José A. Osuna

The Recognition of Acoustical Alarm Signals with Cellular Neural Networks

Hartung-Gorre Verlag Konstanz
1995

Diss. ETH No. 11058

The Recognition of Acoustical Alarm Signals with Cellular Neural Networks

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Engineering Sciences

presented by
JOSÉ A. OSUNA
dipl. El.-Ing. ETH
born May 6, 1965
citizen of Spain

accepted on the recommendation of
Prof. Dr. G. S. Moschytz, examiner
Prof. Dr. M. Hasler, co-examiner

1995

A mis Abuelos

Consuelo y Victorino

quienes iniciaron todo.

Acknowledgments

It was in late September 1991 after the ECCTD-91 in Kopenhagen[1] that Prof. Moschytz suggested I have a look at two interesting papers. He had met Prof. Chua at the conference who had told him about the Cellular Neural Networks and Prof. Moschytz immediately foresaw their potential for signal processing. I read the two introductory papers with great interest; this was the initiation of this thesis.

I mention the anecdote above because it is typical for the way in which Prof. Moschytz guided my thesis during the time at the laboratory: he always made the right suggestions giving me very precious freedom in what I finally decided to do. I am very thankful to him for this valuable working environment, but also for his warm-hearted conduct.

Prof. Hasler spontaneously agreed, in spite of the short deadline, to be co-examiner. His technical comments have improved the quality of this thesis. I am indebted to him for carefully reading through my work.

Prof. Chua's enthusiasm for his Cellular Neural Networks is immense. It is due to his push last year during his three-month stay at our laboratory that I did not give up trying to find a solution for the classification task. I thank him also for having made such fascinating research possible by inventing the Cellular Neural Networks.

Prof. Roska invited me for a three-month stay at the Hungarian Academy of Sciences in spring 1993. I gratefully acknowledge his interest in my project. I also wish to thank the people from his group for their friendly support, especially Tibor Kozek for the e-mail contact we had after I left Budapest.

[1] European Conference on Circuit Theory and Design

I was very fortunate to have the opportunity to share the office with Gusti Kaelin before he was elected Professor. He taught me in long discussions how to approach scientific problems. His modest way of presenting excellent solutions served me to evaluate other people's work.

Thanks go to *all* members of the laboratory. It is a great pleasure to work in a place where everyone spontaneously helps the others. Dr. Max Dünki deserves special mention because most of my knowledge on computer systems originates from his deep insight into almost everything related to computers.

My sincere thanks are also extended to two student groups who accelerated my thesis by completing excellent semester projects: Ivo Hasler and Björn Tiemann developed the fastest and most flexible simulator of Cellular Neural Networks, and Justus Bernold and Martin Hänggi investigated several methods on how to process the transformed acoustical signals with Cellular Neural Networks.

I would like to thank Dr. Urs Müller, Michael Kocheisen, Bernhard Bäumle and the other members of the Electronics Laboratory who gave me their support concerning NeuroBasic and the MUSIC parallel computer. My thanks are due to Dr. Xavier Arreguit of the Centre Suisse d'Electronique et de Microtechnique (CSEM) in Neuchâtel for his assistance on the binaural chip.

Drahoslav Lím's help was twofold: he revised the whole thesis by correcting my English, and he provided me with his LaTeX 2ε distribution for the Sun workstations at the laboratory and my Linux box at home. Without his fast and thorough corrections I would not had been able to keep to the deadline. My thanks to him for his priceless help.

I do not want to miss the opportunity of thanking my parents for encouraging me to follow my interests. They always supported me in all aspects concerning my education. Their efforts led to a free choice in what I could work on after attaining my engineering diploma.

Finally, I wish to express my loving gratitude to my wife Ruth and my children Rafael, Ester and Pablo for their patience and understanding in all these days and nights they had to do without a normal family life.

Abstract

This thesis investigates Cellular Neural Networks (CNNs), especially, their capabilities for signal processing. A CNN is a time-continuous, nonlinear dynamical system which is composed of locally interconnected cells that work in parallel. Analogue chips based on this structure are presently being developed for use in applications where sophisticated signal processing at low power consumption is required.

We organized this thesis into two parts. The first part presents some theoretical results on CNNs. The second part presents the solution of an acoustical-signal classification problem, thereby showing a possible application of CNNs in hearing aids.

Part I consists of three chapters. In the first, we introduce the basic CNN structure. Chapter 2 presents a method for the determination of the network parameters for a certain class of CNNs. Finally, in Chapter 3 we give lower and upper bounds on the number of different tasks that can be performed by CNNs with binary input values.

Chapter 4 gives an overview of the contents of Part II. The problem of the recognition of given sets of acoustical alarm signals is stated, and its solution with CNNs is briefly discussed. The first section of Chapter 5 deals with the preprocessing stage needed to transform the signals under consideration into images in order to provide inputs suitable for the CNN. The phase-preserving transformation considered there may well be performed by a CNN as well, as demonstrated in the second section of Chapter 5, which gives the solution for the realization of arbitrary second-order linear filters with CNNs. Next, Chapter 6 presents three processing methods on how to generate a characteristic image for every given acoustical alarm signal: different image-processing

steps and a simple time–to–space mapping are used in every method. The resulting characteristic images are classified by a one-layer perceptron (Chapter 7). Thus, the idea of using CNNs for the classification of acoustical alarm signals is to transform the problem into one of image classification.

In Chapter 8 we mention the classification skills of healthy people, and list the results achieved with our system. A discussion of the results can be found in the last chapter. By comparing the performance of our system to that of other schemes we can state that, first, CNNs may well be used for the recognition of acoustical alarm signals in hearing aids, and, second, CNNs are capable of processing non-stationary signals.

In the appendix we present the tools used to simulate our system. The flexible and easy–to–use simulator of CNNs runs on a parallel computer and presently achieves the fastest iteration speed of simulators reported in recent literature.

Kurzfassung

Die vorliegende Dissertation handelt von zellularen neuronalen Netzwerken, kurz CNN (Cellular Neural Networks), und im speziellen von deren Einsatz für die Verarbeitung nichtstationärer Signale. Ein CNN ist ein zeitkontinuierliches, nichtlineares, dynamisches System, das aus parallel arbeitenden Zellen besteht, welche lokal miteinander verbunden sind, d.h., nur Verknüpfungen zwischen benachbarten Zellen aufweisen. Analoge integrierte Schaltungen, die auf dieser Struktur basieren, sind weltweit Gegenstand laufender Forschungsarbeiten. Es wird das Ziel verfolgt, die CNN in Anwendungen zum Einsatz kommen zu lassen, welche hohe Signalverarbeitungsgeschwindigkeiten bei geringer Leistungsaufnahme voraussetzen.

Die Abhandlung ist in zwei Teile gegliedert. Im ersten Teil werden theoretische Resultate zu den CNN präsentiert, im zweiten Teil wird diese Netzwerkstruktur für die Klassifikation akustischer Alarmsignale eingesetzt. Es wird damit gezeigt, dass Hörgeräte ein mögliches Anwendungsgebiet der CNN sind.

Teil I besteht aus drei Kapiteln. Im ersten wird der Aufbau der CNN vorgestellt. Kapitel 2 zeigt, wie die Netzwerkparameter für eine spezielle CNN-Klasse ermittelt werden können. Schliesslich werden in Kapitel 3 untere und obere Schranken für die Anzahl verschiedener Aufgaben angegeben, die mit den CNN bei zweiwertigen Eingangsdaten gelöst werden können.

Kapitel 4 gibt eine Übersicht zu Teil II. Das Problem der Erkennung vorgegebener Klassen akustischer Alarmsignale wird formuliert und dessen CNN-Lösung skizziert. Der erste Abschnitt von Kapitel 5 behandelt die Vorverarbeitung der betrachteten Signale. Diese werden zu Bildern umgewandelt, um eine für die CNN geeignete Darstellung der Eingangsdaten zu bekommen. Die verwendete, phasenerhaltende Abbildung kann mit den CNN selbst vorgenom-

men werden, wie im zweiten Abschnitt von Kapitel 5 anhand der Realisierung beliebiger linearer Filter zweiter Ordnung mit einer CNN-kompatiblen Struktur gezeigt wird. Kapitel 6 stellt drei Verarbeitungsmethoden zur Generierung eines charakteristischen Bildes für jedes gegebene akustische Alarmsignal vor. Dabei werden verschiedene Bildverarbeitungsschritte und eine einfache Umwandlung der Zeitachse in eine örtliche Dimension vorgenommen. Die resultierenden charakteristischen Bilder werden mit einem Einschichtperceptron klassiert (Kapitel 7). Die CNN werden also dazu verwendet, das Problem der Erkennung akustischer Alarmsignale auf eine Bildklassifikation zu führen.

In Kapitel 8 wird anhand von Untersuchungen gezeigt, wie gut der nicht hörgeschädigte Mensch die akustischen Alarmsignale erkennt, und die Erkennungsfehlerraten des vorgeschlagenen Systems werden präsentiert. Die Besprechung der Resultate erfolgt im letzten Kapitel. Durch den Vergleich mit anderen Klassifikationsmethoden kann erstens festgehalten werden, dass sich die CNN anbieten, die Klassifikation akustischer Alarmsignale in Hörgeräte zu implementieren. Zweitens lässt der Vergleich der Resultate die Aussage zu, dass sich die CNN eignen, um nichtstationäre Signale zu verarbeiten.

Im Anhang werden die Simulationsprogramme vorgestellt, die in Zusammenhang mit den gemachten Untersuchungen entwickelt wurden. Der Simulator der CNN ist vielseitig und einfach zu bedienen. Er wurde auf einem Parallelrechner implementiert und weist zur Zeit die kürzesten Iterationszeiten im Vergleich zu anderen neulich publizierten CNN-Simulatoren auf.

Resumen

En esta tesis doctoral se investigan las redes neuronales celulares, o CNNs (Cellular Neural Networks), y especialmente su capacidad para el procesado de señal. La CNN es un sistema dinámico no lineal en tiempo continuo compuesto por células que trabajan paralelamente y con conexiones locales, es decir que toda célula está conectada únicamente con células vecinas. Actualmente se están desarollando circuitos integrados analógicos basados en esta estructura para utilizarlos en aplicaciones que requieren un procesado de señal sofisticado conjuntamente con un consumo mínimo de potencia eléctrica.

La tesis se compone de dos partes. En la primera se presentan resultados teóricos sobre las CNNs. La segunda parte soluciona un problema de clasificación de señales acústicas demostrando que las CNNs se pueden utilizar en aparatos de ayuda para personas con problemas auditivos.

La parte I la forman tres capítulos. En el primero se introduce la estructura básica de la CNN. El capítulo 2 presenta un método para determinar los parámetros de la red teniendo en cuenta una cierta clase de CNNs. Finalmente, en el capítulo 3, se derivan el límite inferior y superior del número de diferentes funciones que se pueden realizar con CNNs en el caso de señales de entrada binarias.

El capítulo 4 presenta una visión general del contenido de la parte II. Se define el problema de la clasificación de determinadas clases de señales acústicas de alarma, y se presenta brevemente su solución con CNNs. La primera sección del capítulo 5 trata la etapa de preprocesado necesaria para transformar en imágenes las señales consideradas, de forma que la CNN sea alimentada con señales apropiadas para esta red. La transformación utilizada conserva la información sobre la fase de la señal de entrada, y también se puede realizar

con una CNN, como se demuestra en la segunda sección del capítulo 5 al presentar la solución de la realización de cualquier filtro lineal de segundo orden con CNNs. El capítulo 6 muestra tres formas de generar una imagen característica para cada señal acústica de alarma considerada: cada método emplea diferentes pasos de procesado de imagen y una simple transformación de la dimensión temporal en una espacial. Las imágenes características producidas se clasifican con un perceptron de una sola capa (capítulo 7). Por consiguiente, la idea de utilizar las CNNs para la clasificación de señales acústicas de alarma es la de resolver el problema a través de una clasificación de imágenes.

En el capítulo 8 se incluyen las capacidades de clasificación de personas con audición normal, y se presentan los resultados obtenidos con el sistema propuesto por esta tesis. La discusión de los resultados se encuentra en el último capítulo. A través de la comparación con otros métodos de clasificación se puede constatar que, primero, las CNNs se prestan para la clasificación de señales acústicas de alarma y su implementación en aparatos acústicos, y, segundo, las CNNs son capaces de procesar señales no estacionarias.

En el apéndice se presentan los programas que se desarrollaron con respecto a las investigaciones realizadas en este trabajo. El simulador de CNNs es muy completo y facil de manejar. Está implementado en un ordenador paralelo y presenta los tiempos de iteración más cortos entre los simuladores de CNNs publicados recientemente.

Contents

Part I

Cellular Neural Networks

The first part of this thesis gives a short introduction to the concept of Cellular Neural Networks. Further, a design procedure for a special class of Cellular Neural Networks is presented, and boundaries for the diversity of the networks are computed.

Chapter 1

The CNN Structure

A Cellular Neural Network (CNN) is a time-continuous, nonlinear dynamical system which performs a signal-processing task on a multidimensional input signal. The system dynamics is completely determined by a small set of network parameters. In most applications the input signal is constant and two-dimensional, representing an image, and the network parameters are set such that the CNN converges to a stable state. The CNN is then said to perform an image-processing task by carrying out a nonlinear mapping of an input image onto an output image.

1.1 Introduction

L. O. Chua and L. Yang presented the basic CNN structure in 1988 [1, 2]. The concept of a CNN is similar to that of cellular automata [3] with the main difference that a CNN, as introduced in [1, 2], is a *time-continuous* network, whereas cellular automata are logical operators which operate in discrete time.

A CNN is an analogue nonlinear dynamical system with the following characteristics:

1 Information is processed in parallel by individual units, the so-called cells, which are all *identical*.

2 Due to the *local* connectivity between the cells, a chip implementation using analogue techniques can be realized also for CNNs with a large number (e.g., thousands) of cells.

3 The system dynamics is controlled by a *small set of parameters*, the so-called templates.

Most applications of CNNs can be found in image processing, i.e., the CNN is used to map an input image onto a desired output image. The CNN is then viewed as a mapping device, and not as a dynamical system: the final state of the network is considered to be the result for a given input, or for a given initial state, and the transient behaviour is of no interest. Moreover, one wishes to perform the mapping as fast as possible.

The convergence time of a CNN depends basically on the technology that is used to implement it. Normally, the convergence time, i.e., the time that is needed to perform an image-processing task such as, e.g., edge extraction, is in the range of microseconds [4, 5]. Also, due to the inherent analogue structure, all the processing may be performed with low power consumption [6], in spite of the high operating speed.

Conventional neural networks [7] can be viewed as rudimentary models of biological nerve systems. One feature of a biological nerve system is the high connectivity between the neurons, i.e., the individual nerve cells. Thus, in a neural network the "artificial" neurons are also largely interconnected in order to correspond best to the biological system. However, the placement and routing of many long wires in integration implementations on silicon is a difficult problem. CNNs avoid it by connecting only adjacent cells to each other.

Of course, a CNN suffers some restrictions compared to its fully-connected counterpart, the Hopfield network [8], but we will see that it still can be used to perform many tasks.

A few parameters control the behaviour of the CNN. However, in the general case, and for time-continuous CNNs,[1] there is no efficient learning algorithm or analytical method to compute the parameter values in order to allow the CNN to perform a desired processing task. A template library for a large variety of image processing tasks is available [10, 11], but for new situations, especially in the case where the network parameters influence the dynamics through a nonlinearity (nonlinear templates) [12], experience and intuition are required.[2]

As yet, only very few programmable CNN chips are available. Programmability means that for each processing task required, appropriate parameter values (templates) can be changed from outside the chip. The last point is also a topic of CNN research [5]. In the referenced work, a single programmable CNN cell has successfully been implemented on a modular chip. However, for a large number of cells, CNNs still have to be simulated on a digital computer (see Appendix A.2).

In the last years, the definition of the CNN given in [1] has been expanded to the so-called CNN *universal machine* [13, 14], an analogue, multipurpose computer. We present in the following sections the basic structure given in the original publication. The additional definitions used in the CNN universal machine will be discussed only in as much as they are introduced in this thesis.

1.2 The One-Layer CNN

Let us describe the CNN structure in its original form for a one-layer CNN.[3] A CNN consists of an $M \times N$ array of identical cells $C(\cdot,\cdot)$, where $C(i,j)$ represents the cell at position (i,j), i.e., at the i^{th} row and j^{th} column, with $1 \leq i \leq M$ and $1 \leq j \leq N$. The neighbourhood $N_r(i,j)$ of cell $C(i,j)$ with

[1] Discrete-time CNNs have also been defined [9], but they do not add any new concepts to the cellular automata mentioned above.

[2] In Chapter 2, an exact design of the template values is developed for the special case of so-called *reciprocal* CNNs with binary input values.

[3] For a detailed description of *multilayer* CNNs, see [1]. We do not consider them here, because they are hardly implementable at present.

radius r is defined as

$$N_r(i,j) = \{C(k,l) | \max\{|k-i|, |l-j|\} \leq r,\ 1 \leq k \leq M;\ 1 \leq l \leq N\} \quad (1.1)$$

i.e., all cells at a distance r from cell $C(i,j)$ belong to the neighbourhood $N_r(i,j)$, as well as cell $C(i,j)$ itself.[4] All cells in the network have the same neighbourhood radius r. Figure 1.1 shows the $M \times N$ array of a one-layer CNN

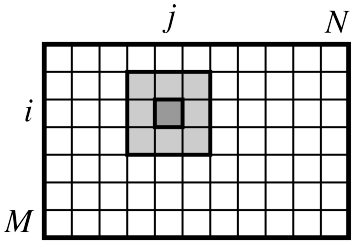


Figure 1.1: *$M \times N$ CNN with neighbourhood $N_1(i,j)$*

with a neighbourhood radius $r = 1$. Usually, CNNs use a neighbourhood radius $r = 1$. This property, known as *local connectivity*, makes CNNs suitable for integration implementations because then the placement and routing of wires is restricted to the neighbourhood of the cell.

1.3 Cell Dynamics

As mentioned above, all cells in the CNN are identical. Figure 1.2 shows the cell circuit for cell $C(i,j)$. The voltages u_{ij}, x_{ij} and y_{ij} are the input, state, and output of cell $C(i,j)$, respectively. I is the output current of a constant current source. $I_n^{y_{ij}}$ and $I_n^{u_{ij}}$ are the output currents of two sets of linear voltage-controlled current sources (VCCSs). We will discuss their meaning below. The voltage-controlled voltage source (VCVS) at the output of the cell is nonlinear, and is driven by x_{ij}. Its characteristic is shown in Figure 1.3. Equation (1.2) defines the piecewise-linear function mathematically.

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t)+1| - |x_{ij}(t)-1|) \quad (1.2)$$

[4] Cells at the borders of the $M \times N$ array, i.e., cells $C(i,j) \in \{C(k,l) | 1 \leq k \leq r \vee 1 \leq l \leq r \vee M-r < k \leq M \vee N-r < l \leq N\}$, have "degenerated" neighbourhoods with fewer members.

