Sven Behnke

Hierarchical Neural Networks
for Image Interpretation

June 13, 2003

Foreword

It is my pleasure and privilege to write the foreword for this book, whose results I have been following and awaiting for the last few years. This monograph represents the outcome of an ambitious project oriented towards advancing our knowledge of the way the human visual system processes images, and about the way it combines high level hypotheses with low level inputs during pattern recognition. The model proposed by Sven Behnke, carefully exposed in the following pages, can be applied now by other researchers to practical problems in the field of computer vision and provides also clues for reaching a deeper understanding of the human visual system.

This book arose out of dissatisfaction with an earlier project: back in 1996, Sven wrote one of the handwritten digit recognizers for the mail sorting machines of the Deutsche Post AG. The project was successful because the machines could indeed recognize the handwritten ZIP codes, at a rate of several thousand letters per hour. However, Sven was not satisfied with the amount of expert knowledge that was needed to develop the feature extraction and classification algorithms. He wondered if the computer could be able to extract meaningful features by itself, and use these for classification. His experience in the project told him that forward computation alone would be incapable of improving the results already obtained. From his knowledge of the human visual system, he postulated that only a two-way system could work, one that could advance a hypothesis by focussing the attention of the lower layers of a neural network on it. He spent the next few years developing a new model for tackling precisely this problem.

The main result of this book is the proposal of a generic architecture for pattern recognition problems, called Neural Abstraction Pyramid (NAP). The architecture is *layered*, *pyramidal*, *competitive*, and *recurrent*. It is *layered* because images are represented at multiple levels of abstraction. It is *recurrent* because backward projections connect the upper to the lower layers. It is *pyramidal* because the resolution of the representations is reduced from one layer to the next. It is *competitive* because in each layer units compete against each other, trying to classify the input best. The main idea behind this architecture is letting the lower layers interact with the higher layers. The lower layers send some simple features to the upper layers, the uppers layers recognize more complex features and bias the computation in the lower layers. This in turn improves the input to the upper layers, which can refine their hypotheses, and so on. After a few iterations the network settles in the best interpretation. The architecture can be trained in supervised and unsupervised mode.

Here, I should mention that there have been many proposals of recurrent architectures for pattern recognition. Over the years we have tried to apply them to non-trivial problems. Unfortunately, many of the proposals advanced in the literature break down when confronted with non-toy problems. Therefore, one of the first advantages present in Behnke's architecture is *that it actually works*, also when the problem is difficult and really interesting for commercial applications.

The structure of the book reflects the road taken by Sven to tackle the problem of combining top-down processing of hypotheses with bottom-up processing of images. Part I describes the theory and Part II the applications of the architecture. The first two chapters motivate the problem to be investigated and identify the features of the human visual system which are relevant for the proposed architecture: retinotopic organization of feature maps, local recurrence with excitation and inhibition, hierarchy of representations, and adaptation through learning.

Chapter 3 gives an overview of several models proposed in the last years and provides a gentle introduction to the next chapter, which describes the NAP architecture. Chapter 5 deals with a special case of the NAP architecture, when only forward projections are used and features are learned in an unsupervised way. With this chapter, Sven came full circle: the digit classification task he had solved for mail sorting, using a hand-designed structural classifier, was outperformed now by an automatically trained system. This is a remarkable result, since much expert knowledge went into the design of the hand-crafted system.

Four applications of the NAP constitute Part II. The first application is the recognition of meter values (printed postage stamps), the second the binarization of matrix codes (also used for postage), the third is the reconstruction of damaged images, and the last is the localization of faces in complex scenes. The image reconstruction problem is my favorite regarding the kind of tasks solved. A complete NAP is used, with all its lateral, feed-forward and backward connections. In order to infer the original images from degraded ones, the network must learn models of the objects present in the images and combine them with models of typical degradations.

I think that it is interesting how this book started from a general inspiration about the way the human visual system works, how then Sven extracted some general principles underlying visual perception and how he applied them to the solution of several vision problems. The NAP architecture is what the Neocognitron (a layered model proposed by Fukushima the 1980s) aspired to be. It is the Neocognitron gotten right. The main difference between one and the other is the recursive nature of the NAP. Combining the bottom-up with the top-down approach allows for iterative interpretation of ambiguous stimuli.

I can only encourage the reader to work his or her way through this book. It is very well written and provides solutions for some technical problems as well as inspiration for neurobiologists interested in common computational principles in human and computer vision. The book is like a road that will lead the attentive reader to a rich landscape, full of new research opportunities.

Berlin, June 2003   Raúl Rojas

Preface

This thesis is published in partial fulfillment of the requirements for the degree of 'Doktor der Naturwissenschaften' (Dr. rer. nat.) at the Department of Mathematics and Computer Science of Freie Universität Berlin. Prof. Dr. Raúl Rojas (FU Berlin) and Prof. Dr. Volker Sperschneider (Osnabrück) acted as referees. The thesis was defended on November 27, 2002.

Summary of the Thesis

Human performance in visual perception by far exceeds the performance of contemporary computer vision systems. While humans are able to perceive their environment almost instantly and reliably under a wide range of conditions, computer vision systems work well only under controlled conditions in limited domains.

This thesis addresses the differences in data structures and algorithms underlying the differences in performance. The interface problem between symbolic data manipulated in high-level vision and signals processed by low-level operations is identified as one of the major issues of today's computer vision systems. This thesis aims at reproducing the robustness and speed of human perception by proposing a hierarchical architecture for iterative image interpretation.

I propose to use hierarchical neural networks for representing images at multiple abstraction levels. The lowest level represents the image signal. As one ascends these levels of abstraction, the spatial resolution of two-dimensional feature maps decreases while feature diversity and invariance increase. The representations are obtained using simple processing elements that interact locally. Recurrent horizontal and vertical interactions are mediated by weighted links. Weight sharing keeps the number of free parameters low. Recurrence allows to integrate bottom-up, lateral, and top-down influences.

Image interpretation in the proposed architecture is performed iteratively. An image is interpreted first at positions where little ambiguity exists. Partial results then bias the interpretation of more ambiguous stimuli. This is a flexible way to incorporate context. Such a refinement is most useful when the image contrast is low, noise and distractors are present, objects are partially occluded, or the interpretation is otherwise complicated.

The proposed architecture can be trained using unsupervised and supervised learning techniques. This allows to replace manual design of application-specific

computer vision systems with the automatic adaptation of a generic network. The task to be solved is then described using a dataset of input/output examples.

Applications of the proposed architecture are illustrated using small networks. Furthermore, several larger networks were trained to perform non-trivial computer vision tasks, such as the recognition of the value of postage meter marks and the binarization of matrixcodes. It is shown that image reconstruction problems, such as super-resolution, filling-in of occlusions, and contrast enhancement/noise removal, can be learned as well. Finally, the architecture was applied successfully to localize faces in complex office scenes. The network is also able to track moving faces.

Acknowledgements

Table of Contents

1. Introduction

1.1 Motivation

1.1.1 Importance of Visual Perception

Visual perception is important for both humans and computers. Humans are visual animals. Just imagine how loosing your sight would effect you to appreciate its importance. We extract most information about the world around us by seeing.

This is possible because photons sensed by the eyes carry information about the world. On their way from light sources to the photoreceptors they interact with objects and get altered by this process. For instance, the wavelength of a photon may reveal information about the color of a surface it was reflected from. Sudden changes in the intensity of light along a line may indicate the edge of an object. By analyzing intensity gradients, the curvature of a surface may be recovered. Texture or the type of reflection can be used to further characterize surfaces. The change of visual stimuli over time is an important source of information as well. Motion may indicate the change of an object's pose or reflect ego-motion. Synchronous motion is a strong hint for segmentation, the grouping of visual stimuli to objects because parts of the same object tend to move together.

Vision allows us to sense over long distance since light travels through the air without significant loss. It is non-destructive and, if no additional lighting is used, it is also passive. This allows for perception without being noticed.

Since we have a powerful visual system, we designed our environment to provide visual cues. Examples include marked lanes on the roads and traffic lights. Our interaction with computers is based on visual information as well. Large screens display the data we manipulate and printers produce documents for later visual perception.

Powerful computer graphic systems have been developed to feed our visual system. Today's computers include special-purpose processors for rendering images. They produce almost realistic perceptions of simulated environments.

On the other hand, the communication channel from the users to computers has a very low bandwidth. It consists mainly of the keyboard and a pointing device. More natural interaction with computers requires advanced interfaces, including computer vision components. Recognizing the user and perceiving his or her actions are key prerequisites for more intelligent user interfaces.

Computer vision, that is the extraction of information from images and image sequences, is also important for applications other than human-computer interaction. For instance, it can be used by robots to extract information from their environment. In the same way visual perception is crucial for us, it is for autonomous mobile robots acting in the world designed for us. A driver assistance system in a car, for example, must perceive all the signs and markings on the road, as well as other cars, pedestrians, and many more objects.

Computer vision techniques are also used for the analysis of static images. In medical imaging, for example, it can be used to aid the interpretation of images by a physician. Another application area is the automatic interpretation of satellite images. One particularly successful application of computer vision techniques is the reading of documents. Machines for check reading and mail sorting are widely used.

1.1.2 Performance of the Human Visual System

Human performance for visual tasks is impressive. The human visual system perceives stimuli of a high dynamic range. It works well in the brightest sunlight and still allows for orientation under limited lighting conditions, e.g. at night. It has been shown that we can even perceive single photons.

Under normal lighting, the system has high acuity. We are able to perceive object details and can recognize far-away objects. Humans can also perceive color. When presented next to each other, we can distinguish thousands of color nuances.

The visual system manages to separate objects from other objects and the background. We are also able to separate object-motion from ego-motion. This facilitates the detection of change in the environment.

One of the most remarkable features of the human visual system is its ability to recognize objects under transformations. Moderate changes in illumination, object pose, and size do not affect perception. Another invariance produced by the visual system is color constancy. By accounting for illumination changes, we perceive different wavelength mixtures as the same color. This inference process recovers the reflectance properties of surfaces, the object color. We are also able to tolerate deformations of non-rigid objects. Object categorization is another valuable property. If we have seen several examples of a category, say dogs, we can easily classify an unseen animal as dog if it has the typical dog features.

The human visual system is strongest for the stimuli that are most important to us: faces, for instance. We are able to distinguish thousands of different faces. On the other hand, we can recognize a person although he or she has aged, changed hair style and now wears glasses.

Human visual perception is not only remarkably robust to variances and noise, but it is fast as well. We need only about 100ms to extract the basic gist of a scene, we can detect targets in naturalistic scenes in 150ms, and we are able to understand complicated scenes within 400ms.

Visual processing is mostly done subconsciously. We do not perceive the difficulties involved in the task of interpreting natural stimuli. This does not mean that this task is easy. The challenge originates in the fact that visual stimuli are frequently
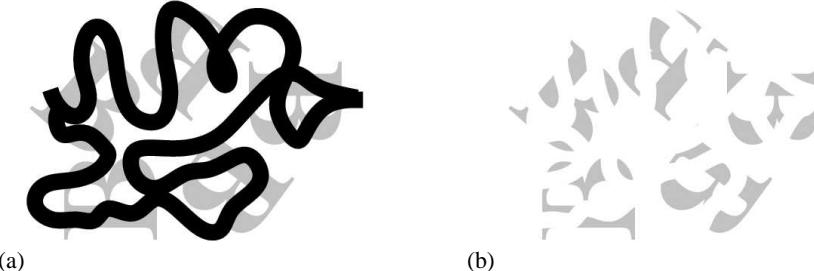
Fig. 1.1. Role of occluding region in recognition of occluded letters: (a) letters 'B' partially occluded by a black line; (b) same situation, but the occluding line is white (it merges with the background; recognition is much more difficult) (image from [164]).
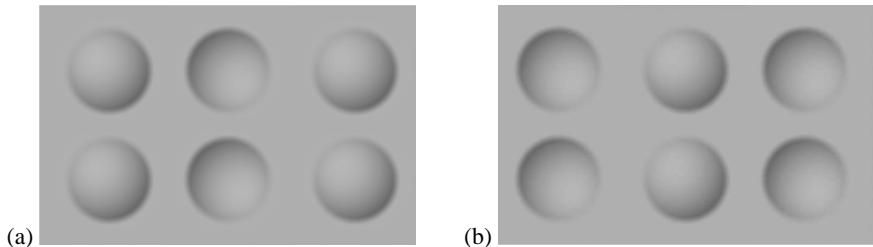
Fig. 1.2. Light-from-above assumption: (a) stimuli in the middle column are perceived as concave surfaces whereas stimuli on the sides appear to be convex; (b) rotation by 180° makes convex stimuli concave and vice versa.

ambiguous. Inferring three-dimensional structure from two-dimensional images, for example, is inherently ambiguous. Many 3D objects correspond to the same image. The visual system must rely on various depth cues to infer the third dimension. Another example is the interpretation of spatial changes in intensity. Among their potential causes are changes in the reflectance of an object's surface (e.g. texture), inhomogeneous illumination (e.g. at the edge of a shadow) and the discontinuity of the reflecting surface at the object borders.

Occlusions are a frequent source of ambiguity as well. Our visual system must guess what occluded object parts look like. This is illustrated in Figure 1.1. We are able to recognize the letters 'B', which are partially occluded by a black line. If the occluding line is white, the interpretation is much more challenging, because the occlusion is not detected and the 'guessing mode' is not employed.

Since the task of interpreting ambiguous stimuli is not well-posed, prior knowledge must be used for visual inference. The human visual system uses many heuristics to resolve ambiguities. One of the assumptions, the system relies on, is that light comes from above. Figure 1.2 illustrates this fact. Since the curvature of surfaces can be inferred from shading only up to the ambiguity of a convex or a concave interpretation, the visual system prefers the interpretation that is consistent with a light source located above the object. This choice is correct most of the time.
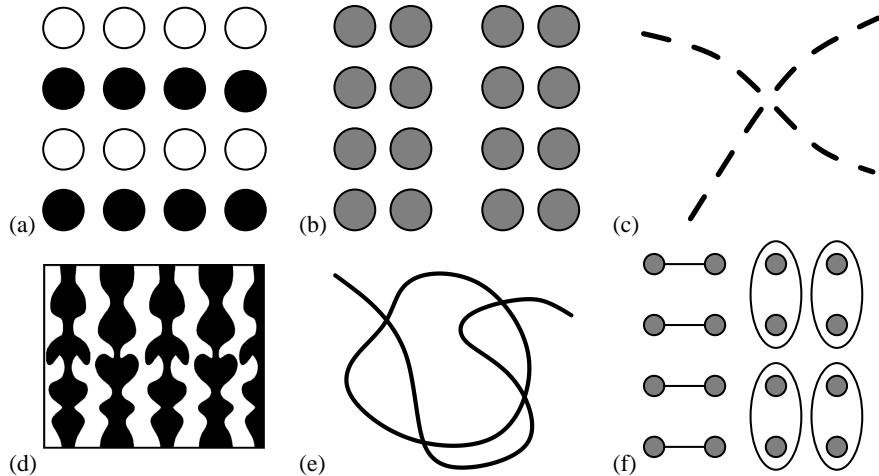
Fig. 1.3. Gestalt principles of perception [125]: (a) similar stimuli are grouped together; (b) proximity is another cue for grouping; (c) line segments are grouped based on good continuation; (d) symmetric contours form objects; (e) closed contours are more salient than open ones; (f) connectedness and belonging to a common region cause grouping.
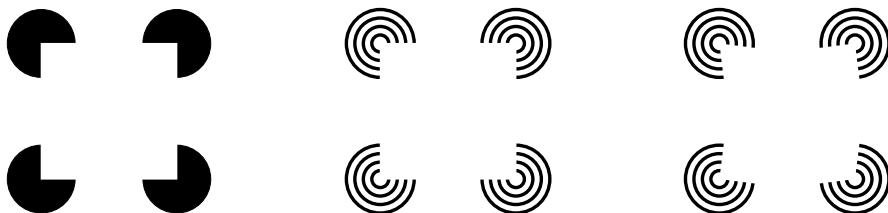
Fig. 1.4. Kanizsa figures [118]. Four inducers produce the percept of a white square partially occluding four black disks. Line endings induce illusory contours perpendicular to the lines. The square can be bent if the opening angles of the arcs are slightly changed.

Other heuristics are summarized by the Gestalt principles of perception [125]. Some of them are illustrated in Figure 1.3. Gestalt psychology emphasizes the Prägnanz of perception: stimuli group spontaneously into the simplest possible configuration. Examples include the grouping of similar stimuli (see Part (a)). Proximity is another cue for grouping (b). Line segments are connected based on good continuation (c). Symmetric or parallel contours indicate that they belong to the same object (d). Closed contours are more salient than open ones (e). Connectedness and belonging to a common region cause grouping as well (f). Last, but not least, common fate (synchrony in motion) is a strong hint that stimuli belong to the same object.

Although such heuristics are correct most of the time, sometimes they fail. This results in unexpected perceptions, called visual illusions. One example of these illusions are Kanizsa figures [118], shown in Figure 1.4. In the left part of the figure, four inducers produce the percept of a white square in front of black disks, because
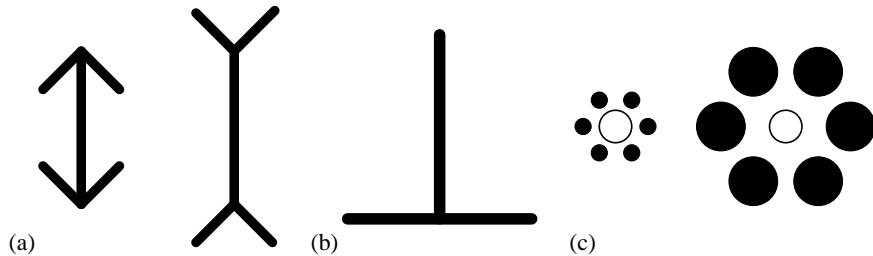
Fig. 1.5. Visual illusions: (a) Müller-Lyer illusion [163] (the vertical lines appear to have different lengths); (b) horizontal-vertical illusion (the vertical line appears to be longer than the horizontal one); (c) Ebbinghaus-Titchener illusion (the central circles appear to have different sizes).
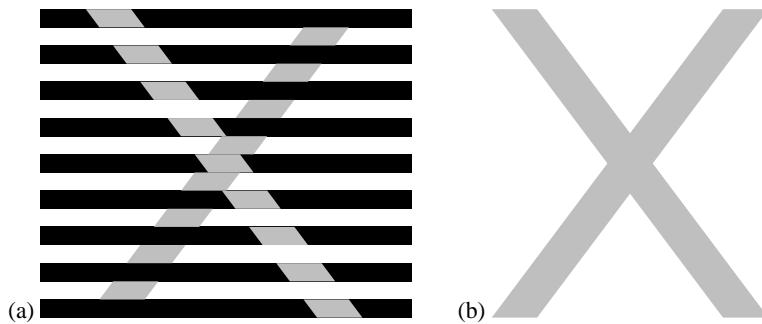
Fig. 1.6. Munker-White illusion [224] illustrates contextual effects of brightness perception: (a) both diagonals have the same brightness; (b) same situation without occlusion.

this interpretation is the simplest one. Illusory contours are perceived between the inducers, although there is no intensity change. The middle of the figure shows that virtual contours are also induced at line endings perpendicular to the lines because occlusions are likely causes of line endings. In the right part of the figure it is shown that one can even bend the square, if the opening angles of the arc segments are slightly changed.

Three more visual illusions are shown in Figure 1.5. In the Müller-Lyer illusion [163] (Part (a)), two vertical lines appear to have different lengths, although they are identical. This perception is caused by the different three-dimensional interpretation of the junctions at the line endings. The left line is interpreted as the convex edge of two meeting walls, whereas the right line appears to be a concave corner. Part (b) of the figure shows the horizontal-vertical illusion. The vertical line appears to be longer than the horizontal one, although both have the same length. In Part (c), the Ebbinghaus-Titchener illusion is shown. The perceived size of the central circle depends on the size of the black circles surrounding it.

Contextual effects of brightness perception are illustrated by the Munker-White illusion [224], shown in Figure 1.6. Two gray diagonals are partially occluded by a black-and-white pattern of horizontal stripes. The perceived brightness of the diagonals is very different, although they have the same reflectance. This illustrates that

THE CAT IN THE HAT

Fig. 1.7. Contextual effects of letter perception. The letters in the middle of the words 'THE', 'CAT', and 'HAT' are exact copies of each other. Depending on the context they are either interpreted as 'H' or as 'A'.

```
TT T TTT T TT  TT              Q              Q Q  QQ QQQ Q
 T TT  T TTT  T T          Q       Q           Q QQ  QQQ OQQ
T O T T TT  TTT          Q   Q  Q           QQQ  QQQ Q Q Q
T TTT T T  T TT T          Q    O  Q            Q QQ QQ Q QQQQ
  TT TTT  TT   TTT              Q              Q QQ  QQ QQQ
```

Fig. 1.8. Pop-out and sequential search. The letter 'O' in the left group of 'T's is very salient because the letters stimulate different features. It is much more difficult to find it amongst 'Q's that share similar features. Here, the search time depends on the number of distractors.

the visual system does not perceive absolute brightness, but constructs the brightness of an object by filling-in its area from relative brightness percepts that have been induced at its edges. Similar filling-in effects can be observed for color perception.

Figure 1.7 shows another example of contextual effects. Here, the context of an ambiguous letter decides whether it is interpreted as 'H' or as 'A'. The perceived letter is always the one that completes a word. A similar top-down influence is known as word-superiority effect, described first by Reicher [189]. The performance of letter perception is better in words than in non-words.

The human visual system uses a high degree of parallel processing. Targets that can be defined by a unique feature can be detected quickly, irrespective of the number of distractors. This visual 'pop-out' is illustrated in the left part of Figure 1.8. However, if the distractors share critical features with the target, as in the middle and the right part of the figure, search is slow and the detection time depends on the number of distractors. This is called sequential search. It shows that the visual system can focus its limited resources on parts of the incoming stimuli in order to inspect them closer. This is a form of attention.

Another feature of the human visual system is active vision. We do not perceive the world passively, but move our eyes, the head, or even the whole body in order to to improve the image formation process. This can help to disambiguate a scene. For example, we move the head sideways to look around an obstacle and we rotate objects to view them from multiple angles in order to facilitate 3D reconstruction.

1.1.3 Limitations of Current Computer Vision Systems

Computer vision systems consist of two main components: image capture and interpretation of the captured image. The capture part is usually not very problematic. 2D CCD image sensors with millions of pixels are available. Line cameras produce

Noise suppression Deblurring Image enhancement Edge detection	Compression Feature extraction	Texture segregation Colour recognition Clustering	Template matching Feature-based recognition	Scene analysis Object arrangement
Preprocessing	Data reduction	Segmentation	Object recognition	Image understanding
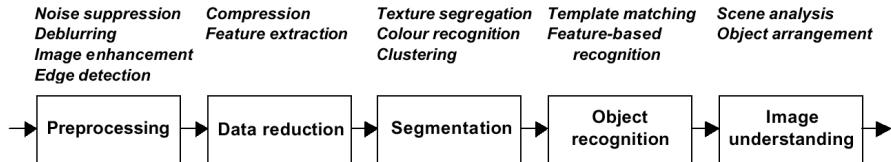
Fig. 1.9. Feed-forward image processing chain (image adapted from [61]).

images of even higher resolution. If a high dynamic range is needed, logarithmic image sensors need to be employed. For mobile applications, like cellular phones and autonomous robots, CMOS sensors can be used. They are small, inexpensive, and consume little power.

The more problematic part of computer vision is the interpretation of captured images. This problem has two main aspects: speed and quality of interpretation. Cameras and other image capture devices produce large amounts of data. Although the processing speed and storage capabilities of computers increased tremendously in the last decades, processing high-resolution images and video is still a challenging task for today's general-purpose computers. Limited computational power constrains image interpretation algorithms much more for mobile real-time applications then for offline or desktop processing. Fortunately, the continuing hardware development makes the prediction possible that these constraints will relax within the next years, in the same way as the constraints for processing less demanding audio signals relaxed already.

This may sound like one would only need to wait to see computers solve image interpretation problems faster and better than humans do, but this is not the case. While dedicated computer vision systems already outperform humans in terms of processing speed, the interpretation quality does not reach human level. Current computer vision systems are usually employed in very limited domains. Examples include quality control, license plate identification, ZIP code reading for mail sorting, and image registration in medical applications. All these systems include a possibility for the system to indicate lack of confidence, e.g. by rejecting ambiguous examples. These are then inspected by human experts. Such partially automated systems are useful though, because they free the experts from inspecting the vast majority of unproblematic examples. The need to incorporate a human component in such systems clearly underlines the superiority of the human visual system, even for tasks in such limited domains.

Depending on the application, computer vision algorithms try to extract different aspects of the information contained in an image or a video stream. For example, one may desire to infer a structural object model from a sequence of images that show a moving object. In this case, the object structure is preserved, while motion information is discarded. On the other hand, for the control of mobile robots, analysis may start with a model of the environment in order to match it with the input and to infer robot motion.

Two main approaches exist for the interpretation of images: bottom-up and top-down. Figure 1.9 depicts the feed-forward image processing chain of bottom-up
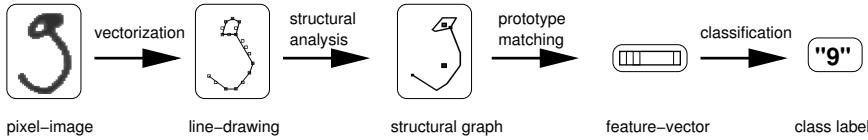
Fig. 1.10. Structural digit classification (image adapted from [21]). Information irrelevant for classification is discarded in each step while the class information is preserved.

analysis. It consists of a sequence of steps that transform one image representation into another. Examples for such transformations are edge detection, feature extraction, segmentation, template matching, and classification. Through these transformations, the representations become more compact, more abstract, and more symbolic. The individual steps are relatively small, but the nature of the representation changes completely from one end of the chain, where images are represented as two-dimensional signals to the other, where symbolic scene descriptions are used.

One example of a bottom-up system for image analysis is the structural digit recognition system [21], illustrated in Figure 1.10. It transforms the pixel-image of an isolated handwritten digit into a line-drawing, using a vectorization method. This discards information about image contrast and the width of the lines. Using structural analysis, the line-drawing is transformed into an attributed structural graph that represents the digit using components like curves and loops and their spatial relations. Small components must be ignored and gaps must be closed in order to capture the essential structure of a digit. This graph is matched against a database of structural prototypes. The match selects a specialized classifier. Quantitative attributes of the graph are compiled into a feature vector that is classified by a neural network. It outputs the class label and a classification confidence. While such a system does recognize most digits, it is necessary to reject a small fraction of the digits to achieve reliable classification.

The top-down approach to image analysis works the opposite direction. It does not start with the image, but with a database of object models. Hypotheses about the instantiation of a model are expanded to a less abstract representation by accounting, for example, for the object position and pose. The match between an expanded hypothesis and features extracted from the image is checked in order to verify or reject the hypothesis. If it is rejected, the next hypothesis is generated. This method is successful if good models of the objects potentially present in the images are available and verification can be done reliably. Furthermore, one must ensure that the correct hypothesis is among the first ones that are generated. Top-down techniques are used for image registration and for tracking of objects in image sequences. In the latter case, the hypothesis can be generated by predictions which are based on the analysis results from the preceding frames.

One example of top-down image analysis is the tracking system designed to localize a mobile robot on a RoboCup soccer field [235], illustrated in Figure 1.11. A model of the field walls is combined with a hypothesis about the robot position and mapped to the image obtained from an omnidirectional camera. Perpendicular to the walls, a transition between the field color (green) and the wall (white) is
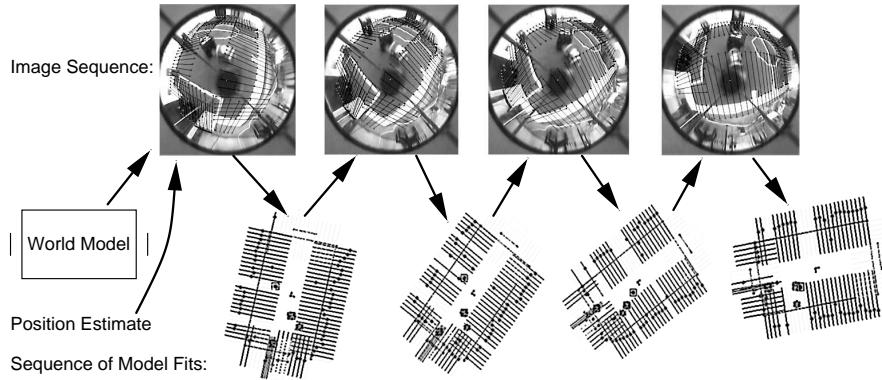
Fig. 1.11. Tracking of a mobile robot in a RoboCup soccer field (image adapted from [235]). The image is obtained using an omnidirectional camera. Transitions from the field (green) to the walls (white) are searched perpendicular to the model walls that have been mapped to the image. Located transitions are transformed into local world coordinates and used to adapt the model fit.

searched for. If it can be located, its coordinates are transformed into local world coordinates and used to adapt the parameters of the model. The ball and other robots can be tracked in a similar way. When using such a tracking scheme for the control of a soccer playing robot, the initial position hypothesis must be obtained using a bottom-up method. Furthermore, it must be constantly checked, whether the model fits the data well enough; otherwise, the position must be initialized again. The system is able to localize the robot in real time and to provide input of sufficient quality for playing soccer.

While both top-down and bottom-up methods have their merits, the image interpretation problem is far from being solved. One of the most problematic issues is the segmentation/recognition dilemma. Frequently, it is not possible to segment objects from the background without recognizing them. On the other hand, many recognition methods require object segmentation prior to feature extraction and classification.

Another difficult problem is maintaining invariance to object transformations. Many recognition methods require normalization of common variances, such as position, size, and pose of an object. This requires reliable segmentation, without which the normalization parameters cannot be estimated.

Processing segmented objects in isolation is problematic by itself. As the example of contextual effects on letter perception in Figure 1.7 demonstrated, we are able to recognize ambiguous objects by using their context. When taken out of the context, recognition may not be possible at all.

1.1.4 Iterative Interpretation through Local Interactions in a Hierarchy

Since the performance of the human visual system by far exceeds that of current computer vision systems, it may prove fruitful to follow design patterns of the hu-

bottom–up:

– data driven

– analysis

– feature extraction

top–down:

– hypothesis driven

– synthesis

– feature expansion

lateral: – grouping   – competition   – associative memory
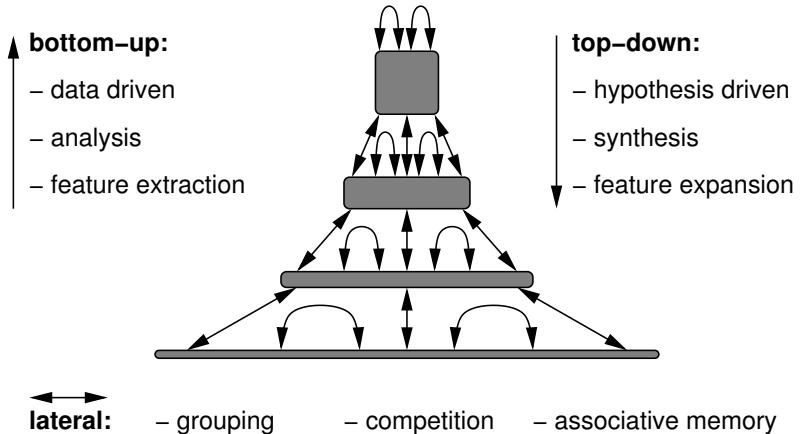
Fig. 1.12. Integration of bottom-up, lateral, and top-down processing in the proposed hierarchical architecture. Images are represented at different levels of abstraction. As the spatial resolution decreases, feature diversity and invariance to transformations increase. Local recurrent connections mediate interactions of simple processing elements.

man visual system when designing computer vision systems. Although the human visual system is far from being understood, some design patterns that may account for parts of its performance have been revealed by researchers from neurobiology and psychophysics.

The thesis tries to overcome some limitations of current computer vision systems by focussing on three points:

– hierarchical architecture with increasingly abstract analog representations,
– iterative refinement of interpretation through integration of bottom-up, top-down, and lateral processing, and
– adaptability and learning to make the generic architecture task-specific.

Hierarchical Architecture. While most computer vision systems maintain multiple representations of an image with different degrees of abstraction, these representations usually differ in the data structures and the algorithms employed. While low-level image processing operators, like convolutions, are applied to matrices representing discretized signals, high-level computer vision usually manipulates symbols in data structures like graphs and collections. This leads to the difficulty of establishing a correspondence between the symbols and the signals. Furthermore, although the problems in high-level vision and low-level vision are similar, techniques developed for the one cannot be applied for the other. What is needed is a unifying framework that treats low-level vision and high-level vision in the same way.

In the thesis, I propose to use a hierarchical architecture with local recurrent connectivity to solve computer vision tasks. The architecture is sketched in Figure 1.12. Images are transformed into a sequence of analog representations with an increasing degree of abstraction. As one ascends the hierarchy, the spatial resolution of
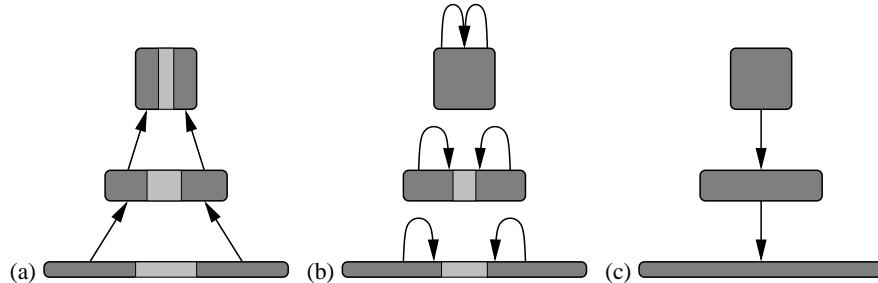
Fig. 1.13. Iterative image interpretation: (a) the image is interpreted first at positions where little ambiguity exists; (b) lateral interactions reduce ambiguity; (c) top-down expansion of abstract representations bias the low-level decision.

these representations decreases, while the diversity of features and their invariance to transformations increase.

Iterative Refinement. The proposed architecture consists of simple processing elements that interact with their neighbors. These interactions implement bottom-up operations, like feature extraction, top-down operations, like feature expansion, and lateral operations, like feature grouping.

The main idea is to interpret images iteratively, as illustrated in Figure 1.13. While images frequently contain parts that are ambiguous, most image parts can be interpreted relatively easy in a bottom-up manner. This produces partial representations in higher layers that can be completed using lateral interactions. Top-down expansion can now bias the interpretation of the ambiguous stimuli.

This iterative refinement is a flexible way to incorporate context information. When the interpretation cannot be decided locally, the decision is deferred, until further evidence arrives from the context.

Adaptability and Learning. While current computer vision systems usually contain adaptable components, such as trainable classifiers, most steps of the processing chain are designed manually. Depending on the application, different preprocessing steps are applied and different features are extracted. This makes it difficult to adapt a computer vision system for a new task.

Neural networks are tools that have been successfully applied to machine learning tasks. I propose to use simple processing elements to maintain the hierarchy of representations. This yields a large hierarchical neural network with local recurrent connectivity for which unsupervised and supervised learning techniques can be applied.

While the architecture is biased for image interpretation tasks, e.g. by utilizing the 2D nature and hierarchical structure of images, it is still general enough to be adapted for different tasks. In this way, manual design is replaced by learning from a set of examples.

1.2 Organization of the Thesis

The thesis is organized as follows:

Part I: Theory

Chapter 2. The next chapter gives some background information on the human visual system. It covers the visual pathways, the organization of feature maps, computation in layers, neurons as processing units, and synapses as adaptable elements. At the end of the chapter, some open questions are discussed, including the binding problem and the role of recurrent connections.

Chapter 3. Related work is discussed in Chapter 3, focussing on two aspects of the proposed architecture: hierarchy and recurrence. Generic signal decompositions, neural networks, and generative statistical models are reviewed as examples of hierarchical systems for image analysis. The use of recurrence is discussed in general. Special attention is paid to models with specific types of recurrent interactions: lateral, vertical, and the combination of both.

Chapter 4. The proposed architecture for image interpretation is introduced in Chapter 4. After giving an overview, the architecture is formally described. To illustrate its use, several small example networks are presented. They apply the architecture to local contrast normalization, binarization of handwriting, and shift-invariant feature extraction.

Chapter 5. Unsupervised learning techniques are discussed in Chapter 5. An unsupervised learning algorithm is proposed for the suggested architecture that yields a hierarchy of sparse features. It is applied to a dataset of handwritten digits. The produced features are used as input to a supervised classifier. The performance of this classifier is compared to other classifiers, and it is combined with two existing classifiers.

Chapter 6. Supervised learning is covered in Chapter 6. After a general discussion of supervised learning problems, gradient descent techniques for feed-forward neural networks and recurrent neural networks are reviewed separately. Improvements to the backpropagation technique and regularization methods are discussed, as well as the difficulty of learning long-term dependencies in recurrent networks. It is suggested to combine the RPROP algorithm with backpropagation through time to achieve stable and fast learning in the proposed recurrent hierarchical architecture.

Part II: Applications

Chapter 7. The proposed architecture is applied to recognize the value of postage meter marks. After describing the problem, the dataset, and some preprocessing steps, two classifiers are detailed. The first one is a hierarchical block classifier that recognizes meter values without prior digit segmentation. The second one is a neural

classifier for isolated digits that is employed when the block classifier cannot produce a confident decision. It uses the output of the block classifier for a neighboring digit as contextual input.

Chapter 8. The second application deals with the binarization of matrix codes. After the introduction to the problem, an adaptive thresholding algorithm is proposed that is employed to produce outputs for undegraded images. A hierarchical recurrent network is trained to produce them even when the input images are degraded with typical noise. The binarization performance of the trained network is evaluated using a recognition system that reads the codes.

Chapter 9. The application of the proposed architecture to image reconstruction problems is presented in Chapter 9. Super-resolution, the filling-in of occlusions, and noise removal/contrast enhancement are learned by hierarchical recurrent networks. Images are degraded and networks are trained to reproduce the originals iteratively. The same method is also applied to image sequences.

Chapter 10. The last application deals with a problem of human-computer interaction: face localization. A hierarchical recurrent network is trained on a database of images that show persons in office environments. The task is to indicate the eye positions by producing a blob for each eye. The network's performance is compared to a hybrid localization system, proposed by the creators of the database.

Chapter 11. The thesis concludes with a discussion of the results and an outlook for future work.

1.3 Contributions

The thesis attempts to overcome limitations of current computer vision systems by proposing a hierarchical architecture for iterative image interpretation, investigating unsupervised and supervised learning techniques for this architecture, and applying it to several computer vision tasks.

The architecture is inspired by the ventral pathway of the human visual system. It transforms images into a sequence of representations that are increasingly abstract. With the level of abstraction, the spatial resolution of the representations decreases, as the feature diversity and the invariance to transformation increase.

Simple processing elements interact through local recurrent connections. They implement bottom-up analysis, top-down synthesis, and lateral operations, such as grouping, competition, and associative memory. Horizontal and vertical feedback loops provide context to resolve local ambiguities. In this way, the image interpretation is refined iteratively.

Since the proposed architecture is a hierarchical recurrent neural network with shared weights, machine learning techniques can be applied to it. An unsupervised learning algorithm is proposed that yields a hierarchy of sparse features. It is applied to a dataset of handwritten digits. The extracted features are meaningful and facilitate digit recognition.

Supervised learning is also applicable to the architecture. It is proposed to combine the RPROP optimization method with backpropagation through time to achieve stable and fast learning. This supervised training is applied to several learning tasks.

A feed-forward block classifier is trained to recognize meter values without the need for prior digit segmentation. It is combined with a digit classifier if necessary. The system is able to recognize meter values that are challenging for human experts.

A recurrent network is trained to binarize matrix codes. The desired outputs are produced by applying an adaptive thresholding method to undegraded images. The network is trained to produce the same output even for images that have been degraded with typical noise. It learns to recognize the cell structure of the matrix codes. The binarization performance is evaluated using a recognition system. The trained network performs better than the adaptive thresholding method for the undegraded images and outperforms it significantly for degraded images.

The architecture is also applied for the learning of image reconstruction tasks. Images are degraded and networks are trained to reproduce the originals iteratively. For a super-resolution problem, small recurrent networks are shown to outperform feed-forward networks of similar complexity. A larger network is used for the filling-in of occlusions, the removal of noise, and the enhancement of image contrast. The network is also trained to reconstruct images from a sequence of degraded images. It is able to solve this task even in the presence of high noise.

Finally, the proposed architecture is applied for the task of face localization. A recurrent network is trained to localize faces of different individuals in complex office environments. This task is challenging due to the high variability of the dataset used. The trained network performed significantly better than the hybrid localization method, proposed by the creators of the dataset. It is not limited to static images, but can track a moving face in real time.

Part I

Theory

2. Neurobiological Background

Learning from nature is a principle that has inspired many technical developments. There is even a field of science concerned with this issue: bionics. Many problems that arise in technical applications have already been solved by biological systems because evolution has had millions of years to search for a solution. Understanding nature's approach allows us to apply the same principles for the solution of technical problems.

One striking example is the 'lotus effect', studied by Barthlott and Neinhuis [17]. Grasping the mechanisms, active at the microscopic interface between plant surfaces, water drops, and dirt particles, led to the development of self-cleaning surfaces. Similarly, the design of the first airplanes was inspired by the flight of birds and even today, though aircraft do not resemble birds, the study of bird wings has lead to improvements in the aerodynamics of planes. For example, birds reduce turbulence at their wing-tips using spread feathers. Multi-winglets and split-wing loops are applications of this principle. Another example are eddy-flaps which prevent sudden drops in lift generation during stall. They allow controlled flight even in situations where conventional wings would fail.

In the same vein, the study of the human visual system is a motivation for developing technical solutions for the rapid and robust interpretation of visual information. Marr [153] was among the first to realize the need to consider biological mechanisms when developing computer vision systems. This chapter summarizes some results of neurobiological research on vision to give the reader an idea about how the human visual system achieves its astonishing performance.

The importance of visual processing is evident from the fact that about one third of the human cortex is involved in visual tasks. Since most of this processing happens subconsciously and without perceived effort, most of us are not aware of the difficulties inherent to the task of interpreting visual stimuli in order to extract vital information from the world.

The human visual system can be described at different levels of abstraction. In the following, I adopt a top-down approach, while focusing on the aspects most relevant for the remainder of the thesis. I will first describe the visual pathways and then cover the organization of feature maps, computation in layers, neurons as processing elements, and synapses that mediate the communication between neurons. A more comprehensive description of the visual system can be found in the book edited by Kandel, Schwartz, and Jessel [117] and in other works.
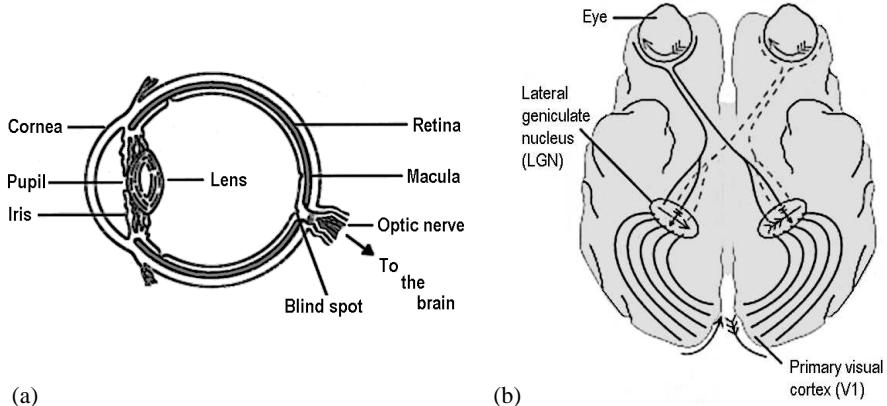
Fig. 2.1. Eye and visual pathway to the cortex. (a) illustration of the eye's anatomy; (b) visual pathway from the eyes via the LGN to the cortex (adapted from [117]).

2.1 Visual Pathways

The human visual system captures information about the environment by detecting light with the eyes. Figure 2.1(a) illustrates the anatomy of the eye. It contains an optical system that projects an image onto the retina. We can move the eyes rapidly using saccades in order to inspect parts of the visual field closer. Smooth eye movements allow for pursuit of moving targets, effectively stabilizing their image on the retina. Head and body movements assist active vision.

The iris regulates the amount of light that enters the eye by adjusting the pupil's size to the illumination level. Accommodation of the lens focuses the optics to varying focal distances. This information, in conjunction with stereoscopic disparity, vergence, and other depth cues, such as shading, motion, texture, or occlusion, is used to reconstruct the 3D structure of the world from 2D images.

At the retina, the image is converted into neural activity. Two morphological types of photoreceptor cells, rods and cones, transduce photons into electrical membrane potentials. Rods respond to a wide range of wavelengths. Since they are more sensitive to light than cones, they are most useful in the dark. Cones are sensitive to one of three specific ranges of wavelengths. Their signals are used for color discrimination and they work best under good lighting conditions. There are about 120 million rods and only 6.5 million cones in the primate retina. The cones are concentrated mainly in the fovea at the center of the retina. Here, their density is about 150,000/mm^2, and no rods are present.

The retina does not only contain photoreceptors. The majority of its cells are dedicated to image processing tasks. Different types of neurons are arranged in layers which perform spatiotemporal compression of the image. This is necessary because the visual information must be transmitted through the optic nerve, which consists of only about 1 million axons of retinal ganglion cells.
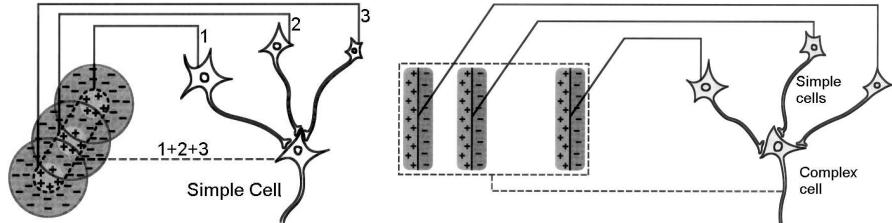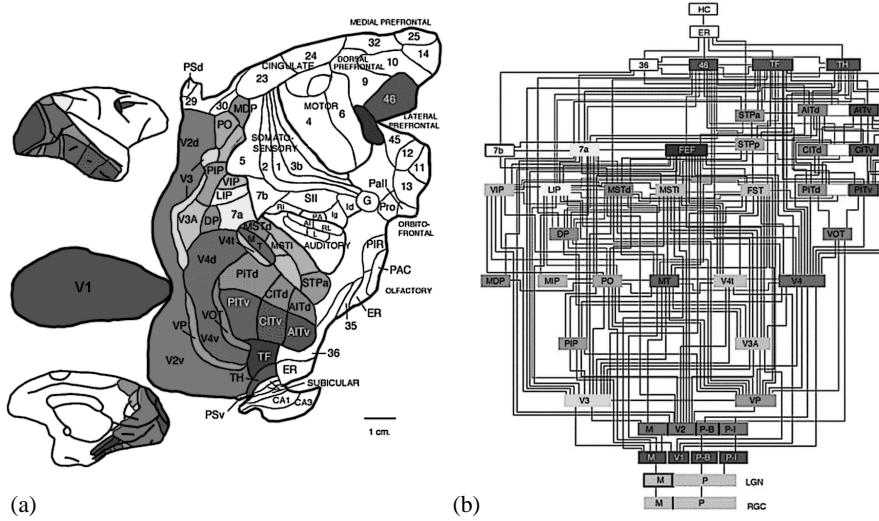
Fig. 2.2. Simple and complex cells. According to Hubel and Wiesel [105] simple cells combine the outputs of aligned concentric LGN cells. They respond to oriented stimuli and are phase sensitive. The outputs of several simple cells that have the same orientation, but different phases are combined by a complex cell, which shows a phase invariant response to oriented edges (adapted from [117]).

These cells send action potentials to a thalamic region, called lateral geniculate nucleus (LGN). Different types of retinal ganglion cells represent different aspects of a retinal image patch, the receptive field. Magnocellular (M) cells have a relatively large receptive field and respond transiently to low-contrast stimuli and motion. On the other hand, parvocellular (P) ganglion cells show a sustained response to color contrast and high-contrast black-and-white detail.

The optical nerve leaves the eye at the blind spot and splits into two parts at the optical chiasma. Axons from both eyes that carry information about the same hemisphere of the image are routed to the contralateral LGN, as can be seen in Figure 2.1(b). In the LGN, the axons from both eyes terminate in different layers. Separation of P-cells and M-cells is maintained as well. The LGN cells have center-surround receptive fields, and are thus sensitive to spatiotemporal contrast. The topographic arrangement of the ganglion receptive fields is maintained in the LGN. Hence, each layer contains a complete retinal map. Interestingly, about 75% of the inputs to the LGN do not come from the retina, but originate in the cortex and the brain stem. These top-down connections may be involved in generating attention by modulating the LGN response.

From the LGN, the visual pathway proceeds to the primary visual cortex (V1). Here, visual stimuli are represented in terms of locally oriented receptive fields. Simple cells have a linear Gabor-like [79] response. According to Hubel and Wiesel [105], they combine the outputs of several aligned concentric LGN cells (see Fig. 2.2(a)). Complex cells show a phase-invariant response that may be computed from the outputs of adjacent simple cells, as shown in Figure 2.2(b). In addition to the orientation of edges, color information is also represented in V1 blobs. As in the LGN, the V1 representation is still retinotopic – information from neighboring image patches is processed at adjacent cortical locations. The topographic mapping is nonlinear. It enlarges the fovea and assigns fewer resources to the processing of peripheral stimuli.

Area V2 is located next to V1. It receives input from V1 and projects back to it. V2 cells are also sensitive to orientation, but have larger receptive fields than those in V1. A variety of hyper-complex cells exists in V2. They detect line endings,

Fig. 2.3. Hierarchical structure of the visual system. (a) Felleman and Van Essen's [65] flat map of the Macaque brain with marked visual areas; (b) wiring diagram of the visual areas.

corners, or crossings, for instance. It is believed that V2 neurons play a crucial role in perceptual grouping and line completion since they have been shown to respond to illusory contours.

V1 and V2 are only the first two of more than 30 areas that process visual information in the cortex. A cortical map illustrates their arrangement in Figure 2.3(a). Part (b) of the figure shows a wiring diagram. It can be seen that these areas are highly interconnected. The existence of about 40% of all possible connections has been verified. Most of these connections are bidirectional, as they carry information forward, towards the higher areas of the cortex, and backwards, from higher areas to lower ones.

The visual areas are commonly grouped into a dorsal stream that leads to the parietal cortex, and a ventral stream that leads to the inferotemporal cortex [39], as shown in Figure 2.4. Both pathways process different aspects of visual information.

The dorsal or 'where' stream focuses on the fast processing of peripheral stimuli to extract motion, spatial aspects of the scene, and stereoscopic depth information. Stimuli are represented in different frames of reference, e.g. body-centered and hand-centered. It works with low resolution and serves real-time visuomotor behaviors, such as eye movements, reaching and grasping. For instance, neurons in the middle temporal area MT were found to be directionally sensitive when stimulated with random dot patterns. There is a wide range of speed selectivity and also selectivity for disparity. These representations allow higher parietal areas, such as MST, to compute structure from motion or structure from stereopsis. Also, ego-motion, caused by head and eye movements, is distinguished from object motion.

In contrast, the ventral or 'what' stream focuses on foveal stimuli that are processed relatively slowly. It is involved in form perception and object recognition

Fig. 2.4. Dorsal and ventral visual streams. The dorsal stream ascends from V1 to the parietal cortex. It is concerned with spatial aspects of vision ('where'). The ventral stream leads to the inferotemporal cortex and serves object recognition ('what') (adapted from [117]).

tasks. A hierarchy of areas represents aspects of the visual stimuli that are increasingly abstract.

As illustrated in Figure 2.5, in higher areas the complexity and diversity of the processed image features increases, as do receptive field size and invariance to stimulus contrast, size, or position. At the same time spatial resolution decreases. For instance, area V4 neurons are broadly selective for a wide variety of stimuli: color, light and dark, edges, bars, oriented or non-oriented, moving or stationary, square wave and sine wave gratings of various spatial frequencies, and so on. One consistent feature is that they have large center-surround receptive fields. Maximum response is produced when the two regions are presented with different patterns or colors. Recently, Pasupathy and Connor [176] found cells in V4 tuned to complex object parts, such as combinations of concave and convex borders, coarsely localized relative to the object center. V4 is believed to be important for object discrimination and color constancy.

The higher ventral areas, such as area IT in the temporal cortex, are not necessarily retinotopic any more since neurons cover most of the visual field. Neurons in IT respond to complex stimuli. There seem to exist specialized modules for the recognition of faces or hands, as illustrated in Figure 2.6. These stimuli deserve specialized processing since they are very relevant for our social interaction.

Both streams do not work independently, but in constant interaction. Many reciprocal connections between areas of different streams exist that may mediate the binding of spatial and recognition aspects of an object to a single percept.
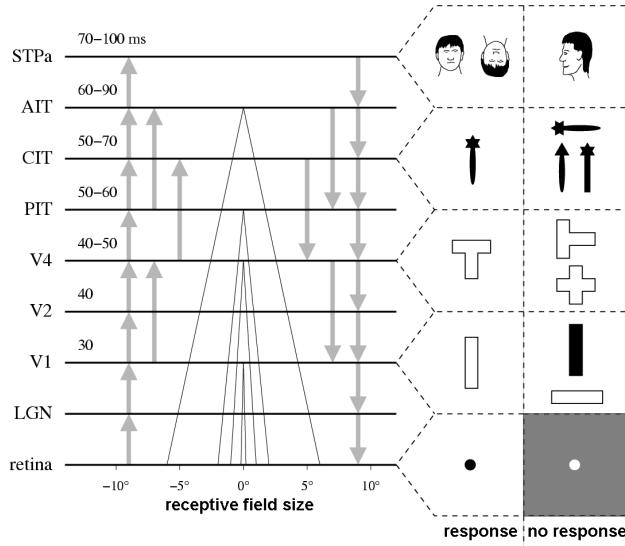
Fig. 2.5. Hierarchical structure of the ventral visual pathway. Visual stimuli are represented at different degrees of abstraction. As one ascends towards the top of the hierarchy receptive field size and feature complexity increase while variance to transformations and spatial resolution decrease (adapted from [243]).
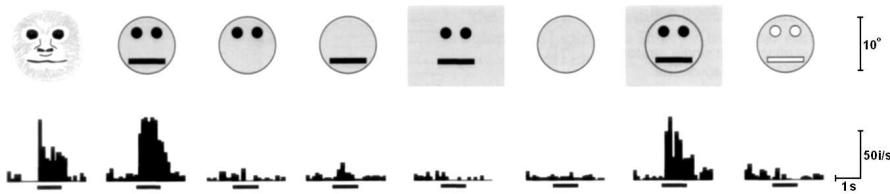
Fig. 2.6. Face selectivity of IT cells. The cell responds to faces and face-like figures, but not to face parts or inverted faces (adapted from [117].

2.2 Feature Maps

The visual areas are not retinotopic arrays of identical feature detectors, but they are covered by regular functional modules, called hypercolumns in V1. Such a hyper-column represents the properties of one region of the visual field.

For instance, within every $1mm^2$ patch in area V1, a complete set of local orientations is represented, as illustrated in Figure 2.7. Neurons that share the same orientation and have concentric receptive fields are grouped vertically into a column. Adjacent columns represent similar orientations. They are arranged around singular points, called pinwheels, where all orientations are accessible in close proximity.

In addition to the orientation map, V1 is also covered by a regular ocular dominance pattern. Stripes that receive input from the right and the left eye alternate. This makes interaction between the information from both eyes possible, e.g. to extract
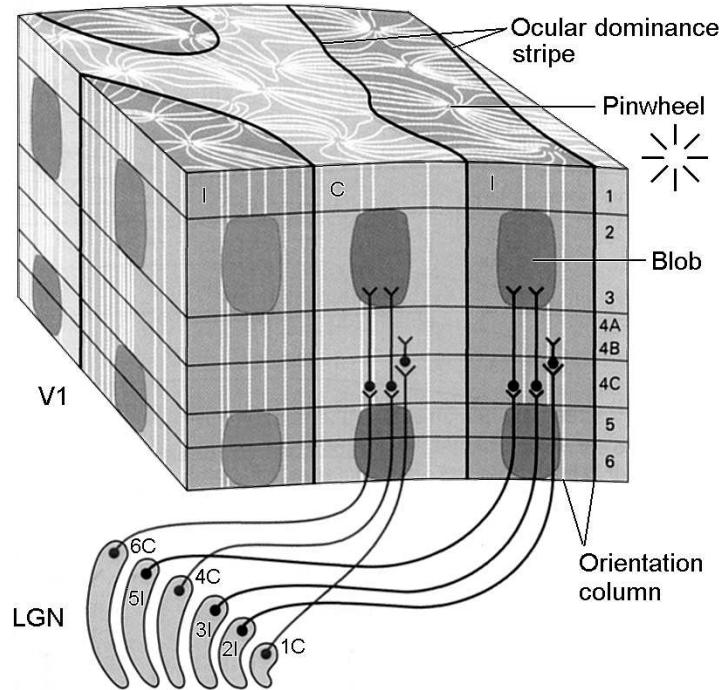
Fig. 2.7. Hypercolumn in V1. Within 1mm^2 of cortex all features of a small region of the visual field are represented. Orientation columns are arranged around pinwheels. Ocular dominance stripes from the ipsilateral (I) and the contralateral (C) eye alternate. Blobs represent color contrast (adapted from [117]).

disparity. A third regular structure in V1 is the blob system. Neurons in the blobs are insensitive to orientation, but respond to color contrast. Their receptive fields have a center-surround shape, mostly with double color opponency.

Similar substructures exist in the next higher area, V2. Here, not columns, but thin stripes, thick stripes, and interstripes alternate. The stripes are oriented orthogonally to the border between V1 and V2. A V2 'hyperstripe' covers a larger part of the visual field than a V1 hypercolumn and represents different aspects of the stimuli present in that region. As illustrated in Figure 2.4, the blobs in V1 send color information primarily to the thin stripes in V2, while the orientation sensitive interblobs in V1 connect to interstripes in V2. Both thin and interstripes project to separate substructures in V4. Layer 4B of V1 that contains cells sensitive to the magnocellular (M) information projects to the thick stripes in V2 and to area MT. Thick stripes also project to MT. Hence, they also belong to the M pathway.

These structured maps are not present at birth, but depend for their development on visual experience. For example, ocular dominance stripes in V1 are reduced in size if during a critical period of development input from one eye is deprived. The

development of the hierarchy of visual areas probably proceeds from lower areas to higher areas.

The repetitive patterns of V1 and V2 lead to the speculation that higher cortical areas, like V4, IT, or MT contain even more complex interwoven feature maps. The presence of many different features within a small cortical patch that belong to the same image location has the clear advantage that they can interact with minimal wire length. Since in the cortex long-range connections are costly, this is such a strong advantage that the proximity of neurons almost always implies that they interact.

2.3 Layers

The cortical sheet, as well as other subcortical areas, is organized in layers. These layers contain different types of neurons and have a characteristic connectivity. The best studied example is the layered structure of the retina, illustrated in Figure 2.8.

The retina consists of three layers that contain cell bodies. The outer nuclear layer contains the photosensitive rods and cones. The inner nuclear layer consists of horizontal cells, bipolar cells, and amacrine cells. The ganglion cells are located in the third layer. Two plexiform layers separate the nuclear layers. They contain dendrites connecting the cells.

Information flows vertically from the photoreceptors via the bipolar cells to the ganglion cells. Two types of bipolar cells exist that are either excited or inhibited by the neurotransmitters released from the photoreceptors. They correspond to on/off centers of receptive fields.
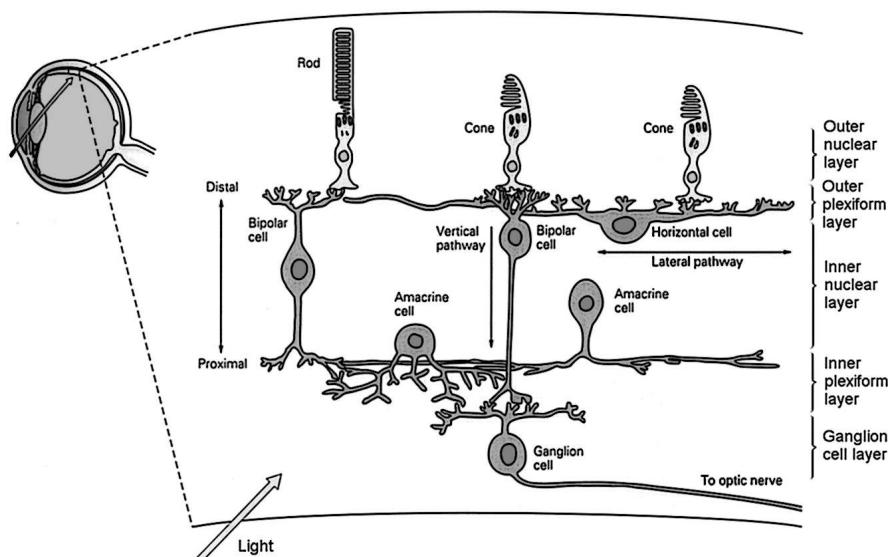
Fig. 2.8. Retina. Spatiotemporal compression of information by lateral and vertical interactions of neurons that are arranged in layers (adapted from [117]).

Information flows also laterally through the retina. Photoreceptors are connected to each other by horizontal cells in the outer plexiform layer. The horizontal cells mediate an antagonistic response of the center cell when the surround is exposed to light. Amacrine cells are interneurons that interact in the inner plexiform layer. Several types of these cells exist that differ greatly in size and shape of their dendritic trees. Most of them are inhibitory. Amacrine cells serve to integrate and modulate the visual signal. They also bring the temporal domain into play in the message presented to a ganglion cell.

The result of the vertical and horizontal interaction is a visual signal which has been spatiotemporally compressed and that is represented by different types of center-surround features. Automatic gain control and predictive coding are achieved. While all the communication within the retina is analog, ganglion cells convert the signal into all-or-nothing events, the action potentials or spikes, that travel fast and reliably the long way through the optic nerve.

Another area for which the layered structure has been investigated in depth is the primary visual cortex, V1. As all cortical areas do, the 2mm thick V1 has six layers that have specific functions, as shown in Figure 2.9. The main target for input from the LGN is layer 4, which is further subdivided into four sublayers. While the axons from M cells terminate principally in layer $4C\alpha$, the P cells send their output to layer $4C\beta$. Interlaminar LGN cells terminate in the blobs present in layers 2 and 3. Not shown in the figure is feedback input from higher cortical areas that terminates in layers 1 and 2.

Two major types of neurons are present in the cortex. Pyramidal cells are large and project to distant regions of the cortex and to other brain structures. They are always excitatory and represent the output of the computation carried out in their cortex patch. Pyramidal cells from layers 2, 3, and 4B of V1 project to higher cortical areas. Outputs from layers 5 and 6 lead to the LGN and other subcortical areas.

Stellate cells are smaller than pyramidal cells. They are either excitatory (80%) or inhibitory (20%) and serve as local interneurons. Stellate cells facilitate the in-
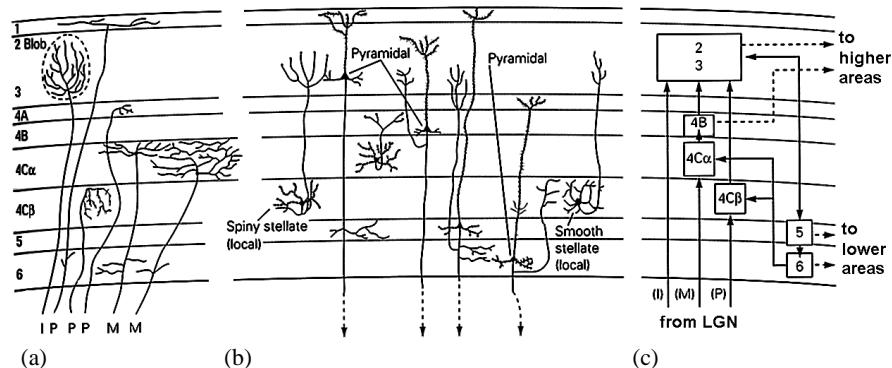
Fig. 2.9. Cortical layers in V1: (a) inputs from LGN terminate in different layers; (b) resident cells of various type; (c) recurrent information flow (adapted from [117]).

teraction of neurons belonging to the same hypercolumn. For instance, the M and P input from LGN is relayed by excitatory spiny stellate cells to layers 2 and 3.

The pyramidal output is also folded back into the local circuit. Axon collaterals of pyramidal cells from layers 2 and 3 project to layer 5 pyramidal cells, whose axon collaterals project both to layer 6 pyramidal cells and back to cells in layers 2 and 3. Axon collaterals of layer 6 pyramidal cells project back to layer 4C inhibitory smooth stellate cells.

Although many details of the connectivity of such local circuits are known, the exact function of these circuits is far from being understood. Some possible functions could be the aggregation of simple features to more complex ones, as happens in V1 with the aggregation from center-surround to linear oriented to phase-invariant oriented responses. Furthermore, local gain control and the integration of feed-forward and feedback signals are likely functions of such circuits.

In addition to local recurrent computation and vertical interactions, there is also heavy lateral connectivity within a cortical area. Figure 2.10 shows a layer 3 pyramidal cell that connects to pyramidal cells of similar orientation within the same functional column and with similarly oriented pyramidal cells of neighboring aligned hypercolumns. These specific excitatory connections are supplemented by unspecific inhibition via interneurons.

The interaction between neighboring hypercolumns may mediate extra-classical effects of receptive fields. In these cases, the response of a neuron is modulated by the presence of other stimuli outside the classical receptive field. For instance, neurons in area V1 are sensitive not just to the local edge features within their receptive fields, but are strongly influenced by the context of the surrounding stimuli. These contextual interactions have been shown to exert both facilitatory and inhibitory effects from outside the classical receptive fields. Both types of interactions can affect the same unit, depending on various stimulus parameters. Recent cortical models by Stemmler *et al.* [220] and Somers *et al.* [219] described the action of the surround as
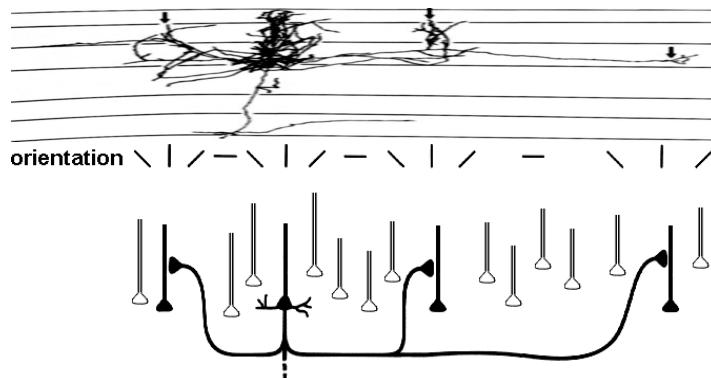
Fig. 2.10. Lateral connections in V1. Neighboring aligned columns of similar orientation are linked with excitatory lateral connections. There is also unspecific local inhibition via interneurons (adapted from [117]).

a function of the relative contrast between the center stimulus and the surround stimulus. These mechanisms are thought to mediate such perceptual effects as filling-in [237] and pop-out [123].

Lateral connections may also be the substrate for the propagation of activity waves that have been observed in the visual cortex [208] as well as in the retina. These waves are believed to play a important role for the development of retinotopic projections in the visual system [245].

2.4 Neurons

Individual nerve cells, neurons, are the basic units of the brain. There are about 10^{11} neurons in the human brain that can be classified into at least a thousand different types. All neurons specialize in electro-chemical information processing and transmission. Furthermore, around the neurons many more glia cells exist, which are believed to play only a supporting role.

All neurons have the same basic morphology, as illustrated in Figure 2.11. They consist of a cell body and two types of specialized extensions (processes): dendrites and axons. The cell body (soma) is the metabolic center of the cell. It contains the nucleus as well as the endoplasmatic reticulum, where proteins are synthesized.

Dendrites collect input from other nerve cells. They branch out in trees containing many synapses, where postsynaptic potentials are generated when the presynaptic cell releases neurotransmitters in the synaptic cleft. These small potentials are aggregated in space and time within the dendrite and conducted to the soma.

Most neurons communicate by sending action potentials down the axon. If the membrane potential at the beginning of the axon, the axon hillock, exceeds a thresh-
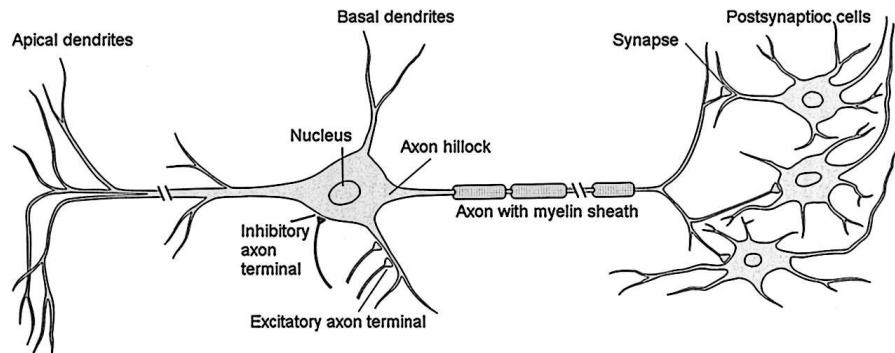
Fig. 2.11. Structure of a neuron. The cell body contains the nucleus and gives rise to two types of specialized extensions: axons and dendrites. The dendrites are the input elements of a neuron. They collect postsynaptic potentials, integrate them and conduct the resulting potential to the cell body. At the axon hillock an action potential is generated if the membrane voltage exceeds a threshold. The axon transmits this spike over long distances. Some axons are myelinated for fast transmission. The axon terminates in many synapses that make contact with other cells (adapted from [117]).

old, a wave of activity is generated and actively propagated towards the axon terminals. Thereafter, the neuron becomes insensitive to stimuli during a refractory period of some milliseconds. Propagation is based on voltage sensitive channels in the axon's membrane. For fast transmission, some axons are covered by myelin sheaths, interrupted by nodes of Ranvier. Here, the action potential jumps from node to node, where it is regenerated. The axon terminates in many synapses that make contacts with other cells.

Only some neurons, that have no axons or only very short axons, use the graded potential directly for neurotransmitter release at synapses. They can be found, for instance, in the retina and in higher areas of invertebrates. Although the graded potential contains more information than the all-or-nothing signal of an action potential [87], it is used for local communication only since it decays exponentially when conducted over longer distances. In contrast, the action potential is regenerated and thus is not lost. Action potentials have a uniform spike-like shape with a duration of 1ms. The frequency of sending action potentials and the exact timing of these potentials relative to each other and relative to the spikes of other cells or to other sources of reference, such as subthreshold oscillations or stimulus onset, may contain information.

Neurons come in many different shapes as they form specific networks with other neurons. Depending on their task, they collect information from many other neurons in a specific way and distribute their action potential to a specific set of other cells. Although neurons have been modeled as simple leaky integrators with a single compartment, it is more and more appreciated that more complex computation is done in the dendritic tree than passive conductance of postsynaptic potentials. For example, it has been shown that neighboring synapses can influence each other e.g. in a multiplicative fashion. Furthermore, active spots have been localized in dendrites, where membrane potentials are amplified. Finally, information also travels backwards into the dendritic tree when a neuron is spiking. This may influence the response to the following presynaptic spikes and also be a substrate for modification of synaptic efficacy.

2.5 Synapses

While neurons communicate internally by means of electric potentials, communication between neurons is mediated by synapses. Two types of synapses exist: electrical and chemical.

