added term should be weighted (cf. choosing the regularisation parameter). The MSE can start very high but usually drops rapidly. The squared covariance part also falls in the range $[0, \infty\rangle$, but it may well be the case that it cannot be completely brought down to zero, or only at a significant cost to the error. The latter effect should be avoided: the main training goal is to reach an optimal solution in the MSE sense. Therefore, the covariance information is used in the derivative function *only*, not in the line minimisation. The squared covariance gradient, $\mathbf{d}^{\text{cov}}$, is normalised to the length of the ordinary gradient $\mathbf{d}^E$ (just its direction is used) and weighed with a factor λ; i.e. $\mathbf{d} = \mathbf{d}^E + \lambda \frac{||\mathbf{d}^E||}{||\mathbf{d}^{\text{cov}}||} \mathbf{d}^{\text{cov}}$, where $\mathbf{d}^{\text{cov}} = \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \frac{\partial}{\partial \mathbf{w}} \text{cov}(\mathbf{w}^k(0), \mathbf{w}^l(0))^2$.

Note that the derivative of the squared covariance is only calculated once for each pair of weight sets and attributed to only one of the weight sets. This allows one weight set to learn a globally optimal function, while the second set is trained to both lower the error and avoid covariance with the first set. It also allows initialisation with fixed values, since the asymmetrical contribution of the squared covariance term provides a symmetry breaking mechanism (which can even improve performance in some classification problems, see de Ridder et al., 1999). However, the outcome of the DCGD training process is still dependent on the choice of a number of parameters. DCGD even introduces a new one (the weight factor λ). If the parameters are chosen poorly, one will still not obtain understandable feature detectors. This is a problem of ANNs in general, which cannot be solved easily: a certain amount of operator skill in training ANNs is a prerequisite for obtaining good results. Furthermore, experiments with DCGD are reproducable due to the possibility of weight initialisation with fixed values.

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

4.2.3 Training ANN$_3$ using DCGD

ANN$_3$ was trained using DCGD. Weights and biases were initialised to a fixed value of 0.01 (i.e. $\mu = 0.01$, $\sigma = 0.0$) and $N = 10$ directions were kept conjugate at a time. The only parameter varied was the weighting factor of the squared covariance gradient, λ, which was set to 0.5, 1, 2 and 5. Training converged but was slow. The MSE eventually reached the values obtained using CGD (1.0×10^{-6}, cf. section 4.2.1); however, DCGD training was stopped when the MSE reached about 1.0×10^{-5}, after about 500-1000 cycles, to prevent overtraining. In all cases, classification was perfect.

Figure 23 shows the feature detectors found in ANN$_3$ trained using DCGD. Squared correlations C^2 between them are very small, showing that the minimisation was succesful (the squared covariance was, in all cases, nearly 0). For $\lambda = 1$ and $\lambda = 2$, the feature detectors are more clear than those found using standard CGD, in section 44.24.2.1 Their frequency responses resemble those of the feature detectors shown in figure 22 (b) and, due to the fixed weight initialisation, are guaranteed to be found when training is repeated. However, λ should be chosen with some care; if it is too small ($\lambda = 0.5$), the squared covariance term will have too little effect; if it is too large ($\lambda = 5$), minimisation of the squared covariance term becomes too important and the original functionality of the network is no longer clearly visible.

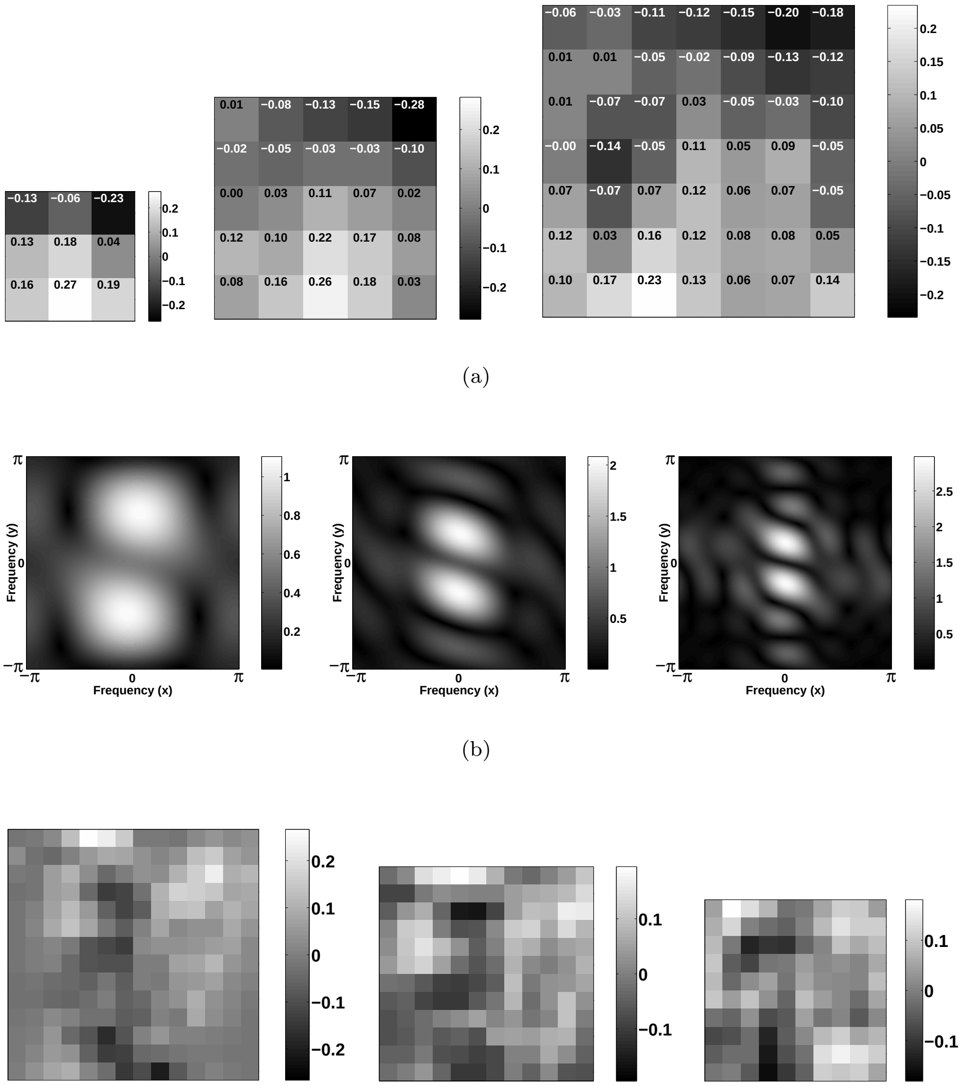
The features detected seem to be diagonal bars, as seen before, and horizontal edges. This is confirmed by inspecting the output of the two feature maps in ANN$_3$ trained with DCGD, $\lambda = 1$, for a number of input samples (see figure 24). For samples of class "1", these outputs are lower than for class "7", i.e. features specific for digits of class "7" have been found. Furthermore, the first feature detector clearly enhances the stem of "7" digits, whereas the second detector amplifies the top stroke.

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

4.3 Discussion

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

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



−0.13	−0.06	−0.23
0.13	0.18	0.04
0.16	0.27	0.19

0.01	−0.08	−0.13	−0.15	−0.28
−0.02	−0.05	−0.03	−0.03	−0.10
0.00	0.03	0.11	0.07	0.02
0.12	0.10	0.22	0.17	0.08
0.08	0.16	0.26	0.18	0.03

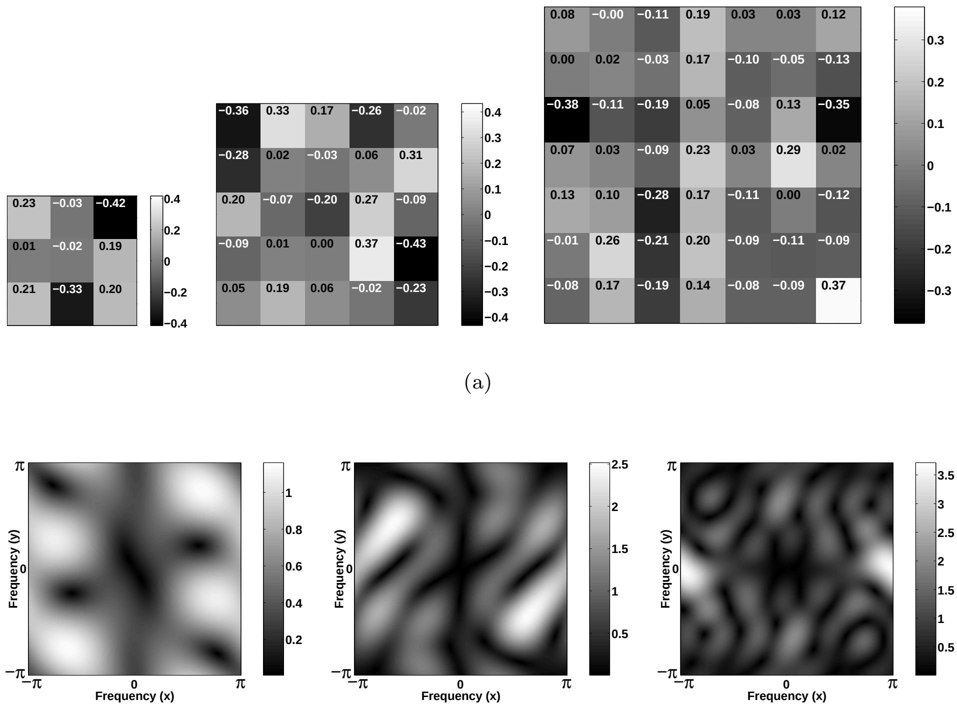
−0.06	−0.03	−0.11	−0.12	−0.15	−0.20	−0.18
0.01	0.01	−0.05	−0.02	−0.09	−0.13	−0.12
0.01	−0.07	−0.07	0.03	−0.05	−0.03	−0.10
−0.00	−0.14	−0.05	0.11	0.05	0.09	−0.05
0.07	−0.07	0.07	0.12	0.06	0.07	−0.05
0.12	0.03	0.16	0.12	0.08	0.08	0.05
0.10	0.17	0.23	0.13	0.06	0.07	0.14

(a)

(b)

(c)

Figure 18: (a) Feature detectors found in the receptive fields of $\text{ANN}_1^{3\times3}$, $\text{ANN}_1^{5\times5}$ and $\text{ANN}_1^{7\times7}$. (b) The corresponding frequency response magnitudes. (c) Weights between hidden layer and output layer.



(a)

(b)

Figure 19: (a) The feature detectors found in the receptive fields of $\text{ANN}_2^{3\times3}$, $\text{ANN}_2^{5\times5}$ and $\text{ANN}_2^{7\times7}$. (b) The corresponding frequency response magnitudes.

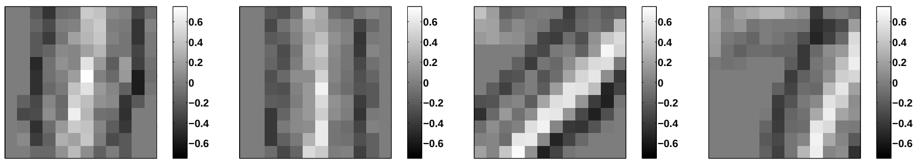


Figure 20: The output of the hidden layer of $\text{ANN}_2^{5\times5}$, for two samples of class "1" (left) and two samples of class "7" (right).

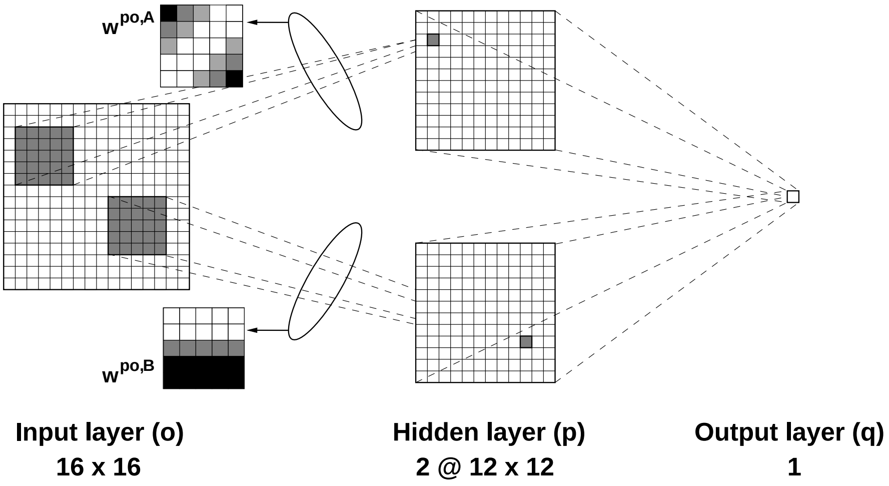
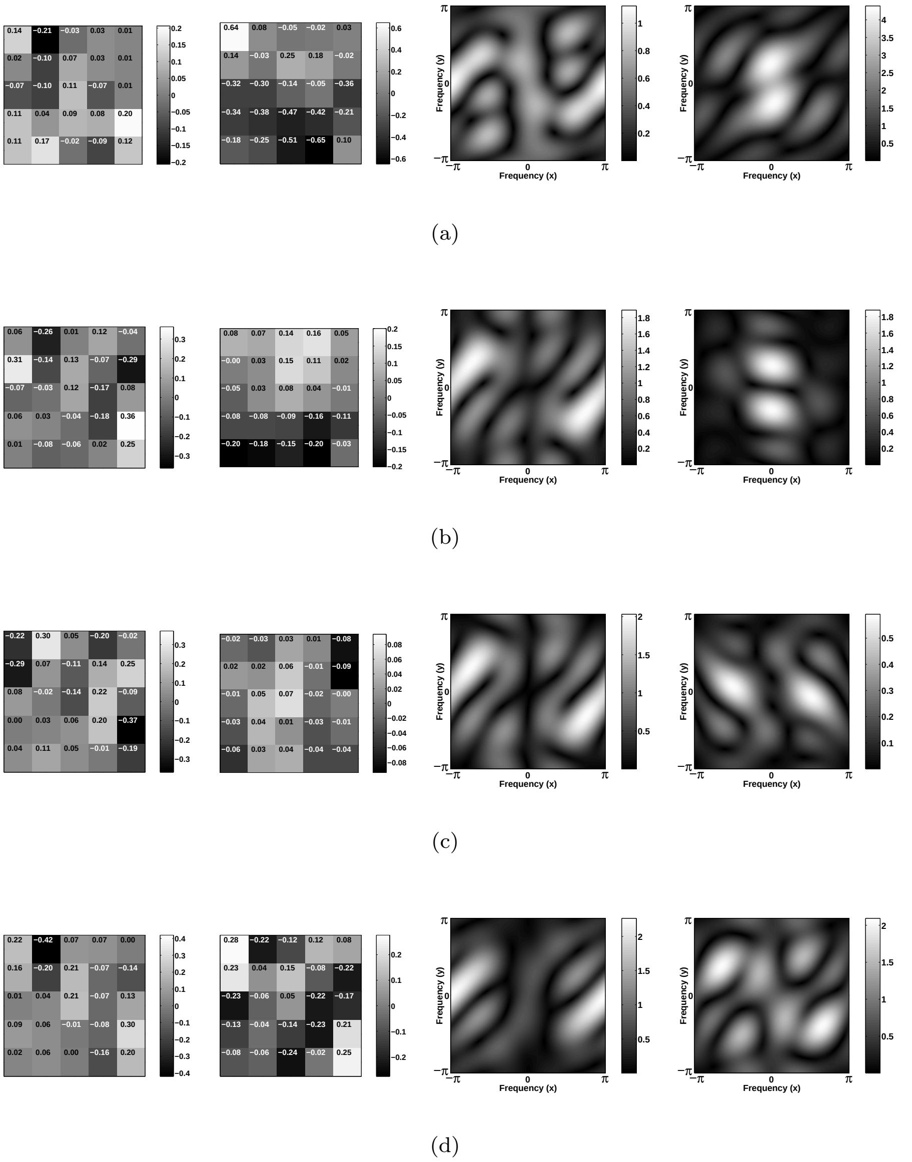


Figure 21: ANN_3, with two 5×5 pixel feature detectors. Biases and weights between the hidden layer and output layer have not been indicated.