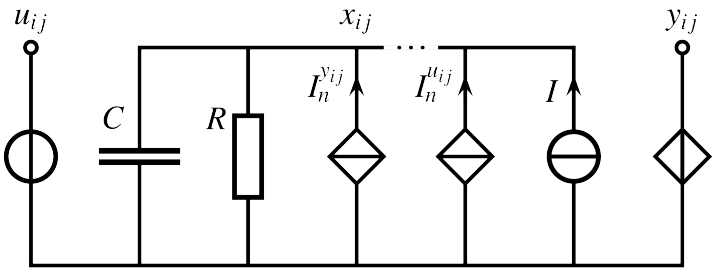


Figure 1.2: *Cell circuit of cell $C(i, j)$*

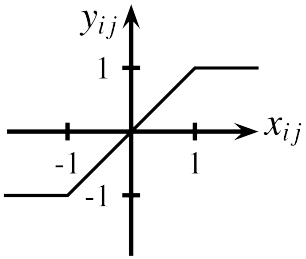


Figure 1.3: *Characteristic of the nonlinear VCVS at the output of cell $C(i, j)$*

Applying Kirchhoff's laws to the cell circuit[5] we get the nonlinear differential equation (1.3), which, together with equation (1.2), defines the dynamics for $C(i, j)$.[6]

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{n=1}^{(2r+1)^2} I_n^{y_{ij}} + \sum_{n=1}^{(2r+1)^2} I_n^{u_{ij}} + I \tag{1.3}$$

Because $1 \leq i \leq M$ and $1 \leq j \leq N$, and due to the property of identical cells, the dynamics of the CNN is completely described by $M \cdot N$ nonlinear differential equations of the form (1.3) and $M \cdot N$ equations of the form (1.2).

The terms $I_n^{y_{ij}}$ and $I_n^{u_{ij}}$ correspond to the influence of the cells in the neigh-

[5]The current through capacitance C equals the sum of all current sources minus the current through resistance R.

[6](1.3) is nonlinear because $\dot{x}_{ij}$ depends nonlinearly on x_{ij} through the VCVS at the output of the cell.

bourhood $N_r(i,j)$ on $C(i,j)$. The outputs of the cells within $N_r(i,j)$ influence x_{ij} through the set $I_n^{y_{ij}}$, and the inputs of the cells within $N_r(i,j)$ influence x_{ij} through the set $I_n^{u_{ij}}$. The dependencies are linear, i.e., the output currents $I_n^{y_{ij}}$ and $I_n^{u_{ij}}$ depend linearly on the driving voltages of the VCCSs. Figure 1.4 illustrates these relations for $r = 1$ and $I_n^{y_{ij}}$. The coefficients $a_1, \ldots, a_9$ [7] are ar-

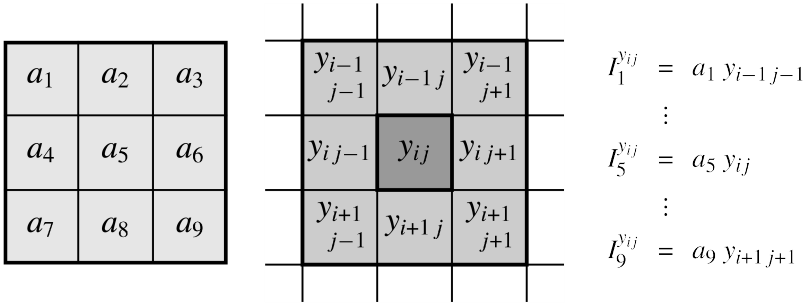


Figure 1.4: $I_n^{y_{ij}}$, $1 \le n \le (2r+1)^2$, *are the outputs of the VCCSs controlled by the output voltages* y_{kl} *within* $N_r(i,j)$. *(*$r = 1$*)*

ranged into the so-called feedback or A template (left part of Figure 1.4). It is space invariant, i.e., the output voltages y_{kl} within $N_r(i,j)$ are multiplied by the same coefficients $a_1, \ldots, a_9$ independently of the absolute position (i,j).[8] Analogously, the input voltages u_{kl} of the cells in the neighbourhood of $C(i,j)$ are weighted by the control or B template with elements $b_1, \ldots, b_9$ (see Table 1.1).

A		
a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

B		
b_1	b_2	b_3
b_4	b_5	b_6
b_7	b_8	b_9

$I = c$

Table 1.1: *Templates of a CNN with neighbourhood radius* $r = 1$

[7] From now on we will assume $r = 1$.

[8] Actually, one can define the template as space variant [1], but then the useful property of a *cloning* template which simplifies chip integration gets lost.

Equation (1.4) shows how $I_n^{y_{ij}}$ and $I_n^{u_{ij}}$ depend on y_{kl} and u_{kl}, respectively. Setting $r = 1$ in (1.3), we obtain

$$\begin{aligned} C\frac{dx_{ij}(t)}{dt} &= -\frac{1}{R}x_{ij}(t) \\ &\quad + a_1\, y_{i-1\,j-1} + a_2\, y_{i-1\,j} + a_3\, y_{i-1\,j+1} \\ &\quad + a_4\, y_{i\,j-1} + a_5\, y_{ij} + \cdots + a_9\, y_{i+1\,j+1} \\ &\quad + b_1\, u_{i-1\,j-1} + b_2\, u_{i-1\,j} + b_3\, u_{i-1\,j+1} \\ &\quad + b_4\, u_{i\,j-1} + b_5\, u_{ij} + \cdots + b_9\, u_{i+1\,j+1} \\ &\quad + I \end{aligned} \tag{1.4}$$

1.4 Analysis of a Linear Template

We now analyse the dynamical behaviour of cell $C(i, j)$ when a CNN with the template given in Table 1.2 is used.[9]

A		
0	0	0
0	a_5	0
0	0	0

B		
b	b	b
b	b_5	b
b	b	b

I

Table 1.2: *A linear template with four parameters, where we assume that* $a_5 > 1$. *The initial state of the CNN is to be set equal to the input image.*

The cell dynamics is described by the differential equation (1.5) (see the differential equation (1.4)).[10]

$$\dot{x}_{ij} = -x_{ij} + a_5 y_{ij} + \mathcal{U}_{ij} + b_5 u_{ij} + I \tag{1.5}$$

where

$$\mathcal{U}_{ij} = b\left(u_{i-1\,j-1} + u_{i-1\,j} + u_{i-1\,j+1} + u_{i\,j-1} + u_{i\,j+1} + u_{i+1\,j-1} + u_{i+1\,j} + u_{i+1\,j+1}\right) \tag{1.6}$$

[9] Here, the notion *template* stands for all network parameters, i.e., for templates A and B, and the constant current source I.

[10] Without loss of generality, we treat the normalized case, i.e., $C = R = 1$.

$\mathcal{U}_{ij}$ denotes the contribution of the inputs of the off-center cells. The relation between y_{ij} and x_{ij}, i.e., the nonlinearity at the output of the cell, is given in (1.2) on page 6 (see also Figure 1.3). The inputs are assumed to have values normalized between ± 1, i.e., $|u_{ij}| \leq 1, \forall\, i, j$.

To compute the solution of (1.5), we can exploit the property that (1.2) is piecewise linear. Thus, (1.5) can be split into the three portions given in (1.7).[11]

$$\dot{x} = (a_5 - 1)x + \mathcal{C} \quad , \text{for } |x| \leq 1 \tag{1.7a}$$

$$\dot{x} = -x + \mathcal{C} + a_5 \quad , \text{for } 1 \leq x \tag{1.7b}$$

$$\dot{x} = -x + \mathcal{C} - a_5 \quad , \text{for } x \leq -1 \tag{1.7c}$$

where

$$\mathcal{C} = \mathcal{U} + b_5 u + I \tag{1.8}$$

The solutions of the linear differential equations given in (1.7a) to (1.7c) with the corresponding initial conditions (1.9), are (1.10a), (1.10b) and (1.10c), respectively.

$$x(0) = u \quad , \text{for } |x| \leq 1 \tag{1.9a}$$

$$x(0) = 1 \quad , \text{for } 1 \leq x \tag{1.9b}$$

$$x(0) = -1 \quad , \text{for } x \leq -1 \tag{1.9c}$$

$$x = \left((\ \mathcal{C} + a_5 \cdot u - u\ )\, e^{(a_5-1)t} - \mathcal{C}\ \right)/(a_5 - 1) \quad , \text{for } |x| \leq 1 \tag{1.10a}$$

$$x = (-\mathcal{C} - a_5 + 1)\, e^{-t} + \mathcal{C} + a_5 \quad , \text{for } 1 \leq x \tag{1.10b}$$

$$x = (-\mathcal{C} + a_5 - 1)\, e^{-t} + \mathcal{C} - a_5 \quad , \text{for } x \leq -1 \tag{1.10c}$$

We would like to know under what conditions the state x of the cell goes to 1 for (1.10a). It can be easily shown that this occurs for the case of (1.11).[12]

$$(a_5 - 1)u + \mathcal{C} > 0 \tag{1.11}$$

[11] For convenience, we drop the indices i and j since we know that we are dealing with one cell.

[12] The sign of the exponential function determines if the derivative of x is positive or negative.

Thus, the state x tends to leave the region $|x| \leq 1$ at 1 only if condition (1.11) is satisfied. Alternatively, x tends to -1 only in the case of (1.12).

$$(a_5 - 1)u + \mathcal{C} < 0 \tag{1.12}$$

Interesting behaviour can be observed in the case when $(a_5 - 1)u + \mathcal{C} = 0$: from the solution (1.10a) of the differential equation (1.7a) we see that the state of the cell remains at its initial condition, i.e., $x = u$, for $\forall\, t \geq 0$.

Once the state leaves the region $|x| \leq 1$ at 1 or -1 because the corresponding inequality (1.11) or (1.12) is satisfied, the dynamics of the cell is determined by the linear differential equations in (1.7b) and (1.7c), respectively. From their solutions, one can see that the state does not fall back into the region $|x| < 1$: the factors of the exponential functions in (1.10b) and (1.10c) have the right sign, i.e., for $1 \leq x$ the sign is not positive, and for $x \leq -1$ it is not negative. This can be proven from the steady-state value $x(t \to \infty) = x^\infty$ of the cell for $1 \leq x$ and $x \leq -1$:[13]

$$x^\infty = \mathcal{C} + a_5 \geq \quad 1 \tag{1.13a}$$

$$x^\infty = \mathcal{C} - a_5 \leq -1 \tag{1.13b}$$

The inequalities in (1.13a) and (1.13b) are always satisfied for (1.11) and (1.12), respectively.[14]

In summary, the sign of $(a_5 - 1)u + \mathcal{C}$ determines the output value $y(t \to \infty) = y^\infty$ of the cell.[15] Taking (1.8) into consideration, we can state the following behaviour for the cell:

$$\mathcal{U} > -(a_5 - 1 + b_5)\,u - I \qquad \Longrightarrow y^\infty = \quad 1 \tag{1.14a}$$

$$\mathcal{U} < -(a_5 - 1 + b_5)\,u - I \qquad \Longrightarrow y^\infty = -1 \tag{1.14b}$$

[13] To compute the steady-state values, set $t \to \infty$ in (1.10b) and (1.10c), respectively.

[14] To prove the first case, rewrite the inequality in (1.13a) as $\frac{\mathcal{C}}{(a_5-1)} \geq -1$, and then use (1.11) and the assumption $|u| \leq 1$.

[15] In the case of $(a_5 - 1)u + \mathcal{C} = 0$, $y^\infty = x(0) = u$.

1.5 Global Dynamics

Assuming space invariance of the templates A and B, the dynamics of a one-layer $M \times N$ CNN with a neighbourhood of radius $r = 1$ is determined by nineteen parameters independently of the number of cells: $a_1, \ldots, a_9$, $b_1, \ldots, b_9$, and I (which is supposed to be equal for all cells). The question that now arises is how these parameters have to be set in order to achieve a desired dynamic behaviour. Normally, and this means for CNN applications in image processing, the transient behaviour is of no interest, i.e., it is not important how the CNN reaches a desired stable state. The parameters are set such that all output voltages y_{ij} saturate either to -1 or $+1$ (see Figure 1.3) for $t \to \infty$.[16] Thus, the parameters are chosen such that they determine a certain mapping of an input image onto its corresponding output image. By contrast, we will see in Section 5.2, that in the case where the CNN is operated as a linear filter, it remains in a transient (unsettled) state, and the templates have to determine the CNN dynamics such that the transient behaviour of the CNN corresponds to that of a filter biquad.

1.6 Example

We will give a simple example of how the CNN is used to perform an image-processing task: *edge extraction* .[17] The normalized input and output values u_{ij} and y_{ij}, respectively, are coded as grey-scale values, i.e., $-1 \leq u_{ij}, y_{ij} \leq 1$, where -1 corresponds to white and 1 to black. Every image pixel is attached to a cell, i.e., there are as many cells as image pixels, where the input image is stored in $u_{ij}, 1 \leq i \leq M$, $1 \leq j \leq N$, and the desired output image — in our example the edges of the input image — is expected to appear at y_{ij}, $1 \leq i \leq M$, $1 \leq j \leq N$ after the CNN has converged.[18] Figure 1.5 illustrates this example for a 16×16 image. In fact, we will show below (see also [11]) that with the templates given in Table 1.3 the edges of a black and white image are correctly extracted independently of the initial state $x_{ij}(0), 1 \leq i \leq M$, $1 \leq j \leq N$, and the number of cells.

[16] Positive feedback ($a_5 > 1$) and a symmetric A template ($a_1 = a_9, a_2 = a_8, a_3 = a_7, a_4 = a_6$, see Figure 1.4) are sufficient conditions to guarantee convergence [1, 15]. A CNN with symmetric A template is called a *reciprocal* CNN.

[17] For the definition of edge extraction see page 21.

[18] Convergence time in a CNN chip is in the range of microseconds independently of M and N [4, 5]. Propagation-type applications (see Section 2.5) may take longer.

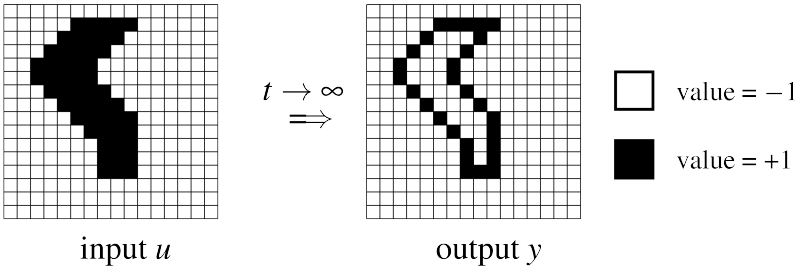


Figure 1.5: *Input image and desired output image for edge extraction ($M = N = 16$)*

A		
0	0	0
0	$1+q/4$	0
0	0	0

B		
0	$-q$	0
$-q$	$4q$	$-q$
0	$-q$	0

$I = -q/2, \quad q > 0$

Table 1.3: *Templates for edge extraction. Note that the solution depends on a parameter q. (These template values are valid for the normalized nonlinear differential equation with $C = R = 1$.)*

1.6.1 Derivation of parametric template values for edge extraction

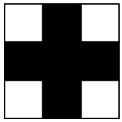
For the situation in Table 1.3, the corresponding normalized[19] differential equation of cell $C(i, j)$ within the range $|x_{ij}| \leq 1$ is the following (see (1.3) and Table 1.1):

$$\frac{dx_{ij}}{dt} = (a_5 - 1)x_{ij} + d \tag{1.15}$$

where d is a constant which depends on the input values around cell $C(i, j)$.

[19] $R = C = 1$

Because of symmetry, we can set all the off-center elements of template B to the same value b_2. Then, for the input situation



around cells $C(i,j)$, $1 < i < M$, $1 < j < N$, the constant $d = 4b_2 + b_5 + c$,[20] and for cells $C(i,j)$, $i = M$, $1 < j < N$, the following input situation

leads to $d = 3b_2 + b_5 + c$.[21] The corresponding outputs for these two input situations must be -1 and 1, respectively, because, in the first situation, the center cell belongs to the inner part of the figure and does not form part of the edge, and, in the second situation, the middle cell of the second row, i.e., cell $C(i,j)$, is at the lower boundary of the CNN array, and therefore, it cannot belong to the inner part of a figure. Thus, for $b_5 = -4b_2$, the following conditions can be derived from equation (1.15) for both input situations:[22]

$$\frac{dx_{ij}}{dt} = (a_5 - 1)x_{ij} \qquad + c \; < 0 \tag{1.16a}$$

$$\frac{dx_{ij}}{dt} = (a_5 - 1)x_{ij} - b_2 + c \; > 0 \tag{1.16b}$$

Setting $a_5 = 1 - b_2/4$ and $c = b_2/2$ we get the following inequalities from (1.16a) and (1.16b):

$$-b_2/4 \; x_{ij} + b_2/2 < 0 \tag{1.17a}$$

$$-b_2/4 \; x_{ij} - b_2/2 > 0 \tag{1.17b}$$

[20] The parameters b_1, b_3, b_7 and b_9 are supposed to be zero, and $I = c$, see Table 1.3.

[21] It can be shown that the discussion of these two input situations is sufficient to bound the parameter values.

[22] For negative and positive derivatives of x_{ij}, the state x_{ij}, and therefore also the output of the cell, tend to -1 and 1, respectively.

Inequalities (1.17a) and (1.17b) are always fulfilled for $|x_{ij}| \leq 1$ and $b_2 < 0$.

Now, we have to consider the case were $|x_{ij}| \geq 1$. The differential equations for both input situations around cell $C(i,j)$ are the following:[23]

$$\frac{dx_{ij}}{dt} = -x_{ij} \pm (1 - b_2/4) + b_2/2 \tag{1.18a}$$

$$\frac{dx_{ij}}{dt} = -x_{ij} \pm (1 - b_2/4) - b_2/2 \tag{1.18b}$$

For (1.18a) we have to ask for the steady state $x_{ij}(t \to \infty)$ to be ≤ -1 (first input situation), for (1.18b) the steady state has to be ≥ 1 (black output value for the second input situation). First, we analyse (1.18a):

$$\frac{dx_{ij}}{dt} = -x_{ij} + 1 + b_2/4, \qquad \text{for } x_{ij} \geq 1 \tag{1.19a}$$

$$\frac{dx_{ij}}{dt} = -x_{ij} - 1 + 3/4\ b_2, \quad \text{for } x_{ij} \leq -1 \tag{1.19b}$$

The expression (1.19a) is always < 0 for $b_2 < 0$, i.e., the transient moves in the right direction towards the linear part of the piecewise-linear function. (1.19b) is also < 0 for $b_2 < 0$ and $-1 + 3/4\ b_2 < x_{ij} \leq -1$. At $x_{ij} = -1 + 3/4\ b_2$ (final state) the derivative $dx_{ij}/dt = 0$, and for $x_{ij} < -1 + 3/4\ b_2$ and, again, $b_2 < 0$, the derivative dx_{ij}/dt is positive and pushes the value of x_{ij} towards $x_{ij} = -1 + 3/4\ b_2$.