Electrical synapses couple the membranes of two cells directly. Small ions pass through gap-junction channels in both directions between the cells. Electrical transmission is graded and occurs even when the currents in the presynaptic cell are below the threshold for an action potential. This communication is very fast, but unspecific and not flexible. It is used, for instance, to make electrically connected cells fire in synchrony. Gap-junctions play also a role in glia cells, where Ca^{2+} waves travel through networks of astrocytes.
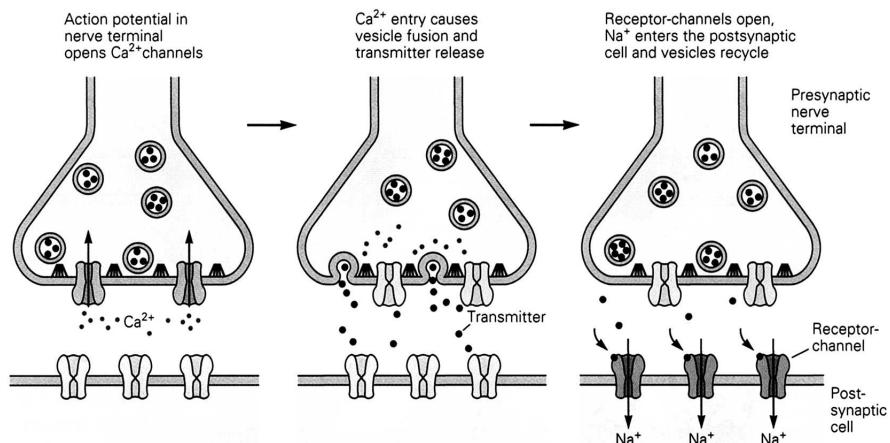
Fig. 2.12. Synaptic transmission at chemical synapse. Presynaptic depolarization leads to the influx of Ca^{2+} ions through voltage gated channels. Vesicles merge with the membrane and release neurotransmitters into the synaptic cleft. These diffuse to receptors that open or close channels in the postsynaptic membrane. Changed ion flow modifies the postsynaptic potential (adapted from [117]).

Chemical synapses allow for more specific communication between neurons since they separate the potentials of the presynaptic and postsynaptic cells by the synaptic cleft. Communication is unidirectional from the presynaptic to the postsynaptic cell, as illustrated in Figure 2.12.

When an action potential arrives at a synaptic terminal, voltage gated channels in the presynaptic membrane are opened and Ca^{2+} ions flow into the cell. This causes vesicles containing neurotransmitters to fuse with the membrane at specific docking sites. The neurotransmitters are released and diffuse through the synaptic cleft. They bind to corresponding receptors on the postsynaptic membrane that open or close ion channels. The modified ion flux now changes the postsynaptic membrane potential.

Neurotransmitters act either directly or indirectly on ion channels that regulate current flow across membranes. Direct gating is mediated by ionotropic receptors that are an integral part of the same macromolecule which forms the ion channel. The resulting postsynaptic potentials last only for few milliseconds. Indirect gating is mediated by activation of metabotropic receptors that are distinct from the channels. Here, channel activity is modulated through a second messenger cascade. These effects last for seconds to minutes and are believed to play a major role in adaptation and learning.

The postsynaptic response can be either excitatory or inhibitory, depending on the type of the presynaptic cell. Figure 2.13 shows a presynaptic action potential along with an excitatory (EPSP) and an inhibitory postsynaptic potential (IPSP). The EPSP depolarizes the cell from its resting potential of about -70mV and brings it closer towards the firing threshold of -55mV. In contrast, the IPSP hyperpolarizes the cell beyond its resting potential. Excitatory synapses are mostly located at spines
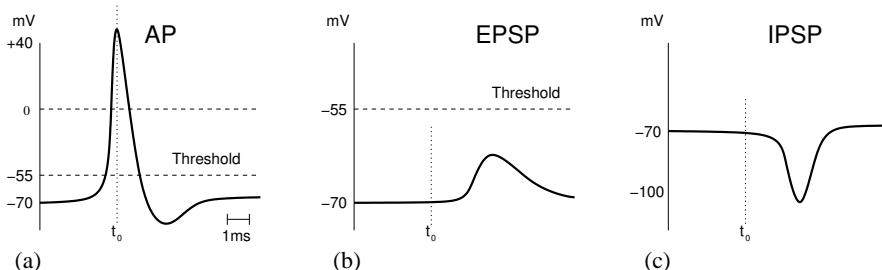
Fig. 2.13. Electric potentials on a synapse: (a) presynaptic action potential; (b) excitatory postsynaptic potential; (c) inhibitory postsynaptic potential (after [117]).

in the dendritic tree and less frequently at dendritic shafts. Inhibitory synapses often contact the cell body, where they can have a strong effect on the graded potential that reaches the axon hillock. Hence, they can mute a cell.

The synaptic efficacy, the amplification factor of a chemical synapse, can vary greatly. It can be changed on a longer time scale by processes called long term potentiation (LTP) and long term depression (LTD). These are believed to depend on the relative timing of pre- and postsynaptic activity. If a presynaptic action potential precedes a postsynaptic one, the synapse is strengthened, while it is weakened when a postsynaptic spike occurs shortly before a presynaptic one.

In addition, transient modifications of synaptic efficacy exist, that lead to effects of facilitation or depression of synapses by series of consecutive spikes. Thus, bursts of action potentials can have a very different effect on the postsynaptic neuron than regular spike trains. Furthermore, effects like gain control and dynamic linking of neurons could be based on the transient modification of synaptic efficacy. This short-term dynamics can be understood, for instance, in terms of models that contain a fixed amount of a resource (e.g. neurotransmitter) which can be either available, effective, or inactive.

2.6 Discussion

The top-down description of the human visual systems stops here, at the level of synapses, although many interesting phenomena exist at deeper levels, like at the level of channels or at the level of neurotransmitters. The reason for this is that it is unlikely that specific low-level phenomena, like the generation of action potentials by voltage sensitive channels, are decisive for our remarkable visual performance, since they are common to all types of nervous systems.

For the remainder of this thesis, these levels serve as a substrate that produces macroscopic effects, but they are not analyzed further. However, one should keep in mind that these deeper levels exist and that subtle changes at the microscopic level, like the increase of certain neurotransmitters after the consumption of drugs, can have macroscopic effects, like visual hallucinations generated by feedback loops with uncontrolled gains.

The visual system has received much attention in neurobiology and psychophysics. In fact, more research has been done for vision than for all other senses together. Many details about the organization of the visual system are known at the various levels of description. However, as of today, the function of the system has not been understood completely.

For instance, there is debate about the neural code used by the brain. One of the questions is whether cortical neurons are mainly driven by the average firing rates of presynaptic neurons or by temporally coherent firing events. It is likely that both coding schemes are used in situations where they are appropriate. In this sense, Tsodyks and Markram [230] argue that the code depends on the rate of synaptic depression and that a continuum between rate codes and temporal codes exists.

Another open issue is the so called binding problem. How is information about color, motion, depth, and form, which is carried by separate neuronal pathways, organized into cohesive perceptions of objects? Since different features of a visual scene are represented by the activity of specialized neurons that are distributed through the visual system, all aspects of an object must be brought temporally into association.

Treisman *et al.* [229] and Jualesz [114] have shown that such associations require focused attention. They found that distinctive elementary properties, such as brightness, color, and orientation of lines, create distinctive boundaries that make objects preattentively salient. They suggest that in a first phase of perception, all features of the visual field are processed in parallel in a bottom-up way. In their model a spotlight of attention highlights the features of individual objects in a serial manner after the initial analysis. This reflects the effects of top-down attention. The spotlight of attention requires a master map that combines details from individual feature maps which are essential for recognition.

Another view on the effect of attention was recently proposed by Reynolds and Desimone [190]. They assume that attention acts to increase the competitive advantage of the attended stimulus so that the effect of attention is to shrink the effective size of a neuron's receptive field around the attended stimulus, as illustrated in Figure 2.14. Now, instead of many stimuli with different characteristics, such as color and form, only one stimulus is functionally present in the receptive field.

A different approach to the binding problem has been proposed by Singer and Gray [216] and Eckhorn *et al.* [59]. They found that when an object activates a population of neurons in the visual cortex, these neurons tend to oscillate and to synchronize their action potentials. To bind together different visual features of the same object, the synchrony would extend across neurons in different visual areas.

Another puzzling problem is the role of the recurrent connections, ubiquitous in the visual system, with respect to conscious visual experience. Visual perception is usually explained in the context of the feed-forward model of visual processing. This model starts from the anatomical hierarchy of cortical areas, with areas V1 and V2 at the lowest levels and the inferotemporal and frontal cortex at the highest stages. Selectivity of a neuron at a given stage is assumed to result from the organized convergence of feed-forward inputs from neurons located at lower stages.
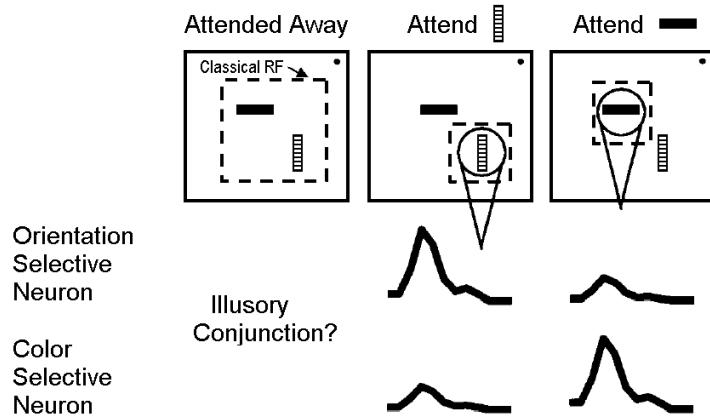
Fig. 2.14. Binding by shrinkage of receptive fields to attended stimuli, as proposed by Reynolds and Desimone [190]. The orientation selective neuron responds to a vertical bar anywhere in its classical receptive field while the color sensitive neuron responds to any dark bar, regardless of its orientation. Thus, when attention is drawn away, the response of the neurons to the two objects is ambiguous since both are active. In contrast, when attention is directed to one of the two stimuli, both neurons respond as if only the attended object were present.

Because of this connectivity rule, neurons at low levels of the hierarchy have small and relatively simple receptive fields, whereas neurons at the highest stages have large and very specialized receptive fields. Activity of neurons at the highest stages of the hierarchy is important for conscious vision, as suggested by the results of imaging studies in humans and recordings in monkeys with bistable visual stimuli. Although this model explains a large number of observations in visual perception, it fails to account for the very dense network of feedback connections that connect cortical areas in the reverse direction.

Super *et al.* [222] investigated what goes wrong when salient stimuli sometimes go undetected. They showed that figure/ground contextual modulation in V1 [130] is influenced strongly by whether stimuli are either 'seen' or 'not seen'. The figure/ground contextual modulation not only makes V1 neurons respond better, but this enhancement is spatially uniform within the figure. Both 'detected' and 'non detected' stimuli evoke similar early neuronal activity. In both cases, the visual input reaches V1 and produces a clear neural response. Only the contextual modulation reflects in a qualitative manner whether the stimulus has been processed up to the level of 'detection'. They conclude that this perceptual level is situated between purely sensory and decision or motor stages of processing.

In line with these findings, Lee *et al.* [139] present neurophysiological data, which shows that the later part of V1 neuron responses reflects higher order perceptual computations, such as figure/ground segmentation. They propose that, because of V1 neurons precise encoding of orientation and spatial information, higher level perceptual computations that require high resolution details, fine geometry and spa-
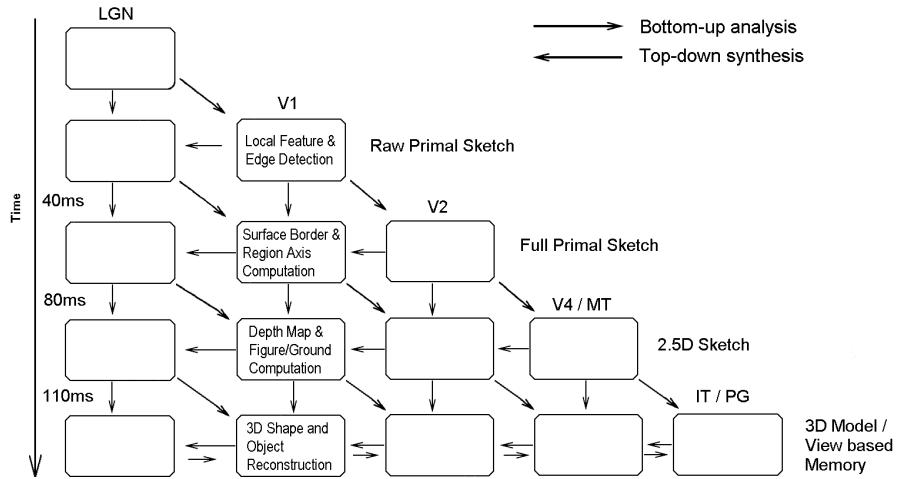
Fig. 2.15. Illustration of the model of Lee *et al.* for the role of V1. Image segmentation, figure/ground, shape computation and object recognition in this framework occur concurrently and interactively in a constant feed-forward and feedback loop that involves the entire hierarchical circuit in the visual system. Signals of higher level visual representations, such as a 2.5D surface sketch, 3D model or view-based object memory, are likely reflected in the later part of V1's activities. (Adapted from [139]).

tial precision would necessarily involve V1 and be reflected in the later part of its neurons activities. This is illustrated in Figure 2.15.

This model is supported by the report of Doninger *et al.* [54], who found that electric potentials reflecting closure have a latency of 290ms when incomplete patterns must be recognized. Since higher ventral areas are activated much earlier, this initial activity does not produce a coherent percept of the incomplete object. They suggest that the objects must be first completed by feed-forward/feedback interactions with lower visual areas before they can be recognized. Modulated activity in lower areas may reflect these interactions.

Furthermore, a recent report by Pascual-Leone and Walsh [174] using transcranial magnetic stimulation (TMS) suggests that activation of feedback connections to the lowest stages of the hierarchy might be essential for conscious vision. They stimulated area V5/MT and area V1 asynchronously and investigated how the interaction of both stimuli affected perceived phosphenes (moving flashes of light). They found that TMS over V1 with a latency of 5 to 45ms after TMS over V5 disrupted the perception of the phosphene, while neither earlier nor later V1 stimuli nor a conditioning V5 stimulus did affect the percept.

Based on these findings, Bullier [37] proposed that areas V1 and V2, instead of simply transmitting information, might act as 'active blackboards' that integrate the results of computations performed in higher order areas, at least for the early stages of processing. This would be an efficient way to solve the problem of computations that involve interactions between features which are not present in neighboring neurons in any one cortical area.

While the exact role of the feedback connections is not yet fully understood, it can be safely stated that there is strong evidence that recurrent connections are crucial for the performance of the visual system. This is not obvious when probing it with isolated, high contrast stimuli which can be processed in a single feed-forward sweep. However, in the regular mode of operation, natural visual stimuli contain much ambiguity, e.g. due to occlusions, low contrast, and noise. In these situations, feedback is used to bias low-level decisions based on attention and on the context of partial scene interpretations. It seems that only after the match between higher-level models and the low-level visual stimuli, a visual perception is relayed to prefrontal areas and becomes conscious.

2.7 Conclusions

In Chapter 4, an architecture for computer vision will be introduced that is motivated by the ventral pathway of the human visual system. It resembles key features of that system, such as:

– computation by simple processing elements arranged in layers,
– retinotopic organization of interwoven feature maps,
– local recurrent connection structure with specific excitation and unspecific inhibition,
– hierarchy of representations with increasing feature complexity, receptive field size, invariance, and number of features,
– iterative refinement of image interpretation,
– integration of top-down, bottom-up, and lateral influences, and
– adaptation to the statistics of visual stimuli through learning.

Not all aspects covered in the previous chapter will be used in the remainder of the thesis. For instance, the proposed architecture focuses on the ventral processing stream and does not reflect the dorsal processing. Furthermore, eye movements, the log-polar mapping between the retina and V1, and color processing are not investigated, although they are important for the performance of the human visual system. The reason for this restriction is that coverage of all these aspects would compete for the available resources with the in-depth discussion of the selected aspects.

It is also important to note that the degree of biological realism in the remainder of the thesis will only be very limited. The aim of the proposed architecture is not to model neurobiological data, as is done in computational neuroscience, but to solve computer vision problems based on inspiration from the human visual system.

3. Related Work

In the previous chapter, we saw that object recognition in the human visual system is based on a hierarchy of retinotopic feature maps with local recurrent connectivity. The following chapter reviews several applications of the concepts of hierarchy and recurrence to the representation, processing, and interpretation of images with computers.

3.1 Hierarchical Image Models

The world is hierarchical and so are images of it. Objects consist of parts and these of subparts. Features can be decomposed into subfeatures all the way down to pixel intensities. Thus, a visual scene can be represented at different degrees of abstraction.

Marr [153] was one of the first to propose analyzing visual stimuli at different levels of abstraction. He proposed using local image operators to convert a pixel image into a primal sketch. He suggested, for example, to use the zero-crossings of the smoothed intensities's second derivative as edge detector. In Marr's approach to vision, the detected edges are grouped according to Gestalt principles [125] to produce the full primal sketch. Adding other features, such as contour, texture, stereopsis, and shading, yields a $2\frac{1}{2}$D sketch. This representation is still viewer-centered and describes properties of surface patches, such as curvature, position, depth, and 3D orientation. Finally, a 3D representation is obtained. It is object-centered and consists of volumetric primitives, generalized cones, organized as a hierarchy. Marr's computational theory of vision has considerably inspired computer vision research. However, its utility in practice has been limited by the use of symbolic representations which do not reflect ambiguities inherent in visual stimuli.

In the following sections, some subsymbolic hierarchical image representation approaches are discussed. I group the different methods into generic signal decompositions, neural networks, and generative statistical models.

3.1.1 Generic Signal Decompositions

Some techniques decompose signals into a hierarchy of generic features, which are efficient to compute and can be inverted. These decompositions are applicable to

G_4 20×15

G_3 40×30 L_3

G_2 80×60 L_2

160×120 L_1

G_1

320×240 L_0

G_0

(a)   (b)
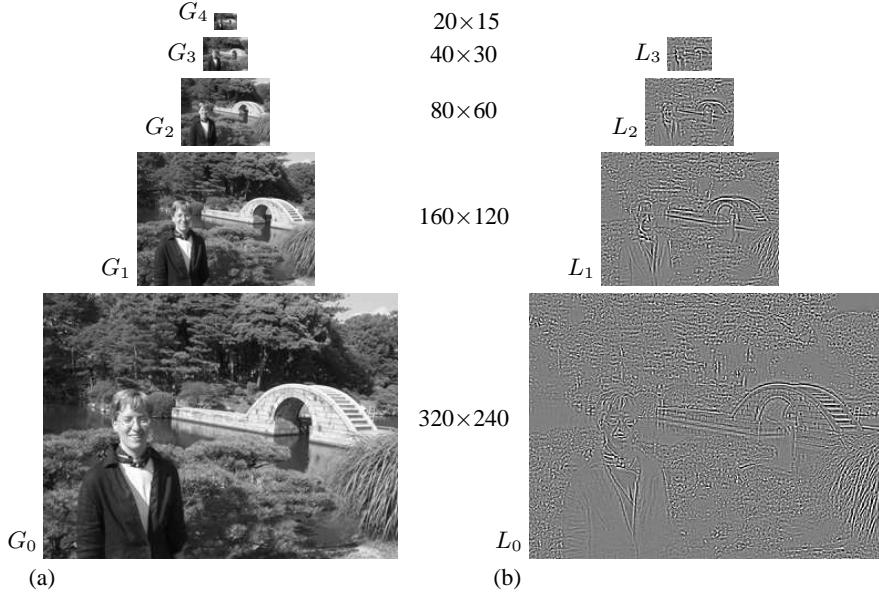
Fig. 3.1. Image pyramids: (a) Gaussian pyramid, representing only coarse structures at the higher levels; (b) Laplacian pyramid, containing the differences between Gaussian levels (amplified for better visibility).

a wide range of signals, including images, but offer only limited adaptability to a specific dataset.

Image Pyramids. A widely used tool in image processing and computer graphics are multiresolution representations called image pyramids. In an image pyramid, the image is not only represented at the given high resolution G_0, but through a sequence $G_0, G_1, \ldots, G_k$ of 2D pixel arrays with decreasing resolutions.

A reduce operation computes the next higher level G_{i+1} from the level G_i using only local operations. Most common is the dyadic Gaussian pyramid, where a pixel $G_{i+1}(i,j)$ is computed as the weighted average of the pixels around the corresponding position $G_i(2i + \frac{1}{2}, 2j + \frac{1}{2})$ in the lower level. Each step reduces the image resolution by a factor of two in both dimensions. Figure 3.1(a) shows such a Gaussian pyramid for an example image. Its total size is slightly less than $1\frac{1}{3}$ the size of G_0.

While image details are visible only in the lower levels of the Gaussian pyramid, the higher levels make larger objects accessible in small windows. This allows one to design coarse-to-fine algorithms [196] for image analysis. Such algorithms start to analyze an image at the coarsest resolution that can be processed quickly. As they proceed to the finer levels, they use coarse results to bias the finer analysis. For instance, when searching for an object, a small number of hypotheses can be established by inspecting the coarse resolution. The finer resolutions are analyzed
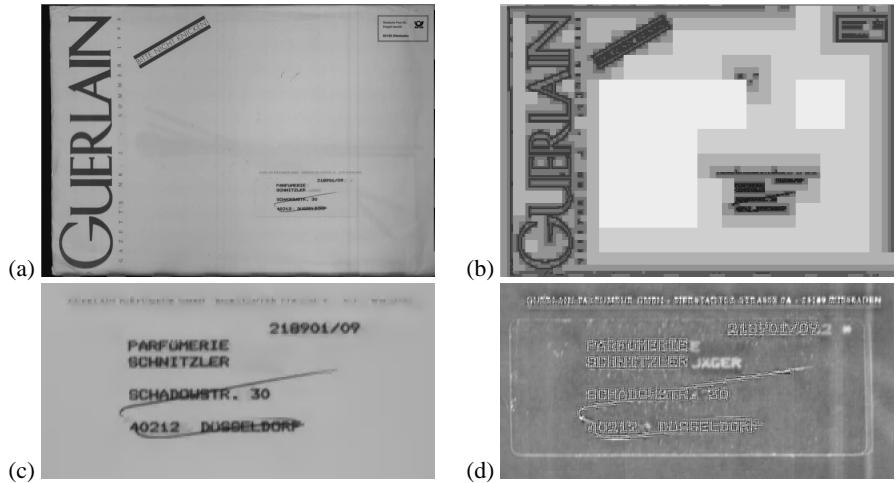
Fig. 3.2. Image compression using pruned pyramids: (a) original image of a letter; (b) resolution used after pruning (darker shading corresponds to higher resolution, the compression ratio is 150:1); (c) reconstructed address region; (d) difference of the reconstruction to the original (amplified for visibility).

only at the corresponding positions to verify and to refine the hypotheses. This saves computational costs, compared to a high-resolution search.

Burt and Adelson [38] proposed the use of differences $L_0, L_1, \ldots, L_{k-1}$ between the levels of a Gaussian pyramid as low-entropy representation for image compression. The set of L_i's is called a Laplacian pyramid. The L_i are computed as pixel-wise differences between G_i and its estimate $\widetilde{G}_i = \text{expand}(G_{i+1})$, obtained by supersampling G_{i+1} to the higher resolution and interpolating the missing values. Fig. 3.1(b) shows the Laplacian pyramid for the example. It decomposes the image into a sequence of spatial frequency bands. Perfect reconstruction of G_0 is possible when G_k and $L_0, L_1, \ldots, L_{k-1}$ are given by using the recursion $G_i = \widetilde{G}_i + L_i$. Since for natural images the values of L_i are mostly close to zero, they can be compressed using quantization. The reconstruction proceeds in a top-down fashion. Thus, progressive transmission of images is possible with this scheme.

Since the pyramid has a tree structure, it can be pruned to reduce its size. This method works well if the significant image details are confined within small regions. Figure 3.2 shows an image of a letter with size $2,048 \times 1,412$. Most of the area can be represented safely by using only the lower resolution levels, while the higher resolutions concentrate at the edges of the print. Although pruning compresses the image by a ratio of 150:1, the address is still clearly readable.

Another application of image pyramids is hierarchical block matching, proposed by Bierling [31] for motion estimation in video sequences. Since the higher levels of the pyramid are increasingly invariant to translations, image motion is estimated in the coarsest resolution first. The estimated displacement vectors are used as starting

points for block matching in finer resolutions. This makes the matching process faster and more reliable, compared to matching in the highest resolution only.

Wavelets. In image pyramids, the image is represented by a single feature, typically the smoothed intensity, at a certain resolution. Two-dimensional discrete dyadic wavelet analysis is a way to construct invertible multiscale image representations that describe an image location with typically three features per level.

A wavelet is a square-integrable function ψ with zero average: $\int_{-\infty}^{+\infty} \psi(x)dx = 0$, which is dilated with a scale parameter s, and translated by u: $\psi_{u,s}(x) = \frac{1}{\sqrt{s}}\psi(\frac{x-u}{s})$. These functions are localized in the space-frequency plane with a space spread proportional to s and a frequency spread proportional to $1/s$. Thus, the product of space and frequency localization is constant, which corresponds to the Heisenberg principle of uncertainty [93]. The wavelet transform of a function f at a scale s and position u is computed by correlating f with a wavelet atom $\psi_{u,s}$: $W(u,s) = \int_{-\infty}^{+\infty} f(x)\psi_{u,s}(x)dx$.

By critically sampling the parameters $s = 2^j$ and $u = s \cdot n$ $\big((j,n) \in \mathbb{Z}^2\big)$, some wavelets form an orthonormal basis of $\mathbf{L}^2(\mathbb{R})$. The simplest orthonormal wavelet is the function proposed by Haar [88] that has a value of one in the interval $[0, \frac{1}{2})$, minus one in $[\frac{1}{2}, 1)$, and zero elsewhere. Daubechies [49] showed that smooth orthonormal wavelets with compact support exist as well. Most wavelets have an associated scaling function ϕ, which is used to generate them. It has an integral of one. The scaling function of the Haar wavelet has a value of one in the interval $[0, 1)$ and is zero elsewhere.

Mallat [151] proposed a fast algorithm for computing a critically sampled discrete wavelet transform (DWT). The high-resolution discrete signal is convolved with two quadrature mirror kernels of compact support and subsampled by a factor of two to separate it into an approximation and a representation of the details. The

(a)   (b)
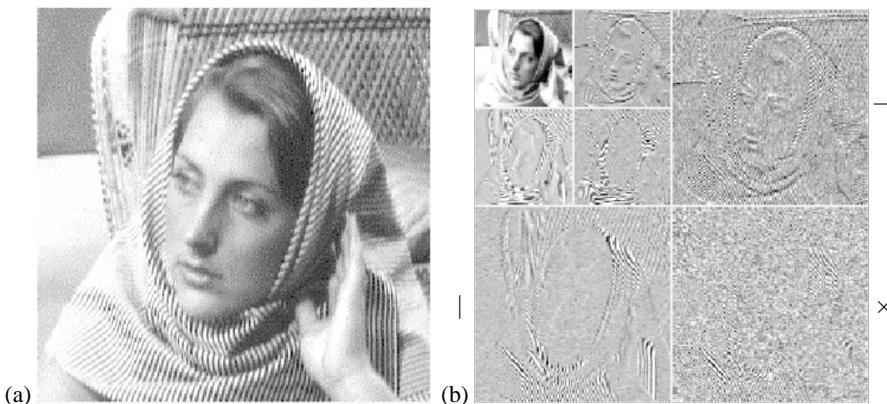
Fig. 3.3. Discrete wavelet transformation (DWT): (a) original image; (b) DWT decomposition after application to two levels. Each application separates the image into smaller scale horizontal ($-$), vertical ($|$), and diagonal ($\times$) detail images as well as a subsampled intensity image for which the DWT can be applied recursively.

approximation is produced by a low-pass kernel L that is associated with the scaling function ϕ, while the details are produced by a high-pass kernel H associated with the wavelet ψ. Perfect reconstruction of the signal is possible by supersampling the approximation and the details and convolving with reversed kernels.

For two-dimensional signals, such as images, the decomposition is applied consecutively to both dimensions, e.g. first to the rows and then to the columns. This yields four types of lower-resolution coefficient images: the approximation produced by applying two low-pass filters (LL), the diagonal details, computed with two high-pass kernels (HH), and the vertical and horizontal details, output of a high-pass/low-pass combination (LH and HL). This is illustrated in Figure 3.3. The low-resolution approximation of the signal can be decomposed recursively by applying the same procedure. The resulting representation has the same size as the input image with $\frac{3}{4}$ of the coefficients describing the details of the finest resolution.

One of the major applications of wavelets is image compression and denoising. It relies on the fact that most natural images are represented sparsely in wavelet coefficient space. Furthermore, additive zero mean i.i.d. Gaussian pixel noise spreads uniformly over the coefficients. Thus, setting small coefficients to zero and keeping only the few significant ones yields compression and suppression of noise. Donoho and Johnstone [55] showed that such a wavelet shrinkage in an appropriate basis can be a nearly optimal non-linear estimator for noise reduction.

Wavelet representations are also used for other computer vision tasks. For instance, local maxima can be tracked through multiple resolutions to extract edges robustly [152]. Since many functions can be used as wavelets, the choice of the basis can be targeted to the application at hand. Coifman *et al.* [43] proposed to further decompose not only the approximation side of the coefficients, but also the details. This yields a nested sequence of wavelet packet decomposition trees that all form an orthonormal basis of the signal if the wavelet itself is orthonormal.

Fourier Transformation. The size of a level in the wavelet-representation decreases exponentially with height. Thus, the representational power also decreases. Higher levels of the wavelet decomposition represent only the coarse image structure, but it can be desirable to have a complete representation of the signal at each level of the hierarchy. One way to hierarchically transform one complete representation into another is the fast Fourier transformation (FFT), introduced by Cooley and Tukey [44].

A finite-energy signal f can be decomposed into a sum of sinusoids $\{e^{i\omega x}\}_{\omega \in \mathbb{R}}$: $f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\omega) e^{i\omega x} dx$, where $\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(x) e^{-i\omega x} dx$ is the Fourier transformation of f. The amplitude of $\hat{f}(\omega)$ describes, how much the sinusoidal wave $e^{i\omega x}$ contributes to the signal f.

For a discrete signal of length $N = 2^j$, it suffices to sample the frequency ω N times to form an orthonormal basis. The discrete Fourier transformation (DFT) is then: $F(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} e^{-i2\pi kn/N}$ ($k = 0, \ldots, N-1$). It can be computed efficiently by decomposing a N-point DFT into two DFTs of $N/2$ points that process the even samples $f_e(n) = f(2n)$ and the odd samples $f_o(n) = f(2n+1)$ separately:
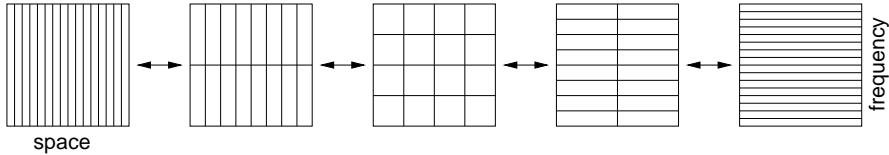
Fig. 3.4. Coverage of the space-frequency plane by the coefficients of the representations used in the fast Fourier transformation. The transformation maps a space-localized representation step by step into a frequency-localized representation. The size of the representation is not changed, since the number of cells and the area of a cell stays constant.

256×128 × 1×1   128×64 × 2×2   64×32 × 4×4   32×16 × 8×8   16×8 × 16×16
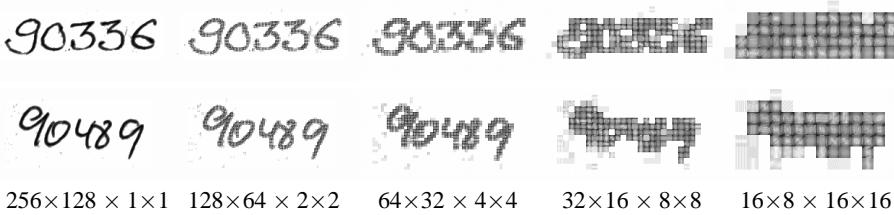
Fig. 3.5. Two-dimensional fast Fourier transformation. The absolute values of the first five hierarchy levels for two example images are shown on a logarithmic scale. White corresponds to zero.

$$
\begin{aligned}
FFT_N(k, f) &= FFT_{N/2}(k, f_e) + T_N(k)FFT_{N/2}(k, f_o); \\
FFT_N(k + N/2, f) &= FFT_{N/2}(k, f_e) - T_N(k)FFT_{N/2}(k, f_o); \\
k = 0, \ldots, N/2 - 1; &\qquad T_N(k) = e^{-i2\pi k/N}.
\end{aligned}
$$

These smaller DFTs can be decomposed recursively, until the number of pixels decreases to two, where the DFT is trivial. To compute an FFT coefficient, only two coefficients of the lower level need to be accessed. Information flows in a butterfly graph from all pixels in the signal to the FFT coefficients.

Figure 3.4 illustrates the coverage of the space-frequency plane by the coefficients of the representations used in the FFT. The transformation maps a representation localized in space into a representation localized in the frequency domain in $\log(N)$ steps. All intermediate representations describe the complete signal that can be reconstructed perfectly from each level. Thus, operations that are local in space can be applied in the space representation, while operations local in frequency can be applied in the frequency domain.

The FFT can be generalized to two-dimensional signals (images) by applying it to each dimension separately. Figure 3.5 shows the first five steps of the 2D-FFT, applied to two 256×128 images which contain handwritten digit blocks. One can see how the energy that is concentrated at the lines in the space-localized representation is distributed by the transformation as the representation becomes more localized in the frequency domain. The produced intermediate representations are useful for operations that are local in space as well as in frequency. For instance, one can use the fact that the magnitudes of the FFT coefficients become increasingly invariant to translations in space.

The success of Gabor filters [79] also shows that the intermediate representations are interesting. These filters are localized in space at u and in frequency at ξ with Gaussian envelopes g:

$$g_{u,\xi}(x) = g(x-u)e^{i\xi x}; \qquad \hat{g}_{u,\xi}(\omega) = \hat{g}(\omega - \xi)e^{-iu(\omega-\xi)}.$$

Gabor filters resemble properties of V1 simple neurons in the human visual system and are very useful for texture discrimination [231], for example.

3.1.2 Neural Networks

The hierarchical image representations discussed so far had very few, if any, parameters to adapt to a specific set of images. Neural networks with more free parameters have been developed that produce representations which can be tuned to a dataset by learning procedures. These representations need not to be invertible since they are used, for instance, for classification of an object present in the image.

Neocognitron. One classical example of such adaptable hierarchical image representations is the Neocognitron, proposed by Fukushima [77] for digit recognition. The architecture of this network is illustrated in Figure 3.6. It consists of several levels, each containing multiple cell planes. The resolution of the planes decreases from the input towards the upper levels of the hierarchy. The cell planes consist of identical feature detectors that analyze a receptive field located in the input.

The size of the receptive fields increases with height, as do the invariance to small translations and the complexity of the features. The cells in the first level of the network analyze only a small input region and extract edge features. Cells located at the second level receive input from the edge features and extract lines and corners. Increasingly complex features, such as digit parts, are extracted at the third level. Feature detectors at the topmost level react to the entire image and represent digit classes.
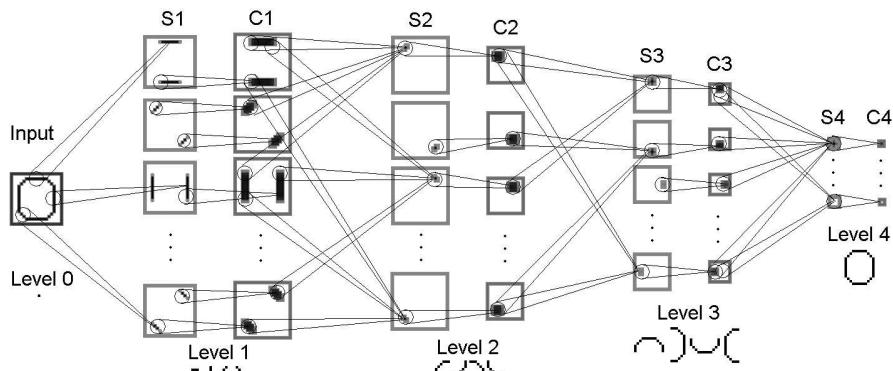
Fig. 3.6. The Neocognitron proposed by Fukushima [77]. Digit features of increasing complexity are extracted in a hierarchical feed-forward neural network.

Each level consists of three layers that contain different cell types. The S-layer is the first layer of a level. It contains S-cells that receive excitatory input via adjustable weights from small windows centered at the corresponding position in all C-planes of the layer below. S-cells in Level 0 access the input image directly. Not shown in the figure are V-cells that provide inhibitory input to the S-cells. V-cells are excited by all C-cells of the corresponding position in the lower level and compute a smoothed activity sum to control the gain of S-cells. The output $\phi(\frac{1+e}{1+ri} - 1)$ of an S-cell depends on the total excitation e, the total inhibition i, and a selectivity parameter r. It is passed through a rectifying function ϕ that is zero for negative activations. The weights and the selectivity are chosen such that the S-cell activity is very sparse. An S-cell reacts to features that resemble its specific excitatory weight matrix. All S-cells of a plane share the same weights and thus extract the same feature at different locations.

Invariance is produced in the network by the connections from the S-cells to the C-cells, which reside in the second layer of a level. These excitatory weights are not adjustable. They are prewired in such a way that a C-cell responds if any of the S-cells from a small window in the associated S-plane at the corresponding position is active. Hence, C-representations are blurred copies of S-activities that are less variant to input distortions.

The Neocognitron is trained level by level, starting at the bottom of the hierarchy. The adaptable excitatory weights of the S-cells can be trained either in a unsupervised mode or with supervision. For unsupervised training, the S-cells of a layer that correspond to similar positions first compete to react to an input pattern. The winning cell is then updated, such that it will react more strongly the next time the same pattern appears. In the supervised training mode [78], a human operator selects the features that a cell should respond to and the weights are updated according to a Hebbian rule that is multiplied with a Gaussian window to give the features in the center of the receptive field an advantage. Inhibition and excitation are increased simultaneously to make the cells more and more specific.

Although the network is able to learn to recognize distorted patterns from relatively few training examples, training has been reported to be rather difficult [147] due to the sensitivity of the network's performance to the choice of parameters like the S-cell selectivity r. It was recommended to chose a high selectivity in the lower levels and to decrease it towards the top of the hierarchy.

HMAX Model of Object Recognition. A modern version of a hierarchical feature extraction network is the HMAX model, proposed by Riesenhuber and Poggio [192]. The architecture of the network is sketched in Figure 3.7. Similar to the Neocognitron, it consists of alternating S-layers and C-layers. The S-layers contain feature extracting cells that compute a weighted sum of their inputs, followed by a rectifying transfer function. S-cells receive their inputs from C-cells at corresponding positions in the next lower layer. C-cells are used to pool a group of S-cells that share some parameters, but differ in one or more other parameters. They compute the maximum of the activities of these S-cells. Hence, C-cell responses are invariant to the parameters spanned by their associated S-cells.
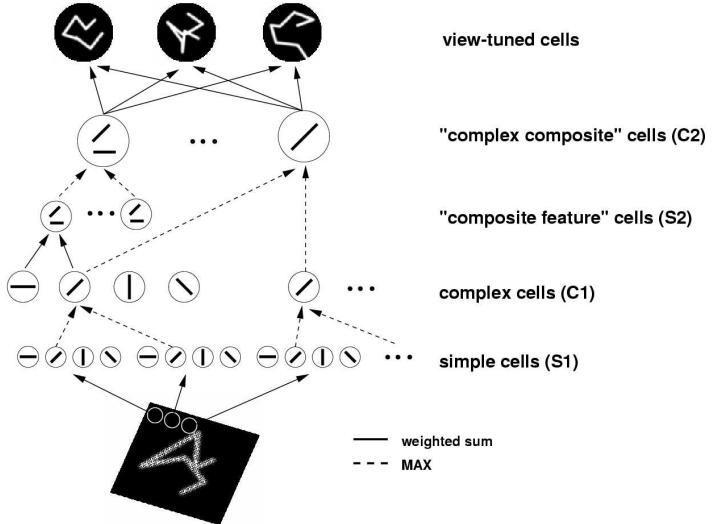
Fig. 3.7. HMAX model of object recognition proposed by Riesenhuber and Poggio. The network consists of alternating S-layers and C-layers that extract features of increasing complexity, size, and invariance. S-cells extract features by template matching while C-cells produce invariance by pooling of S-cells with a maximum operator (image from [192]).

Again, when going up the hierarchy, the receptive field size of the feature detectors is enlarged, the feature complexity rises, and the responses become more and more invariant to input transformations, such as shifts or rotations. Cells in layer S1 correspond to V1 simple cells. They analyze the 160×160 input image and extract oriented features at different positions, scales, and orientations. Space is sampled at every pixel, 12 scales are used, and four orientations are extracted, yielding 1,228,800 cells. The huge number of S1-cells is reduced in layer C1 to 46,000 by pooling cells with the same orientation, similar position, and similar scale. C1 cells correspond to V1 complex cells that detect oriented image structure invariant to the phase. S2 cells receive input from 2×2 neighboring C1 units of arbitrary orientation, yielding a total of almost three million S2 cells of 256 different types. They detect composite features, such as corners and line crossings. All cells of a certain type are pooled to a single C2 cell that is now totally invariant to stimulus position. At the top of the hierarchy reside view-tuned cells that have Gaussian transfer functions. They receive input from a subset of typically 40 of the 256 C2 cells.

Almost all weights in the network are prewired. Only the weights of the view-tuned cells can be adapted to a dataset. They are chosen such that a view-tuned unit receives inputs from the C2 cells most active when the associated object view is presented at the input of the network.

Riesenhuber and Poggio showed that these view-tuned cells have properties similar to the cells found in the inferotemporal cortex (IT). They also demonstrated that view-invariant recognition of 3D paper clips is possible by combining the outputs of units tuned to different views of an object. In addition, the model was used re-

cently for categorization tasks, such as the distinction of images showing dogs and cats. Riesenhuber and Poggio argue that in such an architecture the binding problem might not be as severe as originally perceived [192]. Since the lower levels of the hierarchy contain retinotopic representations, features of spatially separated objects do not interact and hence are bound by spatial proximity. Features in the higher levels are complex combinations of simple features. Since there are many such combinations, it is unlikely that the features of two objects can be combined to a valid third object. However, the experiments showed that recognition performance decreased slightly when two non-overlapping objects were present, but recognition was impaired severely if the two objects overlapped.

The HMAX architecture was designed to recognize a single object in a feed-forward manner. The use of the maximum operation for pooling makes the cell responses invariant to input transformations and also suppresses noise. The response of a C-cell that reacts to a feature is not changed by nearby clutter, as long as the strongest S-cell response to the feature is stronger than the S-responses to the distractor. However, a C-cell cannot tell the difference between one or more instances of the same feature within its receptive field.

Convolutional Networks. The creation of features by enumeration of all possible subfeature-combinations is easy, but computationally inefficient. For practical applications, such as optical character recognition (OCR) and the interpretation of handwritten text, the network size plays an important role since real-time conditions must be met for the network recall.

If more of the network parameters can be adapted to a specific task, smaller networks suffice to extract the relevant features. One example of a fully adaptable hierarchical neural network is the convolutional network proposed by LeCun *et al.* [133] for the recognition of isolated normalized digits. A recent version of such a network, which is called LeNet-5 [134], is illustrated in Figure 3.8.

The network consists of seven layers and an input plane that contains a digit. It has been normalized to 20×20 pixels and centered in the 32×32 frame. The input intensities are scaled such that the white background becomes -0.1 and the black
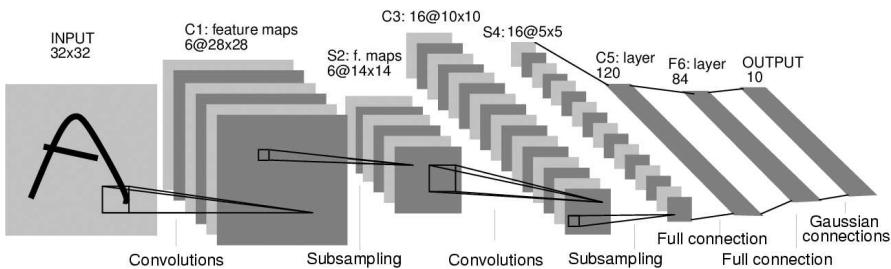
Fig. 3.8. Convolutional neural network LeNet-5 developed by LeCun *et al.* [134] for digit recognition. The first layers compute an increasing number of feature maps with decreasing resolution by convolution with 5×5 kernels and subsampling. At the higher layers, the resolution drops to 1×1 and the weights are fully connected (image adapted from [134]).

foreground becomes 1.175 to obtain inputs with approximately zero mean and unit variance.

The first five network layers are alternating convolutional (C) and subsampling (S) layers that contain an increasing number of feature maps. A convolutional layer computes local image features by convolving the previous representation with 5×5 kernels. These layers decrease in size since only such pixels for which the receptive field lies entirely in the previous layer are computed. If the previous representation consists of multiple feature maps, multiple feature windows describing the same image location are combined to compute a more complex feature. For C1 and C5 all S-features of the previous layer are used, while C3 features access different subsets of at least three S2 features. The size of the feature maps is further reduced by the subsampling layers that compute the average of 2×2 windows of an associated feature map in the next lower C-layer. They have a single adaptable parameter which determines how this average is scaled.

The upper two layers of the network have full connectivity with the previous layer. Layer F6 has a size of 7×12 and represents the desired output in a distributed code which is an icon that looks like an idealized digit. This has the advantage that similar patterns are represented by similar icons, facilitating postprocessing if these patterns are confused. The neurons in the first six layers of the network pass their activations through a sigmoidal transfer function $f(a) = \alpha \tanh(\beta a)$ that limits the output values to $[-\alpha, +\alpha](\alpha = 1.7159, \beta = \frac{2}{3}$, such that $f(1) = 1, f(-1) = -1$ and $|f''(a)|$ is maximal at 1 and -1). In contrast, the 10 output units in the topmost layer compute the difference between their weight vector and the F6 activity and pass it through a Gaussian transfer function. Hence, they are radial basis function (RBF) units that signal the class of the digits in a 1-out-of-10 code.

While the weights of the RBF-units are fixed to represent the icon associated with the class, all other weights are trained by gradient descent. The gradients of the weights with respect to a loss-function are computed by the backpropagation method [193]. Since shared weights are used, the gradients of the weight instances must be averaged when updating a weight. The degree of weight sharing is high in the lower levels of the network. This also allows for sharing of examples since many small windows are contained in a single digit. However, to train the relatively large number of about 60,000 weights present in the upper layers of the network, a large number of examples is needed. This can be seen by observing that the test error on the MNIST database [132] decreases from 1.7% to 0.95% when the size of the training set is increased from 15,000 to 60,000. Adding 540,000 digits with random distortions decreases the error further to 0.8%.

When trained with a high amount of salt-and-pepper noise (10% of the pixels inverted), the same network becomes quite invariant to variations in size, slant, aspect ratio, and stroke width of the digits. Figure 3.9(a) displays the response of the network to three different versions of the digit four.

While the performance of the network for the recognition of isolated normalized digits is impressive, in real-world situations it is difficult to segment the digits reliably. Explicit segmentation can be avoided by sweeping a recognizer along an input
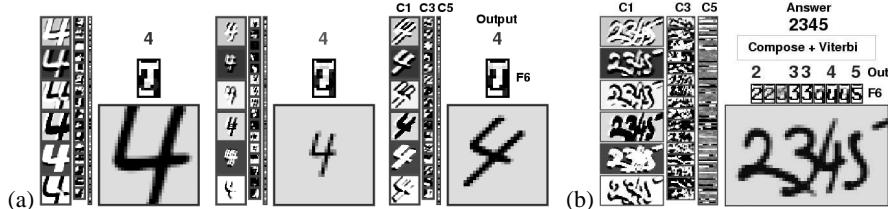
Fig. 3.9. Convolutional neural network activities: (a) isolated digits of different sizes and orientations can be recognized with LeNet-5; (b) digits need not to be segmented, but can be recognized in context by a larger SDNN network (images adapted from [134]).

string, but this is computationally expensive. A horizontally enlarged version of the LeNet-5 network, called space displacement neural network (SDNN), has been developed that also transforms the last two layers into feature maps. It is trained to recognize digits in their context. Since the digit positions and sizes are needed to generate the desired outputs, artificial three-digit blocks and blanks flanked by two digits were used for training. Figure 3.9(b) shows the response of the SDNN network to an example that is not easy to segment. The outputs of the network indicate the presence of digits. A postprocessing step is needed to merge multiple outputs for the same digit and to suppress spurious detections.

3.1.3 Generative Statistical Models

The feed-forward feature extraction used in the previous section is not the only way to implement discrimination. Since the distribution of images is far from being uniform, it is also possible to model the probability densities of certain object classes in image space and use the Bayes rule for inferring the class of an object from an observation and a generative model [57]. Generative models can also be used for purposes other than discrimination. For instance, they offer a systematic way to deal with missing data. Furthermore, generative image models frequently produce compact image descriptions that allow efficient transmission and storage.

In the following, three examples of hierarchical generative image models are reviewed: Helmholtz machines, hierarchical products of experts, and hierarchical Kalman filters.

Helmholtz Machine. The Helmholtz machine, proposed by Dayan *et al.* [50], is illustrated in Figure 3.10(a). It consists of several layers which contain binary stochastic processing units. These units are turned on with a probability $P(s_i = 1) = \sigma(a_i) = \frac{1}{1+e^{-a_i}}$. Two sets of weights, ϕ and θ, connect adjacent layers. Recognition weights ϕ_{ij} drive a unit j from the activities s_i of the units i in the next lower layer. These weights are used in the so called wake-mode of the network to compute higher level representations from the input. Generative weights θ_{kj} work in the opposite direction. A unit j is driven from the units k in the next higher layer in the so called sleep-mode. In this mode, higher-level representations are expanded to lower-level 'fantasies':
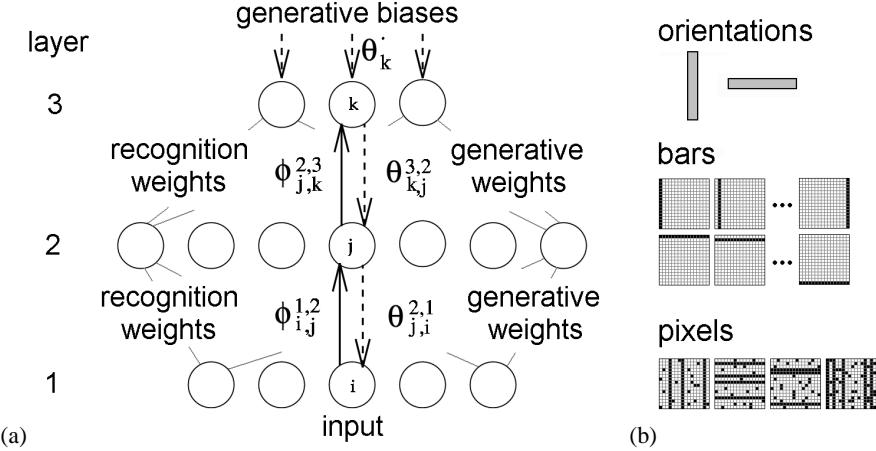
Fig. 3.10. Helmholtz machine, proposed by Dayan *et al.* [50] for discovering hierarchical structure in data: (a) sketch of the architecture (image adapted from [50]); (b) illustration of the bars problem.

$$[\text{wake}] \quad q_j = \sigma(\sum_i s_i \phi_{ij} + \phi_{0j}); \qquad p_j = \sigma(\sum_k s_k \theta_{kj} + \theta_{0j}) \quad [\text{sleep}].$$

To estimate network parameters, Hinton *et al.* [98] proposed the wake-sleep algorithm. The recognition weights ϕ are trained in the sleep mode to reproduce the higher-level representations from the generated fantasies. Symmetrically, the generative weights θ are trained during the wake phase to produce fantasies from the higher-level representations that match the current inputs:

$$[\text{wake}] \quad \Delta\theta_{kj} = \epsilon s_k(s_j - p_j); \qquad \Delta\phi_{ij} = \epsilon s_i(s_j - q_j) \quad [\text{sleep}].$$

Frey *et al.* [75] showed that this algorithm is able to discover hierarchical structure from data. They used the bars problem, illustrated in Figure 3.10(b). Data is generated as follows. First, it is decided randomly if the vertical or the horizontal orientation is used. Next, the lines or the columns of a 16×16 image are turned on with $P = 0.25$, depending on the chosen orientation. Finally, individual pixels are turned on with a probability of $P = 0.25$. The authors used a three-layer network with 36 units in the middle layer and 4 units in the top layer. To enforce a solution where individual bars are added to the image, the middle-to-bottom weights were constrained to be non-negative. The generative biases of the middle units were initialized to -4 to facilitate a sparse representation.

After running the wake-sleep algorithm, the generative weights of 32 middle units resembled vertical or horizontal bars. Furthermore, one of the top units indicated the orientation by exciting the vertical bar units and inhibiting the horizontal bar units in the middle layer. Hence, the network discovered the data generation mechanism. However, if the non-negativity constraint was not used and the bias was initialized to zero, the network did not find the optimal solution and modeled the data in a more complicated way.

Hierarchical Products of Experts. Another approach that makes the learning of multi-level statistical image models possible, is the products of experts (PoE) method that Hinton [97] recently proposed. Each expert specifies a probability distribution $p_m(\mathbf{d}|\theta_m)$ over the visible variables $\mathbf{d}$ and the n experts are combined by multiplying these distributions together and renormalizing: $p(\mathbf{d}|\theta_1,\ldots,\theta_n) = \prod_m p_m(\mathbf{d}|\theta_m)/\sum_{\mathbf{c}} \prod_m p_m(\mathbf{c}|\theta_m)$, where $\mathbf{c}$ enumerates all possible vectors in data space. The motivation for multiplying the experts is that the combined distribution can be much sharper than the individual expert models. For example, each expert can constrain only a small subset of the many image space dimensions and the product will constrain all of them if the subsets cover the dimensions. Furthermore, the PoE construction makes it easy to infer the values of the latent variables of each expert because the latent variables of different experts are conditionally independent, given the data.

One expert type for which this inference is tractable are restricted Boltzman machines (RBM) [218]. These networks consist of one visible layer and one hidden layer. They have no intralayer connections. The vertical connections between the binary stochastic units are symmetrical. Each hidden unit can be viewed as an expert since the probability of the network generating a data vector is proportional to the product of the probabilities that the data vector is generated by each of the hidden units alone [74].

Because it is time-consuming to train RBMs with the standard Boltzman machine learning algorithm, Hinton proposed to minimize not the Kullback-Leibler divergence, $Q^0 \| Q^\infty$, between the data distribution Q^0 and the equilibrium distribution of fantasies over the visible units Q^∞, but to minimize the difference, called contrastive divergence, between $Q^0 \| Q^\infty$ and $Q^1 \| Q^\infty$. Q^1 is the distribution of one-step reconstructions of the data that are produced by first choosing hidden states according to their conditional distribution, given the data, and then choosing binary visible states, given the hidden states. For image data this leads to the learning rule: $\Delta w_{ij} \propto \langle p_i p_j \rangle_{Q^0} - \langle p_i p_j \rangle_{Q^1}$, where p_i are the pixel intensities that have been scaled to [0,1], $p_j = 1/(1 + \exp(-\sum_i w_{ij} p_i))$ is the expected value of the hidden units, and $\langle . \rangle_{Q^k}$ denotes the expected value after k network updates.

Since the hidden-unit activities are not independent, they can also be viewed as data generated by a second PoE network. The hidden units of this second network will then capture some of the remaining structure, but may still have dependencies which can be analyzed by a third PoE network. Mayraz and Hinton [154] applied this idea to the recognition of handwritten digits. They trained a separate hierarchy of three PoE networks for each digit class using the MNIST [132] dataset. After training, they observed that the units of the first hidden layer had localized receptive fields, which described common local deviations from a class prototype. They used $\prod_m p_m(\mathbf{d}|\theta_m)$ as log-probability scores to measure the deviation of a digit from a class-model. All 30 scores were fed to a linear classifier which was trained on a validation set. When 500 hidden units were used in each layer, a test set error rate of 1.7% was observed.

Hierarchical Kalman Filters. If one does not use binary stochastic processing units, but the generation model is a weighted sum of basis functions with added Gaussian noise, inference is tractable as well. The Kalman filter [116] allows to infer the hidden causes from data, even if the causes change in time according to a linear dynamical system. Rao [186] proposed using Kalman filters to learn image models. Segmentation and recognition of objects and image sequences was demonstrated in the presence of occlusions and clutter.

To account for extra-classical receptive-field effects in the early visual system, Rao and Ballard [187] combined several simplified Kalman filters in a hierarchical fashion. In this model, static images I are represented in terms of potential causes $\mathbf{r}$: $\mathsf{I} = \mathsf{U}\mathbf{r} + \mathbf{n}$, where $\mathbf{n}$ is zero mean Gaussian noise with variance σ^2. The matrix U contains the basis vectors U_j that mediate between the causes and the image. To make the model hierarchical, the causes $\mathbf{r}$ are represented in terms of higher-level causes $\mathbf{r^h}$: $\mathbf{r} = \mathbf{r^{td}} + \mathbf{n^{td}}$, where $\mathbf{r^{td}} = \mathsf{U^h}\mathbf{r^h}$ is a top-down prediction of $\mathbf{r}$ and $\mathbf{n^{td}}$ is zero mean Gaussian noise with variance σ_{td}^2.

The goal is now to estimate, for each hierarchical level, the coefficients $\mathbf{r}$ for a given image and, on a longer time scale, learn appropriate basis vectors U_j. This is achieved by minimizing:

$$E = \frac{1}{\sigma^2}(\mathsf{I} - \mathsf{U}\mathbf{r})^T(\mathsf{I} - \mathsf{U}\mathbf{r}) \ + \ \frac{1}{\sigma_{td}^2}(\mathbf{r} - \mathbf{r^{td}})^T(\mathbf{r} - \mathbf{r^{td}}) \ + \ g(\mathbf{r}) \ + \ h(\mathsf{U}),$$

where $g(\mathbf{r}) = \alpha \sum_i \mathbf{r}_i^2$ and $h(\mathsf{U}) = \lambda \sum_{i,j} \mathsf{U}_{i,j}^2$ are the negative logarithms of the Gaussian prior probabilities of $\mathbf{r}$ and U, respectively. The two first terms of E describe the negative logarithms of the probability of the data, given the parameters. They are the squared prediction errors for Level 1 and Level 2, weighted with the inverse variances.

An optimal estimate of $\mathbf{r}$ can be obtained by gradient descent on E with respect to $\mathbf{r}$:

$$\frac{d\mathbf{r}}{dt} \ = \ -\frac{k_1}{2}\frac{\partial E}{\partial \mathbf{r}} \ = \ \frac{k_1}{\sigma^2}\mathsf{U}^T(\mathsf{I} - \mathsf{U}\mathbf{r}) \ + \ \frac{k_1}{\sigma_{td}^2}(\mathbf{r^{td}} - \mathbf{r}) \ - \ k_1\alpha\mathbf{r},$$

where k_1 is a positive constant. This computation is done in the predictive estimator (PE) module, sketched in Figure 3.11(a). It combines the bottom-up residual error $(\mathsf{I} - \mathsf{U}\mathbf{r})$ that has been passed through U^T with the top-down error $(\mathbf{r^{td}} - \mathbf{r})$ to improve $\mathbf{r}$. Note that all the information required is available locally at each level.

A synaptic learning rule for adapting the weights U can be obtained by performing gradient descent on E with respect to U after the estimate $\mathbf{r}$ becomes stable:

$$\frac{d\mathsf{U}}{dt} \ = \ -\frac{k_2}{2}\frac{\partial E}{\partial \mathsf{U}} \ = \ \frac{k_2}{\sigma^2}(\mathsf{I} - \mathsf{U}\mathbf{r})\mathbf{r}^T \ - \ k_2\lambda\mathsf{U},$$

where k_2 is the learning rate. This is a Hebbian [91] type of learning with weight decay.

Rao and Ballard applied this optimization to the three-layered network sketched in Figure 3.11(b). In Level 0, three 16×16 image patches enter the network which
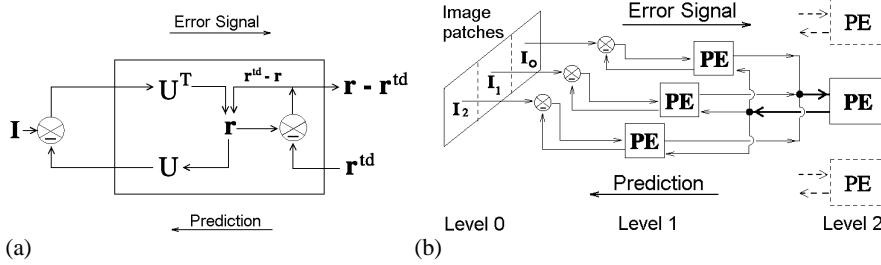
Fig. 3.11. Hierarchical Kalman filter proposed by Rao and Ballard [187]: (a) predictive estimator (PE) module that integrates top-down predictions $\mathbf{r}^{\mathbf{td}}$ and a feed-forward error signal $(\mathsf{I}-\mathsf{U}\mathbf{r})$ to an estimate $\mathbf{r}$ of the causes of an image I; matrix U mediates between the image and the causes; (b) general architecture of the system: local PEs are combined by a higher-level PE (images adapted from [187]).
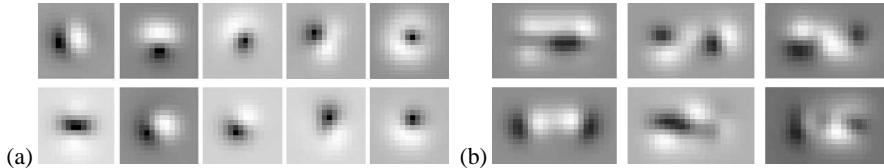
Fig. 3.12. Hierarchical Kalman filter receptive fields: (a) Level 1 receptive fields resemble Gabor-like responses of simple cells; (b) Level 2 receptive fields cover a larger area and are more complex (images from [187]).

have been passed through a center-surround filter and have been weighted with a Gaussian window. They are extracted from adjacent image windows that have an offset of 5 pixels horizontally. Level 1 contains three identical PEs that maintain $\mathbf{r}$ with 32 neurons. On Level 2, a single PE receives input from all three local PEs and represents $\mathbf{r}^{\mathbf{h}}$ with 128 neurons. Its receptive field has a size of 26×16 pixels.

Some of the receptive fields that emerge when the network is trained on natural image patches are shown in Figure 3.12. The Level 1 neurons have Gabor-like receptive fields that detect local orientation. These responses resemble V1 simple cells. Level 2 neurons have more complex receptive fields that are obtained by combining Level 1 features.

Rao and Ballard demonstrated that Level 1 neurons display end-stopping behavior that is explained by predictive coding. Since longer oriented lines are more probable in natural images than short lines, an orientation-selective cell responds stronger to a short line inside its classical receptive field than to a longer line, which can be predicted by a higher-level module. Since the cell signals only the difference between this prediction and the input, it is less active.

Such a predictive coding scheme could be an efficient way to communicate between the levels of the visual system. It removes redundancy because only those parts of the signal that are not already known to the receiver are sent. Several mechanisms in the visual system can be viewed from this perspective. Center-surround receptive fields in the retina and the LGN compute the difference between the cen-

ter's intensity and its prediction from the surroundings. Phasic responses of cells indicate a difference between the actual input to a cell and its prediction from past values. Color-opponent channels might reflect predictive coding in the chromatic domain since the wavelength-response profiles of the three cone types overlap. Evidence for predictive coding in higher visual areas like MT and IT also exists.

While some of these predictions can be computed locally, e.g. using lateral connections, it might well be that a hierarchy of PEs explains the functional role of reciprocal feed-forward/feedback connections in the visual system.

3.2 Recurrent Models

Although it was not the focus of the previous section, the hierarchical Kalman filter already used the concept of recurrent computation to infer hidden causes from observations. While feed-forward networks transform an input x into an output $y = f(x)$, recurrent networks respond both to the input and their own state. In the discrete-time case this can be described by: $y_{t+1} = f(y_t, x)$.

Such iterative computation is common in mathematics and scientific computing for problems where closed-form solutions cannot be found or are too expansive to compute. One of the best known examples of iterative algorithms is Newton's method [167] for computing the root of a function. The general idea is to provide an initial guess for the root and to apply a simple method for the improvement of the approximation that is applied repeatedly, until the solution is good enough.

Recurrent computation is much more powerful than feed-forward computation. While feed-forward neural networks with a single hidden layer can approximate any continuous function over a compact domain, they may need exponentially many hidden units to solve this task. In contrast, recurrent neural networks of finite size can emulate a Turing machine in linear time [211]. One striking example that demonstrates the advantages of the use of recurrence is the parity function with many inputs. Feed-forward networks with a single hidden layer have difficulties learning the parity problem for two inputs and need $\Theta(2^n)$ hidden units for n inputs. Recurrent networks that process the inputs in a serial fashion need to store only a single bit representing the current sum of the input bits modulo two. Similar recurrent circuits are widely used in VLSI designs.

On the other hand, the increase of computational power comes at a cost. First, each processing element must not only be computed once, but in every time step. This may slow down simulation of recurrent networks on a serial machine. Second, the non-linear dynamics, described by the recurrent network, can produce rich behaviors that do not necessarily reflect the intentions of the network designer. Care must be taken to avoid runaway activity, convergence to uninteresting attractors, oscillations, and chaotic behavior if they are not desired.

Despite these difficulties, recurrent neural networks have been used for a wide range of applications. Associative memories store patterns and allow content addressable information retrieval with noisy and incomplete data [172]. Recurrent networks have also been used for spatio-temporal pattern analysis, e.g. for speech

recognition [173]. In addition, small recurrent neuro-controllers [175] have been designed that solve non-trivial control tasks. In the last years, it has been realized that Pearl's belief propagation algorithm [177] can be applied to graphical probability models that contain loops [76]. These message-passing schemes have been used successfully for the decoding of error-correcting codes [155]. Last, but not least, recurrence has been successfully applied to combinatorial optimization problems [217].

The concepts of attractors and energy functions have been central to the theory of recurrent neural networks. Hopfield [101] investigated symmetrically connected networks with binary units that were asynchronously updated. He showed that each update does not increase an energy function $E = -\frac{1}{2}\sum_{ij} w_{ij} S_i S_j$, where S_k is the state of unit k and w_{ij} is a weight connecting units i and j. This yields monotonic convergence of the network's state towards an attractor that has a locally minimal value of the energy E.

The deterministic Hopfield network might get trapped in local minima of the energy function. To avoid this, stochastic neural units have been introduced. This leads to the Boltzman machine that samples the states of the network according to their Boltzman probability distribution [1]. To adapt the distribution of the visible units of a Boltzman machine to a desired distribution, a simple learning algorithm [2] is available. It performs gradient descent on the divergence between the two distributions. Although learning is slow, hidden units allow Boltzman machines to capture the higher order statistics of a data distribution.

Because fully connected recurrent networks have too many free parameters to be applicable to image processing tasks, in the following, models that have specific types of recurrent connectivity are reviewed: lateral interactions, vertical feedback, and the combination of both.

3.2.1 Models with Lateral Interactions

Lateral interactions are the ones that are easiest to realize in the cortex, since they require only short links between neighboring neurons within a feature map. Hence, it is likely that the neurons of the visual system have been arranged such that the most intensive interactions can be realized with lateral links. Lateral interactions have also been used in some image processing algorithms.

For instance, the compatibility between a recognized primitive and its neighborhood is the basis for relaxation labeling [195] techniques. The compatibilities define constraints for the interpretation of image patches which are satisfied iteratively using stochastic label updates. Relaxation labeling has been applied to edge linking and to segmentation problems.

Another example for the use of lateral interactions in image processing is anisotropic diffusion [178]. Here, the image is smoothed by a diffusion process that depends on the local intensity gradient. Thus, smoothing occurs tangential to an edge, but not in the direction orthogonal to the edge. Anisotropic diffusion is a robust procedure to estimate a piecewise constant image from a noisy input image.

Models that involve lateral interactions can be found in the neural networks literature as well. In the remainder of this section, some of these models are reviewed.

Linear threshold networks with local excitation and global inhibition. Among the simplest models of lateral interaction are models with global inhibition. Global inhibitory networks can, for instance, implement a winner-take-all dynamics. Hahnloser [89] analyzed networks with excitatory linear threshold units and a single inhibitory unit that computed the sum of excitatory activity. When the excitatory units implement a perfect autapse, a unit that maintains its activity by self-excitation, only network states with a single active neuron are stable. This neuron is the one that receives the strongest external input. All other units have an output of exactly zero because the global feedback lowers the activity below the threshold of the transfer function.

The behavior of the network is more complex if the excitatory units interact directly. Hahnloser *et al.* [90] designed a chip consisting of a ring of neurons with local excitatory connections. A single neuron computed the average activity and provided global inhibitory feedback.

The analysis of the network demonstrated the coexistence of digital selection and analog sensitivity. The authors identified two types of neuron subsets in the network. The activity of forbidden sets is not stable, while persistent activity of a permitted set can be maintained by the network. It was shown that all subsets of permitted sets are also permitted and all supersets of forbidden sets are forbidden.

Local excitatory connections widen the set of active units in a winner-takes-all dynamics from a single unit to a patch of consecutive units that have a blob-shaped activity. In the network, a linear relation between the amplitude of the blob and the level of uniform background input exists.

If more than one unit receives external input, the network places the blob at the location of the strongest stimulus. The network also showed hysteresis behavior. An already selected stimulus wins the competition although a different unit receives a slightly larger input. If the difference between the two stimuli exceeds a threshold, the activity blob jumps to the stronger stimulus.

Neural Fields. Amari [7] was among the first to analyze networks with lateral connectivity. He simplified the analysis by using a linear threshold activation function $f(x) = \max(0, x)$. Amari generalized the discrete neurons to a continuous field. The simplest case of such a model is a one-dimensional field consisting of one layer:

$$\tau \frac{\partial u(x,t)}{\partial t} = -u + \int w(x, x^*) f[u(x^*)] dx^* + h + s(x, t),$$

where $u(x)$ is the membrane potential at position x, and $h < 0$ determines the resting potential. Amari assumed space-invariant symmetric lateral connectivity $w(x, x^*) = \omega(|x - x^*|)$. For constant input $s(x)$ he proved the existence of five types of pattern dynamics:

– monostable field in which all excitations will die out,
– monostable field which is entirely excited,
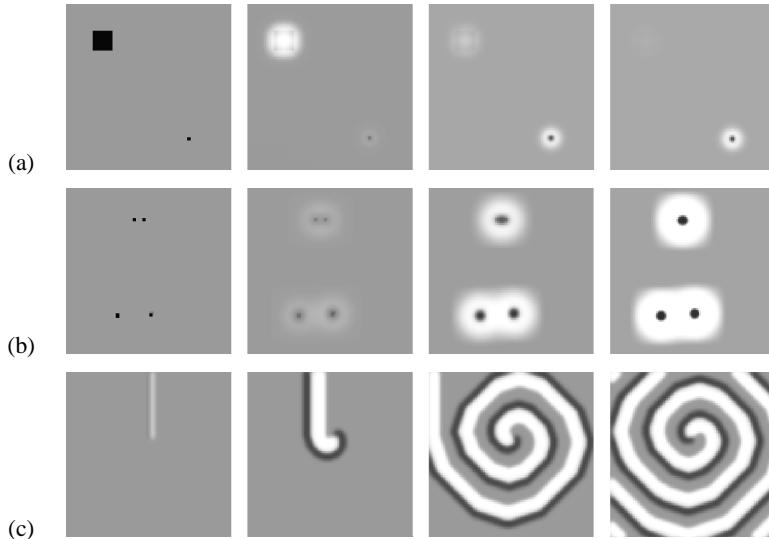
Fig. 3.13. Neural field dynamics: (a) vanishing and stable activity; (b) merger and coexistence of activity blobs; (c) spiral wave in a two-layered neural field (image adapted from [239]).

– explosive type bistable field in which localized excitations up to a certain range spread out without limit over the entire field, but vanish if the range of excitation area is too narrow,
– bistable field in which initial excitations either become localized excitations of a definite length or die out; localized excitations move in direction to the maximum of the input s, and
– field showing spatially periodic excitation patterns depending on the average stimulation level.