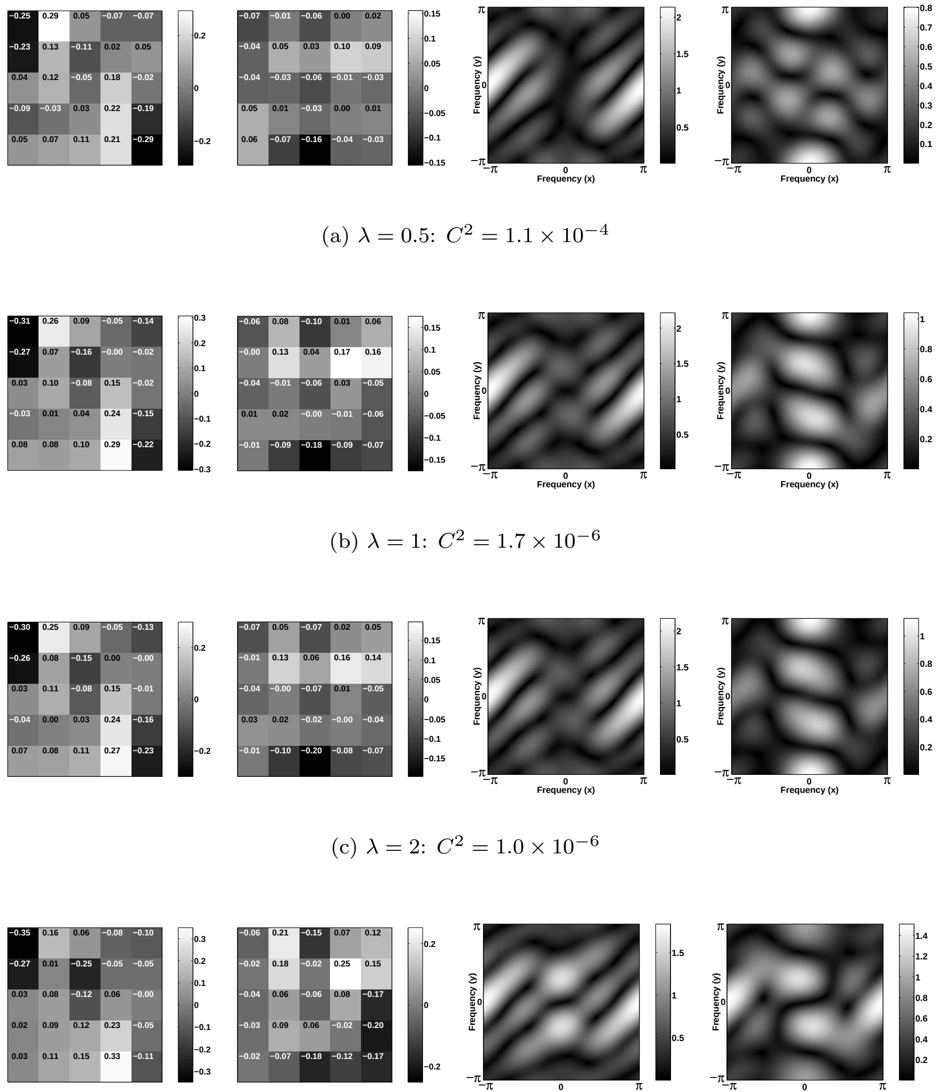
(a)

(b)

(c)

(d)

Figure 22: Feature detector pairs found in ANN_3, for four different random weight initialisations ((a)-(d)).



−0.25	0.29	0.05	−0.07	−0.07
−0.23	0.13	−0.11	0.02	0.05
0.04	0.12	−0.05	0.18	−0.02
−0.09	−0.03	0.03	0.22	−0.19
0.05	0.07	0.11	0.21	−0.29

−0.07	−0.01	−0.06	0.00	0.02
−0.04	0.05	0.03	0.10	0.09
−0.04	−0.03	−0.06	−0.01	−0.03
0.05	0.01	−0.03	0.00	0.01
0.06	−0.07	−0.16	−0.04	−0.03

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

−0.31	0.26	0.09	−0.05	−0.14
−0.27	0.07	−0.16	−0.00	−0.02
0.03	0.10	−0.08	0.15	−0.02
−0.03	0.01	0.04	0.24	−0.15
0.08	0.08	0.10	0.29	−0.22

−0.06	0.08	−0.10	0.01	0.06
−0.00	0.13	0.04	0.17	0.16
−0.04	−0.01	−0.06	0.03	−0.05
0.01	0.02	−0.00	−0.01	−0.06
−0.01	−0.09	−0.18	−0.09	−0.07

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

−0.30	0.25	0.09	−0.05	−0.13
−0.26	0.08	−0.15	0.00	−0.00
0.03	0.11	−0.08	0.15	−0.01
−0.04	0.00	0.03	0.24	−0.16
0.07	0.08	0.11	0.27	−0.23

−0.07	0.05	−0.07	0.02	0.05
−0.01	0.13	0.06	0.16	0.14
−0.04	−0.00	−0.07	0.01	−0.05
0.03	0.02	−0.02	−0.00	−0.04
−0.01	−0.10	−0.20	−0.08	−0.07

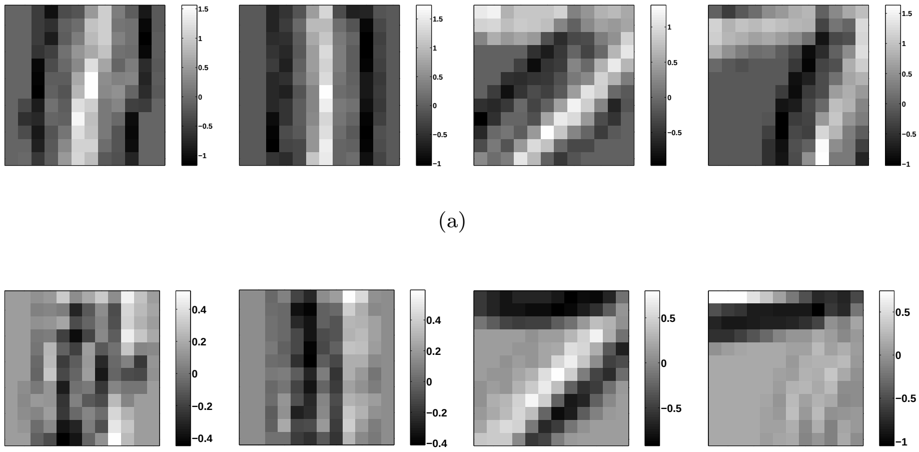
(c) $\lambda = 2$: $C^2 = 1.0 \times 10^{-6}$

−0.35	0.16	0.06	−0.08	−0.10
−0.27	0.01	−0.25	−0.05	−0.05
0.03	0.08	−0.12	0.06	−0.00
0.02	0.09	0.12	0.23	−0.05
0.03	0.11	0.15	0.33	−0.11

−0.06	0.21	−0.15	0.07	0.12
−0.02	0.18	−0.02	0.25	0.15
−0.04	0.06	−0.06	0.08	−0.17
−0.03	0.09	0.06	−0.02	−0.20
−0.02	−0.07	−0.18	−0.12	−0.17

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

Figure 23: Feature detector pairs found in ANN_3 using DCGD with various values of weight factor λ ((a)-(d)). C^2 is the squared correlation between the feature detectors after training.



(a)

(b)

Figure 24: The output of (a) the first and (b) the second feature map of ANN$_3$ trained with DCGD ($\lambda = 1$), for two samples of class "1" (left) and two samples of class "7" (right). The samples used were, for both digits, the leftmost two in figure 17.

5 Regression networks for image restoration

This section will study whether standard regression feed-forward ANNs can be applied successfully to a nonlinear image filtering problem. If so, what are the prerequisites for obtaining a well-functioning ANN? A second question (as in the previous section) is whether these ANNs correspond to classic image processing approaches to solve such a task. Note that again the goal here is not to simply apply ANNs to an image processing problem, nor to construct an ANN that will perform better at it than existing techniques. Instead, the question is to what extent ANNs can learn the nonlinearities needed in some image processing applications.

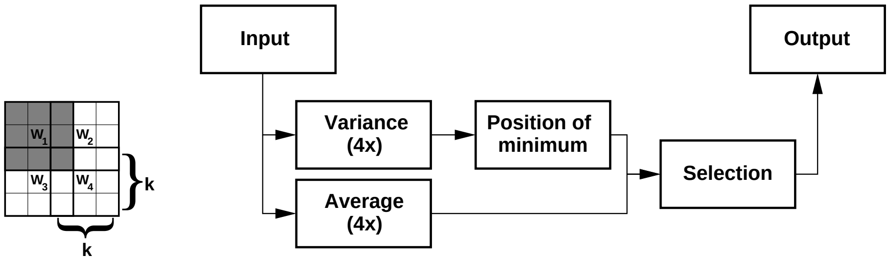
To investigate the possibilities of using feed-forward ANNs and the problems one might encounter, the research concentrates on a single example of a nonlinear filter: the Kuwahara filter for edge-preserving smoothing (Kuwahara et al., 1976). Since this filter is well-understood and the training goal is exactly known, it is possible to investigate to what extent ANNs are capable of performing this task. The Kuwahara filter also is an excellent object for this study because of its inherent modular structure, which allows splitting the problem into smaller parts. This is known to be an advantage in learning (Anand et al., 1995) and gives the opportunity to study subproblems in isolation. (Pugmire et al., 1998) looked at the application of ANNs to edge detection and found that structuring learning in this way can improve performance; however, they did not investigate the precise role this structuring plays.

ANNs have previously been used as image filters, as discussed in section 22.32.3.1 However, the conclusion was that in many applications the ANNs were non-adaptive. Furthermore, where ANNs *were* adaptive, a lot of prior knowledge of the problem to be solved was incorporated in the ANN's architectures. Therefore, in this section a number of modular ANNs will be constructed and trained to emulate the Kuwahara filter, incorporating prior knowledge in various degrees. Their performance will be compared to standard feed-forward ANNs. Based on results obtained in these experiments, in section 6 it is shown that several key factors influence ANN behaviour in this kind of task.

5.1 Kuwahara filtering

The Kuwahara filter is used to smooth an image while preserving the edges (Kuwahara et al., 1976). Figure 25 (a) illustrates its operation. The input of the filter is a $(2k-1)\times(2k-1)$ pixel neighbourhood around the central pixel. This neighbourhood is divided into 4 overlapping subwindows W_i, $i = 1, 2, 3, 4$, each of size $k \times k$ pixels. For each of these subwindows, the average μ_i and the variance σ_i^2 of the k^2 grey values is calculated. The output of the filter is then found as the average μ_m of the subwindow W_m having the smallest grey value variance ($m = \arg\min_i \sigma_i^2$). This operation can be applied in a scan-wise manner to filter an entire image. For an example of the effect of the filter, see figure 26.

The filter is nonlinear. As the selection of the subwindow based on the variances is data-driven, edges are not blurred as in normal uniform filtering. Since a straight edge will always lie in at most three subwindows, there will always be at least one subwindow that does not contain an edge and therefore has low variance. For neighbouring pixels in edge regions, different subwindows will be



(a) (b)

Figure 25: (a) The Kuwahara filter: $k \times k$ subwindows in a $(2k-1) \times (2k-1)$ window; here $k = 3$. (b) Kuwahara filter operation as a sequence of operations.

selected (due to the minimum operation), resulting in sudden large differences in grey value. Typically, application of the Kuwahara filter to natural images will result in images which have an artificial look but which may be more easily segmented or interpreted.

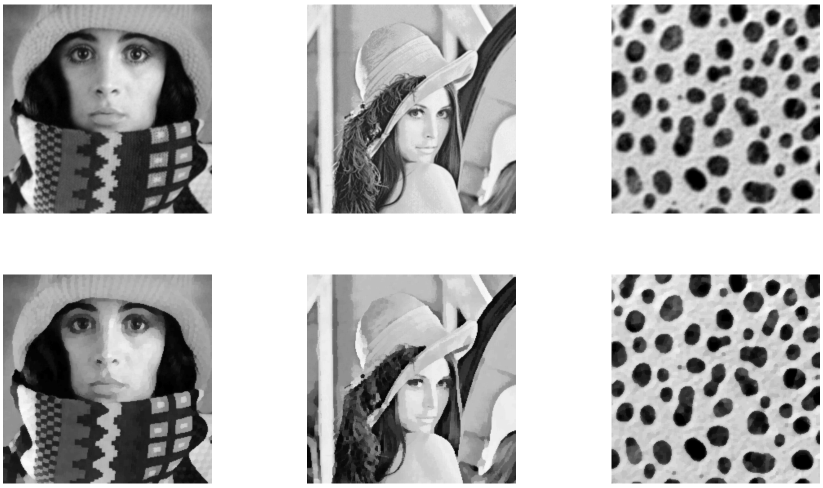
This filter was selected for this research since:

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

- It is *modular* (figure 25 (b) illustrates this). This means the operation can be split into subtasks which can perhaps be learned more easily than the whole task at once. It will be interesting to see whether an ANN will need this modularity and complexity in order to approximate the filter's operation. Also, it offers the opportunity to study an ANN's operation in terms of the individual modules.

5.2 Architectures and experiments

In the previous section, it was shown that when studying ANN properties, such as internal operation (which functions are performed by which hidden units) or generalisation capabilities, one often encounters a phenomenon which could be described as an ANN interpretability trade-off (section 4.1.3). This trade-off, controlled by restricting the architecture of an ANN, is between the possibility of understanding how a trained ANN operates and the degree to which the experiment is still true-to-life. In order to cover the spectrum of possibilities, a number of modular ANNs with varying degrees of freedom was constructed. The layout of such a modular ANN is shown in figure 27. Of the modular ANNs, four types were created, $\mathrm{ANN}^{\mathrm{M}}_1 \ldots \mathrm{ANN}^{\mathrm{M}}_4$. These are discussed below in descending order of artificiality; i.e., the first is completely hand-designed, with every weight set to an optimal value, while the last consists of only standard feed-forward



(a) Image A

(b) Image B

(c) Image C

Figure 26: Images used for (a) training and (b)-(c) testing purposes. The top images are the originals; the bottom images are the Kuwahara filtered versions (for image A, the training target). For presentation purposes, the contrast of the images has been stretched (Young et al., 1998).

modules.

5.2.1 Modular networks

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

- For $\mathbf{ANN_1^M}$, the modules were hand-designed for the tasks they are to perform. In some cases, this meant using other than standard (i.e. sigmoid, linear) transfer functions and very unusual weight settings. Figure 29 shows the four module designs and the weights assigned to their connections:
  - The **average module** ($\mathrm{MOD}^{\mathrm{Avg}}$, figure 29 (a)) uses only linear transfer functions in units averaging the inputs. Four of these modules can be used to calculate $\mu_1, ..., \mu_4$.
  - The **variance module** ($\mathrm{MOD}^{\mathrm{Var}}$, figure 29 (b)) uses a submodule (on the left) to calculate the average of the subwindow it is presented. The other submodule (on the right) just transports the original data to lower layers[10]. The calculated averages are then subtracted from the original inputs, followed by a layer of units using a $f(a) = \tanh^2(a)$ transfer function to approximate the square of the input[11] (see figure 28 (a)). Four of these modules can be used to find $\sigma_1^2, \ldots, \sigma_4^2$.

[10] This part is not strictly necessary, but was incorporated since links between non-adjacent layers are difficult to implement in the software package used (Hoekstra et al., 1996).

[11] This function is chosen since it approximates a^2 well on the interval it will be applied to,

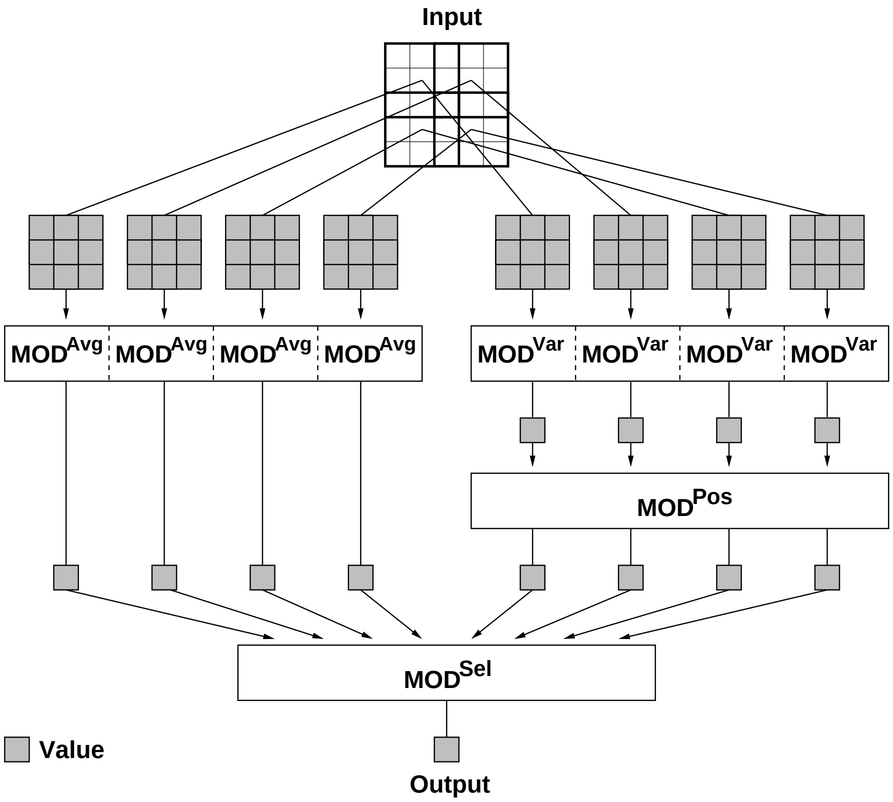


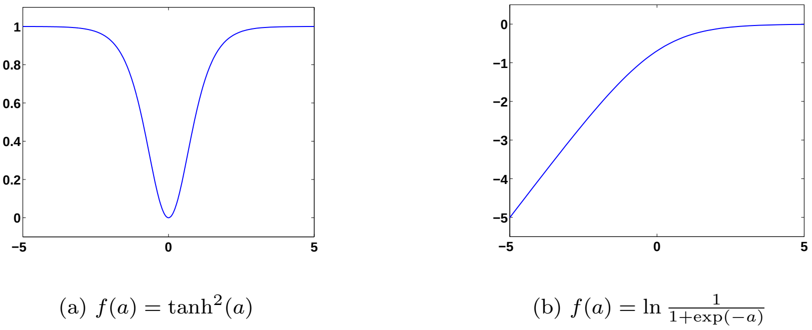
Figure 27: A modular ANN. MOD$^{\mathrm{Avg}}$, MOD$^{\mathrm{Var}}$, MOD$^{\mathrm{Pos}}$ and MOD$^{\mathrm{Sel}}$ denote the ANN modules, corresponding to the operations shown in figure 25 (b). The top layer is the input layer. In this figure, shaded boxes correspond to values transported between modules, not units.