The discussion of (1.18b) can proceed in a similar way:

$$\frac{dx_{ij}}{dt} = -x_{ij} + 1 - 3/4\ b_2, \quad \text{for } x_{ij} \geq 1 \tag{1.20a}$$

$$\frac{dx_{ij}}{dt} = -x_{ij} - 1 - b_2/4, \qquad \text{for } x_{ij} \leq -1 \tag{1.20b}$$

The expression (1.20a) is > 0 for $b_2 < 0$ and $1 \leq x_{ij} < 1 - 3/4\ b_2$. It is $= 0$ for $x_{ij} = 1 - 3/4\ b_2$ (final state), and for $b_2 < 0$ and $1 - 3/4\ b_2 < x_{ij}$ it is negative pushing the value of x_{ij} towards $x_{ij} = 1 - 3/4\ b_2$. (1.20b) is always > 0 for $b_2 < 0$.

Setting $q = -b_2$ we get the result of Table 1.3.

[23] The sign of the expression in brackets is positive for $x_{ij} \geq 1$, and negative for $x_{ij} \leq -1$.

1.7 Summary

A CNN is a time-continuous, nonlinear dynamical system. It consists of an array of identical cells which are locally connected, i.e., there are no direct links between distant cells. The way a given cell is influenced by its neighbouring cells is controlled by the network parameters. They are the same for all cells (cloning templates).

The cells of a CNN are all simultaneously activated from and by an input signal array (i.e., an image); the processing therefore takes place in all cells simultaneously, i.e., *in parallel*. Because each cell is identical, the dynamics of the CNN is determined by the dynamics *of an individual cell and its neighbours*. This goes for the state of convergence. Thus, by and large, both the analysis of the dynamics of a CNN, and the design itself, can be narrowed down to the dynamics and design of an individual cell and its neighbours. The resulting simplicity with regard to insight, predictability, and amenability to analysis, as compared to, say, a perceptron type artificial neural network (ANN) is considerable. More important still, this simplicity also carries over into the physical realizability, in particular in VLSI form, of a CNN.

CNNs are normally used to perform image-processing tasks. The templates are set such that a given image is mapped onto a desired output image. Every image pixel is processed by a cell. The *analysis* of the behaviour of a CNN cell for given template values can easily be performed, because the nonlinear function at the output of the cell is piecewise linear. The *synthesis* of the templates for a given task is a more difficult problem. We showed how a parametric solution for the simple edge-extraction problem was determined. A design method which bounds all possible solutions is given in the next chapter.

Due to the local connectivity, CNNs with a large number of cells can be implemented on analogue chips, although most present realizations have fixed template values, i.e., the CNN chips are not programmable. Their convergence time is in the range of microseconds which means that the potential processing power of large programmable CNN chips is high.

Chapter 2

Exact Design of Reciprocal CNNs with Binary Inputs

Based on two classes of equilibrium equations, a design method for reciprocal CNNs with binary inputs is presented. The local rules defining the task to be accomplished by the network are directly mapped into a set of linear inequalities that bound the solution space of the network parameters for the given problem. All points in the solution space guarantee the correct operation of the network. A solution can be computed by the relaxation method for solving sets of linear inequalities.

2.1 Obtaining the Network Parameters

A CNN performs a nonlinear mapping of an input signal onto an output (see previous chapter). The mapping is completely defined by the space-invariant network parameters of the CNN. Each application such as, e.g. edge extraction or shadowing, needs its own parameter values to process the input to the desired output. Locally connected CNNs with space-invariant parameters are able to carry out tasks which can completely be described by local rules involving only neighbouring cells.

Several design methods for synthesizing CNNs have been proposed. In [16] a training rule was presented to determine the weights of the network from input image patterns and desired output image patterns. By learning from examples, it is assumed that, after training, the network correctly processes inputs which have never been shown before. Unfortunately, however, the proposed design method does not guarantee the desired outputs for a given input learning set.

An analytic method for designing simple CNNs has also been published [17]. It uses rules that explicitly describe the task to be accomplished by the network. These rules establish a set of inequalities that must be satisfied by the network parameters. A solution of this set of inequalities guarantees correct operation of the network. Nevertheless, it may be difficult to construct such a set of inequalities for a given task, and for some cases the method is too restrictive, resulting in an empty solution space.

The design method proposed in this thesis *maps the rules that define the application directly into a set of linear inequalities* using two classes of equilibrium equations: The first class contains *desired* equilibrium states, the second contains *forbidden* ones. *Any point in the solution space bounded by this set of linear inequalities guarantees correct operation of the network for a given task.* The parameter values can be computed by the relaxation method for the solution of sets of linear inequalities [16]. The design method can be applied to local-type, as well as to propagation-type applications, such as shadowing (see Section 2.5).

2.2 Desired and Forbidden Equilibrium States

The first-order nonlinear differential equations defining the dynamics of a reciprocal CNN with neighbourhood $N_1(i,j)$ (radius equal to 1), and with binary input values can be written as follows:[1]

$$\begin{aligned} \frac{dx_{ij}}{dt} + x_{ij} &= \mathbf{y}_{ij}^T \mathbf{a} + \mathbf{u}_{ij}^T \mathbf{b} + c \\ &= \boldsymbol{\gamma}_{ij}^T \boldsymbol{\Theta} \end{aligned} \tag{2.1a}$$

$$1 \le i \le M \ , \ 1 \le j \le N$$

where

$$\begin{aligned} \boldsymbol{\gamma}_{ij}^T &= \left[\mathbf{y}_{ij}^T \ \mathbf{u}_{ij}^T \ 1 \right] \\ &= \left[\gamma_1^{jj} \ \gamma_2^{jj} \ \cdots \ \gamma_{19}^{jj} \right] \end{aligned} \tag{2.1b}$$

$$\begin{aligned} \boldsymbol{\Theta}^T &= \left[\mathbf{a}^T \ \mathbf{b}^T \ c \right] \\ &= [\ a_1 \ \ a_2 \ \cdots \ a_9 \ \ b_1 \ \ b_2 \ \cdots \ b_9 \ \ c\ ] \end{aligned} \tag{2.1c}$$

$$y_{ij}(t) = \frac{1}{2}\left(|x_{ij}(t)+1| - |x_{ij}(t)-1| \right) \tag{2.1d}$$

(piecewise-linear function)

The CNN is subjected to the following restrictions:

$$|u_{ij}| = 1 \ , \quad \forall \ i,j \tag{2.1e}$$

(binary input)

$$\begin{aligned} a_1 = a_9 \ \ , \ \ a_2 = a_8 \\ a_3 = a_7 \ \ , \ \ a_4 = a_6 \end{aligned} \tag{2.1f}$$

(symmetry condition)

$$a_5 > 1 \tag{2.1g}$$

(parameter assumption)

[1] Equation (2.1a) is the normalized version of equation (1.3) or, equivalently, equation (1.4), i.e., $R = C = 1$, and $I = c$ (where c stands for a constant).

Equation (2.1e) restricts the input to assume binary values, (2.1f) is the symmetry condition for reciprocal CNNs, and (2.1g) forces positive feedback. Table 2.1 illustrates the correspondence between vector **a** and the space-invariant cloning template *A*.

a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

Table 2.1: *Component correspondence for vector* **a** *and template A (see also Table 1.1 on page 8)*

The output values y_{ij} of the $M \times N$ CNN converge to ± 1 for $t \to \infty$ because of the restriction to reciprocal CNNs ((2.1f) — symmetry condition), parameter assumption (2.1g) and the piecewise-linear characteristic (2.1d) of the nonlinear function [1]. This means that each vector component γ_m^{ij}, $1 \leq m \leq 9$, converges to ± 1 for $t \to \infty$. On the other hand, because of the binary input condition (2.1e), $|\gamma_m^{ij}| = 1$ results for $10 \leq m \leq 18$ and $t \geq 0$. Thus, $\boldsymbol{\gamma}_{ij}(t \to \infty) = \boldsymbol{\gamma}_{ij}^{\infty} \in \{-1,\ 1\}^{19}$ contains binary components ± 1 which correspond to output and input values in the neighbourhood $N_1(i, j)$ of cell $C(i, j)$.[2]

Let Γ be the set of all $\mathcal{N}$ possible vectors $\boldsymbol{\gamma}_i^{\infty}$, $i = 1,\ 2, \cdots, \mathcal{N}$, with different component values. Next let Γ_d and Γ_f be disjoint subsets of Γ containing those desired and forbidden vectors $\boldsymbol{\gamma}_d^{\infty}$ and $\boldsymbol{\gamma}_f^{\infty}$, whose components correspond to input/output combinations which are prescribed by, respectively should not appear in, a specific application, i.e., a specific mapping function F

$$F:\ \{-1,\ 1\}^{M \times N} \ \longrightarrow\ \{-1,\ 1\}^{M \times N} \tag{2.2}$$

Then for all $\boldsymbol{\gamma}_i^{\infty} \in \Gamma_d$, i.e., for vectors $\boldsymbol{\gamma}_d^{\infty}$, the following inequalities formulate the conditions that the parameter, or, template vector $\boldsymbol{\Theta}$ must satisfy to enable the CNN to settle to a desired output state:

$$\boldsymbol{\gamma}_d^T \boldsymbol{\Theta} \geq \quad 1, \quad \text{for } \gamma_5^d = \quad 1 \tag{2.3a}$$

$$\boldsymbol{\gamma}_d^T \boldsymbol{\Theta} \leq -1, \quad \text{for } \gamma_5^d = -1 \tag{2.3b}$$

For clarity, the shorter notation $\boldsymbol{\gamma}_d^T$ has been used instead of $(\boldsymbol{\gamma}_d^{\infty})^T$. [3]

[2] The last component of vector $\boldsymbol{\gamma}_{ij}^{\infty}$ is always equal to unity due to $\gamma_{19}^{ij} = 1$ (see (2.1b)), and is not related to input or output values.

[3] Inequalities (2.3) result from equations (2.1a) and (2.1d), and from the fact that $\frac{dx_i}{dt} = 0$ and $y_i = \pm 1$ for $t \to \infty$ [1].

On the other hand, vectors $\boldsymbol{\gamma}_f^\infty$ belonging to the set Γ_f of forbidden combinations define the following inequalities that prevent the transient of the CNN from converging to an undesired output state for a given input:

$$\boldsymbol{\gamma}_f^T \boldsymbol{\Theta} < \quad 1, \quad \text{for } \gamma_5^f = \quad 1 \tag{2.4a}$$

$$\boldsymbol{\gamma}_f^T \boldsymbol{\Theta} > -1, \quad \text{for } \gamma_5^f = -1 \tag{2.4b}$$

Here $\boldsymbol{\gamma}_f^T$ stands for $(\boldsymbol{\gamma}_f^\infty)^T$. With inequalities (2.4) and for a fixed template vector $\boldsymbol{\Theta}$, any state variable x_i that produces an undesired output $y_i = \pm 1$ is pushed back from the outer parts of the piecewise-linear function into the linear region $|x_i| < 1$ in order to let the CNN search for another equilibrium state. If the resulting vector $\boldsymbol{\gamma}_i$ of this new equilibrium state also belongs to Γ_f then the state variable is pushed back again until a desired equilibrium state is reached. Inequalities (2.3) assure that such an equilibrium state exists.

In summary, inequalities (2.3) and (2.4) together with (2.1g) and symmetry condition (2.1f) bound the solution space of template vector $\boldsymbol{\Theta}$. The correct operation of the CNN for any point within the solution space is guaranteed by this procedure. The elements of Γ_d and Γ_f can be directly derived from the local rules which define F, as demonstrated by the next examples.

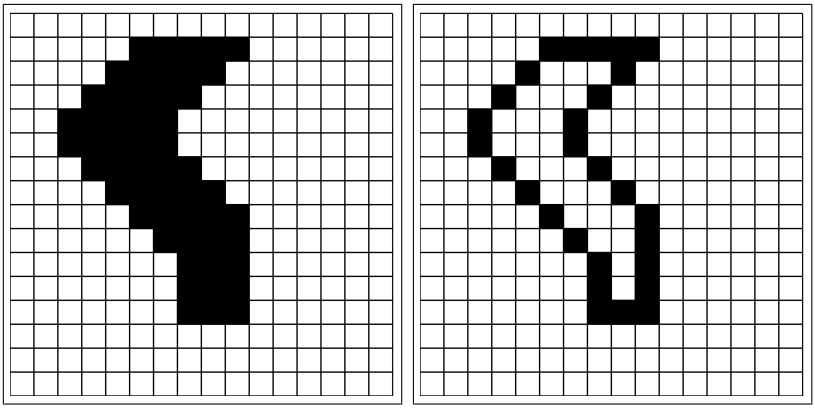
2.3 Example 1: Edge Extraction

The task of extracting the edge of a figure can be completely described by local rules which only involve cells $C(k,l)$ belonging to the corresponding neighbourhood $N_1(i,j)$. This is essential because applications which need the direct influence of a cell $C(k,l) \notin N_1(i,j)$ on $C(i,j)$ cannot be realized with the common definition (2.1) of a CNN, because there is no direct connection between cells outside the neighbourhood and $C(i,j)$. Tasks which are defined by local rules have the advantage that the correct operation of the network does not depend on its size, i.e., on the values of constants M and N. The rules for edge extraction can be stated as follows (compare with Figure 2.1):
A cell does not belong to the edge of a figure if

Rule 1: the cell does not belong to the figure itself, i.e., if its input value is white (or, equivalently, -1, because -1 corresponds to white)

Rule 2: the cell is in the inner part of the figure, i.e., if its input value is black (or equivalently 1, because 1 corresponds to black) *and* the input values of cells $C(i-1,j)$, $C(i,j-1)$, $C(i,j+1)$ and $C(i+1,j)$ are also

black.[4]



(a) (b)

Figure 2.1: *(a) input pattern,*
(b) the desired output pattern for edge extraction in a 16×16 *layer* [5]

These rules deal only with five cells, namely the center cell and the off-diagonal cells within the neighbourhood. For the center cell the input and the output values are required, for the off-diagonal cells only the input value is needed (see Table 2.2). Moreover, the space symmetry of the edge-extraction

	u_2^{ij}	
u_4^{ij}	u_5^{ij}	u_6^{ij}
	u_8^{ij}	

y_5^{ij}

Table 2.2: *Inputs and the output involved in the edge-extraction problem*

problem allows us to weigh the off-diagonal input values with the same parameter $b_\#$. Thus, a reduced template vector is sufficient for the correct operation

[4] Another definition of the edge of a figure may take also cells $C(i-1, j-1)$, $C(i-1, j+1)$, $C(i+1, j-1)$ and $C(i+1, j+1)$ into account [2].

[5] This is the two-dimensional image that was used in [16] to train the network.

of the network:

$$\boldsymbol{\Theta}^T = [\ a_5\ \ b_\#\ \ b_5\ \ c\ ], \tag{2.5a}$$

where

$$b_\# = b_2 = b_4 = b_6 = b_8$$

Vectors $\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}_i^\infty$ are modified and now contain only the relevant input and output values and have again the same size as template vector $\boldsymbol{\Theta}$:

$$\begin{aligned} \boldsymbol{\gamma}_i^T &= \left[\gamma_5^i\ \ \gamma_{11}^i + \gamma_{13}^i + \gamma_{15}^i + \gamma_{17}^i\ \ \gamma_{14}^i\ \ 1\right] \\ \Rightarrow \boldsymbol{\gamma}_i^\infty &= \left[{}^*\gamma_1^i\ \ {}^*\gamma_2^i\ \ {}^*\gamma_3^i\ \ 1\ \right]^T \end{aligned} \tag{2.5b}$$

where

$$\begin{aligned} {}^*\gamma_1^i &= \gamma_5^i(t \to \infty) \\ {}^*\gamma_2^i &= \gamma_{11}^i + \gamma_{13}^i + \gamma_{15}^i + \gamma_{17}^i \\ {}^*\gamma_3^i &= \gamma_{14}^i \end{aligned}$$

Vector component ${}^*\gamma_1^i$ corresponds to the output of cell $C(\cdot,\cdot)$, ${}^*\gamma_3^i$ contains its input value, and ${}^*\gamma_2^i$ is the sum of the off-diagonal inputs around cell $C(\cdot,\cdot)$. ${}^*\gamma_3^i$ and ${}^*\gamma_1^i$ can take on values ± 1, i.e., the cell's input and output are either black or white.

Next, we have to differentiate between cells at a corner, at the border, or in the inner part of the CNN in order to compute the values of vector component ${}^*\gamma_2^i$. It is obvious that for an inner cell ${}^*\gamma_2^i$ can take on values $-4, -2,\ 0,\ 2,\ 4$ depending on the sum of white (-1) and black (1) off-diagonal cells[6] around cell $C(\cdot,\cdot)$. For example, if $C(\cdot,\cdot)$ is surrounded by white off-diagonal cells then ${}^*\gamma_2^i = -4$, but if one of these cells is black then ${}^*\gamma_2^i = -3 + 1 = -2$. For border cells, one of the elements of the sum $\gamma_{11}^i + \gamma_{13}^i + \gamma_{15}^i + \gamma_{17}^i$ is zero because a border cell is surrounded only by three off-diagonal cells. For corner cells two elements of this sum disappear. The following relations summarize the results:

$${}^*\gamma_1^i,\ {}^*\gamma_3^i \in \{-1,\ 1\ \} \tag{2.6a}$$

$${}^*\gamma_2^i \in \begin{cases} \{-2,\ 0,\ 2\ \} & \text{(for corner cells)} \\ \{-3, -1,\ 1,\ 3\ \} & \text{(for border cells)} \\ \{-4, -2,\ 0,\ 2,\ 4\ \} & \text{(for inner cells)} \end{cases} \tag{2.6b}$$

[6] The input values of the cells are added.

From the first rule of the edge-extraction problem it follows that if the input of a cell is white then the cell has to produce a white output, i.e., whenever ${}^*\gamma_3^i = -1$ the output ${}^*\gamma_1^i$ must also be -1. The second rule states that whenever ${}^*\gamma_3^i = 1$ (black input), ${}^*\gamma_1^i$ also has to be 1 unless the inputs of the off-diagonal cells are all black, i.e., ${}^*\gamma_2^i = 4$. From this, the set Γ_d can be constructed in a straightforward procedure assigning to it all vectors $\boldsymbol{\gamma}_i^\infty$ whose components accomplish one of the following conditions:

(*i*) ${}^*\gamma_3^i = -1 \ \wedge \ {}^*\gamma_1^i = -1$

(*ii*) ${}^*\gamma_3^i = 1 \ \wedge \ {}^*\gamma_1^i = 1 \ \wedge \ {}^*\gamma_2^i \in \{-4, -3, -2, -1, \ 0, \ 1, \ 2, \ 3 \}$

(*iii*) ${}^*\gamma_3^i = 1 \ \wedge \ {}^*\gamma_1^i = -1 \ \wedge \ {}^*\gamma_2^i = 4$

Equation (2.7) shows the result.

$$\Gamma_d = \left\{ \underbrace{\begin{bmatrix} -1 \\ -2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \end{bmatrix},}_{\text{corner cells and inner cells}} \right.$$

$$\underbrace{\begin{bmatrix} -1 \\ -3 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 3 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -3 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \\ 1 \\ 1 \end{bmatrix},}_{\text{border cells}} \tag{2.7}$$

$$\left. \underbrace{\begin{bmatrix} -1 \\ -4 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 4 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -4 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 4 \\ 1 \\ 1 \end{bmatrix}}_{\text{inner cells}} \right\}$$

Γ_f can easily be constructed using the disjoint property of both sets Γ_d and Γ_f and the fact that *all* other vectors $\boldsymbol{\gamma}_i^\infty$ whose components do not accomplish

any of the conditions (*i*) – (*iii*) lead to forbidden equilibrium states:

$$\Gamma_f = \Gamma \backslash \Gamma_d \tag{2.8}$$

We now have all the inequalities that bound the solution space of template vector $\boldsymbol{\Theta} \in \mathbb{R}^4$ such that any point in the solution space guarantees the correct operation of the CNN for edge extraction, independently of the sizes M and N and the initial state.[7] A solution that satisfies inequalities (2.1g), (2.3) and (2.4) can be computed by the relaxation method for solving sets of linear inequalities [16]. The following equation gives a solution of the template vector $\boldsymbol{\Theta}$ for the edge-extraction problem:

$$\boldsymbol{\Theta} = \begin{bmatrix} 1.0559 \\ -0.2615 \\ 1.2436 \\ -0.3419 \end{bmatrix} \tag{2.9}$$

Table 2.3 orders the same result in template notation.