Most interesting is the coexistence of several stable blobs of activity that is achieved when the connectivity is positive in the center and negative for a larger neighborhood.

The complexity of the network's behavior increases if one adds a second layer to the field. In this case, one can further detect oscillatory patterns and traveling waves. The dynamics of neural fields is closely related to excitable media [156], which have the ability to propagate signals without damping. Such models have been used to describe a wide range of natural phenomena.

Wellner and Schierwagen [239] investigated the behavior of neural fields using simulations that were discrete in space and time. Figure 3.13 shows interesting cases of the field dynamics. Initial activity vanishes if it is too large or stabilizes if it fits the excitatory region of lateral interaction. If two small spots of initial activity are close together, they are merged to a single blob of sustained activity. However, if they are fare enough apart both blobs coexist.

Neural fields have been applied to several problems arising in perception and control. For instance, Giese [82] applied them to motion perception tasks. He used

temporally asymmetric Mexican hat shaped lateral interactions. Stable solutions were traveling pulses that followed a motion sequence. The lateral dynamics was used to integrate activity over time. Vijayakumar *et al.* [234] used neural fields as saliency map to control attention and to generate saccadic eye movements for a humanoid robot.

Cellular Neural Networks. While continuous neural fields facilitate analysis, they must be discretized to be applicable in practice. Chua and Roska [41, 42] proposed a simplified model that represents space with discrete cells, the cellular neural network (CNN). This network has a strictly local connectivity. A cell communicates e.g. to the cells within its 8-neighborhood. The space-invariant weights are described by templates. A cell is computed as follows:

$$C\frac{dx_{ij}}{dt}(t) = -\frac{1}{R}x_{ij}(t) + A_{ij,kl}y_{kl}(t) + B_{ij,kl}u_{kl}(t) + z; \quad y_{kl}, u_{kl} \in N(ij),$$

where A describes the influence of neighboring cells, B is the receptive field on the input u, and C and R determine the time-constant of a cell. Parameter z determines the resting potential. The output $y_{ij} = \sigma(x_{ij})$ of a cell is a non-linear function σ of its state x_{ij}. Frequently, a piecewise linear function that saturates at -1 and 1 is used. While above equation is used for continuous time, there are also discrete-time CNN variants.

The actual computation of the continuous network dynamics is done by relaxation within a resistor-capacity network. It is supplemented with logic operations and analog image memories in a universal CNN machine, used for image processing purposes. Low-level image processing operations, such as spatiotemporal filters, thresholding, and morphologic operations, have been implemented in this framework.

The CNN cells can also be combined with photosensors to avoid I/O bottlenecks. Analog VLSI implementations for focal plane processing up to a size of 128×128 [143] have already been realized. The massively parallel architecture achieves a throughput that would require a supercomputer if the same operations were realized with general-purpose CPUs.

The CNN approach has been applied to areas other than image processing. For instance, it has been used for the control of a walking hexapode robot [9] with 18
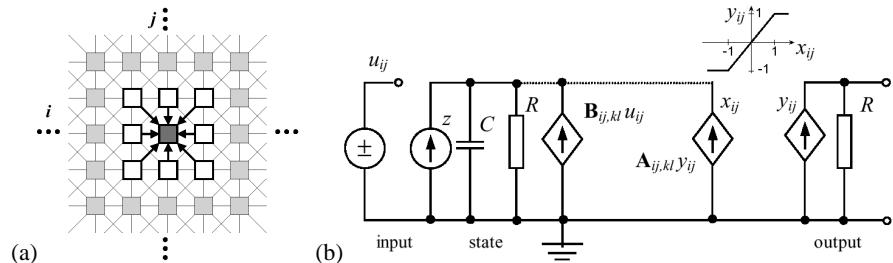
Fig. 3.14. Cellular neural network model of Chua and Roska [41, 42]: (a) processing elements are arranged in a grid and connected locally; (b) core cell of continuous time analog CNN.

degrees of freedom. A reaction-diffusion system is implemented for gait generation and altitude control. The gait is induced by a central pattern generator. Waves of activity propagate in a ring of cells. Individual legs receive movement commands at different times. Local sensors can change the locomotion pattern, e.g. if a joint saturates. The produced gait is a function of the sensory stimuli from the environment and the intended movement.

Recently, vertical interactions have been added to the CNN framework. For instance, Roska and Werblin proposed a ten-layered network as a model for retinal processing [197]. This model has been implemented with a CNN [14].

Model of Contextual Interaction in V1. The retina is only the first computational module in the human visual system. Lateral interactions are crucial in higher areas as well, like in the primary visual cortex. Recently, Li [141] proposed a model for lateral feedback in visual area V1. Its architecture is illustrated in Figure 3.15.

The model consists of a single sheet of columns. Each column represents oriented stimuli by several excitatory neurons that have different preferred orientations. The excitatory neurons are connected monosynaptically to excitatory neurons of similar orientation in their neighborhood if they are aligned on a straight line or an arc segment.
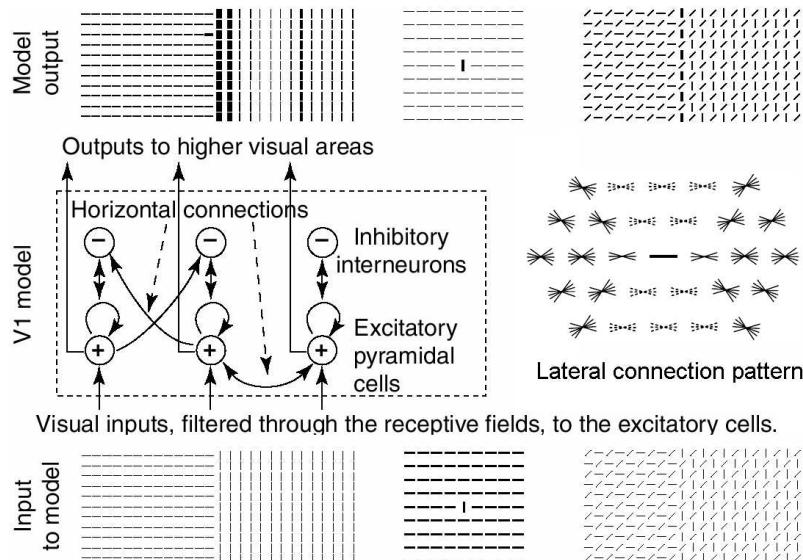
Fig. 3.15. Model of contextual interaction in V1 proposed by Li [141]. Each position is represented by several orientation-selective cells. Excitatory and inhibitory neurons form pairs that are reciprocally connected. Local lateral interaction is mediated by monosynaptic excitatory connections and disynaptic connections via inhibitory interneurons according to the connection pattern shown. Aligned cells of similar orientation excite each other, while non-aligned cells of similar orientation have inhibitory connections. The model's response to three different input images is also shown. The model performs texture segmentation, contour enhancement and perceptual pop-out (images adapted from [141, 142]).

Because the excitatory neurons cannot inhibit each other directly, each excitatory cell is accompanied by an inhibitory interneuron. The units in such an E-I pair are reciprocally connected. The lateral connection pattern to the interneurons is such that similar orientations in columns that are not aligned are suppressed. While the excitatory neurons have a transfer function which saturates for high inputs, the transfer function of the inhibitory neurons does not saturate. Hence, if the network's activity becomes too high inhibition exceeds excitation and lowers the activity again.

Initially, the network's activities are determined by the direct visual inputs within the classical receptive fields of the units. As the network dynamics evolves, the activities are quickly modulated by contextual influences mediated by the recurrent lateral connections. Li analyzed the network dynamics and demonstrated that the network performs the tasks of texture segmentation, contour enhancement, and perceptual pop-out. This is also illustrated in the figure.

Li [142] recently proposed that the contextual interactions in this V1 model, which make consistent stimuli more salient, represent bottom-up attentional effects. This mechanism discards inconsistent stimuli and focuses the limited resources of the higher visual system to the most salient objects.

Networks with Spiking Neurons. Lateral coupling of spiking neurons can be used to produce coherent firing. For instance, Hopfield and Brody [102, 103] proposed a network where different features that belong to the same object are laterally coupled. The coupling uses balanced excitation and inhibition and thus has little effect on firing rates. In this network, synchronization occurs if the features have approximately the same activity. The synchronized firing of neuron groups is recognized by a neuron in a higher layer that has short integration times and acts as a coincidence detector. Time-warp invariant recognition of real-world speech data was demonstrated in the network. However, Hopfield and Brody used a limited vocabulary of only ten words.

A similar idea was applied by Henkel [94] to the problem of stereovision. He used arrays of local disparity estimators with slowly changing parameters. Neighboring cells are coupled laterally. In this network, smooth changes of disparity produce coherent firing that represents dense disparity maps. Local ambiguities are resolved and noise is filtered out by the lateral interactions.

3.2.2 Models with Vertical Feedback

While horizontal connections mediate lateral interactions within an area, vertical connections link areas of different degrees of abstraction. The connections from lower areas, that are closer to the visual input and represent less complex features, to higher areas are called feed-forward or bottom-up links. They serve feature extraction purposes. The connections in the reverse direction are called top-down or feedback links. They expand abstract representations to less abstract ones.

In the following, some models that involve vertical feedback are reviewed.
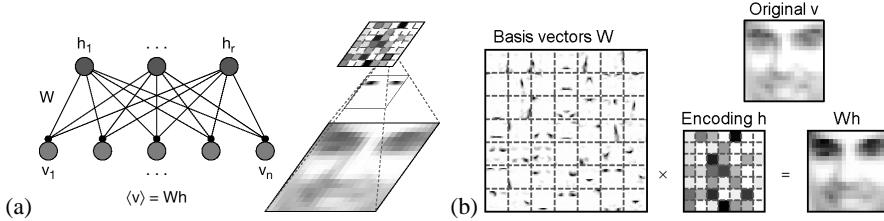
Fig. 3.16. Non-negative matrix factorization: (a) architecture of the network; (b) reconstruction of a face from its hidden representation; shown are also the extracted basis vectors and the activities of the hidden units (images from [137]).

Non-negative Matrix Factorization. Lee and Seung [137] recently introduced a generative data model that can be interpreted in terms of vertical feedback. They decompose a $n \times m$ non-negative matrix V approximately into non-negative matrix factors: $V \approx WH$. The m columns of V consist of n-dimensional data vectors. The r columns of W contain basis vectors of dimension n. Each data vector is represented by a column of H that contains r coefficients. This corresponds to a two-layered neural network which represents the data vector in a visible layer and the coefficients in a hidden layer. The matrix W describes the weights that connect both layers. This situation is illustrated in Figure 3.16(a).

One measure of the factorization quality is the square of the Euclidean distance $\|A-B\|^2 = \sum_{ij}(A_{ij} - B_{ij})^2$ between V and its reconstruction WH. $\|V-WH\|^2$ is minimized by the update rules:

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T WH)_{a\mu}}; \qquad W_{ia} \leftarrow W_{ia} \frac{(VH^T)_{ia}}{(WHH^T)_{ia}}.$$

Another measure is the divergence $D(A\|B) = \sum_{ij}(A_{ij} \log \frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij})$. $D(V\|WH)$ is minimized by:

$$H_{a\mu} \leftarrow H_{a\mu} \frac{\sum_i W_{ia} V_{i\mu}/(WH)_{i\mu}}{\sum_k W_{ka}}; \qquad W_{ia} \leftarrow W_{ia} \frac{\sum_\mu H_{a\mu} V_{i\mu}/(WH)_{i\mu}}{\sum_\nu H_{a\nu}}.$$

Lee and Seung [138] proved that these update rules find local minima of the respective objective functions. Each update consists of a multiplicative factor that is unity if $V = WH$. The multiplicative update does not change the sign of W or H. Hence, if they are initialized to positive values no further constraints are necessary to enforce their non-negativity.

The model was applied to a dataset that contained 2,429 normalized faces. The frontal views were hand-aligned in a 19×19 grid. Pixel intensities were linearly scaled such that their mean and standard deviation equaled 0.25 and were then clipped to the interval $[0, 1]$, where a value of zero corresponds to white. The matrices W and H were initialized randomly. The basis vectors that were present after 500 iterations of the update rule (minimizing the divergence) are shown in Figure 3.16(b). They consist of localized patches of high values that resemble typical

dark regions of facial images, such as the eyes and the shadow of the nose. The figure also shows the encoding h of a face and its reconstruction. Because both the weights and the coefficients of h contain a large number of vanishing components, the encoding is sparse. The reason for this is that the model is only allowed to add positively weighted non-negative basis-vectors to the reconstruction. Thus, different contributions do not cancel out, as for instance in principal components analysis.

Although the generative model is linear, inference of the hidden representation h from an image v is highly non-linear. The reason for this is the non-negativity constraint. It is not clear how the best hidden representation could be computed directly from W and v. However, as seen above, h can be computed by a simple iterative scheme. Because learning of weights should occur on a much slower time-scale than this inference, W can be regarded as constant. Then only the update-equations for H remain. When minimizing $\|v - Wh\|^2$, h is sent in the top-down direction through W. Wh has dimension n and is passed in the bottom-up direction through W^T. The resulting vector $W^T W h$ has the same number r of components as h. It is compared to $W^T v$, which is the image v passed in the bottom-up direction through W^T. The comparison is done by element-wise division yielding a vector of ones if the reconstruction is perfect: $v = Wh$. In this case, h is not changed.

When minimizing $D(v\|Wh)$, the similarity of v and its top-down reconstruction Wh is measured in the bottom-layer of the network by element-wise division $v_i/(Wh)_i$. The n-dimensional similarity-vector is passed in the bottom-up direction through W^T, yielding a vector of dimension r. Its components are scaled down with the element-wise inverse of the vector of ones passed through W^T to make the update factors for h unity if the reconstruction is perfect.

This scheme of expanding the hidden representation to the visible layer, measuring differences to the observations in the visible layer, contracting the deviations to the hidden layer, and updating the estimate resembles the operation of a Kalman filter [116]. The difference is that in a Kalman filter deviations are measured as differences and update is additive, while in the non-negative matrix factorization deviations are measured with quotients and updates are multiplicative. Because the optimized function is convex for a fixed W, the iterative algorithm is guaranteed to find the optimal solution.

Learning Continuous Attractors. In most models of associative memories, patterns are stored as attractive fixed points at discrete locations in state space, as
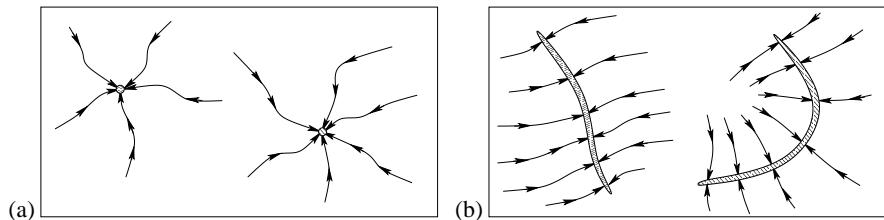
Fig. 3.17. Representing objects by attractors: (a) discrete attractors represent isolated patterns; (b) continuous attractors represent pattern manifolds (images after [209]).
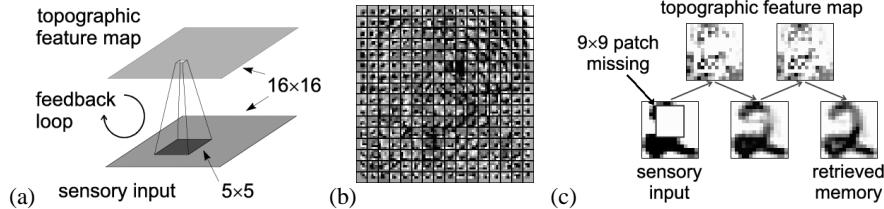
Fig. 3.18. Iterative pattern completion proposed by Seung [209]: (a) architecture of the network (two layers are connected by vertical feedback loops); (b) learned receptive fields (topographic feature map); (c) iterative pattern completion (images adapted from [209]).

sketched in Figure 3.17(a). For patterns with continuous variability, such discrete attractors may not be appropriate. Seung [209] proposed to represent continuous pattern manifolds with attractive manifolds of fixed points, continuous attractors, as illustrated in Figure 3.17(b). These attractors are parameterized by the instantiation or pose descriptors of the object. All instantiations have similar low energy, such that a change in pose can be achieved without much effort. When confronted with an incomplete pattern, the network dynamics quickly evolves towards the closest object representation. Thus, the incomplete pattern is projected orthogonally against the manifold and hence completed to a pattern with the same pose.

Seung suggested using a neural network with vertical feedback to learn such continuous attractors, as shown in Figure 3.18(a). The network consists of two 16×16 sheets of neurons that compute a weighted sum of their inputs, followed by a rectification nonlinearity. Both layers are connected by 5×5 local receptive fields. The sensory input is initialized to the incomplete pattern and trained to reconstruct the original pattern after two iterations. Normalized images of the handwritten digit two that have been degraded by setting a 9×9 patch, placed at a random location, to zero are used as incomplete patterns. By training with gradient descent on the completion error, the weights shown in Figure 3.18(b) emerge. They form a topographic map of localized oriented features. Figure 3.18(c) illustrates the reconstruction process for an example. One can see that the network is indeed able to fill-in the missing image parts. Note that this is not as difficult as it seems, since the network knows a-priori that the target image will be a normalized digit of class two.

Somato-Dendritic Interactions Integrating Top-Down and Bottom-Up Signals.
Siegel *et al.* [210] proposed a model that involves vertical feedback between two areas, as sketched in Figure 3.19(a). Both areas are reciprocally connected by excitatory axons. The excitatory neurons have two sites of synaptic integration. The apical dendrite integrates top-down influences, while bottom-up projections terminate in the basal dendritic tree. The areas also contain inhibitory neurons that project to all excitatory neurons.

Each area is modeled as a one-dimensional array. Both are connected by local retinotopic links. The neurons are simulated using a conductance-based model with active sodium and potassium conductances for spike generation. Synaptic conductances are implemented for glutamergic and two types of gabaergic transmission.
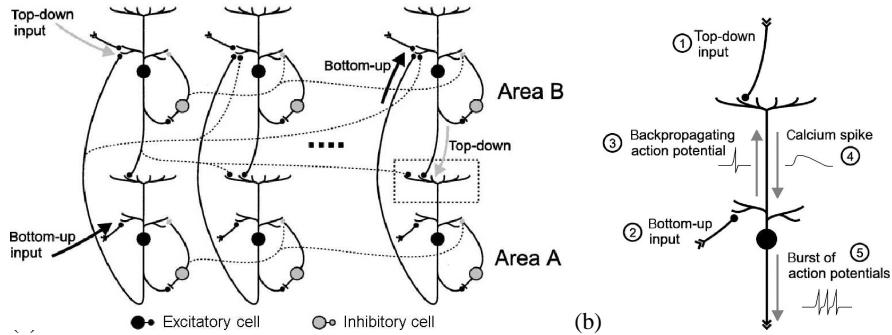
Fig. 3.19. Integrating top-down and bottom-up sensory processing by somato-dendritic interactions proposed by Siegel *et al.* [210]: (a) areas A and B reside at different hierarchical levels and are reciprocally connected (each area consists of excitatory and inhibitory neurons; inhibitory neurons project to all neurons within an area; excitatory neurons from both areas are connected by bottom-up and top-down projections); (b) somato-dendritic interaction and burst generation (if excitatory input of top-down projections ① is strong enough and bottom-up input ② initiates an action potential that propagates back into the apical dendrite ③, a dendritic calcium spike is triggered ④ that in turn causes a burst of action potentials ⑤) (images adapted from [210]).

The model reflects recently discovered physiological properties, such as the back-propagation of action potentials into the apical dendrite and the generation of slow dendritic calcium spikes that drastically lower the threshold for burst generation.

Siegel *et al.* [210] propose a functional interpretation for these somato-dendritic interactions. In the presence of a backpropagating action potential, the subthreshold top-down input at the apical dendrite can trigger a dendritic calcium spike leading to a burst of axonal action potentials, as illustrated in Figure 3.19(b). This burst signal is much more robust to noise than the total number of action potentials. It indicates a match between bottom-up and top-down stimuli. The authors also found that priming an interpretation of the bottom-up stimulus by additional input to the higher area leads to faster and more reliable recognition and biases processing if multiple stimuli are present.

The model accounts for the asymmetry of bottom-up and top-down pathways where feed-forward inputs mainly drive the activity of cells, whereas feedback has rather modulatory effects on the total spike counts. Nevertheless, the integration of top-down and bottom-up information leads to a robust burst signal. The authors propose that this bursting pattern could be a basis for the binding of corresponding high-level and low-level features.

3.2.3 Models with Lateral and Vertical Feedback

Neither lateral interactions nor top-down/bottom-up recurrent interactions alone are sufficient to explain the performance of the visual system. Because both types are present in the cortex, models that incorporate horizontal as well as vertical loops are
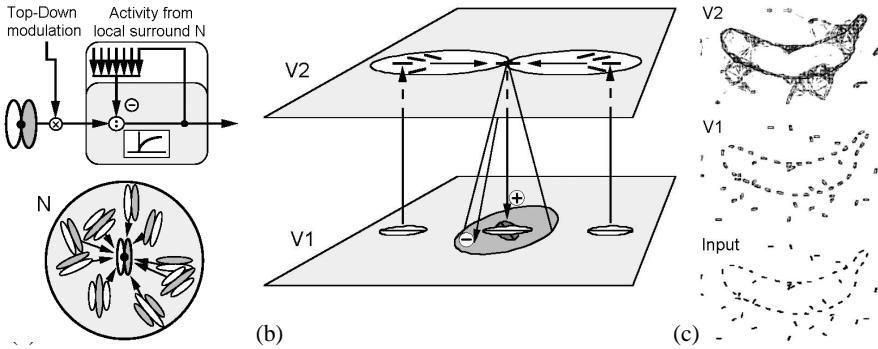
Fig. 3.20. Recurrent V1-V2 interaction proposed by Neumann and Sepp [166]: (a) computational units extract orientation selective features, are modulated by top-down feedback, and compete in a center-surround interaction; (b) V2 cells have two lobes that form a curvature template; they detect if aligned oriented features are present in both lobes; the corresponding orientation in V1 is excited in the center, while all other orientations in the surround are inhibited; (c) grouping of fragmented shape by cortico-cortical feedback processing (images from [166]).

good candidates for achieving fast and robust interpretation of visual stimuli. In the following, two such models are reviewed.

Recurrent V1-V2 Interaction. Neumann and Sepp [166] recently proposed a model for boundary processing in the early visual system. The model consists of two layers, V1 and V2, with local recurrent lateral connectivity that are connected retinotopically by vertical loops. The orientation selective processing elements in V1 are sketched in Figure 3.20(a). They detect local oriented contrast, are modulated by top-down feedback, and interact laterally in an on-center/off-surround pattern. This divisive interaction occurs in the space-orientation domain and implements a shunting inhibition. It amplifies salient features and suppresses noise.

Similar processing occurs in V2, where cells have two large lobes that act as curvature templates. A V2 cell becomes active only if both lobes are excited by aligned V1 features. This is achieved by multiplicative combination of the individual lobe's activations, yielding a nonlinear 'AND'-gate. V2 cells also compete laterally via center-surround interactions. The most active cell, locally, feeds back to the lower layer. In the center of its receptive field V1 neurons with compatible orientations are strengthened. This excitatory feedback is modulatory. It can amplify existing V1 activity, but cannot create V1 activity by itself. All other orientations in the larger V1 neighborhood are inhibited. This vertical V1-V2 interaction is illustrated in Figure 3.20(b). It leads to the representation of illusory contours in V2 that enhance compatible V1 features and suppress incompatible ones.

The network's response to a fragmented input pattern is shown in Figure 3.20(c). The line-segments defining the contour of an object are grouped together into a ring of coherent activity. The model is biologically plausible and was successfully used to reproduce psychophysical data. However, application to real-world problems seems to be difficult, since all connections in the system are prewired and it
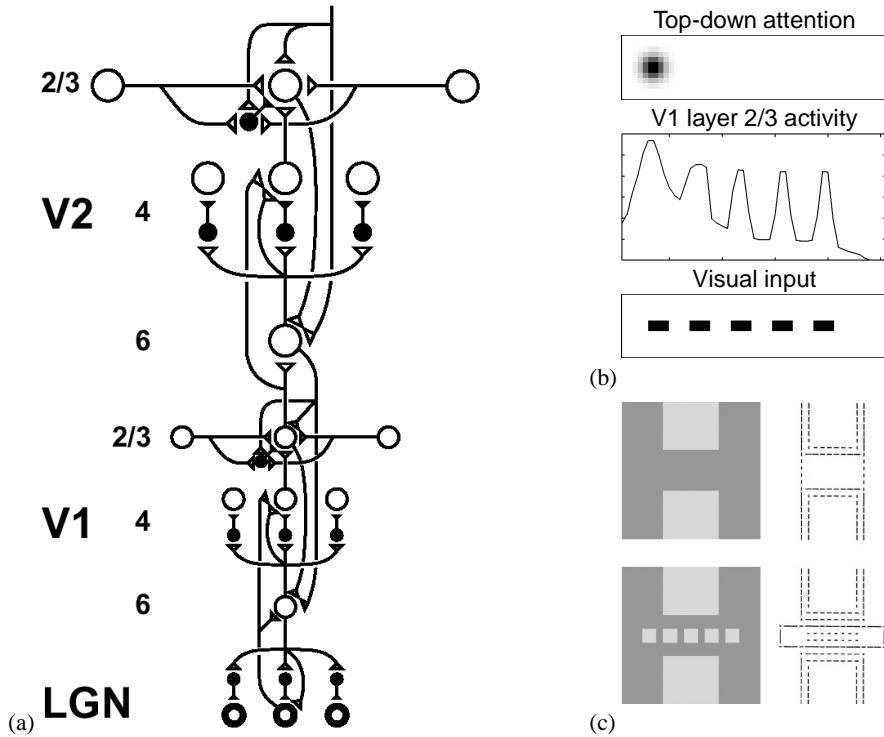
Fig. 3.21. Grouping and attention in the LAMINART model proposed by Grossberg [84]: (a) architecture of the system (feed-forward, feedback, and horizontal interactions within and between LGN, V1, and V2; open symbols indicate excitation; closed symbols represent inhibition); (b) top-down attention and bottom-up stimuli interact (the attention spreads along the illusory contour); (c) perceptual grouping (left: visual input; right: V1 response; top: vertical grouping; bottom: horizontal grouping) (images from [183, 85]).

is not clear, how to extend the model to represent more complex features in higher visual areas.

LAMINART Model. Grossberg [84] proposed a model for the laminar circuits in V1 and V2 that he called LAMINART. This model accounts for perceptual effects, such as grouping, orientation contrast, and attention. It is based on lateral and vertical feedback connections that have been found in the cortex. The architecture of the model is sketched in Figure 3.21(a).

Three visual areas are arranged in a hierarchical fashion. LGN is connected to V1 via vertical feedback loops. The same connectivity pattern exists between V1 and V2. On-center/off-surround type interactions between adjacent layers implement an ART-like resonance [40] between features of different complexity. Corresponding features strengthen each other, while incompatible features are inhibited.

Within an area, horizontal connections facilitate perceptual grouping of collinear cells that have the same orientation. The range of these connections is larger in V2 than in V1. Figure 3.21(c) shows the grouping results of two stimuli. While in the

upper part of the figure two vertical bar segments are grouped together, this grouping is prevented in the lower part of the figure by a horizontal line of distractors, which are grouped horizontally. In each case, grouping creates illusory contours. These contours form the basis for surface-oriented computations, such as the filling-in of color.

The vertical interactions also mediate attentional effects. Figure 3.21(b) shows, how top-down spatial attention and bottom-up visual stimuli are integrated in layer 2/3 of V1. In this layer, the isolated collinear line-segments are grouped together. Attention flows along this illusory contour and biases the entire object.

Many perceptual effects have been modeled with variants of this architecture. The model is biologically plausible and it suggests micro-modules that could be repeated to model higher visual areas. On the other hand, the model's architecture is rather complex and it remains open, how the system performs when confronted with natural visual stimuli.

3.3 Conclusions

A review of related work can never be comprehensive. Many approaches to image interpretation exist in the literature that were not covered because the focus of this chapter was to make the reader familiar to the concepts of hierarchy and recurrence, which are central to the thesis.

While many models describe isolated aspects of human visual performance on different levels of abstraction, so far no model is available that is biologically plausible, involves horizontal and vertical recurrent interactions, and can be adapted efficiently to perform different visual tasks.

Thus, there is clearly a need for further research. Neurobiology needs to find out details of the neural circuitry that leads to the impressive performance of the human visual system. Computational neuroscience must produce generic models that capture the essential mechanisms without unnecessary detail. Psychophysics can investigate properties of the visual system predicted by these models to test them. Computer vision finally has the possibility to transfer these models to real-world applications to validate their utility.

4. Neural Abstraction Pyramid Architecture

The last two chapters reviewed what is known about object recognition in the human brain and how the concepts of hierarchy and recurrence have been applied to image processing. Now it is time to put both together.

In this chapter, an architecture for image interpretation is defined that will be used for the remainder of this thesis. I will refer to this architecture as the Neural Abstraction Pyramid. The Neural Abstraction Pyramid is a neurobiologically inspired hierarchical neural network with local recurrent connectivity. Images are represented at multiple levels of abstraction. Local connections form horizontal and vertical feedback loops between simple processing elements. This allows to resolve ambiguities by the flexible use of partial interpretation results as context.

4.1 Overview

Before going to the details, this section gives an overview of the proposed architecture. It covers the hierarchical network structure, the use of distributed representations, local recurrent connectivity, and the idea of iterative refinement.

4.1.1 Hierarchical Network Structure

As the name implies, the Neural Abstraction Pyramid has a hierarchical network structure. It is sketched in Figure 4.1. The network consists of several two-dimensional layers that represent images at different degrees of abstraction. Each layer is composed of multiple feature arrays that contain discrete cells, called feature cells. When going up the hierarchy, the number of feature arrays per layer increases, while the spatial resolution decreases.

Unlike most neural networks that have no spatial organization within the layers, layers in the Neural Abstraction Pyramid have a two-dimensional organization that corresponds to the 2D nature of images. This is motivated by the observation that correlations between image locations that are close together are higher than correlations between locations that are far-apart. This simple fact is prewired in the network structure in the same way as it is prewired in the retinotopic organization of cortical areas in the human visual system (compare to Chapter 2).

The hierarchical network architecture resembles the hierarchy of areas in the ventral visual pathway (see Section 2.1). The idea of a stack of 2D layers with
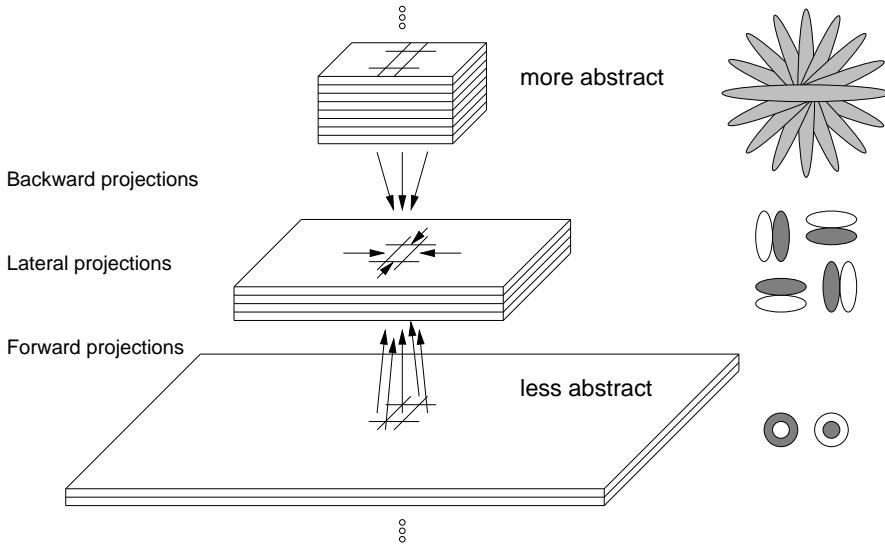
Fig. 4.1. Neural Abstraction Pyramid architecture. The network consists of several layers. Each layer is composed of multiple feature arrays that contain discrete cells. When going up the hierarchy, spatial resolution decreases while the number of features increases. The network has a local recurrent connection structure. It produces a sequence of increasingly abstract image representations. At the bottom of the pyramid, signal-like representations are present while the representations at the top are almost symbolic.

decreasing resolution has been used before, e.g. in image pyramids and wavelet representations (see Section 3.1.1). In these architectures, the number of feature arrays is constant across all layers. Hence, the representational power of the higher layers of these architectures is very limited. In the Neural Abstraction Pyramid, this effect is avoided by increasing the number of feature arrays when going up the hierarchy.

In most example networks discussed in the remainder of the thesis, the number of cells per feature decreases from $I \times J$ in layer l to $I/2 \times J/2$ in layer $(l+1)$ while the number of features increases from K to $2K$. Figure 4.2 illustrates this.

The successive combination of simple features to a larger number of more complex ones would lead to an explosion of the representation size if it were not counteracted by an implosion of spatial resolution. This principle is applied also by the visual system, as is evident from the increasing size of receptive fields when going along the ventral pathway. The dyadic reduction of resolution is natural to an implementation with binary computers. It allows one to map addresses into adjacent layers by simple shift operations. Of course, the concept can be generalized to any pyramidal structure.

It may be desirable that the number of feature cells per layer stays constant, as in the fast Fourier transformation described in Section 3.1.1. This is only possible with reasonable computational costs, if the number of accesses to the activities of other feature cells when computing a feature cell is kept below a constant. If access

$$2I \times 2J \times K/2 \qquad\qquad I \times J \times K \qquad\qquad I/2 \times J/2 \times 2K$$
layer $(l-1)$ layer l layer $(l+1)$
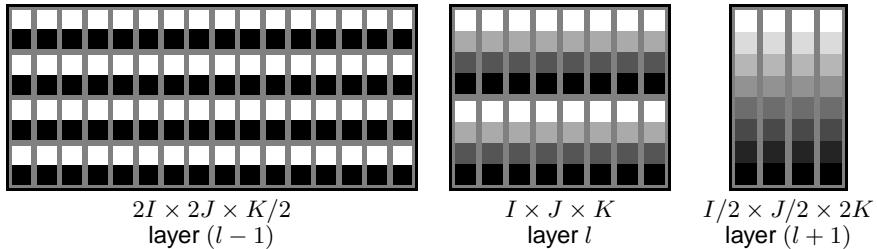
Fig. 4.2. As the spatial resolution decreases, when going up the hierarchy, the number of feature arrays increases. Different grayscales represent different features. In the lower layers few grayscales are available at many locations while in the higher layers many variations of shading can be accessed at few positions.

to all feature arrays of a layer is required to compute a feature cell, the total number of connections rises by a factor of four when increasing the number of features to 4K, instead of 2K, in layer $(l+1)$. The choice of 2K features leads to a constant number of total connections within each layer and to the reduction by a factor of two in representation size when going up one layer. Hence, in this case the size of the entire pyramid is less than double the size of its lowest layer and the total number of connections is linear in the number of layers and in the number of bottom-layer cells. Since the size of the representation decreases with height, not all details of an image can be represented in the higher layers. Thus, there is some incentive in discovering image structure that can be used to efficiently encode the abstract representations.

4.1.2 Distributed Representations

The feature cells in the Neural Abstraction Pyramid contain simple processing elements that make a single value, the activity, available to other cells. The activity of a cell represents the strength of the associated feature at a certain position. This resembles the computation by neurons in the brain. Such a massively parallel approach to image processing requires millions of processing elements for high-resolution images. Thus, it is necessary to make the individual processors simple to keep the costs of simulating these on a serial machine or implementing them in VLSI hardware within reasonable bounds. No complex data structures are used in the architecture. The communication between processing elements only requires access to cell activities via weighted links.

Figure 4.3 magnifies one layer l of the pyramid. All feature cells that share the same location (i, j) within a layer form a hypercolumn. A hypercolumn describes all aspects of the corresponding image window in a distributed sparse representation. Neighboring hypercolumns define a hyper-neighborhood.

This definition is motivated by the interwoven feature maps present in cortical areas, like in the primary visual cortex (compare to Section 2.2). All feature cells that describe the same image location at the same level of abstraction are accessible within close proximity. This facilitates interaction between these cells. Such inter-
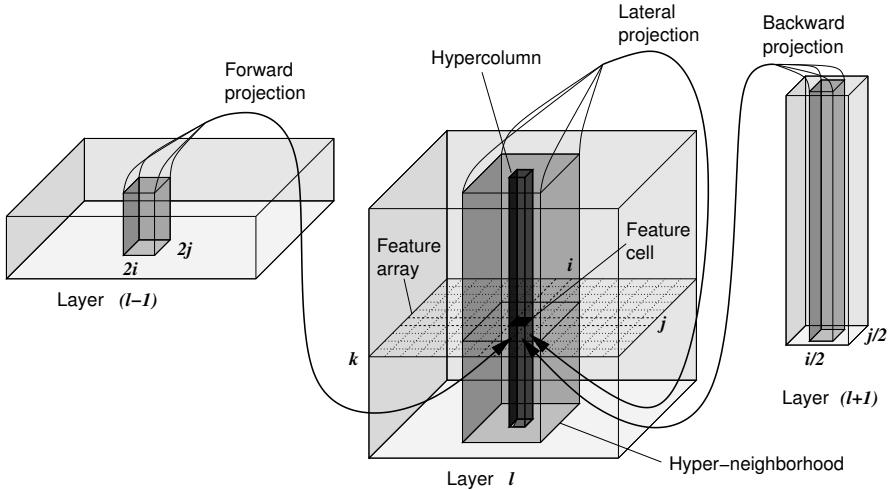
Fig. 4.3. A feature cell with its projections. Such a cell is addressed by its layer l, its feature array number k, and its array position (i, j). Lateral projections originate from the hyper-neighborhood in the same layer. Forward projections come from the hyper-neighborhood of the corresponding position $(2i, 2j)$ in the next lower layer $(l-1)$. Backward projections start from the hyper-neighborhood at position $(i/2, j/2)$ in layer $(l+1)$.

action is not only necessary between neighboring cells within a feature array k, but also across arrays since the code used is a distributed one.

The use of distributed codes is much more efficient than the use of localized encodings. A binary local 1-out-of-N code can provide at most $\log N$ bits of information while in a dense codeword of the same length, N bits can be stored. The use of sparse codes lowers the storage capacity of a code, but it facilitates decoding and associative completion of patterns [172]. Sparse codes are also energetically efficient since most spikes are devoted to the most active feature-detecting cells.

One important idea of the Neural Abstraction Pyramid architecture is that each layer maintains a complete image representation in an array of hypercolumns. The degree of abstraction of these representations increases as one ascends in the hierarchy. At the bottom of the pyramid, features correspond to local measurements of a signal, the image intensity. Subsymbolic representations, like the responses of edge detectors or the activities of complex feature cells are present in the middle layers of the network. When moving upwards, the feature cells respond to image windows of increasing size, represent features of increasing complexity, and are increasingly invariant to image deformations. At the top of the pyramid, the images are described in terms of very complex features that respond invariantly to large image parts. These representations are almost symbolic, but they are still encoded in a distributed manner.

This sequence of more and more abstract representations resembles the abstraction hierarchy found along the ventral visual pathway. Every step changes the nature of the representation only slightly, but all steps follow the same direction. They

move away from localized measurements of a signal towards estimates of the presence of complex features. One can compare this to the fast Fourier transformation that modifies a spatially localized representation step-by-step into a representation that is localized in frequency.

If the Neural Abstraction Pyramid is used for classification, an output layer may be added to the network that contains a single cell for each class to represent the classification result in localized code. This final step of the transformation makes the class-information explicit. The localized code facilitates access to the classification result because it is easier to interpret than a distributed code, but it is not biologically compatible. Note that such an output layer would not be necessary if the pyramidal perception network would be followed by an action network with the shape of an inverted pyramid that expands the abstract representations step-by-step to concrete motor commands.

4.1.3 Local Recurrent Connectivity

Retinotopic projections mediate between the layers of the Neural Abstraction Pyramid. Three types of projections are used in the network to compute a cell at position (i, j) in layer l:

– **Forward** projections originate in the adjacent lower layer $(l - 1)$. These projections have access to all features of the hyper-neighborhood centered at the corresponding position $(2i, 2j)$ and are used for feature extraction.
– **Lateral** projections stay within a layer. They access all features at positions close to (i, j) and make feature cell activities within a hyper-neighborhood consistent to each other.
– **Backward** projections come from the hyper-neighborhood centered at position $(i/2, j/2)$ of the next higher layer $(l + 1)$. They expand abstract features to less abstract ones.

This local recurrent connection structure resembles the horizontal and vertical feedback loops found in the cortex. The restriction to a local connectivity is necessary to keep computational costs down [140]. Compared to a quadratic number of possible connections, a local connection structure is much less expensive since it is linear in the number of cells. In the hierarchical network this advantage is most obvious in the lower layers, where the hyper-neighborhood of a cell contains only a small fraction of all cells of a layer. Towards the top of the pyramid this advantage is less striking, since the ratio between the number of cells in a hyper-neighborhood and the total number of cells in a layer approaches one.

There are more advantages of a local connection structure than the low number of connections alone. Local connections require only short wires when implemented in hardware. They also facilitate the distribution of labor between parallel machines with distributed memory. Even when simulating such networks on serial machines, locality of memory access patterns is an advantage, since it increases the probability of cache hits.

A local connection structure is sufficient for image interpretation tasks. Feature cells at corresponding positions in adjacent layers communicate via reciprocal forward and backward projections. The vertical connections mediate between the layers and capture the correlations between complex features and their corresponding lower-complexity subfeatures. Because the image window that is covered by a hyper-neighborhood increases with height in the pyramid, lateral interaction between distant image parts is possible in the higher layers of the pyramid. While lateral projections in the lower layers of the network capture correlations of nearby low-level features, lateral connections in higher layers capture correlations of far-apart abstract image features.

If correlations between far-apart low-level features are important, they must be mediated through a hierarchy of abstract features for the intermediate positions. This is efficient since it involves only $\Theta(\log D)$ steps if the cell-distance between the low-level features is D.

Such detailed long-distance correlations are frequently not important. This is indicated by the fact that the human visual system is often unable to detect long-distance correlations of low-level stimuli. One example for this is how difficult it is to detect a marginal difference between two similar images that are presented side-by-side. In contrast, when both images are overlaid, the differences are very salient, since the corresponding low-level features are now close together.

4.1.4 Iterative Refinement

The Neural Abstraction Pyramid has been designed for the iterative interpretation of images. The refinement of initial image interpretations through local recurrent vertical and horizontal interactions of simple processing elements in a hierarchy is the central idea of the architecture.

Such a refinement is needed to resolve ambiguities. In natural images, local ambiguities are common. For example, the contrast between an object's surface and the background may be very low at parts of the object's boundary. Occlusions may hide other object parts. Non-homogeneous lighting and object transformations, like scaling and rotation, are further sources of ambiguity. To recover the 3D structure of objects from 2D images is an inherently ambiguous problem.

The human visual system resolves such ambiguities fast and reliably. It does so by focusing on those features which are most reliable in a certain situation and by the flexible use of context information. This is exactly what the iterative image interpretation does. The interpretation of ambiguous stimuli is postponed until reliably detected features are available as context. Horizontal and vertical feedback loops allow contextual influences between neighboring image locations and between representations in adjacent layers, respectively. Information flow is asymmetric: reliable features bias the unreliable ones. This can happen in any direction. Lateral neighbors have the same reliability a-priory. Only the current stimulus decides which locations cannot be interpreted without contextual bias. The bottom-up flow of information is most common, since the function of the ventral visual pathway is to recognize

objects from images. However, one must not overlook the top-down direction of information flow. It serves attentional purposes and the match of higher-level object models with detected abstract features biases the corresponding low-level feature detectors. This is evident from edge-detecting neurons in the primary visual cortex that respond to illusory contours [185].

Iterative image interpretation has the features of an anytime algorithm. Usable partial image interpretations are available very early at the top of the hierarchy, starting when the first feed-forward wave of activity arrives. This initial response may even be perfect, e.g. if the image does not contain ambiguities. The rapid information processing in the human visual system within the first 150ms after stimulus onset [226] may correspond to this mode of operation. This initial feed-forward interpretation may trigger a behavioral response to ambiguous stimuli in situations where reaction time is precious, e.g. when spotting a dangerous animal. However, this rapid information processing bears the risk of misinterpreting visual stimuli since there is no time to resolve ambiguities. The reader may remember situations where an object that appeared to be harmful at first sight triggered a reaction, but turned out to be harmless when inspected more closely.

Because in most situations the correct interpretation of stimuli is more important than pure reaction time, the iterative refinement of initial interpretations is an effective way to improve the interpretation quality. It increases the chances for correctly interpretation of ambiguous stimuli tremendously at moderate costs of additional computation time for these stimuli.

4.2 Formal Description

The general concept of the Neural Abstraction Pyramid was introduced above. Let us now turn to a more formal description of the architecture.

4.2.1 Simple Processing Elements

The computation of a feature cell at position (i, j) for feature array k in layer l is illustrated in Figure 4.4. The basic processing element consists of P_{kl} projection units and a single output unit. The activity $a_{ijkl}^t \in \mathbb{R}$ of the cell at time t is computed as follows:

$$a_{ijkl}^t = \psi_{kl}(\sum_{p=1}^{P_{kl}} v_{kl}^p \, b_{ijkl}^{tp} \, + \, v_{kl}^0). \tag{4.1}$$

The output unit computes a weighted sum of the projection potentials $b_{ijkl}^{tp} \in \mathbb{R}$ with the weighting factors described by $v_{kl}^p \in \mathbb{R}$. A bias value of v_{kl}^0 is also added to the sum before it is passed through the output transfer function ψ_{kl}.

The computation of the individual projection potentials is described by:

$$b_{ijkl}^{tp} = \phi_{kl}^p(\sum_{q=1}^{Q_{kl}^p} w_{kl}^{pq} \, a_{i^* j^* k^* l^*}^{t^*} \, + \, w_{kl}^{p0}). \tag{4.2}$$
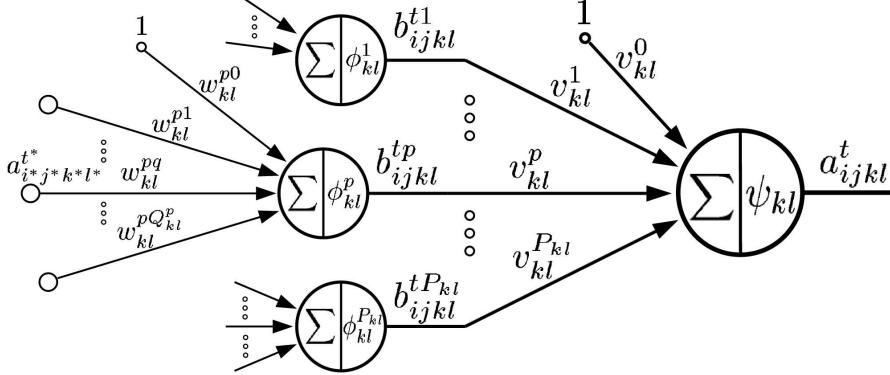
Fig. 4.4. Processing element that computes a feature cell. It consists of P_{kl} projection units and one output unit that produces the activity a^t_{ijkl}. The output unit computes the weighted sum of the potentials b^{tp}_{ijkl} of the individual projections and passes this sum through a transfer function ψ_{kl}. Each projection unit computes the weighted sum of feature-cell activities and passes it through a transfer function ϕ^p_{kl}. Bias weights of the projection units and the output unit connect to a node with the fixed activity one.

Each projection computes a weighted sum of activities $a^{t^*}_{i^*j^*k^*l^*}$ with the weighting factors described by $w^{pq}_{kl} \in \mathbb{R}$. The number of contributions to a projection p is Q^p_{kl}. In addition, a bias value of w^{p0}_{kl} is added before the sum is passed through the projection transfer function ϕ^p_{kl}.

The address $i^*j^*k^*l^*t^*$ of a source feature cell is described by:

$$t^* = T^p_{kl}(t); \tag{4.3}$$

$$l^* = L^p_{kl}; \tag{4.4}$$

$$k^* = K^{pq}_{kl}; \tag{4.5}$$

$$j^* = J^{pq}_{kl}(j) = \Upsilon_{ll^*}(j) + \mathsf{J}^{pq}_{kl}; \tag{4.6}$$

$$i^* = I^{pq}_{kl}(i) = \Upsilon_{ll^*}(i) + \mathsf{I}^{pq}_{kl}. \tag{4.7}$$

$T^p_{kl}(t)$ determines if the source activity is accessed in a direct or in a buffered mode. L^p_{kl} describes the layer of the source. K^{pq}_{kl} addresses the feature array within layer l^*. $J^{pq}_{kl}(j)$ and $I^{pq}_{kl}(i)$ describe the source location within the array k^* as a function of the destination location (i, j). The source is accessed relative to the corresponding position $(\Upsilon_{ll^*}(i), \Upsilon_{ll^*}(j))$, where $\Upsilon_{ll^*}(x)$ maps coordinates from layer l to layer l^* and $(\mathsf{I}^{pq}_{kl}, \mathsf{J}^{pq}_{kl})$ describes the source offset relative to the corresponding position. Details of the addressing will be discussed later.

The choice of the basic processing element as feed-forward neural network with a hidden layer of projection units and a single output unit is motivated as follows. It is simple enough to be computed quickly and it is powerful enough to compute the activity of a feature cell as a non-linear function of feature-cell activities.

Many aspects of biological neurons are not modeled by the basic processing element. For instance, the discrete-time computation of activity is only a coarse

Table 4.1. Notation used for the basic processing element shown in Figure 4.4.

a_{ijkl}^{t}	–	activity after iteration t of feature k in layer l at position (j, j)
b_{ijkl}^{tp}	–	potential of projection p that contributes to a_{ijkl}^{t}
ψ_{kl}	–	transfer function for feature k in layer l
ϕ_{kl}^{p}	–	transfer function for projection p of feature k in layer l
v_{kl}^{p}	–	weight of pth potential b_{ijkl}^{tp} that contributes to activity a_{ijkl}^{t}
w_{kl}^{pq}	–	weight of qth activity $a_{i^*j^*k^*l^*}^{t^*}$ that contributes to potential b_{ijkl}^{tp}
P_{kl}	–	number of potentials that contribute to activity a_{ijkl}^{t}
$T_{kl}^{p}(t)$	–	source access mode of projection klp: direct of buffered
L_{kl}^{p}	–	source layer of projection klp
K_{kl}^{pq}	–	source feature index of weight $klpq$
$J_{kl}^{pq}(j)$	–	source row of weight $klpq$, depends on destination row j
$I_{kl}^{pq}(i)$	–	source column of weight $klpq$, depends on destination column i
$\Upsilon_{ll^*}(x)$	–	mapping of coordinates from layer l to layer l^*
J_{kl}^{pq}	–	row offset of weight $klpq$
I_{kl}^{pq}	–	column offset of weight $klpq$

approximation of the continuous-time dynamics of neurons. Another example is the modeling of synapses by a single multiplicative weight, as compared to a dynamical system that describes facilitation and depression of the synaptic efficacy. Furthermore, in contrast to cortical neurons that produce action potentials, the basic processing element outputs a graded response. These simplifications were made to make network simulations feasible, despite the fact that typical Neural Abstraction Pyramid networks will contain thousands to millions of such processing elements. More complex processing elements might be more powerful, but they induce higher computational costs and need more parameters for their description.

The basic processing element is already quite powerful. This can be seen from the success of feed-forward neural networks with a single hidden layer. When equipped with a large-enough number of projection units that have sigmoidal transfer functions it can approximate, with arbitrary accuracy, any continuous function of its inputs [48]. However, in typical networks the number of projection units will be small and the computed functions will be simple. The weights that determine the behavior of the network can be changed through learning. If the transfer functions ψ_{kl} and ϕ_{kl}^{p} are differentiable, partial derivatives of the output error with respect to a weight can be computed by backpropagation and gradient descent techniques can be applied to adjust the weights.

4.2.2 Shared Weights

Where do the inputs $a_{i^*j^*k^*l^*}^{t^*}$ to a projection p of a feature cell $ijkl$ come from? All weights of a projection originate in the same layer. This source layer is called

l^* and it is determined by the index L_{kl}^p that depends on the feature array kl and the projection index p, but not on the cell position (i, j) within its feature array. Since the connections have to be local, they originate either in same layer $L_{kl}^p = l$ for lateral projections or in an adjacent layer $L_{kl}^p = l \pm 1$ for forward/backward projections.

The feature index k^* of the feature cell accessed by a weight q of projection p is determined by the index $K_{kl}^{pq} \in \{0, \ldots, K_{l^*} - 1\}$, where K_{l^*} is the number of feature arrays in the source layer l^*. Hence, access to any feature is allowed.

The position of the accessed feature cell depends on the position (i, j) of the computed cell. A function $\Upsilon_{ll^*}(x)$ maps positions from layer l to layer l^*. If the resolution of the two layers differs by a factor of two it is computed as follows:

$$\Upsilon_{ll^*}(x) = \begin{cases} 2x & : & l^* = l - 1 & [\text{forward}] \\ x & : & l^* = l & [\text{lateral}] \\ \lfloor x/2 \rfloor & : & l^* = l + 1 & [\text{backward}] \end{cases} . \tag{4.8}$$

In case that the source layer l^* consists of only a single hypercolumn at position $(0, 0)$, all positions in l are mapped to this hypercolumn: $\Upsilon_{ll^*}(x) = 0$. The hypercolumn of the accessed feature depends also on the weight q. An offset $(\mathrm{I}_{kl}^{pq}, \mathrm{J}_{kl}^{pq})$ is added to the corresponding position $(\Upsilon_{ll^*}(i), \Upsilon_{ll^*}(j))$. These offsets are usually small and access only a $M \times N$ hyper-neighborhood of $(\Upsilon_{ll^*}(i), \Upsilon_{ll^*}(j))$. For forward projections, the offsets are usually chosen such that M and N are even since the offsets (0,0), (0,1), (1,0), and (1,1) describe the source-hypercolumns that correspond to a higher-level hypercolumn when the resolution is changed by a factor of two between the layers. Common are 4×4 forward projections that overlap with eight neighboring projections. For lateral projections, odd dimensions of the neighborhood are used to produce a symmetric connection structure. A 3×3 lateral neighborhood is common.

All feature cells of a feature array kl share the same forward and lateral projections. This weight sharing is motivated by the success of convolutional neural networks (see Section 3.1.2). While it is not biologically plausible that distant weights are forced to have the same value, it is likely that similar stimuli that occur at different positions of a feature map lead to the development of similar feature detectors. This translational invariance of feature detection that must be learned by cortical feature maps is prewired in the Neural Abstraction Pyramid architecture. It is appropriate for the processing of images since low-level stimuli, like edges and lines, are likely to occur at several positions in the image.

Weight sharing leads to descriptions of the processing elements with templates that have few parameters. It also allows for a sharing of examples since a single image contains multiple small windows with different instances of low-level features. Both few parameters and example sharing facilitate generalization. The degree of weight sharing is high in the lower layers of the network and decreases towards the top of the pyramid. Low-complexity features are described by few weights, while many parameters are needed to describe the extraction of complex features. Hence, the network is able to learn low-level representations from relatively few examples,

but it needs a larger number of examples to determine the parameters of more abstract representations.

In the dyadic case, where the resolution between the layers is reduced by a factor of two in both dimensions, four hypercolumns correspond to a single hypercolumn in the next higher layer. To make specific backward projections possible, each of the four lower-level hypercolumns has to maintain its own backward projection. Thus, in this case the backward weights are shared with only one fourth of the feature cells. This connection structure can be viewed as distributed storage of a single larger projection that is computed in the reverse direction. For instance, when a reversed backward projection has a size of 2×2, it covers all lower-level hypercolumns without overlap. Such a projection is realized as four different 1×1 backward projections with offsets (0,0) that are distributed between the 2×2 corresponding lower-level hypercolumns.

A special case for the forward/backward projections is the drop of resolution to 1×1 hypercolumns at the top of the pyramid. Here, the offsets are usually chosen in such a way that a complete connection structure between the topmost layer and the layer below it is achieved. No weight sharing is possible in this case.

4.2.3 Discrete-Time Computation

The activities of the feature cells are computed at discrete points t of time. They are accessed either directly or as a buffered copy. The cell activities must be initialized and can be clamped to input values. Care must be taken to handle border effects.

Update Order. All feature cells are computed in a predetermined order. Usually, the update of the activities proceeds layer by layer in a bottom-up manner. This is done to speed up the forward flow of information. Within the layers, the features are sometimes assigned to groups. For instance, excitatory and inhibitory features can constitute two different groups. The fixed update order can assure that all features of one group are updated before the first feature of another group is updated. This makes fast lateral interactions, like fast inhibition, possible.

Direct Access. All weights of a projection access activities from the same point of time t^*, described by the function $T_{kl}^p(t)$. For direct access, activities that have already been computed in the same time step are used: $T_{kl}^p(t) = t$. This is possible only if the earlier update of the sources can be ensured. The direct access is commonly used for forward projections and for fast lateral projections, like the ones from excitatory to inhibitory features. This fast inhibition prevents a delay between monosynaptic excitation and disynaptic inhibition. Fast inhibition is biologically plausible since inhibitory synapses typically contact neurons mostly near to the soma or even at the soma, while excitatory synapses typically connect to more distant parts of the dendritic tree which induces some delay.

Buffered Access. Of course, not all projections can be realized with direct access. It is also possible to access the activity of a feature cell from the previous time step: $T_{kl}^p(t) = t - 1$. This buffered access is used for backward projections and for

delayed lateral connections, e.g. from inhibitory features to excitatory ones. It can also be used for forward projections, e.g. to compare activities of two consecutive time instances in order to detect activity change.

Initialization. At the beginning of the computations when $t = 0$, it is not possible to access the activities of the previous time step. For this reason, the activities must be initialized before the iterative update starts. The initialization of a feature-array k in layer l is done uniformly. All its feature cells are set to the activity a_{kl}^0. Thus, the uniform initialization adds only a single parameter to the template that describes the computation of feature kl.

Network Input. To present input to the network some of its cells are not computed by basic processing elements, but are clamped to static or dynamic inputs. In this case, predetermined components of the input vector are copied to the cell outputs. In general, different feature cells of an input array will receive different activities. The input cells are accessed in the same way as all other cells.

Due to the recurrent network connectivity, inputs can occur at any layer. Signal-like inputs, such as images, are presented at the lower layers of the pyramid, while higher-level feature cells can be clamped to abstract features, such as class indicators.

Network Output. Analogously, any feature cell can serve as a network output. Outputs are not treated differently during the update of network activities. In particular, their activity is fed back into the network and influences other feature cells and hence the output activities at later instances of time. Output cells play a special role for supervised learning when predetermined components of a target vector are compared to the activity of output units.

Feature cells which are neither input nor output play the role of hidden processing elements. They maintain intermediate representations and mediate between inputs and outputs.

Border Handling. The computation of the source hypercolumn's address may yield positions that are outside the feature arrays. To ease the handling of such border effects, the arrays are framed with a border, such that all weights have a source that is either a valid feature cell or part of the frame. The activity of frame cells is determined after all feature cells of its feature array have been computed. Different update modes are implemented. The easiest mode is to set the frame to a constant value, e.g. to zero. In this case, it must be ensured that no discontinuity is created between the feature cells and the frame cells. Another common update mode is to set the frame cell activities to copies of feature cell activities. For instance, the feature cells can be copied such that cyclic wrap-around border conditions are achieved. In this case, it must be ensured that no discontinuities occur between the opposite sides of the feature array. Other less common possibilities of border updates are the fade-out of activity with increasing distance from the feature cells or the copying with reflection at the border. The frame cells are accessed in the same way as all other cells.

4.2.4 Various Transfer Functions

Both the projection units and the output units of the basic processing element are equipped with transfer functions ϕ_{kl}^{p} and ψ_{kl}, respectively. These functions determine the interval of possible projection potentials and cell activities. They are also the only source of nonlinearity in the Neural Abstraction Pyramid.

The simplest transfer function is the identity: $f_{\mathsf{id}}(x) = x$. It is used if no nonlinearity and no scaling is desired. Since it does not limit its output values, the identity transfer function is frequently used for the projection units only. The choice of $\phi_{kl}^{p} = f_{\mathsf{id}}$ reduces the basic processing element to a single $\sum$-unit. In this case, the weights of the individual projections are treated as if they would contribute directly to the weighted sum of the output unit.

Common choices for the output transfer function ψ_{kl} are functions that limit the activities to a finite interval. For instance, the functions

$$f_{\mathsf{sat}}(x) = \begin{cases} 0 & : & x \leq -\alpha \\ \frac{1}{2} + \frac{x}{2\alpha} & : & -\alpha < x < \alpha \\ 1 & : & x \geq \alpha \end{cases} \quad \text{and} \quad f_{\mathsf{sig}}(x) = \frac{1}{1 + e^{-\beta x}}$$

limit the outputs to the interval $[0, 1]$ and the functions

$$f_{\mathsf{pn\_sat}}(x) = \begin{cases} -1 & : & x \leq -\alpha \\ \frac{x}{\alpha} & : & -\alpha < x < \alpha \\ 1 & : & x \geq \alpha \end{cases} \quad \text{and} \quad f_{\mathsf{pn\_sig}}(x) = \frac{2}{1 + e^{-\beta x}} - 1$$

limit the outputs to the range $[-1, 1]$. The graphs of these functions are drawn in Figure 4.5. While the piecewise-linear saturation functions f_{sat} and $f_{\mathsf{pn\_sat}}$ have a derivative of zero outside the interval $[-\alpha, \alpha]$, the sigmoidal functions f_{sig} and $f_{\mathsf{pn\_sig}}$ have a derivative that is nonzero everywhere. This property is important when an error-signal must be backpropagated.

The use of such nonlinear functions is crucial for the stability of the network dynamics. When the activity of a unit is driven towards saturation, the effective gain of the transfer function is reduced considerably. This avoids the explosion of activity in the network.
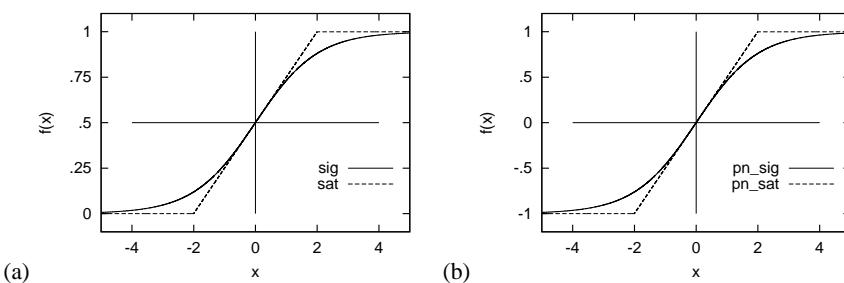
Fig. 4.5. Saturating transfer functions: (a) limiting the activities to the interval $[0, 1]$ (f_{sat}: $\alpha = 2$; f_{sig}: $\beta = 1$); (b) limiting the activities to $[-1, 1]$ ($f_{\mathsf{pn\_sat}}$: $\alpha = 2$; $f_{\mathsf{pn\_sig}}$: $\beta = 1$).
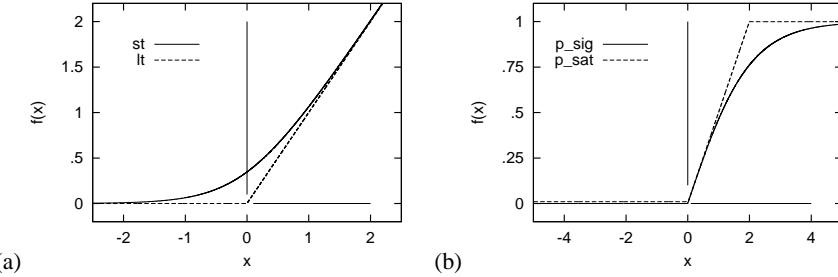
Fig. 4.6. Rectifying transfer functions: (a) without saturation (f_{lt}: $\alpha = 1$; f_{st}: $\beta = 2$); (b) with saturation ($f_{\text{p\_sat}}$: $\alpha = 2$; $f_{\text{p\_sig}}$: $\beta = 1$).

Furthermore, nonlinear units are needed to make decisions. Because the linear combination of linear units yields a linear function of the inputs, decisions cannot be made with linear units alone.

Another important class of transfer functions are the rectifying functions, shown in Figure 4.6(a):

$$f_{\text{lt}}(x) = \left\{ \begin{array}{ccc} 0 & : & x \le 0 \\ \alpha x & : & x > 0 \end{array} \right. \quad \text{and} \quad f_{\text{st}}(x) = \frac{\log(1 + e^{\beta x})}{\beta}.$$

The linear threshold function f_{lt} has derivative zero for negative arguments and a constant derivative α for positive arguments. A smooth approximation to this function is f_{st}. Its derivative is nonzero everywhere. Such rectifying functions are used in models that resemble the nonnegative activity of spiking neurons. These models assume that the growth of activity is limited by strong recurrent inhibition and that the functionally important nonlinearity of biological neurons is not the saturation of the firing rate for high input currents, but the muting of spiking neurons when the net input is inhibitory.

Another possibility to limit the growth of activity is to use saturation for high input values. The saturating rectifying functions

$$f_{\text{p\_sat}}(x) = \max(0, f_{\text{pn\_sat}}) \quad \text{and} \quad f_{\text{p\_sig}}(x) = \max(0, f_{\text{pn\_sig}})$$

are shown in Figure 4.6(b). They limit their output values to the interval $[0, 1]$.

If not only half-rectification is desired, but the full energy of a signal is required, a combination of square transfer function $f_{\text{sqr}}(x) = x^2$ for the projection units and square root $f_{\text{sqrt}}(x) = \sqrt{x}$ for the output unit can be used. These transfer functions are illustrated in Figure 4.7(a). They are applied in some models of complex cells, where two orthogonal orientation-sensitive projections are combined to a phase-invariant orientation-sensitive response.

Another special-purpose pair of transfer functions, shown in Figure 4.7(b), is the logarithmic function $f_{\text{log}}(x) = \log(x)$, used in the projection units, followed by an exponential $f_{\text{exp}}(x) = e^x$ in the output unit. Here, it must be ensured that the argument of the logarithm is positive. These transfer functions lead to the conversion of the output unit into a $\prod$-unit that multiplies powers of the projection sums:
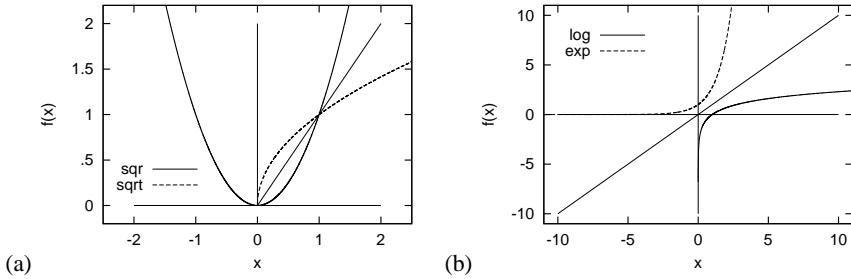
Fig. 4.7. Other transfer functions: (a) square and square root; (b) logarithm and exponential.

$$a^t_{ijkl} = e^{v^0_{kl}} \cdot \prod_{p=1}^{P_{kl}} (\sum_{q=1}^{Q^p_{kl}} w^{pq}_{kl} \, a^{t^*}_{i^*j^*k^*l^*} + w^{p0}_{kl})^{v^p_{kl}}. \qquad (4.9)$$

Such $\sum \prod$-units resemble the alternating operations of the sum-product algorithm that implements Kalman filters, hidden Markov models and fast Fourier analysis in factor graphs [129]. For that reason, it is important that the basic processing element can implement products of sums.

The transfer functions discussed above are not the only possible choices. Nevertheless, they illustrate the possibilities to create representations with different properties and network dynamics with different behavior in the Neural Abstraction Pyramid framework.

4.3 Example Networks

To illustrate the possible use of the Neural Abstraction Pyramid architecture, the following section presents some small example networks that were designed manually.

4.3.1 Local Contrast Normalization

The first example focuses on horizontal and vertical interaction in a hierarchy, but does not yet increase the number of features when decreasing the resolution. It implements local contrast normalization in the Neural Abstraction Pyramid.

Contrast normalization helps to overcome the limited dynamic range of linear image sensors. Typical sensors measure the local intensity with an accuracy of 8 bits per pixel. This leads to problems if very high and very low intensities are present simultaneously in an image. Figure 4.8(a) shows such a problematic image, taken with an entry-level digital still camera. The foreground is very dark, while the background, visible through the window, is very bright. The limited dynamic range of the camera impairs the visibility of details. Global normalization of brightness does

G_4 20×15 R_4
G_3 40×30 R_3
         80×60 R_2
G_2
         160×120 R_1
G_1
         320×240
G_0 R_0
(a)   (b)
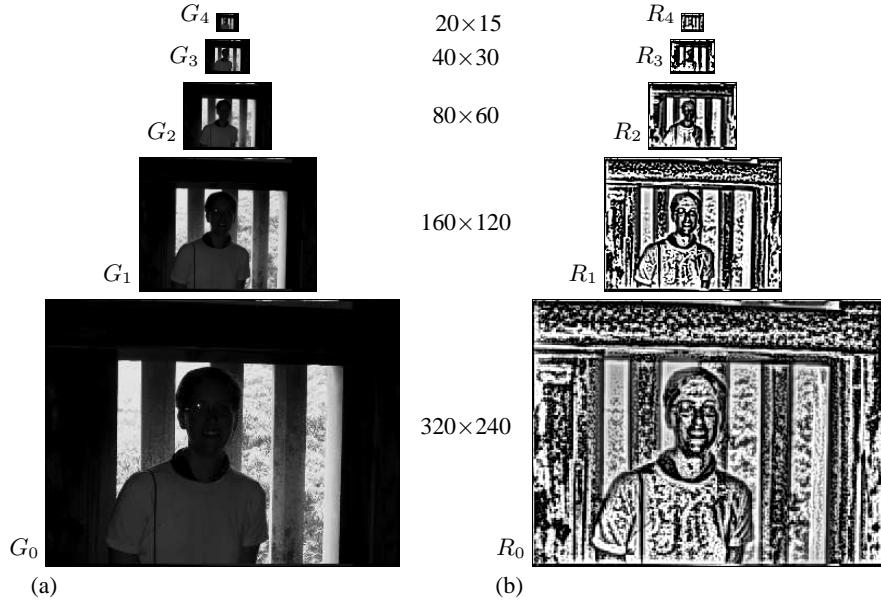
Fig. 4.8. Local contrast normalization: (a) Gaussian pyramid; (b) multiscale result after 15 iterations of a Neural Abstraction Pyramid.

not help in such a situation, since it would either discard the foreground, or the background details.

The human visual system easily handles such a difficult lighting situation. One of the reasons is the logarithmic scaling of perception described by the Weber-Fechner law [236, 64]: $R = k \ \log(I/I_0)$, where R is the perceived stimulus strength, k is a constant, I is the stimulus intensity, and I_0 is the perception threshold. Such a logarithmic transfer function results in a high dynamic range of perception. Intensities are not measured absolutely, but relative to their surround. This is evident from the just-noticeable intensity difference ΔI that is a multiple of the surround intensity: $\frac{\Delta I}{I} = k$.

The human visual system uses local normalization of contrast to achieve the desired dynamic range [35]. Contrast is related to the average local contrast. This leads to facilitation effects. In regions where contrast is low, local deviations are amplified. Another property produced by local contrast normalization is masking. The threshold for contrast detection is increased in the vicinity of high-contrast edges.