– The **position-of-minimum module** for selecting the position of the minimum of four inputs (MOD$^{\mathrm{Pos}}$, figure 29 (c)) is the most complicated one. Using the logarithm of the sigmoid as a transfer function,

$$f(a) = \ln \frac{1}{1 + \exp(-a)} \tag{19}$$

(see figure 28 (b)), units in the first three hidden layers act as *switches* comparing their two inputs. Alongside these switches, linear transfer function units are used to *transport* the original values to deeper layers. Weights w^A and w^B are very high to enable the units to act as switches. If the input connected using weight w^A (input I^A) is greater than the input connected using weight w^B (input I^B), the sum will be large and negative, the output of the sigmoid will approach 0.0 and the output of the unit will be $-\infty$. If $I^B > I^A$, on the other hand, the sum will be

but is bounded: it asymptotically reaches 1 as the input grows to $\pm\infty$. The latter property is important for training the ANN, as unbounded transfer functions will hamper convergence.



(a) $f(a) = \tanh^2(a)$

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

Figure 28: The non-standard transfer functions used in (a) $\mathrm{MOD}^{\mathrm{Var}}$ and (b) $\mathrm{MOD}^{\mathrm{Pos}}$.

large and positive, the output of the sigmoid part will approach 1.0 and the final output of the unit will be 0.0. This output can be used as an inhibiting signal, by passing it to units of the same type in lower layers. In this way, units in the third hidden layer have as output – if inputs are denoted as $\sigma_1, \sigma_2, \sigma_3$ and σ_4 – :

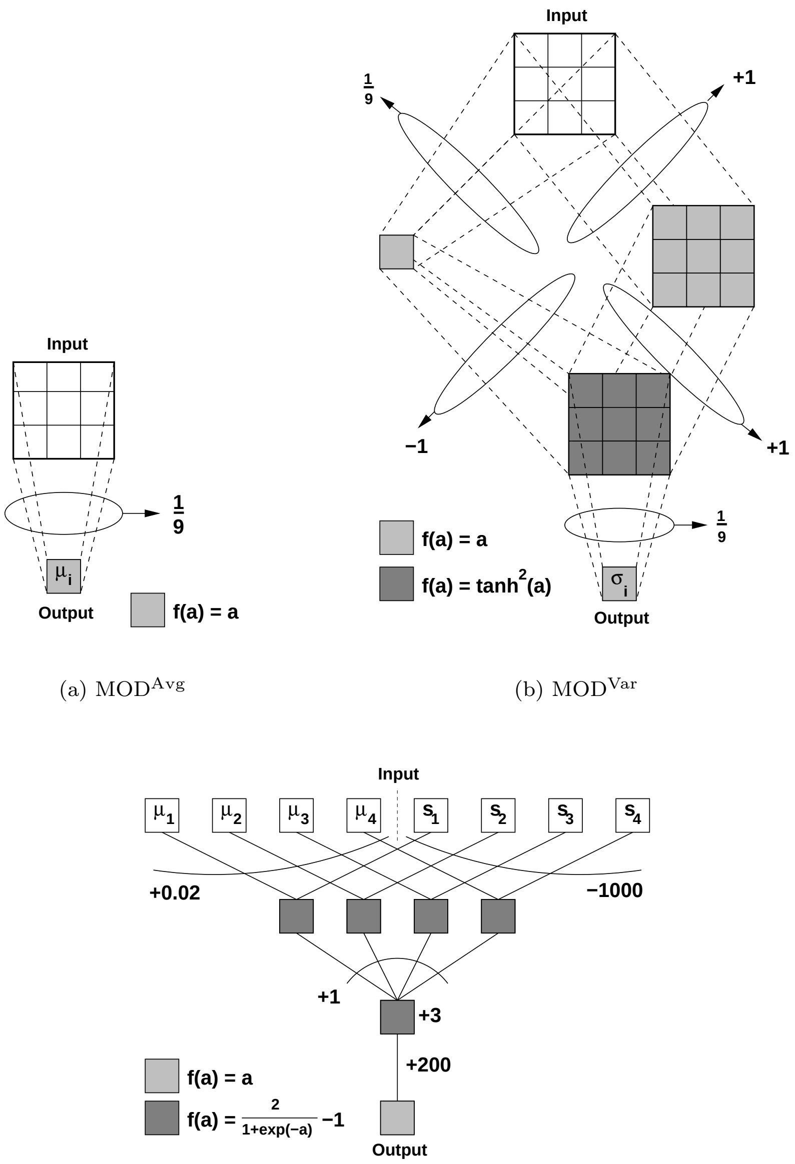
$$s_i = \begin{cases} 0.0 & \sigma_i < \min_{m=1,\ldots,4 \wedge m \neq i} \sigma_m \\ 0.5 & \text{otherwise} \end{cases} \tag{20}$$

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

– The **selection module** ($\mathrm{MOD}^{\mathrm{Sel}}$, figure 29 (d)) uses large weights coupled to the position-of-minimum module outputs (inputs s_1, s_2, s_3 and s_4) to suppress the unwanted average values μ_i before adding these. The small weights with which the average values are multiplied and the large incoming weight of the output unit are used to avoid the nonlinearity of the transfer function.

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

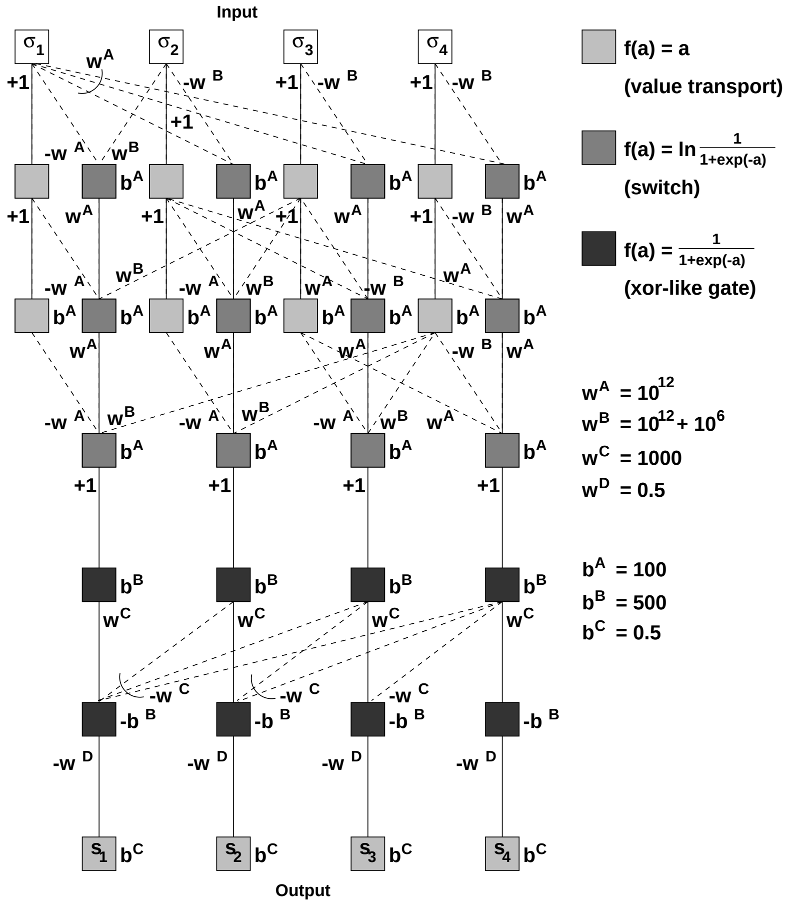
- $\mathbf{ANN_2^M}$ modules have the same architectures as those of $\mathrm{ANN}_1^{\mathrm{M}}$. However, in this case the weights were not fixed, hence the modules could be trained. These modules were expected to perform poorly, as some of the optimal weights (as set in $\mathrm{ANN}_1^{\mathrm{M}}$) were very high and some of the transfer functions are unbounded (see figure 28 (b)).



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

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

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



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

Figure 29: The modules for (a) calculating the average, (b) calculating the variance, (c) finding the position of the minimum variance and (d) selecting the right average. In all modules, the top layer is the input layer. Differently shaded boxes correspond to units with different transfer functions.

- In $\mathbf{ANN_3^M}$ modules, non-standard transfer functions were no longer used. As a result, the modules MOD^{Var} and MOD^{Pos} had to be replaced by standard ANNs. These ANNs contained 2 layers of 25 hidden units, each of which had a double sigmoid transfer function. This number of hidden units was thought to give the modules a sufficiently large number of parameters, but keeps training times feasible.

- In the final type, $\mathbf{ANN_4^M}$, all modules consisted of standard ANNs with 2 hidden layers of 25 units each.

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

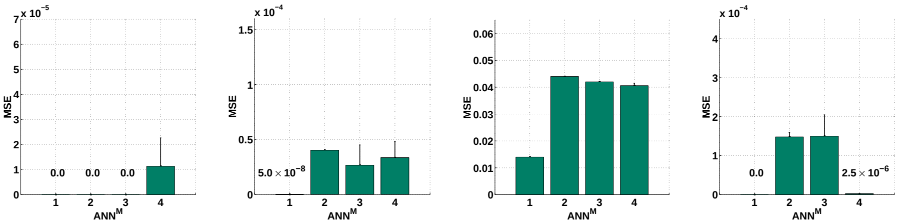
Note that the ANN_1^{M} architecture is probably not the only error-free implementation possible using ANN units. It should be clear from the discussion, though, that any architecture should resort to using non-standard transfer functions and unconventional weight settings to perform the nonlinear operations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

5.2.2 Standard networks

As section 3 showed, the use of prior knowledge in ANN design will not always guarantee that such ANNs will perform better than standard architectures. To validate results obtained with the ANNs described in the previous section, experiments were also performed with standard, fully connected feed-forward ANNs. Although one hidden layer should theoretically be sufficient (Funahashi, 1989; Hornik et al., 1989), the addition of a layer may ease training or lower the number of required parameters (although there is some disagreement on this). Therefore, ANNs having one or two hidden layers of 1, 2, 3, 4, 5, 10, 25, 50, 100 or 250 units each were used. All units used the double sigmoid transfer function. These ANNs will be referred to as $\text{ANN}_{L\times U}^{\text{S}}$, where L indicates the number of hidden layers (1 or 2) and U the number of units per hidden layer. ANN_L^{S} will be used to denote the entire set of ANNs with L hidden layers.

5.2.3 Data sets and training

To train the ANNs, a training set was constructed by drawing samples randomly, using a uniform distribution, from image A (input) and its Kuwahara filtered version (output), both shown in figure 26 (a). The original 8-bit 256 grey value image was converted to a floating point image and rescaled to the range $[-0.5, 0.5]$. Three data sets were constructed, containing 1,000 samples each: a training set, a validation set and a testing set. The validation set was used to prevent overtraining: if the error on the validation set did not drop below the minimum error found so far on that set for 1,000 cycles, training was stopped. Since in all experiments only $k = 3$ Kuwahara filters were studied, the input to each ANN was a 5×5 region of grey values and the training target was 1 value. For the modular ANNs, additional data sets were constructed from these



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

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

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

(g) $\mathrm{MOD}^{\mathrm{Sel}}$

Figure 30: Performance of the individual modules on the testing set in each of the modular ANNs, $\mathrm{ANN}_1^{\mathrm{M}} \ldots \mathrm{ANN}_4^{\mathrm{M}}$.

original data sets to obtain the mappings required by the individual ANNs (average, variance, position-of-minimum and selection).

For training, the standard stochastic back propagation algorithm (Rumelhart et al., 1986) was used. Weights were initialised to random values drawn from a uniform distribution in the range $[-0.1, 0.1]$. The learning rate was set to 0.1; no momentum was used. Training was stopped after 25,000 cycles or if the validation set indicated overtraining, whichever came first. All experiments were repeated five times with different random initialisations; all results reported are averages over five experiments. Where ever appropriate, error bars indicate standard deviations.

5.2.4 Results

Results are given in figures 30 and 31. These will be discussed here for the different architectures.

Modules The different modules show rather different behaviour (figure 30). Note that in these figures the MSE was calculated on a testing set of 1,000 samples. As was to be expected, the MSE is lowest for the hand-constructed $\mathrm{ANN}_1^{\mathrm{M}}$ modules: for all ANNs except $\mathrm{MOD}^{\mathrm{Pos}}$, it was 0. The error remaining for the $\mathrm{MOD}^{\mathrm{Pos}}$ module may look quite high, but is caused mainly by the ANN choosing a wrong minimum when two or more input values σ_i are very similar. Although the effect on the behaviour of the final module ($\mathrm{MOD}^{\mathrm{Sel}}$) will be negligible, the MSE is quite high since one output which should have been 0.5 is incorrectly set to 0.0 and vice versa, leading to an MSE of 0.25 for that input pattern. For the other ANNs, it seems that if the manually set weights are dropped ($\mathrm{ANN}_2^{\mathrm{M}}$), the modules are not able to learn their function as well as possible (i.e., as well as $\mathrm{ANN}_1^{\mathrm{M}}$). Nonetheless, the MSE is quite good and comparable to $\mathrm{ANN}_3^{\mathrm{M}}$ and $\mathrm{ANN}_4^{\mathrm{M}}$.

When the individual tasks are considered, the average is obviously the easiest function to approximate. Only for $\mathrm{ANN}_4^{\mathrm{M}}$, in which standard modules with two hidden layers were used, is the MSE larger than 0.0; apparently these modules generalise less well than the hand-constructed, linear $\mathrm{MOD}^{\mathrm{Avg}}$s. The variance too is not difficult: MSEs are $\mathcal{O}(10^{-5})$. Clearly, the position-of-minimum task is the hardest. Here, almost all ANNs perform poorly. Performances on the

Type	MSE	MSE with $\mathrm{MOD}^{\mathrm{Pos}}$ of $\mathrm{ANN}_1^{\mathrm{M}}$
$\mathrm{ANN}_2^{\mathrm{M}}$	$9.2 \times 10^{-1} \pm 5.2 \times 10^{-1}$	$8.7 \times 10^{-4} \pm 1.7 \times 10^{-4}$
$\mathrm{ANN}_3^{\mathrm{M}}$	$1.2 \times 10^{-1} \pm 1.2 \times 10^{-1}$	$1.0 \times 10^{-3} \pm 2.0 \times 10^{-4}$
$\mathrm{ANN}_4^{\mathrm{M}}$	$3.6 \times 10^{-2} \pm 1.7 \times 10^{-2}$	$1.2 \times 10^{-3} \pm 2.4 \times 10^{-4}$

Table 2: Dependence of performance, in MSE on the image A testing set, on the $\mathrm{MOD}^{\mathrm{Pos}}$ module. Values given are average MSEs and standard deviations.

selection problem, finally, are quite good. What is interesting is that the more constrained modules ($\mathrm{ANN}_2^{\mathrm{M}}$, $\mathrm{ANN}_3^{\mathrm{M}}$) perform less well than the standard ones. Here again the effect that the construction is closely connected to the optimal set of weights plays a role. Although there is an optimal weight set, the training algorithm did not find it.

Modular networks When the modules are concatenated, the initial MSEs of the resulting ANNs are poor: for $\mathrm{ANN}_2^{\mathrm{M}}$, $\mathrm{ANN}_3^{\mathrm{M}}$ and $\mathrm{ANN}_4^{\mathrm{M}}$ $\mathcal{O}(1)$, $\mathcal{O}(10^{-1})$ and $\mathcal{O}(10^{-2})$ respectively. The $\mathrm{MOD}^{\mathrm{Pos}}$ module is mainly responsible for this; it is the hardest module to learn due to the nonlinearity involved (see the discussion in section 5.2.4). If the trained $\mathrm{MOD}^{\mathrm{Pos}}$ in $\mathrm{ANN}_2^{\mathrm{M}} \ldots \mathrm{ANN}_4^{\mathrm{M}}$ is replaced by the constructed $\mathrm{ANN}_1^{\mathrm{M}}$ module, the overall MSE always decreases significantly (see table 2). This is an indication that, although its MSE seems low ($\mathcal{O}(10^{-2})$), this module does not perform well. Furthermore, it seems that the overall MSE is highly sensitive to the error this module makes.

However, when the complete ANNs are trained a little further with a low learning rate (0.1), the MSE improves rapidly: after only 100-500 learning cycles training can be stopped. In (Pugmire et al., 1998), the same effect occurs. The MSEs of the final ANNs on the entire image are shown in figures 31 (a), (e) and (i) for images A, B and C, respectively. Images B and C were pre-processed in the same way as image A: the original 8-bit (B) and 5-bit (C) 256 grey value images were converted to floating point images, with grey values in the range $[-0.5, 0.5]$.