0	−0.2615	0
−0.2615	1.2436	−0.2615
0	−0.2615	0

cloning template B

0	0	0
0	1.0559	0
0	0	0

cloning template A

$I = -0.3419$

constant current source

Table 2.3: *Edge Extraction: same parameter values as in equation (2.9) ordered in template notation*

[7] The local rules for edge extraction do not involve the initial state of the CNN.

2.3.1 Sensitivity of the solution point

Comparing Tables 2.3 and 1.3, one can see that the edge-extraction solution given in the previous section is not in accordance with the templates given in Table 1.3, *although both guarantee the correct operation of the CNN for edge extraction independently of the initial state of the network*. The parametric templates given in the previous chapter, Table 1.3, take only a subset of all possible solutions into account, and the solution point in (2.9) *lies outside the corresponding subspace*, i.e., it is one of the points not taken into account by the parametric solution. But the solution found here has the drawback that it is more sensitive to deviations from the exact template values: changes in the non-zero template values which are greater than $\pm 1.7\%$ push the point out of the solution space [18], whereas template values with $q = 1$ in Table 1.3 have a tolerance of $\pm 2.5\%$. (For $q = 2$ and the less practical value $q = 10$ the tolerances are $\pm 2.7\%$ and $\pm 2.8\%$, respectively.) Thus, the point given in (2.9) is nearer to the border of the solution space than the points in the subspace given by the parametric solution in Table 1.3.

The sensitivity of the solution plays a role in the chip implementation of analogue CNNs, e.g., because tolerances of some integration processes make it nearly impossible to realize exact parameter values [6]. Thus, the requirement for a low sensitivity of the solution point to deviations from the exact parameter values means that a point in the "middle" of the solution space should be considered, in order to (possibly) avoid that it leaves the solution space due to parameter mismatch. This is an additional constraint that should be taken into account in the design process of templates [19].

2.4 Example 2: Horizontal Line Detection

As a second example for the design of reciprocal CNNs, we apply our method to the example of horizontal line detection. In the original paper [2], a horizontal line detector with the following templates was presented:

$$A = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} \qquad B = 0 \qquad I = 0$$

Table 2.4: *Templates given in [2] for the detection of horizontal lines*

The input pattern is loaded into the initial state $x_{ij}(0)$, $1 \leq i \leq M$, $1 \leq j \leq N$. Unfortunately, however, a CNN with these template values does not always operate correctly, as can be observed from simulations. We will derive in this section template values that *guarantee the correct operation* of the network for horizontal line detection.

First, let us define the parameter set in the following way:[8]

A		
a_4	a_5	a_4

$B = 0 \quad I = c \neq 0$

Table 2.5: *Template definition for the horizontal line detector*

Next, we determine the elements of sets Γ_d and Γ_f (see page 20). Table 2.6 lists the eight possible output situations for the three cells $C(i, j-1)$, $C(i, j)$

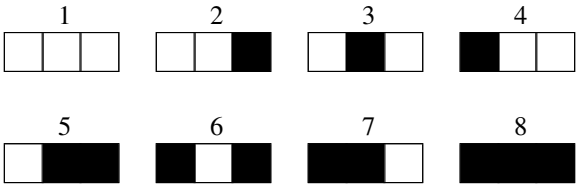


Table 2.6: *The eight possible output situations within the neighbourhood of an* inner *cell* $C(i, j)$

and $C(i, j+1)$ within the neighbourhood of an *inner* cell $C(i, j)$, $1 \leq i \leq M$, $1 < j < N$. At the right and left border of the layer, i.e., for cells $C(i, j)$ with $1 \leq i \leq M, j = 1, N$, we have the output situations of Table 2.7.[9]

Table 2.7: *The four possible output situations within the neighbourhood of a* border *cell* $C(i, j)$ *(right hand border)*

The multiplicative factor of parameter a_4 can take on values $-2, -1, 0, 1, 2$

[8] Because of symmetry, we can set $a_6 = a_4$.

[9] Again, because of symmetry, it is sufficient to look only at the right hand border of the CNN layer.

depending on the black and white output values of the off-center cells.[10] For parameter a_5 only the two weights ± 1 are possible, i.e. black or white.

Now, we only have to assign the elements of Γ

$$\Gamma = \left\{ \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.10}$$

to the sets of desired and forbidden combinations. Here is the result:[11]

$$\Gamma_d = \left\{ \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.11a}$$

$$\Gamma_f = \left\{ \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.11b}$$

From the elements of Γ_d and Γ_f, inequalities (2.3) and (2.4) can be constructed which, together with parameter assumption (2.1g), bound the solution space of the CNN templates for horizontal line detection. The values of Tables 2.8 and 2.9 satisfy all these inequalities, thus, for the given solution points a correct operation of the network is guaranteed.[12]

$$A = \begin{array}{|c|c|c|} \hline 0.5 & 1.75 & 0.5 \\ \hline \end{array} \qquad B = 0 \qquad I = -0.5$$

Table 2.8: *Template values for the horizontal line detector: this parameter point lies within the solution space for correct operating templates.*

[10] The value -1 is possible for the first two situations of Table 2.7.

[11] The task of horizontal line detection can be defined with the following rule: isolated (black) pixels must not appear at the output. Thus, the situations 3 and 2 in Tables 2.6 and 2.7, respectively, have to be assigned to the set Γ_f of forbidden combinations.

[12] Both solution points were randomly chosen within the solution space, i.e., they are not optimized in terms of, e.g., sensitivity.

A		
1	2	1

$B = 0 \quad I = -1$

Table 2.9: *Template values for the horizontal line detector: a second parameter point within the solution space for correct operating templates*

If we compare the values of Tables 2.4 and 2.9 we see that the only difference lies in the value for the constant current source I, but this difference is sufficient to guarantee proper operation for the latter solution.

2.5 Example 3: Shadowing

If we think of a black pattern on the CNN layer as an object illuminated from the right by a parallel-light source, then the cells on the left of the pattern have to produce black output values representing the shadow of the object.

A template set that performs this task was first published in [20] (see Table 2.10). The input u_{ij}, $1 \leq i \leq M$, $1 \leq j \leq N$, is initialized with the in-

A			B			
0	2	2	0	2	0	$I = 0$

Table 2.10: *Templates used in [20] for shadowing*

put pattern, $x_{ij}(0)$ is set to 1 for all cells.

In [17] a solution with the control operator $B = 0$ could be derived from the presented analytic design method (see Table 2.11). Here, the initial

A		
0	2	1

$B = 0 \quad I = 0.5$

Table 2.11: *Templates used in [17] for shadowing: the control operator B is equal to zero.*

state $x_{ij}(0)$, $1 \leq i \leq M$, $1 \leq j \leq N$, is initialized with the input pattern.

The solutions given in Tables 2.10 and 2.11 both have *asymmetrical* feedback templates A. With the exact design method presented in this chapter,

A		
q	$1+q$	q

B		
$-q$	$2q$	0

$I = 3q, \quad q > 0$

Table 2.12: *Templates for shadowing: the feedback template A is symmetric. The solution depends on a parameter q. (The bigger q the faster the shadow propagates to the left hand side.)*

a solution with *symmetric* template A can be given (see Table 2.12). The asymmetry is transferred to the control operator B. The input u_{ij}, $1 \leq i \leq M$, $1 \leq j \leq N$, is initialized with the input pattern, and the initial state $x_{ij}(0)$ is set equal to -1 for all cells. This solution cannot be found with the analytic design method of [17] because it requires too many restrictions.

We will derive in the following sections both, the asymmetrical solution given in Table 2.11, and the symmetric solution given in Table 2.12. Of course, in the case of the asymmetrical feedback template A, an additional stability analysis would be needed. With our design method, this analysis usually can be avoided by considering only reciprocal CNNs which are always stable (see Footnote 16 on page 12). Since the design method of [17] guarantees the solution to be stable, we skip the stability analysis (see below).

2.5.1 Exact design applied to a non-reciprocal CNN

The shadowing problem can be completely defined with the following three rules:

Rule 1 A black output has to remain black.

Rule 2 A white output has to become black if the output of the cell at the right hand side is black.

Rule 3 A white output keeps its value as long as the output of the cell at the right hand side is white.

With these three rules the shadow propagates from the left-most cell whose output is black to the left CNN border cell by cell.[13] The parameter set can be defined in the following way:

[13] The radius of the neighbourhood of cell $C(i, j)$ is restricted to $r = 1$, thus, we have to proceed cell by cell.

$$A$$

0	a_5	a_6

$B = 0 \quad I = c$

Table 2.13: *Template definition for shadowing from the right hand side*

Because of the relation $a_4 = 0 \neq a_6$ (asymmetrical feedback template A), the corresponding CNN is non-reciprocal [21]. Again, although our exact design method usually is used for reciprocal CNNs, it *can* be applied to the shadowing task. Intuitively, we can already now say that the three rules given above guarantee convergence for our shadowing task because we force the correct output for any given state.[14]

The elements of Γ are given in the next equation. (The first four elements correspond to all possible output combinations for cells $C(i, j)$, $1 \leq i \leq M$, $1 \leq j < N$, and the last two elements to those for cells $C(i, j)$, $1 \leq i \leq M$, $j = N$, i.e., at the right hand border of the CNN layer.)

$$\Gamma = \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right\} \tag{2.12}$$

It is easy to assign the right elements to the sets Γ_d and Γ_f of *desired* and *forbidden* combinations, respectively. The next equations show the result:

$$\Gamma_d = \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right\} \tag{2.13a}$$

$$\Gamma_f = \left\{ \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.13b}$$

Now, we can construct from the sets Γ_d and Γ_f given in (2.13a) and (2.13b), respectively, the two classes of inequalities (2.3) and (2.4) that, together with (2.1g), bound the solution space for the shadowing templates. The solution given in Table 2.11 satisfies the set of inequalities.

[14] In fact, the solution given in Table 2.11 which, as we will see, belongs to the solution space derived in this section, has been proved to be stable [17]. (Moreover, the analytic design method in [17] automatically produces stable solutions.)

2.5.2 Shadowing with a symmetric template A

For shadowing with a reciprocal CNN (see Footnote 16 on page 12), we define the templates in the following way:

A			B			
a_4	a_5	a_4	b_4	b_5	0	$I = c$

Table 2.14: *Template definition for shadowing with a reciprocal CNN*

Instead of using parameters b_4 and b_5, one could think of taking b_5 and b_6 into account. But this parameter set would not give enough information about the actual state of the network:
Consider the following situation. The inputs of cells $C(i, j)$ and $C(i, j+1)$ are both white. Because the outputs of cells $C(i, j-1)$ and $C(i, j+1)$ are both weighted with the same parameter a_4, the cell $C(i, j)$ does not know whether the output of cell $C(i, j-1)$ or that of cell $C(i, j+1)$ has become black. For the latter case, $C(i, j)$ has to produce a black output, i.e., it must propagate the shadow, but for a black input at cell $C(i, j-1)$, which is invisible at position (i, j) for $b_4 = 0$, $C(i, j)$ does not need to produce a black output. Thus, cell $C(i, j)$ does not know which output it should produce. If cell $C(i, j)$ has information on the input value of cell $C(i, j-1)$ ($b_4 \neq 0$) then it knows whether the black output value is at position $(i, j-1)$ or $(i, j+1)$. Two further remarks have to be made: First, we also need parameter b_5 because otherwise there would be a lack of information at the right hand border of the CNN layer. And secondly, the initial state $x_{ij}(0)$ has to be set equal to -1 for all cells to avoid false decisions caused by the symmetric weighting of the outputs of cells $C(i, j-1)$ and $C(i, j+1)$.[15]

Thus, for the parameter set of Table 2.14 we have the input situations with the corresponding desired and forbidden output patterns listed in Table 2.15.

The output patterns

must not be forbidden, otherwise the output pattern

[15] Of course, one can design a template set with the three parameters b_4, b_5 and b_6, all $\neq 0$.

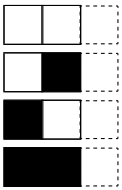
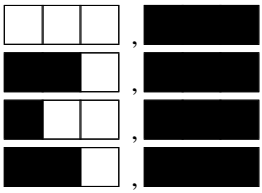
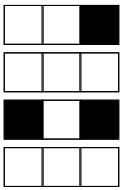
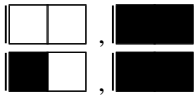
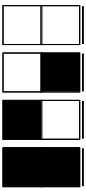
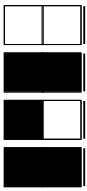
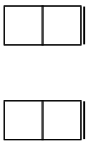
cell $C(i,j)$, $1 \leq i \leq M$	input situation	*desired* output patterns	*forbidden* output patterns
$1 < j < N$			
$j = 1$			
$j = N$			

Table 2.15: *Input situations and the corresponding desired and forbidden output patterns within the neighbourhood $N_1(i,j)$ of cell $C(i,j)$*

cannot be reached (the cell outputs become black, step by step). If the input of cell $C(i, j)$ is white then the output patterns

cannot be forbidden otherwise the solution space results in an empty one.[16]

From Table 2.15 the elements of Γ_d and Γ_f can directly be derived.[17]

$$\Gamma_d = \left\{ \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \right.$$
$$\left. \begin{bmatrix} 0 \\ -1 \\ -1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 0 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.14a}$$

$$\Gamma_f = \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 0 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{2.14b}$$

[16] In any case, the two output patterns cannot appear as false final states, i.e., they do not lead to erroneous shadows, because they are already covered by other cases. (Actually, they correspond to the initial state.)

[17] The correspondence of the vector elements can be seen from the template vector: $\boldsymbol{\Theta}^T = [\ a_4\ \ a_5\ \ a_6\ \ b_4\ \ b_5\ \ c\ ]$, where $a_6 = a_4$.

The solution given in Table 2.12 satisfies the symmetry condition $a_4 = a_6$, parameter assumption $a_5 > 1$, and the inequalities (2.3) and (2.4) that can be constructed from the sets Γ_d and Γ_f, respectively.

2.6 Summary

In this chapter, an exact design method for reciprocal CNNs with binary inputs has been presented. Given an application, two disjoint sets of vectors can be derived directly from the local rules that define the task to be accomplished by the network. The sets contain desired and forbidden input/output combinations, leading to appropriate inequalities which bound the parameter space. *Any point in the solution space, i.e., any set of parameter values that satisfy all inequalities, guarantees the correct operation of the CNN for the given task.*

The exact design method has been demonstrated in detail first for the problem of edge extraction. A CNN with parameter values that satisfy the constructed inequalities extracts the edge of any pattern at any position in a two-dimensional layer of arbitrary size, independently of the initial state.[18]

The second example was horizontal line detection. We derived templates which guarantee the correct operation of the CNN for the detection of horizontal lines. The solution was computed in a straightforward procedure by listing all possible output situations within the neighbourhood of inner and border cells.

Finally, we constructed two different solutions for a propagation-type application: shadowing. The feedback template A of the first solution was asymmetrical. Thus, our design method can also be used for non-reciprocal CNNs, although a stability analysis is then required. Secondly, we applied our method to derive a solution with a symmetric A template for shadowing. The presented design method can be used to find template values for other propagation-type applications such as, e.g., hole filling [22] or connected component detection (CCD) [23].

[18]The convergence time depends on the initial state, e.g., if the initial state is set equal to the edge of the figure at the input of the CNN, the convergence time is zero, because the solution is already given by the initial state.

Chapter 3

Separating Capability

The CNN is able to perform different image-processing tasks depending on the template values, i.e., on the network parameters, used. In the case of linear templates, the parameter space is divided into different regions by hyperplanes. Every region is associated with a task, such that all points within that region let the CNN perform the desired task. In this chapter, a lower and an upper bound for the number of regions that can be separated with binary-input CNNs are given, thus answering the question of how many different tasks such a CNN can perform.

3.1 Different CNN Applications

Every CNN application, such as edge extraction or shadowing, requires its own parameter values to process the input into the desired output. Figure 2.1 on page 22 shows an input pattern (a) and the corresponding output pattern (b) for edge extraction.

An exact design method for reciprocal CNNs with binary input values has been presented in Chapter 2. Given an application (in our first example edge extraction), the parameter values which let the CNN process the input into the desired output can be computed by a straightforward procedure. The values given in Table 2.3 on page 25 solve the task of extracting the edge of an *arbitrary* figure in a two-dimensional layer independently of the initial state. These values correspond to one point in the solution space of our specific application.

Assuming the CNN's structure remains fixed, it is interesting to know how many different applications the CNN can be used for, if only the template values are varied. This question is related to the capability of a given network to solve different tasks by changing the parameter values, or, stated differently, the required complexity or size of a CNN for a given task. In what follows we shall refer to the *Separating Capability of a CNN*, meaning the number S of regions in a given parameter space that a given CNN can separate. Each region is associated with a different processing task such as, e.g., edge extraction, shadowing or hole filling, and every point within such a region guarantees the correct operation of the network for the corresponding task. However, we cannot predict which tasks have a solution, i.e., there might be empty regions. This is due to the fact that the maximum number of regions that can be separated with a given CNN may be smaller than the number of possible tasks. Whether a task can be performed or not with a given CNN has to be checked numerically.