The result of such a local contrast normalization in multiple scales with interactions between the scales is shown in Figure 4.8(b). In this image, much more details are visible. The 8-bit linear dynamic range is not wasted to represent the intensity level, but it is used to represent local contrast. In the visual system, such a normalization not only increases information transfer, but also keeps the firing rates of neurons within a physiologically plausible range.
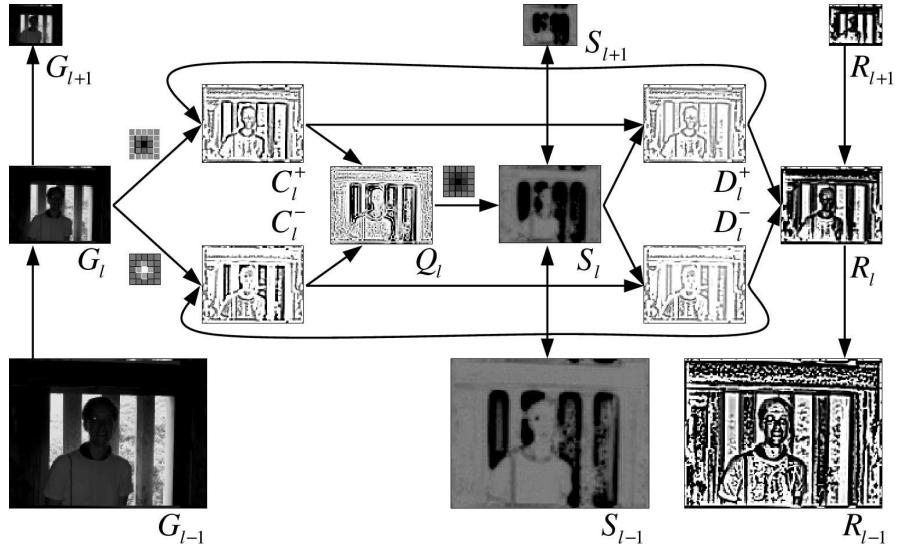
Fig. 4.9. Local contrast normalization. View of layer $l = 2$. The contrast $C_l^{\pm}$ is divided by the local contrast level S_l. See text for a more detailed explanation.

The normalized image was computed by a Neural Abstraction Pyramid that was iterated 15 times. The network consists of five layers with resolutions from 320×240 decreasing to 20×15. It computes subsampled versions G_l of the high-resolution input G_0. In each layer, local contrast is detected using 5×5 center-surround kernels and divided by the smoothed squared contrast. The normalized contrast is combined in a top-down fashion.

Figure 4.9 illustrates the iterative operation of a single network layer. In the following, the templates used for the computation of the different features are described in detail. The feature arrays are updated in the listed order. If not stated otherwise, the forward and lateral projections are direct and the backward projections are buffered. The projections have only one input from the offset $(0, 0)$ with the weight one and are summed by the output unit. The units have linear transfer functions and zero bias. The activity of the cells is initialized to zero. The intervals indicate the scaling of activities used in the figure.

- G_l – intensity $[0, 1]$

– contains shrunken versions of the original image
– has only a single forward projection that averages 2×2 windows of G_{l-1}

- $C_l^{\pm}$ – contrast $[0, 1]$

– contains local contrast, separated by sign
– lateral projection has center-surround input from G_l (DoG 5×5 - 3×3) and linear threshold transfer function f_{lt}
– buffered lateral projection receives input from corresponding cell in $D_l^{\pm}$

| 1 | 2 | 4 | 7 | 11 |

Fig. 4.10. Local contrast normalization. The high-resolution output R_0 is shown over time.

- Q_l – squared contrast $[0, 1]$

  – contains local energy
  – two lateral projections from $C_l^{\pm}$ with square transfer function f_{sqr}

- S_l – smoothed contrast $[0, 2]$

  – lateral projection computes smoothed version of Q_l using $5{\times}5$ binomial kernel
  – forward projection from S_{l-1} that averages $2{\times}2$ windows with total weight 0.5
  – backward projection from S_{l+1} that expands to $2{\times}2$ cells of S_l with weight 0.5
  – bias weight of output unit 0.1

- $D_l^{\pm}$ – normalized contrast $[0, 0.5]$

  – lateral projection from $C_l^{\pm}$ and logarithmic transfer function f_{log}
  – lateral projection from S_l and logarithmic transfer function f_{log}
  – weight from smoothed contrast projection to output sum is -1
  – output unit has exponential transfer function and computes $C_l^{\pm}/S_l$

- R_l – result $[-0.5, 0.5]$

  – lateral projection subtracts D_l^{-} from D_l^{+}
  – backward projection from R_{l+1} that expands to $2{\times}2$ cells of R_l with weight 0.5

The central operation of the network is the division $C_l^{\pm}/S_l$. It is implemented with two features, D_l^{+} and D_l^{-}, since the arguments of the logarithmic transfer function f_{log} must be nonnegative. Another property of the implementation is that the smoothed contrast level S_l is not only computed within a scale, but that contrast present at adjacent scales increases S_l. This extends the lateral competition to a competition between scales and produces masking effects in scale. Small high-contrast details can mask larger-scale contrasts and vice versa.

Figure 4.10 displays the development of the high-resolution output R_0 over time. After the first iteration, only small-scale contrast is present in the output since the backward projection is not effective yet. During the following iterations, larger-scale contributions arrive. The change of the network's activity decreases monotonically after the initial iterations. The network dynamics converges quickly towards the attractor shown in Figure 4.8(b).

4.3.2 Binarization of Handwriting

In the previous example, we have seen that local interaction in a hierarchy can implement globally interesting computations. However, the representational power of the network used for contrast normalization was limited, since the constant number of features per layer did not counteract the decrease of resolution towards the top of the pyramid.

In the second example network the number of features increases by a factor of two when going up one layer, as described in Section 4.1.1. The increasing number of features is used to build a hierarchical model of handwriting for the task of binarization. Figure 4.11 shows some examples from the dataset used for the experiments. The original images were extracted by Siemens AG from large-sized letters, called flats, for the purpose of automated mail sorting. They contain handwritten German ZIP codes on a relatively dark background.

Binarization of these images is one step towards the recognition of the ZIP codes. It assigns the pixels to one of two classes: the foreground or the background. The goal is to assign the pixels belonging to the strokes of the digits to the foreground class, and all other pixels to the background. Binarization discards variance that is not relevant for recognition, such as brightness of the lighting and the structure of the paper and keeps recognition-relevant aspects, such as the shape of the lines. This task is non-trivial due to different sources of noise and variance. For instance, the line thickness varies considerably because different pens have been used to write the digits. Next, the image contrast is sometimes low because of the darkness of the paper and the weakness of the writing device. Furthermore, the structure of the paper and background clutter are sources of noise. Finally, due to the height of the letters, some images have been captured outside of the camera's focal plane which leads to unsharp line borders.

Histogram-based thresholding techniques are among the most popular binarization methods described in the literature [122]. They assign pixels to the two classes based on the intensity alone. Pixels that are darker than a threshold are assigned to the foreground and all other pixels to the background class. If the intensity histogram of the image is bimodal, the two peaks correspond to the foreground and the background pixels. One can search for a local minimum in the smoothed histogram between the two peaks to determine a binarization threshold. Figure 4.11(b) shows thresholded versions of the original images. It can be observed that thresholding breaks weak lines into pieces and also assigns small dark clutter to the foreground.
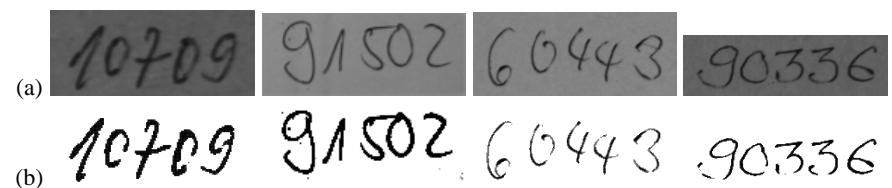
Fig. 4.11. ZIP code binarization dataset: (a) original grayscale images; (b) binarized using thresholding.
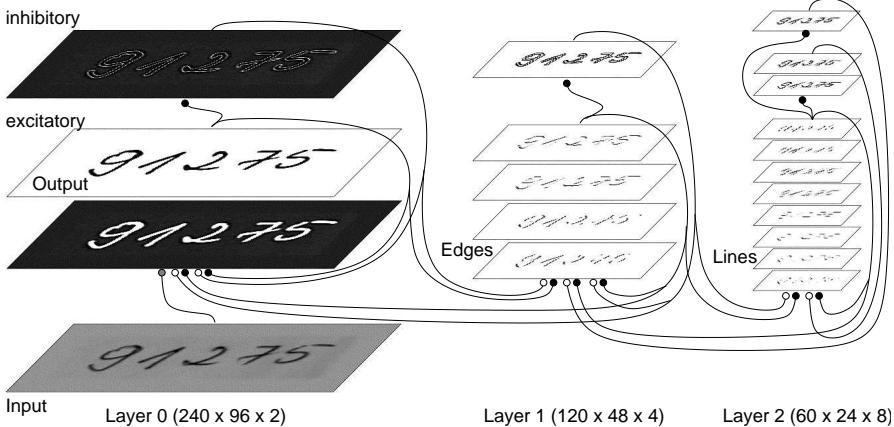
Fig. 4.12. ZIP code binarization – network architecture. The Neural Abstraction Pyramid consists of three layers. The bottom layer represents the image in terms of foreground/background features. The middle layer contains detectors for horizontal and vertical step edges. In the top layer, the lines are represented by the activities of eight orientation selective line features.

This behavior is not ideal for recognition. The structure of digits is altered considerably by broken lines and additional foreground pixels may also mislead recognition especially if they are close to the lines.

The reason for these binarization problems is the limited use of context information in the thresholding method. Only global context via the intensity histogram is used to determine the binarization threshold, but the local context of a pixel is not considered for the binarization decision. In the following, a Neural Abstraction Pyramid is described that makes this decision based on the local context. The idea motivating the network's construction is to detect the lines and use them to bias binarization. A pixel belonging to a line should be assigned to the foreground class, even if it is not much darker than its neighborhood. On the other hand, dark pixels should be assigned to the background if they are not supported by a line.

The network's architecture is sketched in Figure 4.12. It consists of three layers that represent the image at three levels of abstraction:

- **Layer 0** contains the input image, two excitatory feature arrays that represent the foreground/background assignment, and one inhibitory feature array that contains the sums of the foreground and the background features.
- **Layer 1** contains four feature arrays that represent horizontal and vertical step edges. One inhibitory feature contains the sum of the edges.
- **Layer 2** contains eight excitatory feature arrays that represent lines in different orientations. Two inhibitory feature arrays compute the sums of the more horizontal and the more vertical lines, respectively. One inhibitory feature array sums lines of all orientations.

I ← framed input with mean 0.5

F
input: ←4.031 $*I$ - 0.8
lateral: ←.387 $*F$ + .141 $*S_{FB}$
backward: ← inverse of forward proj. from Edges

B
input: ←4.031 $*I$ + 0.75
lateral: ←.413 $*B$ + .105 $*S_{FB}$
backward: ← inverse of forward proj. from Edges

S_{FB} ← $F + B$
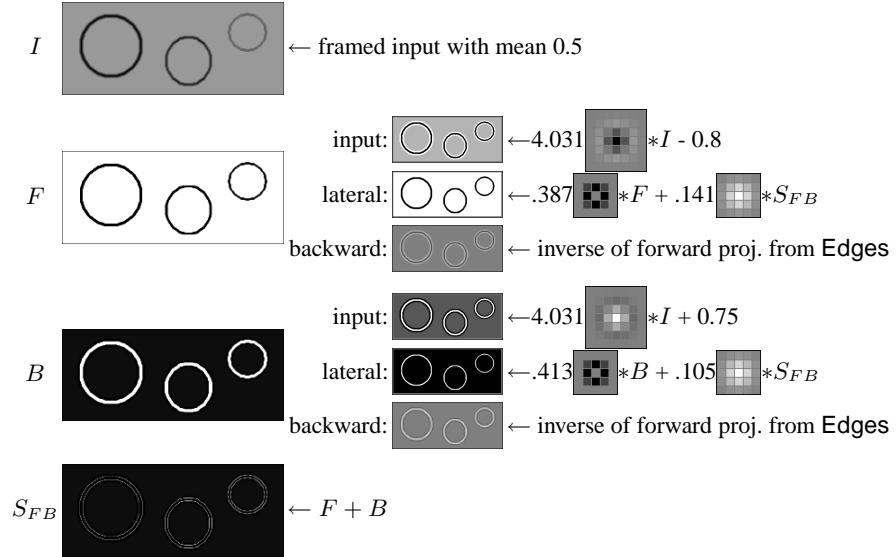
Fig. 4.13. ZIP code binarization – Layer 0 features. The image is represented in terms of foreground (F) and background (B) features. The activities of the feature arrays as well as the potentials of the contributing projections are shown. The weight-templates are scaled such that the weight with the largest magnitude is drawn in black or in white.

The resolution of the layers decreases from 240×96 to 120×48 to 60×24 hypercolumns, as the image-patch corresponding to a single hypercolumn increases from 1×1 to 2×2 to 4×4 pixels. All three layers are surrounded by a three-pixel wide border that is updated using wrap-around copying. The transfer functions of all projection units are linear, as are the transfer functions of the inhibitory output units. In contrast, the output units of the excitatory features have a transfer function $f_{\mathsf{p\_sig}}$ ($\beta = 2$, see Section 4.2.4) that is zero for negative inputs and approaches one for large inputs. Hence, inhibition grows faster than excitation if the network's activity increases. All bias values in the network are zero, and the projection weights to the output units are one if not noted otherwise. Input projections and forward projections to excitatory features as well as projections to inhibitory features are computed with direct access to avoid unnecessary delays. Lateral projections and backward projections to excitatory features need buffered access since they receive input from features that are updated later.

The separation of excitatory and inhibitory features forces the network designer to use specific excitatory and unspecific inhibitory projections. This, together with the nonnegative transfer function of the excitatory output units, makes the activity of most feature arrays sparse. The design of the network's connectivity is motivated by the Gestalt principles, discussed in Chapter 1. In particular, the principle of good continuation plays an important role for grouping aligned features to objects. In the following, the design of the individual layers is described in more detail.

Layer 0: Foreground/Background. Figure 4.13 summarizes the templates used for the processing elements of the pyramid's bottom layer features and shows the stable response of the network to a test pattern consisting of three circles.

The input array I is set to a version of the input image that has been shifted in intensity to make the mean value equal to 0.5. Furthermore, the image was reduced in size by a factor of two if it did not fit into a 232×88 window, and then smoothly framed to match the array size of 240×96.

The input projections to the forward feature F and the backward feature B have a center-surround structure. They have been set to differences between 3×3 and 7×7 binomial kernels. The central weight has an amplitude of 4.031 and the projections have a DC part of ± 1.5. This is offset by a bias of 0.75 to the background input projection. The foreground bias is set to -0.8, suppressing responses to intensities that are slightly larger than average. Hence, the forward potentials of the foreground react best to a dark center that is darker than its neighborhood (a line), and the forward potentials of the background react best to a bright center that is surrounded by dark lines (a loop center).

Lateral projections to the two excitatory features have a specific excitatory and an unspecific inhibitory part. Excitation comes from the 3×3 neighborhood of the same feature and inhibition from a 5×5 window of the sum S_{FB} of the two features. The feature cells do not excite themselves but inhibit themselves via S_{FB}. Hence, the lateral connectivity favors blob-like activities that extend over multiple neighboring pixels and suppresses isolated active cells. The lateral excitation for the background is stronger than the one for the foreground. The opposite applies to the inhibition. Thus, the lateral competition between the two features favors the background. Initial foreground responses are removed if they are not supported by neighboring foreground pixels or by edges detected from Layer 1.

Top-down support comes from the backward projections which are the inverse of excitatory forward projections to the edge-features. They expand the edge representation to the higher-resolution foreground/background representation. Unspecific backward inhibition comes from the sum of the edges S_E.

Layer 1: Edges. The middle layer of the binarization network is summarized in Figure 4.14. Four features detect step edges. E_T responds to the top edge of horizontal lines and E_B to their bottom edge. The left and right edges of vertical lines excite E_L and E_R.

The specific excitatory weights of the 6×6 forward projections resemble the oriented foreground/background double line that is characteristic for step edges in Layer 0. Unspecific forward inhibition comes from S_{FB} weighted with a 6×6 binomial kernel. The forward projections have a bias weight of -0.05 to prevent reaction to spurious edges. The sum of the edge features is computed by S_E.

Lateral projections mediate cooperation between aligned edges of same or similar orientations by 3×3 excitatory kernels and unspecific competition via a 5×5 binomial kernel, folded with S_E. Since edge cells do not excite themselves, they must be supported by other edges or line features to survive the competition.

The backward projections to the edge features expand the line features from Layer 2 using the inverse of their excitatory forward weights. Unspecific backward inhibition comes from S_L.

Layer 2: Lines. The top layer of the binarization network is illustrated in Figure 4.15. Eight excitatory features $L_0, L_1, \ldots, L_7$ detect lines of different orientations. They receive 6×6 specific excitatory input from parallel oriented step edges. Unspecific forward inhibition weights S_E with a 6×6 binomial kernel. The forward projections have a bias weight of -0.05 that prevents responses to spurious lines.

Using 3×3 specific excitatory weights to aligned lines of same or similar orientations, line cells cooperate. Competition between line features is mediated by two inhibitory features S_V and S_H. They sum the more vertical and the more horizontal lines, respectively, and inhibit them again. This construction restricts competition to lines of similar orientation. Since there is no competition between horizontal and vertical lines, crossings of two such lines can be represented without the need to suppress one of them.
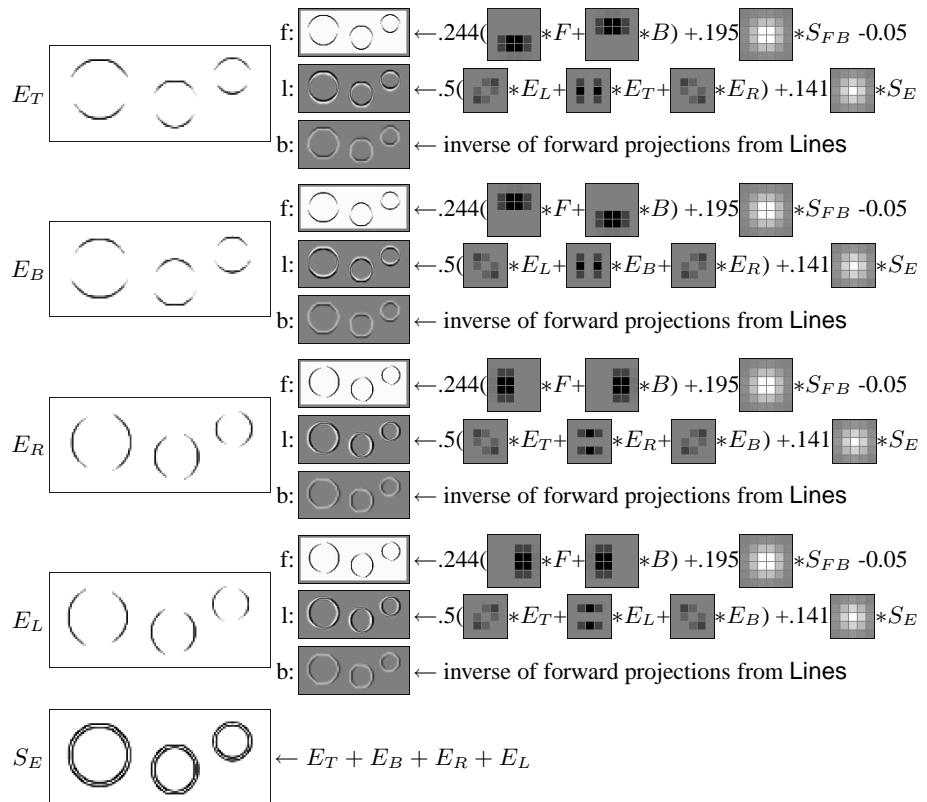
Fig. 4.14. ZIP code binarization – Layer 1 features. The image is represented in terms of horizontal and vertical step edges.
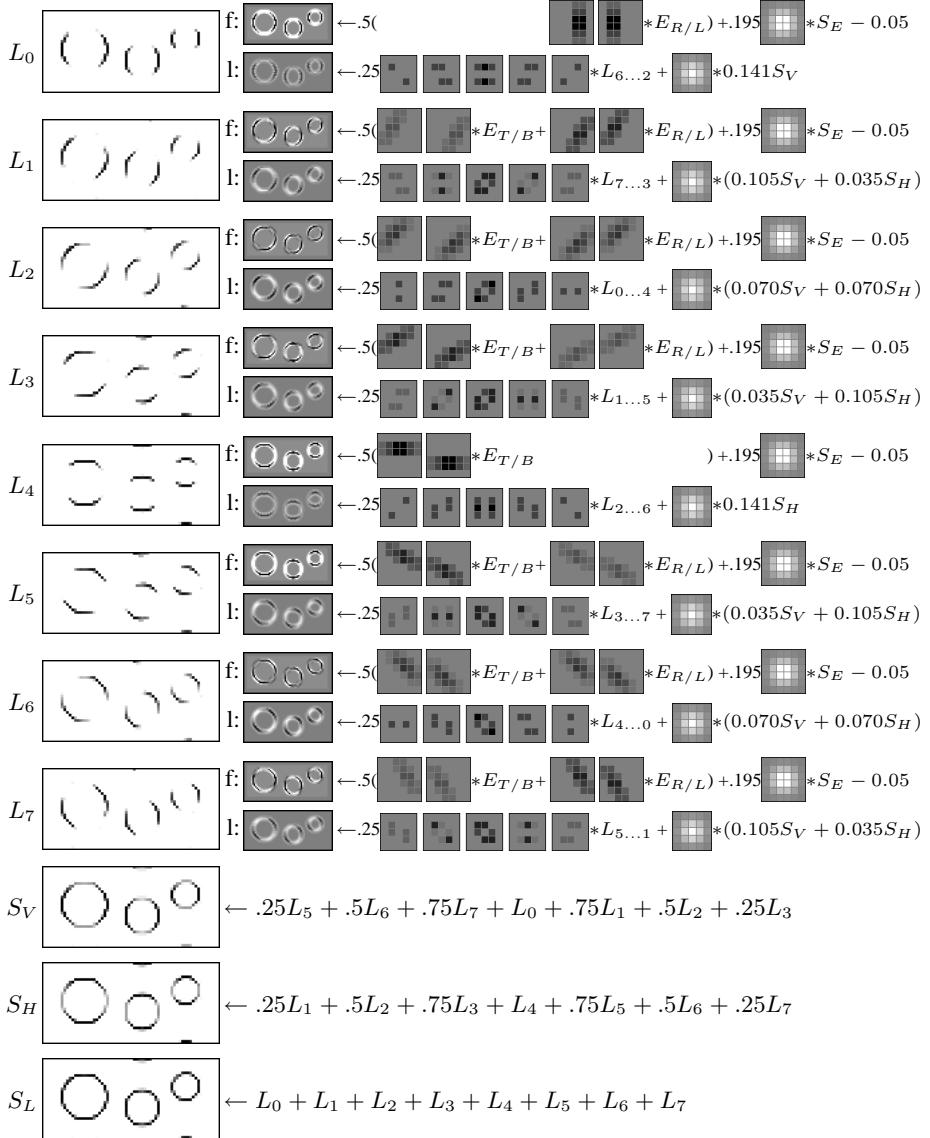
Fig. 4.15. ZIP code binarization – Layer 2 features. The image is represented in terms of oriented lines. The lateral competition via two inhibitory features allows horizontal and vertical lines to coexist at the same position. This is useful for the representation of line crossings.

Network Dynamics. In the preceding figures, the steady-state response of the binarization network to a test pattern was shown. In contrast, Figure 4.16 shows the network activity over time when an example ZIP code image is presented at the input array. One can see that the initial response of the feature detectors after the first iteration is relatively weak. In subsequent iterations, the features cooperate and compete with each other until the hierarchical representation remains stable.

While the binarization of the lines and their immediate neighborhood is decided quickly, the network needs some more iterations to decide that locations far-away from the lines belong to the background. This is visible most clearly in the lower right corner of the image where non-uniform lighting created a contrast between the dark paper and the added frame.

Figure 4.17 shows the stable foreground feature response to the difficult examples from Figure 4.11(a). Some additional input/output examples of the network's operation are shown in Figure 4.18. One can see that the network is able to solve the binarization task much better than the thresholding method. It is able to assign

Fig. 4.16. ZIP code binarization – activity over time. The initial response to the image is weak. Features on different hierarchical levels cooperate and compete until a stable representation remains. Most difficult is the assignment of locations far-away from the lines to the background.

Fig. 4.17. ZIP code binarization – network output. The network's stable foreground feature responses to the examples from Figure 4.11(a) are shown.
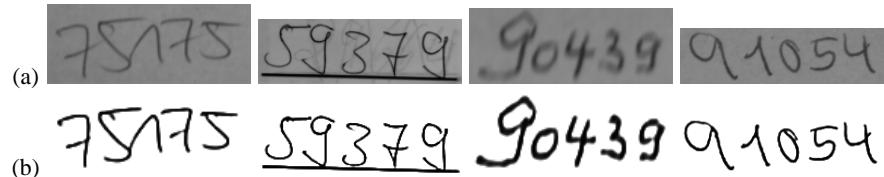
Fig. 4.18. ZIP code binarization – results. (a) network input; (b) stable foreground feature response.

line pixels to the foreground, although they are not much darker than their neighborhood. Also, small clutter and serifs are removed from the foreground and assigned to the background.

This kind of binarization improves the recognition of the ZIP code. In [24] I showed that a similar network increases the acceptance rate of a ZIP code recognition system significantly without decreasing its reliability.

4.3.3 Activity-Driven Update

In Section 4.2.3 it was stated that the update of the cells in the Neural Abstraction Pyramid occurs in a predetermined order: layer by layer in a bottom-up manner and group by group within a layer. The following experiment investigates the effects of relaxing the constraint of a predetermined update order.

This is motivated by the work of Thorpe *et al.* [226]. They found that the human visual system is capable of rapid object categorization within 150ms. Thorpe and Gautrais [227] proposed using a rank-order code, where a neuron emits at most one spike to achieve such rapid feed-forward processing. Using this framework, Van-Rullen *et al.* [232] showed that contour integration is possible even when neurons fire asynchronously at most one spike. Integration of stimuli at less active neurons is influenced by spikes emitted from more active neurons. This can facilitate or suppress the response to less salient stimuli.

The interpretation performance of the Neural Abstraction Pyramid depends on the update order of its feature cells as Fig. 4.19 illustrates using a simple example. It shows how the activity in a one-dimensional feature array develops in three different update modes. The initial cell activities are set to the input stimulus, shown in the front of the figure. It resembles a plateau that increases slightly from the edges (0.5) towards the middle (0.56). The successive iterations show how the cell activities develop over time under a dynamics described by:
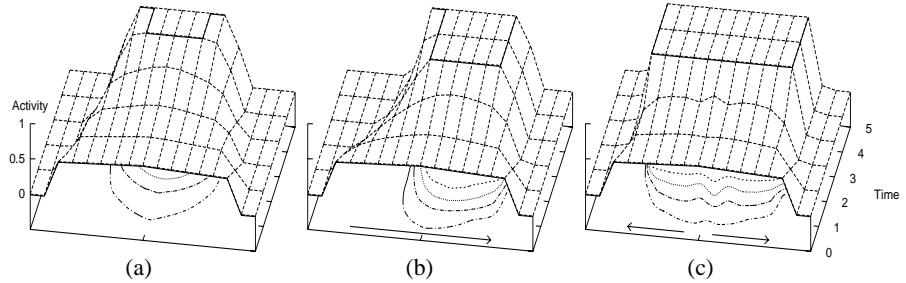
Fig. 4.19. Different update methods: (a) buffered, (b) unbuffered, (c) activity-driven.

$$a_i^{t+1} = \max\left(0, \min\left[1, a_i^t + \frac{a_{i-1}^{t^-} + a_{i+1}^{t^+}}{2} - \frac{1}{2}\right]\right), \qquad (4.10)$$

where $t^\pm$ denotes the time the source activity was updated: either in the previous step t or in the same step $(t + 1)$. Neighboring cells are connected by excitatory links and the activity is mapped to $[0, 1]$ using a negative bias and saturation. The update modes differ in the handling of the activities of neighboring cells.

- **Buffered update** is conservative. All cells of the array have to be computed in time step t before the resulting activities can be used in step $(t + 1)$. This makes the result independent of the update sequence within a time step. All cells can be computed in parallel since no dependencies exist. On the other hand, the buffered dynamics is relatively slow because information travels horizontally with a speed of only one cell per time step.
- **Unbuffered update** computes the cells in a predetermined order, here from left to right. The resulting activity a_{i-1}^{t+1} of a cell is used immediately to compute the activity a_i^{t+1} of its right neighbor. The unbuffered dynamics converges much faster since information travels the full array length from left to right within the same time step. However, the information flow from right to left is still slow which results in an undesired asymmetric response of the system.
- **Activity Driven Update** uses the same unbuffered strategy to speed up convergence. It prevents undesired asymmetric responses by making the update sequence dependent on the array activity. The cells are updated in the order of their activity from the last time step with the most active cell updated first. Fast communication occurs now from the more active to the less active parts of the cell array. Since the activities represent confidences of feature presence in image interpretation tasks, the image parts that are easy to interpret are updated first, which in turn biases and speeds up the interpretation of the more ambiguous image parts. If multiple interpretations compete, the one that first receives support from the context is likely to win the competition.

Activity-driven update also speeds up computation because cell activities that become zero will not get active again and hence do not need to be updated any more.

If the representations are sparse, the vast majority of cells will become inactive quickly. The network design must ensure that cells become inactive only if they are not needed for further computation.

Ordered update does not require global communication. If integrate-and-fire neurons are used as processing elements, those cells that receive a salient stimulus that fits their receptive field will fire earlier than cells that get a suboptimal stimulus. The firing cells trigger their neighbors via excitatory links if the neighboring cells are already close enough to the firing threshold. This leads to an avalanche effect that produces a fast traveling wave of activity. The wave is actively propagated until it either collides with a wave from the opposite direction or it reaches locations that have too low activity to be triggered. If the cells have approximately the same refractory time, all cells that participated in the wave will synchronously become sensitized for a new trigger event again.

The ordered update of cells effectively converts the recurrent lateral connectivity into a feed-forward network where the graph structure depends on the relative activities. In [23] I applied the activity-driven update to a binarization network similar to the one described in the previous section. I demonstrated that binarization using activity-driven update improved ZIP code recognition performance as compared to the buffered update mode.

Although the activity-driven update offers some advantages over buffered update, it will not be used in the remainder of the thesis. The reason for that decision are the computational costs involved with implementing the activity-driven update on a serial machine. However, if the basic processing elements were chosen to be integrate-and-fire neurons implemented with an event-based simulator, activity-driven update would occur naturally. In this case, the ordering would be done using a priority queue for the events.

4.3.4 Invariant Feature Extraction

The last example of this chapter demonstrates that invariant feature extraction is possible in the Neural Abstraction Pyramid architecture. In Section 2.1, we saw that the ventral stream of the human visual system extracts features which are increasingly invariant to object transformations. One example of the supporting neurobiological evidence was published by Ito *et al.* [108]. They found invariance of neural responses to object size and position in the inferotemporal cortex (IT). Such invariance is useful for object recognition since transformations, such as translation, rotation and scaling, do not alter the identity of an object.

However, perfect invariance to such transformation is not desirable. If rotation invariance were perfect, the digits 6 and 9 could not be distinguished and if scale invariance were perfect, a model car could not be distinguished from its full-size original. There is neurobiological evidence that only limited invariance is implemented only for familiar stimuli in the human visual system. For example, Logothetis *et al.* [146] found view-tuned cells in area IT that responded to complex stimuli and showed limited invariance to several transformations. Nazir and O'Regan [165] and Dill and Fahle [53] found evidence against invariance for random dot patterns.

Learning at one position did not help recognition at another position. These results support the view that feature detectors are pooled in the visual system to produce invariance. This invariance does not transfer to unfamiliar patterns for which no specialized feature detectors exist.

Although the human visual system shows invariance to several transformations, in the following, only invariance to translations is discussed to simplify the discussion. Generalization to other transformations should be straightforward.

When implementing invariance to retinal position of a stimulus, one must not forget that the retinal stimulus position depends on eye movements. Saccades and smooth pursuit movements are able to center the object of interest at the fovea. Thus, the neural circuitry has only to implement limited translational invariance for the recognition of objects that are away from the fixation point.

In the Neural Abstraction Pyramid architecture, the degree of possible invariance to translations increases with height. The reason for this is the fixed topographical mapping of positions between the representations at different hierarchical levels. Assuming that the resolution of the layers decreases by a factor of two for each step in height, the following behavior can be observed: a shift of the original image by eight pixels in Layer 0 corresponds to a shift of Layer 1 representations by four cells. Representations in Layer 2 and Layer 3 are shifted by two cells and one cell, respectively. Higher-level representations move only by fractions of the cell size.

Total invariance to translations is only possible at the top of the pyramid, where the resolution drops to a single hypercolumn. For example, the average intensity of an image could be represented there, as it is totally invariant to translation. This feature is not computable in a single step using local connections only. However, it can be computed by a hierarchy of local averaging and subsampling operations, as in image pyramids (see Section 3.1.1).

The reduction in resolution alone does not ensure invariance to linear transformations, because the higher-level representation may change significantly when moved by sub-cell amounts. This aliasing effect is one of the most serious limitations of orthogonal wavelet representations. The critical sampling of their basis functions causes a redistribution of the signal's energy between the levels of the representation. To avoid this effect, Simoncelli *et al.* [214] introduced the concept of shiftability. Intermediate coefficients of a shiftable transformation can be written as a weighted sum of the transform's coefficients, computed at a fixed number of positions only. As a consequence, the sum of the energy of the coefficients does not change when the signal is shifted. The price one must pay for shiftability is generally an increase of the sampling rate as determined by the Nyquist criterion [168], e.g. to twice the critical rate.

The discrete Fourier transformation, discussed in Section 3.1.1, is shiftable by design, but computes global features. When its sinusoidal basis functions are weighted with Gaussian envelopes, Gabor functions are produced that are optimally localized in space and in frequency. For the extraction of shift-invariant features, I use a discrete approximation to Gabor filters.

Table 4.2. Invariant feature extraction – filter design. Two sinusoids that have a wavelength of 4 and a phase shift of 1 are weighted with a binomial kernel of length 8. To make the DC response of the filters zero, $\frac{1}{16}$ of the binomial is subtracted.

Offset i	-3	-2	-1	0	1	2	3	4
Binomial B_8	1	7	21	35	35	21	7	1
$Q_S = \sin(\frac{\Pi}{2}i)$	1	0	-1	0	1	0	-1	0
$Q_C = \cos(\frac{\Pi}{2}i)$	0	-1	0	1	0	-1	0	1
$R_S = Q_S B_8$	1	0	-21	0	35	0	-7	0
$R_C = Q_C B_8$	0	-7	0	35	0	-21	0	1
$S_S = R_S - \frac{1}{16}B_8$	0.9375	-0.4375	-22.3125	-2.1875	32.8125	-1.3125	-7.4375	-0.0625
$S_C = R_C - \frac{1}{16}B_8$	-0.0625	-7.4375	-1.3125	32.8125	-2.1875	-22.3125	-0.4375	0.9375
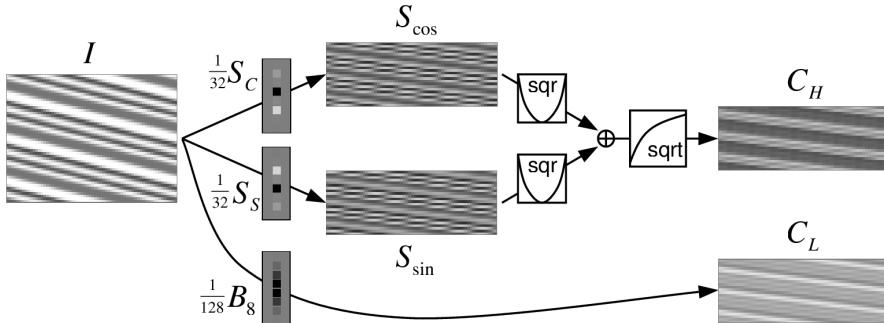
Fig. 4.20. Invariant feature extraction – basic decomposition. The moving one-dimensional signal I is folded with the filter masks $\frac{1}{32}S_S$ and $\frac{1}{32}S_C$ and subsampled to produce the responses $S_{\sin}$ and $S_{\cos}$, respectively. The local energy of these high-frequencies is computed by adding the squared responses and taking the square root. This local energy C_H has a relatively low spatial frequency. Folding I with $\frac{1}{128}B_8$ and subsampling yields a signal C_L of low spatial frequency that represents the low frequencies of I. Both C_H and C_L can be decomposed recursively.

Table 4.2 summarizes the filter design. Sinusoids with wavelength 4 and a phase shift of 1 cell ($Q_S = \sin(\frac{\Pi}{2}i)$ and $Q_C = \cos(\frac{\Pi}{2}i)$) are weighted with a binomial kernel B_8 of length 8. This yields symmetric filters R_S and R_C that have a coefficient sum of 8. To make the DC response of the filters zero, $\frac{1}{16}B_8$ is subtracted from the filters. The resulting filters S_S and S_C are scaled by $\frac{1}{32}$ and used as weights for simple-cell like projection units $S_{\sin}$ and $S_{\cos}$. Figure 4.20 shows the filters as well as the responses of the projections to a moving input signal I in its upper left part.

The vertical one-dimensional input signal I was produced by cyclically shifting a 64 pixel wide signal with a speed of one pixel per step, followed by subsampling to 16 pixels. The horizontal axis is used for the time dimension. The 64 steps shown correspond to a complete motion cycle. The 16 input cells are framed at the bottom

Layer 0 (16×1)   Layer 1 (8×2)   Layer 2 (4×4)   Layer 3 (2×8)
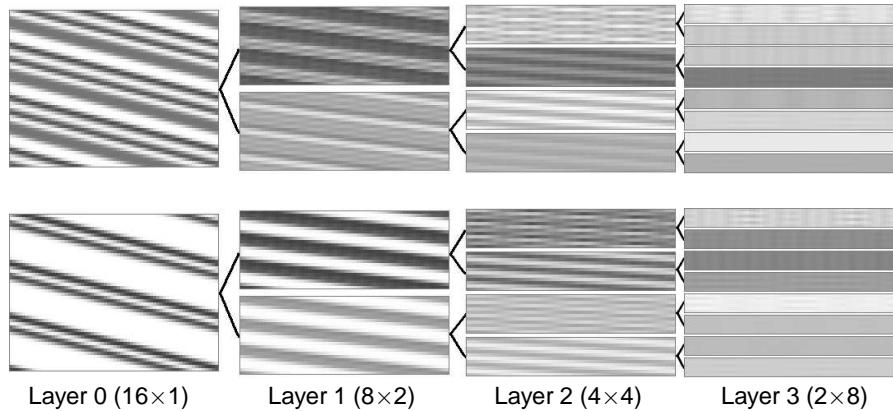
Fig. 4.21. Invariant feature extraction – hierarchical decomposition. Two moving signals are transformed hierarchically by recursive application of the basic decomposition (see Figure 4.20). With height the number of features and their invariance to translation increases. The different input patterns yield different invariant representations.

and at the top by 16 border cells, updated using wrap-around copying of activities. All other feature arrays of the figure have a vertical resolution of eight cells, framed by an eight cell wide border. The filter responses $S_{\sin}$ and $S_{\cos}$ are squared, added together, and passed through the square root transfer function of the output unit C_H. Its response represents the local energy of the high-frequency components of I. This feature is complemented by a feature C_L, representing the low-frequency components of I. It is produced by setting the projection weights of C_L to the low-pass filter $\frac{1}{128}B_8$, as illustrated in the lower part of the figure.

Both the high-frequency energy C_H and the low frequency part C_L lack high frequencies. They move with half the speed of I and intermediate responses can be interpolated easily from the responses of the 8 cells as visible in the diagonal line structure. The one-dimensional invariant feature extraction can be generalized to two dimensions in the same way as two-dimensional wavelets or the 2D DFT are constructed.

As shown in Figure 4.21, the basic decomposition into two invariant features can be applied recursively. This yields a sequence of representations with decreasing resolutions and increasing invariance to translations. The figure shows the response of the invariance hierarchy to two different moving input patterns. The eight Layer 3 responses of length two are almost constant as the pattern moves, but change considerably between patterns. This shows that the high-level features not only have a high degree of invariance to translations, but also possess high representational power.

4.4 Conclusions

The previous chapter introduced the Neural Abstraction Pyramid architecture and presented some simple examples for its use. The examples highlighted different features of the architecture, like its ability to implement local normalization, the cooperation and competition of features on different hierarchical levels, the effects of the update order, and the extraction of invariant features.

All these examples were designed manually. No learning was used so far. The following two chapters discuss, how unsupervised and supervised learning techniques can be applied in the Neural Abstraction Pyramid framework.

5. Unsupervised Learning

The example networks presented so far were designed manually to highlight different features of the Neural Abstraction Pyramid architecture. While the manually designed networks are relatively easy to interpret, their utility is limited by the low network complexity. Only relatively few features can be designed manually. If multiple layers of abstraction are needed, the design complexity explodes with height, as the number of different feature arrays and the number of potential weights per feature increase exponentially.

Hence, there is no choice but to use machine learning techniques to automatically design the network's connectivity from a dataset that describes the application at hand. Generally, three types of machine learning are distinguished [159]:

- **Supervised Learning:** A sequence of input/output pairs $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots,$ $(\mathbf{x}_N, \mathbf{y}_N)$ is given to the learning machine. Its goal is to produce the correct output $\mathbf{y}_i$ if it is confronted with a new input $\mathbf{x}_i$.
- **Unsupervised Learning:** The machine sees only the input sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots,$ $\mathbf{x}_N$. Its goal is to build representations that can be used for reasoning, decision making, predictions, communication, and other tasks.
- **Reinforcement Learning:** The learning machine is now a situated agent that can produce actions $a_1, a_2, \ldots, a_N$ which affect the state of the world around it and hence the later inputs x. The agent receives rewards $r_1, r_2, \ldots, r_N$ and has the goal to maximize them in the long term.

Reinforcement learning [223] requires an agent acting within a world. It is much more general than the other two types of learning but cannot be applied to a perception network alone. If the Neural Abstraction Pyramid were complemented by an inverse pyramidal network that expands abstract decisions to concrete actions, reinforcement learning would be a promising technique for training that agent.

Supervised learning is covered in the next chapter. The remainder of this chapter discusses how unsupervised learning techniques can be applied in the Neural Abstraction Pyramid framework. The chapter is organized as follows: In the next section, I briefly discuss several techniques for unsupervised learning. Then, an algorithm for learning a hierarchy of sparse features in the Neural Abstraction Pyramid is proposed. In Section 5.3 this algorithm is applied to a dataset of handwritten digits. The emerging features are used as input for a supervised digit classifier in Section 5.4.

5.1 Introduction

Unsupervised learning [16] techniques are applicable where supervised learning is not: if desired outputs for the learning machine are unavailable, one can still try to discover the structure underlying a dataset. Since data can be always interpreted in many different ways, some bias is needed to determine which aspects of the input's structure should be captured in the output of the learning machine.

In general, unsupervised learning has the goal of finding useful representations of the given data, for example, by:

– grouping examples to clusters,
– reduction of data dimensionality,
– discovering hidden causes of the data, or
– modeling the data density.

If unsupervised learning is successful, the produced representations can be applied to tasks, such as data compression, outlier detection, classification or to make other learning tasks easier.

The last application refers to the preprocessing step of pattern recognition systems. One of the most important problems in pattern recognition is the extraction of meaningful features from input signals. To compute symbolic information, such as the class of an observed object, it is often useful to aggregate characteristic aspects of the observation into a feature vector that is presented to a classification system. This generally reduces the dimensionality of the data and facilitates generalization by discarding aspects of the signal that correspond to variances not relevant for classification or to noise.

A variety of feature extraction methods exist, e.g., for the problem of handwritten digit recognition [242]. Some methods use the normalized pixel image as input for a powerful statistical or neural classifier [22]. Others use features having a medium degree of abstraction, such as moments [204] or coefficients of the KL-transformation [86]. The most abstract features are extracted by methods that use the digit's structure for recognition [21]. All these features usually need specific tuning towards the task at hand. This makes the transfer to other applications difficult. For this reason, it would be desirable to construct abstract features from a dataset of example images by means of unsupervised learning techniques.

The Kalman filter and non-negative matrix factorization are unsupervised learning methods that have already been discussed in Chapter 3.

Clustering. One of the best known methods of unsupervised learning is the K-means algorithm [145] for clustering of input vectors. It is also known as LBG method [144] for vector quantization. The algorithm assumes that the data vectors $\mathbf{x}_i$ can be grouped into K clusters and replaced by the mean μ_{c_i} of the assigned cluster c_i without much loss of information. The K-means algorithm optimizes iteratively a squared error criterion:

$$\sum_{i=1}^{N} \|\mathbf{x}_i - \mu_{c_i}\|^2. \tag{5.1}$$

The centroids μ_j are initialized arbitrarily, e.g. to randomly chosen examples. Each iteration consists of two steps:

- Step1: Assign the data vectors to the closest centroid: $c_i = \underset{1 \leq j \leq K}{\operatorname{argmin}} \|\mathbf{x}_i - \mu_j\|$.
- Step2: Move the centroids to the mean of the assigned examples: $\mu_j = \langle \mathbf{x}_i \rangle_{c_i = j}$.

In a probabilistic framework, the K-means algorithm is a special case of the expectation-maximization (EM) algorithm [52] applied to mixture density estimation. Since each step decreases the quantization error until the assignment does not change any more, the K-means algorithm finds a local minimum of the error function 5.1. The quality of the approximation depends on the number of centroids and on the initialization. One can start with a low number of clusters that are split recursively to avoid initialization problems and to determine the number of clusters needed.

Competitive Learning. The algorithm described above can be viewed as competition between the centroids to respond to the data vectors. Similar competition is achieved in winner-takes-all (WTA) networks. These neural networks consist of an input layer and a layer of laterally connected processing elements which assess the similarity between the current data vector $\mathbf{x}_i$ and their weight vector $\mathbf{w}_j$. The assessment can be done using a distance function $d(\mathbf{x}_i, \mathbf{w}_j)$, as in self organizing maps (SOM) [126], or by computing the scalar product $\mathbf{x}_i \cdot \mathbf{w}_j$. If the weight vectors and the inputs are normalized, the scalar product equals the cosine of the angle spanned by both vectors.

The unit with the smallest distance or the largest scalar product is called the winner. Its output is set to one, while the outputs of all other units are set to zero. This operation requires global information about the similarities. It can be implemented by lateral inhibition.

Competitive learning [201] in WTA networks can be achieved by adapting only the weight vector $\mathbf{w}_k$ of the winning unit k. One possibility is to add a fraction of the current data vector: $\Delta \mathbf{w}_k = \eta \mathbf{x}_i$, where η is a learning rate that decreases over time, followed by a renormalization of the weight length: $\mathbf{w}_k \leftarrow \mathbf{w}_k / \|\mathbf{w}_k\|$. This leads to a redistribution of weight strengths from the weights connecting to inactive input components to the weights connecting to active inputs. Hence, the unit will respond more strongly if the same input is presented again. The weight vectors of the units loosing the competition remain unchanged.

It is also possible to use the difference between the weight vector and the input to make the weights similar to the inputs: $\Delta \mathbf{w}_k = \eta(\mathbf{x}_i - \mathbf{w}_k)$. If the network has a topological structure, such as in the SOM, neighboring units can be moved in the same direction to produce a topological feature map.

Principal Component Analysis. One of the key ingredients of unsupervised learning methods are Hebbian [91] weight updates: $\Delta w_i = \eta x_i y$, where x_i is the activity of the presynaptic unit and $y = \sum_i w_i x_i = \mathbf{w} \cdot \mathbf{x}$ is the activity of the postsynaptic unit. The Hebbian learning rule can be stated informally as: 'Neurons that fire together – wire together.' If the inputs are zero mean, it captures the correlations

$\mathbf{C}_{ij} = \langle x_i x_j \rangle$ between the input units: $\langle \Delta \mathbf{w} \rangle = \eta \mathbf{C} \mathbf{w}$. The correlation matrix $\mathbf{C}$ can be viewed as linear transformation of the weight vector. In the long run, the eigenvector $\mathbf{e}$ with the largest eigenvalue λ will dominate the weight change: $\mathbf{C}\mathbf{e} = \lambda \mathbf{e}$. If the Hebbian rule is combined with normalization, the weights develop towards the principal component of the data.

Generic Hebbian learning is unstable since the weights grow without limits. To avoid this effect, Oja [169] proposed adding a weight-decay term to the update rule: $\Delta w_i = \eta y (x_i - y w_i)$. It implements a self-normalization of the weight vector. The unit's output y represents the orthogonal projection of a data vector $\mathbf{x}$ onto the weight vector $\mathbf{w}$. Its variance $\langle y^2 \rangle$ is maximized by the learning rule.

If more than the first principal component is desired, the reconstruction $\mathbf{r} = y \mathbf{w}^T = \mathbf{w} \mathbf{x} \mathbf{w}^T$ of the data vector that is based on y can be subtracted from $\mathbf{x}$ to produce new examples $\mathbf{x}' = \mathbf{x} - \mathbf{r}$, which can be analyzed by a second unit to extract the second principal component. Another possibility is to extend the learning rule for a multi-unit network to $\Delta \mathbf{w}_r = \eta y_r \left(\mathbf{x} - \sum_{s \leq r} y_s \mathbf{w}_s \right)$ as proposed by Sanger [203]. The principal component analysis (PCA) network decorrelates its outputs y_k and hence removes linear dependencies from the input. Because the number of output units can be chosen smaller than the number of input components, the linear PCA transformation can be used to reduce the dimensionality of the data with minimal loss of variance.

Independent Component Analysis. Another unsupervised learning technique is called independent component analysis (ICA) [115, 26]. Its goal is to find a linear transformation of the data vectors $\mathbf{x}$ that produces a representation $\mathbf{y} = \mathbf{W} \mathbf{x}$ with components which are not only uncorrelated, but statistically independent. This is motivated by the assumption that the data vectors have been produced as a linear mixture $\mathbf{x} = \mathbf{A} \mathbf{s}$ of independent sources s_i. If this is the case, ICA can be used to separate the sources by estimating an unmixing matrix $\mathbf{W} = \mathbf{A}^{-1}$, a problem known as blind source separation.

ICA is applicable if at most one of the sources has a Gaussian distribution. Principal component analysis and whitening are usually required as preprocessing steps to remove second order correlations from the data vectors. This discards information about sign and amplitude of the sources.

Some ICA methods use the fact that if two sources s_1 and s_2 are independent, then any nonlinear transformations $g(s_1)$ and $h(s_2)$ are uncorrelated. Thus, they perform nonlinear decorrelation to separate the sources.

According to the central limit theorem, sums of nongaussian random variables are closer to a Gaussian than the original ones. This is used in ICA methods that maximize the non-gaussianity of the output components. To measure non-gaussianity, cumulants of higher-order moments, such as the kurtosis, the normalized form of the fourth central moment measuring the peakedness of a distribution, are used.

Because the estimation principles discussed above use non-quadratic functions, the computations needed usually cannot be expressed using simple linear algebra.

Numerical optimization methods, e.g. gradient descent or the fixed-point algorithm called FastICA [106], are employed to estimate $\mathbf{W}$.

Other Unsupervised Learning Techniques. Because the goals of unsupervised learning can vary greatly, there exist many different unsupervised learning techniques that have not been discussed so far.

One example is slow feature analysis (SFA), recently proposed by Wiskott and Sejnowski [244]. This method focuses on finding representations that change only slowly as input examples undergo a transformation. SFA expands the input signal non-linearly and applies PCA to this expanded signal and its time derivative. The components with the lowest variance are selected as slow features. Temporal smoothing of the network's output is also the basis of the method proposed by Földiak [69] for the learning of invariant features.

Another example of unsupervised techniques is the learning of sparse features. Sparse representations can be viewed as generalization to the local representations generated by WTA networks. While in local representations exactly one unit is active, in sparse representations multiple units can be active, but the ratio between the active and the inactive units is low. This increases the representational power of the code, facilitates generalization, allows for controlled inference, increases the capacity of associative memories, implements fault tolerance, and allows for the simultaneous representation of multiple items by superposition of individual encodings [70]. There is substantial evidence that the human visual system utilizes sparse coding to represent properties of visual scenes [215].

A simple local unsupervised algorithm for learning such representations in a nonlinear neural network was proposed by Földiak [68]. It uses Hebbian forward connections to detect non-accidental features, an adaptive threshold to keep the activity ratio low, and anti-Hebbian decorrelating lateral connections to keep redundancy low. It produces codes with few active units for frequent patterns, while less probable patterns are encoded using a higher number of active units.

Other algorithms for the learning of sparse features adjust connection weights by explicitly maximizing measures of sparseness, successfully producing V1 simple cell-like features [170]. This class of algorithms is closely related to ICA since sparse distributions are also non-Gaussian.

Beyond sparseness, another interesting property of a representation is the interpretability of encodings. While a randomly chosen codeword could only signal the presence of an item, Barlow [15] suggested that the cortex might use sparse codes where the individual units signal the presence of meaningful features in the input. In this scheme, items are encoded by combinations of features.

In the following section, I introduce an unsupervised learning algorithm for the forward projections of the Neural Abstraction Pyramid. It is based on Hebbian weight updates and lateral competition and yields a sequence of more and more abstract representations. With increasing height, the spatial resolution of feature arrays decreases, feature diversity increases and the representations become increasingly sparse.

5.2 Learning a Hierarchy of Sparse Features

In order to make the Neural Abstraction Pyramid approach to image interpretation work, a sequence of increasingly abstract models of the potential image content is needed. In the last chapter, such models were designed manually. In Section 4.3.2, for instance, the Gestalt approach was used to construct weight templates for the extraction of foreground/background features, step edges, and oriented lines. The feature cells in that network cooperated and competed to achieve binarization of handwritten digits.

In the previous section, several unsupervised learning methods were discussed. They produced new representations from a set of data vectors. The representations found by the different methods have various properties that sometimes are contradictory. For example, while PCA tries to preserve variance, SFA focuses on the least variant features. All methods discussed so far are single-step transformations. In contrast, the Neural Abstraction Pyramid represents the image content on different levels of abstraction. Hence, a sequence of transformations is needed to extract features which become increasingly abstract.

One way to produce such features is by repeated application of an unsupervised learning technique. This imposes a constraint on the learning method: Its output must be admissible as input for the next stage of learning. Hence, unsupervised learning methods that drastically change the nature of the representation cannot be used for this task. Another constraint is that features within a level should be treated equally. This excludes methods which produce an ordered sequence of features.

In the following, I present an unsupervised learning method that produces representations with the following desired properties:

– **Completeness:** All salient features of the input image should be represented.
– **Sparseness:** The value of a feature should be zero at most positions and high at only a few positions.
– **Fairness:** All feature arrays of a layer should contribute approximately equally to the representation.

The proposed method is based on Hebbian weight updates and lateral competition. It can be applied repeatedly to learn a hierarchy of sparse features.

Training starts at Layer 1 of the pyramid that analyzes small windows of Layer 0 representations. It proceeds upwards from layer to layer. Using the topmost representation on layer $(l-1)$ as input, it learns weight templates for the forward projections of feature cells that reside on layer l. Since the number of layers is logarithmic in the image size, only a few steps are needed to train the entire hierarchy.

5.2.1 Network Architecture

Figure 5.1 illustrates the architecture of the network that is used for the unsupervised learning of a hierarchy of sparse features. It is a Neural Abstraction Pyramid (compare to Chapter 4) with six layers. Each layer l consists of $4 \cdot 2^l$ excitatory feature arrays and a feature sum array. All but the bottom layer also contain an array
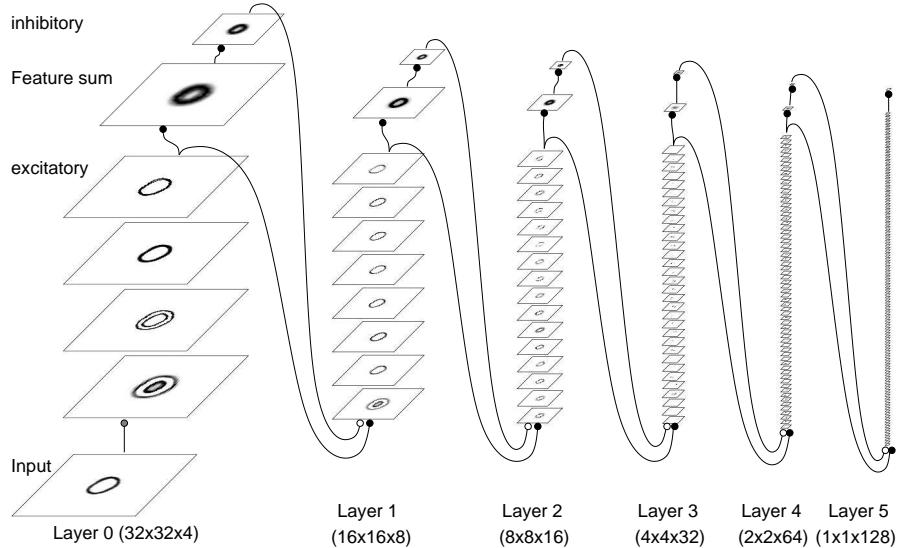
Fig. 5.1. Learning a hierarchy of sparse features – network architecture. The Neural Abstraction Pyramid consists of six layers. Only forward projections are used. Excitation is specific while unspecific inhibition is mediated by the subsampled smoothed feature sums.

that represents the subsampled sum of the features in the layer below. This feature is inhibitory.

Patterns are presented to the input feature array located in the bottom layer of the network. The input is analyzed by four excitatory feature arrays of Layer 0 that compute center-surround features. They each have a single lateral projection with direct access to the input array. The weights of these projections have a difference-of-Gaussian structure with two different scales and two polarities. Fine and coarse foreground and background features are detected. The projection unit has a linear transfer function and contributes with weight one to the output unit which has a saturating rectifying transfer function $f_{\text{p\_sat}}$ ($\alpha = 1$, see Section 4.2.4) that limits the activities to the interval $[0, 1]$. This transfer function is also used for the output units of the excitatory feature cells in the higher layers.

The feature sum S_l has only a single lateral projection with direct access to all excitatory features of a layer. It weights the 3×3 neighborhood of its hypercolumn with a binomial kernel that is scaled with a gain factor. The gain decreases with height, such that the central weight decreases from 0.125 in Layer 0 to 0.015625 in Layer 5. Both the transfer function of the projection unit and the one of its output unit are linear. On the next higher layer the inhibitory feature array $\hat{S}_l$ computes the average of a 2×2 window of S_l.

The basic processing elements used for the excitatory features in Layer 1 to Layer 5 have two projections. One is the specific excitatory forward projection that directly accesses overlapping 4×4 windows of all excitatory feature arrays in the layer below. The other is the unspecific inhibitory projection that accesses the sub-

sampled feature sum $\hat{S}_l$ with weight -1. The transfer functions of both projections are linear.

In the following, the weight from the specific excitatory projection to the output unit of feature kl is called $\mathcal{E}_{kl}$, and the weight from the inhibitory projection is called $\mathcal{I}_{kl}$. Both gain factors determine, how specific a feature cell will react to stimuli. If the excitation is large, compared to the inhibition, the cell reacts unspecifically to many stimuli that partially match its excitatory projection. On the other hand, if inhibition is large the cell is sharply tuned to the stimuli that exactly match the specific weights of its excitatory projection.

5.2.2 Initialization

The weights w_{kl}^{pq} of the excitatory projections are initialized unspecifically: Larger positive weights are used in the center and weaker weights are used towards the periphery of the receptive field window. The weights have a random component and are normalized to a sum of one. This normalization of total excitatory weight strength is maintained during learning. The excitatory weights are not allowed to become negative.

The excitatory gain $\mathcal{E}_{kl}$ is initialized to 2.0, while the inhibitory gain $\mathcal{I}_{kl}$ is initialized to zero. Hence, initially the excitatory features will react very unspecific to all stimuli present on the lower layer.

The bias weights of all projection units and output units are set to zero and not changed during learning.

5.2.3 Hebbian Weight Update

A combination of winner-takes-all learning and Hebbian weight updates [91] is used to make the excitatory weights specific. The idea is to change the weight template of the locally most active feature cell such that it becomes more specific to the current input. This means it will react more strongly to the same stimulus and react less strongly to other stimuli.

For each training step, an image is chosen randomly from the dataset. It is loaded into the input feature array at bottom layer of the pyramid, and the activities of all feature cells are computed in the appropriate order.

The following learning rules are applied only at positions where the subsampled feature sum $\hat{S}_{(l-1)}$ of the inputs and the smoothed sum of the outputs S_l are nonzero. This avoids learning when there is nothing to learn.

Furthermore, the subsampled input sum $\hat{S}_{(l-1)}$ must have at most two larger values in its 8-neighborhood. This focuses the features to respond mostly to local maxima and ridges of the input sum.

For the hypercolumns (i, j) of layer l meeting the above criteria, the most active feature k_{max} and the feature k_{sec} with the second highest activity are determined. The qth weight $w_{k_{\mathsf{max}}l}^{pq}$ of the excitatory projection p of the winning feature k_{max} is changed as follows:

$$\Delta w^{pq}_{k_{\mathrm{max}}l} = \eta_l \cdot a_{\mathsf{in}} \cdot a_{\mathsf{out}},$$

$$\text{with} \quad a_{\mathsf{in}} = \mathcal{H}(\mathsf{I}^{pq}_{k_{\mathrm{max}}l}, \mathsf{J}^{pq}_{k_{\mathrm{max}}l}) \cdot a_{i^*j^*k^*l^*},$$

$$a_{\mathsf{out}} = a_{ijk_{\mathrm{max}}l} - a_{ijk_{\mathrm{sec}}l}.$$

(5.2)

A weight is increased by an amount that is proportional to the product of the scaled input activity a_{in} and the amount a_{out} by which the activity of the winning feature cell exceeds the one of the second best feature. The use of the difference of the two most active features instead of the cell activity has a decorrelating effect on the features. They are forced to respond to different stimuli. How the address $i^*j^*k^*l^*$ of the source feature cell is determined is explained in Section 4.2.2.

Because more example windows are available to determine the lower-layer features than for the higher-layer ones, the learning rate η_l increases with height, e.g. $\eta_l = 0.001 K_l$, where K_l is the number of excitatory features in layer l.

The scaling factor $\mathcal{H}(\mathsf{I}^{pq}_{k_{\mathrm{max}}l}, \mathsf{J}^{pq}_{k_{\mathrm{max}}l})$ used for the input activity depends on the offset of a weight relative to the position $(\Upsilon_{ll^*}(i), \Upsilon_{ll^*}(j))$ in the source layer $l^* = (l-1)$ that corresponds to the position (i, j) in layer l. $\mathcal{H}$ is one in the center of the window and descends to zero towards the periphery. The weighting enforces a centered response of the feature cells. This is done because non-centered stimuli can be represented by neighboring feature cells.

The Hebbian term (5.2) makes the excitatory weights larger. To prevent unlimited weight growth, the sum of the excitatory weights is kept at a value of one by scaling down all weights by a common factor. The net effect of the normalized Hebbian update is that the weights receiving strong input are increased and the other weights are decreased.

5.2.4 Competition

The normalization of the sum of the weights of excitatory projections, described above, is a form of competition. The weights compete to receive a large portion of the projection's constant weight sum.

In addition, competition between the K_l excitatory features of layer l is needed to achieve the desired properties for the produced representation. Care must be taken that a learning rule enforcing one constraint does not interfere with another rule enforcing a different constraint.

To fulfill the fairness constraint, the winning frequency of all templates should be about the same. In order to achieve this, a feature's inhibitory gain $\mathcal{I}_{kl}$ is increased each time it wins the competition; otherwise, it is decreased. This makes features whose winning frequency is above average less active and more specific. Consequently, these features will not win too frequently in the future. On the other hand, features that win less often become more active and less specific and therefore now win more often. The change is done by adding a small constant $\Delta^{\mathcal{I}f}_{kl}$ to $\mathcal{I}_{kl}$, such that the net effect for a feature that has an average winning frequency is zero:

$$\Delta^{\mathcal{I}f}_{kl} = \begin{cases} \eta_f & : \quad k = k_{\mathrm{max}} \quad [\mathsf{winning}] \\ -\frac{\eta_f}{K_l - 1} & : \quad k \neq k_{\mathrm{max}} \quad [\mathsf{not\ winning}] \end{cases},$$

(5.3)

where η_f is a constant.

To achieve a complete representation, the features are forced to react to all significant input stimuli by constraining the smoothed sum S_l of the features in layer l to be equal to the subsampled sum $\hat{S}_{(l-1)}$ of the input features from layer $(l-1)$:

$$\Delta_{kl}^{\mathcal{E}c} = \frac{\eta_c}{K_l}\, a_{ijkl}(\hat{S}_{ij(l-1)} - S_{ijl}),$$

$$\Delta_{kl}^{\mathcal{I}c} = -\Delta_{kl}^{\mathcal{E}c}, \tag{5.4}$$

where η_c is a constant. If the activity of the features is too low, the excitatory gains of the active features are increased, and they are disinhibited at the same time. The opposite behavior applies when the features are too active.

To enforce sparseness, the activity of a winning feature must be made large, e.g. to $\mathcal{V} = 0.75$:

$$\Delta_{kl}^{\mathcal{E}s} = \left\{ \begin{array}{rcll} \eta_s(\mathcal{V} - a_{ijk_{\max}l}) & : & k = k_{\max} & \text{[winning]} \\ 0 & : & k \neq k_{\max} & \text{[not winning]} \end{array} \right. , \tag{5.5}$$

where η_s is a constant. If the activity of the winner is too small, its excitatory gain is increased; otherwise, it is decreased.

If adding $\Delta_{kl}^{\mathcal{I}f}$ and $\Delta_{kl}^{\mathcal{I}c}$ to $\mathcal{I}_{kl}$ makes the inhibitory gain negative, its weight is added to $\mathcal{E}_{kl}$, and $\mathcal{I}_{kl}$ is set to zero. Vice versa, if $\mathcal{E}_{kl}$ should become negative from adding $\Delta_{kl}^{\mathcal{E}c}$ and $\Delta_{kl}^{\mathcal{E}s}$ it is set to zero, and its weight is added to $\mathcal{I}_{kl}$.

The efficacy of the constraint enforcing rules, described above, can be controlled by the learning constants. One possible choice could be: $\eta_f = \eta_c = \eta_s = 0.1\eta_l$. The rules are designed such that their net effect goes to zero if the learned representation has the desired properties. Then the templates describing the computation of the features become stable, and the training can be stopped.

The number of training images needed to determine the weights of the weight templates for a layer increases with the height of that layer since the number of examples per image decreases and the number of weights per layer increases.

Because the emerging representations are sparse, most of the weights will be close to zero after training and can be pruned away without significant loss. This speeds up the computation and saves memory.

5.3 Learning Hierarchical Digit Features

The properties of the described unsupervised learning algorithm can be illustrated by applying it to a dataset of handwritten digits. Here, digits are used which have been extracted by Siemens AG from German ZIP codes written on large-size letters.

The available examples are partitioned as follows: 44,619 digits constitute the training set (TRN), 5,379 digits are available for testing the performance of a recognition system and to stop training (TST), and 6,313 digits are used for final validation (VAL).

Table 5.1. Learning a hierarchy of sparse features – emerging representations.

layer	name	feature arrays	hypercolumns	feature cells	input size
5	digits	128	1×1	128	32×32
4	curves	64	2×2	256	16×16
3	strokes	32	4×4	512	8×8
2	lines	16	8×8	1024	4×4
1	edges	8	16×16	2048	2×2
0	contrasts	4	32×32	4096	1×1

Since the digits show a high degree of variance, some preprocessing steps are necessary prior to presentation to the pyramid. Preprocessing consists of binarization, size and slant normalization. The images are scaled to 24×24 pixels and are centered into the 32×32 input array at the bottom layer of the pyramid.

The Neural Abstraction Pyramid is initialized at the lowest level ($l = 0$) with contrast detectors. These have a center-surround type receptive field that analyzes the intensities of the input image. Four different features are used: center-on/off-surround and center-off/on-surround in two scales, representing the fine and coarse details of the foreground and the background, respectively. The feature arrays are surrounded by a border of the same width that is set to zero.

Repeated application of the unsupervised learning method, described above, yields following representations (compare to Table 5.1):

– **Edges:** Vertical, horizontal, and diagonal step edges are detected at Layer 1.
– **Lines:** At Layer 2 short line segments with 16 different orientations are detected.
– **Strokes:** Larger line segments that have a specific orientation and a specific curvature are detected at Layer 3. Detectors for line endings and specific parallel lines emerge as well.
– **Curves:** The feature detectors at Layer 4 react to typical large substructures of digits, such as curves, crossings, junctions, etc.
– **Digits:** The feature cells at the topmost Layer 5 see the entire digit. Consequently, detectors for typical digit shapes emerge.

Figure 5.2 shows in its upper right part a preprocessed input digit. On the upper left, the activities of the contrast detectors are shown. They provide input to the edge features via the specific weights of the excitatory projections. On the left side of the figure, the activity of the edge feature arrays is shown. It can be seen that the feature cells detect oriented step edges. For instance, the feature in the first row detects edges on the lower side of horizontal lines. It receives input from foreground features in the upper part of its projection and from background features in the lower part of the projection. The right side of the figure shows the four best stimuli of the training set that excited the features maximally. In the center of these stimuli, the 2×2 area of responsibility of Layer 1 features is shown in the original contrast. Its neighborhood is shown with a lower contrast.

Fig. 5.2. Learning a hierarchy of sparse features – edge features. Shown are from left to right: activity of the feature arrays for a digit (Input "0"), excitatory weights to contrast features, excitatory and inhibitory gain, stimuli that caused the highest winning activity. There are pairs of step-edges for horizontal, vertical and (/) diagonal lines. The diagonal (\) is represented by a single feature only. The feature in the last row detects background and reacts most strongly to the inner part of small loops.

All learned edge features are different. There are pairs of horizontal, vertical, and lower-left to upper-right diagonal step edges. The upper-left to lower-right diagonal is represented by only one feature that responds to the center of the line. This is no surprise since lines of this orientation are less frequent in handwriting. The feature in the last row serves a special purpose. It is excited by centered background and responds most to the inner part of small loops.

The edge features are not very specific. Since the inhibitory gain is small, they respond also to suboptimal stimuli. Only the last feature is strongly inhibited by the sum of the contrast features to avoid responses to foreground.

On the next higher level, Layer 2, the 16 features respond to lines of different orientations. The line detectors show a sharper orientation tuning than the edge features. Four line features that detect approximately horizontal lines are shown in
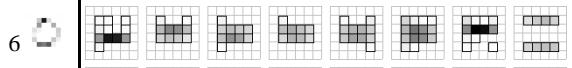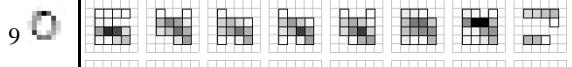
Line(k)	Excitatory Weights								$\mathcal{E}_{kl}$ $\mathcal{I}_{kl}$	Best Stimuli

Fig. 5.3. Learning a hierarchy of sparse features – line features. Four of the 16 features were chosen that respond to approximately horizontal lines. The other features respond to lines of other orientations.
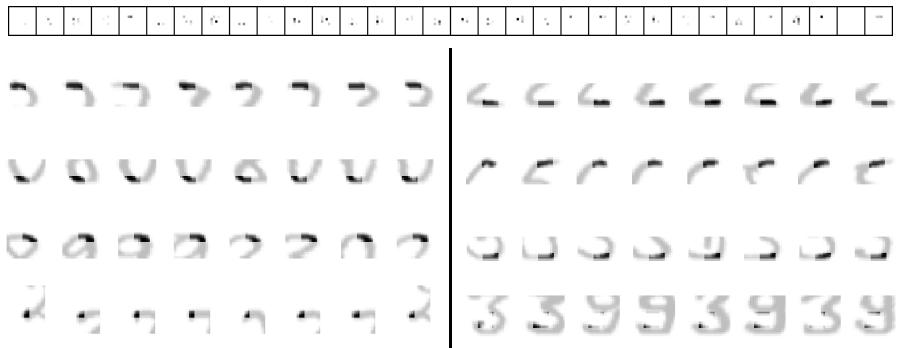
Fig. 5.4. Learning a hierarchy of sparse features – stroke features. Shown are the eight best stimuli of eight features that detect horizontal strokes with different curvature. The upper part of the figure shows the sparse activity of all 32 stroke features.