To get an idea of the significance of these results, re-initialised versions of the same ANNs were also trained. That is, all weights of the concatenated ANNs were initialised randomly without using the prior knowledge of modularity. The results of these training runs are shown in figures 31 (b), (f) and (j). Note that only $\mathrm{ANN}_2^{\mathrm{M}}$ cannot be trained well from scratch, due to the non-standard transfer functions used. For $\mathrm{ANN}_3^{\mathrm{M}}$ and $\mathrm{ANN}_4^{\mathrm{M}}$ the MSE is comparable to the other ANNs. This would indicate that modular training is not beneficial, at least according to the MSE criterion.

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

Standard networks Results for the standard ANNs, $\mathrm{ANN}^{\mathrm{S}}$s, are shown in figure 31 (c)-(d), (g)-(h) and (k)-(l) for images A, B and C. In each case, the first figure gives the results for ANNs with one hidden layer; the second figure for ANNs with two hidden layers. What is most striking is that for almost all sizes of the ANNs the MSEs are more or less the same. Furthermore, this MSE is nearly identical to the one obtained by the modular ANNs $\mathrm{ANN}_2^{\mathrm{M}} \ldots \mathrm{ANN}_4^{\mathrm{M}}$. It also seems that the smaller ANNs, which give a slightly larger MSE on Image A and Image B, perform a bit worse on Image C. This is due to the larger amount of edge pixels in Image C; the next section will discuss this further.

5.3 Investigating the error

The experiments in the previous section indicate that, no matter which ANN is trained (except for $\mathrm{ANN}_1^{\mathrm{M}}$), the MSE it will be able to reach on the images is equal. However, visual inspection shows small differences between images filtered by various ANNs; see e.g. the left and centre columns of figure 32. To gain more insight in the actual errors the ANNs make, a technique can be borrowed from the field of Bayesian learning, which allows the calculation of error bars for each output of the ANN (Bishop, 1995). The computation is based on the Hessian of the ANN output w.r.t. its weights $\mathbf{w}$, $\mathbf{H} = \nabla^2_{\mathbf{w}} R(\mathbf{x};\ \mathbf{w})$, which needs to be found first. Using $\mathbf{H}$, for each input $\mathbf{x}$ a corresponding variance σ^2_{tot} can be found. This makes it possible to create an image in which each pixel corresponds to $2\sigma_{tot}$, i.e. the grey value equals half the width of the error bar on the ANN output at that location. Conversely, the inverse of σ_{tot} is sometimes used as a measure of *confidence* in an ANN output for a certain input.

For a number of ANNs, the Hessian was calculated using a finite differencing approximation (Bishop, 1995). To calculate the error bars, this matrix has to be inverted first. Unfortunately, for the $\mathrm{ANN}^{\mathrm{M}}$s, inversion was impossible as their Hessian matrices were too ill-conditioned because of the complicated architectures, containing fixed and shared weights. Figures 32 (b) and (c) show the results for two standard ANNs, $\mathrm{ANN}^{\mathrm{S}}_{1\times 25}$ and $\mathrm{ANN}^{\mathrm{S}}_{2\times 25}$. In the left column the ANN output for Image A (26 (a)) is shown. The centre column shows the absolute difference between this output and the target image. In the third column the error bars calculated using the Hessian are shown.

The figures show that the error the ANN makes is not spread out evenly over the image. The highest errors occur near the edges in Image A, as can be seen by comparing the centre column of figure 32 with the gradient magnitude of $|\nabla I_A|$ of Image A, shown in figure 33 (a). This gradient magnitude is calculated as (Young et al., 1998)

$$|\nabla I_A| = \sqrt{\left(\frac{\delta I_A}{\delta x}\right)^2 + \left(\frac{\delta I_A}{\delta y}\right)^2} \qquad (21)$$

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

The error bar images, in the right column of figure 32, show that the standard deviation of ANN output is also highest on and around the edges. Furthermore, although the output of the ANNs look identical, the error bars show that the ANNs actually behave differently.

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

5.4 Discussion

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

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

- *the problem may simply be too hard to be learned by a finite-size ANN.* This does not seem plausible, since even for a two-hidden layer ANN with 250 hidden units per layer, resulting in a total of 69,000 free parameters, the MSE is no better than for very simple ANNs. One would at least expect to see some enhancement of results;

- *it is possible that the sample size of 1,000 is too small*, as it was rather arbitrarily chosen. An experiment was performed in which $\text{ANN}^{\text{S}}_{1\times 50}$ was trained using training sets with 50, 100, 250, 500, 1,000 and 2,000 samples. The results, given in figure 33 (b), show however that the chosen sample size of 1,000 seems sufficient. The decrease in MSE when using 2,000 samples in the training set is rather small;

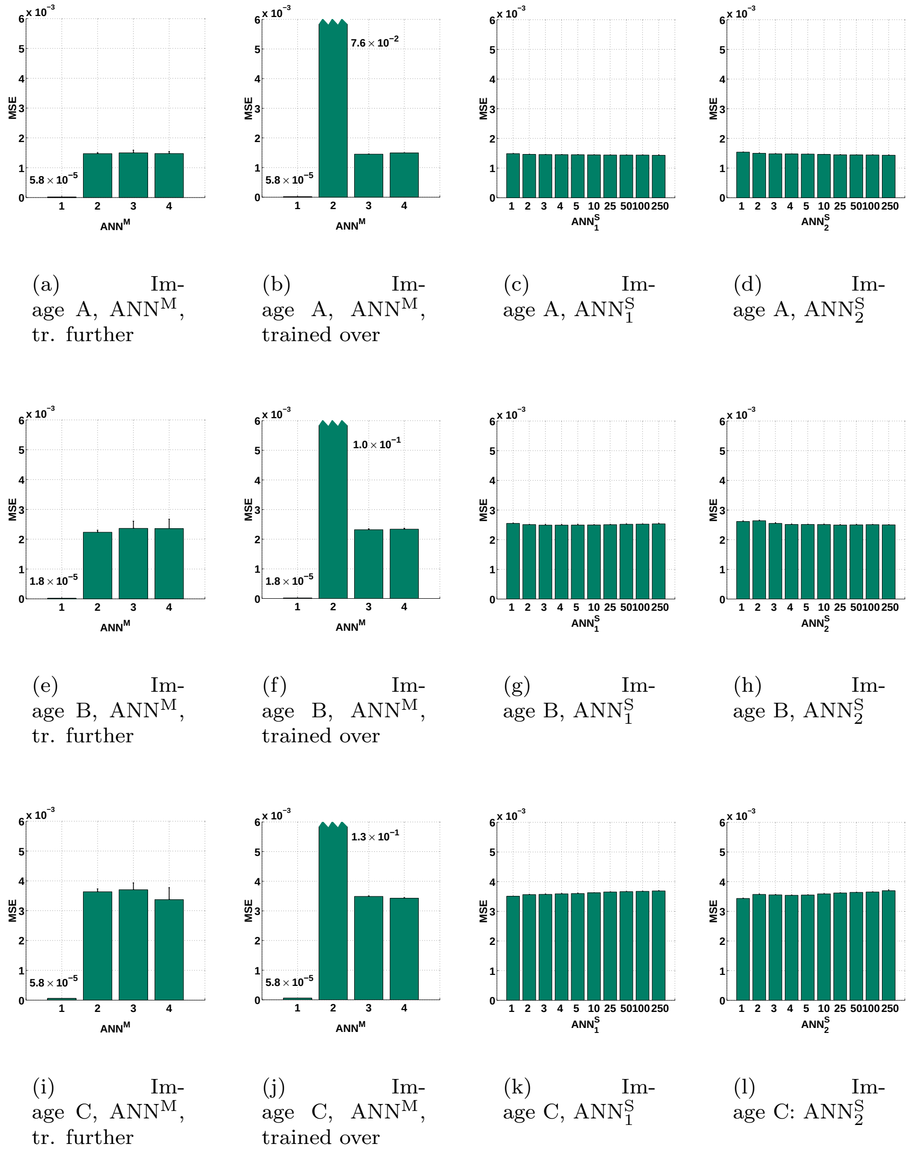
- *the training set may not be representative for the problem*, i.e. the nature of the problem may not be well reflected in the way the set is sampled from the image;

- *the error criterion may not be fit for training the ANNs or assessing their performance.* It is very well possible that the MSE criterion used is of limited use in this problem, since it weighs both the interesting parts of the image, around the edges, and the less interesting parts equally;

- *the problem may be of such a nature that local minima are prominently present in the error surface*, while the global minima are very hard to reach, causing suboptimal ANN operation.

6 Inspection and improvement of regression networks

This section tries to answer the questions raised by the experiments in the previous section, by investigating the influence of the data set, the appropriateness of the MSE as a performance measure and the trained ANNs themselves.



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

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

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

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

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

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

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

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

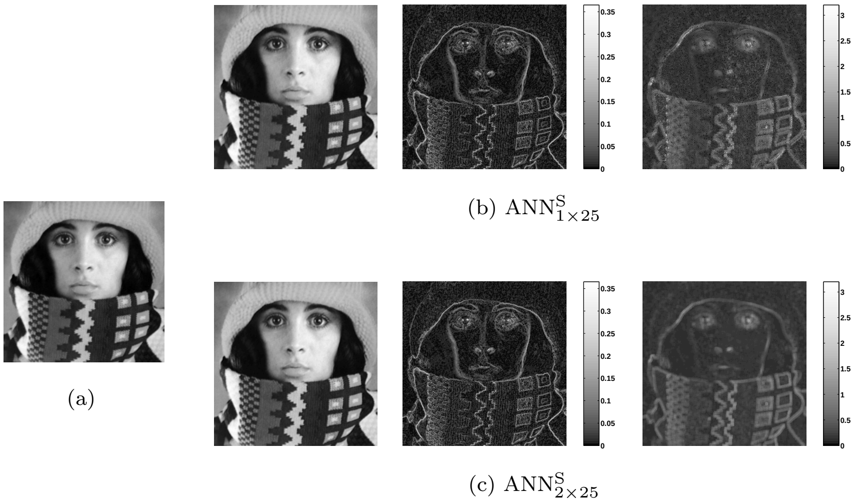
(i) Image C, ANN^{M}, tr. further

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

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

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

Figure 31: Performance of all ANN^{M}s and ANN^{S}s on the three images used: (a)-(d) on image A (fig. 26 (a)), (e)-(h) on image B (fig. 26 (b)) and (i)-(l) on image C (fig. 26 (c)). For the ANN^{S}s, the x-axis indicates the number of hidden units per layer.

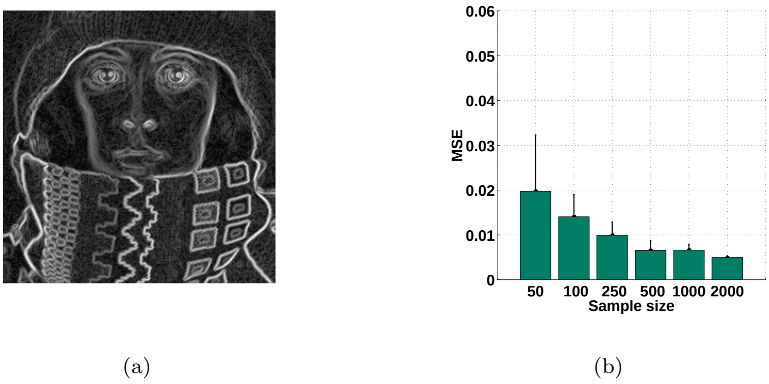


(b) $\text{ANN}^{\text{S}}_{1\times 25}$

(a)

(c) $\text{ANN}^{\text{S}}_{2\times 25}$

Figure 32: (a) The original Image A. (b) and (c), from left to right: outputs of two ANN^{S}s on Image A; absolute differences between target image and ANN output and ANN output error bar widths plotted as grey values.



(a)

(b)

Figure 33: (a) The gradient magnitude of Image A, $|\nabla I_A|$. (b) Performance of $\text{ANN}^{\text{S}}_{1\times 50}$ for various training set sample sizes.

6.1 Edge-favouring sampling

Inspection of the ANN outputs and the error bars on those outputs led to the conclusion that the ANNs had learned to emulate the Kuwahara filter well in most places, except in regions near edges (section 55.3). A problem in sampling a training set from an image[12] for this particular application is that such interesting regions, i.e. the regions where the filter is nonlinear, are very poorly represented. Edge pixels constitute only a very small percentage of the total number of pixels in an image (as a rule of thumb, $\mathcal{O}(\sqrt{n})$ edge pixels on $\mathcal{O}(n)$ image pixels) and will therefore not be represented well in the training set when sampling randomly using a uniform distribution.

To learn more about the influence of the training set on performance, a second group of data sets was created by sampling from Image A (figure 26 (a)) with a probability density function based on its gradient magnitude image $|\nabla I_A|$ (eqn. 21). If $|\nabla I|$ is scaled by a factor c such that $\int_x \int_y c \cdot |\nabla I(x,y)| dydx = 1$, and used as a probability density function when sampling, edge regions have a much higher probability of being included in the data set than pixels from flat regions. This will be called *edge-favouring sampling*, as opposed to *normal sampling*.

6.1.1 Experiments

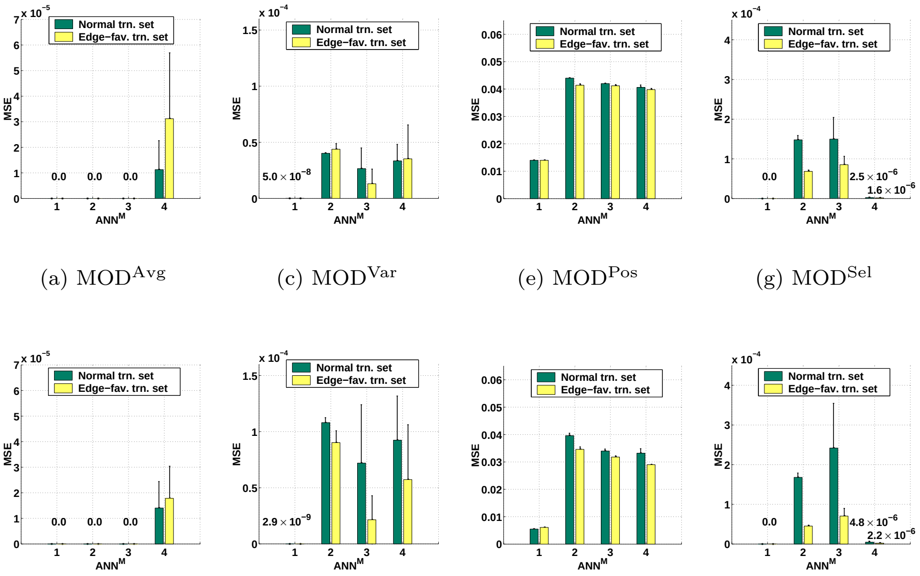
Performances (in MSE) of ANNs trained on this edge-favouring set are given in figures 34 and 35. Note that the results obtained on the normal training set (first shown in figure 31) are included again to facilitate comparison. The sampling of the data set clearly has an influence on the results. Since the edge-favouring set contains more samples taken from regions around edges, the task of finding the mean is harder to learn due to the larger variation. At the same time, it eases training the position-of-minimum and selection modules. For all tasks except the average, the final MSE on the edge-favouring testing set (figures 34 (b), (d), (f) and (h)) is better than that of ANNs trained using a normal training set. The MSE is, in some cases, even lower on the normal testing set (figures 34 (e) and (g)).

Overall results for the modular and standard ANNs (figure 35) suggest that performance *decreases* when ANNs are trained on a specially selected data set (i.e., the MSE increases). However, when the quality of the filtering operation is judged by looking at the filtered images (see e.g. figure 36), one finds that these ANNs give superior results in approximating the Kuwahara filter. Clearly, there is a discrepancy between performance as indicated by the MSE and visual perception of filter quality. Therefore, below we will investigate the possibility of finding another way of measuring performance.

6.2 Performance measures for edge-preserving smoothing

The results given in section 66.16.1.1 show that it is very hard to interpret the MSE as a measure of filter performance. Although the MSEs differ only slightly, visually the differences are quite large. If images filtered by various

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



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

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

Figure 34: Performance of the individual modules in each of the modular ANNs, $\mathrm{ANN}_1^{\mathrm{M}} \ldots \mathrm{ANN}_4^{\mathrm{M}}$ on the normal testing set (top row) and edge-favouring testing set (bottom row).

ANNs trained on the normal and edge-favouring data sets are compared, it seems clear which ANN performs better. As an example, figure 36 shows two filtered images. The left image was filtered by $\mathrm{ANN}_4^{\mathrm{M}}$ trained on an edge-favouring training set. The image on the right is the output of $\mathrm{ANN}_{1\times100}^{\mathrm{S}}$ trained on a normal data set. Although the MSEs are nearly equal (1.48×10^{-3} for the left image versus 1.44×10^{-3} for the right one), in the left image the edges seem much crisper and the regions much smoother than in the image on the right; that is, one would judge the filter used to produce the left image to perform better.