3.2 Separating Regions in Parameter Space with Hyperplanes

The different regions in parameter space that stand for different applications are separated by hyperplanes,[1] which, in the case of CNNs, divide the

[1] Hyperplanes in the case of linear templates.

parameter space in a particular manner. For convenience, let us again write the first-order nonlinear differential equations (2.1) which define the dynamics of a reciprocal CNN with binary input values:

$$\begin{aligned} \frac{dx_{ij}}{dt} + x_{ij} &= \mathbf{y}_{ij}^T \mathbf{a} + \mathbf{u}_{ij}^T \mathbf{b} + c && \text{(3.1a)} \\ &= \boldsymbol{\gamma}_{ij}^T \boldsymbol{\Theta} \\ 1 \le i \le M &\ , \ 1 \le j \le N \\ \text{where} \qquad \boldsymbol{\gamma}_{ij}^T &= \left[\mathbf{y}_{ij}^T \ \mathbf{u}_{ij}^T \ 1 \right] && \text{(3.1b)} \\ &= \left[\gamma_1^{ij} \ \gamma_2^{ij} \ \cdots \ \gamma_{19}^{ij} \right] \\ \boldsymbol{\Theta}^T &= \left[\mathbf{a}^T \ \mathbf{b}^T \ c \right] && \text{(3.1c)} \\ &= [\ a_1 \ \ a_2 \ \cdots \ a_9 \ \ b_1 \ \ b_2 \ \cdots \ b_9 \ \ c\ ] \\ y_{ij}(t) &= \frac{1}{2}(|x_{ij}(t)+1| - |x_{ij}(t)-1|) && \text{(3.1d)} \end{aligned}$$

with the following restrictions:

$$|u_{ij}| = 1 \ , \qquad \forall\, i, j \tag{3.1e}$$

$$\begin{aligned} a_1 = a_9 \ \ , \ \ a_2 = a_8 \\ a_3 = a_7 \ \ , \ \ a_4 = a_6 \end{aligned} \tag{3.1f}$$

$$a_5 > 1 \tag{3.1g}$$

Each vector component γ_m^{ij}, $1 \le m \le 9$, converges to ± 1 for $t \to \infty$ [1]. The vector components γ_m^{ij}, $10 \le m \le 18$, are ± 1 for $t \ge 0$, see the binary-input restriction (3.1e). Thus, $\boldsymbol{\gamma}_{ij}(t \to \infty) = \boldsymbol{\gamma}_{ij}^{\infty} \in \{-1,\ 1\}^{19}$ contains binary components ± 1 which correspond to output and input values in the neighbourhood $N_1(i, j)$ of cell $C(i, j)$.[2] Let Γ be the set of all $\mathcal{N} = 2^{n-1}$ possible vectors $\boldsymbol{\gamma}_i^{\infty}$, $i = 1,\ 2, \cdots, \mathcal{N}$, with different component values $\{-1,\ 1\}$.[3] The variable n denotes the dimension of the parameter space, i.e., the number of parameters (which, in the above case is 19). Next, Γ_d and Γ_f, as defined on page 20,

[2] Again, see page 20, the vector $\boldsymbol{\gamma}_{ij}^{\infty}$ does not strictly belong to $\{-1,\ 1\}^{19}$ because its last component $\gamma_{19}^{ij} = 1$.

[3] See Sections 2.3 to 2.5 for the case where the vector components may take on other values than $\{-1,\ 1\}$.

are disjoint subsets of Γ containing those desired and forbidden vectors $\boldsymbol{\gamma}_d^\infty$ and $\boldsymbol{\gamma}_f^\infty$, whose components correspond to input/output combinations which are prescribed by, respectively should not appear in, a specific application, i.e., a specific mapping function F (see (2.2)). For all $\boldsymbol{\gamma}_i^\infty \in \Gamma_d$, i.e., for vectors $\boldsymbol{\gamma}_d^\infty$, the following inequalities formulate the conditions that the parameter, or, template vector $\boldsymbol{\Theta}$ must satisfy to enable the CNN to settle to a desired output state (identical to inequalities (2.3)):

$$\boldsymbol{\gamma}_d^T \boldsymbol{\Theta} \geq \quad 1, \quad \text{for } \gamma_5^d = \quad 1 \tag{3.2a}$$

$$\boldsymbol{\gamma}_d^T \boldsymbol{\Theta} \leq -1, \quad \text{for } \gamma_5^d = -1 \tag{3.2b}$$

Here, again, $\boldsymbol{\gamma}_d^T$ stands for $(\boldsymbol{\gamma}_d^\infty)^T$.

On the other hand, vectors $\boldsymbol{\gamma}_f^\infty$ belonging to the set Γ_f of forbidden combinations define the following inequalities to assure that the transient of the CNN cannot converge to an undesired output state for the given input (identical to inequalities (2.4)):

$$\boldsymbol{\gamma}_f^T \boldsymbol{\Theta} < \quad 1, \quad \text{for } \gamma_5^f = \quad 1 \tag{3.3a}$$

$$\boldsymbol{\gamma}_f^T \boldsymbol{\Theta} > -1, \quad \text{for } \gamma_5^f = -1 \tag{3.3b}$$

Now, $\boldsymbol{\gamma}_f^T$ stands for $(\boldsymbol{\gamma}_f^\infty)^T$. Inequalities (3.2) and (3.3) together with (3.1g) and symmetry condition (3.1f) bound the solution space of template vector $\boldsymbol{\Theta}$. The correct operation of a reciprocal CNN with binary input values is guaranteed by this procedure for any point within the solution space. The elements of Γ_d and Γ_f can be directly derived from the local rules which define F, i.e., from the specific application, as shown in Chapter 2 (see also [24]).

From inequalities (3.2a) and (3.3a) on the one hand (when $\gamma_5^i = 1$), and (3.2b) and (3.3b) on the other ($\gamma_5^i = -1$), we can define two different types of hyperplanes in the parameter space:

$$\boldsymbol{\gamma}_i^T \boldsymbol{\Theta} - 1 = 0, \quad \text{type I) when } \gamma_5^i = \quad 1 \tag{3.4}$$

$$\boldsymbol{\gamma}_i^T \boldsymbol{\Theta} + 1 = 0, \quad \text{type II) when } \gamma_5^i = -1 \tag{3.5}$$

It can easily be shown that whenever $\gamma_5^i = 1$ (i.e., type I)) the corresponding hyperplane $\boldsymbol{\gamma}_i^\infty$ passes through the point $P_1 : [0\ 0\ \cdots\ 0\ 1]^T$ in parameter space. For type II) the hyperplane defined by vector $\boldsymbol{\gamma}_i^\infty$ is shifted through

point $P_2 : [0\ 0\ \cdots\ 0\ -1]^T$. The correspondence of $\boldsymbol{\gamma}_i^\infty$ either to Γ_d or Γ_f decides which side of the hyperplane $\boldsymbol{\gamma}_i^\infty$ (going either through P_1 or P_2) the solution has to lie on. Γ, as defined above, is the set of all possible vectors $\boldsymbol{\gamma}_i^\infty$ with different component values. Thus, half of all possible combinations contain vector component $\gamma_5^j = 1$, the other half $\gamma_5^j = -1$.

3.3 Results

From the previous section, let us now assume that the CNN with a fixed structure leads to $\mathcal{N} = \mathcal{N}_1 + \mathcal{N}_2$ different hyperplanes $\boldsymbol{\gamma}_i^\infty$, of which $\mathcal{N}_1$ hyperplanes pass through point P_1 and $\mathcal{N}_2$ through point P_2.[4] There are $\mathcal{A} = 2^\mathcal{N}$ possible assignments of the $\mathcal{N}$ hyperplanes $\boldsymbol{\gamma}_i^\infty$ either to Γ_d or Γ_f, although not all such assignments may be meaningful, e.g., it makes no sense to assign all $\boldsymbol{\gamma}_i^\infty$ to Γ_f. As we already mentioned above, the assignment either to Γ_d or Γ_f decides which side of the hyperplane the solution space has to be on (equations (3.2) and (3.3), respectively). Thus, the solution space is the intersection of $\mathcal{N}$ half-spaces, one for each hyperplane $\boldsymbol{\gamma}_i^\infty$.

Not all intersections, and therefore not all possible assignments, may lead to a non-empty solution space. This is guaranteed only for $n \geq \mathcal{N}$, where n is the dimension of the parameter space [25]. For $n < \mathcal{N}$, and even in the most general case, i.e., when the $\mathcal{N}$ hyperplanes are in *general position* [5] in the space, there might be assignments without common intersection of all half-spaces. Figure 3.1 illustrates the situation for $n = 2$ and $\mathcal{N} = 3$: there is no common intersection of the three shaded half-spaces. Interpreting this for our application, it means that in such a case there is no solution $\boldsymbol{\Theta}$ that satisfies inequalities (3.2) and (3.3).

We will derive a lower and an upper bound on the number S of regions in parameter space that can be separated by a CNN. We start from the general case mentioned above, i.e., when the hyperplanes are not restricted to pass through any given points in parameter space, and then the CNN-specific situation will be introduced step by step.

The number of possible assignments which do not result in an empty solution space for $\mathcal{N}$ hyperplanes in general position, corresponds to the number

[4] We generalize the situation although we already know that $\mathcal{N}_1 = \mathcal{N}_2 = \mathcal{N}/2$.

[5] In *general position* means that no two hyperplanes are parallel and no $n+1$ hyperplanes meet at the same point.

of separable regions in parameter space. The result for the number $s(\mathcal{N}, n)$ of different regions that can be separated in $\mathbb{R}^n$ by $\mathcal{N}$ hyperplanes of dimension $(n-1)$ in general position and in the case where $n < \mathcal{N}$, can be found in a mathematical treatise of the mid-nineteenth century [25–27]. The result relevant for our problem is repeated in equation (3.6).[6]

$$s(\mathcal{N}, n) = \sum_{k=0}^{n} \left(\begin{array}{c} \mathcal{N} \\ k \end{array} \right) \tag{3.6}$$

Figure 3.2 numbers the different $s = 7$ regions for $n = 2$ and $\mathcal{N} = 3$. Thus, in this example, only seven out of $2^3 = 8$ assignments lead to a non-empty solution space.

The number $s_o(\mathcal{N}, n)$ of separable regions in $\mathbb{R}^n$ when all $\mathcal{N}$ hyperplanes of dimension $(n-1)$ pass through the same point P in the space, is given in equation (3.7) [25, 26, 28].

$$s_o(\mathcal{N}, n) = 2 \sum_{k=0}^{n-1} \left(\begin{array}{c} \mathcal{N}-1 \\ k \end{array} \right) \tag{3.7}$$

This situation may be interpreted in the case of $n = 2$ and $\mathcal{N} = 3$, as if region 7 in Figure 3.2 had shrunk to point P. All the separable regions s_o are *unbounded*, or, "open", regions.

The number $s_c(\mathcal{N}, n)$ of regions that get lost by the restriction of making all hyperplanes pass through a common point P is the difference between $s(\mathcal{N}, n)$ and $s_o(\mathcal{N}, n)$:

$$s_c(\mathcal{N}, n) = s(\mathcal{N}, n) - s_o(\mathcal{N}, n) \tag{3.8}$$

[6] The relation $s(\mathcal{N}, n) < \mathcal{A} = 2^{\mathcal{N}}$ holds, as expected for $n < \mathcal{N}$.

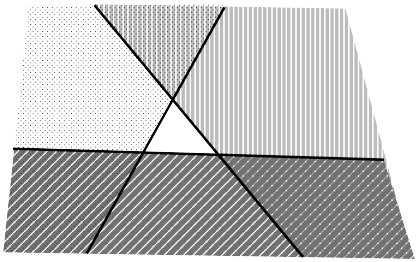


Figure 3.1: *Possible empty solution space in $\mathbb{R}^n$, $n = 2$, for $\mathcal{N} = 3$: there is no common intersection of all three half-spaces*

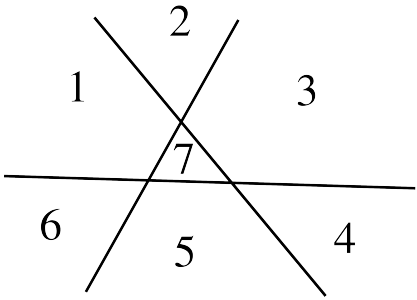


Figure 3.2: *For $\mathcal{N} = 3$ straight lines in general position in the plane, the number of separable regions is $s = 7$.*

These regions are *bounded* (convex) regions. Region 7 in Figure 3.2 is a bounded region.

The goal is still to compute the number S of regions that can be separated by a CNN in parameter space, i.e., in $\mathbb{R}^n$. The configuration is the one given above, i.e., $\mathcal{N}_1$ hyperplanes passing through point P_1 and $\mathcal{N}_2$ through point P_2. We assume that apart from this condition, the hyperplanes are in a general position. To develop the solution, the number $\triangle s_c(\mathcal{N}, n)$ of *bounded* regions which add to the (unbounded and bounded) regions in $\mathbb{R}^n$ by increasing the number of hyperplanes by one,[7] is introduced in the next equation:

$$\triangle s_c(\mathcal{N}, n) = s_c(\mathcal{N} + 1, n) - s_c(\mathcal{N}, n) \tag{3.9}$$

Figure 3.3 shows the $\triangle s_c = 2$ new closed regions which appear in the configuration of Figure 3.2 by adding a fourth straight line.

Let us start from the situation where $\mathcal{N}_1$ hyperplanes pass through point P_1. Then, we add $\mathcal{N}_2$ new hyperplanes one by one, all passing through point P_2, until we reach the total number $\mathcal{N}$ of hyperplanes. Each time a new hyperplane is added, the number of bounded regions is increased by $\triangle s_c$. However, these new hyperplanes are not in general position because of the restriction that all new hyperplanes shall pass through point P_2. This restriction is responsible for an annihilation of bounded regions,[8] but only for $\mathcal{N}_2 > n$. The total number of *unbounded* regions is not affected by the CNN-specific configuration, i.e., by the restriction that $\mathcal{N}_1$ hyperplanes pass through point P_1 and the rest through point P_2.

[7] In general position

[8] See the remark above where region 7 in Figure 3.2 shrinks to one point.

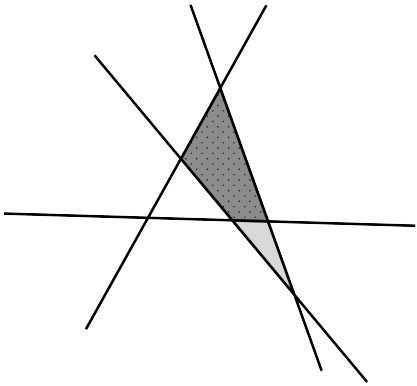


Figure 3.3: $\triangle s_c = 2$ *new closed regions are introduced in the configuration of Figure 3.2 by adding a fourth straight line in general position.*

From the previous discussions, the maximum number s_r of regions that can be separated with the configuration given above, i.e., $\mathcal{N}_1$ hyperplanes passing through point P_1 and $\mathcal{N}_2$ through point P_2 in $\mathbb{R}^n$, can be easily computed. The result is given in the next equation, where $\mathcal{N} = \mathcal{N}_1 + \mathcal{N}_2$:

$$s_r = s_o(\mathcal{N}, n) + \sum_{k=0}^{\mathcal{N}_2-1} \triangle s_c(\mathcal{N}_1 + k, n) - \sum_{j=n}^{\mathcal{N}_2-1} \triangle s_c(j, n) \tag{3.10a}$$

$$= s_o(\mathcal{N}, n) - s_c(\mathcal{N}_1, n) + s_c(\mathcal{N}_1 + \mathcal{N}_2, n) - \left(- \underbrace{s_c(n, n)}_{=0} + s_c(\mathcal{N}_2, n) \right) \tag{3.10b}$$

The first term in (3.10a) corresponds to the total number of unbounded regions. The second term corresponds to the bounded regions that are added by the $\mathcal{N}_2$ hyperplanes which pass through P_2, and the third term to the bounded regions that shrink onto point P_2. The result in (3.10a) is an upper bound because we assume that, apart from the restriction given above, the hyperplanes are in general position. Using equation (3.9), most terms in the sums of (3.10a) annihilate each other. The corresponding simplification of equation (3.10a) is given in (3.10b).

The situation in a CNN

For the situation in a CNN we have to take (3.1g), i.e., that $a_5 > 1$, into account.[9] This introduces another hyperplane in $\mathbb{R}^n$. Moreover, being aware of the fact that for a CNN $\mathcal{N}_1 = \mathcal{N}_2 = \mathcal{N}/2$, the following results can be derived from equation (3.10a):

$$S_{\max}(\mathcal{N},n) = s_o(\mathcal{N}+1,n) + \sum_{k=0}^{\mathcal{N}/2} \triangle s_c(\mathcal{N}/2+k,n) - \sum_{j=n}^{\mathcal{N}/2-1} \triangle s_c(j,n) \quad (3.11a)$$

Again, cancelling terms within the sums, we obtain:

$$S_{\max}(\mathcal{N},n) = s_o(\mathcal{N}+1,n) + s_c(\mathcal{N}+1,n) - 2s_c(\mathcal{N}/2,n) \quad (3.11b)$$

Equations (3.11) give the upper bound $S_{\max}$ for the separating capability S of a CNN, i.e., the *maximum* number S of regions in parameter space that can be achieved by a CNN, as its parameter values are varied. The explanation why (3.11) is the upper bound for the separating capability and not the exact number of regions in parameter space, is the following: since the $\mathcal{N}$ hyperplanes $\boldsymbol{\gamma}_i^{\infty}$ are determined by the combination of output and input values in the neighbourhood of a CNN cell, they may not lie in general position (apart from passing through points P_1 and P_2, see above), as required for the computation of $S_{\max}$. Moreover, the exact number of separable regions could only be given by investigating how the hyperplanes lie in parameter space.

Given the $\mathcal{N}$ hyperplanes $\boldsymbol{\gamma}_i^{\infty}$ (which are all different), the lower bound for the separating capability of a CNN is given by

$$S_{\min}(\mathcal{N},n) = s_o(\mathcal{N}+1,n) = 2\sum_{k=0}^{n-1} \begin{pmatrix} \mathcal{N} \\ k \end{pmatrix} \quad (3.12)$$

This is the case when all $\mathcal{N}$ hyperplanes pass through the same point $P_3 \neq P_{1,2}$ in parameter space. The difference $(S_c)_{\max}$ between $S_{\max}$ and $S_{\min}$

$$(S_c)_{\max}(\mathcal{N},n) = S_{\max}(\mathcal{N},n) - S_{\min}(\mathcal{N},n) \quad (3.13)$$

[9] The symmetry condition (3.1f) is irrelevant because it leads to an appropriate reduction of the parameter space.

corresponds to the *maximum* number of bounded convex regions in parameter space.

The even number $\mathcal{N}$ of hyperplanes $\boldsymbol{\gamma}_i^{\infty}$ in parameter space is equal to the number of elements in Γ (see above). The set Γ, and therefore its size $\mathcal{N}$, can be derived immediately from the structure of the CNN, i.e., from its template definitions. The variable n denotes the number of parameters.