Figure 5.3. They receive input mostly from the pair of horizontal step edges. The lower horizontal edge feature is accessed by the lower part of the forward projections, while the upper horizontal edge is accessed by the upper part of the projection. Step edges of other orientations contribute less to horizontal line features. The access to the Layer 1 background feature that is done by the upper and the lower row of projection weights is also interesting.

The 32 stroke features at Layer 3 are not as easy to describe as the line features. They react to different local line shapes. Figure 5.4 shows the eight best stimuli for eight of the stroke features that react to approximately horizontal lines. In addition to the orientation preference, some of the stroke features are also sensitive to line curvature. Others seem to react to line endings. The feature in the lower right corner is stimulated optimally by two parallel horizontal lines. It responds to the background between the lines. The figure also shows in its upper part the activity of
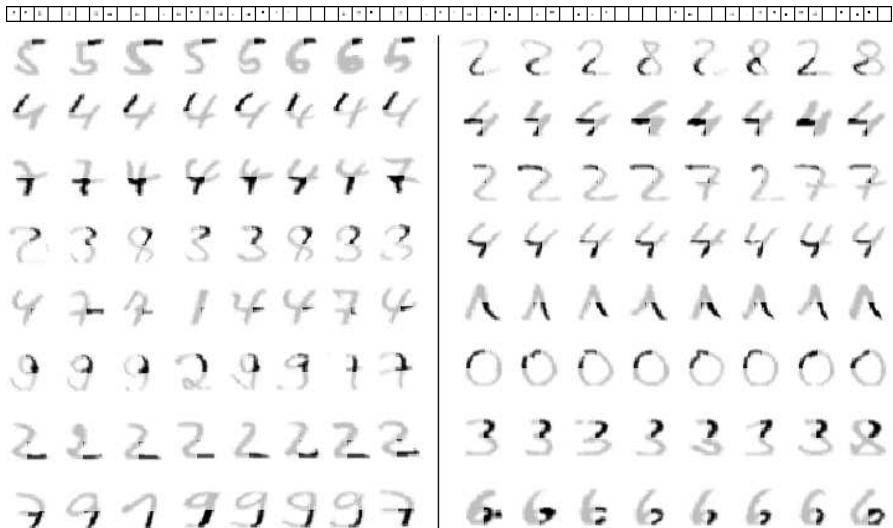
Fig. 5.5. Learning a hierarchy of sparse features – curve features. Shown are the eight best stimuli of the 16 first features. They respond to typical digit parts. The upper part of the figure shows the sparse activity of all 64 curve features.

all stroke features when the input from Figure 5.2 is presented to the network. One can see that the representation is already quite sparse.

Figure 5.5 shows the eight best stimuli of the first 16 of the 64 curve features that reside on Layer 4 of the pyramid. They detect typical digit parts, such as open and closed loops, line crossings, and major strokes. It can be observed that for most curve features all of the best stimuli belong to the same digit class. The activity of the curve features is sparse since not all typical configurations of strokes are contained in a single digit.

The best stimuli of some of the top-layer digit features are shown in Figure 5.6. For the left side of the figure, digit features that react best to one of the ten digit classes were selected. The right side shows digit features that were selected because they react to examples from different classes. They seem to focus on some aspect of the digit, such as to the presence of a vertical line or to a characteristic curve. One must ask the question: 'What do the best stimuli have in common?' to find out what a specific feature cell detects.

The emerging feature detectors do not represent all possible combinations of substructures, but only the typical ones. The more frequent combinations of lower level features are represented by multiple similar features with greater detail than the less frequent ones. When going up in the hierarchy of representations, the correlation of feature activities with the digit class increases. This is remarkable since no class information has been presented to the system so far.
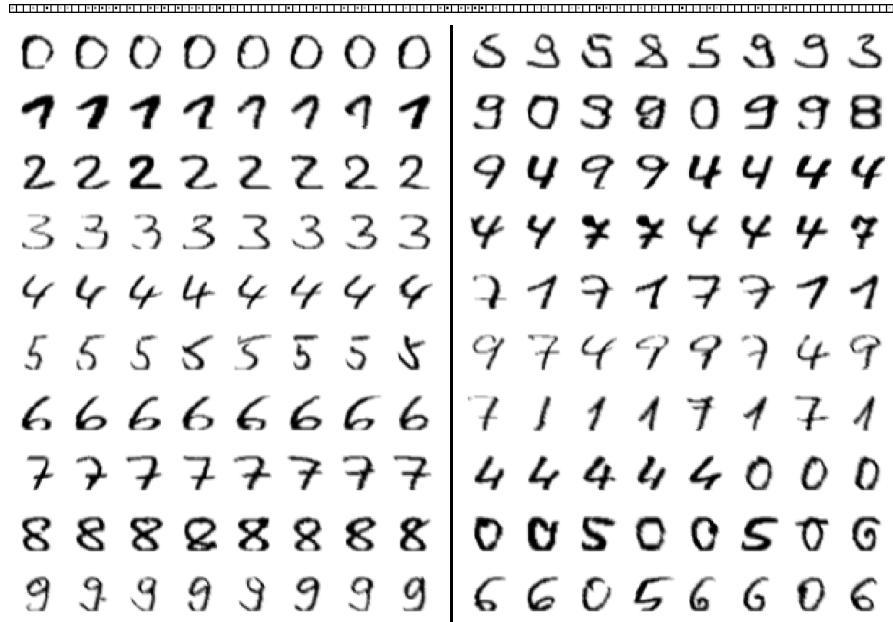
Fig. 5.6. Learning a hierarchy of sparse features – digit features. Shown are the eight best stimuli. For the left column features were chosen that correspond to a single class. The right column shows features that focus on some other aspect of a digit.

5.4 Digit Classification

In the following experiments, the usefulness of the learned sparse features for digit recognition is investigated. First, only two layers of the pyramid are constructed. The resulting representation is based on features that represent oriented lines. It has the same total size as the input image. Table 5.2 shows the performance of a KNN classifier and two feed-forward neural networks (FFNN) that were trained with backpropagation using the digit's gray values and the extracted lines as fea-

Table 5.2. Learning a hierarchy of sparse features – classification of low-level features. Zero-reject substitution rates of different classifiers.

features classifier	Gray		Lines	
	TST	VAL	TST	VAL
KNN 15	2.98	2.87	4.53	4.36
FFNN 1024 − 10	5.82	6.48	2.04	2.14
FFNN 1024 − 64 − 10	2.49	2.65	1.90	2.04

Table 5.3. Learning a hierarchy of sparse features – classification of abstract features. Zero-reject substitution rates of different classifiers that input the upper four layers of the learned hierarchy of sparse features.

classifier	TST	VAL
KNN 15	3.59	3.64
FFNN $1920 - 10$	1.71	1.66
FFNN $1920 - 16 - 10$	1.99	1.77
FFNN $1920 - 32 - 10$	1.71	1.73
FFNN $1920 - 64 - 10$	1.67	1.68
FFNN $1920 - 128 - 10$	1.65	1.49

tures. One can see that the performance of the neural networks is better for the more abstract features.

In the second experiment, the top four layers of the feed-forward pyramid are fed into a $1920 - 128 - 10$ FFNN to classify the digits. After 120 epochs of online-training with a learning rate of $\eta = 0.01$, a zero-reject substitution rate of 1.65% on the test set and a rate of 1.49% on the validation set was observed. Table 5.3 shows the results for different numbers of hidden units, as well as for a network without hidden units and a KNN classifier. These rates compare favorably to the results published in [21] for the same dataset. One can also reject ambiguous digits by looking at the two best classes. The substitution rate drops to 0.55% when 2.52% of the validation set are rejected and to 0.21% for 7.9% rejects. Figure 5.7 shows the substitution-reject curve of this classifier compared to the structural classifier and a time-delay neural network (TDNN) classifier [21]. Clearly, the classifier that uses the features extracted by the Neural Abstraction Pyramid performs about as well as the combination of the other two classifiers. The figure also shows the results when the new classifier is combined sequentially [22] with the other two. Now the zero-reject substitution rate drops to 1.17%. The substitution rate can be reduced to 0.30% with 3.60% and to 0.11% with 9.20% rejects. These results are the best the author knows for this dataset.

5.5 Discussion

This chapter presented an unsupervised learning algorithm for the design of the forward projections in the Neural Abstraction Pyramid. The algorithm was applied to a dataset of handwritten digits to produce a sequence of increasingly abstract digit representations. The emerging feature detectors are meaningful and can be interpreted in terms of detected combinations of digit substructures. This leads to a hierarchical image description that is distributed and sparse. When looking at the
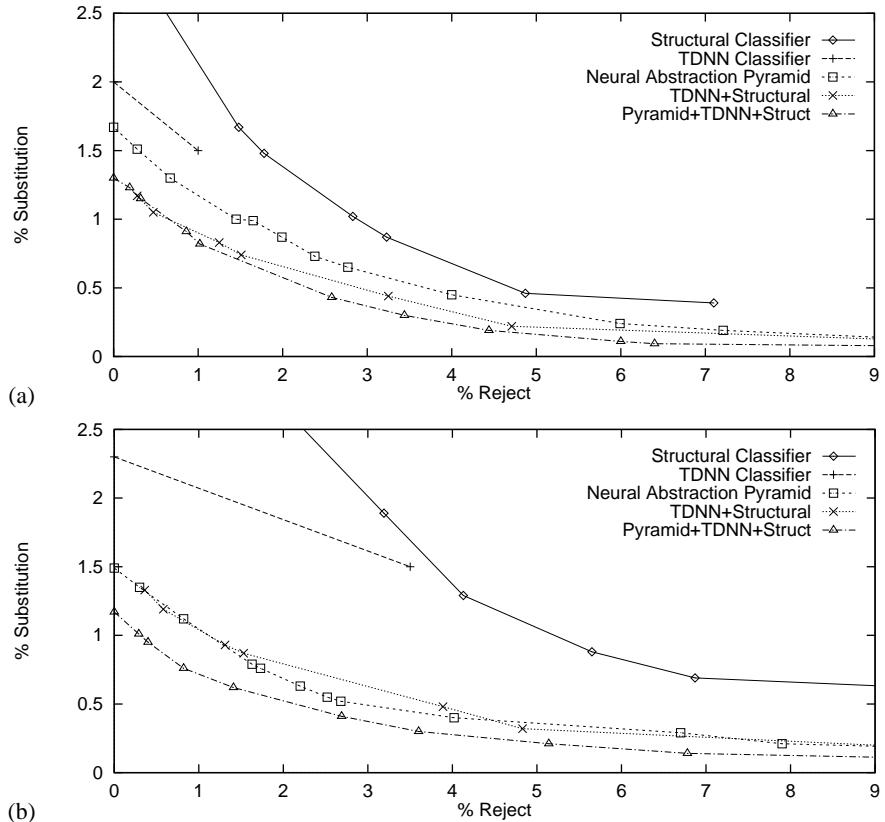
Fig. 5.7. Learning a hierarchy of sparse features – performance of different digit classifiers: (a) test set; (b) validation set.

best stimuli for the feature detectors, one can see that these are not similar in terms of a simple pixel-based distance measure, but in terms of their recursive decomposition to substructures. Hence, the pyramidal digit representation becomes increasingly invariant against distortions when going up in the hierarchy.

The extracted features facilitate recognition of the digits. When used as input to an FFNN-classifier, the recognition performance observed was very satisfactory. It outperforms any single classifier that has been tested on that dataset and is about as good as the combination of the TDNN and the structural digit recognizer. When combined with these two classifiers, the recognition performance improves further.

6. Supervised Learning

In the last chapter, supervised learning has already been used to classify the outputs of a Neural Abstraction Pyramid that was trained with unsupervised learning. In this chapter, it is discussed how supervised learning techniques can be applied in the Neural Abstraction Pyramid itself.

After an introduction, supervised learning in feed-forward neural networks is covered. Attention is paid to the issues of weight sharing and the handling of network borders, which are relevant for the Neural Abstraction Pyramid architecture. Section 6.3 discusses supervised learning for recurrent networks. The difficulty of gradient computation in recurrent networks makes it necessary to employ algorithms that use only the sign of the gradient to update the weights.

6.1 Introduction

Supervised learning is more precisely defined than unsupervised learning. Given a training set of input/output vector pairs $(\mathbf{x}_i, \mathbf{y}_i)$, the goal of supervised learning is to produce for unseen inputs $\mathbf{x}_j$ (generated from the same distribution) outputs $\mathbf{o}_j$ which are as close as possible to the desired outputs $\mathbf{y}_j$. It does not suffice to memorize the training set since generalization to new examples is desired.

Two supervised learning problems can be distinguished:

- **Classification:** Here, the output vector $\mathbf{y}$ represents the class of an object to recognize. The examples are assigned to a discrete number of classes. If the classification system is also able to produce a classification confidence, this quantity can be used to reject ambiguous examples.
- **Function approximation:** This problem is also known as regression. Here, the input/output examples are samples from a function $\mathbf{y} = f(\mathbf{x})$. The output of the learning machine is continuous.

Classification can be viewed as a special case of function approximation. For example, an approximation to the characteristic function of a set can be used to classify whether or not examples belong to the class described by the set.

6.1.1 Nearest Neighbor Classifier

One particularly simple supervised classifier is the nearest-neighbor (NN) classifier [46]. It assigns an input vector $\mathbf{x_i}$ to the class $\mathbf{y}_k$ of the training vector $\mathbf{x}_k$ that is

closest to it according to some distance measure $d(\mathbf{x}_i, \mathbf{x}_j)$. While the NN classifier achieves perfect performance on the training set, generalization on a test set might not be satisfactory. Better generalization is generally achieved if not only the closest training vector is considered, but the classification decision is based on the K nearest neighbors. If the majority rule is used for the decision, this system is called KNN classifier.

The KNN classifier stores all examples of the training set. While this requires no training time, it is frequently advantageous to invest some time for learning in order to extract the essence of a dataset for later use as data model. This speeds up the recall and improves generalization.

6.1.2 Decision Trees

Several supervised learning techniques have been proposed in the literature. One example is decision trees. Breiman *et al.* [36] proposed classification and regression trees (CART) for supervised learning problems. Here, the input-vector components represent attributes of the examples. The inner nodes of decision trees use one attribute at a time to recursively split the input space. Depending on the node's decision, one of its child nodes is visited until a leaf node is reached. The leaves store the output of the tree. The order in which the individual attributes are queried is important for the performance of decision trees. Several algorithms have been proposed for the top-down inference of decision trees from a dataset. One example is the ID3 algorithm, proposed by Quinlan [181] for classification. It asks the question first that is most informative about the class. Decision trees can learn any consistent dataset perfectly, but this may not be desirable. For this reason, ID3 has been extended to the C4.5 algorithm [182], which uses a Shannon entropy criterion for the pruning trees. Another possibility to limit the tree size is to grow a tree only until additional splitting of nodes produces no significant information gain. The preference for small trees is motivated by the principle of Occam's razor [32] which prefers a simpler explanation of a dataset over a more complicated one.

Decision trees work well if the input attributes are discrete, and the dimensionality of the input vectors is not too large. However, the class boundaries produced by decision trees are axis-parallel to the dimensions of the input space. This may cause difficulties if the true class boundaries are non-aligned.

6.1.3 Bayesian Classifier

The theoretically optimal classifier is based on Bayes' theorem of conditional probability. It is described by the Bayesian classification rule:

$$c(\mathbf{x}) = \operatorname*{argmax}_{j}\ p(H_j|\mathbf{x}) = \operatorname*{argmax}_{j}\ p(H_j) \cdot p(\mathbf{x}|H_j),$$

where $p(H_j)$ is the *a-priori* probability of class j, $p(\mathbf{x}|H_j)$ describes the distribution of the examples from class j in the input space, and $c(\mathbf{x})$ is the classification produced.

The Bayesian classification rule is optimal but difficult to apply in real-world situations. The reason for this is that neither $p(H_j)$ nor $p(\mathbf{x}|H_j)$ are known. While estimation of $p(H_j)$ from the dataset is straightforward, it is difficult to model the example distributions $p(\mathbf{x}|H_j)$ if the dimensionality of the input space is large.

This problem is known as 'the curse of dimensionality' [27]. If, for instance, a grid-based representation is used to estimate a distribution, the number of grid cells grows exponentially with the dimension of the space when the resolution for each dimension is kept constant. Furthermore, in high-dimensional spaces the distance of randomly chosen points tends to be constant for any distance measure [29]. Finally, sets of points that cannot be separated linearly in a low-dimensional space may become linearly separable when transformed into a space of higher dimension.

6.1.4 Support Vector Machines

The last property of high-dimensional spaces is exploited by kernel methods. The idea behind kernel methods is to transform data vectors into a feature space that usually has a huge or even infinite dimensionality. The kernel trick allows for working in that feature space without expressing it explicitly. High-dimensional dot products are computed by kernels as functions of the original data vectors.

If multiple linear separations of data vectors are possible, which one of the separations should be chosen? This question is considered by the theory of structural risk minimization. While empirical risk minimization focuses on good performance for the training set only, structural risk minimization tries to find the learning machine that yields a good trade-off between low empirical risk and small capacity.

The principle of structural risk was proposed by Vapnik and Chervonenkis [233]. According to this principle the number of training examples must be large compared to the degrees of freedom of a learning machine. The capacity of a learning machine is measured by the VC dimension. It is defined as the size of the largest set of points that can be split by the learning machine into two subsets in all possible ways.

Large-margin classifiers are one application of structural risk minimization. The idea is that a linear separation that has a large margin to all training examples is preferred against a separation with a small margin. This improves generalization since it reduces the degrees of freedom of the classifier.

In the last years, support-vector machines (SVM) [47] have become popular classifiers. They express the classification decision function in terms of a subset of the training examples which are close to a decision boundary, the support vectors. Using the kernel trick and structural risk minimization, they separate the classes in high-dimensional spaces with large margins. Powerful optimization techniques have been developed for support-vector machines [207].

6.1.5 Bias/Variance Dilemma

All supervised learning systems face the bias/variance dilemma [81]. The error of such a system can be divided into three components:

- **Bias:** systematic component of approximation error, measuring how close the average classifier produced by the learning algorithm will be to the target function;
- **Variance:** sensitivity of approximation to the finite size of the training sample, measuring how much each of the learning algorithm's guesses will vary with respect to each other;
- **Intrinsic target noise:** due to the noise of the training sample, measuring the minimum classification error associated with the Bayes optimal classifier for the target function.

A tradeoff between the bias component and the variance component of the system must always be made. While reducing the degrees of freedom of a learning machine lowers the variance error, the restriction of possible solutions increases the bias error. For this reason, it is important to ensure that a restricted learning system is still appropriate for the task to be learned. In the context of the Neural Abstraction Pyramid architecture, such restrictions include hierarchical network structure, local connectivity, and weight sharing. In Chapter 4 it was discussed, why such restrictions are appropriate for image interpretation tasks.

In general, it is hard to assess the generalization of a learning system using the training set alone. For this reason, one may hold back some examples of the dataset from training to test generalization [238]. One way to restrict the degrees of freedom of a learning system trained with an incremental algorithm is to use early stopping [179]. This terminates the training if the performance on a test set starts to degrade. Another way to assess generalization is by cross-validation [205] which uses different subsets of the training set to train multiple classifiers. If random subsamples of the training set are used instead of subsets, the method is called bootstrapping [60].

6.2 Feed-Forward Neural Networks

Artificial neural networks are popular tools for supervised learning. Feed-forward neural networks (FFNN) compute complex functions in directed acyclic graphs of primitive functions. Usually, the nodes of the graph are arranged in layers. The primitive functions computed by the nodes access other nodes via weighted links. For example, $\sum$-units compute a weighted sum of their inputs. This sum is passed through a transfer function that may be non-linear.

The first trainable neural network was proposed in 1958 by Rosenblatt [194]. The classical perceptron is a multilayer network that could be trained to recognize patterns on a retina. The processing units, perceptrons, computed weighted sums of their inputs, followed by a threshold nonlinearity.

The model was simplified and analyzed by Minsky and Papert [158]. They showed that the perceptron learning algorithm can solve linearly separable problems, but cannot learn all possible boolean functions. This result applies to any feed-forward neural networks without hidden units. For example, the XOR function

cannot be computed by such networks. Later, it was shown that feed-forward networks with enough nonlinear hidden units can approximate any continuous function over a compact domain [104].

6.2.1 Error Backpropagation

Key to the success of feed-forward neural networks was gradient-based learning. Gradient-based learning algorithms minimize a cost function E by gradient descent. Frequently, the quadratic approximation error is used as cost function:

$$E = \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{o}_i - \mathbf{y}_i\|^2, \tag{6.1}$$

where $\mathbf{o}_i$ is the output of the network when the ith example $\mathbf{x}_i$ is presented to it, and $\mathbf{y}_i$ is the desired output for that example.

In a gradient descent method, a parameter w is modified according to the partial derivative of the error function E with respect to it:

$$\Delta w = -\eta \frac{\partial E}{\partial w}, \tag{6.2}$$

where $\eta > 0$ is the learning rate. If η is chosen small enough, the repeated application of (6.2) lowers E, until a local minimum is reached.

The simplest example of gradient descent learning is the delta rule. It is applicable for linear networks without hidden units: $o^{(i)} = \mathbf{w}\mathbf{x}_i$. For the weights w_j of the network's output unit, the learning rule (6.2) can be rewritten as:

$$\Delta w_j = -\eta \sum_{i=1}^{N} (o^{(i)} - y^{(i)}) x_j^{(i)}, \tag{6.3}$$

where $x_j^{(i)}$ is the jth component of the input vector $\mathbf{x}_i$.

In order to make the gradient descent idea work, differentiable transfer functions are needed. In the example above, the transfer function was linear and hence could be omitted from the analysis. In multi-layered networks, it is necessary to have nonlinear transfer functions for the hidden units, to make them more powerful than networks without hidden units. One frequently used non-linear transfer function is the sigmoid $o = f_{\mathsf{sig}}(\xi) = \frac{1}{1+e^{-\xi}}$ that has already been discussed in Section 4.2.4. Its derivative can be expressed in terms of its output: $f'_{\mathsf{sig}}(\xi) = o(1 - o)$.

For the gradient computation in multi-layered networks, the idea of backpropagation of an error signal was introduced e.g. by Rumelhart *et al.* [200]. The error backpropagation technique is an efficient method to compute the partial derivative of a cost function with respect to a weight, in the same way as the ordered update of the network activity in the feed-forward mode is an efficient method to compute the network output. As the name suggests, backpropagation visits the nodes of the network in the opposite order of the feed-forward step.

Because the cost function (6.1) sums the error contributions of individual examples, its partial derivative with respect to a weight w_{jk} from unit j to unit k is a sum of components $\frac{\partial E^{(i)}}{\partial w_{jk}}$ that can be computed for each example i separately. The key idea of backpropagation is that $\frac{\partial E^{(i)}}{\partial w_{jk}}$ can be expressed in terms of a backpropagated error $\delta_k^{(i)}$ and the source activity $o_j^{(i)}$ present at the weight:

$$\frac{\partial E^{(i)}}{\partial w_{jk}} = o_j^{(i)} \cdot \delta_k^{(i)}. \tag{6.4}$$

The backpropagated error $\delta_k^{(i)}$ of a hidden unit is a weighted sum of the errors $\delta_l^{(i)}$ of all units l receiving input from unit k, multiplied with the derivative of the transfer function f_k, producing the output $o_k^{(i)} = f_k(\xi_k^{(i)})$ of unit k:

$$\delta_k^{(i)} = \frac{df_k}{d\xi_k^{(i)}} \sum_l w_{kl} \delta_l^{(i)} \qquad \text{[hidden unit]}, \tag{6.5}$$

with $\xi_k^{(i)} = \sum_j w_{jk} o_j^{(i)}$ describing the weighted sum of the inputs to k.

If unit k is an output unit, its error component can be computed directly:

$$\delta_k^{(i)} = \frac{df_k}{d\xi_k^{(i)}} (o_k^{(i)} - y_k^{(i)}) \qquad \text{[output unit]}, \tag{6.6}$$

where $y_k^{(i)}$ is the component of the target vector $\mathbf{y}_i$ that corresponds to unit k.

The backpropagation technique can be applied to the Neural Abstraction Pyramid architecture. Since the basic processing element, described in Section 4.2.1, is a two-layered feed-forward neural network, directed acyclic graphs of such processing elements form a large feed-forward neural network with shared weights.

A simple modification is needed for the update of shared weights: the sum of all weight-updates, which have been computed for the individual instances of a weight, is added to it. By replacing the weight-instances with multiplicative units that receive an additional input from a single unit which outputs the value of the shared weight, one can show that this indeed modifies the weight in the direction of the negative gradient [193].

When implementing error backpropagation in the Neural Abstraction Pyramid, one must also take care to handle the border effects correctly. The simplest case is when the border cells of a feature array are set to a constant value. Since the derivative of a constant is zero, the error component arriving at these border cells does not need to be propagated any further. In contrast, if the activity of a border cell is copied from a feature cell, the error component arriving at it must be added to the error component of that feature cell.

Because the weights of a projection unit are stored as an adjacency list in the template of the unit, it is easiest to implement the sum in Equation 6.5 by accumulating contributions from the units receiving inputs from it. As the network is traversed

in reverse order, each unit multiplies its accumulated error with the derivative of its transfer function and increments the accumulators of its source units by this quantity, weighted with the efficacy of the connection. This requires the same address computations as the forward-step that computes the activities of the network. In the same loop, the weight modifications can be computed since the source activity can be easily accessed. It is also useful to store in the forward step not only the outputs of the units, but also their weighted input sums that have not been passed through the transfer function since they may be needed for the computation of the derivative.

The choice of the learning rate η is also important for the convergence of gradient descent. It should be chosen inversely proportional to the square root of the degree of weight sharing [135] since the effective learning rate is increased by sharing the weight. Hence, the learning rate is low in the lower layers of the pyramid and increases with height.

6.2.2 Improvements to Backpropagation

Since the training of neural networks with backpropagation can be time-consuming, several improvements to the basic method have been developed to speed up training.

Online Training. On example of such improvements is online training. In the original formulation, the contributions for the weight update from all examples of the training set must be computed before a weight is modified. This is called batch training. If the number of training examples is large, this may be computationally inefficient. Online training updates the weights after every example presentation. To avoid oscillations, the examples must be presented in a randomized order. Online training is only an approximation to gradient descent. Nevertheless, due to the noise introduced by the randomized example presentation it may escape small local minima of the error function and even improve generalization of the network [30]. If the training set contains much redundancy, online training can be significantly faster than batch training since it estimates the gradient from subsets of the training set and updates the weights more often.

Momentum Term. Another modification to speed up gradient descent is the addition of a momentum term:

$$\Delta w^{(t)} = -\eta \frac{\partial E}{\partial w} + \alpha \Delta w^{(t-1)}, \tag{6.7}$$

with $0 \leq \alpha \leq 1$. It adds a fraction of the last weight update to the current update. The momentum term makes gradient descent analogous to particles moving through a vicious medium in a force field [180]. This averages out oscillations and helps to overcome flat regions in the error surface.

Advanced optimization methods have been developed for the supervised training of neural networks [135]. They include conjugate gradient, second order methods, and adaptive step size algorithms.

Conjugate Gradient. The main idea of conjugate gradient [66, 161] is to find a descent direction which does not disturb the result of the previous iteration. It picks a descent direction, e.g. the gradient, finds a minimum along this direction using a line-search, and moves into the conjugate direction where the gradient does not change its direction, only its length. Conjugate gradient can be viewed as an advanced way of choosing the momentum. It requires fewer iterations than generic gradient descent. However, each iteration is more expensive, and it can only be used for batch training since an accurate line search must be performed.

Second Order Methods. Second order methods [18] consider more information about the shape of the error function than the mere gradient. They use a quadratic approximation to estimate the curvature of the error function. Quasi-Newton methods, for example, attempt to keep a positive definite estimate of the inverse Hessian directly, without the need for matrix inversion. They are based on gradient information only but require a line search. One popular algorithm is the BFGS method [67] that uses $O(N)$ operations for an iteration and needs to store a $N \times N$ matrix, where N is the number of weights.

Another popular second order method is the Quickprop algorithm, proposed by Fahlman [63]. It uses a one-dimensional quadratic approximation to the error function for each weight, yielding the learning rule:

$$\Delta w_{ij}^{(t)} = \Delta w_{ij}^{(t-1)} \cdot \frac{\nabla_{ij} E^{(t)}}{\nabla_{ij} E^{(t-1)} - \nabla_{ij} E^{(t)}}, \qquad (6.8)$$

where $\nabla_{ij} E^{(t)}$ denotes the partial derivative $\frac{\partial E^{(t)}}{\partial w_{ij}}$. Quickprop can be initialized using a gradient descent step. Care must be taken to avoid weight updates that are too large.

Adaptive Step Size. The choice of the learning rate η is crucial for both the stability and the convergence speed of gradient descent. One can try to adapt a global learning rate automatically [202], but this is difficult when the error function has different properties in different dimensions.

For this reason, several methods have been developed that maintain a local learning rate for each weight. The algorithm proposed by Silva and Almeida [212], for example, increases the learning rate if successive weight updates go in the same direction and decreases it otherwise. Another example is the SuperSAB algorithm [228] that combines the adaptive learning rate with a momentum term.

One of the fastest and most robust methods for training neural networks is the resilient propagation (RPROP) algorithm, proposed by Riedmiller and Braun [191]. It uses only the sign of the gradient $\nabla_{ij} E^{(t)}$ and an adaptable step size $\Delta_{ij}^{(t)}$ to change the weights:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \quad \text{if } \nabla_{ij} E^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & , \quad \text{if } \nabla_{ij} E^{(t)} < 0 \\ 0 & , \quad \text{else} \end{cases} . \qquad (6.9)$$

The weight is modified only if successive gradients do not change the direction: $\nabla_{ij}E^{(t-1)} \cdot \nabla_{ij}E^{(t)} \geq 0$. The step size is modified according to:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} \cdot \Delta_{ij}^{(t-1)} & , \quad \text{if } \nabla_{ij}E^{(t-1)} \cdot \nabla_{ij}E^{(t)} > 0 \\ \eta^{-} \cdot \Delta_{ij}^{(t-1)} & , \quad \text{if } \nabla_{ij}E^{(t-1)} \cdot \nabla_{ij}E^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \quad \text{else} \end{cases} . \qquad (6.10)$$

It is increased by a factor η^{+} if successive updates go in the same direction and decreased by multiplying it with η^{-} if the gradient direction changes. In the latter case, $\nabla_{ij}E^{(t)}$ is set to zero to avoid a step size update in the next iteration of the algorithm. The factors comply with $0 < \eta^{-} < 1 < \eta^{+}$. Recommended values are $\eta^{-} = 0.5$ for fast deceleration and $\eta^{+} = 1.2$ for cautious acceleration of learning. The step sizes are initialized uniformly to Δ_{o}. It is ensured that they do not leave the interval $[\Delta_{\min}, \Delta_{\max}]$. The RPROP algorithm has been shown to be robust to the choice of its parameters [107]. It is easy to implement and requires only the storage of two additional quantities per weight: the last gradient and the step size.

Mini Batches. RPROP as well as other advanced optimization techniques are batch methods because they need an accurate estimate of the gradient. For real-world tasks, the training set is frequently large and contains many similar examples. In this case, it is very expensive to consider all training examples before updating the weights.

One idea to overcome this difficulty is to only use subsets of the training set, so-called mini batches, to estimate the gradient. This is a compromise between batch training and online learning. The easiest way to implement mini batches is to update every n examples. Another possibility is to work with randomly chosen subsets of the training set. Møller [162] investigated the effect of training with mini batches. He proposed to start with a small set and to enlarge it as the training proceeds.

For the RPROP algorithm, the gradient estimate must not only be accurate but stable as well. Because the signs of successive gradient evaluations determine the adaptation of the learning rates, fluctuations of the gradient estimate may slow down the convergence.

For this reason, I used RPROP with slowly changing random subsets of the training set. A small working set of examples is initialized randomly. In each iteration only a small fraction of the set is replaced by new randomly chosen examples. As training proceeds, the network error for most of the examples will be low. Hence, the size of the working set must be increased to include enough informative examples. The last few iterations of the learning algorithm are done with the entire training set.

Such an approach can speed up the training significantly since most iterations of the learning algorithm can be done with only a small fraction of the training set. In addition, the ability of the network to learn the task can be judged quickly because during the first iterations, the working set is still small.

6.2.3 Regularization

As already discussed in Section 6.1, the capacity of a learning machine must be appropriate for the task to ensure generalization. One way to restrict the capacity of a neural network is to use few adaptable parameters. This can be done by keeping the network small or by sharing weights.

Another way is to lower the capacity of a high-capacity machine by regularization. Regularization constrains the parameters such that only smooth approximations to the training set are possible.

It was already mentioned that early stopping has a regularizing effect. The reason for this is that weights of a neural network are still relatively small when the training is stopped early. This limits the nonlinearity of the network since the transfer functions are almost linear for small inputs. Limited nonlinearity yields decision functions that smoothly approximate the training set.

Weight Decay. Another possibility to regularize a neural network is weight decay. Krogh and Hertz [128] proposed adding a term to the cost function E that penalizes large weights:

$$E_d = E + \frac{1}{2}\lambda \sum_k w_k^2, \qquad (6.11)$$

where λ is a parameter that determines the strength of the penalty. If gradient descent is used for learning, the penalty leads to a new term $-\lambda w_k$ in the weight update:

$$\Delta w_k = -\eta \frac{\partial E_d}{\partial w_k} = -\eta \left(\frac{\partial E}{\partial w_k} + \lambda w_k \right). \qquad (6.12)$$

The new term would decay weights exponentially if no forces from the cost function E were present.

Low-Activity Prior. It is also possible to include terms in the cost function that enforce properties of the representation present at hidden units. For instance, one can force units to have a low average activity, e.g. $\alpha = 0.1$:

$$E_a = E + \frac{1}{2}\lambda \sum_k (\langle o_k \rangle - \alpha)^2, \qquad (6.13)$$

where $\langle o_k \rangle$ denotes the expected value of the activity of unit k. Gradient descent yields the additional term $\lambda(\langle o_k \rangle - \alpha)$ that must be multiplied with the derivative of the transfer function of unit k and added to its error component δ_k. A low-activity prior for hidden units, combined with a force that produces variance, can yield sparse representations.

6.3 Recurrent Neural Networks

So far, the function graph describing the neural network was acyclic. If the graph of primitive functions contains cycles, it is called recurrent neural network (RNN). In

Section 3.2 it was discussed why RNNs are more powerful than FFNNs. Another motivation for the use of RNNs is the fact that the brain is a non-linear dynamical system with recurrent connectivity.

In the following, the supervised training of discrete-time RNNs will be covered. RNNs do not map isolated input vectors $\mathbf{x}_i$ to output vectors $\mathbf{o}_i$, but respond to a sequence of input vectors $\mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(T)}$ with a sequence of activities $\mathbf{o}_i^{(0)}, \mathbf{o}_i^{(1)}, \ldots, \mathbf{o}_i^{(T)}$ at their output units. This must be considered by the cost function:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{t=0}^{T} \gamma_t \|\mathbf{o}_i^{(t)} - \mathbf{y}_i^{(t)}\|^2, \tag{6.14}$$

where $\mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)}, \ldots, \mathbf{y}_i^{(T)}$ is the desired output sequence, and γ_t weights the error of time step t. Both the input and the desired output may be constant. In this case, the RNN is trained to converge towards an attractor that coincides with $\mathbf{y}_i$ for the output units of the network.

Training recurrent networks is difficult due to the non-linear dynamics of the system. Although methods for supervised training of RNNs exist that do not use gradient information [8], most training methods for RNNs are gradient based [11]. Two basic methods have been developed for the computation of the gradient in RNNs: backpropagation through time (BPTT) and forward propagation, also called real-time recurrent learning (RTRL). Both methods will be discussed in the following.

6.3.1 Backpropagation Through Time

The basic idea of BPTT, proposed e.g. by Werbos [240], is very simple. It is illustrated in Figure 6.1. Part (a) of the figure shows a small RNN that has weights
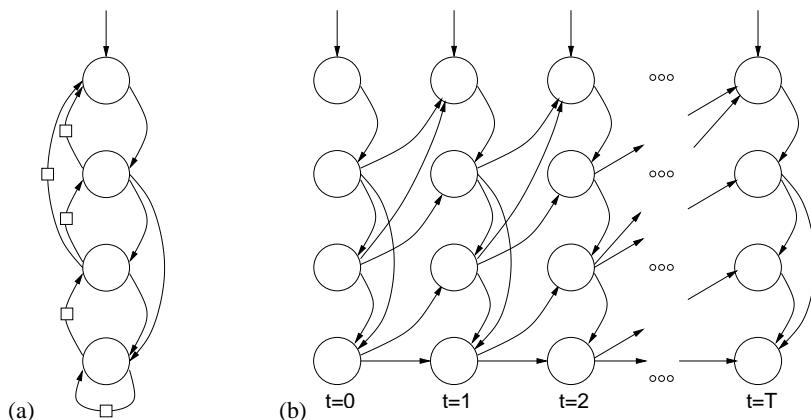
Fig. 6.1. Backpropagation through time. (a) recurrent network, the squares indicate buffered access with a time delay of one step; (b) network unfolded in time. The buffered links now go from one time step to the next. This produces a large FFNN with shared weights.

with direct and weights with buffered access. All units are updated in each time step in top-down order. Only units that have been updated earlier are sources of direct access weights.

The computation done in the RNN is equivalent to the one of a FFNN that has been constructed by unfolding the RNN in time. Part (b) of the figure shows the unfolded network. The units and the direct-access weights were copied for each time step. The buffered-access weights w_{ij} now connect unit i in time step t with unit j in step $(t + 1)$.

Since the unfolded network is a directed acyclic graph, the error backpropagation technique can be applied. It propagates error components in reverse update order and hence backwards in time. Two simple modifications to the generic backpropagation algorithm are necessary for BPTT.

First, the output units of FFNNs used to be sinks of the graph, with no units accessing their activity. This is different in RNNs, where the activity of output units is fed back into the network. Hence, the error component $\delta_k^{(i,t)}$ of an output unit k for example i not only depends on the direct contribution $e_k^{(i,t)} = \gamma_t(o_k^{(i,t)} - y_k^{(i,t)})$ from the cost function (6.14) for time t, but the backpropagated error $\sum_l w_{kl}\delta_l^{(i,t^*)}$ arriving from nodes l accessing it must also be considered. The source time t^* for the error components is either the same step t if the unit is accessed directly or the next step $(t + 1)$ for buffered access. Both contributions must be added before the combined error can be multiplied with the derivative of the transfer function f_k:

$$\delta_k^{(i,t)} = \frac{df_k}{d\xi_k^{(i,t)}} \left(e_k^{(i,t)} + \sum_l w_{kl}\delta_l^{(i,t^*)} \right), \tag{6.15}$$

where $\xi_k^{(i,t)}$ denotes the net activity of unit k for example i at time t.

The second modification needed for BPTT was already used for shared weights in FFNNs. BPTT produces additional weight sharing because a weight is replicated for each time step. As before, the weight updates computed for the individual weight instances must be added to compute the update for the shared weight.

Since BPTT propagates the error backwards through time until it reaches the initial time step $t = 0$, it can not only be used to adapt the weights of the network, but also to modify the initial activities of the units.

6.3.2 Real-Time Recurrent Learning

The BPTT algorithm, presented above, is very efficient, requiring only $O(1)$ operations per weight instance, but it is a batch-method that needs to store the entire history of the recurrent computation for the error backpropagation.

Williams and Zipser [241] proposed computing the gradient of the cost function (6.14) using forward propagation. The resulting algorithm is called real-time recurrent learning (RTRL).

RTRL maintains quantities $\pi_{jkl}^{(i,t)} = \frac{\partial o_j^{(i,t)}}{\partial w_{kl}^{(i,t)}}$ that represent the sensitivity of a unit j with respect to a weight from unit k to unit l. They are initialized to zero for $t = 0$:

$\pi_{jkl}^{(i,0)} = 0$. Parallel to the update of the unit's activities, the sensitivities are updated as well:

$$\pi_{jkl}^{(i,t+1)} = \frac{df_j}{d\xi_j^{(i,t)}} \left[\sum_m w_{jm}^{(i,t)} \cdot \pi_{mkl}^{(i,t)} + \delta_{kj} o_j^{(i,t)} \right], \qquad (6.16)$$

where δ_{kj} denotes the Kronecker delta function. Gradient descent updates on the weights are then achieved by the learning rule:

$$\Delta w_{kl}^{(i,t)} = -\eta \sum_j e_j^{(i,t)} \cdot \pi_{jkl}^{(i,t)}. \qquad (6.17)$$

RTRL does not need to go back in time. Hence, it can be applied in an online fashion. However, it is computationally more expensive than BPTT since $O(n^4)$ operations are needed per time step, with n representing the number of units in the network. If the network is fully connected, this corresponds to $O(n^2)$ operations for each weight per time step. It also needs $O(n^3)$ memory cells to store the sensitivities π_{jkl}.

6.3.3 Difficulty of Learning Long-Term Dependencies

Although the above algorithms for training RNNs have been known as long as the backpropagation algorithm for FFNNs, RNNs are used less often for real-world applications than FFNNs. One of the reasons might be that training RNNs is difficult.

Since RNNs are nonlinear dynamical systems, they can expand or contract the gradient flow. If in a network the magnitude of a loop's gain is larger than one for multiple consecutive time steps, the gradient will explode exponentially. In contrast, if the magnitude of a loop's gain is smaller than one the gradient will decay exponentially.

The gain of a loop in a RNN depends on the magnitudes of the weights involved, and on the derivatives of the transfer functions. Since the networks are frequently initialized with small weights and use sigmoidal transfer function with small derivatives, most of the gradients decay in time.

This affects the learning of long-term dependencies, where in long sequences early inputs determine late desired outputs. Bengio *et al.* [28] showed that the gradient decay is the reason why gradient-based learning algorithms face an increasingly difficult problem as the duration of the dependencies to be captured increases. They showed that it is either impossible to store long-term memories or that the gradient is vanishing. Learning with vanishing long-term gradients is difficult since the total gradient, which is a sum of short-term and long-term gradient components, will be dominated by the short-term influences.

Long Short-Term Memory. Several proposals have been made to overcome this difficulty. One is the use of long short-term memory (LSTM), proposed by Hochreiter and Schmidhuber [100]. This algorithm works in networks that include special-purpose units, called memory cells, that are used to latch information. The

memory cells are linear units that have a fixed self-connection. They enforce a constant, non-exploding, non-vanishing error flow.

Access to memory cells is controlled by multiplicative gate units. Input gate units learn to protect the constant error flow within a memory cell from perturbation by irrelevant inputs. Likewise, output gate units learn to protect other units from perturbation by currently irrelevant memory contents.

Learning is done by a gradient method that is a combination of BPTT and modified RTRL. The LSTM algorithm has been applied to several non-trivial problems. For instance, it has been used to learn the structure of music [58]. Another application was classification of natural language sentences as grammatical or ungrammatical [131].

Hierarchical Recurrent Networks. Another possible approach for learning long-term dependencies was proposed by El Hihi and Bengio [96]. They observed that the problem of vanishing gradients only occurs because long-term dependencies are separated by many time steps. RNNs already utilize the sequential nature of time by using the activities of one time step as input for the next time step.

Hierarchical RNNs are based on the additional assumption that long-term dependencies are robust to small local changes in the timing of events, whereas dependencies spanning short intervals are allowed to be more sensitive to the precise timing of events. This motivates the use of multiresolutional representations of the state information. Long-term context is represented by hidden state variables which are allowed to change very slowly, whereas short-term context is represented by hidden state variables that change faster.

The authors compared the performance of hierarchical and flat recurrent networks for learning tasks involving long-term dependencies. A series of experiments confirmed the advantages of imposing a hierarchical network structure.

The concept of representing time-dependencies at appropriate levels can be applied to the Neural Abstraction Pyramid architecture. It is very similar to the distributed representation of space-dependencies, where short-range dependencies are represented at lower layers and long-range dependencies are represented at higher layers of the network. If the higher layers of the pyramid operate on slower time-scales than the lower layers, they can learn to represent longer-time dependencies. Slowing down higher layers can be done either by less-frequent updates or by the use of larger time-constants for fading memories.

The usefulness of such a time hierarchy has also been confirmed in the field of reactive control of mobile robots [25]. While flat reactive systems face difficulties when required to consider long-term context, a hierarchy of reactive behaviors can provide longer temporal context for lower-level behaviors without large computational costs. Such a hierarchy can handle a high degree of complexity. It was successfully applied to the problem of controlling a team of soccer-playing robots [20].

6.3.4 Random Recurrent Networks with Fading Memories

To avoid the difficulties involved with training recurrent neural networks, recently, the use of random recurrent neural networks was proposed independently by two

groups [148, 109]. Memory traces of an input sequence reverberate in a randomly connected neural network, and the states of this network are mapped by a feed-forward network to the desired outputs.

Echo State Networks. The echo state approach to analyzing and training recurrent neural networks was proposed by Jaeger [109]. He uses discrete-time recurrent networks with large numbers of inhomogeneous units. The units differ in type and time-constants and have random connectivity where the magnitude of gains in loops is smaller than one. Since the network dynamics has a contracting effect on the state, the units implement fading memories. The effect of starting state differences vanishes as the network runs.

The state of the recurrent network can be viewed as a dynamic reservoir of past inputs. This reservoir is accessed by linear read out units. Only the weights of these units are trained to minimize a cost function by approximating desired outputs. This assumes that the desired input-output mapping can be realized as a function of fading memories. Furthermore, since the random recurrent connections of the dynamic reservoir are not trained, it is assumed that the features needed to compute the output will be among the many random features extracted by the units of the reservoir.

Echo state networks have been applied to several non-trivial tasks. They include periodic sequence generators, multistable switches, tunable frequency generators, frequency measurement devices, controllers for nonlinear plants, long short-term memories, dynamical pattern recognizers, and others.

For many of these tasks, feedback from output units to the reservoir was necessary. Since, initially, the outputs do not resemble the desired outputs, the activity of output units was clamped to the target values during training. For testing, the outputs were clamped to the desired outputs during an initial phase. After this phase, the outputs ran free, and the test error was evaluated. When applying such a scheme, one must take care, not to give the network, during the initial phase, information about the outputs desired in the free running phase. Otherwise, the network can learn to store the desired outputs in a delay-line and to replay them for testing.

Liquid State Machine. A similar approach was proposed by Maass *et al.* [148]. It is called liquid state machine (LSM) since a large pool of randomly connected units with contracting dynamics acts like a liquid that reverberates past stimuli. The units used in LSM networks are biologically more realistic. Continuous time is modeled, and the units emit spikes. Furthermore, dynamic synapses and transmission delays resemble properties of biological neurons. The cells of the liquid are chosen as diverse as possible and connected randomly. Feed-forward output networks receive inputs from all units of the liquid. Only these networks are trained to produce desired outputs.

The main focus of the LSM approach is the analysis of the computational power of such networks. It is shown that the inherent transient dynamics of the high-dimensional dynamical system formed by a sufficiently large and heterogeneous neural circuit may serve as a universal analog fading memory. Readout neurons can learn to extract in real-time from the current state of such recurrent neural circuit

information about current and past inputs that may be needed for diverse tasks. Stable internal states are not required for giving a stable output since transient internal states can be transformed by readout neurons into stable target outputs due to the high dimensionality of the dynamical system.

Again, it is assumed that the features needed for the computation of the desired output are already present in the pool of randomly generated features. This is comparable to two existing approaches: first, the liquid could be replaced by exponential delay lines that represent the input history. Second, the use of random connectivity for hidden units is analogous to the classical perceptron [194], where random features were extracted from a retina and only the weights of linear threshold output units were trained with the perceptron learning algorithm to match desired outputs. While such an approach is effective if enough random features are used, when the backpropagation algorithm became available, it turned out that the adaptation of hidden weights allowed solving the same tasks more efficiently with much smaller networks by learning task-specific hidden representations.

6.3.5 Robust Gradient Descent

It was discussed above why supervised training of RNNs is difficult. Fortunately, in the Neural Abstraction Pyramid approach to image interpretation, not all the problems occur at their full scale.

For instance, long-term dependencies are not needed for the interpretation of static images since this task can usually be completed within a few iterations of the network. Hence, the BPTT algorithm can be applied to compute the exact gradient of the cost function, without the need to truncate the history.

Furthermore, the hierarchical network structure facilitates the hierarchical representation of time through the extraction of invariant features. While low-level features change quickly as the input undergoes a transformation, the outputs of higher-level feature cells change more slowly.

Balancing Excitation and Inhibition. The decay/explosion of error flow has been identified as the main problem in training RNNs. If the network is designed such that balanced excitatory and inhibitory effects cancel and as a consequence the network's activity changes slowly, the decay/explosion of the error flow has a long time-constant as well. Hence, it is less harmful.

Balanced effects of excitation and inhibition can be achieved by using transfer functions for inhibitory feature cells that grow faster than the ones of excitatory features. For instance, inhibition could be linear, while excitation saturates for high activities. Such an arrangement stabilizes activities where excitation and inhibition cancel. If the network is too active, inhibition will be stronger than excitation and will lower its activity. On the other hand, if the network is too inactive excitation is far from being saturated and leads to an increase of activity.

Combining BPTT and RPROP. Still, the magnitudes of the backpropagated errors may vary greatly. For this reason, it is very difficult to determine a constant

learning rate for gradient descent that allows for both stable learning and fast convergence.

Since the RPROP algorithm does not use the magnitude of the gradient, it is not affected by very small or very large gradients. Hence, it is advisable to combine this algorithm with BPTT. This training method for RNNs proved experimentally able to avoid the stability problems of fixed-rate gradient descent, while at the same time being one of the most efficient optimization methods.

Learning Attractors. When analyzing static input with a Neural Abstraction Pyramid, the desired network output is usually static as well. Two goals must be combined by the cost function (6.14). First, after T iterations, the final approximation to the desired output should be as close as possible. Second, the network's output should converge as quickly as possible to the desired output.

Hence, it is not sufficient to include only the final approximation error into the cost function. The error weights γ_t for intermediate time steps $t < T$ must be nonzero as well. Depending on the importance of above two goals, the error weights must be chosen appropriately.

Constant weighting of the error components, e.g. $\gamma_t = 1$, does not pay much attention to the final approximation. It may well be that the learning algorithm prefers a coarser approximation if it can be produced faster.

Experiments showed that increasing the error weights linearly, e.g. $\gamma_t = t$ gives the later error components a large enough advantage over the earlier error components, such that the network prefers a longer approximation phase if the final approximation to the desired output is closer.

This effect is even stronger when a quadratic weighting, e.g. $\gamma_t = t^2$, is used, but in this case the network may produce a solution that minimizes the output distance for the last training iteration T at the cost of increasing this distance for later iterations which are not trained.

6.4 Conclusions

This chapter discussed gradient-based techniques for supervised training of feedforward and recurrent neural networks. Several improvements to the basic gradient descent method were covered. Some of these will be used in the remainder of the thesis for supervised training of Neural Abstraction Pyramids.

The RPROP algorithm is used in combination with mini batches to speed up the training. Low-activity priors are employed to enforce sparse representations.

For the case of recurrent pyramids, the BPTT method for computing the gradient is combined with RPROP to ensure stable and fast training, despite large variances in the magnitude of gradients. If the desired output is constant, the weighting of the output error is increased linearly to quickly achieve a good approximation. In this case, attractors are trained to coincide with the desired outputs.

Part II

Applications

7. Recognition of Meter Values

The remainder of the thesis applies the proposed Neural Abstraction Pyramid to several computer vision tasks in order to investigate the performance of this approach.

This chapter deals with the recognition of postage meter values. A feed-forward Neural Abstraction Pyramid is trained in a supervised fashion to solve a pattern recognition task. The network classifies an entire digit block and thus does not need prior digit segmentation. If the block recognition is not confident enough, a second stage tries to recognize single digits, taking into account the block classifier output for a neighboring digit as context. The system is evaluated on a large database.

7.1 Introduction to Meter Value Recognition

Meter stamps are commonly used in many countries to mark letters. They are printed by a postage meter that is part of a mailing machine. The postage meter prints the stamp usually with red ink in the upper right corner of the letter, at the location where otherwise adhesive stamps would be placed. In addition to the postage value the stamp usually contains the identification number of the postage meter, the date and location of sending, and possibly some advertisements of the sender. The sealed postage meter keeps track of the postage used and must be refilled from the postal company when the stored postage has been used. Figure 7.1 shows some historical meter stamps from different countries.

The first meter machines were installed in Scandinavia and the United States of America in the beginning of the 20th century. Figure 7.2 shows the Pitney Bowes Model M postage meter from 1920 which was the first to be licensed by the U. S. Post Office Department. Nowadays, about half the mail in the United States is me-
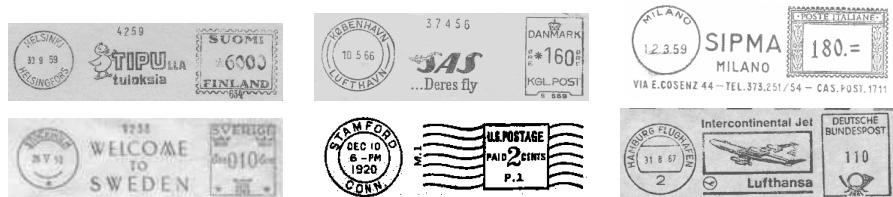
Fig. 7.1. Historical meter stamps from different countries (converted to grayscale).
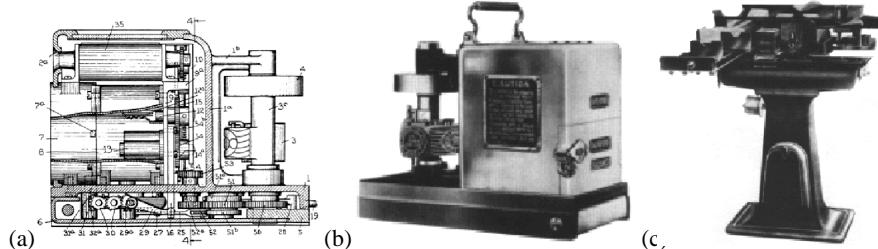
Fig. 7.2. Pitney Bowes Model M postage meter: (a) drawing by inventor Arthur H. Pitney; (b) original postage meter from circa 1920; (c) original Model A mailing machine, which housed the postage meter (pictures adapted from [10]).

tered, representing the largest single source of postage revenue. More than a million postage meters are being used by large and small quantity U. S. mailers in business, industry, and other categories.

During automatic mail processing, not only the letter's address is read for sorting, but also the stamps are recognized. This is necessary in order to compare the postage value of the stamp to the weight of the letter for checking if the postage is sufficient. It is desirable to apply the same check to metered letters.

For a successful recognition of meter values, first, the meter stamps must be detected. Next, the exact location of the meter value must be determined. Finally, the value must be read.

In the following, a system is described that covers only the last step, the actual recognition of the isolated meter value. The system is trainable to recognize the entire meter value, without prior digit segmentation. It is based on the Neural Abstraction Pyramid architecture. If this block classifier cannot make a confident decision, single digit classifiers are combined with its results.

7.2 Swedish Post Database

For the following experiments, a database of Swedish Post meter marks is used which was collected by Siemens ElectroCom Postautomation GmbH. It contains 5,471 examples that were assigned randomly to a training set of size 4,372 and a test set of size 1,099. Figure 7.3 shows some example images from this dataset. As can be seen, the recognition of the meter value is not an easy task. The images are of low resolution and low contrast. Typically, digits have a size of only 10×4 pixels. High variance of print, lighting, and background complicate recognition further. Frequently, the meter values are difficult to read even for humans.

On the other hand, the meter values are not arbitrary combinations of digits, but come from a set of standard postage values. Table 7.1 shows the 16 most frequent values that account for 99.2% of the dataset. The meter values are not uniformly distributed. The five most frequent values cover almost 90% of all examples. Fur-

Fig. 7.3. Some examples of the Swedish Post database (converted to grayscale). While some meter values are clearly readable, others are challenging even for experienced human observers.

Table 7.1. Most frequent meter values of the Swedish Post database. The 16 values shown account for 99.2% of the dataset.

	4.60	4.40	9.20	3.90	4.10	3.80	8.80	7.00	3.50	6.00	8.20	5.00	13.00	12.00	7.80	8.00
#	3176	534	522	341	319	98	91	82	78	64	42	19	18	16	14	14
%	58.1	9.8	9.5	6.2	5.8	1.8	1.7	1.5	1.4	1.2	.77	.35	.33	.29	.26	.26

thermore, one can observe that it suffices to read the two digits next to the point separator in order to uniquely identify the meter value.

In addition to the RGB-image and the meter value, an automatically determined rectangular region is given for each example that should contain the digits belonging to the meter value and nothing else.

7.3 Preprocessing

Before an example can be given to the block recognizer, some preprocessing is needed to make its task easier. The goal of the preprocessing is to reduce the variance of the examples by color filtering and by increasing the image contrast, such that the print becomes black and the background becomes white, and by normalizing slant and position of the meter value.

7.3.1 Filtering

Segmenting the print from the background is not an easy task, since the image quality of the Swedish Post database is quite low. Noise should be discarded, while at the

same time, the lines of meter value should be enhanced to improve readability. The filtering methods described in the following were developed from a low-level model of the print that includes line width, line color, ratio of foreground to background area, and typical noise.

Color Filtering. Because the meter stamps are printed with red ink, it is obvious to use the color as a key to segment the print from the background. The RGB-images were captured with a two-line camera. One sensor line consists of only green pixels, while in the other line blue and red pixels alternate. Thus, the vertical resolution of the green color is twice as high as the resolution for the other two color channels.

Since the lines of the print are only about one pixel wide, frequently they cover only parts of a pixel's sensor area. This leads to color deviations. The red lines then appear to be orange or magenta.

Figure 7.4(a) shows the original rectangular regions of three examples from the Swedish Post database. Parts (b-d) display the three 8-bit RGB-color components of these images. One can observe that the print is best visible in the green component, as this is the complementary color to red. A lower contrast blurred version of the print appears in the blue component. The red component contains almost no differences between the print and the background since the reflectivity for red light of the red color is about as high as the one of the paper.

A pixel-based filter extracts the red colored lines as follows:

$$v = \min(255, \max(0, \ 24 + 2r + 0.125b - 2.25g \,)), \tag{7.1}$$
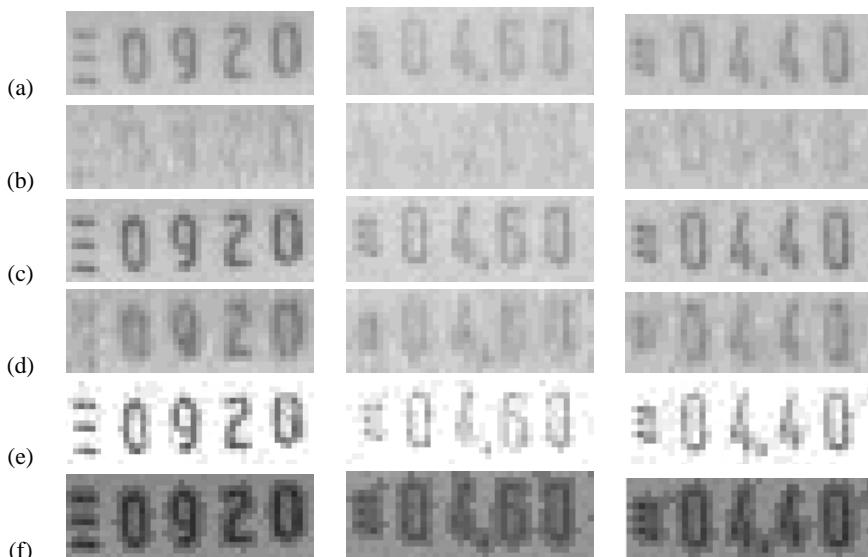
Fig. 7.4. Color filtering: (a) original image (converted to grayscale); (b-d) red, green, and blue components of original image; (e) red color filtered version of the input (shown inverted); (f) output image produced using center-surround filtered green component combined with red neighborhood mask.

where v is the filter output, as shown in Fig. 7.4(e), and r, g, b are the color components. The output of the red-filter is used in the following as a mask to suppress dark lines of other colors.

Figure 7.4(f) shows the result of the color filtering. This image is based on the green component that has been convolved with a $3{\times}3$ center-surround kernel to amplify high image frequencies. If the maximal red-filter response of a pixel's $3{\times}3$-neighborhood does not exceed the average red-filter response by at least a value of eight, then a value of 32 is added to the filter output. Thus, the pixels close to the red lines appear darker than the background.

Contrast Enhancement. The next step of the preprocessing is to enhance the image contrast. This is done by stretching the pixel intensities linearly from an interval $[g_{\min}, g_{\max}]$ to $[0, 224]$. All intensities lower than $g_{\min}$ are set to 0 (black). All intensities larger than $g_{\max}$ are set to 255 (white), the background value.

The lower threshold $g_{\min}$ is determined using the intensity histogram shown in Figure 7.5(a) for the three examples from Fig. 7.4. Because one cannot perceive a clear distinction between the intensities of the print and the background, $g_{\min}$ is chosen as the minimal intensity that has at least 32 darker pixels in the histogram. The upper threshold $g_{\max}$ is initialized similarly by setting it to the minimal intensity value that has $(128 + a/16)$ darker pixels in the histogram, where a is the total number of pixels. It is modified to lie in the closest local minimum of the histogram in order to minimize the segmentation error between the background and the brightest foreground pixels. The described method of choosing the interval borders ensures that, after contrast stretching, some black pixels exist and that most pixels are assigned to the background.

In Figure 7.5(b) the results of this contrast stretching are shown. The readability of the meter values has improved greatly. On the other hand, one can observe that some isolated pixels which correspond to noise were segmented as foreground. Furthermore, some adjacent lines are merged to single foreground blobs. To address these problems, a counter c is computed for the 8-neighborhood of each pixel. It counts the foreground pixels. Dark pixels with an intensity smaller than 128 are counted twice.
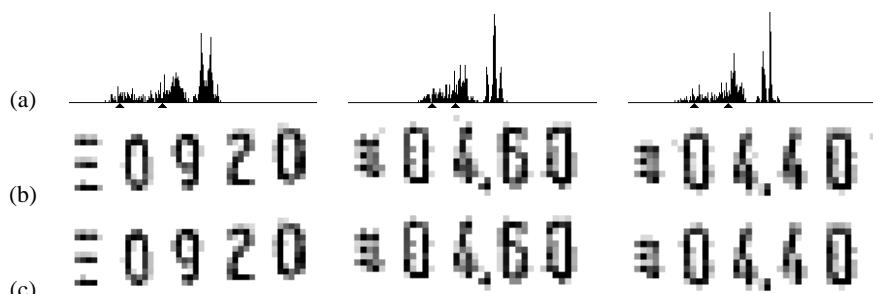
Fig. 7.5. Contrast enhancement. For the examples from Fig. 7.4 are shown: (a) histogram with marked minimal and maximal gray values; (b) contrast stretched; (c) isolated pixels removed and blobs weakened.

A foreground pixel is removed when this counter is small, compared to its intensity. Isolated pixels are always removed. Pixels with a counter of one are removed only when their intensity is greater than 64. Pixels with a counter of two are removed only when their intensity is greater than 128. Similarly, to weaken the inner part of blobs, the intensity of a pixel is increased half the way towards 255 if the counter is large, compared to its intensity.

Figure 7.5(c) shows the resulting images. While the removal of isolated pixels is quite obvious, the weakening of blobs is most visible in the lower parts of the digits four and in the three horizontal lines at the start of the last two blocks.

7.3.2 Normalization

The automatically determined regions of interest containing digits of the meter value have a variable size, while the block classifier is a neural network with a fixed input size. Hence, a mapping must be computed from filtered regions to the input image. This mapping normalizes digit position and slant in order to simplify recognition. If the normalization did not occur, the network would also have to learn translated and slanted variants of the meter values. The size of the digit block is not normalized because the image resolution is so low that an arbitrary scaling would produce significant blur. In addition, it would require a reliable segmentation of the digits from other objects, e.g. from the curved line delimiting the meter mark that is sometimes included in the region of interest.

Slant Normalization. Due to misalignments of letters relative to the stamp during metering, and relative to the camera during capture, some meter values appear slanted in the rectangular region of interest. The slant normalization step estimates this slant and corrects for it.

To estimate the slant, the center of mass of the dark foreground is computed in the first step. It divides the image into a left and a right part. The centers of mass of these two parts are computed next. They define a line that corresponds to the slant estimate. Figure 7.6(a) illustrates this for three examples.

To correct for the slant, a vertical sheer transformation is used that makes the line approximately horizontal. The sheering keeps the position of the center constant and shifts columns only by an integer number of pixels to avoid blurring. Figure 7.6(b) shows the resulting deslanted images for the three examples. Note that this normalization has no effect if the estimated slant is relatively small.
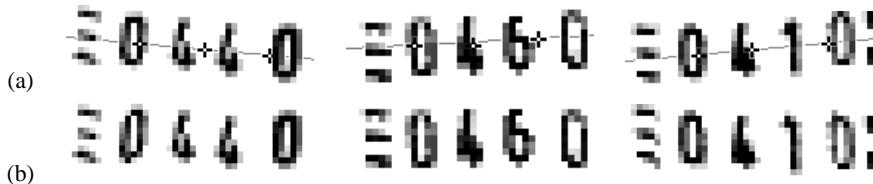
Fig. 7.6. Slant normalization: (a) contrast enhanced image of slanted examples with markers at the center of gravity, as well as at the left and the right center; (b) result of vertical sheering that removes the slant.

Position Normalization. The block recognizer, described below, has an input size of 32×16. A window of this size must be cut from the region of interest. Since the block recognizer has to read the two digits directly to the left and to the right of the point or space delimiting kronor from öre, the window is placed such that the gap between these two digits is centered horizontally. Figure 7.7(b) illustrates how the space between the second and the third digit from the right is found using a smoothed occupancy index that is computed for each column.

The occupancy index is the sum of two components: the column's foreground sum and its top index, as shown in Fig. 7.7(a). To determine the top index, a column's foreground values are accumulated, starting from the topmost row, until the sum exceeds 128. The top index is proportional to the height of this first occurrence of significantly dark foreground decreased by three. This accounts for the delimiting point that is usually located in the lower three rows, which should not cause high top indices.

The occupancy index is smoothed with a binomial kernel of size 9 to reduce the number of local minima. Each digit should now correspond to a single maximum while each space between digits should correspond to a single local minimum. Starting from the right, the beginning of the digits is localized in the occupancy vector. The search then proceeds to find the first and the second local minimum between the digits (see marked columns in the figure).

(a)

(b)

(c)

(d)

Fig. 7.7. Position normalization. For the examples from Fig. 7.5 are shown: (a) top index (upwards) and column sum (downwards); (b) region of interest with smoothed occupancy index (the start column at the right, as well as two local minima and the vertical center of mass are marked); (c) 32×16 window cut from the region of interest; (d) window with faded borders.
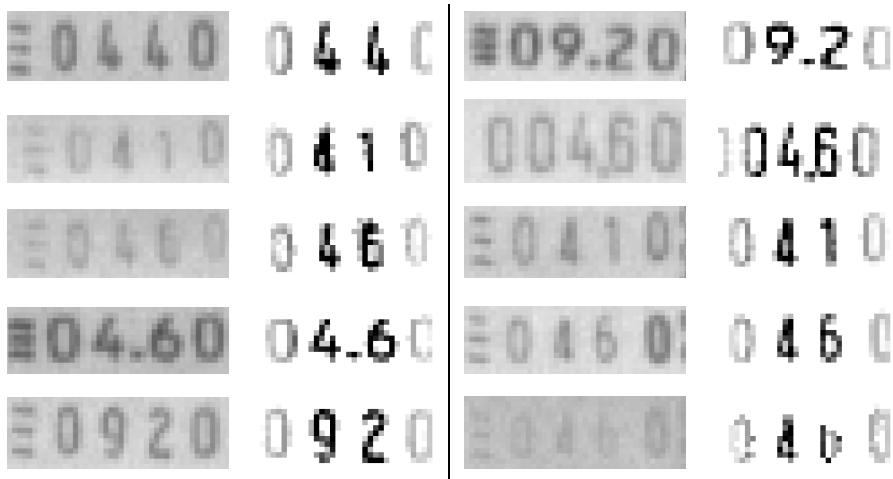
Fig. 7.8. Preprocessing of meter values. For ten randomly selected examples of the Swedish Post database, the original region of interest (converted to grayscale) and the result of the preprocessing are shown.

The second minimum is used to center the 32×16 window horizontally. Its vertical center is chosen to be the vertical component of the center of foreground mass, indicated by the horizontal lines in Figure 7.7(b) for three examples.

Figure 7.7(c) shows the windows cut from the three examples. The two digits of interest have been centered successfully. Because the other digits, as well as the pixels near the upper and the lower window edge, are less important for recognition, image contrast is faded towards the borders of the window. This also reduces border effects in the block recognizer.

Figure 7.7(d) displays the final result of the preprocessing for the three examples. In Figure 7.8 the original regions of interest and the result of the preprocessing are shown for ten randomly selected examples. The two digits of interest are quite salient, centered, and most of them are readable.

7.4 Block Classification

The task of the block classifier is to recognize a meter value from a preprocessed image. Although the preprocessing discarded some of the variances present in the examples and increased their readability, the problem is still challenging.

As can be seen in Figure 7.8, the print varies considerably. The digits come in different sizes and different fonts with varying spaces between them. Some examples contain a delimiting point, while others do not. Some of the loops enclose background pixels, while others only have some brighter foreground pixels in the center. Furthermore, noise is still present in the images.

Fig. 7.9. Problematic preprocessed examples from the Swedish Post database. In some examples the digit segmentation is difficult, while in others the recognition of isolated digits is hard.

Layer 0 (32x16x1)   Layer 1 (16x8x4)   Layer 2 (8x4x16)   Layer 3 (4x2x32)
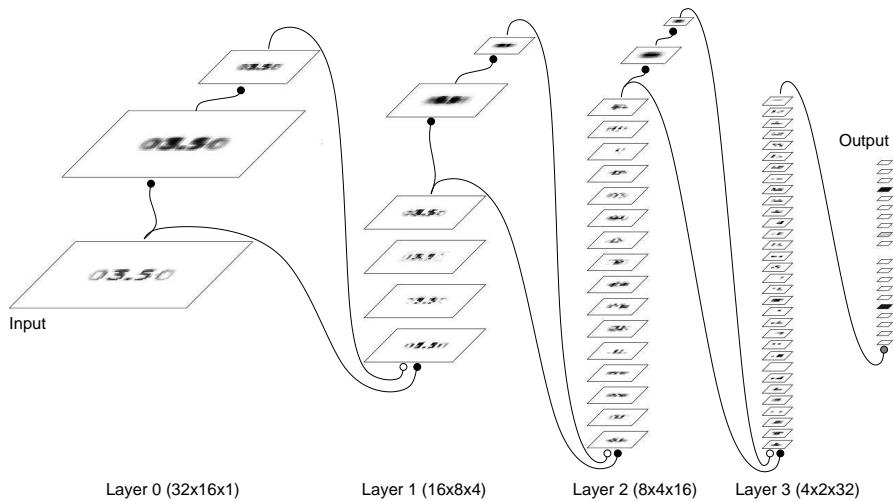
Fig. 7.10. Network architecture for meter value recognition. It is a feed-forward version of the Neural Abstraction Pyramid with specific excitation and unspecific inhibition in the inner layers. The activities of the trained network for a test example are shown. The output feature cells signal the classes of the two digits of interest in a $2\times$(1-out-of-10) code ('3' and '5' for a meter value 3.50).

One could now try to segment the digit block into single digits, recognize them, and combine the digit classifier outputs to a meter value. This approach would require reliable digit segmentation and a reliable digit classification system. Both requirements are not easy to meet. It is fairly hard to segment the digits, and it is also difficult to read isolated digits reliably, as is evident from Figure 7.9 which shows some problematic preprocessed meter values.

For these reasons, a block classifier was developed that recognizes the two digits of interest simultaneously within the context of the neighboring digits. Unlike a digit classifier that can only use the a-priori distribution of single digits, this classifier is able to take advantage of the non-uniform meter value distribution, summarized in Table 7.1.