One would like to find a measure for filter performance which bears more relation to this qualitative judgement than the MSE. The reason why the MSE is so uninformative is that by far the largest number of pixels do not lie on edges. Figure 37 (a) illustrates this: it shows that the histogram of the gradient magnitude image is concentrated near zero, i.e. most pixels lie in flat regions. Since the MSE averages over all pixels, it may be quite low for filters which preserve edges poorly. Vice versa, the visual quality of the images produced by the ANNs trained using the edge-favouring data set may be better while their MSE is worse, due to a large number of small errors made in flat regions.

The finding that the MSE does not correlate well with perceptual quality judgement is not a new one. A number of alternatives have been proposed, among which the mean absolute error (MAE) seems to be the most prominent one. There is also a body of work on performance measures for edge detection, e.g. Pratt's Figure of Merit (FOM) (Pratt, 1991) or Average Risk (Spreeuwers, 1992). However, none of these capture the dual goals of edge sharpening and region smoothing present in this problem.

6.2.1 Smoothing versus sharpening

In edge-preserving smoothing, two goals are pursued: on the one hand the algorithm should preserve edge sharpness, on the other hand it should smooth the image in regions that do not contain edges. In other words, the gradient of an image should remain the same in places where it is high[13] and decrease where it is low.

If the gradient magnitude $|\nabla I|$ of an image I is plotted versus $|\nabla f(I)|$ of a Kuwahara-filtered version $f(I)$, for each pixel $I_{(i,j)}$, the result will look like figure 37 (b). In this figure, the two separate effects can be seen: for a number of points, the gradient is increased by filtering while for another set of points the gradient is decreased. The steeper the upper cloud, the better the sharpening; the flatter the lower cloud, the better the smoothing. Note that the figure gives no indication of the density of both clouds: in general, by far the most points lie in the lower cloud, since more pixels lie in smooth regions than on edges. The graph is reminiscent of the scattergram approach discussed (and denounced) in Katsulai and Arimizu, 1981, but here the scattergram of the gradient magnitude images is shown.

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

$$\mathcal{A} = \left\{ (|\nabla I|_{(i,j)}, |\nabla f(I)|_{(i,j)}) \;\middle|\; |\nabla I|_{(i,j)} \geq |\nabla f(I)|_{(i,j)} \right\} \tag{22}$$

$$\mathcal{B} = \left\{ (|\nabla I|_{(i,j)}, |\nabla f(I)|_{(i,j)}) \;\middle|\; |\nabla I|_{(i,j)} < |\nabla f(I)|_{(i,j)} \right\} \tag{23}$$

Lines $y = ax + b$ can be fitted through both sets using a robust estimation technique, minimising the absolute deviation (Press et al., 1992), to get a *density-independent* estimate of the factors with which edges are sharpened and flat regions are smoothed:

$$(a_\mathcal{A}, b_\mathcal{A}) = \arg\min_{(a,b)} \sum_{(x,y)\in\mathcal{A}} |y - (ax + b)| \tag{24}$$

$$(a_\mathcal{B}, b_\mathcal{B}) = \arg\min_{(a,b)} \sum_{(x,y)\in\mathcal{B}} |y - (ax + b)| \tag{25}$$

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

[13] Or even increase. If the regions divided by the edge become smoother, the gradient of the edge itself may increase, as long as there was no *overshoot* in the original image. Overshoot is defined as the effect of artificially sharp edges, which may be obtained by adding a small value to the top part of an edge and subtracting a small value from the lower part (Young et al., 1998).

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

$$Smoothing(f,I) = \frac{|\mathcal{A}|}{|\mathcal{A}|+|\mathcal{B}|}(a'_{\mathcal{A}} - 1) \quad \text{and} \tag{26}$$

$$Sharpening(f,I) = \frac{|\mathcal{B}|}{|\mathcal{A}|+|\mathcal{B}|}(a_{\mathcal{B}} - 1) \tag{27}$$

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

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

$$f_G(I,\sigma) = I \otimes \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{28}$$

for[14] $\sigma = 0.0, 0.1, \ldots, 2.0$; and an unsharp masking filter

$$f_U(I,k) = I - k \times \left(I \otimes \begin{pmatrix} \text{-1} & 2 & \text{-1} \\ 2 & \text{-4} & 2 \\ \text{-1} & 2 & \text{-1} \end{pmatrix} \right) \tag{29}$$

which subtracts k times the Laplacian[15] from an image, $k = 0.0, 0.1, \ldots, 2.0$.

6.2.2 Experiments

Smoothing and sharpening performance values were calculated for all ANNs discussed in section 66.16.1.1 The results are shown in figure 38. First, lines of performance values for the Gaussian and unsharp masking filters give an indication of the range of possible values. As expected, the Gaussian filter on Images A and B (figures 26 (a) and (b)) gives high smoothing values and low sharpening values, while the unsharp masking filter gives low smoothing values and high sharpening values. The Kuwahara filter scores high on smoothing and low on sharpening. This is exactly as it should be: the Kuwahara filter should smooth while preserving the edges, it should not necessarily sharpen them. If ANNs have a higher sharpening value, they are usually producing overshoot around the edges in the output images.

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

[15] This is an implementation of the continuous Laplacian edge detector mentioned in section 44.14.1.1, different from the discrete detector shown in figure 11.

The measures calculated for Image C (figure 26 (c)) show the limitations of the method. In this image there is a large number of very sharp edges in an otherwise already rather smooth image. For this image the Gaussian filter gives only very low smoothing values and the unsharp masking filter gives no sharpening value at all. This is due to the fact that for this image, subtracting the Laplacian from an image produces a very small sharpening value, together with a *negative* smoothing value, caused by the Laplacian greatly enhancing the amount of noise in the image. Since the values were clipped at 0, the results are not shown in the figure.

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

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

Most standard ANNs perform poorly. Only for $\text{ANN}_{2\times10}^{\text{S}}$, $\text{ANN}_{2\times25}^{\text{S}}$ and $\text{ANN}_{2\times50}^{\text{S}}$ performance is reasonable. In retrospect, this concurs with the drop in the MSE that can be seen in figure 35 (d), although the differences there are very small. $\text{ANN}_{2\times50}^{\text{S}}$ clearly performs best. A hypothesis is that this depends on the training of the ANNs, since training parameters were not optimised for each ANN. To verify this, the same set of standard ANNs was trained in experiments in which the weights were initialised using random values drawn from a uniform distribution over the range $[-1.0, 1.0]$, using a learning rate of 0.5. Now, the optimal standard ANN was found to be $\text{ANN}_{2\times25}^{\text{S}}$, with all other ANNs performing very poorly.

Generalisation is, for all ANNs, reasonable. Even on Image C (figure 26 (c)), which differs substantially from the training image (Image A, figure 26 (a)), performance is quite good. The best standard ANN, $\text{ANN}_{2\times50}^{\text{S}}$, seems to generalise a little better than the modular ANNs.

6.2.3 Discussion

In Dijk et al., 1999, it is shown that the smoothing and sharpening performance measures proposed here correlate well with human perception. It should be noted that in this study, subjects had less problems in discerning various levels of smoothing than they had with levels of sharpening. This indicates that the two measures proposed are not equivalently spaced.

The fact that the measures show that edge-favouring sampling in building a training set increases performance considerably, suggests possibilities for extensions. (Pugmire et al., 1998) claim that learning should be structured, i.e. start with the general problem and then proceed to special cases. This can be easily accomplished in training set construction, by adding a constant to each pixel in the gradient magnitude image before scaling and using it as a probability density function from which window coordinates are sampled. If this constant is gradually lowered, edge-pixels become better represented in the training set. Another, more general possibility would be to train ANNs on normally sampled data first and calculate an error image (such as those shown in the centre column of figure 32). Next, the ANN could be trained further – or re-trained – on a data set sampled using the distribution of the errors the ANN made, a new error image can be calculated, and so on. This is similar to boosting and arcing approaches in classification (Shapire, 1990). An advantage is that this does not use the prior knowledge that edges are important, which makes it more generally applicable.

6.2.4 Training using different criteria

Ideally, the sharpening and smoothing performance measures discussed in the previous section should be used to train ANNs. However, this is infeasible as they are not differentiable. This means they could only be used in learning procedures which do not need the criterion function to be differentiable, such as reinforcement learning (Gullapalli, 1990). This falls outside the scope of the experiments in this section.

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

In the literature, several more general alternatives to the MSE (eqn. 8) have been proposed (Hertz et al., 1991; Burrascano, 1991). Among these, a very flexible family of error criteria based on the L_p norm is:

$$E_p(\mathbf{W}, \mathbf{B}) = \frac{1}{2|\mathcal{L}|} \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{L}} \sum_{k=1}^{m} |R(\mathbf{x}^i;\ \mathbf{W}, \mathbf{B})_k - y_k^i|^p, \tag{30}$$

where $p \in \mathbb{Z}^*$. Note that for $p = 2$, this criterion is equal to the MSE. For $p = 0$, each error is considered equally bad, no matter how small or large it is. For $p = 1$, the resulting error criterion is known as the mean absolute error or MAE. The MAE is more robust to outliers than the MSE, as larger errors are given relatively smaller weights than in the MSE. For $p > 2$, larger errors are given more weight, i.e. the data is considered not to contain outliers. In fact, which p to use should be decided by assuming a noise model for the target data (Burrascano, 1991). The L_1 norm (robust to outliers) corresponds to a noise distribution with large tails, a Laplacian distribution, under which outliers are probable. At the other extreme, L_∞ corresponds to a uniform noise distribution.

As discussed before, the Kuwahara filter is most interesting around the edges in an image, were the filter behaves nonlinearly. It was also shown that exactly around these edges most ANNs make the largest errors (figure 32). Therefore, it makes sense to use an error criterion which puts more emphasis on larger errors, i.e. the L_p norm for $p > 2$. To this end, a number of experiments were run in which different norms were used.

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

The best performing ANN found in section 66.2, $\text{ANN}^{\text{S}}_{2\times 50}$, was trained using CGD with the L_p norm. The parameter p was set to $1, 2, 3, 5$ and 7, and both the normal and the edge-favouring training sets were used. The ANN was trained using the same settings as before; in the CGD algorithm, directions were kept conjugate for 10 iterations.

Figure 39 shows the results. Clearly, using the L_p norm helps the ANN trained on the normal set to achieve better performance (figure 39 (a)). For increasing p, the sharpening performance becomes higher. However, the smoothing performance still lags behind that of the ANN trained using the MSE on the edge-favouring training set (fig. 38 (d)).

When $\text{ANN}^{\text{S}}_{2\times 50}$ is trained using the L_p norm on the edge-favouring data set, smoothing performance actually *decreases* (figure 39 (b)). This is caused by the fact that the training set and error criterion in concert stress errors around edges so much, that the smoothing operation in flat regions suffers. Figure 40 illustrates this by showing the output of $\text{ANN}^{\text{S}}_{2\times 25}$ as well as the absolute difference between this output and the target image, for various values of p. For increasing p, the errors become less localised around the edges; for $p \geq 3$ the error in flat regions becomes comparable to that around edges.

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

6.3 Inspection of trained networks

6.3.1 Standard networks

To gain insight into the relatively poor performance of most of the standard ANNs according to the performance measure introduced in section 66.2, a very simple architecture was created, containing only a small number of weights (see figure 41 (a)). Since the Kuwahara filter should be isotropic, a symmetric weight mask was imposed on the weights (cf. section 4.1.2). Furthermore, linear transfer functions were used to avoid the complications introduced in the analysis by the use of sigmoids. No bias was used. This ANN was trained on the normal data set, using a validation set. The learned weight set is shown in figure 42 (a). In filtering terms, the main component looks like a negative Laplacian-of-Gaussian (i.e. the negative values around the centre and the slightly positive values in the four corners). Further analysis showed that this filter closely resembles a linear combination of a normal Gaussian and a Laplacian-of-Gaussian.

To confirm the hypothesis that standard ANNs learned such linear approximations to the Kuwahara filter, a simple standard ANN was trained in the same way ANN^{K} was, using the DCGD training algorithm (section 44.2.2). This ANN, $\text{ANN}^{\text{S}}_{1\times2}$, is shown in figure 41 (b). All weights were initialised to a fixed value of 0.01, λ was set to 1 and the number of directions to be kept conjugate was set to 10. After training, the MSE on the testing set was 1.43×10^{-3}, i.e. comparable to other standard ANNs (fig. 31), and C^2 was 5.1×10^{-3}. The resulting weight sets show that the filter can indeed be decomposed into a Gaussian-like and a negative Laplacian-like filter. Adding more hidden units and training using DCGD, for which results are not shown here, did not cause any new filters to be found.

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

$$\begin{aligned} f(c_1, \sigma_1, c_2, \sigma_2) = c_1 f_G(\sigma_1) - c_2 f_L(\sigma_2) = \\ c_1 \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right) - c_2 \frac{(x^2+y^2) - \sigma_2^2}{2\pi\sigma_2^6} \exp\left(-\frac{x^2+y^2}{2\sigma_2^2}\right) \end{aligned} \qquad (31)$$

in which c_1 and σ_1 are parameters to be estimated for the Gaussian and c_2 and σ_2 are parameters for the Laplacian. Figure 42 (c) shows these two functions.

A Gauss-Newton fitting procedure (Mathworks Inc., 2000) was used to find the parameters of $f(c_1, \sigma_1, c_2, \sigma_2)$ given the weights shown in figure 42 (a). The resulting model weights are shown in figure 42 (b) and a cross-section is shown in figure 42 (c). Although the fit ($c_1 = 10.21$, $\sigma_1 = 2.87$, $c_2 = 3.41$, $\sigma_2 = 0.99$) is not perfect with a model fit MSE $\varepsilon_{\text{f}} = 2.5 \times 10^{-3}$, the correlation between the model and the actual weights is quite high ($C = 0.96$).

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

The results, given in figure 43 show that, at least for the smaller ANNs, the hypothesis is supported by the data. For the ANNs trained on the normal data set, over a large range of sizes (i.e. 1-5, 10 and 25 hidden units) the model closely fits each hidden unit. Only for larger numbers of hidden units the fit becomes worse. The reason for this is that in these ANNs many units have an input weight distribution which is very hard to interpret. However, these units do not play a large role in the final ANN output, since they are weighted by small weights in the next layer.

For the ANNs trained on the edge-favouring set the fit is less good, but still gives a reasonable correlation. Note however that ANNs which have high performance w.r.t. the smoothing and sharpening measures (section 66.26.2.2) do not necessarily show the lowest correlation: ANN^{S}s with more hidden units give even lower correlation. An opposite effect is playing a role here: as ANNs become too large, they are harder to train.

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

6.3.2 Modular networks

It is interesting to see whether the modular ANNs still use their initialisation. Remember that to obtain good performance, the ANN$^{\mathrm{M}}$s had to either be trained further after the modules were concatenated, or re-initialised and trained over (section 5.2.4). The question is whether the modules are still performing the functions they were initially trained on, or has the ANN – after being trained further for a while – found a better solution? To inspect the ANNs, first the modules were evaluated on the sets they were trained with. Next, the concatenated ANN$^{\mathrm{M}}$s were taken apart and the modules were evaluated on the same sets. Figures 44 and 45 show some examples of such plots.

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

- for well-performing ANNs, the mapping in each module is no longer evident. Instead, it seems these modules make rather good use of their nonlinearity (figure 44 (c)). The poorly performing ANNs still show a reasonably linear behaviour (figure 45 (a));

- there is a progressive increase in nonlinearity for ANN$_2^{\mathrm{M}}$, ANN$_3^{\mathrm{M}}$ and ANN$_4^{\mathrm{M}}$ (figures 44 (a)-(c), 45 (a)-(c) and (d)-(f)). The added complexity allows the modules more flexibility when they are trained further. Note however that the basic mapping is still preserved, i.e. the trend is still visible for all units;

- there is an increase in nonlinearity when ANNs are trained on the edge-favouring set instead of the normal set (figures 45 (a)-(c) v. (d)-(f));

- as was to be expected, ANN$^{\mathrm{M}}$s trained from scratch generally do not find the modular structure (figures 44 (d)-(e)).

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

For the MOD$^{\mathrm{Var}}$s and MOD$^{\mathrm{Pos}}$s, the differences are less clear. Most of these modules seem to have no function left in the final ANNs: the outputs are clamped at a certain value or vary a little in a small region around a value. For MOD$^{\mathrm{Var}}$, only ANN$_4^{\mathrm{M}}$ modules have enough flexibility. Here too, training on the edge-favouring set increases the nonlinearity of the output (figures 46 (a)-(c)). MOD$^{\mathrm{Pos}}$, finally, is clamped in almost all architectures. Only ANN$_4^{\mathrm{M}}$

modules give some variation in output (figures 46 (d)-(e)). Networks trained from scratch are always clamped too.