In summary, the separating capability S of a CNN varies within the range

$$S_{\min} \leq S \leq S_{\max} \tag{3.14}$$

and, as we see from the following equations, its value can be split into two parts S_o and S_c corresponding to unbounded and bounded convex regions, respectively:

$$S = S_o + S_c \tag{3.15a}$$

where

$$S_o = S_{\min} \tag{3.15b}$$

$$0 \leq S_c \leq (S_c)_{\max} \tag{3.15c}$$

3.4 Simplification of Formulas

Equations (3.8)–(3.11) can be simplified in the following way: for (3.8) we have

$$\begin{aligned} s_c(\mathcal{N}, n) &= s(\mathcal{N}, n) - s_o(\mathcal{N}, n) \\ &= \sum_{k=0}^{n} \binom{\mathcal{N}}{k} - 2\sum_{k=0}^{n-1} \binom{\mathcal{N}-1}{k} \\ &= \binom{\mathcal{N}}{n} + \sum_{k=0}^{n-1} \binom{\mathcal{N}}{k} - 2\sum_{k=0}^{n-1} \frac{\mathcal{N}-k}{\mathcal{N}} \binom{\mathcal{N}}{k} \\ &= \binom{\mathcal{N}}{n} + \sum_{k=0}^{n-1} \frac{2k-\mathcal{N}}{\mathcal{N}} \binom{\mathcal{N}}{k} \\ &= \binom{\mathcal{N}}{n} - \frac{n}{\mathcal{N}} \binom{\mathcal{N}}{n} \end{aligned}$$

$$= \left(1 - \frac{n}{\mathcal{N}}\right) \binom{\mathcal{N}}{n} \tag{3.16}$$

Thus, the number $s_c(\mathcal{N}, n)$ of bounded regions can be computed with equation (3.16) avoiding the evaluation of sum terms.

Using this result, we can give the following explicit formula for equation (3.9):

$$\begin{aligned}
\triangle s_c(\mathcal{N}, n) &= s_c(\mathcal{N}+1, n) - s_c(\mathcal{N}, n) \\
&= \left(1 - \frac{n}{\mathcal{N}+1}\right) \binom{\mathcal{N}+1}{n} - \left(1 - \frac{n}{\mathcal{N}}\right) \binom{\mathcal{N}}{n} \\
&= \frac{\mathcal{N}+1-n}{\mathcal{N}+1} \frac{\mathcal{N}+1}{\mathcal{N}+1-n} \binom{\mathcal{N}}{n} - \left(1 - \frac{n}{\mathcal{N}}\right) \binom{\mathcal{N}}{n} \\
&= \frac{n}{\mathcal{N}} \binom{\mathcal{N}}{n}
\end{aligned} \tag{3.17}$$

Next, equation (3.10b) becomes

$$\begin{aligned}
s_r &= s_o(\mathcal{N}, n) + s_c(\mathcal{N}_1 + \mathcal{N}_2, n) - s_c(\mathcal{N}_1, n) - s_c(\mathcal{N}_2, n) \\
&= 2 \sum_{k=0}^{n-1} \binom{\mathcal{N}-1}{k} \\
&\quad + \frac{\mathcal{N}_1 + \mathcal{N}_2 - n}{\mathcal{N}_1 + \mathcal{N}_2} \binom{\mathcal{N}_1 + \mathcal{N}_2}{n} - \frac{\mathcal{N}_1 - n}{\mathcal{N}_1} \binom{\mathcal{N}_1}{n} \\
&\quad - \frac{\mathcal{N}_2 - n}{\mathcal{N}_2} \binom{\mathcal{N}_2}{n}
\end{aligned} \tag{3.18}$$

Finally, equation (3.11b) simplifies to:

$$\begin{aligned}
S_{\max}(\mathcal{N}, n) &= s_o(\mathcal{N}+1, n) + s_c(\mathcal{N}+1, n) - 2 s_c(\mathcal{N}/2, n) \\
&= 2 \sum_{k=0}^{n-1} \binom{\mathcal{N}}{k}
\end{aligned}$$

$$
\begin{aligned}
&+\frac{\mathcal{N}+1-n}{\mathcal{N}+1}\binom{\mathcal{N}+1}{n}-2\frac{\mathcal{N}-2n}{\mathcal{N}}\binom{\mathcal{N}/2}{n}\\
&=2\sum_{k=0}^{n-1}\binom{\mathcal{N}}{k}\\
&\quad+\frac{\mathcal{N}+1-n}{\mathcal{N}+1}\,\frac{\mathcal{N}+1}{\mathcal{N}+1-n}\binom{\mathcal{N}}{n}\\
&\quad-2\frac{\mathcal{N}-2n}{\mathcal{N}}\binom{\mathcal{N}/2}{n}\\
&=2\sum_{k=0}^{n-1}\binom{\mathcal{N}}{k}+\binom{\mathcal{N}}{n}+\frac{2}{\mathcal{N}}\left(2n-\mathcal{N}\right)\binom{\mathcal{N}/2}{n} \qquad (3.19)
\end{aligned}
$$

Thus, the upper bound $S_{\max}$ for the number of regions in parameter space that can be achieved by a CNN, as its parameter values are varied, is given by the explicit formula (3.19).

3.5 Examples

Let us give two examples to show that the number of different regions in parameter space is often sufficiently large and, therefore, that there are generally many different tasks that can be solved with a given CNN.

3.5.1 Full number of parameters

The first example is the one given by the equations which define the class of CNNs dealt with in this chapter, i.e., equations (3.1). Because of symmetry conditions (3.1f), the number n of parameters is reduced from 19 to 15. The parameter space is then $\mathbb{R}^{15}$.[10] As an approximation, i.e., without regard for effects at the boundaries of the CNN array, $\mathcal{N} \approx 3^4 \cdot 2 \cdot 2^9$. (The weights of the four parameters $a_1, \cdots, a_4$ in (3.1c) can have one of the three possible values -2, 0, or 2, and the weights of a_5 and b_1, $b_2 \cdots b_9$ are ± 1.) The resulting bounds $S_{\min}$ and $S_{\max}$ on the number S of separable regions in parameter space,

[10] Of course, we still could stay in $\mathbb{R}^{19}$, but the four symmetry conditions (3.1f) would force the solution to be in $\mathbb{R}^{19-4}$ anyway.

can be computed with equations (3.12) and (3.19), respectively, and are listed in the first row of Table 3.1. These bounds guarantee the existence of a sufficiently large number of different tasks which can be solved with reciprocal, binary-input CNNs, and a neighbourhood radius $r = 1$.

3.5.2 Reduced number of parameters

The number of CNN parameters in the edge-extraction example of Section 2.3 on page 21 is $n = 4$, and the number of different hyperplanes is $\mathcal{N} = 36$. Again, Table 3.1 lists the resulting bounds on the number S of separable regions. Thus, even such a pruned CNN is capable of solving a large number of different tasks.

n and $\mathcal{N}$	$S_{\min}$	$S_{\max}$
$n = 15$, $\mathcal{N} = 82944$	$\approx 1.67 \cdot 10^{58}$	$\approx 4.62 \cdot 10^{61}$
$n = 4$, $\mathcal{N} = 36$	15614	69759

Table 3.1: *Bounds on the number S of separable regions in parameter space for different CNN structures*

3.6 Summary

In this chapter a lower and an upper bound on the separating capability of CNNs with binary input values have been computed. Separating capability means the number of different tasks that can be processed by a CNN of a certain structure. The different tasks are characterized by (unbounded or bounded, see below) regions in parameter space which are separated by hyperplanes. Every hyperplane stands for a possible combination of input and output values within the neighbourhood of a CNN cell. Half of the hyperplanes pass through a certain point in parameter space, the other half through a second point.[11] The upper bound on the separating capability is the result in

[11] There might be a further hyperplane for parameter assumption (3.1g).

the case where, apart from the previous restrictions, the hyperplanes are in a general position, i.e., no two hyperplanes are parallel and no $n+1$ hyperplanes meet at the same point, where n is the dimension of the parameter space. The lower bound is the result for the case where all hyperplanes meet somewhere at a same point in parameter space. How the hyperplanes lie in parameter space has to be investigated individually for the concrete set of possible input and output combinations within the neighbourhood of a specific CNN structure.

The results reveal that the number of different tasks which can be solved by CNNs with a reduced number of parameters, is in the order of tens of thousands. Taking all parameters of a reciprocal CNN with neighbourhood radius $r = 1$ into account, an immense number of input/output-mappings can be performed with CNNs by varying the template values.[12]

The regions in parameter space can be classified into unbounded and bounded regions. Both types of regions may lead to very sensitive parameter values, i.e., small variations in the coordinate values of a point within such a region may push the point out of that region (see Section 2.3.1 for a concrete example). This may not happen for unbounded regions which, as can be seen from the numbers in Table 3.1 and equations (3.13) to (3.15), appear more rarely in parameter space than bounded regions: the absolute sensitivity can be made arbitrarily low by picking a point within the unbounded region which tends to infinity. However, the relative sensitivity cannot be increased arbitrarily.

The results computed in this chapter can also be interpreted from a different point of view: given a minimum number of different input/output-mappings that a CNN should be able to perform, appropriate values n and $\mathcal{N}$ may be chosen. Or, stated differently, if a required mapping function F cannot be realized with a given CNN because the assignment of input/output-combinations to desired and forbidden mappings results in an empty intersection of the corresponding half-spaces, a solution may be found in a parameter space of higher dimension. Thus, by introducing more parameters, i.e., increasing the number of non-zero template elements, the task defined by F may be solved with the tighter interconnected CNN. The bounds presented in this chapter give a quantitative measure for the number of possible tasks that can be performed with a given CNN.

[12] Notably, $S_{\max} \ll 2^{\mathcal{N}}$, but, in our opinion, this is not a severe drawback due to the rather large number of separable regions.

Part II

The Recognition of Acoustical Alarm Signals with CNNs

The second part of this thesis is dedicated to the capability of CNNs to solve an acoustical-signal classification problem, thereby demonstrating that CNNs are generally suitable for the processing of non-stationary signals.

Chapter 4

Introduction

In this chapter, we introduce the concept of acoustical-signal classification with CNNs. The description of the problem and the solution overview are given. This chapter serves as a guide through the second part of this thesis.

4.1 Classification of Alarm Signals for Hearing Aids

A technical solution for hearing aids must accomplish several requirements in order to be accepted by the hearing impaired: among others it has to be small, light and inconspicuous, the power consumption must be negligible, and the performance has to be as close as possible to that of the human auditory system. The requirement of a small size may be accomplished if the involved electronic circuits are integrated in a single chip. *Analogue* integrated circuits fulfill the requirement of low power consumption [6] and, especially for CNNs [1, 2, 14, 29], they provide fast and powerful signal-processing devices needed for an accurate and reliable audio prosthesis.

The aim of building an artificial ear that functions as perfectly as the one provided by nature is unrealistic. Presently, we restrict ourselves to a well-defined, "simple" task: let the technical device classify signals from four given sets of acoustical alarms (tram bells, car horns, phones, and tram rings) [30, 31]. Thus, when presented with one signal out of a given class, the hearing aid should be able to tell the patient, either by visual or tactile means, which class was active. We consider this to be an appropriate task to investigate the capabilities of CNNs with regard to the recognition of non-stationary signals.

The problem solution for the classification of acoustical alarm signals is divided into three parts (see Figure 4.1). The first performs a transformation of

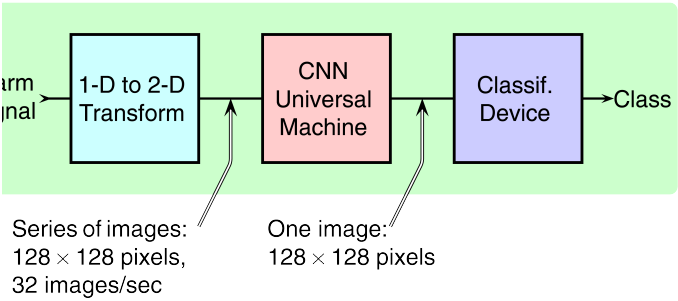


Figure 4.1: *Processing blocks for the classification of acoustical alarm signals*

the one-dimensional acoustical signals into sequences of images. In Figure 4.1, the first block transforms an alarm signal into a series of images which have

a resolution of 128×128 pixels, and which come out of the first block at a rate of 32 images per second. Thus, every input signal is characterized by its corresponding image sequence.

The images serve as input to the second processing block, which is the CNN.[1] The CNN is responsible for the extraction of the "relevant" information carried by the image sequence, and it also performs a time–to–space mapping, concentrating the information distributed over an image sequence into a single image.

For the classification task, the image at the output of the CNN should ideally possess the following properties: output images belonging to signals from the same class should be similar, and output images belonging to signals from different classes should be distinguishable. If this condition is satisfied, the third and last processing block, namely the classification device, is able to separate the different classes of input signals by sorting the output images of the CNN into the correct classes.

4.2 Problem Statement

We take the same sets of signals as in [30, 31] (see Table 4.1). The signals

set	class	# signals	
1	tram bells	57	from 36 trams
2	car horns	63	from 42 cars
3	phone rings	55	from 23 phones
4	tram rings	30	from 26 trams

Table 4.1: *Four different classes of acoustical alarm signals*

were recorded on analogue tape. For our simulations, we use the 8-bit μ-law encoded data[2] sampled at a rate of $f_s = 8$ kHz. The task is to sort a given signal belonging to one of the sets in Table 4.1 into its class by using CNNs. By 'CNNs' we understand the CNN universal machine as presented in [14]. Nevertheless, we require that only those features of the CNN universal machine be used which can be implemented on silicon with reasonable efforts.

[1] This is the reason why the input signals have to be transformed into a 2-D representation in a preprocessing step.

[2] This corresponds to 12-bit linear resolution.

4.3 Solution

As mentioned above, the solution consists of three blocks:

(I) *1-D to 2-D Transformation*
The function of the preprocessing stage is to transform the input signal into an image sequence in order to supply the CNN with images. We will present the method in Section 5.1. Beside representing amplitude information, *the method also preserves phase information of the input signal.*

(II) *CNN*
The CNN's processing is twofold: first, it extracts meaningful patterns from every image of the sequence, and then it combines them by a simple OR-operation to perform the time–to–space mapping. This results in a single image which characterizes the given acoustical alarm signal. The extraction of meaningful patterns can be accomplished by at least three different approaches. They are presented in Chapter 6.

(III) *Classification Device*
A one-layer perceptron is used to classify the images produced by the CNN. It is presented in Chapter 7. The restriction that the one-layer perceptron can separate regions in the input space only linearly (with hyperplanes), is not relevant in our application. This is due to the fact that the characteristic images produced by the CNN fulfill the condition of being similar for signals of the same class, and of being distinguishable for signals of different classes.

Finally, in Chapter 8 the achieved error rates are listed. The rates were achieved without any special tuning to the given alarm signals.

4.4 Summary

In this chapter, a concept for the classification of acoustical alarm signals using CNNs was introduced. The three processing blocks of the solution were sketched pointing to the corresponding chapters with detailed description, thereby providing a guide for the second part of this thesis.

Chapter 5

Phase-Preserving 2-D Representation of Acoustical Signals

The first section of this chapter presents a processing method which transforms acoustical input signals into corresponding image sequences. The most important feature of the method is that besides representing amplitude information, it also preserves phase information of the input signal. Basically, it consists of two parts: the first models the basilar membrane, and the second performs a correlation.

Because the model of the basilar membrane uses filter biquads, i.e., second-order linear filters, we shall present in the second section of this chapter a CNN-compatible realization of arbitrary second-order linear filters or biquads. We shall derive the general biquad solution from two realizations of low-pass filters and from one for band-pass filters. Thus, the CNN realization of arbitrary biquads makes it possible to build up the model of the basilar membrane with CNNs.

5.1 The Binaural Chip

The first part of the so-called *binaural chip* [32] models the basilar membrane in the cochlea of the inner ear. This can accurately be done by a *filter cascade* of second-order low-pass filters [33]. They are equally spaced on the logarithmic frequency range of the ear. Thus, a pole frequency of each filter corresponds to a location along the unrolled basilar membrane where the latter is sensitive to a certain audio frequency (see Figure 5.1). High frequencies are

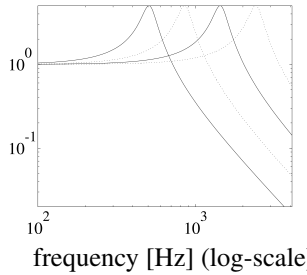


Figure 5.1: *Four equally-spaced low-pass filters along the basilar membrane:* $f_l = 512\ Hz$, $f_u = 4\ kHz$, $q_p = 5$ *(see equations (5.1) and (5.2))*

detected at the beginning of the membrane, low frequencies at the end. The cascade itself acts as a delay line which alters the phase: a low-frequency input signal passes through the first filters without any modification of its magnitude but with a change in phase, i.e., a delay. Considering only the magnitude of the filtered signal and ignoring the phase results in a loss of information. The analogue model of the basilar membrane implemented in the binaural chip preserves that phase information.

Since the CNNs process two-dimensional arrays, the information content of the acoustical signals must be mapped into images. The binaural chip produces a 2-D representation of the audio signals in its second part by correlating the phases of two waves which propagate along two basilar membranes, i.e., along two identical filter cascades.

5.1.1 Model of the binaural chip

Although the analogue binaural chip already works on an experimental level, we built a model of it on a digital computer (see Appendix A.1). This has two advantages: first, because there are no fully programmable CNN chips yet available, and the CNN must be modelled anyway, the whole classification device including the preprocessing stage, i.e., the binaural chip, can be simulated on the computer. The other advantage is that the parameters of the binaural chip can easily be tuned in the model and the behaviour of the tuned parameters observed.

The digital model of the basilar membrane consists of a low-pass filter cascade of K stages. Each stage is the bilinear transformation[1] of the i^{th} second-order low-pass filter in the time-continuous filter cascade. Equation (5.1) shows the transfer function of an analogue low-pass filter where ω_{p_i} denotes the pole frequency of the i^{th} filter, and q_p is the quality factor of the low-pass filters, equal for all filters in the cascade.

$$T_i(s) = \frac{\omega_{p_i}^2}{s^2 + \frac{\omega_{p_i}}{q_p} s + \omega_{p_i}^2} \qquad ; i = 1, 2, \ldots, K \tag{5.1}$$

The pole frequencies of our basilar-membrane model are equally spaced along the logarithmic scale of the frequency range covered by the filter cascade. (See equations (5.2) where f_l and f_u denote the lower and upper frequencies, respectively.)

$$\omega_{p_i} = 2\pi 10^{(K-i)\bar{f} + \log_{10} f_l} \tag{5.2a}$$

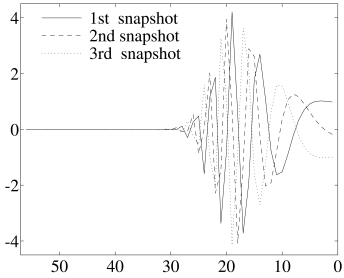
$$\text{where:} \qquad \bar{f} = (\log_{10} f_u - \log_{10} f_l)/K \tag{5.2b}$$

The way in which a sinusoidal input signal travels along the filter cascade is described in [32]. The signal passes the first filter stages without suffering any significant changes in its amplitude,[2] but with a phase shift: the first filters with the highest pole frequencies act as all-pass filters and cause the input signal to be delayed. This way, a wave is generated along the filter cascade.

[1] The bilinear transformation to the z domain preserves the frequency response of the filters in the s domain. The nonlinear distortion of the frequency axis can be compensated in the design procedure of the digital filters [34].

[2] For a sufficiently low input frequency.