7.4.1 Network Architecture and Training

The architecture of the Neural Abstraction Pyramid network used for the recognition of entire meter values is sketched in Figure 7.10. It is a feed-forward network consisting of five layers.

Layer 0 at the bottom of the hierarchy has a resolution of 32×16. It only contains the input feature array. The resolution of the feature arrays decreases from layer to layer by a factor of two in both dimensions, until Layer 3 reaches a size of only 4×2 hypercolumns. Similarly, the width of the border that is set to zero decreases from 16 to 2. At the same time, the number of excitatory features rises from 4 in Layer 1, to 16 in Layer 2, and to 32 in Layer 3.

The network contains 20 output feature cells in the topmost layer which encode the meter value. The output code used is composed of two sections that indicate the identity of the two digits of interest in a 1-out-of-10 code.

The projections of output feature cells receive their inputs directly from all positions of all feature arrays of Layer 3. Their weights are allowed to change sign. The potential of these projections is passed through a sigmoidal transfer function f_{sig} ($\beta = 1$, see Fig. 4.5(a) in Section 4.2.4), which saturates at zero and one.

In contrast, the cells of excitatory features located in Layer 1 to Layer 3 are driven by specific excitation and unspecific inhibition. The weights of their specific excitatory projections originate from overlapping 4×4 windows of the feature arrays in the layer below them. Unspecific inhibitory projections have a single weight to the smoothed and subsampled sum of these features. Both projections have linear transfer functions. The transfer function $f_{\mathsf{p\_sig}}$ ($\beta = 2$, see Fig. 4.6(b)), which is used for the output units is a rectifying function that saturates at activities of one. This ensures that the network learns sparse representations of the digit block since the activity becomes exactly zero if inhibition exceeds excitation.

The feature sums and their subsampled versions, needed for the unspecific inhibition, are computed as described in Section 5.2.1. The network is initialized using the unsupervised learning of sparse features, described in Chapter 5. Supervised training is done with gradient descent on the squared output error until the performance on the test set does not improve any more.

The training enforces the desired signs of the weights. If a specific excitatory weight becomes negative, it is set to zero, and the unspecific inhibitory weight is changed instead. This leads to sparse excitatory weights since, after training, many of them have a value of exactly zero and can be pruned away without loss.

7.4.2 Experimental Results

The trained Neural Abstraction Pyramid network is able to perform the recognition task quite well. After deleting 21 examples from the training set that could not be centered successfully or that were not readable for an experienced human observer, there are only 11 substitutions left. All but one of them can be rejected easily.

The test set has not been modified. In Figure 7.11 some test examples are shown that are difficult, but were recognized successfully, along with some examples for
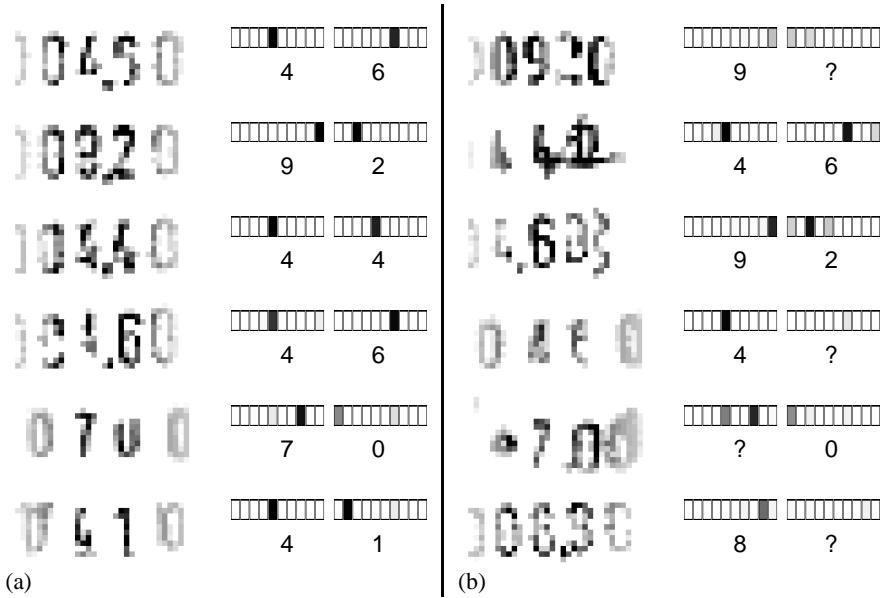
Fig. 7.11. Block recognition examples from the test set. The inputs to the hierarchical block classifier, its outputs, and the interpretation of the output are shown for: (a) examples that were recognized successfully; (b) examples for which recognition failed.

which recognition failed. Analysis of the problematic examples reveals that recognition failure is mainly caused by missing image parts or by failure to center the digits of interest during preprocessing. Such errors in digit segmentation occur more often when the region of interest includes foreground structures in addition to the digits, as in the example in the second and the third row of the figure.

One can observe that for most ambiguous examples, the network is able to indicate its uncertainty by producing outputs that deviate from the desired 1-out-of-10 pattern. This makes it possible to compute a meaningful classification confidence as follows. For both digits, the difference between the maximal output activity and the second largest one is taken as confidence. Since the actives belong to the interval [0,1], the digit confidence has a value of one when a single output is one and all other outputs have zero activity. If more than one output has high activity or all output activities are low, the digit confidence has a low value.

Both digit confidences are combined into a block confidence by taking the average. If one digit confidence is below a reject parameter ρ, or the block confidence is below the block reject parameter $\hat{\rho} = 1 - (1 - \rho)^2$ which is more strict, the example is rejected.

Figure 7.12 summarizes the test performance of the hierarchical block classifier. About 2% of the examples are substituted when the reject parameter $\rho = 0$ is used and all outputs are accepted. By rejecting 2.4% of the examples, half of the substitutions can be avoided. To reduce the substitution rate further, a larger fraction of the examples must be rejected.

ρ	recognized	substituted	rejected
0	1077 (98.0%)	22 (2.0%)	0
0.25	1061 (96.5%)	11 (1.0%)	27 (2.4%)
0.62	1023 (93.1%)	6 (0.54%)	70 (6.4%)
0.70	979 (89.1%)	2 (0.18%)	118 (10.7%)
0.73	927 (84.3%)	1 (0.09%)	171 (15.6%)
0.87	816 (74.2%)	0	283 (25.8%)

(a)   (b)
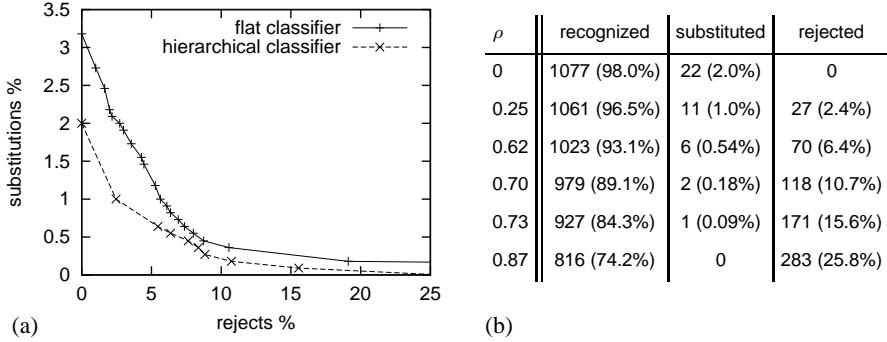
Fig. 7.12. Performance of the meter value block classifier on the test set: (a) flat classifier compared to hierarchical classifier; (b) recognition as a function of the reject parameter ρ for the hierarchical classifier.

For comparison, several fully connected three-layered feed-forward neural networks with sigmoidal activation functions were trained on the same data. The networks had 16, 32, 64, 128, or 256 hidden units. They were trained using gradient descent on the squared output error until the test set performance did not improve any more. The recognition performance of the best flat network, which had 32 hidden units, is also shown in Figure 7.12(a). It substitutes 35 (3.18%) of the 1,099 test examples in the zero-reject case. For higher reject rates the flat network is outperformed by the hierarchical network as well.

Since the rejections necessary for reliable recognition reduce the acceptance rate of the classifier, the next section describes a second recognition system that tries to verify the examples rejected by the hierarchical block classifier.

7.5 Digit Recognition

Because the block recognition system described in the previous section is not a perfect classifier, it is complemented by a digit recognition system as illustrated in Figure 7.13. A separate digit classifier is used for the left and the right digit of interest since they have different a-priori class distributions and are embedded in different context. Both digit classifiers receive the output of the block classifier for the other digit as contextual input in addition to the preprocessed digit.

The digit recognizers are queried only if the block classifier is not confident enough and rejects an example. Digit recognition consists of three steps: digit preprocessing, digit classification, and combination of the digit outputs with the results of the block classifier.

7.5.1 Digit Preprocessing

The image of the preprocessed meter value cannot be given directly to the digit classifier. Some digit-specific preprocessing is necessary to facilitate recognition.
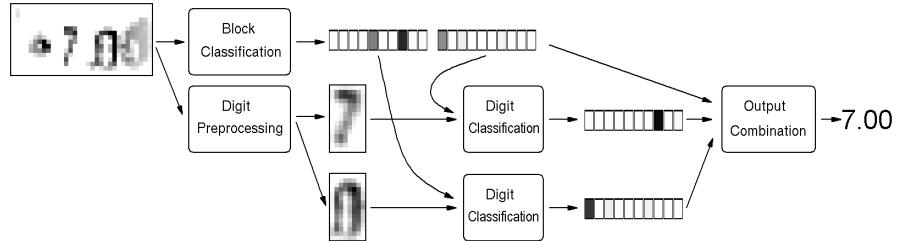
Fig. 7.13. Sketch of the combined meter value recognition system. Digit recognition is only necessary if the block classifier rejects an example.

The digit needs to be segmented from the other digits, and it is normalized to a fixed size.

Segmentation. The goal of the segmentation step is to determine a minimal rectangular region of the 32×16 meter value image that contains the digit to be recognized and nothing else. This is illustrated in Figure 7.14(a) for some examples.

The vertical extension of this rectangle is determined as follows. Starting from the topmost row and the bottom row, the rectangle's borders are moved towards the center, until the row sum of foreground intensity exceeds a threshold. The threshold used is the maximal row sum, divided by 16. Thus, all rows that contain non-negligible foreground pixels are contained between the upper and the lower border, as can be seen in the figure.

Horizontal segmentation is done using an occupancy index that is computed similarly to the one used to center the digits of interest horizontally in the 32×16 window (see Section 7.3.2). Here, analysis is done only within the segmented rows. The occupancy index is smoothed with a smaller binomial kernel of length five to keep more local extrema. The smoothed occupancy index is shown above the examples in Fig. 7.14(a).

To locate a digit horizontally, a local maximum is searched for in the occupancy index array, starting with an offset of four pixels from the center of the digit block. The maximum indicates the digit's center. It is marked by a short vertical line in the figure. The left and the right borders of the digit are searched for, starting with an offset of two pixels from the digit center. The borders are moved away from the center, until a local minimum is found, or the occupancy index falls below a threshold. The threshold used is the sum of the occupancy indices at the digit centers divided by 16. If there are no foreground pixels in that row and the distance to the center exceeds two, the border is moved one column back towards the center. Hence, the segmented digit has a width of at least five pixels. In Figure 7.14(a) the horizontal digit segmentation is indicated by vertical lines.

Size Normalization. The segmented rectangular region of a digit has a variable size, but the digit classifier expects an input image of fixed size. Hence, the digit needs to be scaled to normalize its size. This discards the digit's size variance. An array of 8×15 pixels is used to represent the normalized digit.
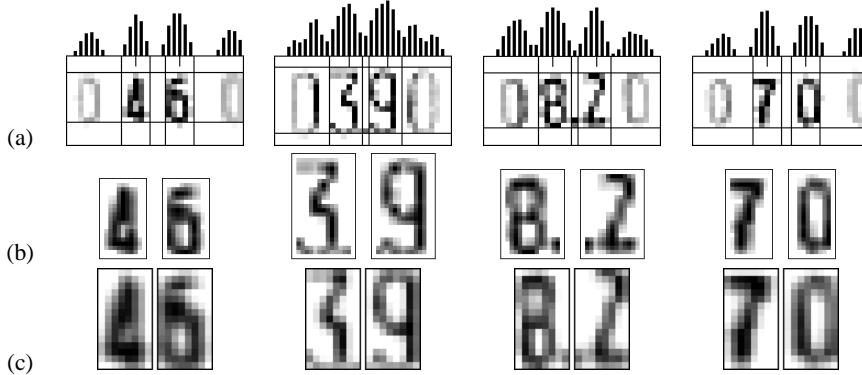
Fig. 7.14. Segmentation and size normalization of meter value digits: (a) digit block with occupancy index and segmentation of the left and the right digit of interest; (b) segmented regions interpolated to higher resolution; (c) digits normalized to 8×15 pixels.

Because the segmented region usually contains fewer pixels than the normalized digit, the resolution of the image is doubled in a first step by inserting interpolated rows and columns. The value v of an interpolated pixel is computed from the values v_1 and v_2 of its neighbors as follows: $v = (v_1 + v_2 + \max(v_1, v_2))/3$. This makes pixels next to the dark lines darker than simple averaging. The higher resolution variants of the digits are shown in Figure 7.14(b). The values of the normalized digits are set now to the response of a 3×3 binomial filter at the corresponding position in the high resolution image.

Finally, the contrast of the normalized image is increased slightly by multiplying the intensities with 1.25, subtracting one fourth of the average pixel value, and clipping the values to the interval $[0, 1]$. This darkens the lines and sets the background pixels to exactly zero (white). Figure 7.14(c) shows the digits after normalization and contrast enhancement. Preprocessing worked well for these examples because the digits were tightly framed by the segmentation and were clearly readable.

Some more problematic examples are shown in Figure 7.15. Part (a) of the figure contains examples that were segmented successfully, although segmentation was not easy. Note that even if a digit is broken into parts, these parts are grouped together and placed at the correct position in the normalized digit image. Part (b) shows some examples for which segmentation failed to select the two digits of interest. This may have been caused by additional foreground structures, as in the example in the second row. Another problem is the miscentering of meter values that leads to the selection of the wrong digit, as in the first and the third row of the figure.

7.5.2 Digit Classification

A three-layered feed-forward neural network is used for digit classification. It is sketched in Figure 7.16. The network's input layer contains the normalized digit to recognize. In the second layer, 32 hidden units detect digit features. They are fully
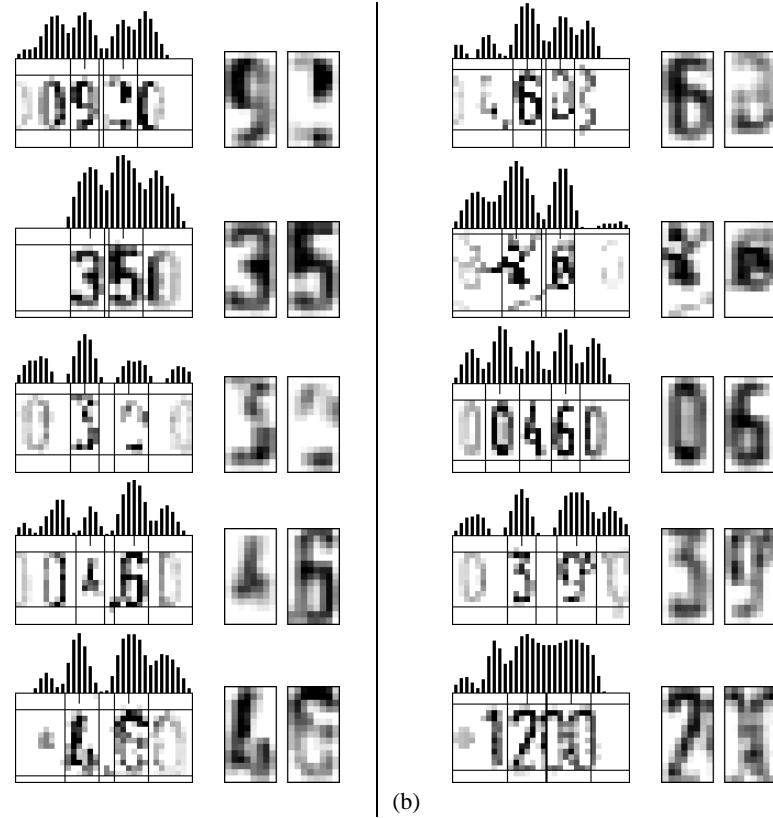
Fig. 7.15. Digit preprocessing for problematic meter value examples: (a) successful digit segmentation; (b) segmentation failure.

connected to the 120 input pixels. Ten context units, which are set to the outputs of the block classifier for the other digit, are also located in the second layer. The third layer consists of 10 output units that are fully connected to these context units as well as to the hidden units. They signal the digit's class in a 1-out-of-10 encoding. The hidden units and the outputs are $\sum$-units that compute a weighted sum of their inputs, followed by a sigmoidal transfer function.

The network is trained to produce the desired outputs using gradient descent on the squared output error. Training is done until the performance on the test set does not improve any more. A separate classifier is trained for the left and the right digit.

After training, the performance on the training set is almost perfect. While all left digits of the 4,351 training examples can be recognized, only two of the right digits are substituted. They can be rejected easily using the difference between the activities of the most active and the second most active output as confidence.
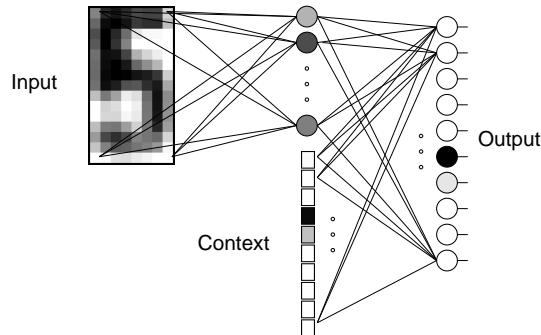
Fig. 7.16. Sketch of the digit classifier network. The network has access to the output of the block classifier for the other digit as context to the normalized digit.

From the test set 4 (0.36%) of the left digit examples are substituted when none are rejected. To achieve a substitution rate of 0.25% of the accepted left digits, the acceptance rate must be lowered to 99.64%.

15 (1.36%) of the right test digits are substituted when all examples are accepted. Lowering the acceptance rate to 98.52% reduces the substitution rate to 0.25%. The right digit is obviously more difficult to recognize than the left one since there is a greater variance in the distribution of right digit labels (see Table 7.1).

Figure 7.17 shows some example inputs and outputs of the right-digit classifier. In Part (a) of the figure, some examples are shown that are not easy to recognize but

(a)   (b)
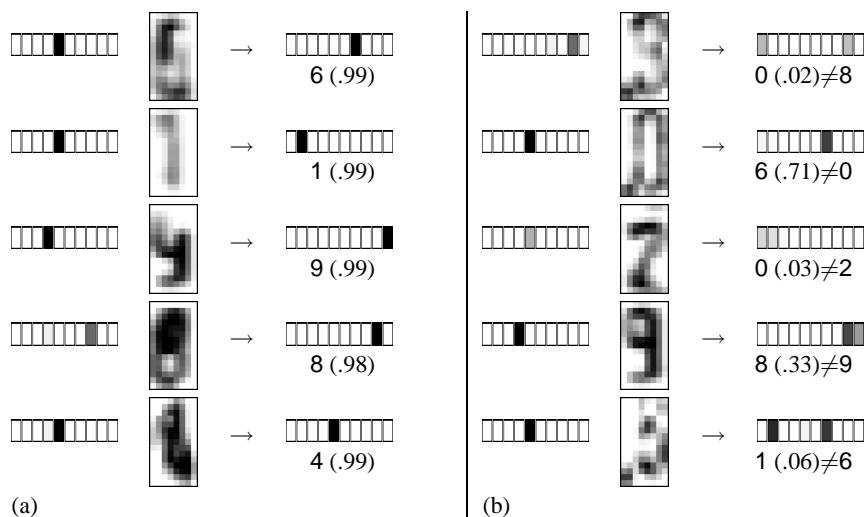
Fig. 7.17. Digit classification for problematic right digit examples. The output of the block classifier for the left digit, the normalized right, and the digit classifier output are shown. The index of the most active output and the confidence label the output vector. (a) successful digit recognition; (b) recognition failure with desired class.

are recognized with high confidence. Part (b) shows examples where recognition failed. Segmentation problems seem to be the most frequent reason for substitutions. If the digit is not tightly framed and some additional foreground structure is present in the normalized image, it deviates from the typical appearance and is therefore difficult to recognize. Missing digit parts also complicate recognition. Unusual context may as well cause substitutions, as in the example in the second row of the figure. Here, the left digit has been correctly recognized as 4 by the block recognizer, but the right digit 0 occurs only very rarely next to a 4 in the dataset. The digits 6, 4, and 1 are much more common in this context. Consequently, the digit is recognized as 6 with medium confidence. While in this particular example, the use of the context information does not seem to be beneficial, in general it facilitates recognition, as can be concluded from the following control experiment.

The same network was trained with a context vector that was set to zero. Without access to the context information, the classification performance degrades. The best test performance for the left digit has now a substitution rate of 1.91%. The best right digit classifier substitutes even 7.25% of the test images when all examples are accepted. These figures show the importance of context for the recognition of isolated digits.

7.5.3 Combination with Block Recognition

Digit recognition is not done for all examples, but only if the block recognizer is not confident enough. If its classification confidence for one of the digits is below a threshold ρ, this digit is preprocessed and presented to the digit classifier. When $\rho = 0.9$ is chosen, 134 (12.2%) blocks are rejected from the 1,099 test examples. 82 (61%) left digits and 101 (75%) right digits are ambiguous.

The outputs of the digit recognizer need to be combined with the ones of the block classifier. This is done by computing the average output $\mathbf{v}_c = (\mathbf{v}_b + \mathbf{v}_d)/2$, where $\mathbf{v}_d$ denotes the output vector of the digit classifier and $\mathbf{v}_b$ is the corresponding section of the block classifier output. The digit's confidence is again set to the difference between the most active and the second most active combined output. It does not exceed the higher one of the two digit confidences.

Figure 7.18 illustrates some typical cases of output combination. If both classifiers are confident and agree on the class, the combined output is confident. If both classifiers disagree on the class, the output is not confident. If one classifier is silent,
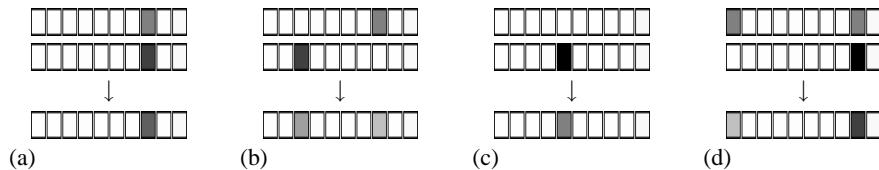
Fig. 7.18. Combination of outputs from block classifier and digit classifier: (a) both classifiers agree; (b) both classifiers disagree; (c) one classifier is inactive, while the other is confident; (d) one classifier is undecided between two classes, while the other is confident.

then the other classifier determines the output. If one classifier is undecided between two classes, the output of the other one can strengthen one of the classes and weaken the other class.

The performance of the combined classifier on the training set is almost perfect. Only one of the 4,351 examples is not recognized. It has a very rare meter value of 4.80 that is substituted for the most frequent value 4.60.

The test set performance is also good. If no examples are rejected, only 10 (0.91%) of the 1,099 examples are substituted. Most substitutions can be rejected easily by using as block confidence the minimum of the left and right digit confidences. The substitution rate can be lowered to 0.45% if only 0.64% of the examples are rejected. Rejecting 3.55% of the examples reduces the substitution rate to 0.18%.

Figure 7.19 summarizes the test set performance of the combined classifier and compares it to the performance of the block classifier alone. Adding the digit classification stage to verify the examples rejected by the block classifier improved the recognition performance significantly.

Figure 7.20 illustrates the combined recognition for some problematic examples. Part (b) of the figure shows the five substitutions from the test set that are most difficult to reject. One reason for these substitutions is that the meter value is not present in the training set or is very rare, as in the examples in the first and the last row. Other substitutions arise from failures to center the two digits of interest during preprocessing. These failures may be caused by some additional strokes, as in the third row, or by a missing digit, as in the fourth row of the figure. Finally, low image contrast may also be a reason for substitutions, as in the example in the second row.

On the other hand, Part (a) of the figure contains some examples for which the person labeling the meter values did not assign a valid label. Although these examples are fairly hard to read, they were successfully recognized by the combined classifier.

ρ	recognized	substituted	rejected
0	1089 (99.1%)	10 (0.91%)	0
0.14	1087 (98.9%)	5 (0.45%)	7 (0.64%)
0.48	1059 (96.4%)	3 (0.27%)	37 (3.37%)
0.52	1058 (96.3%)	2 (0.18%)	39 (3.55%)
0.93	955 (86.9%)	1 (0.09%)	143 (13.0%)
0.97	864 (78.6%)	0	235 (21.4%)

(a)   (b)
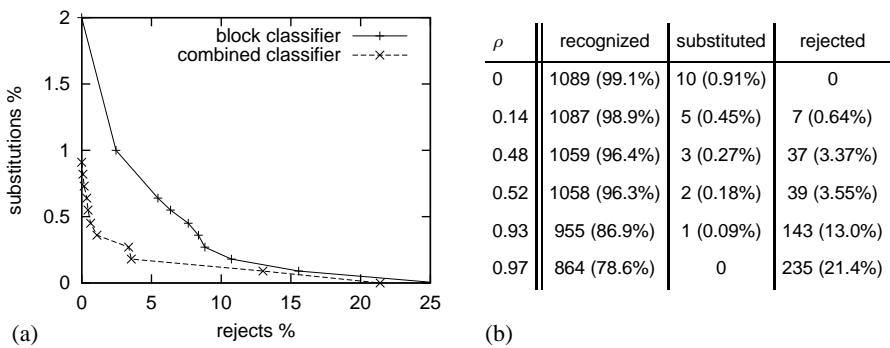
Fig. 7.19. Performance of the combined meter value classifier on the test set: (a) substitutions vs. rejects; (b) recognition as a function of the reject parameter ρ.
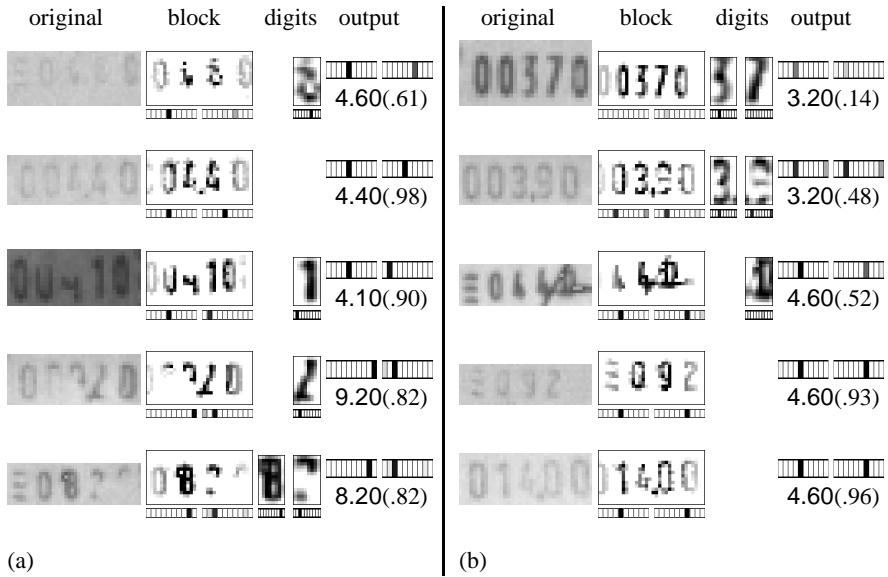
Fig. 7.20. Problematic examples for combined meter value classifier. The original region of interest (converted to grayscale), the preprocessed block along with the block classifier output, the preprocessed digits for which digit recognition is queried along with the digit classification, and the combined classifier output along with the recognized label and the confidence are shown for: (a) some examples for which the person labeling the meter values did not assign a valid label; (b) the five substitutions of the test set that are most difficult to reject.

7.6 Conclusions

This chapter described a meter value recognition system that the author developed in close cooperation with Siemens ElectroCom Postautomation GmbH. The system consists of two stages: block recognition and digit recognition.

The block classifier, which is based on the Neural Abstraction Pyramid architecture, recognizes two digits of interest simultaneously within their context. It performs significantly better than a flat neural classifier.

If block recognition cannot make a confident decision, the system focuses its attention on the ambiguous digits by presenting them to a digit classifier. This digit classifier has access to the block classification output for the other digit as context.

The system was evaluated using a database of Swedish Post meter values. The combined system performs well. If the given region of interest contains a readable meter value, it can be recognized with high accuracy. Even meter values challenging for trained humans can be read.

The analysis of problematic test examples revealed that the recognition performance could be improved further if the training set were larger, such that rare labels were better represented, and if the region of interest contained only the digits of the meter value and no additional objects.

8. Binarization of Matrix Codes

In this chapter, the binarization of matrix codes is investigated as an application of supervised learning of image processing tasks using a recurrent version of the Neural Abstraction Pyramid.

The desired network output is computed using an adaptive thresholding method for images of high contrast. The network is trained to iteratively produce it even when the contrast is lowered and typical noise is added to the input.

8.1 Introduction to Two-Dimensional Codes

Two-dimensional codes are an extension to one-dimensional barcodes, which have been used for many years to mark items with machine readable numbers. In one-dimensional codes the bits are represented by vertical black or white bars of variable width. Figure 8.1(a) shows Code 39 [5], an example of a two-width code.

The first truly two-dimensional bar code was developed by David Allais at Intermec Corporation in 1987 for space applications. As can be seen in Fig. 8.1(b), Code 49 [6] is a stacked barcode with multiple rows. Parity bits, as well as check characters at the end of a line and the end of the code ensure reliable decoding. At most 49 characters can be stored in one symbol.

Figure 8.1(c) shows an example of the Data Matrix [4] code, developed in the late 1980s by International Data Matrix (USA). Data Matrix is a two-dimensional matrix symbology containing dark and light square data modules. It has a finder pattern of two solid lines and two alternating dark and light lines on the perimeter of the symbol. Larger codes contain additional finder patterns within the symbol.

(a)   (b)   (c)
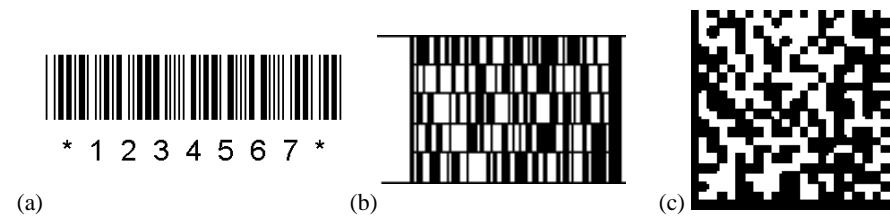
Fig. 8.1. Different codes: (a) Code 39: one-dimensional two-width barcode; (b) Code 49: stacked barcode; (c) Data Matrix: matrix code.

Fig. 8.2. Different meter marks: (a) Meter impression design used by Canada Post; (b) StampIt design used by Deutsche Post AG.

Data Matrix is designed with a fixed level of error correction capability using the Reed-Solomon [188] method. Reconstruction of the content is possible if less than one quarter of the bits have been destroyed. Data Matrix supports industry standard escape sequences to define international code pages and special encoding schemes. It is used for small item marking applications using a wide variety of printing and marking technologies. The code size is variable. Up to 2,334 ASCII-characters can be stored in one symbol.

8.2 Canada Post Database

For the experiments described below, a variant of the Data Matrix code is used that was selected for the electronic letter franking method which is offered by Canada Post as an alternative to postage meter machines and stamps. Figure 8.2 shows an example of such a meter mark, along with a StampIt meter mark used by Deutsche Post AG. Both contain human-readable fields, as well as a Data Matrix symbol.

The meter mark can be printed either directly on the letter, such that it is visible through the address window in the envelope, or it can be printed on the envelope at the upper right corner where the stamp would be placed otherwise. In this case, it is printed using fluorescent red ink to allow for automatic up-right placement of letters. Some examples of the address window variant are shown in Figure 8.3(a). They have a relatively high image contrast. The symbol consists of four quadrants with 22×22 bits each, framed by black or alternating finder patterns. This allows for the storage of 1,936 raw bits. Figure 8.3(b) shows example images from the red ink variant of Canada Post meter marks. Here, each quadrant consists of only 18×18 bits. These images are considerably brighter and have a much lower contrast compared to the address window code variant.

The code matrix contains information about:

- the meter value along with cryptographic key to ensure validity,
- the date of sending,
- the sender, such as the serial number of the meter machine, and
- the addressee, such as the zip code and a short form of the address.

Automatic reading of the Data Matrix code requires localizing and binarizing the symbol, locating finder patterns, reading the bits, correcting for errors, and validating the result.

Here, the focus is on the binarization step only. Because of noise, difficult lighting, printing errors, and the low-contrast of red ink on dark paper, this problem is
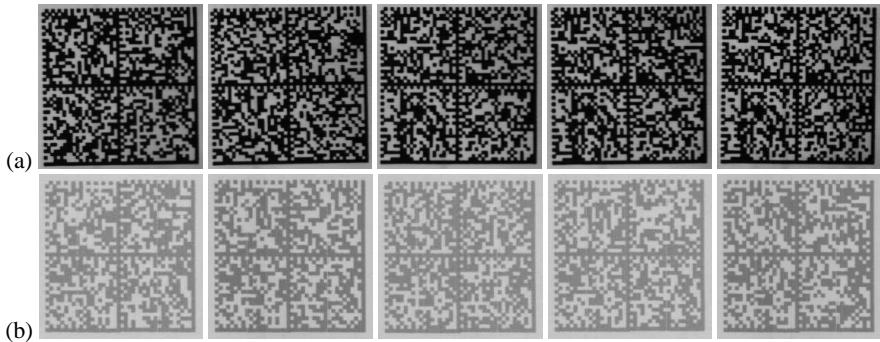
Fig. 8.3. Canada Post Data Matrix original images: (a) high-contrast address window variant; (b) low-contrast red ink variant.

challenging. The purpose of the experiments is to demonstrate that the structure present in the image of the code can be learned and used for binarization.

A database for training and testing the binarization network was provided by Siemens ElectroCom Postautomation GmbH. It consists of 1,209 gray-scale images of size 216×216 containing a Data Matrix each. 515 of the images belong to the high-contrast address window variant (see Fig. 8.3(a)), and 694 low-contrast examples have been printed with red ink (see Fig. 8.3(b)). The high-contrast images can be binarized using simple thresholding methods, while the binarization of the low-contrast images is more difficult.

8.3 Adaptive Threshold Binarization

So far, simple global threshold techniques have been used to binarize the Data Matrix images. However, due to non-uniform lighting for some images it is difficult to determine a single global threshold that separates black pixels from white pixels. One such example can be seen in Figure 8.4(a), where the background in the upper right corner is much darker than in the rest of the image. To correct for this, the intensity of the background $B(i,j)$ is estimated for each location (i,j) by computing

$$B(i,j) = \max_{\|(\Delta_i, \Delta_j)\| \le r} \left[I(i + \Delta_i, j + \Delta_j) - \|(\Delta_i, \Delta_j)\| \right], \qquad (8.1)$$

where $I(i,j)$ is the intensity of the original image, and radius $r = 15$ determines the smoothness of the estimate. Figure 8.4(b) shows the estimated background intensity for the example.

The estimated background is used to correct the image intensity:

$$C(i,j) = \min(255, \max(0,\ I(i,j) + 128 - B(i,j)/2\,)). \qquad (8.2)$$

The corrected example image $C(i,j)$ is shown in Fig. 8.4(c).
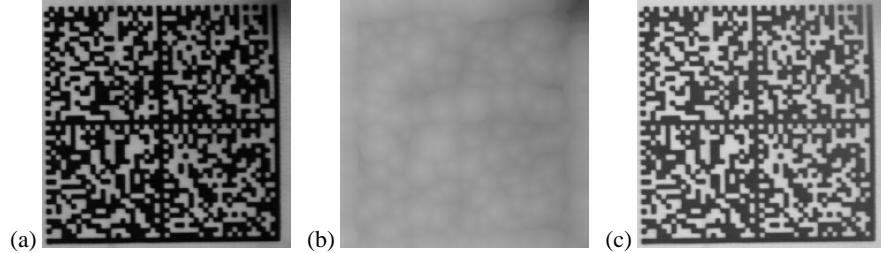
Fig. 8.4. Background correction: (a) original image with non-uniform lighting; (b) estimated background intensity; (c) corrected image.

(a) I_f I_t I_b I_f I_t I_b (b)
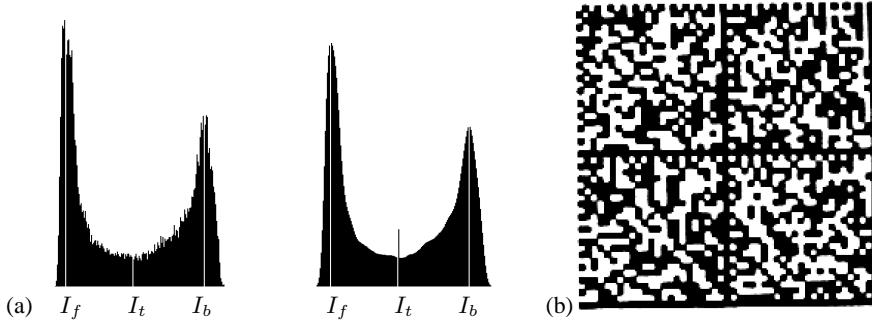
Fig. 8.5. Threshold estimation: (a) histogram of background corrected image and smoothed histogram with estimated threshold, as well as background and foreground intensity; (b) contrast enhanced image.

Figure 8.5(a) shows the intensity histogram of the corrected image, as well as a smoothed version of it. Smoothing was done by repeatedly applying a binomial $1/4$ (1 2 1) kernel until the number of local minima reduced to one. The index of the remaining local minimum is now used as a threshold I_t. The two local maxima represent the intensities of the foreground I_f and the background I_b. Both are used to determine a range $I_t \pm u$ where the contrast is stretched linearly:

$$
\begin{aligned}
u &= (I_b - I_f)/10, \\
w &= \min(255, I_t + u), \\
b &= \max(0, I_t - u), \\
S(i,j) &= \begin{cases} 0 & : \quad C(i,j) < b \\ 255 & : \quad C(i,j) > w \\ (C(i,j) - b) * 255/(w - b) & : \quad \text{else} \end{cases} \quad (8.3)
\end{aligned}
$$

The resulting image $S(i,j)$ is shown in Figure 8.5(b). As can be seen, most pixels are either black or white, but some pixels at borders between cells have been assigned intermediate gray values, representing uncertainty.

Figure 8.6 shows the contrast stretched versions of the images from Fig. 8.3. In general, these images are a good approximation to the desired output, a black and
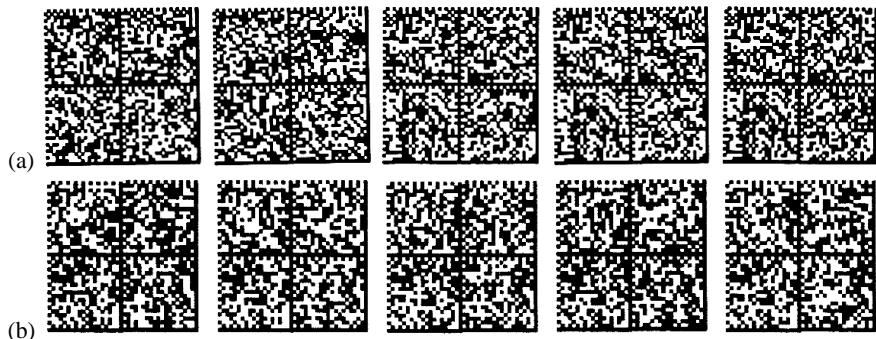
Fig. 8.6. Contrast stretched images (originals in Fig. 8.3): (a) from high-contrast address window variant; (b) from low-contrast red ink variant of the Data Matrix codes.
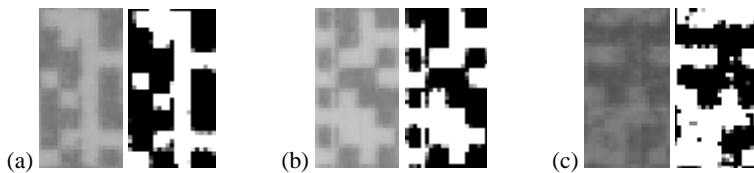
Fig. 8.7. Problems with adaptive thresholding (shown are a part of the original image as well as the corresponding part of the contrast stretched image): (a) vertical dark line; (b) vertical bright line; (c) high noise.

white version of the original Data Matrix. However, closer inspection of the contrast stretched low-contrast images reveals some problems, as illustrated in Fig. 8.7. Most problematic outputs are either due to printing errors (vertical dark or bright lines) or due to noise caused by the paper of the envelope. These cases warrant further attention.

8.4 Image Degradation

The learning of image processing tasks is based on the idea that one can use the output of a simple method for unproblematic examples as desired output for a network that is trained to produce them even if the unproblematic examples are degraded. In the case of binarization of Data Matrix codes, a Neural Abstraction Pyramid is trained to produce the output of the adaptive thresholding method not only for the original images, but for degraded versions of them as well. Degradation is done by:

– adding vertical dark and bright lines,
– adding a smoothly varying background level,
– lowering contrast, and
– adding pixel noise.

These degradations are illustrated in Figure 8.8. The number of vertical lines is uniformly drawn from the interval $[0, 99]$. They are positioned uniformly and have
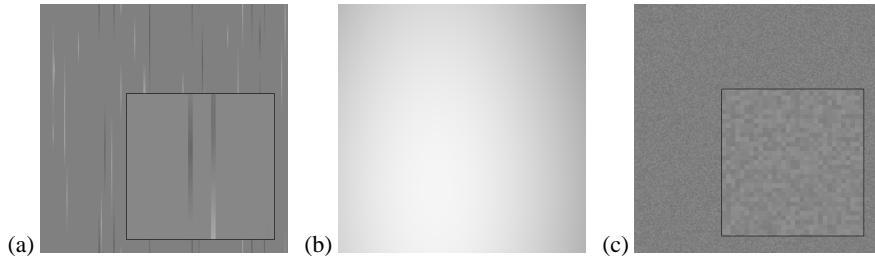
Fig. 8.8. Image degradation: (a) vertical light and dark lines; (b) background level; (c) pixel noise.
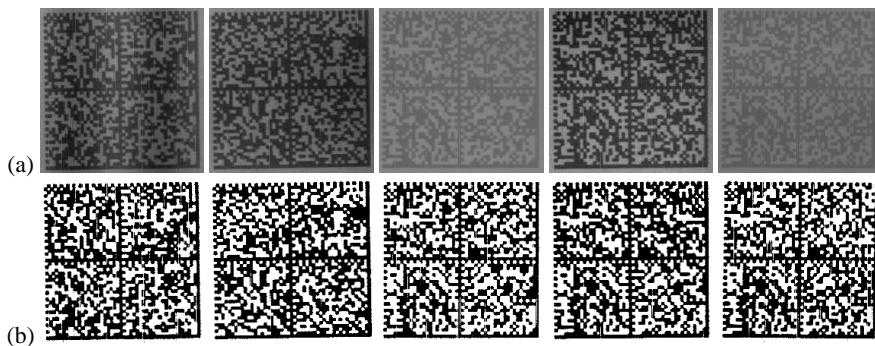
Fig. 8.9. Degraded images (originals in Fig. 8.3(a)): (a) with random degradations; (b) output of the adaptive thresholding for the degraded images.

a length between 21 and 121 pixels. The intensity of a line interpolates linearly between the original gray value at the ends, and the estimated background or foreground intensity in the center, as can be seen in Fig. 8.8(a). The background level, shown in Part (b) of the figure, is computed as sum of a horizontal and a vertical sinusoid. Their random amplitudes can reach the difference between the background and the foreground intensity. The phases are distributed uniformly and the wavelengths are chosen uniformly between 63 and 1420 pixels. Image contrast is lowered to $[0.1, 0.6]$ times the original level. The amplitude of the uniform pixel noise depends linearly on contrast and can reach the value of 24. Fig. 8.8(c) shows an example of such pixel noise.

In Figure 8.9(a) the degraded versions of the images from Fig. 8.3(a) are shown. Part (b) of the figure displays the output of the adaptive thresholding for the degraded images. It can be seen that the outputs are of much lower quality than the ones in Fig. 8.6(a) which were produced from the undegraded images. In particular, the vertical lines cannot be removed by a pixel-based method.

8.5 Learning Binarization

If one wants to develop a binarization method that outperforms adaptive thresholding, one has to utilize the structure present in the data. More specifically, one can expect a method to perform well that recognizes the Data Matrix cells and assigns white or black to an entire cell, and not to single pixels. Of course, one could develop manually an algorithm that works in this way, but the purpose of the following experiment is to demonstrate that it is possible to solve the problem without the need to think about an application-specific algorithm.

The approach followed is to use a general-purpose tool, the Neural Abstraction Pyramid introduced in Chapter 4, and adapt it to the specific task by learning from input-output examples. The generalization performance of the trained network is tested and compared to the adaptive thresholding method.

The architecture of the Neural Abstraction Pyramid network used for binarization is sketched in Figure 8.10. It has four layers with an increasing number of feature arrays and decreasing resolution. Layer 0 contains the input image and two additional feature arrays of size 216×216. The number of feature arrays doubles, while their resolution is halved when going to the next layer, until Layer 3 is reached, where 16 feature arrays of size 27×27 are present. A two pixel wide border surrounds the feature arrays. The activities of the border cells are copied from feature cells using wrap-around.

The network's processing elements contain output-units with a sigmoidal transfer function f_{sig} ($\beta = 1$, see Fig. 4.5(a) in Section 4.2.4). They receive input from

Output

Input

Layer 0 (216x216)   Layer 1 (108x108)   Layer 2 (54x54)   Layer 3 (27x27)
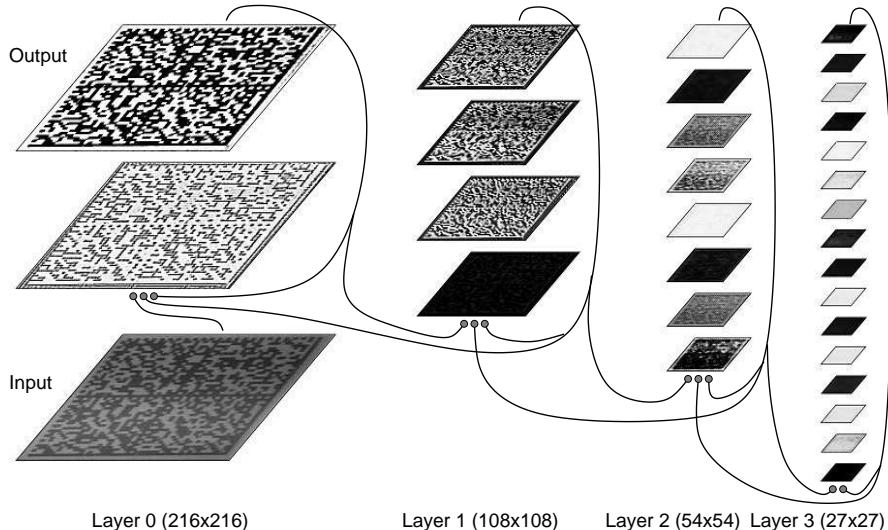
Fig. 8.10. Architecture of the Neural Abstraction Pyramid network used for learning the binarization of Data Matrix codes. The network consists of four layers shown from left to right. As the number of feature arrays increases from layer to layer, the resolution of the layers decreases.

forward, lateral, and backward projections with linear transfer functions. Forward projections come from 4×4 windows of all feature arrays in the layer below. Lateral projections originate from the 5×5 hyper-neighborhood in the same layer and backward projections access a single cell of all feature arrays in the layer above. The weights can have positive or negative values and are allowed to change their sign during training. The network has a total of 11,910 different weights. Most of them are located in the top layer since the few weights in the lower layers are shared far more often than the ones in the higher layers.

The version of the Neural Abstraction Pyramid network that is used for binarization has relatively few feature arrays. The reason for this restriction was the need to limit the computational effort of simulating the pyramid on a PC. Due to the relatively high resolution of the input images, the iterative binarization of one Data Matrix code required about two seconds on a Pentium 4 1.7GHz PC.

The undegraded Data Matrix images as well as their degraded versions are presented to the network without any preprocessing. One of the feature arrays in the bottom layer is used as network output.

The target values that are used as the desired output for the supervised training are computed using the adaptive thresholding method for the undegraded images. The network is trained to iteratively produce them not only for the original images, but for the degraded versions of these images as well. This approach has the advantage that the effort for producing a desired output for low-quality images is not necessary. If one wanted to produce a desired output for the degraded images without relying on the original versions, one would need to use time-consuming manual labeling which is avoided by using the adaptive thresholding for the undegraded originals.

The 515 high-contrast images were partitioned randomly into 334 training images (TRN) and 181 test examples (TST). For each example, one degraded version is added to the sets. The network is trained for ten iterations with a linearly increasing error-weight using backpropagation through time (BPTT) and RPROP, as described in Section 6.

8.6 Experimental Results

After training, the network is able to iteratively solve the binarization task. Figure 8.11 displays how the activities of all features evolve over time for one of the degraded test examples. It can be seen that the lower layers represent the cell structure of the code, while the higher layers are dominated by representations of the background level and the local black-and-white ratio. One can furthermore observe that the network performs an iterative refinement of an initial solution with most changes occurring in the first few iterations and fewer changes towards the end of the computation. In fact, the activities of iteration 7 and 11 are hardly distinguishable.

In Figure 8.12, the activities of the two Layer 0 feature arrays are displayed in more detail. The upper row shows the development of the output. In the first
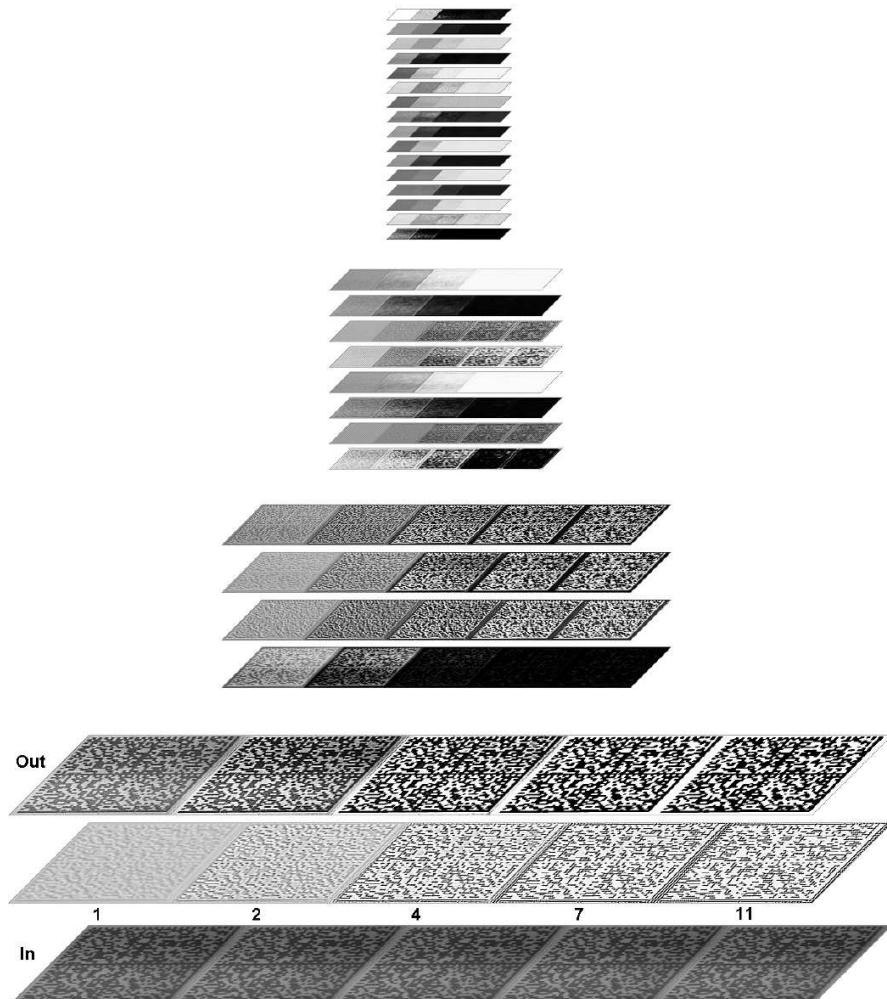
Fig. 8.11. Recall of network trained for binarization of Data Matrix codes. The development of all feature array activities is shown for one of the degraded test examples.

few iterations, the non-uniform background level causes the upper part of the code to have higher activity than the lower part. This inhomogeneity is removed during refinement. Furthermore, the output is driven from intermediate gray values that signal uncertainty towards black and white, which is characteristic of the desired output.

In the lower row of the figure, the only hidden feature array of the lowest layer is displayed. Here, a representation of the cell structure emerges. Bright areas of the input image are broken into a discrete number of cells. For each bright cell, an activity blob rises and remains stable. Adjacent blobs are connected either vertically
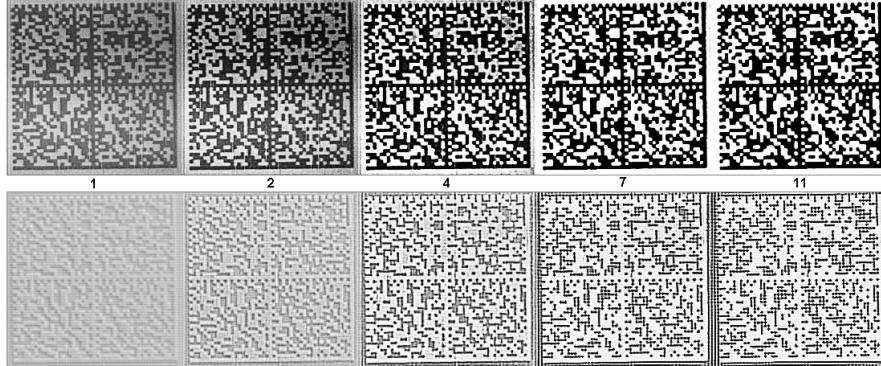
Fig. 8.12. Recall of network trained for binarization of Data Matrix codes. The development of the feature array activities at Layer 0 is shown for one of the degraded test examples.

Fig. 8.13. Recall of network trained for binarization of Data Matrix codes. Shown are activities for parts of a degraded test image after 11 iterations: (a) original image; (b) hidden feature array; (c) output feature array.

or horizontally, depending on the prominent local orientation of the corresponding bright area. If such a local orientation cannot be determined, e.g. in bright areas that have a larger width as well as a larger height, the blobs form a loosely connected matrix.

Figure 8.13 zooms at the lower left corner of the code and displays the activities after 11 iterations to illustrate this behavior. It is evident that the hidden feature array represents discrete cells, covering about 4×4 pixels, rather than single pixels. The blobs inhibit the output feature cells. Hence, network has learned that the output of a cell must be coherent. This suppresses thin vertical lines and pixel noise.

To understand the emergence of the blobs, one can look at the contributions made by input, lateral, and backward weights, as shown in Fig. 8.14. The weak weights of the input projections detect contrast at the upper and right border of a bright area. The contributions of lateral projections shape the blobs through a center-center excitation and a center-surround inhibition. Here, the typical blob distance of about four pixels is enforced. Finally, the backward projections excite or inhibit entire areas, not discrete blobs. Thus, at Layer 1 a coarser representation of black and white areas must exist.
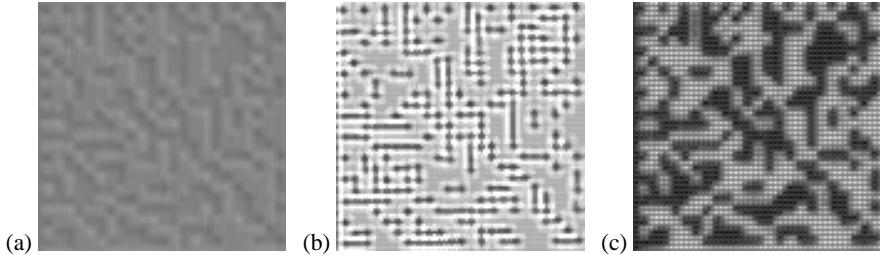
Fig. 8.14. Contributions to the activity of the hidden feature from Fig. 8.13(b) (bright shading represents inhibition, dark shading represents excitation): (a) via input projections; (b) via lateral projections; (c) via backward projections.
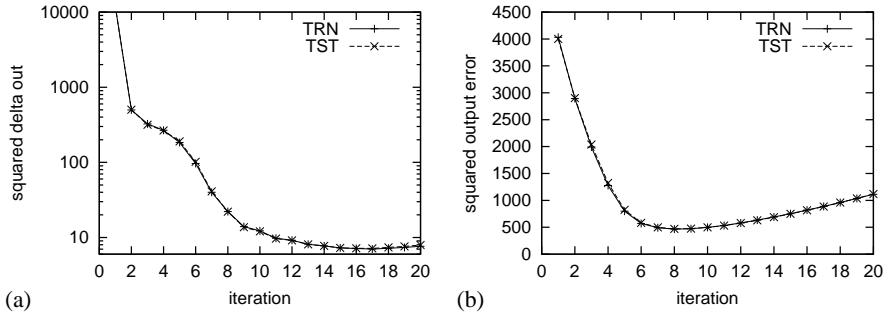
Fig. 8.15. Recall of iterative Data Matrix binarization over time: (a) average squared output change; (b) average squared output difference to desired output.

After analyzing the network's behavior for a single example, its performance for all examples of the high-contrast dataset is measured. Figure 8.15(a) displays the average squared output change over time. In the first iterations, the output changes considerably. The changes decrease quickly until iteration 10 and remain low afterwards. Thus, the network dynamics is stable even when iterating twice as long as it was trained for. The small average changes at higher iterations indicate that the network converges to an attractor for almost every example.

In Figure 8.15(b) the average squared difference to the desired output is shown over time. One can observe that the average error decreases rapidly during the first iterations. It reaches a minimum at about iteration 8 and increases again slowly. Hence, the network's attractors are not identical to the desired outputs.

This is not surprising since the network was trained to produce the desired output only for ten iterations. When iterated further, the dynamics evolves into stable attractors that resemble the cell structure of the Data Matrix codes. This cell structure was not used to produce the desired outputs. In contrast, the desired outputs were computed by a pixel-based adaptive thresholding, as described in Section 8.3.

The desired outputs are not necessarily the ideal outputs but only approximations to the best recognizable ones. Thus, a deviation from the desired outputs does not necessarily indicate a decrease in output quality, as measured in recognition

	TRN		TST	
	degraded	original	degraded	original
confidence	.9646	.9812	.9649	.9811
output error	757.4	238.5	756.6	238.6

(c)
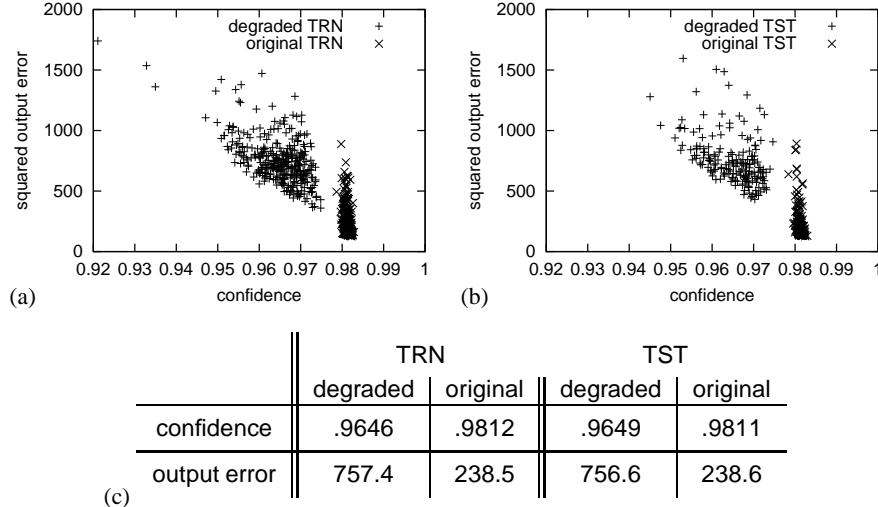
Fig. 8.16. Recall of iterative Data Matrix binarization. Shown are confidence versus squared output error for the original and the degraded images: (a) training set; (b) test set; (c) average confidences and squared errors.

performance of Data Matrix codes. If one wanted to produce a longer decrease of the average difference to the desired output, one could always train the network for more iterations. This was not done here since ten iterations seem to be sufficient to solve the binarization task.

Both parts of Figure 8.15 display data for the training set as well as for the test set. Since both curves are almost indistinguishable, it can be claimed that the network generalized the binarization task well.

The output of the network approaches an attractor that is characterized by black and white cells, represented by activities of one and zero, respectively. Intermediate activities indicate uncertainty. Of course, the network's uncertainty is maximal at the start of the computation and decreases as the network makes decisions about the output. As described above, the network has not reached a stable attractor for all examples at iteration 10. In addition, even at an attractor, the output can remain undecided if the input is ambiguous. To measure the networks certainty, for each output activity a, a confidence c is computed from the minimal squared distance to one of the extreme values:

$$
\begin{aligned}
d_0 &= (0-a)^2 = a^2, \\
d_1 &= (1-a)^2, \\
c &= 1 - 4 \cdot \min(d_0, d_1).
\end{aligned}
\tag{8.4}
$$

Since a is in the interval $[0, 1]$, c is in $[0, 1]$ as well. The confidence of an entire image is the average confidence of all output feature cells.
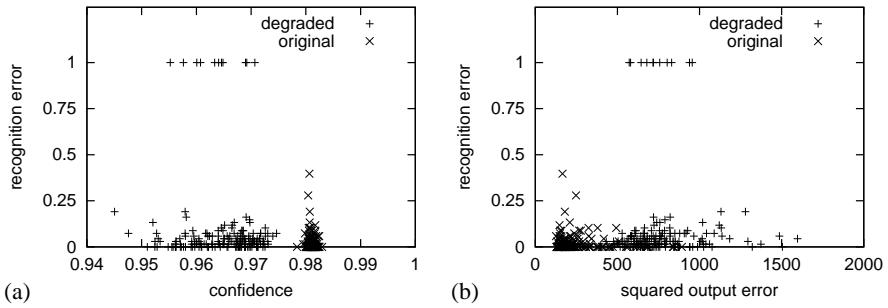
Fig. 8.17. Data Matrix test set recognition. (a) confidence versus recognition error; (b) squared output distance versus recognition error.

In Figure 8.16 the confidence of all training and test examples at iteration 10 is shown versus the squared distance to the desired output. Here, a clear distinction between the original images and the degraded images becomes obvious. For the original undegraded images, outputs with higher confidence and lower distance are produced by the network than for the degraded images. In fact, none of the originals has a lower confidence than any of the degraded images.

Again, the plots for the training set and the test set as well as their average confidences and output errors, summarized in Fig. 8.16(c), are very similar. One can also observe that the confidence is anti-correlated to the output error and can hence be used to reject ambiguous examples that could lead to recognition errors.

For all examples, the output of adaptive thresholding as well as the output of iterative binarization has been presented to a Data Matrix recognition engine. This evaluation was done by Siemens ElectroCom Postautomation GmbH. It produces for each recognized example the percentage of error correction used. This value will be referred to as recognition error. It is set to one if an example could not be recognized at all.

Both methods recognized 514 (99.8%) of the 515 original examples. The degraded images were harder to recognize. From the adaptive thresholding output 476 (92.4%) images could be recognized, while the system recognized 482 (93.5%) examples when the iterative binarization method was used. For comparison, the system could only recognize 345 (70.0%) of the 515 degraded examples if a simple global thresholding method was used for binarization.

Figure 8.17 shows for the iterative binarization method, the recognition error of all test examples plotted against the confidences and against the squared distances to the desired output. In both cases, the relation between the two quantities is not obvious. The degraded images that could be recognized do not need a higher percentage of error correction than the original images. However, all examples that could not be recognized are degraded images having a relatively low confidence and a relatively high distance.

In Figure 8.18(a) the recognition error of the adaptive thresholding and the iterative binarization method are compared for the test set. The performance of the two

(a)

degraded		adaptive thresholding	iterative binarization
rejected		11 (6%)	11 (6%)
average recognition error		0.176	0.094

original		adaptive thresholding	iterative binarization
rejected		0	0
average recognition error		0.0203	0.0193

(b)
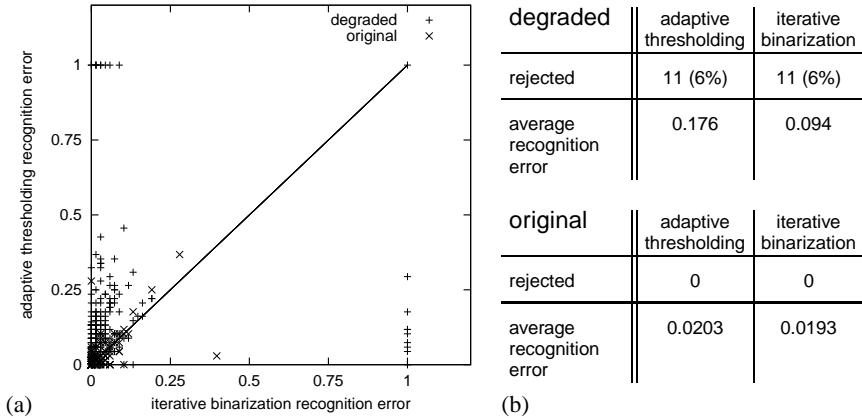
Fig. 8.18. Data Matrix test set recognition: (a) recognition error of iterative binarization versus adaptive thresholding; (b) average recognition errors.

(a)   (b)   (c)
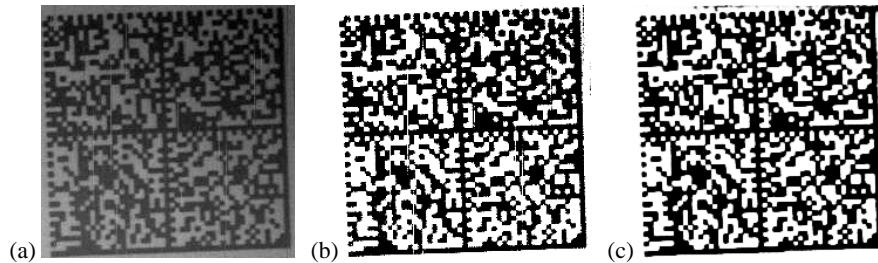
Fig. 8.19. Most difficult Data Matrix code. Shown is the only test example that could neither be recognized using adaptive thresholding nor using iterative binarization: (a) degraded input image; (b) output of adaptive thresholding; (c) output of iterative binarization.

methods is not much different for the original images since the network has been trained to resemble the behavior of adaptive thresholding for such images.

In contrast, for most degraded images, the recognition error is lower when using the iterative binarization method than when adaptive thresholding is used. The average recognition error for iterative binarization (0.094) is much lower than the one of adaptive thresholding (0.176), as summarized in Fig. 8.18(b). Thus, the use of iterative binarization substantially lowers the need for error correction in the recognition engine.

Further, it can be seen in the Figure 8.18(a) that most of the 11 test examples which could not be recognized when using one method could be recognized when using the other method. Only one test example (0.55%), shown in Figure 8.19, was rejected by the recognition engine in both cases. It is difficult due to its rotation, the non-uniform background, and the large number of vertical lines. Hence, the combination of both methods could lower the rejection rate of the system by an order of magnitude, compared to each method alone.
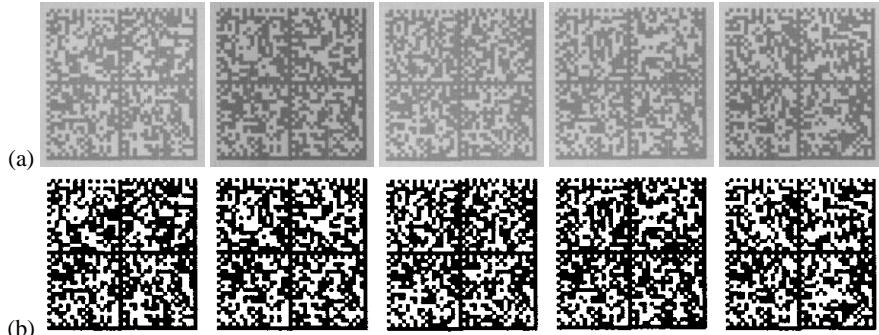
Fig. 8.20. Low-contrast degraded images (originals in Fig. 8.3(b)): (a) with random degradations; (b) output of the improved adaptive thresholding for the degraded images.

So far, only results for the high-contrast address window variant of the Data Matrix codes have been reported. In the remainder of this section, results of binarizing the low-contrast red ink code variant are described.

The 694 images were partitioned randomly into a training set of size 467 and a test set of size 227. Since these images differ in brightness, contrast, and cell size from the address window code variant, a retraining of the network was necessary.

Again, for each image a degraded version was produced. This time the images were degraded only moderately since the image quality was already low. Only up to nine vertical lines were added, and the amplitude of the background level as well as the amplitude of the pixel noise was reduced by a factor of two. Figure 8.20(a) shows degraded versions of the images from Fig. 8.3(b).

The desired outputs were produced using an improved adaptive thresholding method that smoothed the image horizontally by applying a $1/4$ (1 2 1) binomial kernel prior to contrast stretching. This reduced the effects of the vertical dark and bright lines present in the images. Figure 8.20(b) shows the binarized output of this method for the degraded images.

The network was trained to iteratively reproduce the desired outputs not only for the original images, but for their degraded versions as well. After training, the network's behavior was very similar to the behavior of the network trained to binarize the high-contrast address window variant of Data Matrix codes. The network develops a stable representation of the cell structure which is used for binarization.

Figure 8.21 shows the average squared output changes and the average squared difference to the desired output over time. The network converges quickly to an attractor and stays there even when iterated further than the ten iterations it was trained for. This attractor is close to the desired output. Generalization is good since the curves for the training set are virtually identical to the test set curves.

The binarized outputs were evaluated by Siemens ElectroCom Postautomation GmbH. For this experiment the recognition engine was queried a second time with different parameter settings when an example was rejected in the first run. Table 8.1 summarizes the recognition performance for the entire dataset. It can be seen that

(a)

(b)
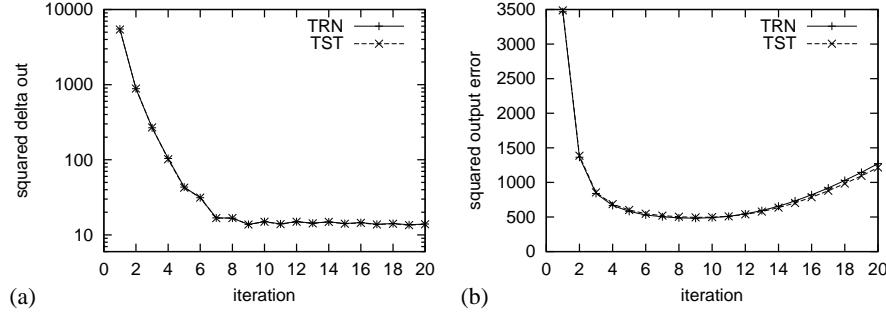
Fig. 8.21. Recall of iterative low-contrast Data Matrix binarization over time: (a) average squared output change; (b) average squared output difference to desired output.

	adaptive thresholding		iterative binarization	
	original	degraded	original	degraded
first run	672 (96.8%)	435 (62.7%)	678 (97.7%)	673 (97.0%)
second run	19 (2.7%)	13 (1.9%)	16 (2.3%)	20 (2.9%)
sum	691 (99.6%)	448 (64.6%)	694 (100%)	693 (99.9%)

Table 8.1. Low-contrast Data Matrix recognition performance: Number of recognized code images.

the second run is able to reduce the number of rejected images considerably. For example, all 20 original examples binarized using the iterative binarization method which were rejected by the first run can be recognized in the second run. In the sum of both runs the iterative binarization performs slightly better on the original images than the adaptive thresholding, yielding perfect recognition, compared to 0.4% rejects. On the degraded images its performance is much better than the one of the adaptive thresholding method. Only one example is rejected, compared to 246 rejects.