In conclusion, it seems that in most ANNs, the modules on the right hand side (MOD^{Var} and MOD^{Pos}, see figure 27) are disabled. However, the ANNs that do show some activity in these modules are the ANNs that perform best, indicating that the modular initialisation to a certain extent is useful. All results indicate that, although the nature of the algorithm can be used to construct and train individual modules, the errors these modules make are such that the concatenated ANNs perform poorly (see section 5.2.4). That is, modules trained separately on perfect data (e.g. pre-calculated positions of the minimal input) are ill-equipped to handle errors in their input, i.e. the output of preceding modules. For the concatenated ANNs, the training algorithm leaves its modular initialisation to lower the overall MSE; trained as a whole, different weight configurations are optimal. The fact that a trained MOD^{Pos} has a very specific weight configuration (with large weights) to be able to perform its function means it is more susceptible to weight changes than other modules and will easily lose its original functionality. In other words, the final concatenated ANN has “worked around” the errors made by MOD^{Pos} by disabling it.

6.4 Discussion

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

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

In the attempts to try to understand the operation of an ANN instead of treating it like a black box, the interpretability trade-off (discussed in section 4) again played a role. For the modular ANNs, as soon as some of the constraints were dropped, ANN performance became much worse: there was no graceful degradation. It was shown too that it is hard to interpret the operation of the modular ANN after training it further; the operation of the ANN is distributed differently than in the original modular initialisation. The one advantage of using the prior knowledge of the modular nature of the problem (for example, as in ANN_4^{M}) is that it helps to avoid pain-staking optimisation of the number of hidden layers and units, which was shown to be quite critical in standard ANNs. Of course, for different problems this prior knowledge may not be available.

The main conclusion is that, in principle, ANNs *can* be put to use as nonlinear image filters. However, careful use of prior knowledge, selection of ANN architecture and sampling of the training set are prerequisites for good operation. In addition, the standard error measure used, the MSE, will not indicate an ANN performing poorly. Unimportant deviations in the output image may lead to the same MSE as significant ones, if there is a large number of unimportant deviations and a smaller number of important ones. Consequently, standard feed-forward ANNs trained by minimising the traditional MSE are unfit for designing adaptive nonlinear image filtering operations; other criteria should be developed to facilitate easy application of ANNs in this field. Unfortunately, such criteria will have to be specified for each application (see also Spreeuwers, 1992). In this light it is not surprising to find a large number of non-adaptive, application-specific ANNs in the literature.

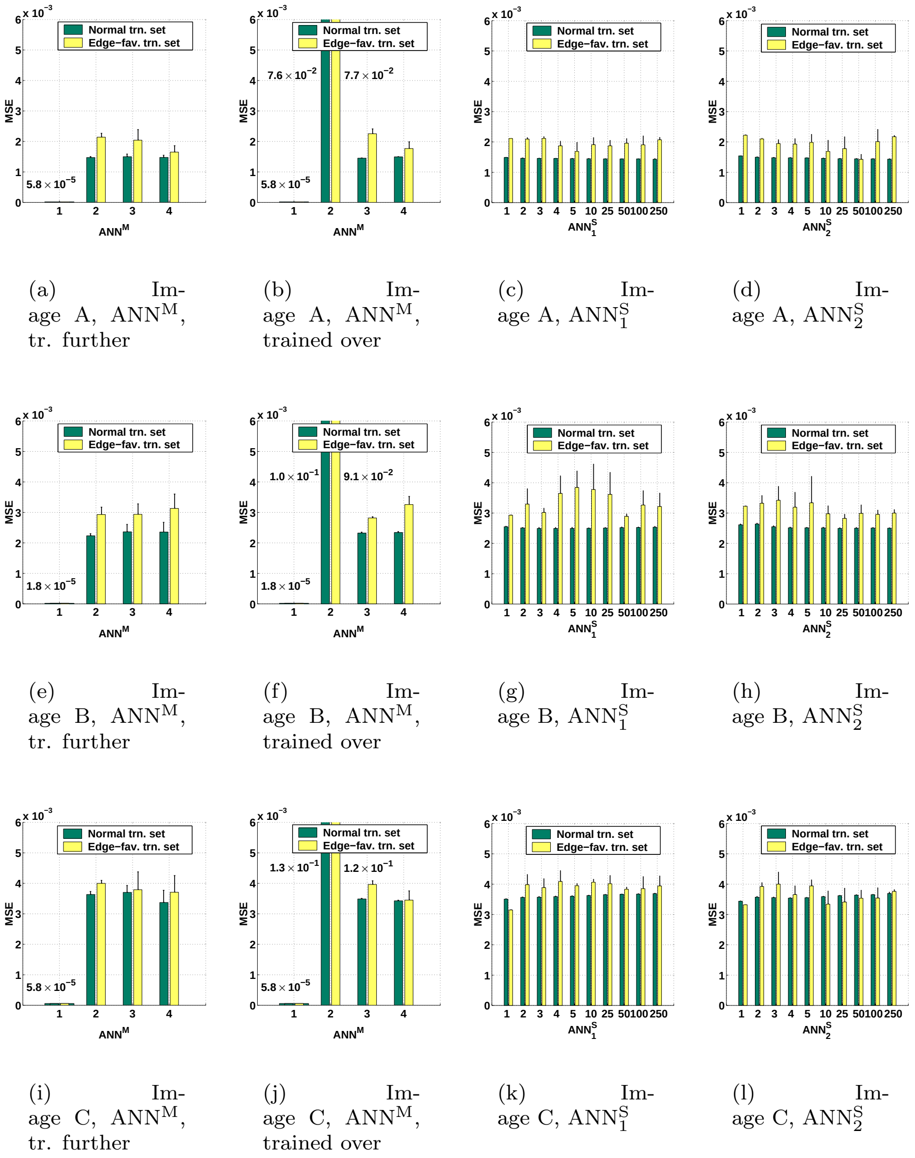
Finally, although all performance measures used in this section suggest that ANNs perform poorly in edge-preserving smoothing, the perceptual quality of the resulting filtered images is quite good. Perhaps it is the very fact that these ANNs have only partially succeeded in capturing the nonlinearity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

7 Conclusions

This paper discussed the application of neural networks in image processing. Three main questions were formulated in the introduction:

- *Applicability*: can (nonlinear) image processing operations be learned by adaptive methods?
- *Prior knowledge*: how can prior knowledge be used in the construction and training of adaptive methods?
- *Interpretability*: what can be learned from adaptive methods trained to solve image processing problems?

Below, answers will be formulated to each of the questions.



(a) Image A, ANN^M, tr. further

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

(c) Image A, ANN^S_1

(d) Image A, ANN^S_2

(e) Image B, ANN^M, tr. further

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

(g) Image B, ANN^S_1

(h) Image B, ANN^S_2

(i) Image C, ANN^M, tr. further

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

(k) Image C, ANN^S_1

(l) Image C, ANN^S_2

Figure 35: Performance of all ANN^Ms and ANN^Ss on the three images used: (a)-(d) on Image A (fig. 26 (a)), (e)-(h) on Image B (fig. 26 (b)) and (i)-(l) on Image C (fig. 26 (c)).

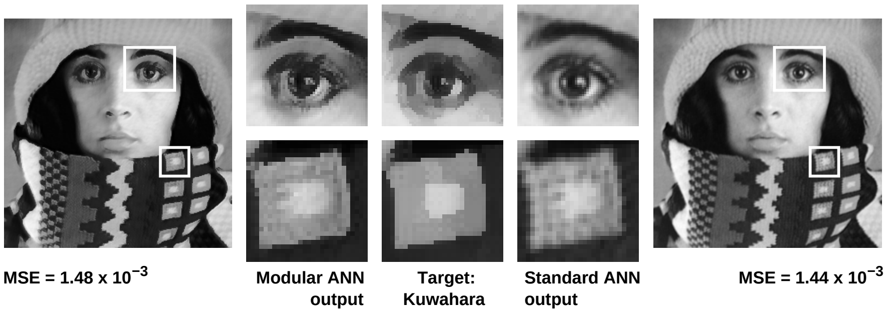
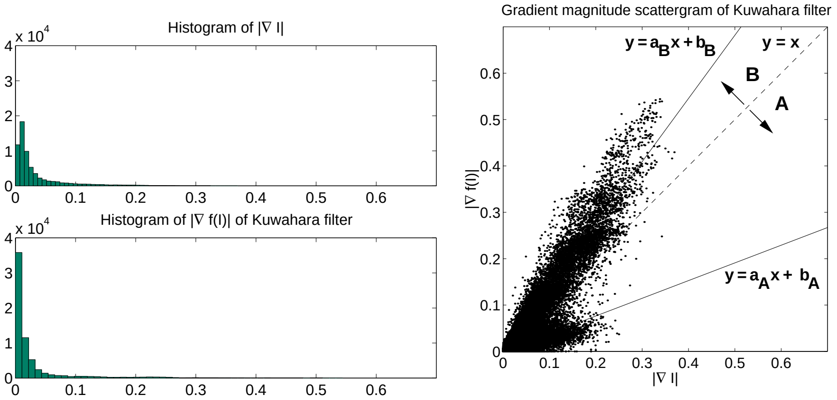
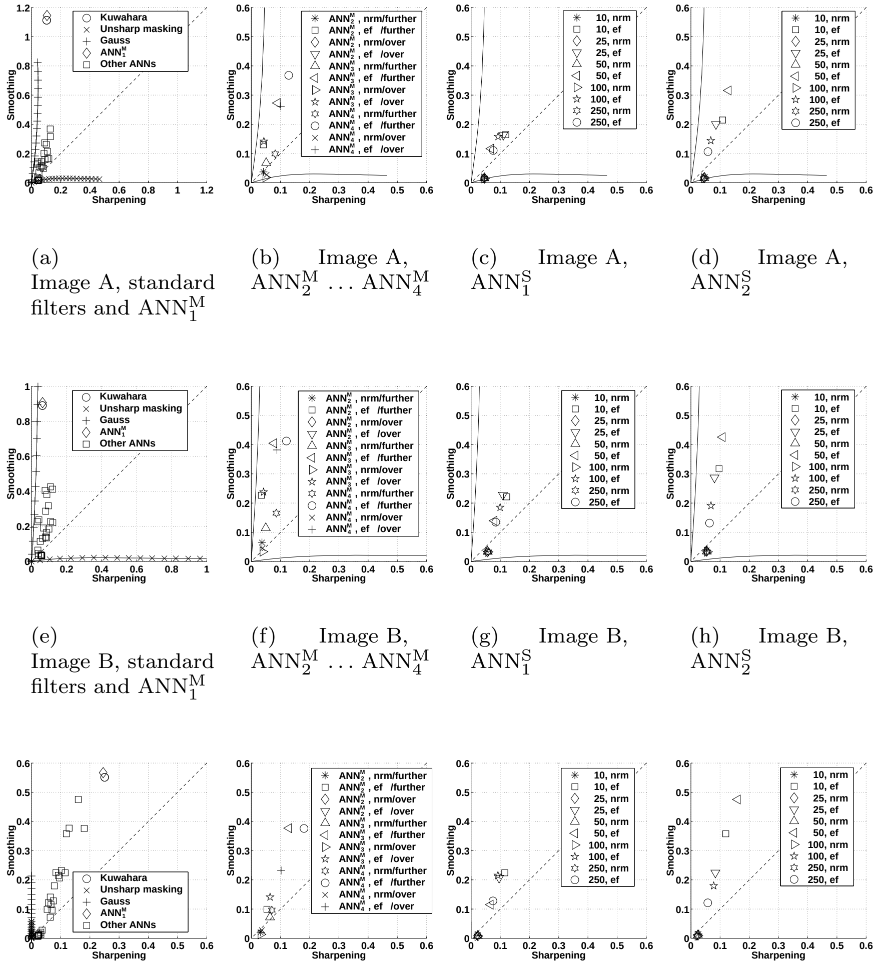


Figure 36: Two ANN output images with details. For the left image, output of ANN_4^{M} trained on the edge-favouring set, the MSE is 1.48×10^{-3}; for the right image, output of $\text{ANN}_{1\times100}^{\text{S}}$ trained on a normally sampled set, it is 1.44×10^{-3}. The details in the middle show the target output of the Kuwahara filter; the entire target image is shown in figure 26 (a).



(a) (b)

Figure 37: (a) Histograms of gradient magnitude values $|\nabla I|$ of Image A (figure 26 (a)) and a Kuwahara filtered version ($k = 3$). (b) Scattergram of the gradient magnitude image pixel values with estimated lines.



(a) Image A, standard filters and ANN$_1^{\mathrm{M}}$

(b) Image A, ANN$_2^{\mathrm{M}}$ ... ANN$_4^{\mathrm{M}}$

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

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

(e) Image B, standard filters and ANN$_1^{\mathrm{M}}$

(f) Image B, ANN$_2^{\mathrm{M}}$ ... ANN$_4^{\mathrm{M}}$

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

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

(i) Image C, standard filters and ANN$_1^{\mathrm{M}}$

(j) Image C, ANN$_2^{\mathrm{M}}$ ... ANN$_4^{\mathrm{M}}$

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

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

Figure 38: Performance of standard filters, all ANN$^{\mathrm{M}}$s and ANN$^{\mathrm{S}}$s on the three images used: (a)-(d) on Image A (fig. 26 (a)), (e)-(h) on Image B (fig. 26 (b)) and (i)-(l) on Image C (fig. 26 (c)). In the legends, *ef* stands for ANNs trained on edge-favouring data sets, as opposed to normally sampled data sets (*nrm*); *further* indicates ANNs initialised by training the individual modules as opposed to ANNs trained from scratch (*over*); and 10, 25 and so on denote the number of units per hidden layer.

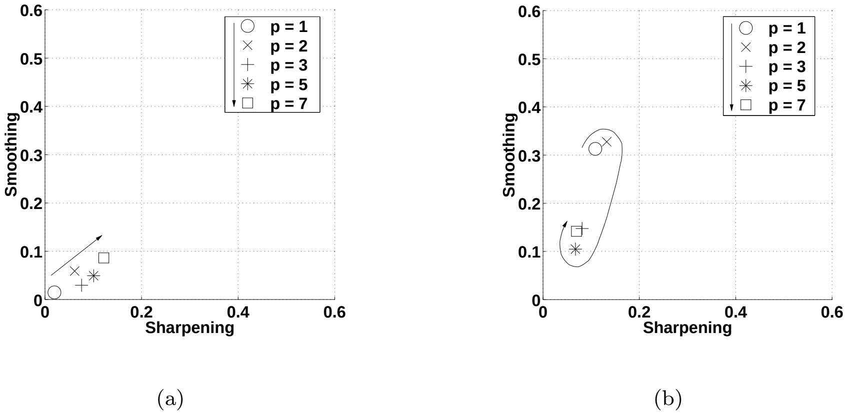
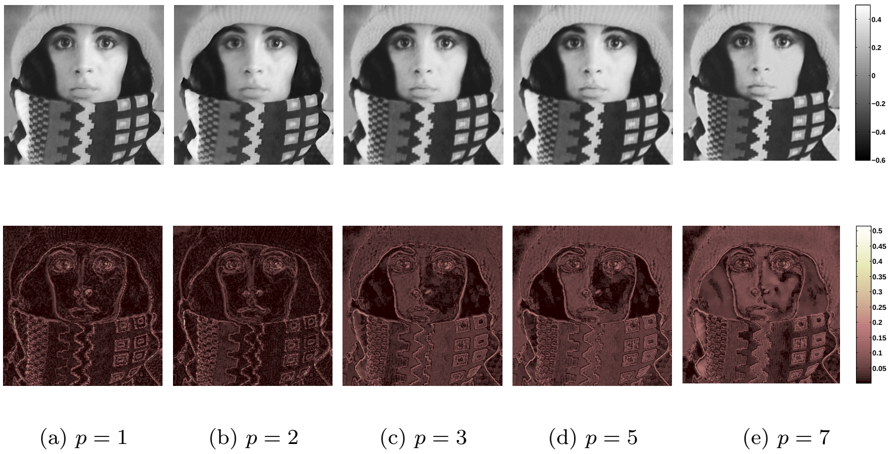
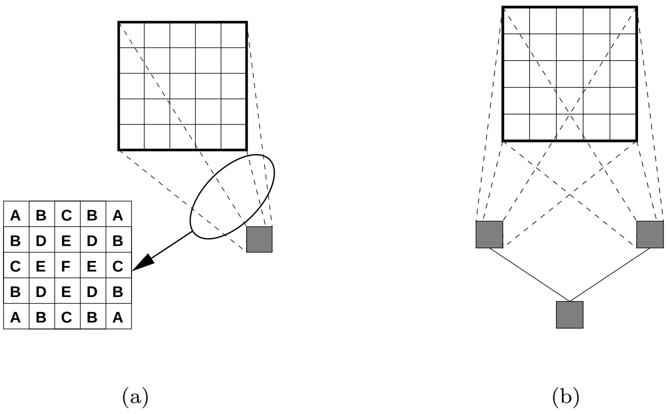


Figure 39: Performance of $\text{ANN}^{\text{S}}_{2\times50}$ on Image A (fig. 26 (a)), trained using different L_p norm error criteria and (a) the normal training set and (b) the edge-favouring training set.



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

Figure 40: Top row: output of $\text{ANN}^{\text{S}}_{2\times50}$ trained using the L_p norm on the edge-favouring data set, for various p. Bottom row: absolute difference between output and target image.