Only when the signal arrives at the filter stages whose pole frequencies are in the range of the frequency of the input signal, the amplitude of the wave which propagates along the filter cascade is first enhanced (for a sufficiently large q_p) and then rapidly damped in the next filter stages. The propagation of the wave and its extinction are drawn in Figure 5.2. It shows three snapshots of a 2 kHz-sine input signal on the filter cascade, i.e., on the model of the basilar membrane: at first the signal passes through the filters without a change in its amplitude (right-hand side). At the location where the membrane is most sensitive to a frequency of 2 kHz we see the highest deflection. After that the wave rapidly dies out. Another illustration of the wave propagation is given in



The figure shows the filter-cascade output which models the shaping of the basilar membrane (ordinate, normalized values). The sine wave enters the cascade of 56 (discrete-time) low-pass filters (abscissa) from the right.

Figure 5.2: *Three time snapshots of a* 2 *kHz-sine wave on the basilar membrane*

Figure 5.3 where different snapshots of the shaping of the basilar membrane are shown one behind the other, i.e., the time axis goes from the front to the back of the image.

The 2D-transformation of the acoustical signals is performed in the second part of the binaural chip. In our application, the same signal is applied to two identical filter cascades (binaural).[3] The first is placed horizontally and the second vertically at the lower and right edge of a square, respectively, both of them having the highest-frequency filter stage at the lower-right corner. The crossings of all filter outputs (see Figure 5.4) form the pixels of the 2-D representation which is computed at a rate of r images per second. In our discrete-time model of the binaural chip, the correlation of the output signals coming out of the filter cascades is performed in the following way: for every time

[3]Feeding the binaural chip with two different (e.g. time-delayed) signals, contributes to an orientation detection of the signal source, but not to the signal classification.

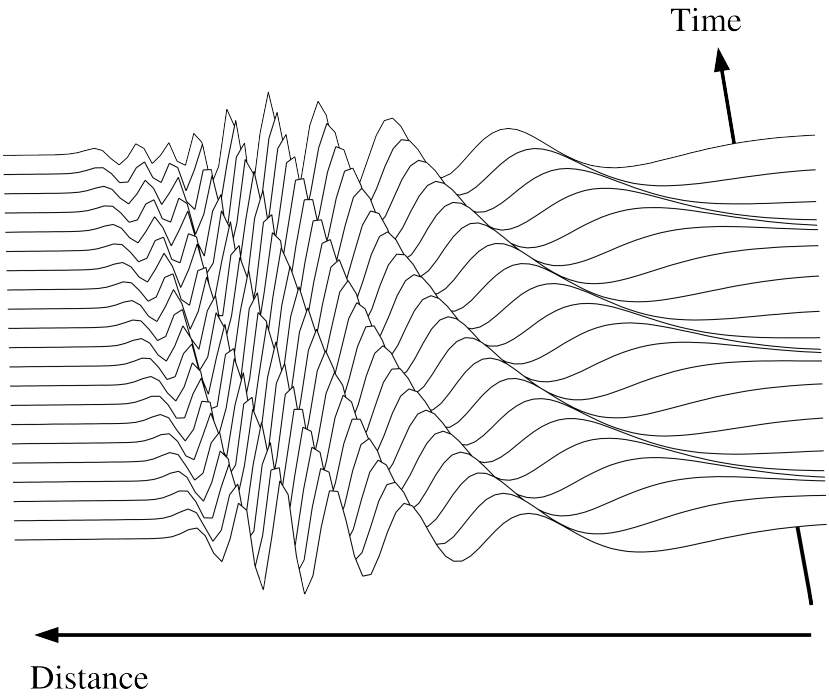
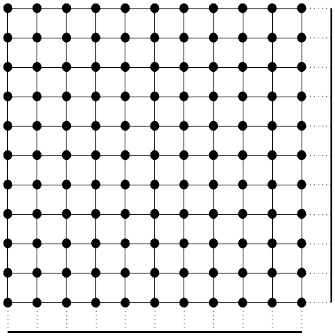


Figure 5.3: *Illustration of the wave propagation on the basilar membrane*

step, the signum function of the filter outputs in both cascades is computed, i.e., the quantitative description of the wave amplitude gets lost, and only its sign is retained. However, small deviations from zero, e.g., in the presence of noise, would then be overvalued, such that a threshold for positive and negative values is introduced within which the filter outputs are still regarded as zero for the computation of the signum function (see variable `threshold` on page 142). Thus, with the signum function we mark the position of the wave on the filter cascade for every time step retaining the phase information of the wave. This reference to the phase at every frequency location, i.e., at every filter output, is used to compute the 2-D representation of the acoustical input signal: at every time step the sign values of the filter outputs of both cascades are multiplied in the way illustrated in Figure 5.4. For every pixel, values $-1, 0$ and 1 are possible. They stand for anticorrelation, no signal, and correlation, respectively. The grey-level pixel values of the final images result from the summation of the values computed at every time step during the



The two filter cascades are placed at the bottom and the right. The crossings of the filter outputs form the pixels of the 2D-representation.

Figure 5.4: *Schematic representation of the 2D-transformation for 11×11 filter outputs*

integration time $1/r$, i.e., during the time window for one image.

The resulting grey-scale images, one every $1/r$ seconds, are symmetric along the diagonal which goes from the upper-left to the lower-right corner because we feed the same signal into two identical filter cascades. Nevertheless, the resulting output signals are spread over two dimensions, i.e., the correlation introduces the second dimension to the binaural-chip output in spite of losing (nearly) half of the pixels in the symmetric region of the output images.

The binaural chip has been implemented as an analogue circuit on a chip [32]. The precision of such networks hardly reaches 8 bits. We took this fact into account by storing the output images of our binaural-chip model with a precision of only 8 bits as well.

5.1.2 Frequency/time trade-off

In our model of the binaural chip, see Section A.1, we can easily change parameter values to check the performance of the whole classification system according to different settings (see Chapter 8). However, the parameter-value selection is constrained by physical properties. Table 5.1 lists the parameters we can set, and how they are denoted in the model.

The number K of filters stands for the frequency resolution of the basilar-membrane model, and is related to the sampling frequency f_s and the image

parameter	name in the model
number K of filters in the cascade	`filters`
lower frequency f_l [Hz]	`fu`
upper frequency f_u [Hz]	`fo`
quality factor q_p	`q`
sampling frequency f_s [Hz]	`fs`
image rate r in images per second	`r`
noise threshold	`threshold`

Table 5.1: *Parameters of the binaural-chip model*

rate r, i.e., the time resolution, in the following way:

$$K = \frac{f_s}{2r} \tag{5.3}$$

Equation (5.3) is a direct consequence of the discrete Fourier transform (DFT) [34]: during the time interval for one image there are f_s/r samples available, and the number of samples on the positive frequency axis is half that amount. Interpreting equation (5.3), it states that the frequency resolution K is proportional to the reciprocal of the time resolution r, i.e.,

$$K \sim \frac{1}{r} \tag{5.4}$$

Thus, if we increase the number of filters in the cascade, we have to lower the rate at which we produce images with our binaural-chip model. The total number P of pixels for the whole image sequence (assuming the signal is L samples long) grows with K^2:

$$P = K^2 \cdot \left(r \frac{L}{f_s} \right) \tag{5.5}$$

The term in parentheses in equation (5.5) denotes the number of images which are produced for the whole signal. From this term, we see that the relation between the number P of pixels and the image rate r is only linear. We are interested in the dependency of P on K taking the frequency/time trade-off of relation (5.4) into account. It results from equations (5.3) and (5.5):

$$P = K \frac{L}{2} \tag{5.6}$$

The lower and upper frequencies f_l and f_u, respectively, should ideally lie in the audible frequency range. The upper frequency f_u and the sampling frequency f_s are related to each other through the sampling theorem [34], i.e., $2f_u < f_s$.

The quality factor q_p is related to the number K of filters. This is due to the fact that on the one hand, q_p has to be sufficiently large to avoid early damping of the wave. On the other hand, if the wave is amplified too much in every filter stage, the deflection may be so large as to cause excessive signal levels in the filters. For a low-pass filter of the form (5.1) (see page 59), the maximum deflection

$$|T(j\omega)|_{\max} = \frac{2q_p^2}{\sqrt{4q_p^2 - 1}} \tag{5.7}$$

is reached at

$$\omega_{\max} = \omega_p \sqrt{1 - \frac{1}{2q_p^2}} \tag{5.8}$$

as can easely be proven by setting the first derivative of $|T(s)|$, $s = j\omega$, equal to zero.

Finally, the noise threshold has been set empirically from simulations.

5.1.3 Output images of the binaural chip

Although the image sequences produced by the binaural chip are best displayed as movies, we will reproduce the results for the four signals in Figure 5.5 on the following pages.[4]

The images in Figures 5.6 to 5.9 are ordered row wise from left to right, i.e., the first image of the sequence is at the upper left corner, the second image is at the right of the first, and so on. The output images have been generated for the following parameter values (see Table 5.1, and page 142):[5]

`filters`	`fu` [Hz]	`fo` [Hz]	`q`	`fs` [Hz]	`r`	`threshold`
64	100	3750	1	8192	32	0.05

[4] We arbitrarily chose the second signal from every class of our data base.

[5] According to equation (5.3), K (=`filters`) could be 128 for $r = 32$. For the representation of the output images in Figures 5.6 to 5.9 we can afford the lower resolution.

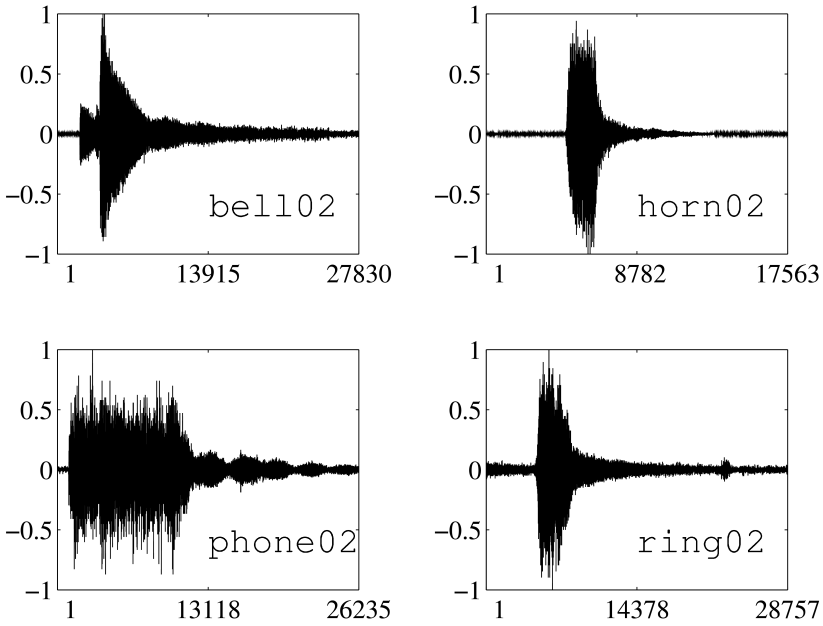


Figure 5.5: *Samples of the acoustical alarm signals (normalized amplitudes) The abscissas correspond to the discrete time samples for a sampling frequency of* $f_s = 8\,kHz$.

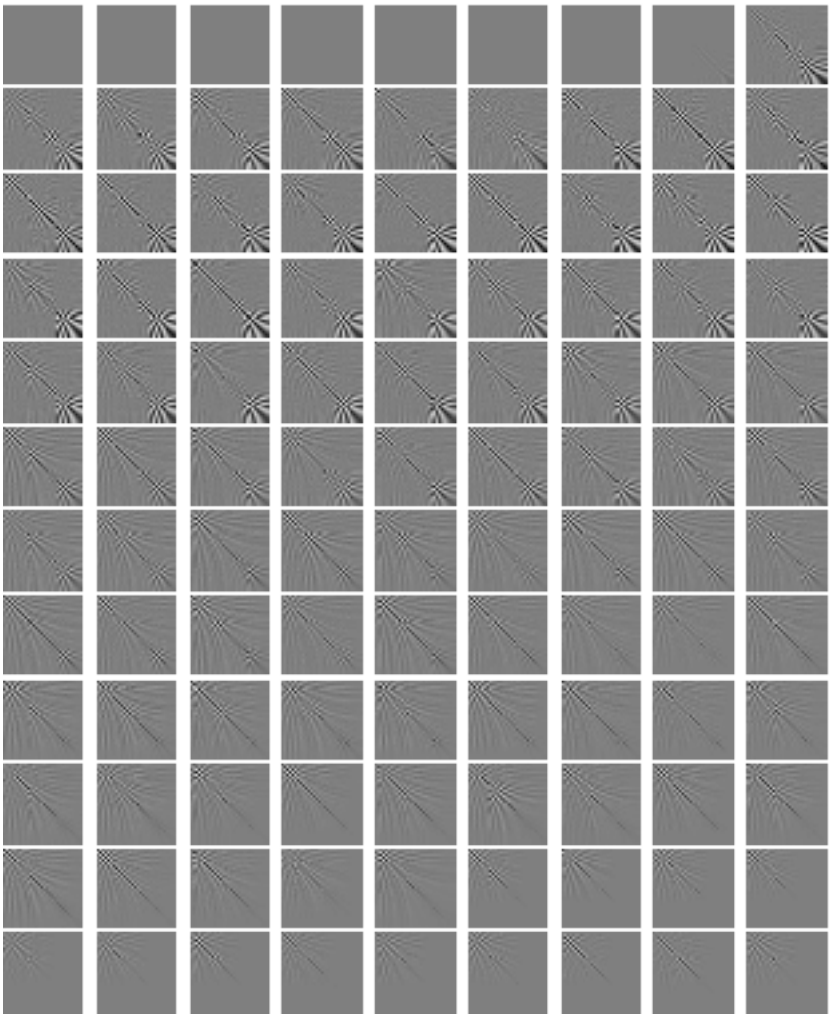


Figure 5.6: *Output images of our binaural-chip model for signal* `bell02` *(see Figure 5.5)*

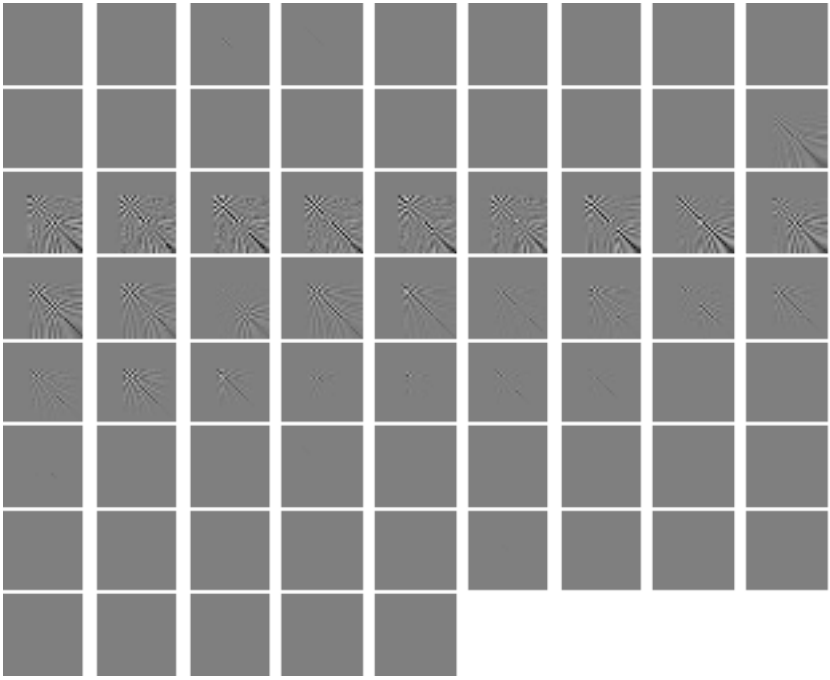


Figure 5.7: *Output images of our binaural-chip model for signal* `horn02` *(see Figure 5.5)*

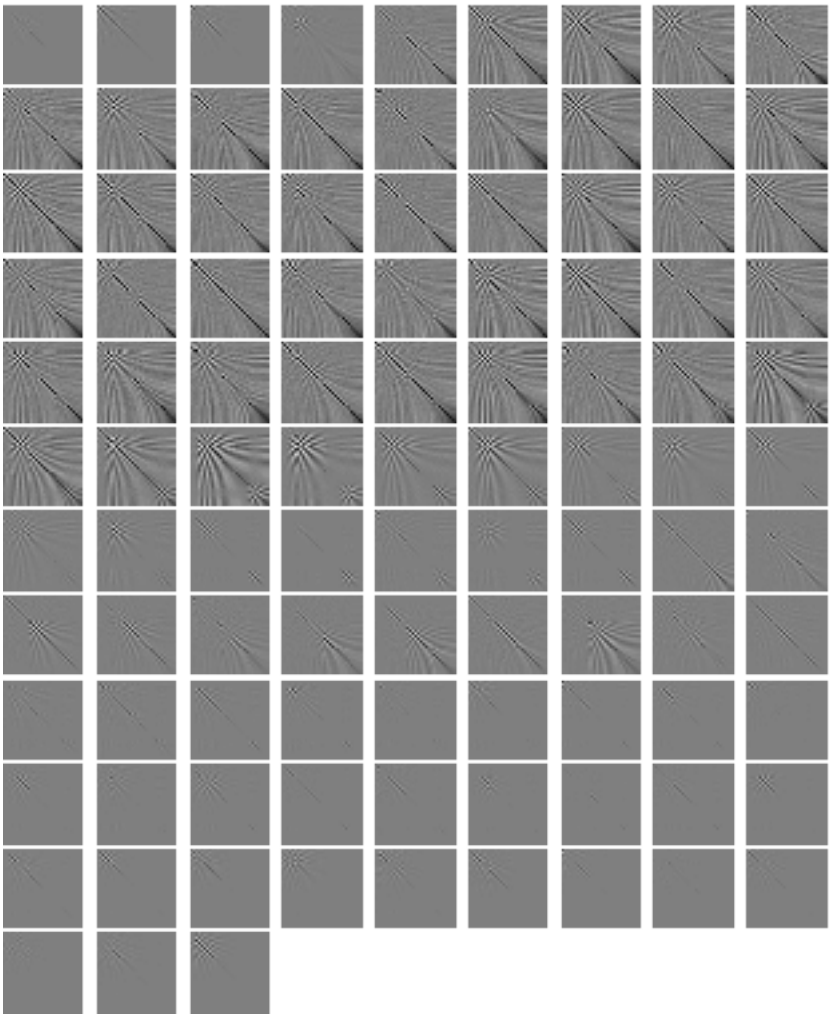


Figure 5.8: *Output images of our binaural-chip model for signal* `phone02` *(see Figure 5.5)*