Figure 8.22(a) plots the recognition error of iterative binarization against the one of the adaptive thresholding method. One can observe that there is some potential for combining both methods since there are examples where one method has a high recognition error while the other has a low one.

As summarized in Fig. 8.22(b), the recognition error of iterative binarization is lower for the original images than the one of adaptive thresholding. It is much lower for the degraded images where the recognition errors differ by a factor of more than twenty.

Thus, while the iterative binarization method performs only slightly better on the original images than the adaptive thresholding method for the red ink low-contrast variant of the Data Matrix codes, it outperforms adaptive thresholding dramatically on the degraded images.
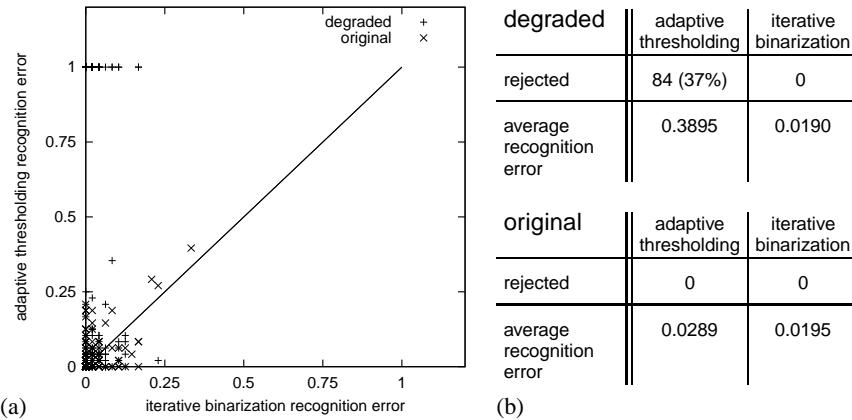
Fig. 8.22. Low-contrast Data Matrix test set recognition: (a) recognition error of iterative binarization versus adaptive thresholding; (b) average recognition errors.

8.7 Conclusions

In this chapter, it was shown that a non-trivial image processing task can be learned by an instantiation of the Neural Abstraction Pyramid architecture. An adaptive thresholding method was developed that is able to successfully binarize high-contrast images of Data Matrix codes. Its results were used as desired output for a Neural Abstraction Pyramid that was trained to iteratively produce them not only for the original images, but also for degraded versions of them.

The network learns to recognize the cell structure of the Data Matrix and to use it for binarization. The performance of both methods was evaluated using a Data Matrix recognition system. The trained network performs as well as adaptive thresholding for the original images, but outperforms it for degraded images. The combination of both methods was able to lower the rejection rate significantly.

For the low-contrast red ink variant of the Data Matrix codes, the advantage of iterative binarization is more obvious. It performs better than adaptive thresholding for the original images and outperforms it dramatically for the degraded images.

9. Learning Iterative Image Reconstruction

Successful image reconstruction requires the recognition of a scene and the generation of a clean image of that scene. In this chapter, I show how to use Neural Abstraction Pyramid networks for both analysis and synthesis of images. The networks have a hierarchical architecture which represents images in multiple scales with different degrees of abstraction. The mapping between these representations is mediated by a local recurrent connection structure.

Degraded images are shown to the networks which are trained to reconstruct the originals iteratively. Through iterative reconstruction, partial results provide context information that eliminates ambiguities.

The performance of this approach is demonstrated in this chapter by applying it to four tasks: super-resolution, filling-in of occluded parts, noise removal / contrast enhancement, and reconstruction from sequences of degraded images.

9.1 Introduction to Image Reconstruction

Frequently, digital images of real-world scenes do not have enough quality for the application at hand since the images have been degraded in some way. These degradations arise in the image formation process (e.g. from occlusions) and from the capturing device (e.g. due to low resolution and sensor noise).

The goal of the image reconstruction process is to improve the quality of the captured images, e.g. by suppressing the noise. To separate noise from objects, models of the noise and the objects present in the images are needed. A scene can then be recognized, and a clean image of it can be generated.

Freeman and Pasztor [72] recently proposed a learning approach to low-level vision that they termed VISTA. It models images and scenes using Markov random fields. The parameters of their graphical models can be trained, e.g. for a super-resolution task. The demonstrated performance of the system is impressive. However, the models have no hidden variables, and the inference via belief propagation is only approximate.

A common problem with image reconstruction is that it is difficult to decide the right interpretation of an image part locally. For example, it might be impossible to decide in a digit binarization task whether or not a pixel belongs to the foreground by looking only at the pixel's intensity. If contrast is low and noise is present, it

could be necessary to bias this decision with the output of a line-detector for that location.

In general, to eliminate such local ambiguities, more context is needed. Feedforward models that cover such large context have many free parameters. They are therefore expensive to compute and difficult to train.

Here, I propose to iteratively transform the image into a hierarchical representation. The image is interpreted first at locations where little ambiguity exists. These partial results are used as context to bias the interpretation of more ambiguous regions. The reconstruction problem is described using examples of degraded images and desired output images, and then a recurrent neural network of suitable structure is trained to solve the problem.

In order to investigate the performance of the proposed approach for iterative image reconstruction, a series of experiments was conducted with images of handwritten digits. The reasons for choosing digits were that large datasets are publicly available, and that the images contain multiscale structure which can be exploited by the learning algorithm. Clearly, if there were no structure to learn, training would not help. The digits were degraded by subsampling, occluding parts, or adding noise, and Neural Abstraction Pyramid networks were trained to reconstruct the originals.

9.2 Super-Resolution

Super-resolution is the process of inferring high-resolution detail from low-resolution images. It is a typical image reconstruction problem that has been investigated by many researchers since it is needed for applications, such as enlarging consumer photographs or converting regular TV signals to HDTV. Different complementary approaches exist to increase the perceptual resolution of an image, as illustrated in Figure 9.1.

The first idea is to sharpen the images, by amplifying existing high-frequency image content. This is a dangerous operation since noise will be amplified as well. The next approach is to fuse multiple low-resolution images that have been captured at slightly different positions. Fusion is based on the constraint that the super-resolution image, when appropriately warped and down-sampled (to model the image formation process), should yield the low-resolution inputs. This is feasible if
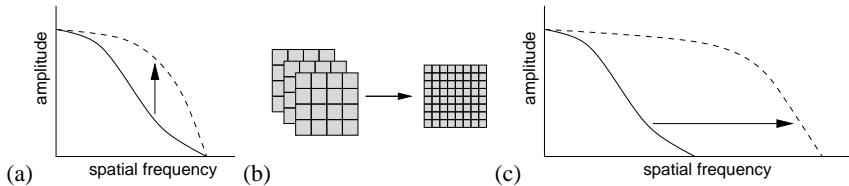
Fig. 9.1. Complementary approaches to increase the perceptual resolution of an image: (a) amplifying existing high frequencies; (b) combining multiple displaced low-resolution images; (c) estimating image details.

such image-sequences are available. However, Baker and Kanade [12] have shown that there exist fundamental limits on the reconstruction quality, even if infinitely many low-resolution images are used. More specifically, since the constrained volume contains many solutions, a smoothness prior is usually used to select one of them. This leads to overly smooth reconstructions.

One would like to have an intelligent method for expanding the resolution of an image. It should keep edges sharp, which are implicitly described in the low-resolution image, and it should make intelligent guesses about the details of textures. A natural solution to this problem is to estimate the missing image details using the non-uniform distribution of images. Since some image structures, such as edges and lines, are more likely than others, a super-resolution method can be biased to reconstruct them.

To make this approach work, a training set of aligned high-resolution and low-resolution images is needed to estimate the prior. The more specific this training set is, the sharper the prior will be. Three of such informed super-resolution methods have been recently proposed.

Baker and Kanade describe in [12] a system that 'hallucinates' high-resolution faces from low-resolution images. They use a database of normalized faces and find from it the image patch most similar to the low-resolution input patch. To measure similarity they use multiscale derivative features called parent structure vectors [34]. The method is also applied to images of text.

Freeman *et al.* [71] proposed example-based super-resolution as a simplified and faster method, compared to an earlier proposed Markov network which works iteratively by applying Belief Propagation [73]. They proceed in scan-line order to match contrast normalized overlapping patches of the input to a database of training images. Thus, spatial consistency between patches is enforced to the left patch and to the previous line only. To measure similarity, the L_2-norm is used.

Hertzmann *et al.* [95] applied a supervised filter design method that they termed 'image analogies' to the super-resolution task. The method works also in scan line order, but uses a multi-scale image synthesis approach. They use a distance measure that enforces spatial consistency. The approach is also applicable to texture transfer tasks and to generate artistic image filters.

All of the above three methods generate plausible image detail from low-resolution images. Although they build data structures, such as trees, from the training set, the models used are very complex and thus need much memory to store many free parameters and require intensive computations to find the best matching training example.

In the following, I show how to condense the information present in the training examples into the few parameters of a hierarchical recurrent neural network through supervised learning. The effort of training the network pays off during the recall phase.
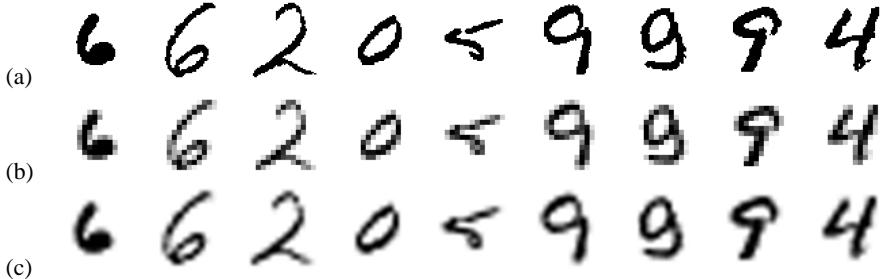
Fig. 9.2. Some digits from the NIST dataset. Shown are: (a) centered images in the original resolution (64×64); (b) subsampled to 16×16 pixels (pixelized); (c) bicubic interpolation to original resolution (blurred).

9.2.1 NIST Digits Dataset

The first reconstruction experiment is done using the original NIST images of segmented binarized handwritten digits [80]. They have been extracted by NIST from hand printed sample forms. The digits are contained in a 128×128 window, but their bounding box is typically much smaller. For this reason, the bounding box was centered in a 64×64 window to produce the desired output Y. Figure 9.2(a) shows some centered sample images from the NIST dataset. The input X to the network consists of subsampled versions of the digits with resolution 16×16, shown for the examples in Fig. 9.2(b), which were produced by averaging 4×4 pixels. Part (c) of the figure demonstrates that bicubic interpolation is not an adequate method to increase the resolution of the NIST digits since it produces blurred images.

9.2.2 Architecture for Super-Resolution

The network used for the super-resolution task is a very small instance of the Neural Abstraction Pyramid architecture. Besides the input and the output feature arrays, determined by the task, it has additional features only in the hidden layer. Such a small network was chosen because it proved to be sufficient for the task.

The architecture of the network is illustrated in Figure 9.3. It consists of three layers. The rightmost Layer 2 contains only a single feature array of resolution 16×16. The activities of its cells are set to the low resolution input image.

Layer 1 has resolution 32×32. It contains four feature arrays that produce a hidden representation of the digit. The leftmost Layer 0 contains only a single feature array that is used as network output. It has the resolution 64×64.

The feature cells of the output feature have lateral and backward projections. The weight matrix of the lateral projections has a size of 3×3. The 2×2 different backward projections each access a single feature cell of each feature array in Layer 1. This corresponds to the inverse of non-overlapping 2×2 forward projections for the four Layer 1 features.

Feature cells in Layer 1 have all three types of projections. Forward projections access 2×2 windows of the output feature array in Layer 0. Lateral projections

originate in the 3×3 hyper-neighborhood of a feature cell in the same layer. Backward projections receive input from a single cell in the low-resolution input array of Layer 2. For each of the four features, there are 2×2 different backward projections.
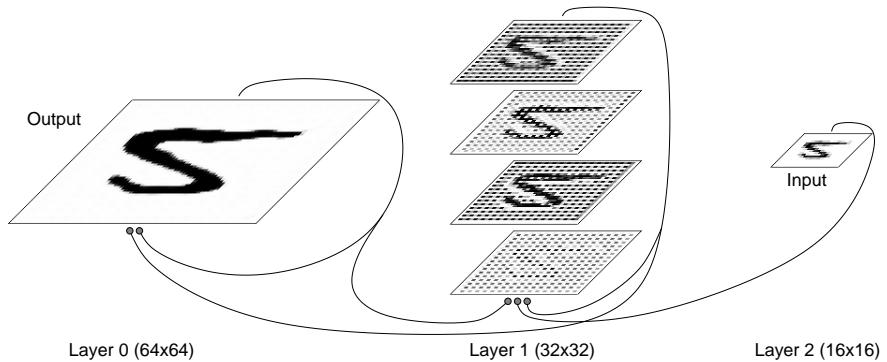
Fig. 9.3. Network for super-resolution. It is a small Neural Abstraction Pyramid with four hidden feature arrays in the middle layer. Each pixel corresponds to a feature cell that is connected to its neighbors. The gray values show the activities after the recurrent network has iteratively reconstructed the high-resolution image.

While all projection units have linear transfer functions, a sigmoidal transfer function f_{sig} ($\beta = 1$, see Fig. 4.5(a) in Section 4.2.4) is used for the output units of the processing elements. The feature arrays are surrounded by a one pixel wide border that is set to zero since the background of the input and the desired output has this value.

The network's 235 free parameters are initialized randomly, and they are trained for ten time steps on a fixed set of 200 randomly chosen example digits. The test set consisted of 200 different randomly chosen examples. Training was done using the back-propagation through time (BPTT) method in combination with the RPROP learning algorithm, as described in Section 6. The weighting of the squared output error increased linearly with time.

9.2.3 Experimental Results

Figure 9.4 shows how the output of the trained network develops over time when the first five digits of the test set are presented at its input. After two iterations, the input can influence the output, but no further interactions are possible yet. Thus, the output looks like a copy of the low-resolution input. In the following iterations, the initial reconstruction is refined. Black lines with sharp smooth borders are generated. After iteration five, the network's output does not change significantly any more.

The outputs produced by the network are close to the targets. The differences are shown in Figure 9.5. Some small details at the edges of the lines, probably caused by noise that was amplified by the binarization procedure, have not been

Input	2	3	5	10	Target

Fig. 9.4. Iterative super-resolution. The activity of the network's output feature array is shown over time along with the low-resolution input and the high-resolution target. The initial output is refined as the number of iterations increases.
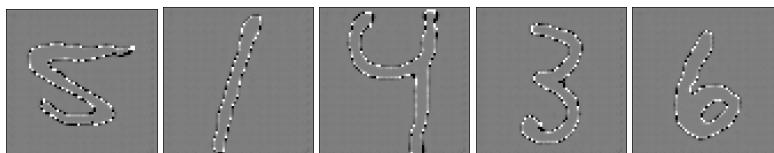
Fig. 9.5. Differences between output of the recurrent super-resolution network and the target. Large deviations occur because the choppy edges of the targets are approximated by smooth contours.

reproduced, but have been replaced with smooth edge segments. For this reason, the reconstructions frequently have a higher perceptual quality than the targets.

Figure 9.6 shows the contributions to the activity of the output feature cells after ten iterations. One can see that the feature cells belonging to the lines are excited via the backward projections, while their neighborhood is weakly inhibited by the Layer 1 features. The backward influence on the rest of the image is weak. The influence of the lateral projections is strongly inhibitory everywhere, except at the lines, where it is excitatory.
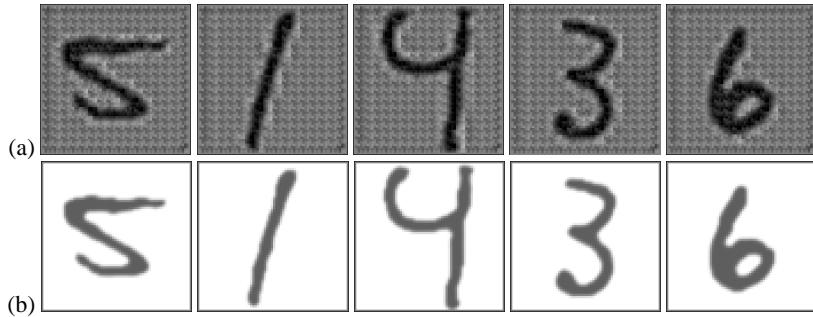
Fig. 9.6. Contributions to the output feature array for the digits of Fig. 9.4 at iteration 10 (dark shading represents excitation, bright shading indicates inhibition): (a) via backward projections (lines are excited and their surround is weakly inhibited); (b) via lateral projections (the background is strongly inhibited, while the lines are weakly excited).
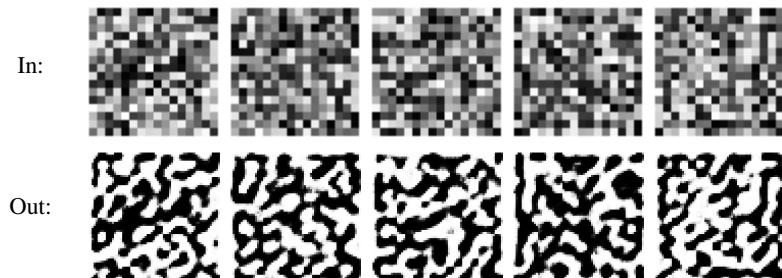
Fig. 9.7. Response of the super-resolution network to uniform noise. Black lines with smooth borders are synthesized at positions where many dark pixels are present in the input image.

The network tries to concentrate the dark foreground present in the low-resolution input images at black lines with smooth borders. To illustrate this behavior, uniform pixel noise drawn from the interval $[0, 1]$ is presented to the network. The stable response after ten time steps is shown in Figure 9.7. The network synthesizes smooth black lines at positions where many dark pixels are present in the input image.

This behavior can also be observed in Figure 9.8. Here, a low-resolution line-drawing is presented as input to a spatially enlarged version of the network. The crisp network output is shown along with a bicubic interpolation that looks blurred.

The effect of the recurrent computation was further investigated by training a version of the recurrent neural network (RNN) that has eight hidden feature arrays in the middle layer as well as two feed-forward neural networks (FFNN) with four and eight features on the same dataset. The units of the FFNNs each accessed 3×3 windows of the previous layer. This choice ensured that the networks had a similar number of adjustable parameters as the corresponding RNNs.

Figure 9.9 shows for the next five test digits the output of the four tested networks after 10 iterations. In general, the reconstructions are good approximations to

Fig. 9.8. Response of the super-resolution network to a line-drawing: (a) original low resolution image (pixelized); (b) bicubic interpolation (blurred); (c) output of iterative super-resolution network (crisp).

Input	RNN	FFNN	RNN	FFNN	Target
	small		large		
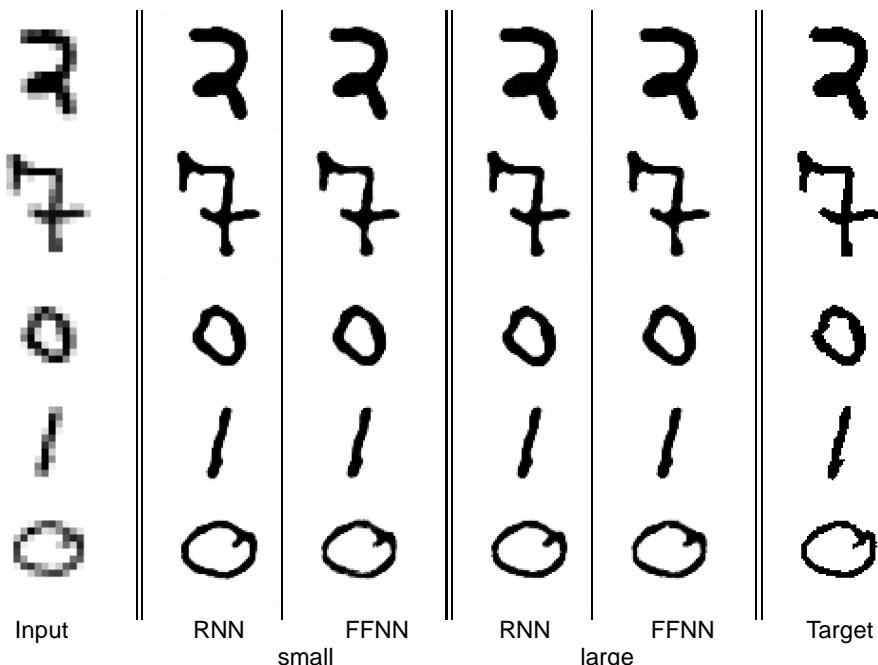
Fig. 9.9. Outputs of different super-resolution networks. The responses of large and small versions of the recurrent network (RNN) and the feed-forward network (FFNN) to the low-resolution input are shown.

the high-resolution targets, given the low-resolution inputs. The RNN outputs have a slightly higher perceptual quality than the responses of the corresponding FFNNs.

In Figure 9.10, the mean square error of all four networks is displayed. The test set reconstruction error of the recurrent networks decreases quickly and remains below the error of the corresponding FFNN after six time steps. At iterations 9 and 10 the small RNN outperforms even the large FFNN. When iterated beyond the trained cycles, the reconstruction error increases slightly again. This behavior could be prevented by training the network for more iterations, but this was not done here since ten iterations seem to be sufficient to solve the super-resolution task.
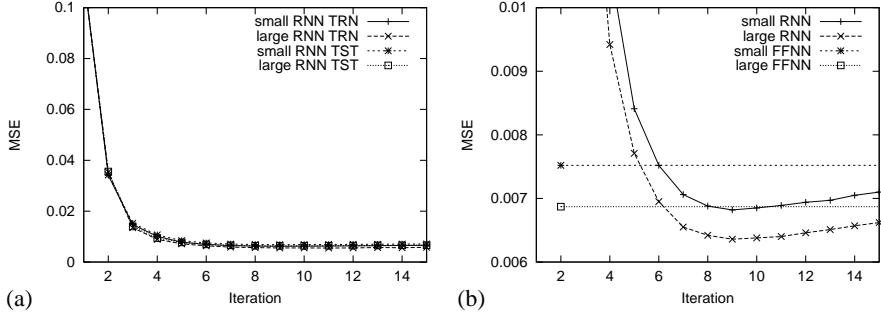
Fig. 9.10. Mean squared error of super-resolution: (a) the recurrent network on the training set and the test set; (b) detailed view of the test set performance, compared to FFNN.

9.3 Filling-in Occlusions

Occlusion is a major source of image degradation when capturing real-world images. Many computer vision systems fail if the object of interest is not completely visible. In contrast, humans can recognize partially occluded objects easily. Obviously, we subconsciously fill-in the missing parts when analyzing a scene. In fact, we are surprised when an occlusion ends and the appearing object part does not look as expected. Johnson *et al.* [113] have shown that this holds even for infants. By 4 months of age, infants appear to perceive the unity of two rod sections extending from behind an occluding object [112, 121].

Few attempts have been made to reproduce this behavior in computer vision systems. One example is the approach of Dell'Acqua and Fisher [51]. They recently proposed a method for the reconstruction of planar surfaces, such as walls, occluded by objects in range scans. The scene is partitioned into surfaces at depth discontinuities and fold edges. The surfaces are then grouped together if they lie on the same plane. If they contain holes, these are filled using linear interpolation between the border points.

Another example is the face recognition system described by O'Toole *et al.* [171]. They used average facial range and texture maps to complete occluded parts of an observed face.

There are several neural models for pattern completion, including the ones proposed by Kohonen [126] and by Hopfield [101]. Usually they work in auto-associative mode. After some training patterns have been stored in the weights of the network, partial or noisy versions of them can be transformed into one of the stored patterns.

In particular, recurrent auto-associative networks can iteratively complete patterns and remove noise by minimizing an energy function. In these networks, patterns are stored as attractors. Bogacz and Chady [33] have demonstrated that a local connection structure improves pattern completion in such networks.

Continuous attractor networks were proposed by Seung [209] to complete images with occlusions. For digits belonging to a common class, he trained a two-layer
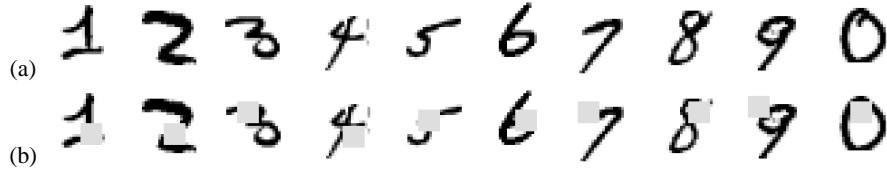
Fig. 9.11. Some examples from the MNIST dataset: (a) original images; (b) with occlusions caused by a randomly placed 8×8 light gray square.

recurrent neural network using gradient descent to reconstruct the original. The network had a local connection structure with many adaptable parameters since no weight sharing was used. The hidden units develop receptive fields that form a topographic feature map. The network is able to complete images of the single digit class it was trained with. However, it remained open if the reconstruction is possible when the digit class is unknown to the network.

In the following, I extend Seung's approach by adding lateral connections, weight sharing, and more layers to the network and by training it to reconstruct digits from all classes without presenting the class label.

9.3.1 MNIST Dataset

For the reconstruction experiments that follow, the MNIST database of handwritten digits [132] is used. The NIST digits [80] have been scaled to 20×20 pixels and were centered in a 28×28 image. Normalization to a fixed size facilitates recognition since one source of variability is removed. Centering removes translational variability as well. The lower resolution is still sufficient to recognize the digits. It allows for the use of smaller networks that facilitate generalization and reduce computational costs. Figure 9.11(a) shows some examples from the MNIST dataset.

Occlusion was simulated with an 8×8 square that is set to an intensity of 0.125 (light gray). The square is placed randomly at one of 12×12 central positions, leaving a four pixel wide border that was never modified, as shown in Figure 9.11(b). The square is placed only at inner positions to make sure that some parts of the digit are occluded.

9.3.2 Architecture for Filling-In of Occlusions

The recurrent reconstruction network is an instance of the Neural Abstraction Pyramid architecture. It consists of four layers, as illustrated in Figure 9.12. The leftmost Layer 0 has a resolution of 28×28 hypercolumns. It contains the input feature array, one hidden feature array, and the output feature array of the network.

Layer 1 contains four feature arrays of resolution 14×14. In Layer 2, the resolution drops to 7×7, while the number of different features increases to eight. The topmost Layer 3 consists of only a single hypercolumn with 16 feature cells.

Both hidden and output feature cells of Layer 0 receive input from 3×3 windows of the input feature array. The three Layer 0 feature arrays are accessed by
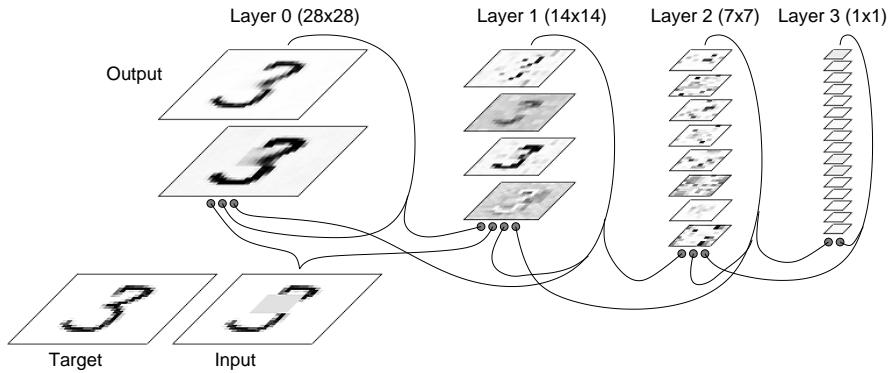
Fig. 9.12. Network architecture for filling-in occluded parts. It is an instance of the Neural Abstraction Pyramid architecture. The activities of all feature arrays are shown after twelve iterations along with the reconstruction target. The same network is also used for contrast enhancement / noise reduction and for reconstruction from sequences of degraded images.

overlapping 4×4 forward projections of Layer 1 features. Layer 2 features have 4×4 forward projections as well.

The forward and backward projections between Layer 2 and Layer 3 implement a full connectivity with $7\times7\times8\times16$ total weights in each direction. Backward projections of Layer 1 and Layer 0 are non-overlapping. For each feature, 2×2 different backward projections access the 1×1 hyper-neighborhood in the next higher layer.

Lateral projections in the first three layers originate in the 3×3 hyperneighborhood of a feature cell. These layers are surrounded by a one pixel wide border. Its activities are copied from feature cells using wrap-around. In the topmost Layer 3, the lateral projections access all 16 feature cells.

While all projection units have linear transfer functions, a sigmoidal transfer function f_{sig} ($\beta = 1$, see Fig. 4.5(a) in Section 4.2.4) is used for the output units of the processing elements. Training is done using a working set of increasing size for twelve time steps using BPTT and RPROP. A low-activity prior for the hidden features ensures the development of sparse representations.

9.3.3 Experimental Results

Figure 9.13 illustrates the reconstruction process for a test example after the network was trained. One can observe that all features contribute to the computation. In the first few time steps, a coarse approximation to the desired output is produced. The hidden feature arrays contain representations of the image content that develop over time. While the activity of the topmost Layer 3 decreases after a few iterations, the representations in the other three layers approach a more interesting attractor. They form a distributed hierarchical representation of the digit.

Fig. 9.14 shows the reconstruction process for the first ten digits of the test set. One can observe that the images change mostly at occluded pixels. This demonstrates that the network recognized the occluding square. Furthermore, the change

Output

Input
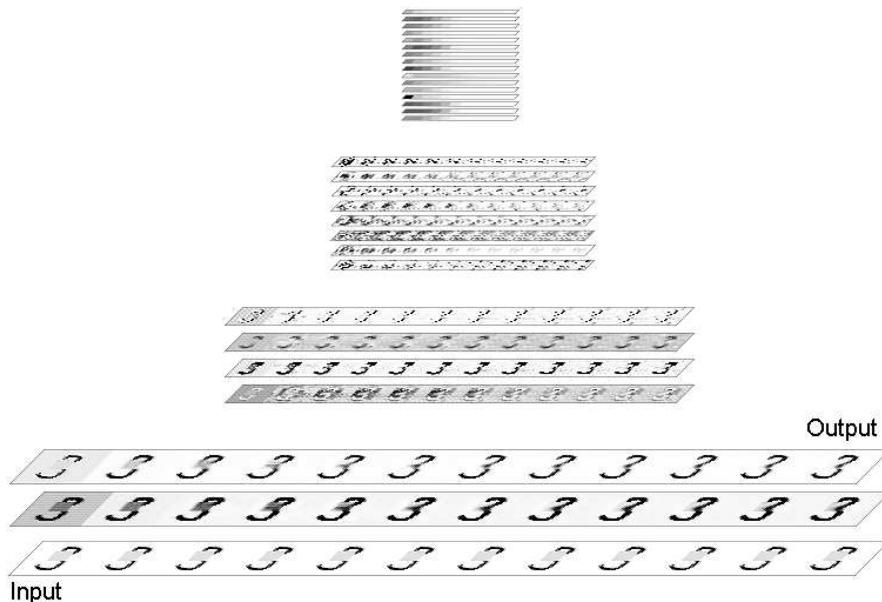
Fig. 9.13. Filling-in of occluded parts. The activities of all feature arrays are shown over time. All features contribute to the computation. The activities change most in the first few iterations. Towards the end of the sequence, the network approaches an attractor that includes the reconstructed digit in the output feature array.

Input   3   6   12   Target   Input   3   6   12   Target
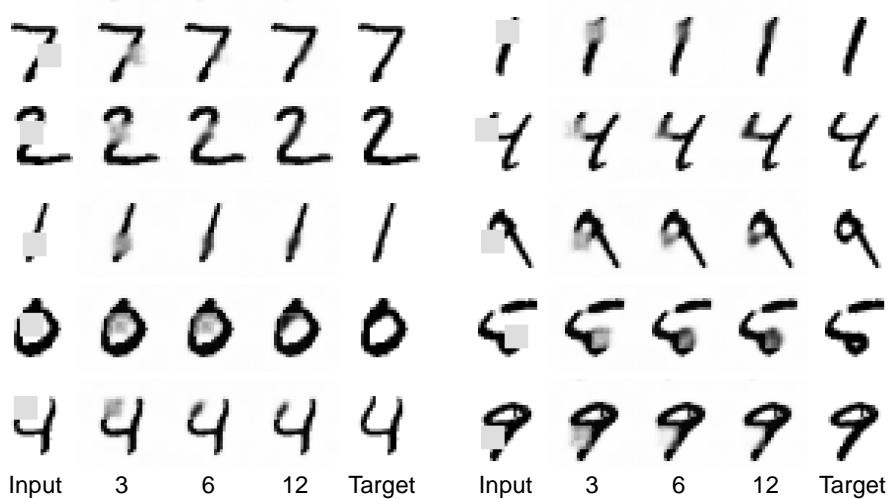
Fig. 9.14. Filling-in of occlusions. The activities of the network's outputs are shown over time for the first ten test images. The recurrent network is able to remove the square, and it produces a reasonable guess of the digit's appearance.
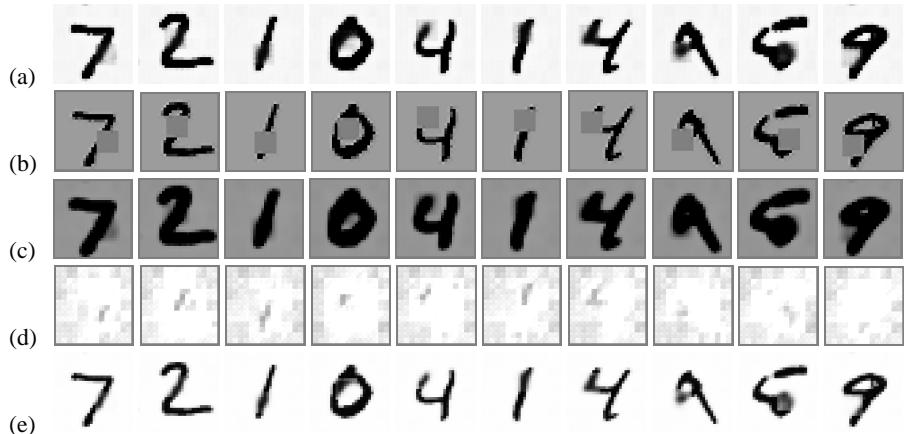
Fig. 9.15. Filling-in of occlusions: (a) hidden feature array in Layer 0; (e) output feature array; contributions to the output activity (b) via input projections; (c) via lateral projections (lines excite themselves and their neighborhood); (d) via backward projections (note the lack of inhibition at the filled-in line segments).

is such that a reasonable guess of the digit's appearance behind the occluder is produced. The network reconnects lines, interrupted by the square. It is also able to extend shortened lines, to close opened loops, and to synthesize corners, junctions, and crossings. In most cases, the reconstructions are very similar to the originals. Since the network never saw the original digits, it obviously has acquired some knowledge about the appearance of typical digits during training.

Figure 9.15 shows the activities of the single hidden feature in the bottom layer and the output feature array together with its contributions for the same digits after twelve iterations. One can observe that the hidden units are more active than the output units. They seem to represent potential lines. The occluding square is still visible in this feature array since it could hide lines.

The contributions from the input projections to the output feature cells are mainly excitatory. They look like a copy of the input and contain the square. Weak inhibition occurs at the edges of the lines and the square. The contributions of lateral projections are strongly excitatory. Lines excite themselves and their surroundings.

More interesting are the contributions via backward projections. They are strongly inhibitory everywhere, except for the border, where no inhibition is necessary, and the filled-in line segments. Hence, a detailed description of these segments must exist at the higher layers of the network.

To quantitatively evaluate the network's performance, the mean squared reconstruction error and the mean squared output changes were computed for the entire training set and all test examples. Both are displayed over time in Figure 9.16. The curves for the two sets are almost identical, indicating good generalization.

The reconstruction error drops quickly during the first iterations and remains low afterwards. It does not reach zero since a perfect reconstruction is frequently not possible due to the information loss caused by the occlusion. All the network
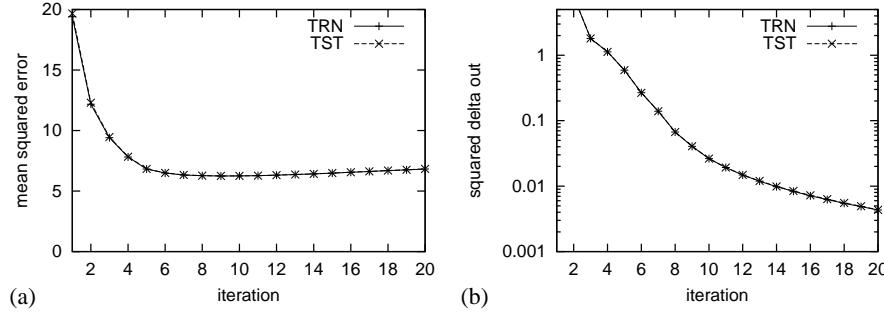
Fig. 9.16. Filling-in of occlusions – performance over time: (a) mean squared error; (b) mean square delta output.

can do is to produce a reasonable guess about the portion of the digit behind the square. No perfect reconstruction should be expected.

The vanishing changes of the output units prove that the network activities indeed converge to an attractor even when iterated further than the twelve time steps it was trained for.

9.4 Noise Removal and Contrast Enhancement

Low image contrast, a varying background level, and noise are common when capturing real-world images. While some of these sources of degradation can be reduced by controlling the setup, e.g. by providing homogeneous lighting, other factors cannot be compensated for. One example is the dark structured paper of larger envelopes that leads to degraded images of the address or other labeling. Image processing can try to reduce noise and improve contrast in order to ease subsequent recognition steps. The challenging aspect of this task is to separate the noise one wants to remove from the objects one wants to amplify.

A large number of methods for image denoising have been proposed in the literature. Only a few can be mentioned here. For example, Malladi and Sethian [150] proposed a level-set method that moves the iso-intensity lines of the image's gray level mountains according to their curvature. The method smoothes contours in a hierarchical fashion. A min/max-flow criterion is used to stop the algorithm. When applied to handwriting, only the larger strokes survive smoothing. They are bounded by sharp edges.

Hierarchical image decompositions using wavelets have been successfully applied to image denoising. Examples are the systems described by Simoncelli and Adelson [213] and by Donoho and Johnstone [56]. They transform the image into a multiscale representation and use the statistics of the coefficients of this representation to threshold them. The back-transformed images are then less noisy. This method works well when the wavelets used match the structure of typical image details, such as edges or lines. What is problematic with these approaches is that
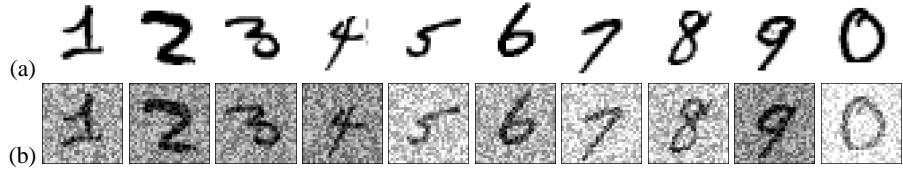
Fig. 9.17. Some examples from the MNIST dataset: (a) original images; (b) degraded with reduced contrast, random background level, noise, and saturation.

the choice of the wavelet transformation is usually fixed, and that the thresholding ignores dependencies between neighboring locations within a scale and between scales.

Yang *et al.* [246] demonstrated that a linear auto-associator can be used for denoising faces if a wavelet representation is used. The output of the system has a high perceptual quality if the noise level is moderate. However, since the system is linear, no hard decisions can be made, and the network's output resembles overlayed faces if the noise level is high. Hence, the need for non-linear behavior becomes obvious.

9.4.1 Image Degradation

In the next experiment, it is demonstrated that the same method, which was useful for filling-in occluded parts, can be applied to noise removal as well. The same network architecture and the same MNIST digits are used to simultaneously learn noise removal and contrast enhancement. The only difference is the image degradation procedure.

To degrade the images, the pixel intensities are scaled from the interval $[0, 1]$ to the range $[0.25, 0.75]$. This reduces image contrast. To simulate variance in lighting and paper brightness, a random background level is added that is uniformly distributed in the range $[-0.25, 0.25]$. Additive uniform pixel noise, drawn from $[-0.25, 0.25]$, simulates sensor noise. Finally, the pixel intensities are clipped at zero and one to resemble sensor saturation. Figure 9.17 shows some digits from the MNIST dataset that have been degraded in this way.

The network was trained to reconstruct the original images on a working set of increasing size for twelve time steps using BPTT and RPROP. A low-activity prior was used to encourage the development of sparse representations.

9.4.2 Experimental Results

Figure 9.18 illustrates the reconstruction process of the trained network for a test example. The activity of the entire network is shown over time. One can observe that all features contribute to the computation. In the first time steps, the confidence of the digit's lines visible in the bottom layer is not very high since locally they look similar to some background structures. The confidence increases as more context influences the result. Finally, background clutter is removed from the output feature array, and the lines are amplified.
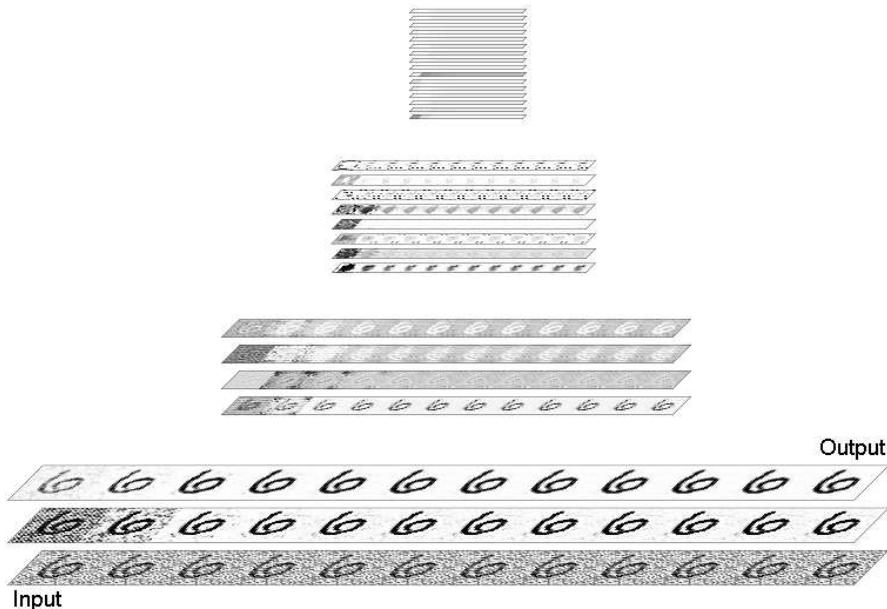
Fig. 9.18. Noise removal and contrast enhancement. The activities of all feature arrays are shown over time. A hierarchical distributed representation of the digit stabilizes. It is used to amplify the lines and to remove background clutter.

A distributed representation of the digit stabilizes in the hierarchical network. Most interesting is the representation of the background level. Three features in Layer 1, one feature in Layer 2, and one feature in Layer 3 seem to estimate the background intensity. The remaining feature arrays form a sparse representation of other digit features.

The progress of the reconstruction process is shown in Figure 9.19 for the first ten test examples. One can observe that the network is able to detect dark lines, to complete them, to remove background clutter, and to enhance the contrast. The interpretation at most locations is decided quickly by the network. Ambiguous locations are kept for some iterations at intermediate values, such that the decision can be influenced by neighboring locations. The reconstructed digits are very similar to the originals.

To illustrate the network's solution to the reconstruction problem, Figure 9.20 shows the activities of the single hidden feature in the networks's bottom layer and the output feature array together with its contributions for the same digits after twelve iterations.

One can observe that the hidden units are less confident than the output units. The hidden feature array seems to represent potential lines. Some background structures are still visible there since adjacent dark pixels could be caused by the presence of a line, rather than by noise.
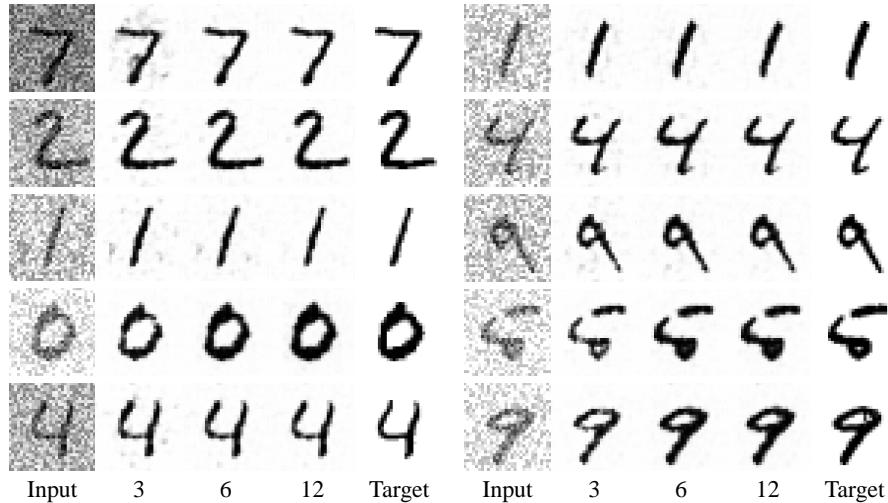
Fig. 9.19. Noise removal and contrast enhancement. The activities of the network's outputs are shown over time. The stable network outputs approximate the targets.

The contributions from input projections to the output units are mainly excitatory. They look like a low-pass copy of the input and contain the background level as well as the noise.

The lateral contributions are mostly excitatory. Lines excite themselves and their surroundings. A wider neighborhood of the lines is inhibited weakly.

Most interesting are the contributions via backward projections. They vary with the background level. Images with darker background receive more inhibition than brighter images. The inhibition is distributed quite uniformly, indicating the existence of a global background level estimate in the higher layers. Exceptions are the lines and the image borders. The network has learned that the lines are more probable in the center of the image than at its border.

Figure 9.21 shows the network's performance over time for the entire dataset. The output error falls rapidly to a lower level than in the occlusion experiment and remains low. This implies that occlusions represent a more severe degradation than low contrast combined with noise. As before, generalization is good, and the network converges to an attractor representing the reconstruction.

9.5 Reconstruction from a Sequence of Degraded Digits

The Neural Abstraction Pyramid architecture is not restricted to reconstruct static images from static inputs. Since the networks are recurrent, they are also able to integrate information over time. Thus, if image sequences are available, they can be used to improve the reconstruction quality.

It has been demonstrated by other researchers that video material can be reconstructed with higher quality when taking into account neighboring frames than
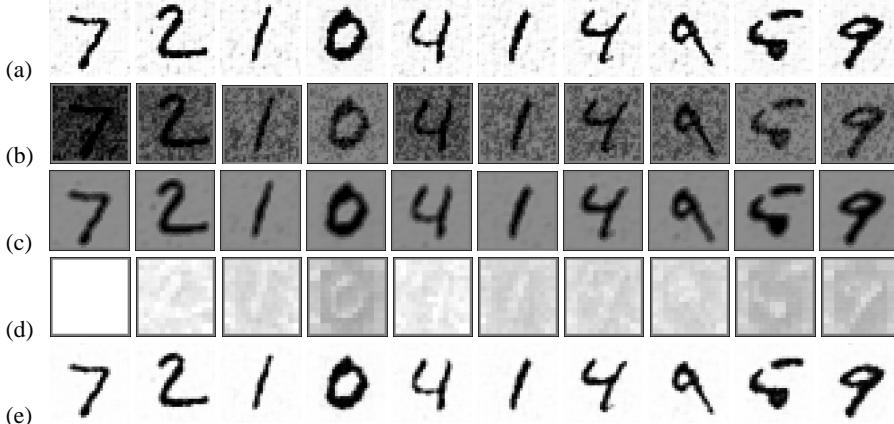
Fig. 9.20. Noise removal and contrast enhancement: (a) hidden feature array in Layer 0; (e) output feature array; contributions to the output activity (b) via input projections; (c) via lateral projections (center-surround interaction); (d) via backward projections (indicating an estimate of the background intensity and the line positions in higher layers).
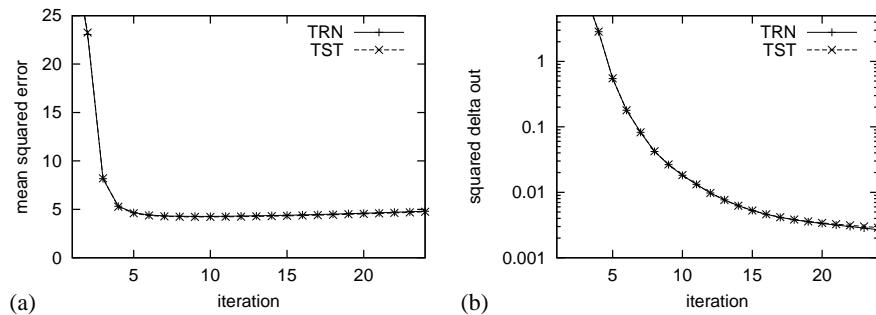
Fig. 9.21. Noise removal and contrast enhancement – performance over time: (a) mean squared error; (b) mean square delta output.

would be possible by considering isolated frames only. For instance, Elad and Feuer [62] proposed a least squares adaptive filtering approach to increase the resolution of continuous video.

Another example is the work of Kokaram and Godsill [127]. They proposed a Bayesian approach to the reconstruction of archived video material. Using Markov chain Monte Carlo methods, they simultaneously detect artifacts, interpolate missing data, and reduce noise.

9.5.1 Image Degradation

In the next experiment, the capability of the same Neural Abstraction Pyramid network, used for the previous two tasks, to reconstruct digits from a sequence of degraded images is explored. The only difference is the image degradation procedure.
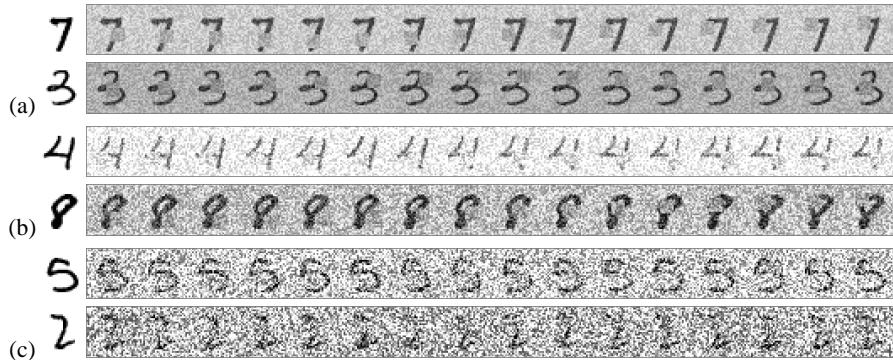
Fig. 9.22. MNIST example sequences of degraded digits. A random background level, contrast reduction, occlusion, and pixel noise were combined: (a) low noise; (b) medium noise; (c) high noise.

A random background level, contrast reduction, occlusion, and pixel noise are combined to produce input sequences.

For each sequence, a random movement direction is selected for the occluding square from 8 possible choices that form an 8-neighborhood. The square moves with a speed of one step per iteration until it is reflected near an image border. Thus, most digit parts are visible at some point of the sequence, and the network has the chance to remember the parts seen before occlusion.

Three variants of the training set and the test set are produced with different degrees of degradation. For the low-noise variant, the uniform pixel noise is chosen from the interval $[-0.125, 0.125]$. The medium noise comes from the range $[-0.25, 0.25]$, and the high noise from $[-0.5, 0.5]$. The background level is chosen from $[-0.125, 0.125]$ for the low-noise variant and from $[-0.25, 0.25]$ for the medium and high-noise variants. Each training set consists of 10,000 randomly selected examples from the MNIST dataset.

Two examples of the low, the medium, and the high-noise sequences are shown in Figure 9.22 next to the original digit. While the low-noise sequences are easily readable for humans, the medium-noise sequences are challenging. The high-noise sequences are so badly degraded that they are almost unreadable when looking at a single image alone. Here, the need to integrate over multiple images of the sequence becomes obvious.

The network was trained separately on the three noise variants. Training was done as in the previous experiments, except this time 16 iterations were used, as determined by the length of the input sequences.

9.5.2 Experimental Results

Figure 9.23 shows the reconstruction of a high-noise test digit performed by the trained network. The activity of the entire network is shown over time. One can observe that all features contribute to the computation. The lines of the digits, visible
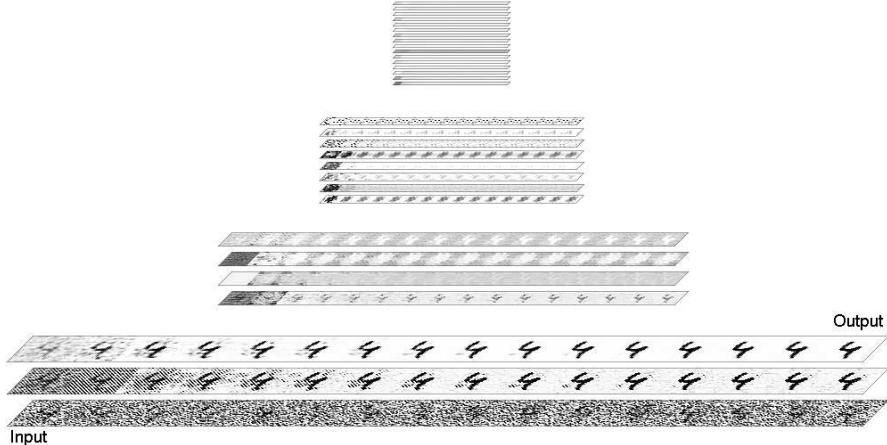
Fig. 9.23. MNIST reconstruction from sequence of degraded images. The activities of all feature arrays are shown over time.
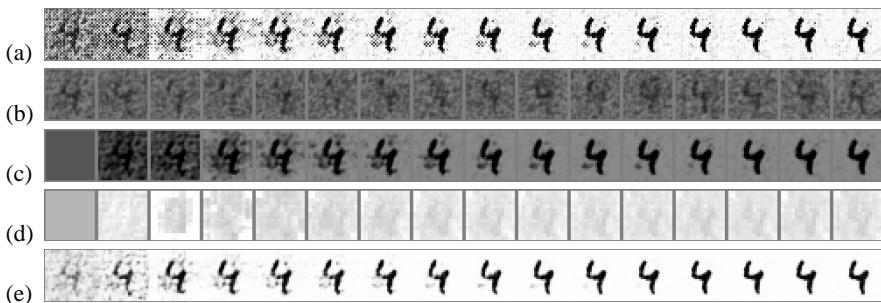
Fig. 9.24. Reconstruction from sequence of degraded images. For the example of Fig. 9.23 are shown over time: (a) hidden feature array in Layer 0 (representing line-candidates); (e) output feature array (the reconstructed digit); contributions to the output activity (b) via input projections (noisy); (c) via lateral projections (center-surround interaction); (d) via backward projections (inhibiting the background stronger than the lines).

in the bottom layer, need more time steps than in the previous experiments to reach a high confidence level. Towards the end of the sequence, the background clutter and the occluding square have been removed by the network from the output feature array, and the lines have been completed. The network's activities converge to a distributed representation of the digit which facilitates the reconstruction of the original.

In Figure 9.24, the activities of the single hidden feature array in Layer 0 and the output feature array, together with its contributions are shown for the same digit over time. The hidden feature is more active than the output feature, representing line candidates. Background structures are highly visible in the first few iterations. They are reduced towards the end of the sequence.

The contributions from the input to the output feature array are weakly excitatory with a low-pass characteristic. Input noise, the background level, and the occluding square are highly visible here.

The contributions from lateral projections are strongly excitatory in the center and weakly inhibitory in the surroundings. Hence, lines excite themselves and their immediate neighborhood, and inhibit their surround.

The contributions via backward projections seem again to inhibit the output feature according to the estimated background level. Interesting is the strong inhibition of the units near the image border at iteration three. This is the first step where information from Layer 3, which has a global view of the image, reaches the output. Towards the end of the sequence, the inhibition is weaker at the units belonging to lines. This shows that lines are represented at higher layers.

Figures 9.25, 9.27, and 9.29 display the reconstruction process of the first ten test digits for low noise, medium noise, and high noise, respectively.

One can see that in all three cases the network is able to produce good reconstructions of the originals, which are also shown in the figures. The less ambiguous image parts are reconstructed faster than the parts that are occluded. The higher the noise is or the stronger the background level deviates from the mean, the more iter-

(a) 1   2   4   7   11   16   (b) 1   2   4   7   11   16   (c)
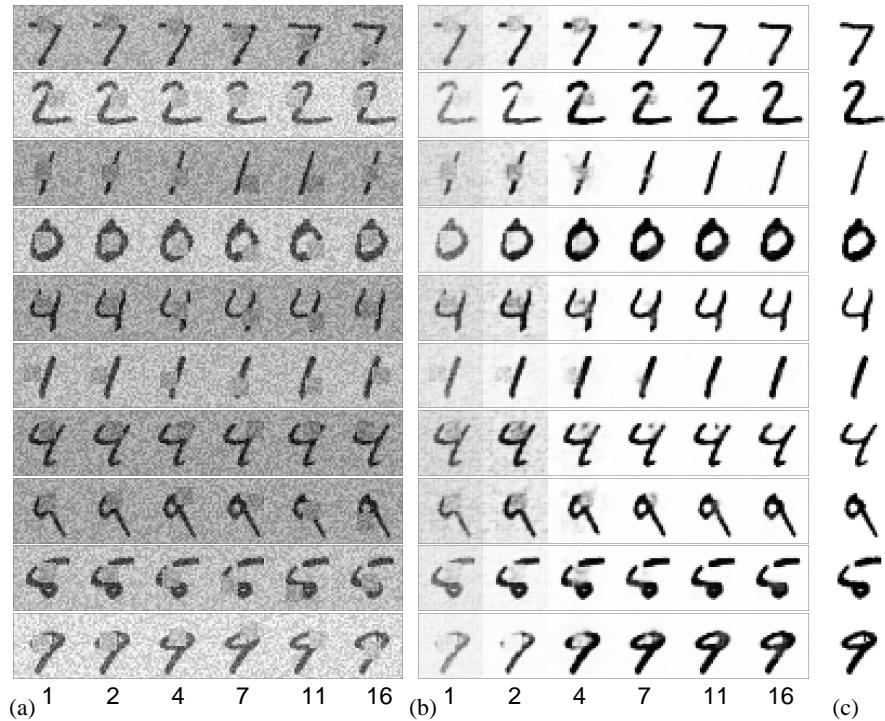
Fig. 9.25. Reconstruction from a sequence of degraded MNIST images (low noise): (a) input over time; (b) output over time; (c) target.
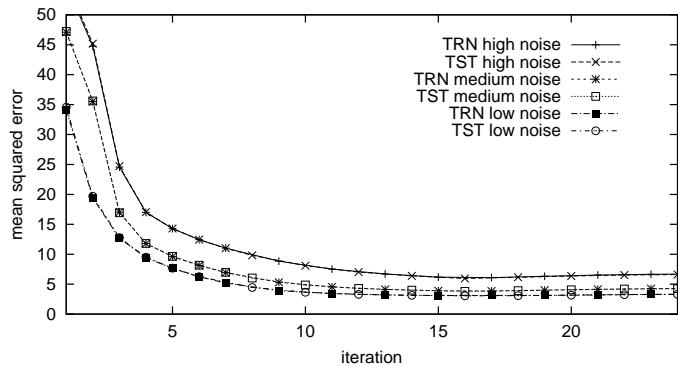
Fig. 9.26. Reconstruction error over time for the reconstruction from a sequence of degraded MNIST digits. Performance for the training set (TRN) and the test set (TST) is very similar. The error drops fast in the first iterations and stabilizes later. Higher noise levels cause higher reconstruction errors.
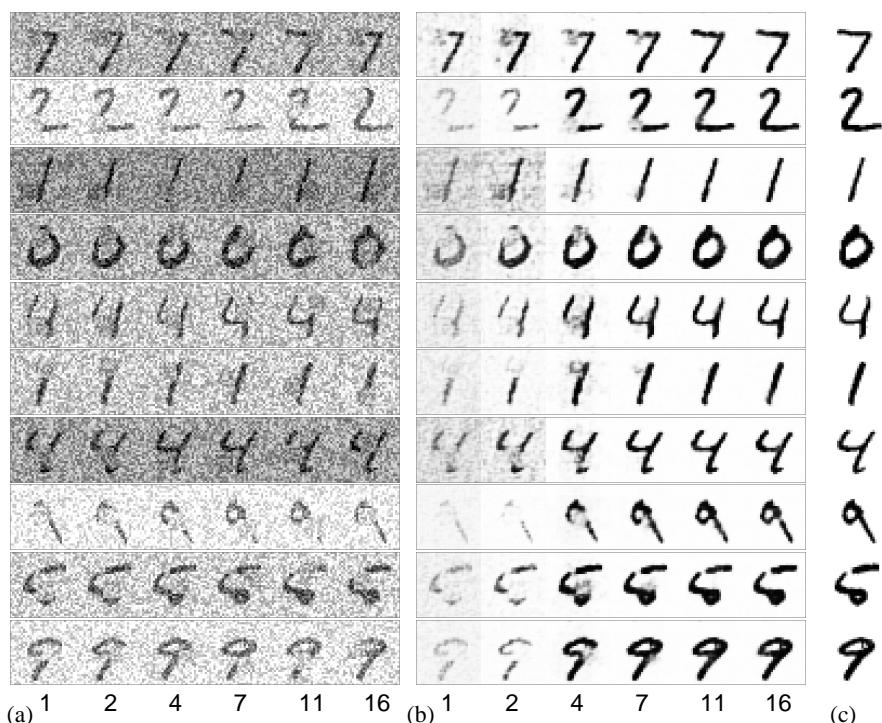
Fig. 9.27. Reconstruction from a sequence of degraded MNIST images (medium noise): (a) input over time; (b) output over time; (c) target.
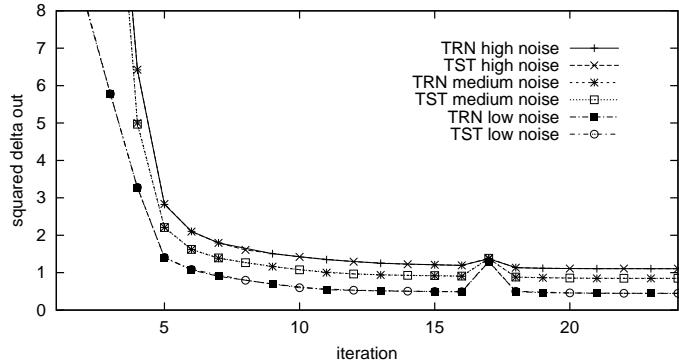
Fig. 9.28. Output changes over time for the reconstruction from a sequence of degraded MNIST digits. Performance for the training set (TRN) and the test set (TST) is very similar. The outputs change most during the first iterations. Output changes are higher for high noise than for lower noise. The bump at iteration 17 reflects a sudden change in the input sequence that starts again with the first frame.
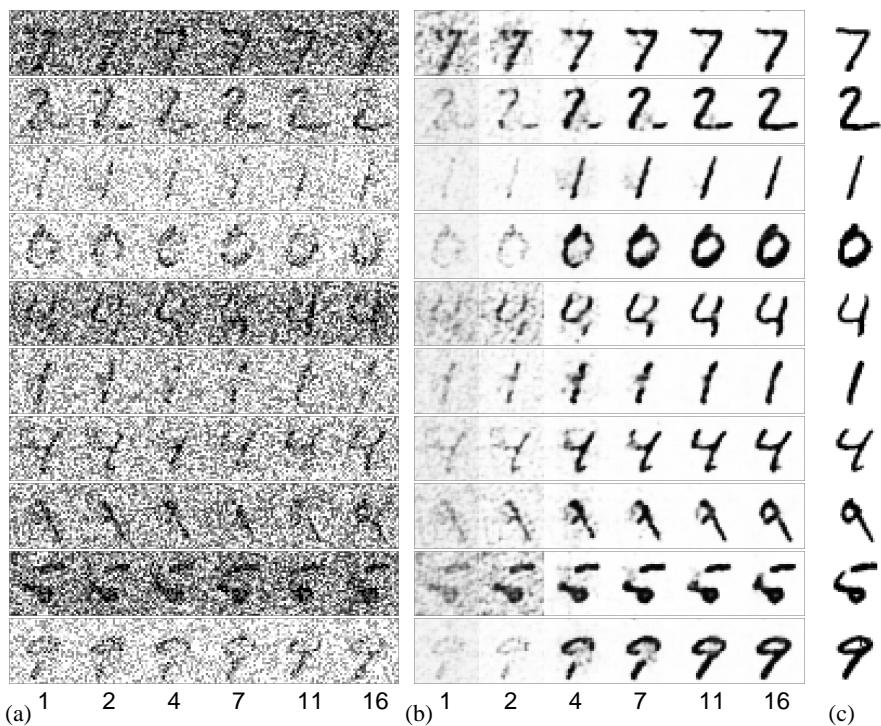
Fig. 9.29. Reconstruction from a sequence of degraded MNIST images (high noise): (a) input over time; (b) output over time; (c) target.
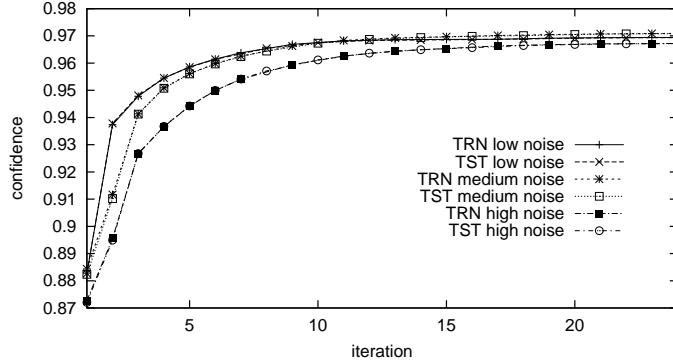
Fig. 9.30. Confidence over time for the reconstruction from a sequence of degraded MNIST digits. Performance for the training set (TRN) and the test set (TST) is very similar. Confidence increases most during the first iterations. For higher noise, confidence rises slower and reaches a lower level than for lower noise.

ations are needed for reconstruction. Towards the end of the sequences, the output changes are small.

To quantitatively evaluate the performance of the networks, the reconstruction error, the output changes, and the output confidences were computed for all image sequences. In all cases, the test set performance is very similar to the performance on the training set, indicating good generalization.

In Figure 9.26, the mean squared reconstruction error of the training set and the test set is displayed over time for the three noise variants. The reconstruction error decreases monotonically until it reaches a level where it remains flat even when iterated longer than the 16 iterations the networks were trained for. The higher the noise level is, the slower the error drops and the higher the final error level is.

Figure 9.28 shows the mean squared changes of the output units. The general behavior is similar to the output error. The changes drop quickly during the first iterations and decrease more and more slowly. One exception is the bump visible at iteration 17. It is caused by a jump of the occluding square in the input image that returns to its initial position after the end of the 16 step sequence. This behavior shows that the networks are still sensitive to changes in the input and are not locked to attractors independent of the input.

Finally, the average output confidences are shown in Figure 9.30. The higher noise network variations remain less confident for a longer time and reach a lower confidence level than the lower noise variations.

9.6 Conclusions

The experiments in this chapter show that difficult non-linear image reconstruction tasks can be learned by instances of the Neural Abstraction Pyramid architecture. Supervised training of the networks was done by a combination of BPTT and

RPROP. The same network was trained to preform different tasks, as specified by different image degradation procedures.

The networks reconstruct images iteratively and are able to resolve local ambiguities by integrating partial results as context. This is similar to the recently demonstrated belief propagation in graphical networks with cycles. The main difference is that the approach proposed here learns horizontal and vertical feedback loops that produce rich multiscale representations to model the images, whereas current belief propagation approaches use either trees or arrays to represent either the vertical or the horizontal dependencies, respectively.

Furthermore, the proposed network can be trained to compute an objective function directly, while inference in belief networks with cycles is only approximate due to the multiple counting of evidence. Recently, Yeddia, Freeman and Weiss proposed generalized belief propagation [247] that allows for better approximations of the inference process. It would be interesting to investigate the relationship between this approach and the hierarchical recurrent neural networks.

The iterative reconstruction is not restricted to static images. In this chapter it was shown that the recurrent Neural Abstraction Pyramid network is able to integrate information over time in order to reconstruct digits from a sequence of images that were degraded by random background level, contrast reduction, occlusion, and pixel noise. The training method allows also for a change of the desired output at each time step. Thus, the networks should be able to reconstruct video sequences.

10. Face Localization

One of the major tasks in human-computer interface applications, such as face recognition and video-telephony, is the exact localization of a face in an image.

In this chapter, I use the Neural Abstraction Pyramid architecture to solve this problem, even in presence of complex backgrounds, difficult lighting, and noise. The network is trained using a database of gray-scale still images to reproduce manually determined eye coordinates. It is able to generate reliable and accurate eye coordinates for unknown images by iteratively refining an initial solution.

The performance of the proposed approach is evaluated against a large test set. It is also shown that a moving face can be tracked. The fast network update allows for real-time operation.

10.1 Introduction to Face Localization

To make the interface between humans and computers more pleasant, computers must adapt to the users. One prerequisite for adaptation is that the computer perceives the user. An important step for many human-computer interface applications, like face recognition, lip reading, reading of the users emotional state, and video-telephony, is the localization of the user's face in a captured image. This is a task humans can perform very well, without perceiving effort, while current computer vision systems have difficulties.

An extensive body of literature exists for face detection and localization problems. Recently, Hjelmas and Low [99] published a survey on automatic face detection methods. They distinguish between feature-based and image-based methods.

Feature-based methods are further classified as either relying on low-level features, such as edges, motion and skin color, as searching for higher-level features, such as a pair of eyes, or as using active shape models, such as snakes or deformable templates.

An example of a feature-based method that uses edges is the approach taken by Govindaraju [83], where edges are extracted, labeled, and matched against a predefined face model. A similar system is described by Jesorsky *et al.* [111]. It consists of an edge extraction stage, a coarse localization that uses a face model, a fine localization that relies on an eye model, as well as a multi-layer perceptron for the exact localization of the pupils.

The system described by Maio and Maltoni [149] consists of three stages. The first stage approximately locates all elliptical objects in a directional image using a generalized Hough transformation [13]. The second stage improves the localization accuracy by using a local optimization of the ellipse's position and size. Finally, the third stage checks whether the objects found are faces by comparing vertical and horizontal projections of directions to a face model.

Many face localization techniques rely on skin color. Terrillon *et al.* [225] provide a comparative study on the utility of different chrominance spaces for this task. Motion information is also useful for face detection. Differences between consecutive frames are likely to be large at the boundaries of moving objects. Spatio-temporal contour filters can also be applied to extract object boundaries. Another example for the use of motion is the approach taken by Lee *et al.* [136]. They compute the optical flow, segment moving face regions using a line-clustering algorithm, and use ellipse fitting to complete the extraction of the face region.

Color and motion features are strong hints for the presence of a face. However, these low-level features are not always available. Furthermore, each low-level feature is likely to be ambiguous since a variety of non-face objects, potentially present in the analyzed images, can trigger them as well. Thus, it may be necessary to use higher-level features.

An example of a face localization method that employs the relative positioning of facial features is the one proposed by Jeng *et al.* [110]. They initially try to establish possible eye locations in binarized images. For each possible eye pair, the algorithm goes on to search for a nose, a mouth, and eyebrows. The system described by Yow and Cipolla [248] employs a probabilistic model of typical facial feature constellations to localize faces in a bottom-up manner.

Unlike the face models described above, active shape models depict the actual physical appearance of features. Once released within close proximity to a face, an active shape model will interact with local image features (edges, brightness) and gradually deform to take the shape of the face by minimizing an energy function.

Several methods use snakes, first introduced by Kass *et al.* [120]. Cootes *et al.* [45] recently proposed the use of a generic flexible model which they called active appearance model. It contains a statistical model of the shape and gray-level appearance of the object of interest which can generalize to almost any valid example. During a training phase the relationship between model parameter displacements and residual errors induced between a training image and a synthesized example are learned.