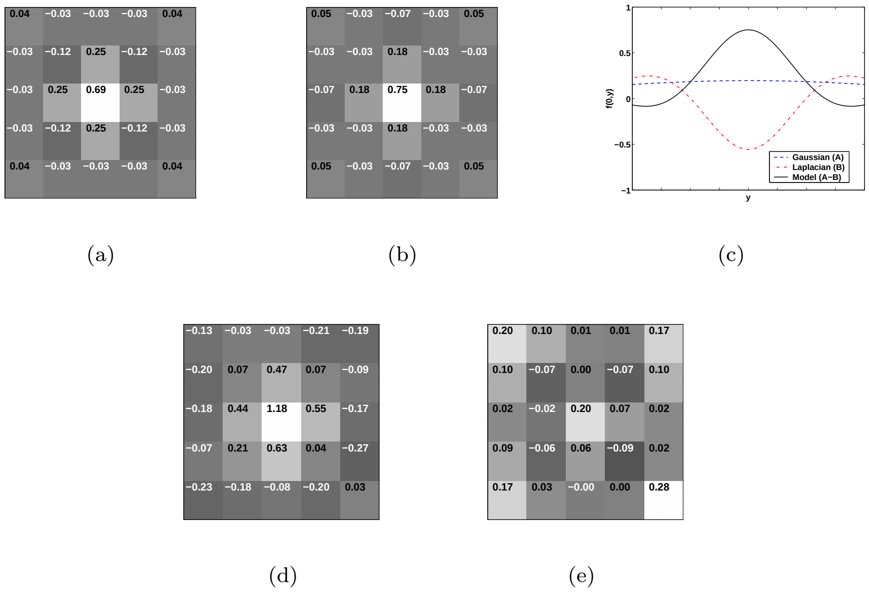


A	B	C	B	A
B	D	E	D	B
C	E	F	E	C
B	D	E	D	B
A	B	C	B	A

(a)

(b)

Figure 41: (a) ANN$^{\mathrm{K}}$, the simplest linear ANN to perform a Kuwahara filtering: a 5×5 unit input layer and one output unit without bias. The ANN contains 6 independent weights indicated in the mask by the letters **A** through **F**. (b) ANN$^{\mathrm{S}}_{1\times2}$: two hidden units, no mask (i.e., no restrictions).



0.04	−0.03	−0.03	−0.03	0.04
−0.03	−0.12	0.25	−0.12	−0.03
−0.03	0.25	0.69	0.25	−0.03
−0.03	−0.12	0.25	−0.12	−0.03
0.04	−0.03	−0.03	−0.03	0.04

(a)

0.05	−0.03	−0.07	−0.03	0.05
−0.03	−0.03	0.18	−0.03	−0.03
−0.07	0.18	0.75	0.18	−0.07
−0.03	−0.03	0.18	−0.03	−0.03
0.05	−0.03	−0.07	−0.03	0.05

(b)

(c)

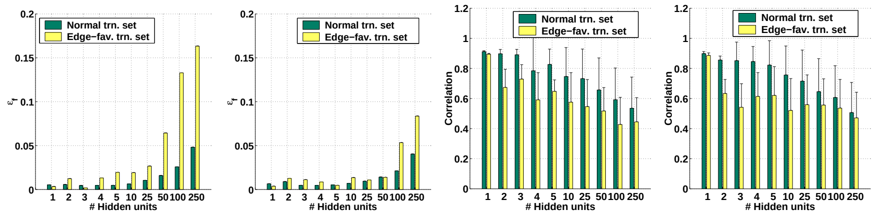
−0.13	−0.03	−0.03	−0.21	−0.19
−0.20	0.07	0.47	0.07	−0.09
−0.18	0.44	1.18	0.55	−0.17
−0.07	0.21	0.63	0.04	−0.27
−0.23	−0.18	−0.08	−0.20	0.03

(d)

0.20	0.10	0.01	0.01	0.17
0.10	−0.07	0.00	−0.07	0.10
0.02	−0.02	0.20	0.07	0.02
0.09	−0.06	0.06	−0.09	0.02
0.17	0.03	−0.00	0.00	0.28

(e)

Figure 42: (a) Weights found in ANN$^{\mathrm{K}}$ (fig. 41 (a)). (b) Weights generated by the fitted model (eqn. 31: $c_1 = 10.21$, $\sigma_1 = 2.87$, $c_2 = 3.41$, $\sigma_2 = 0.99$). (c) A cross section of this model at $x = 0$. (d), (e) Weight matrices found in ANN$^{\mathrm{S}}_{1\times2}$ (fig. 41 (b)) trained using DCGD.



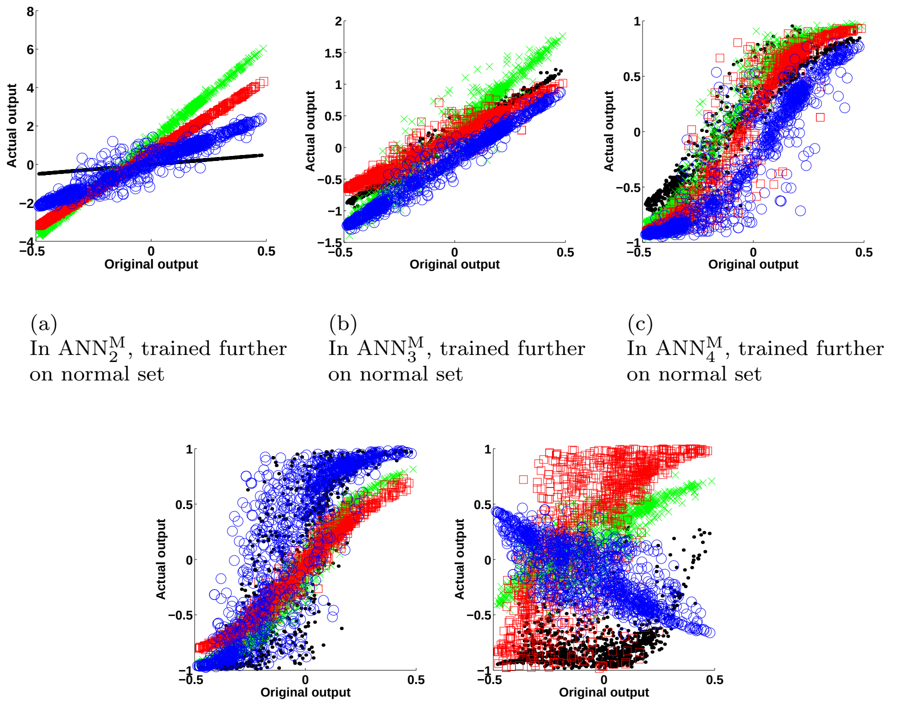
(a) ANN$^{\mathrm{S}}_1$, median ε_{f}

(b) ANN$^{\mathrm{S}}_2$, median ε_{f}

(c) ANN$^{\mathrm{S}}_1$, average C

(d) ANN$^{\mathrm{S}}_2$, average C

Figure 43: A comparison between the actual weights in ANN$^{\mathrm{S}}$s and the fitted models, for both ANN$^{\mathrm{S}}_1$s and ANN$^{\mathrm{S}}_2$s. The median ε_{f} is shown in (a) and (b) as the average ε_{f} is rather uninformative due to outliers.



(a)
In ANN$_2^{\mathrm{M}}$, trained further on normal set

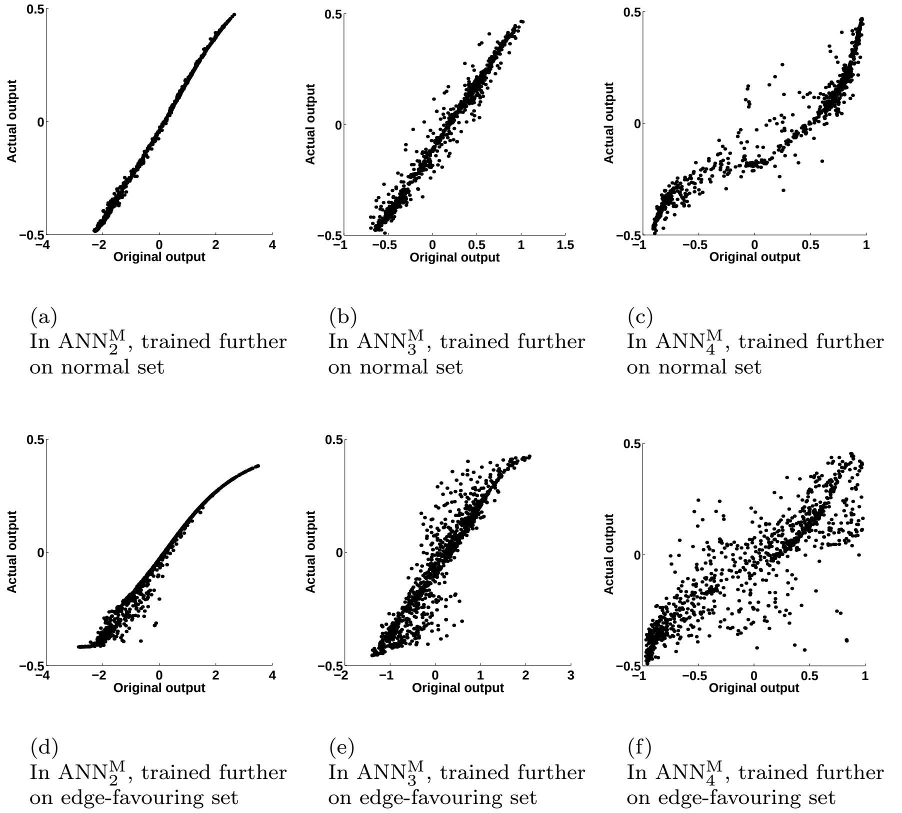
(b)
In ANN$_3^{\mathrm{M}}$, trained further on normal set

(c)
In ANN$_4^{\mathrm{M}}$, trained further on normal set

(d)
In ANN$_4^{\mathrm{M}}$, trained further on edge-favouring set

(e)
In ANN$_4^{\mathrm{M}}$, trained over on edge-favouring set

Figure 44: Plots of outputs of the four MOD$^{\mathrm{Avg}}$s before concatenation against outputs of the same modules after concatenation and training further or over. Different markers indicate different output units. The plots show progressively more freedom as the modules become less restricted ((a)-(c)) and an increase in nonlinearity when modules are trained on the edge-favouring data set ((a)-(c) vs. (d)-(e)).



(a)
In $\mathrm{ANN}_2^{\mathrm{M}}$, trained further on normal set

(b)
In $\mathrm{ANN}_3^{\mathrm{M}}$, trained further on normal set

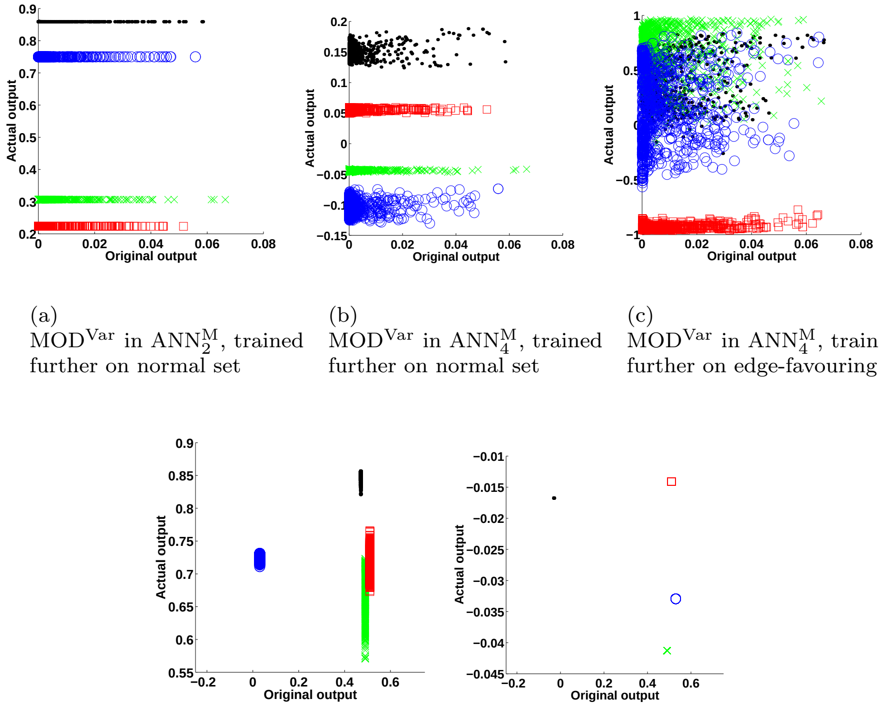
(c)
In $\mathrm{ANN}_4^{\mathrm{M}}$, trained further on normal set

(d)
In $\mathrm{ANN}_2^{\mathrm{M}}$, trained further on edge-favouring set

(e)
In $\mathrm{ANN}_3^{\mathrm{M}}$, trained further on edge-favouring set

(f)
In $\mathrm{ANN}_4^{\mathrm{M}}$, trained further on edge-favouring set

Figure 45: Plots of $\mathrm{MOD}^{\mathrm{Sel}}$ outputs before concatenation against $\mathrm{MOD}^{\mathrm{Sel}}$ outputs after concatenation and training further or over. The plots show progressively more freedom as the modules become less restricted ((a)-(c), (d)-(f)) and an increase in nonlinearity when modules are trained on the edge-favouring data set ((a)-(c) v. (d)-(f)).



(a)
MOD$^{\mathrm{Var}}$ in ANN$_2^{\mathrm{M}}$, trained further on normal set

(b)
MOD$^{\mathrm{Var}}$ in ANN$_4^{\mathrm{M}}$, trained further on normal set

(c)
MOD$^{\mathrm{Var}}$ in ANN$_4^{\mathrm{M}}$, trained further on edge-favouring set

(d)
MOD$^{\mathrm{Pos}}$ in ANN$_4^{\mathrm{M}}$, trained further on normal set

(e)
MOD$^{\mathrm{Pos}}$ in ANN$_4^{\mathrm{M}}$, trained over on normal set

Figure 46: Plots of MOD$^{\mathrm{Var}}$ and MOD$^{\mathrm{Pos}}$ outputs before concatenation against the same outputs after concatenation and training further or over. Different markers indicate different output units. The plots show many module outputs in the concatenated ANNs are clamped at certain values. Note that in the latter two figures, the original output is either 0.0 or 0.5; a small offset has been added for the different units for presentation purposes.

7.1 Applicability

The overview in section 2 discussed how many researchers have attempted to apply artificial neural networks (ANNs) to image processing problems. To a large extent, it is an overview of what can now perhaps be called the "neural network hype" in image processing: the approximately 15-year period following the publications of Kohonen, Hopfield and Rumelhart et al. Their work inspired many researchers to apply ANNs to their own problem in any of the stages of the image processing chain. In some cases, the reason was biological plausibility; however, in most cases the goal was simply to obtain well-performing classification, regression or clustering methods.

In some of these applications the most interesting aspect of ANNs, the fact that they can be trained, was not (or only partly) used. This held especially for applications to the first few tasks in the image processing chain: pre-processing and feature extraction. Another advantage of ANNs often used to justify their use is the ease of hardware implementation; however, in most publications this did not seem to be the reason for application. These observations, and the fact that often researchers did not compare their results to established techniques, casted some doubt on the actual advantage of using ANNs. In the remainder of the paper, ANNs were therefore trained on two tasks in the image processing chain: object recognition (supervised classification), and pre-processing (supervised regression) and, where possible, compared to traditional approaches.

The experiment on supervised classification, in handwritten digit recognition, showed that ANNs are quite capable of solving difficult object recognition problems. They performed (nearly) as well as some traditional pattern recognition methods, such as the nearest neighbour rule and support vector classifiers, but at a fraction of the computational cost.

As supervised regressors, a number of ANN architectures was trained to mimic the Kuwahara filter, a nonlinear edge-preserving smoothing filter used in pre-processing. The experiments showed that careful construction of the training set is very important. If filter behaviour is critical only in parts of the image represented by a small subset of the training set, this behaviour will not be learned. Constructing training sets using the knowledge that the Kuwahara filter is at its most nonlinear around edges improved performance considerably. This problem is also due to the use of the mean squared error (MSE) as a training criterion, which will allow poor performance if it only occurs for a small number of samples. Another problem connected with the use of the MSE is that it is insufficiently discriminative for model choice; in first attempts, almost all ANN architectures showed identical MSEs on test images. Criteria which were proposed to measure smoothing and sharpening performance showed larger differences. Unfortunately, these results indicate that the training set and performance measure will have to be tailored for each specific application, with which ANNs lose much of their attractiveness as all-round methods. The findings also explain why in the literature, many ANNs applied to pre-processing were non-adaptive.

In conclusion, ANNs seem to be most applicable for problems requiring a nonlinear solution, for which there is a clear, unequivocal performance criterion. This means ANNs are more suitable for high-level tasks in the image processing chain, such as object recognition, rather than low-level tasks. For both classification and regression, the choice of architecture, the performance criterion and data set construction play a large role and will have to be optimised for each application.

7.2 Prior knowledge

In many publications on the use of ANNs in image processing, prior knowledge was used to constrain ANNs. This is to be expected; unconstrained ANNs contain large numbers of parameters and run a high risk of being overtrained. Prior knowledge can be used to lower the number of parameters in a way which does not restrict the ANN to such an extent that it can no longer perform the desired function. One way to do this is to construct modular architectures, in which use is made of the knowledge that an operation is best performed as a number of individual sub-operations. Another way is to use the knowledge that neighbouring pixels are related and should be treated in the same way, e.g. by using receptive fields in shared weights ANN.

The latter idea was tested in supervised classification, i.e. object recognition. The shared weight ANNs used contain several layers of feature maps (detecting features in a shift-invariant way) and subsampling maps (combining information gathered in previous layers). The question is to what extent this prior knowledge was truly useful. Visual inspection of trained ANNs revealed little. Standard feed-forward ANNs comparable in the number of *connections* (and therefore the amount of computation involved), but with a much larger number of *weights*, performed as well as the shared weight ANNs. This proves that the prior knowledge was indeed useful in lowering the number of parameters without affecting performance. However, it also indicates that training a standard ANN with more weights than required does not necessarily lead to overtraining.

For supervised regression, a number of modular ANNs was constructed. Each module was trained on a specific subtask in the nonlinear filtering problem the ANN was applied to. Furthermore, of each module different versions were created, ranging from architectures specifically designed to solve the problem (using hand-set weights and tailored transfer functions) to standard feed-forward ANNs. According to the proposed smoothing and sharpening performance measures, the fully hand-constructed ANN performed best. However, when the hand-constructed ANNs were (gradually) replaced by more standard ANNs, performance quickly decreased and became level with that of some of the standard feed-forward ANNs. Furthermore, in the modular ANNs that performed well the modular initialisation was no longer visible (see also the next section). The only remaining advantage of a modular approach is that careful optimisation of the number of hidden layers and units, as for the standard ANNs, is not necessary.

These observations lead to the conclusion that prior knowledge can be used to restrict adaptive methods in a useful way. However, various experiments showed that feed-forward ANNs are not natural vehicles for doing so, as this prior knowledge will have to be translated into a choice for ANN size, connectivity, transfer functions etc., parameters which do not have any physical meaning related to the problem. Therefore, such a translation does not necessarily result in an optimal ANN. It is easier to construct a (rough) model of the data and allow model variation by allowing freedom in a number of well-defined parameters. Prior knowledge should be used in constructing models rather than in molding general approaches.

7.3 Interpretability

Throughout this paper, strong emphasis was placed on the question whether ANN operation could be inspected after training. Rather than just applying ANNs, the goal was to learn from the way in which they solved a problem. In few publications this plays a large role, although it would seem to be an important issue when ANNs are applied in mission-critical systems, e.g. in medicine, process control or defensive systems.

Supervised classification ANNs were inspected w.r.t. their feature extraction capabilities. As feature extractors, shared weight ANNs were shown to perform well, since standard pattern recognition algorithms trained on extracted features performed better than on the original images. Unfortunately, visual inspection of trained shared weight ANNs revealed nothing. The danger here is of over-interpretation, i.e. reading image processing operations into the ANN which are not really there. To be able to find out what features are extracted, two smaller problems were investigated: edge recognition and two-class handwritten digit recognition. A range of ANNs was built, which showed that ANNs need not comply with our ideas of how such applications should be solved. The ANNs took many “short cuts”, using biases and hidden layer-output layer weights. Only after severely restricting the ANN did it make sense in terms of image processing primitives. Furthermore, in experiments on an ANN with two feature maps the ANN was shown to distribute its functionality over these maps in an unclear way. An interpretation tool, the decorrelating conjugate gradient algorithm (DCGD), can help in distributing functionality more clearly over different ANN parts. The findings lead to the formulation of the interpretability trade-off, between realistic yet hard-to-interpret experiments on the one hand and easily interpreted yet non-representative experiments on the other.

This interpretability trade-off returned in the supervised regression problem. Modular ANNs constructed using prior knowledge of the filtering algorithm performed well, but could not be interpreted anymore in terms of the individual sub-operations. In fact, retention of the modular initialisation was negatively correlated to performance. ANN error evaluation was shown to be a useful tool in gaining understanding of where the ANN fails; it showed that filter operation was poorest around edges. The DCGD algorithm was then used to find out why: most of the standard feed-forward ANNs found a sub-optimal linear approximation to the Kuwahara filter. The conclusion of the experiments on supervised classification and regression is that as long as a distributed system such as an ANN is trained on single goal, i.e. minimisation of prediction error, the operation of sub-systems cannot be expected to make sense in terms of traditional image processing operations. This held for both the receptive fields in the shared weight ANNs and the modular setup of the regression ANNs: although they are there, they are not necessarily used as such. This also supports the conclusion of the previous section, that the use of prior knowledge in ANNs is not straightforward.

This paper showed that interpretation of supervised ANNs is hazardous. As large distributed systems, they can solve problems in a number of ways, not all of which necessarily correspond to human approaches to these problems. Simply opening the black box at some location one expects the ANN to exhibit certain behaviour does not give insight into the overall operation. Furthermore, knowledge obtained from the ANNs cannot be used in any other systems, as it only makes sense in the precise setting of the ANN itself.

7.4 Conclusions

We believe that the last few years have seen an attitude change towards ANNs, in which ANNs are not anymore automatically seen as the best solution to any problem. The field of ANNs has to a large extent been re-incorporated in the various disciplines that inspired it: machine learning, psychology and neurophysiology. In machine learning, researchers are now turning towards other, non-neural adaptive methods, such as the support vector classifier. For them the ANN has become a tool, rather than *the* tool it was originally thought to be.

So when are ANNs useful in image processing? First, they are interesting tools when there is a real need for a fast parallel solution. Second, biological plausibility may be a factor for some researchers. But most importantly, ANNs trained based on examples can be valuable when a problem is too complex to construct an overall model based on knowledge only. Often, real applications consist of several individual modules performing tasks in various steps of the image processing chain. A neural approach can combine these modules, control each of them and provide feedback from the highest level to change operations at the lowest level. The price one pays for this power is the black-box character, which makes interpretation difficult, and the problematic use of prior knowledge. If prior knowledge is available, it is better to use this to construct a model-based method and learn its parameters; performance can be as good, and interpretation comes natural.

References

- Anand, R., Mehrotra, K., Mohan, C., and Ranka, S. (1995). Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*. **6**(1), 117–124.

- Bengio, Y. (1996). Neural networks for speech and sequence recognition. International Thompson Computer Press, Boston, MA.

- Bengio, Y., Le Cun, Y., and Henderson, D. (1994). Globally trained handwritten word recognizer using spatial representation, space displacement neural networks and hidden Markov models. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, Cambridge, MA.

- Bishop, C. (1995). Neural networks for pattern recognition. Oxford University Press, Oxford.

- Burrascano, P. (1991). A norm selection criterion for the generalized delta rule. *IEEE Transactions on Neural Networks*. **2**(1), 125–130.

- Ciesielski, V., Zhu, J., Spicer, J., and Franklin, C. (1992). A comparison of image processing techniques and neural networks for an automated visual inspection problem. In Adams, A. and Sterling, L., editors, *Proc. 5^{th} Joint Australian Conference on Artificial Intelligence*, pages 147–152, Singapore. World Scientific.

- De Boer, C. and Smeulders, A. (1996). Bessi: an experimentation system for vision module evaluation. In *Proc. 13th IAPR International Conference on Pattern Recognition (ICPR'96)*, volume C, pages 109–113, Los Alamitos, CA. IAPR, IEEE Computer Society Press.

- de Ridder, D. (1996). Shared weights neural networks in image analysis. Master's thesis, Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology.

- de Ridder, D. (2001). Adaptive methods of image processing. PhD thesis, Delft University of Technology, Delft.

- de Ridder, D., Duin, R., Verbeek, P., and Vliet, L. v. (1999). A weight set decorrelating training algorithm for neural network interpretation and symmetry breaking. In Ersbøll, B. and Johansen, P., editors, *Proc. 11th Scandinavian Conference on Image Analysis (SCIA'99), Vol. 2*, pages 739–746, Copenhagen, Denmark. DSAGM (The Pattern Recognition Society of Denmark), DSAGM.

- Devijver, P. and Kittler, J. (1982). Pattern recognition, a statistical approach. Prentice-Hall, London.

- Dijk, J., de Ridder, D., Verbeek, P., Walraven, J., Young, I., and van Vliet, L. (1999). A new measure for the effect of sharpening and smoothing filters on images. In Ersbøll, B. and Johansen, P., editors, *Proc. 11th Scandinavian Conference on Image Analysis (SCIA'99), Vol. 1*, pages 213–220, Copenhagen, Denmark. DSAGM (The Pattern Recognition Society of Denmark), DSAGM.

- Egmont-Petersen, M., Dassen, W., Kirchhof, C., Heijmeriks, J., and Ambergen, A. (1998a). An explanation facility for a neural network trained to predict arterial fibrillation directly after cardiac surgery. In *Computers in Cardiology 1998*, pages 489–492, Cleveland. IEEE.

- Egmont-Petersen, M., de Ridder, D., and Handels, H. (2002). Image processing using neural networks – a review. *Pattern Recognition*. **35**(10), 119–141.

- Egmont-Petersen, M., Talmon, J., Hasman, A., and Ambergen, A. (1998b). Assessing the importance of features for multi-layer perceptrons. *Neural Networks*. **11**(4), 623–635.

- Fahlman, S. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, Los Altos, CA.

- Fogel, D. (1991). An information criterion for optimal neural network selection. *IEEE Transactions on Neural Networks*. **2**(5), 490–497.

- Fogelman Soulie, F., Viennet, E., and Lamy, B. (1993). Multi-modular neural network architectures: applications in optical character and human face recognition. *International Journal of Pattern Recognition and Artificial Intelligence*. **7**(4), 721–755.

- Fukunaga, K. (1990). Introduction to statistical pattern recognition. Academic Press, New York, NY, 2^{nd} edition.

- Fukushima, K. and Miyake, S. (1982). Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*. **15**(6), 455–469.

- Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*. **2**(3), 183–192.

- Gader, P., Miramonti, J., Won, Y., and Coffield, P. (1995). Segmentation free shared weight networks for automatic vehicle detection. *Neural Networks*. **8**(9), 1457–1473.

- Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias-variance dilemma. *Neural Computation*. **4**(1), 1–58.

- Gonzalez, R. and Woods, R. (1992). Digital image processing. Addison-Wesley, Reading, MA.

- Gorman, R. and Sejnowski, T. (1988). Analysis of the hidden units in a layered network trained to classify sonar targets. *Neural Networks*. **1**(1), 75–89.

- Green, C. (1998). Are connectionist models theories of cognition? *Psycoloquy*. **9**(4).

- Greenhil, D. and Davies, E. (1994). Relative effectiveness of neural networks for image noise suppression. In Gelsema, E. and Kanal, L., editors, *Pattern Recognition in Practice IV*, pages 367–378, Vlieland. North-Holland.

- Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*. **3**(6), 671–692.

- Haralick, R. (1994). Performance characterization in computer vision. *Computer Vision, Graphics and Image Processing: Image understanding*. **60**(2), 245–249.

- Haykin, S. (1994). Neural networks: a comprehensive foundation. Macmillan College Publishing Co., New York, NY.

- Hertz, J., Krogh, A., and Palmer, R. G. (1991). Introduction to the theory of neural computation. Addison-Wesley, Reading, MA.

- Hinton, G., Dayan, P., and Revow, M. (1997). Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*. **8**(1), 65–74.

- Hinton, G., Sejnowski, T., and Ackley, D. (1984). Boltzmann machines: constraint satisfaction networks that learn. Technical Report Report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh, PA.

- Hoekstra, A., Kraaijveld, M., de Ridder, D., and Schmidt, W. (1996). The Complete SPRLIB & ANNLIB. Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology.

- Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.* **81**, 3088–3092.

- Hopfield, J. and Tank, D. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*. **52**(3), 141–152.

- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*. **2**(5), 359–366.

- Hubel, D. and Wiesel, T. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*. **160**, 106–154.

- Jacobs, R., Jordan, M., and Barto, A. (1991). Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks. *Cognitive Science*. **15**, 219–250.

- Katsulai, H. and Arimizu, N. (1981). Evaluation of image fidelity by means of the fidelogram and level mean-square error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **3**(3), 337–347.

- Kohonen, T. (1995). Self-organizing maps. Springer Series in Information Sciences. Springer-Verlag, Berlin.

- Kuwahara, M., Hachimura, K., Eiho, S., and Kinoshita, M. (1976). Digital processing of biomedical images, pages 187–203. Plenum Press, New York, NY.

- Lawrence, S., Giles, C., Tsoi, A., and Back, A. (1997). Face recognition - a convolutional neural-network approach. *IEEE Transactions on Neural Networks*. **8**(1), 98–113.

- Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989a). Backpropagation applied to handwritten zip code recognition. *Neural Computation*. **1**, 541 – 551.

- Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA.

- Le Cun, Y., Jackel, L. J., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989b). Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communication*. **27**(11), 41–46.

- Mathworks Inc. (2000). MATLAB release 11.1.

- McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*. **5**, 115–133.

- Melnik, O. and Pollack, J. (1998). Exact representations from feed-forward networks. Technical Report CS-99-205, Dept. of Computer Science, Volen National Center for Complex Systems, Brandeis University, Waltham, MA.

- Minsky, M. L. and Papert, S. (1969). Perceptrons. MIT Press, Cambridge, MA.

- Murata, N., Yoshizawa, S., and Amari, S. (1994). Network information criterion - determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*. **5**(6), 865–872.

- Nilsson, N. (1965). Learning machines. McGraw-Hill, New York, NY.

- Nowlan, S. and Platt, J. (1995). A convolutional neural network hand tracker. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 901–908. MIT Press, Cambridge, MA.

- Pal, N. and Pal, S. (1993). A review on image segmentation techniques. *Pattern Recognition*. **26**(9), 1277–1294.

- Perlovsky, L. (1998). Conundrum of combinatorial complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **20**(6), 666–670.

- Poggio, T. and Koch, C. (1985). Ill-posed problems in early vision: from computational theory to analogue networks. *Proceedings of the Royal Society London*. **B**(226), 303–323.

- Pratt, W. K. (1991). Digital image processing. John Wiley & Sons, New York, NY, 2^{nd} edition.

- Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). Numerical Recipes in C. Cambridge University Press, Cambridge, MA, 2^{nd} edition.

- Pugmire, R., Hodgson, R., and Chaplin, R. (1998). The properties and training of a neural network based universal window filter developed for image processing tasks. In Amari, S. and Kasabov, N., editors, *Brain-like computing and intelligent information systems*, chapter 3, pages 49–77. Springer-Verlag, Singapore.

- Raudys, S. (1998a). Evolution and generalization of a single neurone: I. Single-layer perceptron as seven statistical classifiers. *Neural Networks*. **11**(2), 283–296.

- Raudys, S. (1998b). Evolution and generalization of a single neurone: II. Complexity of statistical classifiers and sample size considerations. *Neural Networks*. **11**(2), 297–313.

- Richard, M. D. and Lippmann, R. P. (1991). Neural network classifiers estimate Bayesian posterior probabilities. *Neural Computation*. **3**(4), 461–483.

- Rosenblatt, F. (1962). Principles of neurodynamics. Spartan Books, New York, NY.

- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, pages 319–362. MIT Press, Cambridge, MA.

- Schenkel, M., Guyon, I., and Henderson, D. (1995). On-line cursive script recognition using time delay neural networks and hidden Markov models. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP'95)*, volume 2, page 637.

- Sejnowski, T. and Rosenberg, C. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*. **I**, 145–168.

- Setiono, R. (1997). Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*. **9**(1), 205–225.

- Setiono, R. and Liu, H. (1997). Neural network feature selector. *IEEE Transactions on Neural Networks*. **8**(3), 645–662.

- Shapire, R. (1990). The strength of weak learnability. *Machine Learning*. **5**(2), 197–227.

- Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, Philadelphia.

- Simard, P., LeCun, Y., and Denker, J. (1993). Efficient pattern recognition using a new transformation distance. In Hanson, S., Cowan, J., and Giles, L., editors, *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA.

- Solla, S. A. and Le Cun, Y. (1991). Constrained neural networks for pattern recognition. In Antognetti, P. and Milutinovic, V., editors, *Neural networks: concepts, applications and implementations*. Prentice Hall.

- Sontag, E. D. (1992). Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*. **3**(6), 981–990.

- Spreeuwers, L. J. (1992). Image filtering with neural networks, applications and performance evaluation. PhD thesis, Universiteit Twente, Enschede.

- Tickle, A., Andrews, R., Golea, M., and Diederich, J. (1998). The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*. **9**(6), 1057–1068.

- Vapnik, V. (1995). The nature of statistical learning theory. Springer-Verlag, Berlin.

- Verschure, P. (1996). Neural networks and a new AI, chapter Connectionist explanation: taking positions in the mind-brain dilemma, pages 133–188. Thompson, London.

- Viennet, E. (1993). Architectures Connexionistes Multi-Modulaires, Application à l'Analyse de Scène. PhD thesis, Université de Paris-Sud, Centre d'Orsay.

- Wilson, C. L. and Garris, M. D. (1992). Handprinted character database 3. National Institute of Standards and Technology; Advanced Systems division.

- Young, I., Gerbrands, J., and Van Vliet, L. (1998). The digital signal processing handbook, chapter Image processing fundamentals, pages 51/1 – 51/81. CRC Press/IEEE Press, Boca Raton, FL.