In contrast to feature-based methods, image-based approaches handle face detection as a pattern recognition problem. They analyze an image window that has been normalized to a standard size and then classify the presence or absence of a face. Linear subspace methods apply linear statistical tools, like principal component analysis (PCA), linear discriminant analysis (LDA), and factor analysis (FA) to model facial images. For example, Moghaddam and Pentland [160] proposed a face detection technique based on a distance measure from an eigenface model.
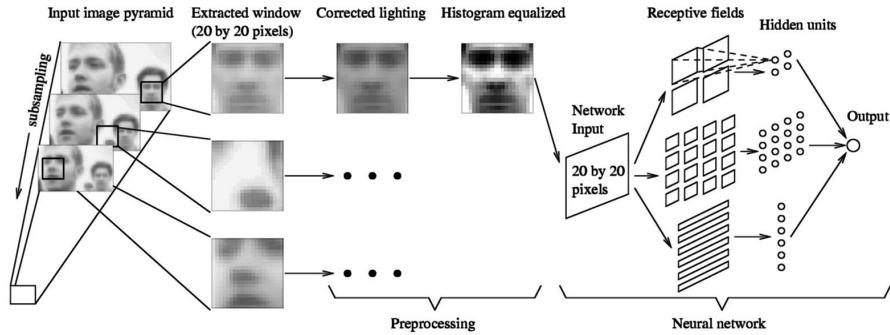
Fig. 10.1. Face detection system proposed by Rowley *et al.* (image adapted from [198]).

Sung and Poggio [221] first proposed the use of a mixture of Gaussians to model faces. They give distances to face cluster centroids as input to a multi-layer perceptron (MLP), trained for face/non-face classification.

Neural networks are a popular technique for pattern recognition problems including face detection. The first advanced neural approach which reported results on a large, difficult dataset was published by Rowley *et al.* [198]. Their system incorporates face knowledge in a retinotopically connected neural network, shown in Fig. 10.1. The neural network is designed to analyze windows of 20×20 pixels. There is one hidden layer with 26 units, where 4 units access 10×10 pixel subregions, 16 look at 5×5 subregions, and 6 receive input from overlapping horizontal stripes of size 20×5. The input window is preprocessed through lighting correction and histogram equalization. Recently, Rowley *et al.* [199] combined this system with a router neural network to detect faces situated at all angles in the image plane.

Apart from linear subspace methods and neural networks, there are several other statistical approaches to image-based face detection, like systems based on information theory or support-vector machines. For example, Schneiderman and Kanade [206] use products of histograms of wavelet coefficients. They employ multiple views to detect 3D objects like cars and faces in different poses. Support vector machines are used e.g. by Heisele *et al.* [92]. They describe a one-step detector for entire faces and a component-based hierarchical detector.

Searching for feature combinations, matching features with translated, rotated, and scaled face models, as well as scanning windows over all positions and scales are time-consuming procedures that may limit the applicability of the above methods to real-time tasks. Furthermore, heuristics must be employed to prevent multiple detections of the same face at nearby locations or scales.

In the following, a method is described that uses an instantiation of the Neural Abstraction Pyramid architecture, introduced in Chapter 4, to localize a face in gray-scale still images. The network operates by iteratively refining an initial solution. Multiresolution versions of entire images are presented directly to the network, and it is trained with supervision to localize the face as fast as possible. Thus, no

Fig. 10.2. Some face images from the BioID dataset. Since the examples vary considerably, the dataset can be considered challenging.

scanning through parameter spaces is needed, and multiple detection candidates cooperate and compete with each other to produce a coherent image interpretation.

10.2 Face Database and Preprocessing

To validate the performance of the proposed approach for learning face localization, the BioID database [111] is used. The database can be downloaded free of charge from http://www.bioid.com/downloads/facedb/facedatabase.html. It consists of 1,521 images that show 23 individuals in front of various complex office backgrounds with uncontrolled lighting. The persons differ in gender, age, and skin color. Some of them wear glasses and some have beards. Since the face size, position, and view, as well as the facial expression vary considerably, the dataset can be considered challenging.

Such real-world conditions are the ones that show the limits of current localization techniques. For instance, while the hybrid localization system, described in [111], correctly localizes 98.4% of the XM2VTS database [157] which has been produced under controlled conditions, the same system localizes only 91.8% of the BioID faces. Figure 10.2 shows some example images from the BioID dataset.

The gray-scale BioID images have a size of 384×288 pixels. To reduce border effects, the contrast is lowered towards the sides of the image. To limit the amount of data, the image is subsampled to 48×36, 24×18, and 12×9 pixels, as shown in Figure 10.3(b). In addition to the images, manually labeled eye positions $C_l, C_r \in \mathbb{R}^2$ are available. They are in general quite reliable but not always as accurate as one could hope.

Figure 10.3(a) shows the marked eye positions for a sample image. A multi-resolutional Gaussian blob is produced for each eye in a set of images that have the above resolutions. The blobs are shown in Figure 10.3(b). Their standard deviation σ is proportional to the distance of the eyes $\|C_l - C_r\|$. Note that with increasing resolution, the area of the blob increases, with respect to the original image. Thus,
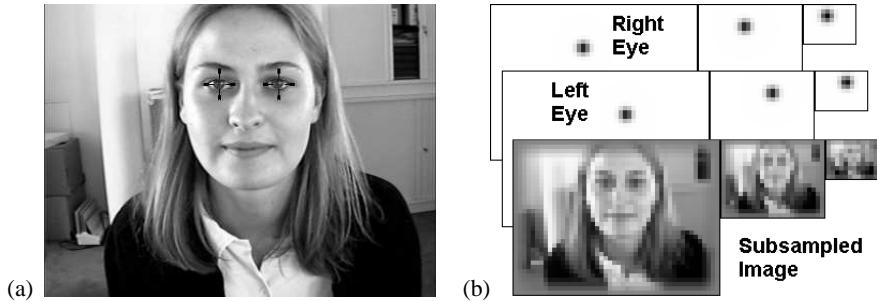
Fig. 10.3. Preprocessing: (a) original image with marked eye positions; (b) eye positions and subsampled framed image in three resolutions.

while the blobs in the highest resolution do not overlap, blobs in the lowest resolution do.

10.3 Network Architecture

The preprocessed images are presented to a hierarchical neural network, structured as a Neural Abstraction Pyramid. As shown in Figure 10.4, the network consists of four layers. The resolution of the layers decreases from Layer 0 (48×36) to Layer 2 (12×9) by a factor of 2 in both dimensions. Layer 3 has only a single hypercolumn. Each layer has excitatory and inhibitory feature arrays. The number of feature arrays
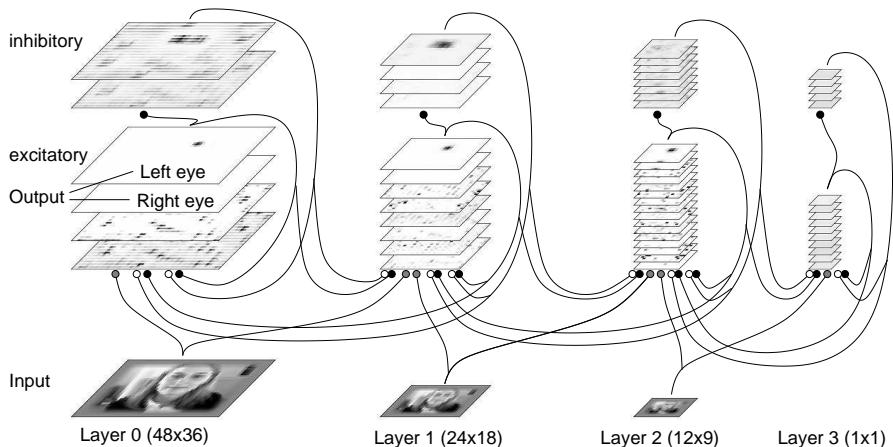
Fig. 10.4. Sketch of the network used for learning face localization. It is an instance of the Neural Abstraction Pyramid architecture. The network consists of four layers, shown from left to right. Each layer contains excitatory and inhibitory feature arrays. Excitatory projections are drawn with filled circles, open circles indicate inhibitory projections, and projections labeled with shaded circles can have any sign.

per layer increases when going from Layer 0 $(4 + 2)$ to Layer 2 (16+8). Layer 3 contains 10 excitatory and 5 inhibitory feature cells. In addition, an input feature array is present in all layers except the topmost one.

Most projections in the network are either excitatory or inhibitory. Weights in projections that access excitatory units are non-negative. Weights from inhibitory units are non-positive. In contrast, weights of projections accessing the input feature array can have any sign. They have a window size of $5{\times}5$ and lead to excitatory features in the same layer or belong to forward projections of excitatory feature cells in the next higher layer.

The excitatory feature cells of Layer 1 and Layer 2 receive forward projections from the $4{\times}4$ hyper-neighborhood in the layer below them. Connections between Layer 2 and the topmost Layer 3 are different since the resolution drops from $12{\times}9$ to $1{\times}1$. Here, the forward and backward projections implement a full connectivity between the excitatory feature cells of one layer and all feature cells of the other layer. The backward projections of Layer 0 and Layer 1 access all feature cells of a single hypercolumn in the next higher layer. $2{\times}2$ different backward projections exist for each excitatory feature. In all layers except the topmost one lateral projections access all features of the $3{\times}3$ hyper-neighborhood around a feature cell. In Layer 3 lateral projections are smaller because all feature cells are contained in a $1{\times}1$ hyper-neighborhood.

The projections of the inhibitory features are simpler. They access $5{\times}5$ windows of all excitatory feature arrays within the same layer. In Layer 3, of course, this window size reduces to $1{\times}1$. While all projection units have linear transfer functions, a smooth rectifying transfer function f_{st} ($\beta = 10$, see Fig. 4.6(a) in Section 4.2.4) is used for the output units of all feature cells.

The feature arrays are surrounded by a two pixel wide border. The activities of the border cells are copied from feature cells using wrap-around.

10.4 Experimental Results

Because the BioID dataset does not specify which images constitute a training set and a testing set, the dataset was divided randomly into 1000 training images (TRN) and 521 test examples (TST). The network was trained for ten iterations on random subsets of the training set with increasing size using backpropagation through time (BPTT) and RPROP, as described in Chapter 6. The weighting of the quadratic error increased linearly in time.

The two first excitatory feature arrays on the three lower layers are trained to produce the desired output blobs that indicate the eye positions. All other features are hidden. They are forced to have low mean activity.

Figure 10.5 shows the development of the trained network's output over time when the test image from Fig. 10.3 is presented as input. One can observe that the blobs signaling the locations of the eyes develop in a top-down fashion. After the first iteration they appear only in the lowest resolution. This coarse localization is used to bias the development of blobs in lower layers. After five iterations, the
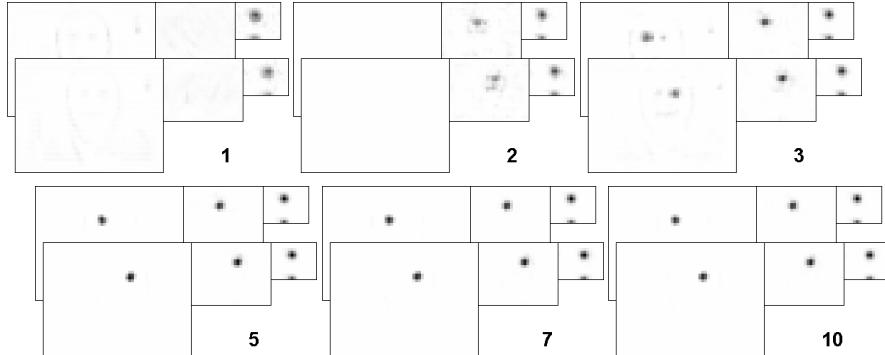
Fig. 10.5. Face localization recall. The activities of the network's output feature arrays are shown over time. Blobs indicating eye positions develop in a top-down fashion.

(a)   (b)   (c)
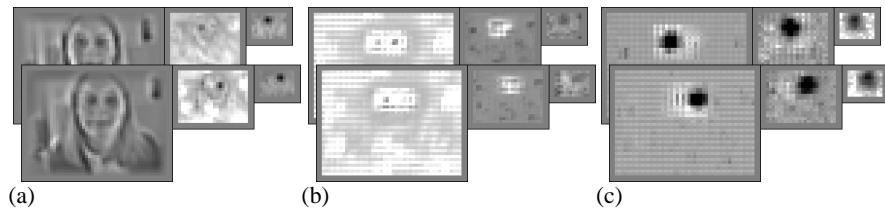
Fig. 10.6. Face localization recall. Shown are the contributions to the activity of the network's output units after iteration 10 of Fig. 10.5 (bright shading represents inhibition, dark shading indicates excitation): (a) via input and forward projections; (b) via lateral projections; (c) via backward projections.

network's output is close to the desired one. It does not change significantly during the next five iterations. Each iteration takes about 22ms on a Pentium 4 1.7GHz PC without much optimization for speed.

The contributions to the network's output activities at iteration 10 are displayed in Figure 10.6. It is evident that the main contribution to a blob's activity comes via backward projections. They excite a larger area at the eye's position in all three resolutions and inhibit its surround.

The effect of the lateral projections can be understood as center-center excitation and center-surround inhibition. Note that the blobs do not exist independently, but interact. This is most visible in the highest resolution, where both eye regions are inhibited and only the center of the opposite eye is weakly excited.

The influence of forward projections and projections from the input feature arrays is generally not that strong and less consistent. The input projections of Layer 0 seem to extract dark vertical lines that are surrounded by bright pixels from the image. In Layer 2 and Layer 3, the forward and input projections excite the center of the eye's blob and inhibit the blob of the opposite eye.

The network's dynamics cannot be understood by looking at the output feature arrays alone. The majority of the computations are done by hidden features. Figure 10.7 shows their activities after ten iterations for the test example from Fig. 10.3.
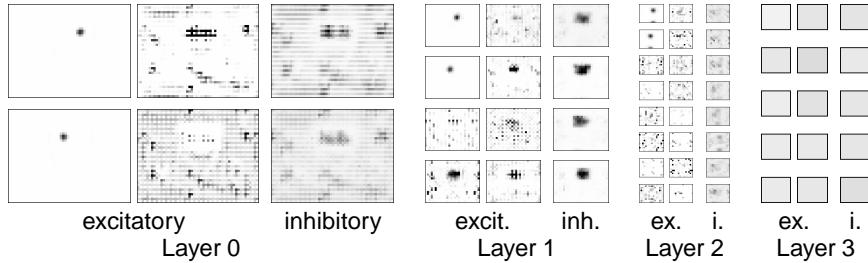
Fig. 10.7. Face localization recall. The activities of all feature arrays are shown after ten iterations when the test image from Fig. 10.3 is presented as input.

The activity pattern forms a sparse distributed multiscale representation of the image content. The hidden features clearly contribute to the development of stable blobs at the eye positions and to the suppression of output activity at other candidate locations, but they are hard to analyze.

The generation of stable blobs is the typical behavior of the network. To evaluate its performance, one has to estimate eye coordinates from the blobs and to compute a quality measure by comparison with the given coordinates.

The position of each eye was estimated separately, as illustrated in Figure 10.8. In a first step, the output unit with the highest activity v_{max} is found in the corresponding high resolution output. For all units in a $l \times l$ window around it, the feature cells belonging to the blob were segmented by comparing their activity with a threshold v_t that increases with greater distance d_{max} from the center and with the activity of the center v_{max}:

$$v_t = 0.5 \cdot v_{max} \cdot d_{max}/l. \tag{10.1}$$

The weighted mean location of the segmented cells is used as the estimated eye position. The figure shows the unproblematic segmentation of the test example from Fig. 10.3 as well as a more problematic case, where a secondary blob has not been removed by the iterative refinement, but is successfully ignored by the blob segmentation.
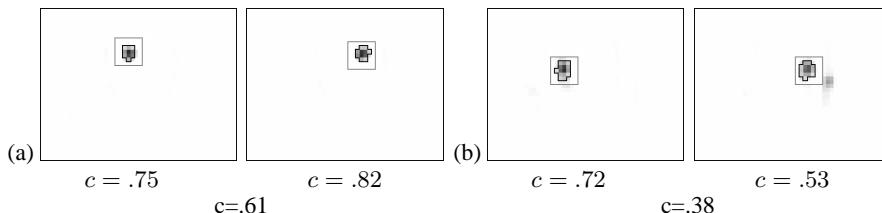
Fig. 10.8. Blob segmentation for face localization. A 7×7 segmentation window around the most active pixel is analyzed. The segmented pixels are framed by a black line: (a) output for example from Fig. 10.3; (b) output for a problematic example where a secondary blob has not been removed. This blob is ignored successfully by the segmentation procedure.
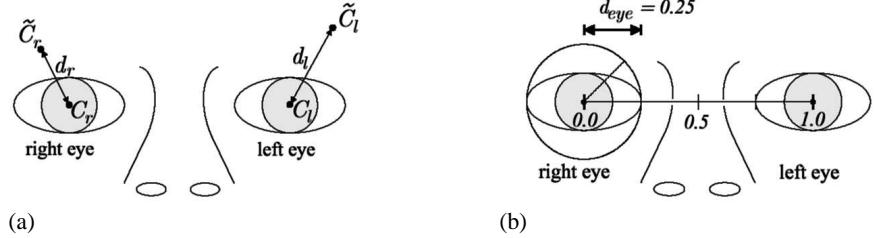
Fig. 10.9. Relative error measure for face localization, as suggested in [111]: (a) manually labeled eye positions (C_l, C_r), estimated eye positions $(\widetilde{C}_l, \widetilde{C}_r)$, and eye distances (d_l, d_r); (b) relative error d_{eye} from the right eye (shown left), a circle with radius $d_{eye} = 0.25$ is drawn around the eye.

After transforming these eye positions into the original coordinate system, a scale-independent relative error measure was computed, as suggested in [111]:

$$
\begin{aligned}
d_l &= \|\widetilde{C}_l - C_l\|, \\
d_r &= \|\widetilde{C}_r - C_r\|, \\
d_{eye} &= \max(d_l, d_r)/\|C_l - C_r\|. \qquad (10.2)
\end{aligned}
$$

The distances of the estimated eye positions $\widetilde{C}_l$ and $\widetilde{C}_r$ to the given coordinates C_l and C_r are denoted by d_l and d_r, respectively. A small relative distance of $d_{eye} < 0.25$ is considered a successful localization since $d_{eye} = 0.25$ corresponds approximately to the half-width of an eye, as illustrated in Figure 10.9.

The estimated eye coordinates, the given coordinates and the relative eye distances are shown in Figure 10.10 for the two test examples from Fig. 10.8. One can verify that for these examples the estimated eye positions are at least as exact as the given ones.

To test how the network is able to localize the other examples from the dataset, the relative distance d_{eye} was computed for all images. Figure 10.11 shows the network's localization performance for the training set (TRN) and the test set (TST)

$$d_{eye} = 0.0211 \qquad\qquad d_{eye} = 0.0107$$
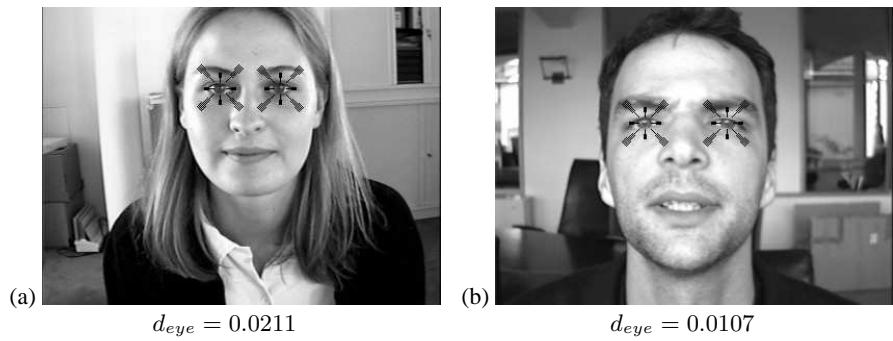
Fig. 10.10. Face localization output for the test examples from Fig. 10.8: $+$ mark the given eye coordinates; $\times$ are drawn at the estimated eye coordinates.
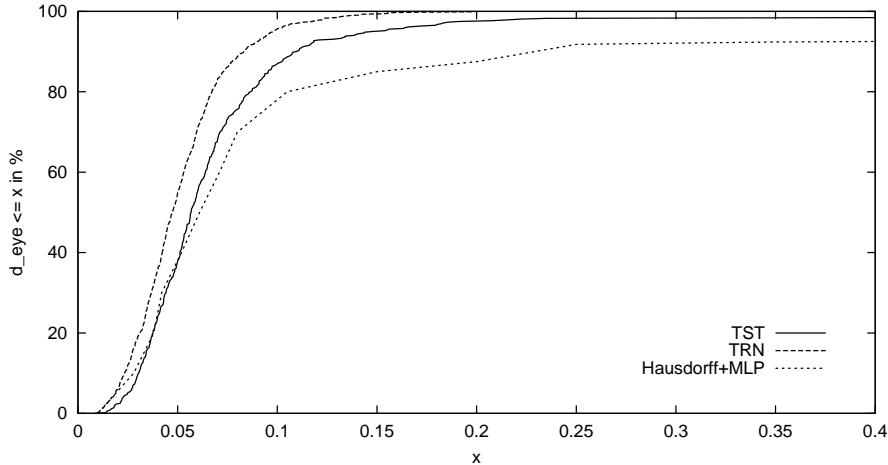
Fig. 10.11. Localization performance: percentage of examples having small d_{eye} for the proposed method (TRN, TST) and for the hybrid system (Hausdorff+MLP) [111].

in comparison to the data taken from [111] (Hausdorff+MLP). All training examples have been localized successfully. The performance on the test set is also good. Only 1.5% of the test examples have not been localized accurately enough. Compare this to the 8.2% mislocalizations in the reference system.

A detailed analysis of the network's output for the mislocalizations showed that in these cases the output is likely to deviate from the one-blob-per-eye pattern. It can happen that no blob or that several blobs are present for an eye.

By comparing the activity a_{blob} of a segmented blob to a threshold $a_{min} = 3$ and to the total activity of its feature array a_{total}, a confidence measure c is computed for each eye:

$$
\begin{aligned}
c_1 &= a_{blob}/a_{total}, \\
c_2 &= \left\{ \begin{array}{rcl} 1 &:& a_{blob} > a_{min} \\ a_{blob}/a_{min} &:& \text{else} \end{array} \right. , \\
c &= c_1 \cdot c_2.
\end{aligned}
\tag{10.3}
$$

The confidences of both eyes are multiplied to produce a single localization confidence. Since the faces in the BioID database are mainly in an upright position, the confidence is reduced if the blobs have a large vertical distance, compared to their horizontal distance. Figure 10.8 shows some example confidences. Figure 10.12 displays the confidence versus the relative eye distance d_{eye}. One can observe that high distances occur only for examples with low confidences. Furthermore, examples with high confidence values have low distances. Thus, the confidence can be used to reject ambiguous examples.

The localization confidence is compared to a reject threshold. In Figure 10.13, one can see that rejecting the least confident test examples lowers the number of
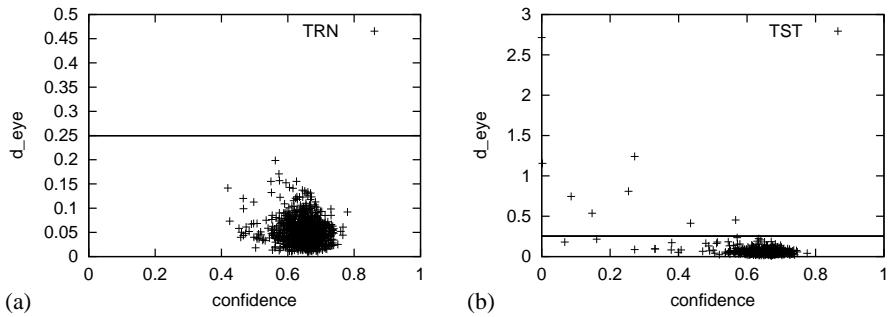
Fig. 10.12. Face localization performance. The confidences are shown versus the relative eye distance d_{eye} for: (a) the training set; (b) the test set.
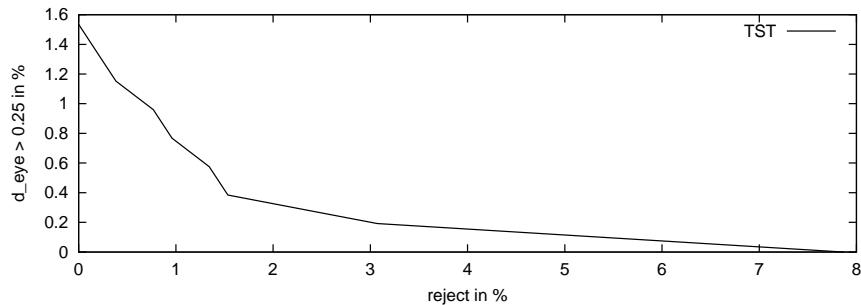
Fig. 10.13. Localization performance: rejecting the least confident examples lowers the number of mislocalizations.

$c = .000$ $c = .000$ $c = .004$ $c = .017$
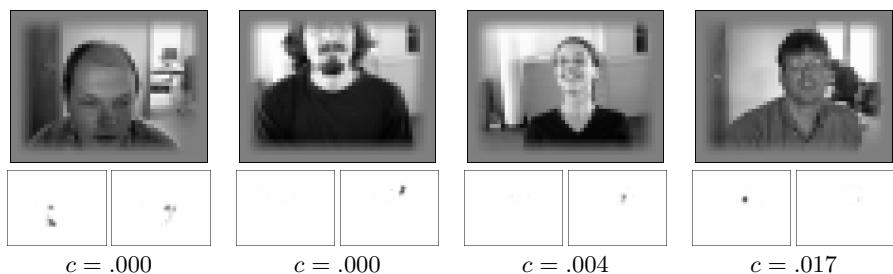
Fig. 10.14. Test examples with lowest confidences. The highest resolution network inputs, the outputs, and the confidences are shown.

mislocalizations rapidly. When rejecting 3.1% of the images, only one mislocalization is left. The average localization error of the accepted examples is $d_{eye} = 0.06$. That is well within the area of the iris and corresponds to the accuracy of the given eye coordinates.

Figure 10.14 shows the four test examples with the lowest confidences. In the leftmost example, the network has produced an extra blob for the mouth which prevents segmentation of the blob that corresponds to the eye. Since the person's face
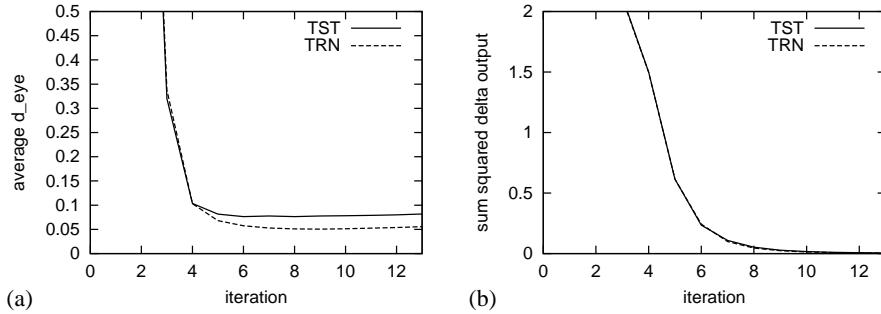
Fig. 10.15. Performance over time: (a) average distance d_{eye}; (b) sum of squared changes in the network's output.

in the second example is almost outside the image, the preprocessing has destroyed the upper part of the head, including the eyes. This leads to localization failure. The third example is difficult as well since the face appears relatively small and in an unusual posture. In the rightmost example, the network is probably distracted by the reflections on the glasses and produces a blob only for one of the eyes. The failure of the network to localize these faces correctly is not problematic since they can be rejected easily.

Figure 10.15 illustrates the network's performance over time. The average relative distance d_{eye} drops rapidly within the first five iterations and stays low afterwards. The average changes in the network's output are large during the first iterations and decrease to almost zero even when updated longer than the ten steps it has been trained for. Thus, the network shows the desired behavior of iterative refinement and produces stable outputs.

To investigate if the network is able to track a moving input, the test example from Figure 10.3 was translated with a speed of one pixel per iteration 40 pixels to the left, then 80 pixels to the right, and finally 40 pixels to the left. The left and right
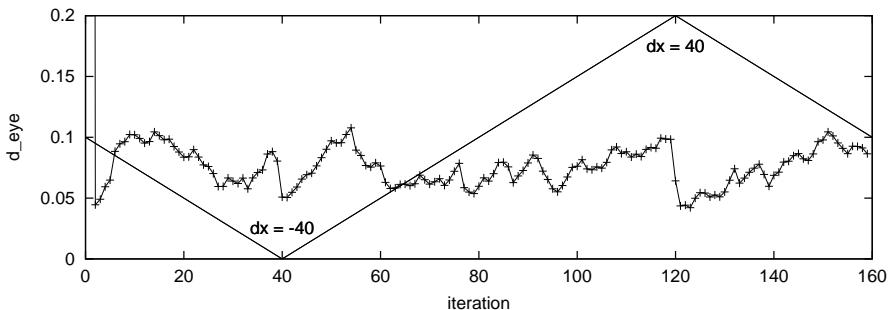
Fig. 10.16. Face localization recall with moving input. The test image from Fig 10.3 is moved 40 pixels to the left, 80 pixels to the right, and 40 pixels to the left. The relative distance d_{eye} of the network's output to the given moving eye positions is shown.

sides of the input images were wrapped around to fill the missing pixels. The moving input was presented to the network that was trained with static images, without any modifications. The network was able to move the output blobs along with the moving input. Thus, it tracks the eyes. Figure 10.16 shows the relative distance d_{eye} over time. After few iterations, the distance reaches a level of about 0.075. It varies around this value until the end of the sequence. Interesting are the steep drops after iterations 40 and 120, where the direction of movement is reversed. Here, the blobs catch up with the movement. Hence, the output blobs follow the input motion with a short delay.

10.5 Conclusions

In this chapter, an approach to face localization was presented that is based on the Neural Abstraction Pyramid architecture. The network is trained to solve this task even in the presence of complex backgrounds, difficult lighting, and noise through iterative refinement.

The network's performance was evaluated on the BioID dataset. It compares favorably to a hybrid reference system that uses a Hausdorff shape matching approach in combination with a multi-layer perceptron.

The proposed method is not limited to gray-scale images. The extension to color is straight forward. Since the network works iteratively, and one iteration takes only a few milliseconds, it would also be possible to use it for real-time face tracking by presenting image sequences instead of static images. It was demonstrated that the network is able to track a moving face.

11. Summary and Conclusions

11.1 Short Summary of Contributions

In order to overcome limitations of current computer vision systems, this thesis proposed an architecture for image interpretation, called Neural Abstraction Pyramid. This hierarchical architecture consists of simple processing elements that interact with their neighbors. The recurrent interactions are described be weight templates. Weighted links form horizontal and vertical feedback loops that mediate contextual influences. Images are transformed into a sequence of representations that become increasingly abstract as their spatial resolution decreases, while feature diversity as well as invariance increase. This process works iteratively. If the interpretation of an image patch cannot be decided locally, the decision is deferred, until contextual evidence arrives that can be used as bias. Local ambiguities are resolved in this way.

The proposed architecture defines a hierarchical recurrent neural network with shared weights. Unsupervised and supervised learning techniques can be applied to it. It turned out that the combination of the RPROP learning and backpropagation through time ensures stable and fast training, despite the difficulties involved in training recurrent neural networks.

The proposed architecture was applied to example problems, including the binarization of handwriting, local contrast normalization, and shift-invariant feature extraction. Unsupervised learning was used to produce a hierarchy of sparse digit features. The extracted features were meaningful and facilitated digit recognition.

Supervised learning was applied to several computer vision tasks. Meter values were recognized by a block classifier without the need for prior digit segmentation. The binarization of matrix codes was learned. The recurrent network discovered the cell structure of the code and used it to improve binarization.

The architecture was also applied for the learning of several image reconstruction tasks. Images were degraded and recurrent networks were trained to reproduce the originals iteratively. For a super-resolution problem, small recurrent networks outperformed feed-forward networks of similar complexity. A larger network was used for the filling-in of occlusions, the removal of noise, and the enhancement of image contrast.

Finally, the proposed architecture was used to localize faces in complex office environments. It developed a top-down strategy to produce blobs that indicate eye positions. The localization performance compared well to the hybrid system, pro-

posed by the creators of the database used in the experiments. The method is not restricted to static images. It was shown that a face could be tracked in real time.

11.2 Conclusions

The successful application of the proposed image interpretation architecture to several non-trivial computer vision tasks shows that the design patterns followed are advantageous for those kinds of problems.

The architectural bias of the Neural Abstraction Pyramid facilitates learning of image representations. The pyramidal networks utilize the two-dimensional nature of images as well as their hierarchical structure. Because the same data structures and algorithms are used in the lower layers of the pyramid and at its top, the interface problem between high-level and low-level representations, characteristic for many current computer vision systems, does not occur.

The use of weight sharing allows for reusing examples that are presented at one location for the interpretation of other locations. While this is not biologically plausible, it helps to limit the number of free parameters in the network and hence facilitates generalization. Restricting the weights to mediate specific excitation and unspecific inhibition constrains the representations used by the networks since it enforces sparse features. A similar effect can be achieved with a low-activity prior.

The use of recurrence was motivated by the ubiquitous presence of feedback in the human visual system and by the fact that an iterative solution to a problem is frequently much easier to obtain than direct one. Recurrence allows for integration of bottom-up, lateral, and top-down influences. If local ambiguities exist, the interpretation decision can be deferred until contextual evidence arrives. This yields a flexible use of context. Parts of the representation that are confident bias the interpretation of less confident parts.

This iterative approach has anytime characteristics. Initial interpretation results are available very early. If necessary, they are refined as the processing proceeds. The advantages of such a strategy are most obvious in situations which are challenging for current computer vision systems. While the interpretation of unambiguous stimuli requires no refinement, the iterative interpretation helps to resolve ambiguities. Hence, the use of the Neural Abstraction Pyramid should be considered when image contrast is low, noise is present, or objects are partially occluded. Furthermore, since the recurrent networks can integrate information over time, they are suitable for the processing of input sequences, such as video streams.

The application of learning techniques to the proposed architecture shows a way to overcome the problematic design complexity of current computer vision systems. While application-specific feature extraction methods must be designed manually when the task changes, supervised learning in the Neural Abstraction Pyramid offers the possibility of specifying the task through a set of input/output examples. Automatic optimization of all parts of the system is possible in order to produce the desired results. In this way, a generic network becomes task-specific.

11.3 Future Work

Several interesting aspects are not covered in the thesis. They include implementation options, the use of more complex processing elements, and the integration of the perception network into a complete system.

11.3.1 Implementation Options

The proposed Neural Abstraction Pyramid has been implemented on general-purpose computers, PCs. While such computers are widely available, relatively inexpensive, and quite flexible, there are some drawbacks of such a choice as well. PCs are too large for truly mobile applications, and the high operating frequencies used cause a significant consumption of electric power.

Due to the mismatch between the proposed architecture and the structure of today's PCs, the implementation of Neural Abstraction Pyramids with general-purpose computers is inefficient. Even though the architecture is fully parallel and the connectivity is local, PCs cannot take advantage of this since memory and processing elements are separated. The key operation that determines the recall speed of the network is the memory access to a weight and to the activity of its source, followed by a multiply-accumulate. While the achieved speed of the current implementation is sufficient for the interpretation of low-resolution images in real-time, a significant speedup would be needed to process high-resolution video. Even more processing power is required for on-line learning and adaptation.

Several implementation options are available to improve the speed or to lower the size/power requirements. All these options trade flexibility for efficiency. One possibility is to utilize the SIMD instructions of modern processors. Pentium 4 processors, for instance, offer MMX, SSE, and SSE2 instructions for parallel processing of 8-bit, 16-bit, and 32-bit integers, as well as floats. Current XScale processors, used in mobile devices, contain dedicated multiply-accumulate units, and Intel plans to add extended MMX instructions to future XScale processors. Programming with such SIMD instructions is less flexible since compiler support is limited, and the algorithms must be adapted to match the capabilities of the SIMD engines. In particular, the memory access pattern must be tuned to produce streaming. If the SIMD processing elements can be fully utilized, speed-up of an order of magnitude seems to be possible compared to the current implementation.

An option to achieve greater speedup is to use parallel computers with multiple CPUs. However, parallel computers are less available, larger, require more power, and are more expensive than PCs. Furthermore, significant development effort is necessary to distribute the processing efficiently between the CPUs.

If one restricts the power of individual processing elements, many of them can be implemented on a single chip. The vision processor VIP128 [184] is an example of such an approach. It contains a 2D array of processing elements that have access to small local memories and to the data of their neighbors. Other examples of special-purpose parallel processors are the XPACT data flow [19] architecture and the Imagine stream processor [119]. Such parallel processors can achieve speedup

similar to parallel computers, while keeping size and power consumption compatible with desktop PCs. It is conceivable that such specialized chips could be used as input processors in future PCs, similar to the dedicated graphic processors widely employed today. An even greater degree or parallelism is possible when using bit-serial computations or systolic arrays with thousands of processing elements.

Of course, designing special-purpose VLSI hardware to match the proposed architecture offers the greatest possibilities for speed-up and reduction of size/power, but involves significant development costs and time. Hence, this is only feasible for high-volume mobile applications, e.g. for use in cars or in PDAs/cellular phones. While field-programmable gate array (FPGA) chips can be used for prototyping, custom design is necessary to take full advantage of cost and efficiency advantages. Since the processing in the Neural Abstraction Pyramid is fully parallel, low operating frequencies can be used. This reduces the voltage needed, and hence the power consumption.

At least one additional order of magnitude can be gained in efficiency by using analog, instead of digital, VLSI [124]. Analog chips use only a single value to represent a quantity, instead of multiple bits. Furthermore, transistors do not switch, but are kept below saturation. Operations that are costly in digital VLSI, such as multiplications, can be implemented with few analog transistors. On the other hand, the precision of these operations is limited, and analog VLSI is susceptible to noise and substrate inhomogeneities. Analog VLSI offers the possibility of integrating processing elements and photosensors on the same chip in order to avoid I/O bottlenecks. One example for such a tight integration is the implementation of cellular neural networks (CNN) on the focal plane [143].

Similar to LCD displays or CMOS cameras, defects of single processing elements can be tolerated if the resolution is high. This allows for producing large chips containing millions of processing elements with high yields. Another exciting possibility is the trend towards 3D integration. Connecting a stack of chips with dense arrays of vias keeps wire length short and allows for the combination of chips that need different production processes. One example of such vertical interconnects is the SOLID process, recently announced by Infineon [3], that reduces the size of vias to $10\mu m \times 10\mu m$. It offers the possibility of establishing a direct correspondence between the layers of the Neural Abstraction Pyramid and the stack levels. The tight integration of image sensors and massively parallel hierarchical processing could yield inexpensive, small, low power devices that have the computational power of today's supercomputers for computer-vision tasks. They will be needed to allow mobile computers to perceive their environment.

11.3.2 Using more Complex Processing Elements

The simple processing elements, used in the Neural Abstraction Pyramid, resemble feed-forward neural networks with a single output-unit. It would be interesting to investigate the use of more complex processing elements. One possibility would be to employ units that are biologically more realistic. They could generate spikes and

have dynamic synapses. Spiking neurons could be used to implement fast temporal dynamics, as illustrated in the activity-driven update example. They would also allow for codes which are based on the precise timing of spikes. The use of synchronization for feature binding and segmentation could be explored. Dynamic synapses could be employed to decode temporal codes. Furthermore, they could be used to change the strength of cooperation/competition between feature cells dynamically.

Another possible line of future research is to give the network activities a probabilistic interpretation. One could view the abstraction pyramid as a graphical belief network and apply belief propagation. This proposal is motivated by the recent success of the belief propagation algorithm in cyclic graphs [76]. Unlike belief propagation in acyclic graphs, the algorithm only approximates inference and is not guaranteed to converge. Generalized belief propagation [247] has been proposed to implement better approximations with a moderate increase in computational costs.

11.3.3 Integration into Complete Systems

The goal of visual processing, in many contexts, is to ultimately control the behavior of a system based on the sensed state of the environment. This calls for an integrated treatment of perception and action. Since not only object identity, but object location is needed for action, the perception network would need to not only model the ventral visual pathway, but the dorsal one as well. Furthermore, an inverse hierarchical network could be used to expand abstract action decisions into low-level action commands. Such an integrated system could be employed to implement active vision. It would also allow for the use of reinforcement learning techniques.

References

1. Emile H.L. Aarts and Jan Korst. *Simulated Annealing and Boltzman Machines – A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Chichester, 1990.
2. David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzman machines. *Cognitive Science*, 9:147–169, 1985.
3. Infineon AG. Infineon enables third dimension of chip integration – develops SOLID stacking technology to connect multiple chips for 'system-in-package' electronics. Press release, 2002.
4. AIM, Inc. ISS Data Matrix. BC11, ANSI/AIM, 1995.
5. AIM, Inc. USS - Code 39. BC1, ANSI/AIM, 1995.
6. AIM, Inc. USS - Code 49. BC6, ANSI/AIM, 1995.
7. Shun-ichi Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27:77–87, 1977.
8. Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions Neural Networks*, 5(1):54–65, 1994.
9. Paolo Arena, Luigi Fortuna, and Mattia Frasca. Attitude control in walking hexapod robots: an analogic spatio-temporal approach. *International Journal on Circuit Theory and Applications*, 30(2):349–362, 2002.
10. ASME. Pitney Bowes Model M postage meter 1920. Designated the 20th international historic mechanical engineering landmark, The American Society of Mechanical Engineers, 1986.
11. Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions Neural Networks*, 11(3):697–709, 2000.
12. Simon Baker and Takeo Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8), August 2002.
13. Dana H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.
14. David Balya, Botond Roska, Tamas Roska, and Frank S. Werblin. CNN framework for modeling parallel processing in a mammalian retina. *International Journal on Circuit Theory and Applications*, 30(2):363–393, 2002.
15. Horace B. Barlow. Single units and sensation. *Perception*, 1:371–394, 1972.
16. Horace B. Barlow. Unsupervised learning. *Neural Computation*, 1(3):295–311, 1989.
17. Wilhelm Barthlott and Christoph Neinhuis. Purity of the sacred lotus, or escape from contamination in biological surfaces. *Planta*, 202(1):1–8, 1997.
18. Roberto Battiti. First and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

19. Volker Baumgarte, Frank May, Armin Nückel, Martin Vorbach, and Markus Weinhardt. PACT XPP - A self-reconfigurable data processing architecture. In *Proceedings of International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA 2001) – Las Vegas*, 2001.

20. Sven Behnke, Bernhard Frötschl, Raúl Rojas, Peter Ackers, Wolf Lindstrot, Manuel de Melo, Andreas Schebesch, Mark Simon, Martin Sprengel, and Oliver Tenchio. Using hierarchical dynamical systems to control reactive behavior. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *LNCS*, pages 186–195. Springer, 2000.

21. Sven Behnke, Marcus Pfister, and Raúl Rojas. Recognition of handwritten digits using structural information. In *Proceedings of International Conference on Neural Networks (ICNN'97) – Houston*, volume 3, pages 1391–1396, 1997.

22. Sven Behnke, Marcus Pfister, and Raúl Rojas. A study on the combination of classifiers for handwritten digit recognition. In *Proceedings of Third International Workshop on Neural Networks in Applications (NN'98) – Magdeburg, Germany*, pages 39–46, 1998.

23. Sven Behnke and Raúl Rojas. Activity driven update in the Neural Abstraction Pyramid. In *Proceedings of International Conference on Artificial Neural Networks (ICANN'98) – Skövde, Sweden*, volume 2, pages 567–572, 1998.

24. Sven Behnke and Raúl Rojas. Neural Abstraction Pyramid: A hierarchical image understanding architecture. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'98) – Anchorage*, volume 2, pages 820–825, 1998.

25. Sven Behnke and Raúl Rojas. A hierarchy of reactive behaviors handles complexity. In M. Hennebauer, J. Wendler, and E. Pagello, editors, *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *LNAI*, pages 125–136, Berlin, 2001. Springer.

26. Anthony J. Bell and Terrence J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

27. Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, Priceton, NJ, 1961.

28. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

29. Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is 'Nearest Neighbor' meaningful? In *Proceedings of 7th International Conference on Database Theory (ICDT'99) – Jerusalem*, pages 217–235, 1999.

30. Michael Biehl and Holm Schwarze. Learning by online gradient descent. *Journal of Physics A*, 28:643–656, 1995.

31. Matthias Bierling. Displacement estimation by hierarchical blockmatching. *SPIE Visual Communications and Image Processing*, 1001:942–951, 1988.

32. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

33. Rafal Bogacz and Marcin Chady. Local connections in a neural network improve pattern completion. In *Proceedings of 3rd International Conference on Cognitive and Neural Systems (CNS'99) – Boston*, 1999.

34. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and sysnthesis of texture images. In *Computer Graphics Proceedings SIGGRAPH'97*, pages 361–368, 1997.

35. Geoffrey M. Boynton, Jonathan B. Demb, Gary H. Glover, and David J. Heeger. Neuronal basis of contrast discrimination. *Vision Research*, 39:257–269, 1999.

36. Leo Breiman, Charles J. Stone, Richard A. Olshen, and Jerome H. Friedman. *Classification and Regression Trees*. Wadsworth International, Belmont, CA, 1984.

37. Jean Bullier. Feedback connections and conscious vision. *Trends Cognitive Science*, 5(9):369–370, 2001.

38. Peter J. Burt and Edward H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

39. Roberto Caminiti, Klaus-Peter Hoffmann, Francesco Lacquaniti, and Jennifer S. Altman, editors. *Vision and Movement Mechanisms in the Cerebral Cortex*. HFSP Workshop Series. Human Frontier Science Program, Strasbourg, 1996.

40. Gail A. Carpenter and Stephen Grossberg. *Pattern Recognition by Self-Organizing Neural Networks*. MIT Press, Cambridge, MA, 1991.

41. Leon O. Chua and Tamas Roska. The CNN paradigm. *IEEE Transactions on Circuits and Systems*, 40(3):147–156, 1993.

42. Leon O. Chua and Tamas Roska. *Cellular Neural Networks and Visual Computing*. Cambridge University Press, Cambridge, UK, 2001.

43. Ronald R. Coifman, Yves Meyer, Steven Quake, and M. Victor Wickerhauser. Signal processing and compression with wave packets. In J. S. Byrnes, editor, *Wavelets and Their Applications*, pages 363–378. Kluwer Academic Press, 1994.

44. James W. Cooley and John W. Tukey. An algorithm for the machine computation of the complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

45. Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, 2001.

46. Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Infromation Theory*, 13(1):21–27, 1967.

47. Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

48. George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

49. Ingrid Daubechies. *Ten Lectures on Wavelets*, volume 61 of *CBMS-NSF Series in Applied Mathematics*. SIAM Publications, 1992.

50. Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The Helmholtz machine. *Neural Computation*, 7:889–904, 1995.

51. Fabio Dell'Acqua and Robert Fisher. Reconstruction of planar surfaces behind occlusions in range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):569–575, 2002.

52. Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

53. Marcus Dill and Manfred Fahle. Limited translation invariance of human visual pattern recognition. *Perception and Psychophysics*, 60(1):65–81, 1998.

54. Glen M. Doninger, John J. Foxe, Micah M. Murray, Beth A. Higgins, John G. Snodgrass, Charles E. Schroeder, and Daniel C. Javitt. Activation timecourse of ventral visual stream object-recognition areas: High density electrical mapping of perceptual closure processes. *Journal of Cognitive Neuroscience*, 12(4):615–621, 2000.

55. David L. Donoho and Iain M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

56. David L. Donoho and Iain M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

57. Richard O. Duda and Peter E. Hart. *Pattern Recognition and Scene Analysis*. John Wiley & Sons, 1973.

58. Douglas Eck and Jürgen Schmidhuber. Learning the long-term structure of the Blues. In J.R. Dorronsoro, editor, *Proceedings of International Conference on Artificial Neural Networks (ICANN 2002) – Madrid*, volume 2415 of *LNCS*, pages 284–289. Springer, 2002.

59. Reinhard Eckhorn, Roman Bauer, W. Jordan, Michael Brosch, Wolfgang Kruse, M. Munk, and H.J. Reitboeck. Coherent oscillations: a mechanism for feature linking in the visual cortex. *Biological Cybernetics*, 60:121–130, 1988.

60. Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, New York, 1993.

61. Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks - a review. *Pattern Recognition*, 35(10):2279–2301, 2002.

62. Michael Elad and Arie Feuer. Super-resolution restoration of continuous image sequence - Adaptive filtering approach. *IEEE Trans. on Image Processing*, 8(3):387–395, 1999.

63. Scott E. Fahlman. Faster-learning variations on backpropagation: An empirical study. In D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School – Pittsburg*, pages 38–51, San Mateo, CA, 1988. Morgan Kaufmann.

64. Gustav Theodor Fechner. Über ein wichtiges psychophysisches Grundgesetz und dessen Beziehung zur Schätzung von Sterngrössen. In *Abk. k. Ges. Wissensch. Math.-Phys. K1,4*, 1858.

65. Daniel J. Felleman and David C. Van Essen. Distributed hierarchical processing in primate cerebral cortex. *Cerebral Cortex*, 1:1–47, 1991.

66. Roger Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Proceedings of 6th Biennial Conference on Numerical Analysis – Dundee, Scotland*, volume 506 of *Lecture Notes in Mathematics*, pages 73–89. Springer, 1976.

67. Roger Fletcher. *Practical Methods of Optimization*. John Wiley, New York, 1987.

68. Peter Földiak. Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*, 64(2):165–170, 1990.

69. Peter Földiak. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991.

70. Peter Földiak. Sparse coding in the primate cortex. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second edition*. MIT Press, 2002.

71. William T. Freeman, Thouis R. Jones, and Egon C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, 2002.

72. William T. Freeman and Egon C. Pasztor. Learning low-level vision. In *Proceedings of IEEE 7th International Conference on Computer Vision (ICCV'99) – Kerkyra, Greece*, pages 1182–1189, 1999.

73. William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

74. Yoav Freund and David Haussler. Unsupervised learning of distributions of binary vectors using 2-layer networks. In J. E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 912–919. Morgan Kaufmann, 1992.

75. Brendan J. Frey, Peter Dayan, and Geoffrey E. Hinton. A simple algorithm that discovers efficient perceptucl codes. In M. Jenkin and L.R. Harris, editors, *Computational and Biological Mechanisms of Visual Coding*, New York, 1997. Cambridge University Press.

76. Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 479–485, Cambridge, MA, 1998. MIT Press.

77. Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

78. Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:826–834, 1983.

79. Dennis Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers*, 93(III):429–457, 1946.

80. Michael D. Garris and R. Allen Wilkinson. NIST special database 3 – handwritten segmented characters. Technical Report HWSC, NIST, 1992.

81. Stuart Geman, Elie Bienenstock, and Rene Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

82. Martin A. Giese. *Dynamic Neural Field Theory for Motion Perception*, volume 469 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, 1998.

83. Venu Govindaraju. Locating human faces in photographs. *International Journal of Computer Vision*, 19:129–146, 1996.

84. Stephen Grossberg. How does the cerebral cortex work? Learning, attention and grouping by the laminar circuits of visual cortex. *Spatial Vision*, 12:163–186, 1999.

85. Stephen Grossberg and James R. Williamson. A neural model of how horizontal and interlaminar connections of visual cortex develop into adult circuits that carry out perceptual groupings and learning. *Cerebral Cortex*, 11:37–58, 2001.

86. Patrick J. Grother and Gerald T. Candela. Comparison of handprinted digit classifiers. Technical Report NISTIR 5209, NIST, 1993.

87. Jürgen Haag and Alexander Borst. Encoding of visual motion information and reliability in spiking and graded potential neurons. *Journal of Neuroscience*, 17:4809–4819, 1997.

88. Alfred Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69:331–371, 1910.

89. Richard H. R. Hahnloser. On the piecewise analysis of networks of linear threshold neurons. *Neural Networks*, 11:691–697, 1998.

90. Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–951, 2000.

91. Donald Hebb. *The Organization of Behaviour*. John Wiley, New York, 1949.

92. Bernd Heisele, Tomaso Poggio, and Massimiliano Pontil. Face detection in still gray images. AI Memo 1687, MIT AI Lab, 2000.

93. Werner Heisenberg. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Zeitschrift für Physik*, 43:172–198, 1927.

94. Rolf D. Henkel. A simple and fast neural network approach to stereovision. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 808–814. MIT Press, 1998.

95. Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Computer Graphics Proceedings SIGGRAPH 2001 – Los Angeles*, pages 327–340, 2001.

96. Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 493–499. MIT Press, 1996.

97. Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

98. Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The Wake-Sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161, 1995.

99. Erik Hjelmas and Boon Kee Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83:236–274, 2001.

100. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

101. John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79:2554–2558, 1982.

102. John J. Hopfield and Carlos D. Brody. What is a moment? 'Cortical' sensory integration over a brief interval. *Proceedings of the National Academy of Sciences, USA*, 97(25):13919–13924, 2000.

103. John J. Hopfield and Carlos D. Brody. What is a moment? Transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Sciences, USA*, 98(3):1282–1287, 2001.

104. Kurt Hornik, Maxwell Stinchcombe, , and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

105. David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

106. Aapo Hyvarinen and Erkki Oja. A fast fixed-point algorithm for independent component analysis. *Neural computation*, 9(7):1483–1492, 1997.

107. Christian Igel and Michael Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.

108. Minami Ito, Hiroshi Tamura, Ichiro Fujita, and Keiji Tanaka. Size and position invariance of neuronal responses in inferotemporal cortex. *Journal of Neurophysiology*, 73(1):218–226, 1995.

109. Herbert Jaeger. The 'echo state' approach to analysing and training recurrent neural networks. Technical Report GMD 148, German National Research Center for Information Technology, 2001.

110. Shi-Hong Jeng, Hong-Yuan Mark Liao, Chin-Chuan Han, Ming-Yang Chern, and Yao-Tsorng Liu. Facial feature detection using geometrical face model: An efficient approach. *Pattern Recognition*, 31(3):273–282, 1998.

111. Oliver Jesorsky, Klaus J. Kirchberg, and Robert W. Frischholz. Robust face detection using the Hausdorff distance. In *Proceedings of Third International Conference on Audio- and Video-based Biometric Person Authentication, LNCS-2091, Halmstad, Sweden*, pages 90–95. Springer, 2001.

112. Scott P. Johnson and Richard N. Aslin. Perception of object unity in young infants: The roles of motion, depth, and orientation. *Cognitive Development*, 11:161–180, 1996.

113. Scott P. Johnson, J. Gavin Bremner, Alan Slater, Uschi Mason, and Kirsty Foster. Young infants perception of object trajectories: Filling in a spatiotemporal gap. In *Proceedings of 12th International Conference on Infant Studies (ICIS2000) – Brighton, England*, 2000.

114. Bela Jualesz. Towards an axiomatic theory of preattentive vision. In G.M. Edelman, W.E. Gall, and W.M. Cowan, editors, *Dynamic Aspects of Neocortical Function*, pages 585–612, New York, 1984. Wiley.

115. Christian Jutten and Jeanny Herault. Blind separation of sources. an adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–31, 1991.

116. Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

117. Eric R. Kandel, James H. Schwartz, and Thomas M. Jessel, editors. *Principles of Neural Science (Fourth Edition)*. McGraw-Hill, 2000.

118. G.K. Kanizsa. *Organization in Vision*. Praeger, New York, 1989.

119. Ujval J. Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The imagine stream processor. In *Proceedings of 20th International Conference on Computer Design (ICCD 2002) – Freiburg, Germany*, pages 282–288, 2002.

120. Michael Kass, Andrew Witkin, and Demetri Terzopoulous. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

121. Philip J. Kellman and Elizabeth S. Spelke. Perception of partly occluded objects in infancy. *Cognitive Psychology*, 15:483–524, 1983.

122. Josef Kittler and John Illingworth. Minimum error thresholding. *Pattern Recognition*, 19(1):41–47, 1986.

123. James J. Knierim and David C. Van Essen. Neuronal responses to static texture patterns in area V1 of the alert macaque monkey. *Journal of Neurophysiology*, 67(4):961–980, 1992.

124. Christof Koch and Hua Li, editors. *Vision Chips: Implementing Vision Algorithms With Analog Vlsi Circuits*. IEEE Computer Society, 1995.

125. Kurt Koffka. *Principles of Gestalt Psychology*. A Harbinger Book. Harcourt Brace & World Inc., New York, 1935.

126. Teuvo Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer, Berlin, 1984.

127. Anil C. Kokaram and Simon J. Godsill. Joint noise reduction, motion estimation, missing data reconstruction, and model parameter estimation for degraded motion pictures. In *Proceedings of SPIE Conference on Bayesian Inference for Inverse Problems (SD'98) – San Diego*, pages 212–223, 1998.

128. Anderse Krogh and John Hertz. A simple weight decay can improve generalization. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957, San Mateo, CA, 1992. Morgan Kaufmann.

129. Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

130. Victor A.F. Lamme. The neurophysiology of figureground segregation in primary visual cortex. *Journal of Neuroscience*, 15:1605–1615, 1995.

131. Steve Lawrence, C. Lee Giles, and Sandiway Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126–140, 2000.

132. Yann LeCun. The MNIST database of handwritten digits. http://www.research.att.com/~yann/exdb/mnist, AT&T Labs, 1994.

133. Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Robert E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufman, 1990.

134. Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

135. Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient back-prop. In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *LNCS*, pages 9–50. Springer, 1998.

136. Choong Hwan Lee, Jun Sung Kim, and Kyu Ho Park. Automatic human face location in a complex background. *Pattern Recognition*, 29:1877–1889, 1996.

137. Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

138. Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

139. Tai Sing Lee, David Mumford, Richard Romero, and Victor A.F. Lamme. The role of the primary visual cortex in higher level vision. *Vision Research*, 38:2429–2454, 1998.

140. Robert A. Legenstein and Wolfgang Maass. Foundations for a circuit complexity theory of sensory processing. In T.K. Leen, T.G. Dietterich, , and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 259–265. MIT Press, 2001.

141. Zhaoping Li. Computational design and nonlinear dynamics of a recurrent network model of the primary visual cortex. *Neural Computation*, 13(8):1749–1780, 2001.

142. Zhaoping Li. A saliency map in the primary visual cortex. *Trends in Cognitive Sciences*, 6(1):9–16, 2002.

143. Gustavo Linan, Servando Espejo, Rafael Dominguez-Castro, and Angel Rodriguez-Vazquez. ACE16k: An advanced focal-plane analog programmable array processor. In *Proceedings of European Solid-State Circuits Conference (ESSCIRC'2001) – Freiburg, Germany*, pages 216–219. Frontier Group, 2001.

144. Yoseph Linde, Andres Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communication*, 28(1):84–95, 1980.

145. Stuart P. Lloyd. Least squares quantization in PCM. Technical note, Bell laboratories, 1957. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

146. Nikos K. Logothetis, Jon Pauls, and Tomaso Poggio. Shape representation in the inferior temporal cortex of monkeys. *Current Biology*, 5(5):552–563, 1995.

147. David R. Lovell. The Neocognitron as a system for handwritten character recognition: limitations and improvements. PhD thesis, University of Queensland, 1994.

148. Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *submitted for publication*, 2001.

149. Dario Maio and Davide Maltoni. Real-time face localization on gray-scale static images. *Pattern Recognition*, 33:1525–1539, 2000.

150. Ravi Malladi and James A. Sethian. Image processing via level set curvature flow. *Proceedings of the National Academy of Sciences, USA*, 92(15):7046–7050, 1995.

151. Stephane G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(7):674–693, 1989.

152. Stephane G. Mallat and Sifen Zhong. Characterization of signals from multiscale edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(7):710–732, 1992.

153. David Marr. *Vision: A Computational investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman and Company, San Francisco, 1982.

154. Guy Mayraz and Geoffrey E. Hinton. Recognizing hand-written digits using hierarchical products of experts. In T. K. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 953–959. MIT Press, 2001.

155. Robert J. McEliece, David J.C. MacKay, and Jung-Fu Cheng. Turbo-decoding as an instance of Pearl's 'Belief Propagation' algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

156. Ehud Meron. Pattern formation in excitable media. *Physics Reports*, 218(1):1–66, 1992.

157. Kieron Messer, Jiri Matas, Josef Kittler, Juergen Luettin, and Gilbert Maitre. XM2VTSDB: The extended M2VTS database. In *Proceedings of Second International Conference on Audio and Video-based Biometric Person Authentication (AVBPA'99) – Washington, DC*, pages 72–77, 1999.

158. Marvin L. Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

159. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

160. Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):696–710, 1997.

161. Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

162. Martin F. Møller. Supervised learning on large redundant training sets. *International Journal Neural Systems*, 4(1):15–25, 1993.

163. Franz C. Müller-Lyer. Optische Urteiltäuschungen. *Archiv für Anatomie und Physilogie, Physilogische Abteilung*, Supplementband 2:263–270, 1889.

164. Ken Nakayama, Shinsuki Shimojo, and Vilayanur S. Ramachandran. Transparency: Relation to depth, subjective contours, luminance, and neon color spreading. *Perception*, 19:497–513, 1990.

165. Thomas A. Nazir and J. Kevin O'Regan. Some results on translation invariance in the human visual system. *Spatial Vision*, 5(2):81–100, 1990.

166. Heiko Neumann and Wolfgang Sepp. Recurrent V1-V2 interaction in early visual boundary processing. *Biological Cybernetics*, 81(5/6):425–444, 1999.

167. Isaac Newton. *Methodus fluxionum et serierum infinitarum*. 1664-1671.

168. Harry Nyquist. Certain topics in telegraph transmission theory. *AIEE Transactions*, 47(3):617–644, 1928.

169. Erkki Oja. A simplified neuron model as principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

170. Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

171. Alice J. O'Toole, Heinrich H. Bulthoff, Nikolaus F. Troje, and Thomas Vetter. Face recognition across large viewpoint changes. In *Proceedings of International Workshop on Automatic Face- and Gesture-Recognition (IWAFGR'95) – Zürich*, pages 326–331, 1995.

172. Günther Palm. On associative memory. *Biological Cybernetics*, 36(1):19–31, 1980.

173. Shahla Parveen and Phil Green. Speech recognition with missing data techniques using recurrent neural networks. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1189–1195, Cambridge, MA, 2002. MIT Press.

174. Alvaro Pascual-Leone and Vincent Walsh. Fast backprojections from the motion to the primary visual area necessary for visual awareness. *Science*, 292(5516):510–512, 2001.

175. Frank Pasemann. Evolving neurocontrollers for balancing an inverted pendulum. *Network: Computation in Neural Systems*, 9:495–511, 1998.

176. Anitha Pasupathy and Charles E. Connor. Shape representation in area V4: Position-specific tuning for boundary conformation. *Journal on Neurophysiology*, 86:2505–2519, 2001.

177. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, Ca, 1988.

178. Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

179. Lutz Prechelt. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, 11:761–767, 1998.

180. Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12:145–151, 1999.

181. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

182. J. Ross Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, Ca, 1993.

183. Rajeev D.S. Raizada and Stephen Grossberg. Context-sensitive bindings by the laminar circuits of V1 and V2: A unified model of perceptual grouping, attention, and orientation contrast. *Visual Cognition*, 8:341–466, 2001.

184. Ulrich Ramacher, Wolfgang Raab, Nico Brüls, Ulrich Hachmann, Christian Sauer, Alex Schackow, Jörg Gliese, Jens Harnisch, Mathias Richter, Elisabeth Sicheneder, Uwe Schulze, Hendrik Feldkämper, Christian Lütkemeyer, Horst Süsse, and Sven Altmann. A 53-GOPS programmable vision processor for processing, coding-decoding and synthesizing of images. In *Proceedings of European Solid-State Circuits Conference (ESSCIRC 2001) – Villach, Austria*, 2001.

185. Benjamin M. Ramsden, Chou P. Hung, and Anna Wang Roe. Real and illusory contour processing in area V1 of the primate: A cortical balancing act. *Cerebral Cortex*, 11(7):648–665, 2001.

186. Rajesh P. N. Rao. An optimal estimation approach to visual perception and learning. *Vision Research*, 39(11):1963–1989, 1999.

187. Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999.

188. Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (J. SIAM)*, 8(2):300–304, 1960.

189. Gerald M. Reicher. Perceptual recognition as a function of meaningfulness of stimulus material. *Journal of Experimental Psychology*, 81(2):275–280, 1969.

190. John H. Reynolds and Robert Desimone. The role of neural mechanisms of attention in solving the binding problem. *Neuron*, 24:19–29, 1999.

191. Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of International Conference on Neural Networks (ICNN'93) – San Francisco*, pages 586–591. IEEE, 1993.

192. Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

193. Raúl Rojas. *Neural Networks – A Systematic Introduction*. Springer, New York, 1996.

194. Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

195. Azriel Rosenfeld, Robert A.Hummel, and Steven W. Zucker. Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man and Cybernetics*, 6:420–433, 1976.

196. Azriel Rosenfeld and Gordon J. Vanderbrug. Coarse-fine template matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(2):104–107, 1977.

197. Botond Roska and Frank S. Werblin. Vertical interactions across ten parallel, stacked representations in the mammalian retina. *Nature*, 410:583–587, 2001.

198. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network based face detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20:23–38, 1998.

199. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Rotation invariant neural network based face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98) – Santa Barbara, CA*, pages 38–44, 1998.

200. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

201. David E. Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

202. Ralf Salomon and J. Leo van Hemmen. Accelerating backpropagation through dynamic self-adaptation. *Neural Networks*, 9(4):589–601, 1996.

203. Terence D. Sanger. Optimal unsupervised learning in a single-layer linear feed-forward neural network. *Neural Networks*, 2(6):459–473, 1989.

204. Harish K. Sardana, M. Farhang Daemi, and Mohammad K. Ibrahim. Global description of edge patterns using moments. *Pattern Recognition*, 27(1):109–118, 1994.

205. Cullen Schaffer. Selecting a classification method by cross-validation. *Machine Learning*, 13(1):135–143, 1993.

206. Henry Schneiderman and Takeo Kanade. A statistical model for 3D object detection applied to faces and cars. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2000) – Hilton Head Island, SC*. IEEE, June 2000.

207. Bernhard Schölkopf and Alex Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

208. David M. Senseman and Kay A. Robbins. High-speed VSD imaging of visually evoked cortical waves: Decomposition into intra- and intercortical wave motions. *Journal of Neurophsyiology*, 87(3):1499–1514, 2002.

209. H. Sebastian Seung. Learning continuous attractors in recurrent networks. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 654–660. MIT Press, 1998.

210. Markus Siegel, Konrad P. Körding, and Peter König. Integrating top-down and bottom-up sensory processing by somato-dendritic interactions. *Journal of Computational Neuroscience*, 8:161–173, 2000.

211. Hava T. Siegelmann and Eduardo D. Sonntag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

212. Fernando M. Silva and Luis B. Almeida. Acceleration techniques for the backpropagation algorithm. In *Proceedings of Neural Networks EURASIP 1990 Workshop – Sesimbra, Portugal*, volume 412, pages 110–119. Springer, 1990.

213. Eero P. Simoncelli and Edward H. Adelson. Noise removal via Bayesian wavelet coring. In *Proceedings of IEEE International Conference on Image Processing (ICIP'96) – Lausanne, Switzerland*, 1996.

214. Eero P. Simoncelli, William T. Freeman, Edward H. Adelson, and David J. Heeger. Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607, 1992.

215. Eero P. Simoncelli and Bruno A. Olshausen. Natural image statistics and neural representation. *Annual Review of Neuroscience*, 24:1193–1216, 2001.

216. Wolfgang Singer and Charles M. Gray. Visual feature integration and the temporal correlation hypothesis. *Annual Review of Neuroscience*, 18:555–586, 1995.

217. Kate Smith, Marimuthu Paliniswami, and Mohan Krishnamoorthy. Neural techniques for combinatorial optimization with applications. *IEEE Transactions on Neural Networks*, 9(6):1301–1318, 1998.

218. Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D.E. Rumelhart, J.L. McClelland, and PDP Research Group, editors, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, volume I: Foundations, pages 194–281. MIT Press, 1986.

219. David C. Somers, Emanuel V. Todorov, Athanassios G. Siapas, Louis J. Toth, Dae-Shik Kim, and Mriganka Sur. A local circuit approach to understanding: Integration of long-range inputs in primary visual cortex. *Cerebral Cortex*, 8(3):204–217, 1998.

220. Martin Stemmler, Marius Usher, and Ernst Niebur. Lateral interactions in primary visual cortex: A model bridging physiology and psycho-physics. *Science*, 269:1877–1880, 1995.

221. Kah Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(1):39–51, 1998.

222. Hans Super, Henk Spekreijse, and Victor A. F. Lamme. Two distinct modes of sensory processing observed in monkey primary visual cortex (v1). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(3):304–310, 2001.

223. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

224. Raiten Taya, Walter H. Ehrenstein, and C. Richard Cavonius. Varying the strength of the Munker-White effect by stereoscopic viewing. *Perception*, 24:685–694, 1995.

225. Jean-Christophe Terrillon, Mahdad Shirazi, Hideo Fukamachi, and Shigeru Akamatsu. Comparative performance of different skin chrominance models and chrominance spaces for the automatic detection of human faces in color images. In *Proceedings of Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG 2000) – Grenoble, France*, pages 54–61, 2000.

226. Simon J. Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381:520–522, 1996.

227. Simon J. Thorpe and Jacques Gautrais. Rank order coding: A new coding scheme for rapid processing in neural networks. In J. Bower, editor, *Computational Neuroscience : Trends in Research*, pages 333–361, New York, 1998. Plenum Press.

228. Tom Tollenaere. SuperSAB: Fast adaptive backpropagation with good scaling properties. *Neural Networks*, 3(5):520–522, 1990.

229. Anne Treisman, Marilyn Sykes, and Gary Gelade. Selective attention stimulus integration. In S. Dornie, editor, *Attention and Performance VI*, pages 333–361, Hilldale, NJ, 1977. Lawrence Erlbaum.

230. Misha V. Tsodyks and Henry Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the National Academy of Sciences, USA*, 94:719–723, 1997.

231. Mark R. Turner. Texture discrimination by Gabor functions. *Biological Cybernetics*, 55:71–82, 1986.

232. Rufin VanRullen, Arnaud Delorme, and Simon J. Thorpe. Feed-forward contour integration in primary visual cortex based on asynchronous spike propagation. *Neurocomputing*, 38-40(1-4):1003–1009, 2001.

233. Vladimir N. Vapnik and Alexei Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

234. Sethu Vijayakumar, Jörg Conradt, Tomohiro Shibata, and Stefan Schaal. Overt visual attention for a humanoid robot. In *Proceedings of International Conference on Intelligence in Robotics and Autonomous Systems (IROS 2001), Hawaii*, pages 2332–2337, 2001.

235. Felix von Hundelshausen, Sven Behnke, and Raúl Rojas. An omnidirectional vision system that finds and tracks color edges and blobs. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: RobotSoccer WorldCup V*, volume 2377 of *LNAI*, pages 374–379. Springer, 2002.

236. Ernst Heinrich Weber. *De pulsu, resorptione, audita et tactu. Annotationes anatomicae et physiologicae*. Koehler, Leipzig, 1834.

237. Peter De Weerd, Robert Desimone, and Leslie G. Ungerleider. Perceptual filling-in: A parametric study. *Vision Research*, 38:2721–2734, 1998.

238. Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn*. Morgan Kaufmann, San Mateo, CA, 1990.

239. Jörg Wellner and Andreas Schierwagen. Cellular-automata-like simulations of dynamic neural fields. In M. Holcombe and R. Paton, editors, *Information Processing in Cells and Tissues*, pages 295–403, New York, 1998. Plenum Press.

240. Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

241. Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

242. Charles L. Wilson. Evaluation of character recognition systems. In *Neural Networks for Signal Processing III – New York*, pages 485–496. IEEE, 1993.

243. Laurenz Wiskott. How does our visual system achieve shift and size invariance? In J.L. van Hemmen and T. Sejnowski, editors, *Problems in Systems Neuroscience*. Oxford University Press, to appear 2003.

244. Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

245. Rachel O. L. Wong. Retinal waves and visual system development. *Annual Reviews on Neuroscience*, 22:29–47, 1999.

246. Fan Yang, Michel Paindavoine, Herve Abdi, and Johel Miteran. A new image filtering technique combining a wavelet transform with a linear neural network: Application to face recognition. *Optical Engineering*, 38:2894–2899, 2000.

247. Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695. MIT Press, 2001.

248. Kin Choong Yow and Roberto Cipolla. Feature-based human face detection. *Image and Vision Computing*, 15(9):713–735, 1997.

Index