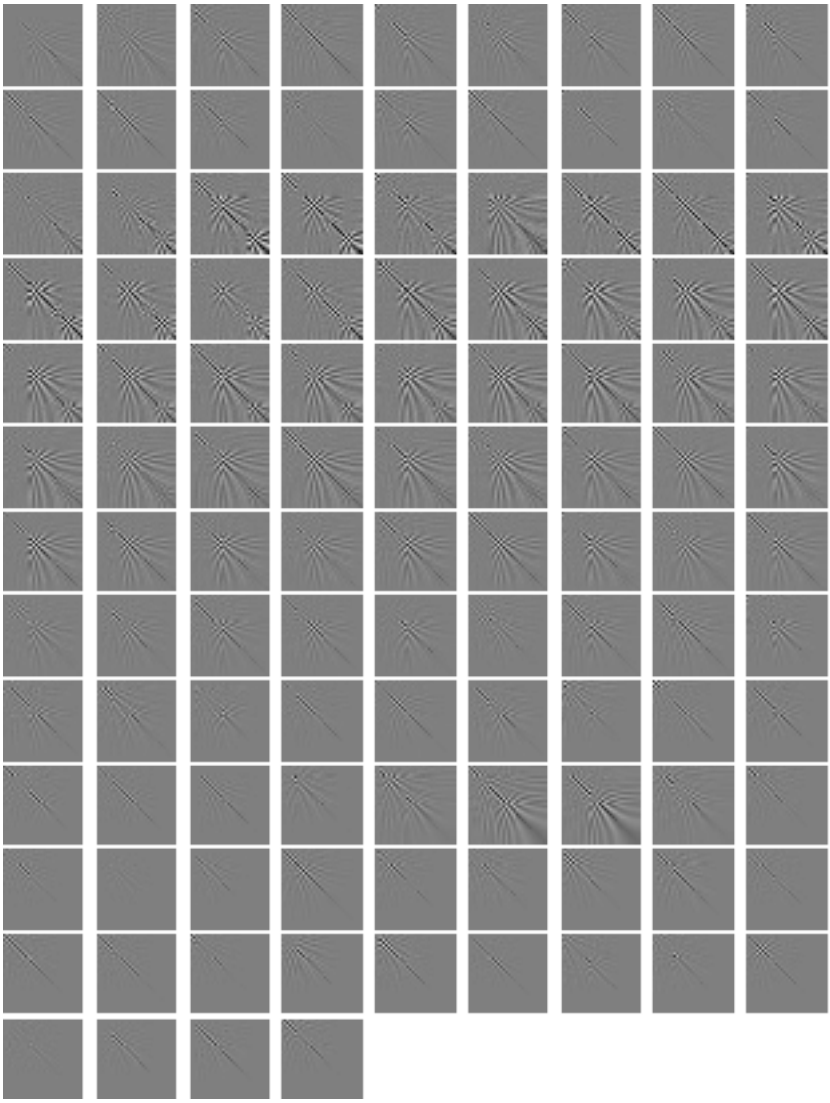


Figure 5.9: *Output images of our binaural-chip model for signal* `ring02` *(see Figure 5.5)*

5.2 Realizing Filter Biquads with CNNs

We are using CNNs for the classification of acoustical alarm signals. In a first preprocessing step we model the basilar membrane with a filter cascade of linear second-order low-pass filters (see the previous section and [35]). The basilar membrane is located in the inner ear (cochlea), and performs a frequency and phase analysis of the incoming sounds. The waves which propagate along two identical basilar membranes are correlated in a second step such that a 2-D representation of the acoustical input signals is generated. The entire signal preprocessing may be incorporated into the CNN itself, if the filter cascade, i.e., the low-pass filters, can be modelled with CNNs.

Efforts are presently under way to incorporate a CNN into a so-called *"universal machine"*. Thus, although it is essentially a nonlinear network, such a universal machine should also be capable of modelling linear systems. In this section we show how to realize arbitrary second-order linear filters (i.e., biquads) with a CNN structure. The time-continuous signal to the CNN may well be time-varying, and the CNN may well be operated as a dynamical system, thus it will not be viewed only as a mapping device as in image processing.

First of all, we will describe how a filter biquad — whose biquadratic transfer function is given in equation (5.9) — can be realized with a CNN-compatible structure. To understand the meaning of the building blocks within the biquad cell we will derive the solution for the special case of a second-order low-pass filter and treat the general case later.

5.2.1 Solution

The general standard transfer function of a second-order linear filter or biquad, is given in the next equation [36]:

$$T(s) = K\frac{s^2 + 2\sigma_z s + \omega_z^2}{s^2 + 2\sigma_p s + \omega_p^2} \tag{5.9}$$

The two-layer CNN cell with nonlinear templates[6] given in Figure 5.10 realizes a biquad as defined in (5.9). The corresponding transfer function is given in equation (5.10).

[6] As in [12], we say that a CNN has *nonlinear templates* whenever at least one of its voltage-controlled current sources cannot be described by $i = k \cdot u$, where i denotes the output and u the input to the VCCS, and k is a constant.

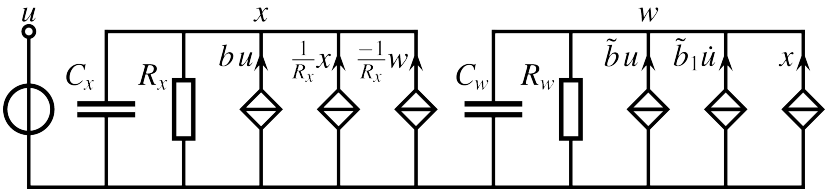


Figure 5.10: *Solution for a CNN realization of a biquad*

$$T(s) = \frac{W(s)}{U(s)} = \frac{\tilde{b}_1}{C_w}\,\frac{s^2 + \frac{\tilde{b}}{\tilde{b}_1}s + \frac{b}{\tilde{b}_1 C_x}}{s^2 + \frac{1}{T_w}s + \frac{1}{T_x C_w}} \tag{5.10}$$

where $T_x = R_x C_x,\ T_w = R_w C_w$

The templates for the CNN biquad are given in (5.11).[7]

$$\begin{array}{llllll} A_{11} = \frac{1}{R_x} & A_{12} = \frac{-1}{R_x} & B_{11} = b & I_1 = 0 & & \\ A_{22} = 0 & A_{21} = 1 & B_{21} = \tilde{b} & I_2 = 0 & \hat{B}_{21} = \tilde{b}_1 \frac{du(t)}{dt} & \end{array} \tag{5.11}$$

At this point we have to emphasize three properties of the solution:

- The biquad is realized by a 1×1 CNN where the effects of surrounding cells are not needed.[8] Thus, a biquad is realized by a single cell (of a two-layer CNN, see below). Of course, we can take $M \times N$ such cells and provide many different biquads in a matrix structure with a two-layer CNN. Adjacent filters are connected to each other through the local connections within the cell neighbourhoods. Thus, higher-order filters may be realized with such a cascade of biquads.

- A biquad is a second-order network. The cell as introduced in Section 1.3 (see Figure 1.2) is a first-order network. Using a multilayer CNN [1] with two layers helps us to increase the order. Of course, we can interpret the addition of the second layer as a new definition of the CNN cell, now of second order.

- Finally, the biquad is a linear system. Replacing the piecewise-linear function given in equation (1.2) with an identity output function, $y_{ij}(t) = x_{ij}(t)$, where i and j are now restricted to 1, leads to the desired linear system.

[7] Here, again, we deal with normalized values.

[8] This is equivalent to a neighbourhood radius $r = 0$.

The current $\tilde{b}_1 \dot{u}$ (see Figure 5.10) prevents a zero of the transfer function from existing at infinity, i.e., it ensures that an s^2 term exists in the numerator of the transfer function. Because of this current, we have to speak of a CNN with nonlinear templates [12]: the output current of the corresponding VCCS does not depend linearly on the driving voltage. This is, again, a matter of interpretation. Instead of the nonlinear template $\hat{B}_{21}$ in (5.11) we can define a linear template $B_{22} = \tilde{b}_1$ and set the derivative of u as input to the second layer.

The B templates control the filter function of the biquad (low-pass, band-pass, high-pass filter, etc.). The A templates[9] together with the time constants T_x and T_w determine the poles of the filter. Because T_x and T_w determine the coefficients of the denominator in (5.10) independently of each other, the poles of the transfer function can lie anywhere in the left-half s plain.[10] The poles will not be on the $j\omega$-axis because $T_w = R_w C_w$ is supposed to be finite.

The current through R_x and the current $\frac{1}{R_x} x$ annihilate each other. Nevertheless, we do not remove the corresponding components, because, first, we want to remain CNN compatible, i.e., R_x in a CNN is supposed to have a finite value, and, second, in the chip realization leakage currents cannot be avoided and may be modelled with R_x.

5.2.2 The low-pass filter

Let us now consider the low-pass filter as a special case of a biquad. This has the advantage that we do not have to worry about the zeros of the transfer function, and can concentrate only on the characteristic polynomial of the system, i.e., on the poles. The transfer function of the low-pass filter is given in equation (5.12a), and equation (5.12b) determines the corresponding differential equation.

$$T(s) = K \frac{\omega_p^2}{s^2 + \frac{\omega_p}{q} s + \omega_p^2} \qquad \text{where } q = \frac{\omega_p}{2\sigma_p} \tag{5.12a}$$

$$\ddot{y} + \frac{\omega_p}{q} \dot{y} + \omega_p^2 y = K \omega_p^2 u \tag{5.12b}$$

[9] Note that we replaced the piecewise-linear function with $y(t) = x(t)$ as mentioned in the third remark on page 71.

[10] This means also that both layers may have different time constants.

We now wish to find a CNN which behaves like the given low-pass filter, i.e., one whose differential equation equals that given in (5.12b). We start with the general two-layer CNN shown in Figure 5.11 which is described by the linear differential equation given in (5.13).

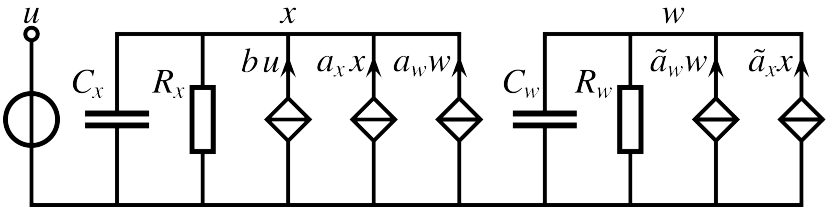


Figure 5.11: *A* 1×1 *CNN with two layers*

$$\begin{aligned} & & & \ddot{w} + \\ & & \left[\frac{1}{T_w} + \frac{1}{T_x} - \frac{a_x}{C_x} - \frac{\tilde{a}_w}{C_w}\right] & \dot{w} + \\ \left[-\frac{\tilde{a}_w}{T_x C_w} + \frac{1}{T_x T_w} - \frac{a_x}{T_w C_x} + \frac{a_x \tilde{a}_w}{C_x C_w} - \frac{\tilde{a}_x a_w}{C_x C_w}\right] & w & = & \frac{\tilde{a}_x b}{C_x C_w} u \end{aligned} \quad (5.13)$$

Comparing both linear differential equations (5.12b) and (5.13) we obtain the conditions (5.14) which must be fulfilled to guarantee equivalence of the low-pass filter and its CNN realization.[11]

$$\left[\frac{1}{T_w} + \frac{1}{T_x} - \frac{a_x}{C_x} - \frac{\tilde{a}_w}{C_w}\right] \stackrel{!}{=} \frac{\omega_p}{q} \quad (5.14a)$$

$$\left[-\frac{\tilde{a}_w}{T_x C_w} + \frac{1}{T_x T_w} - \frac{a_x}{T_w C_x} + \frac{a_x \tilde{a}_w}{C_x C_w} - \frac{\tilde{a}_x a_w}{C_x C_w}\right] \stackrel{!}{=} \omega_p^2 \quad (5.14b)$$

$$\frac{\tilde{a}_x b}{C_x C_w} \stackrel{!}{=} K \omega_p^2 \quad (5.14c)$$

Note that we have only three conditions to determine nine unknown values, namely C_x, C_w, b, a_x, a_w, $\tilde{a}_w$, $\tilde{a}_x$, T_x and T_w. With the relations given together with (5.10), we are able to express T_x and T_w as functions of C_x and C_w, respectively. R_x and R_w then appear as parameters. Thus, the number of unknown

[11] Of course, y has been replaced by w.

variables reduces to seven (C_x, C_w, b, a_x, a_w, $\tilde{a}_w$ and $\tilde{a}_x$), but we still have an under-determined set of equations.

A first trial

One way of eliminating four unknown variables is to introduce only C_x, C_w, and b as shown in Figure 5.12. The linear differential equation which describes the dynamics of this simplified network is given in (5.15).

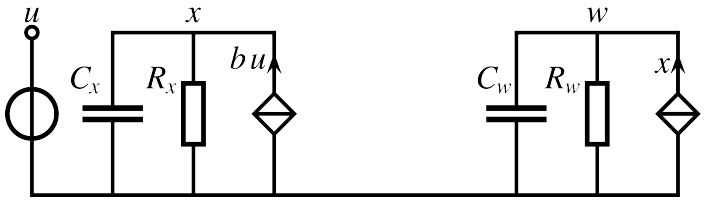


Figure 5.12: *A first trial for the solution of the low-pass filter*

$$\ddot{w} + \left[\frac{1}{T_x} + \frac{1}{T_w}\right] \dot{w} + \frac{1}{T_x T_w} w = \frac{1}{C_x C_w} b u \tag{5.15}$$

Unfortunately, this approach provides real-valued coefficients if and only if $q \leq \frac{1}{2}$. Thus, the poles can be only on the negative real axis in the s plain. Equations (5.16) show the values of the coefficients for $q = \frac{1}{2}$.[12]

$$C_x = \frac{1}{R_x \omega_p} \tag{5.16a}$$

$$C_w = \frac{1}{R_w \omega_p} \tag{5.16b}$$

$$b = \frac{K}{R_x R_w} \tag{5.16c}$$

[12] Again, R_x and R_w are user-defined parameters.

A first solution

In order to obtain conjugate-complex poles in the s plain the coefficients of both terms $\dot{w}$ and w in (5.15) must be independent of each other. Thus, we need a linear differential equation with only three unknowns, whose coefficients do not impose any constraints on the location of the poles. Equation (5.17) gives us such a relation.

$$\ddot{w} + \frac{1}{T_w}\dot{w} + \frac{1}{C_w T_x} w = \frac{1}{C_x C_w} b\, u \tag{5.17}$$

The solution for the unknown terms C_x, C_w and b is given in equations (5.18).[13]

$$C_x = \frac{R_w}{R_x}\frac{1}{q\,\omega_p} \tag{5.18a}$$

$$C_w = \frac{q}{R_w \omega_p} \tag{5.18b}$$

$$b = \frac{K}{R_x} \tag{5.18c}$$

Although we have now a solution for the low-pass filter, we still do not know what the corresponding CNN looks like. Let us go back to the general approach given in Figure 5.11. We have to force this network, described by equation (5.13), to behave the same as one whose differential equation is (5.17). Thus, we have to set (5.13) equal to (5.17). This gives us the four additional conditions (5.19) which, together with the three conditions in (5.14), determine all seven unknown values C_x, C_w, b, a_x, a_w, $\tilde{a}_w$ and $\tilde{a}_x$.

$$\frac{1}{T_x} - \frac{a_x}{C_x} - \frac{\tilde{a}_w}{C_w} \stackrel{!}{=} 0 \tag{5.19a}$$

$$\tilde{a}_w \stackrel{!}{=} -1 \tag{5.19b}$$

$$\frac{1}{T_x T_w} - \frac{a_x}{T_w C_x} + \frac{a_x \tilde{a}_w}{C_x C_w} - \frac{\tilde{a}_x a_w}{C_x C_w} \stackrel{!}{=} 0 \tag{5.19c}$$

$$\tilde{a}_x \stackrel{!}{=} 1 \tag{5.19d}$$

[13] To compute the solution compare the coefficients in (5.17) with those in (5.12b) and use the relations given with (5.10).

The templates and the capacitances for the CNN realization of a low-pass filter with no constraints on the location of the poles are given in (5.20).

$$\begin{aligned} A_{11} &= a_x = \tfrac{{R_w}^2+q^2}{R_x q^2} & A_{12} &= a_w = -\tfrac{{R_w}^2+R_w+q^2}{R_x q^2} \\ A_{22} &= \tilde{a}_w = -1 & A_{21} &= \tilde{a}_x = 1 \\ B_{11} &= b = \tfrac{K}{R_x} \\ C_x &= \tfrac{R_w}{R_x}\,\tfrac{1}{q\omega_p} & C_w &= \tfrac{1}{R_w}\,\tfrac{q}{\omega_p} \end{aligned} \tag{5.20}$$

Naturally, the values for C_x, C_w and b are equal to those already given in equations (5.18).

A simpler solution

As the title of this section suggests, there is a simpler solution for the CNN realization of the low-pass filter. What can be improved in the previous approach? Among other things we can ask for conditions simpler than those given in (5.19) to achieve equality between (5.13) and (5.17). In other words, we might do better defining other additional conditions than those given by (5.19). We will see in the next section why we replace (5.19b) and (5.19c) by (5.21b) and (5.21c), respectively.

$$\frac{1}{T_x} - \frac{a_x}{C_x} - \frac{\tilde{a}_w}{C_w} \stackrel{!}{=} 0 \tag{5.21a}$$

$$a_w \stackrel{!}{=} -\frac{C_x}{T_x} \tag{5.21b}$$

$$\frac{1}{T_x} \stackrel{!}{=} \frac{a_x}{C_x} \tag{5.21c}$$

$$\tilde{a}_x \stackrel{!}{=} 1 \tag{5.21d}$$

From (5.21a) and (5.21c) we can immediately say that $\tilde{a}_w = 0$. (5.21b) and (5.21c) lead to $a_w = \frac{-1}{R_x}$ and $a_x = \frac{1}{R_x}$, respectively, using the relation $T_x = R_x C_x$. The simpler[14] solution shown in Figure 5.13 for the CNN realization of a low-pass filter is given in (5.22).

[14] Simpler than (5.20) in terms of fewer templates and no dependencies of their values on q.

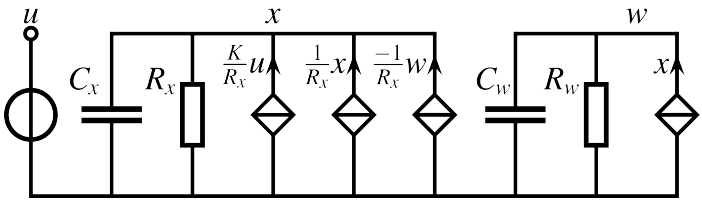


Figure 5.13: *CNN realization of a low-pass filter*

$$A_{11} = \tfrac{1}{R_x} \quad A_{12} = \tfrac{-1}{R_x} \quad B_{11} = \tfrac{K}{R_x} \quad C_x = \tfrac{R_w}{R_x}\tfrac{1}{q\omega_p} \quad A_{21} = 1 \quad C_w = \tfrac{1}{R_w}\tfrac{q}{\omega_p} \tag{5.22}$$

5.2.3 The band-pass filter

The transfer function of a band-pass filter is given in equation (5.23a), and equation (5.23b) determines the corresponding differential equation.

$$T(s) = K\frac{\omega_p s}{s^2 + \frac{\omega_p}{q}s + \omega_p^2} \qquad \text{where } q = \frac{\omega_p}{2\sigma_p} \tag{5.23a}$$

$$\ddot{y} + \frac{\omega_p}{q}\dot{y} + \omega_p^2 y = K\omega_p \dot{u} \tag{5.23b}$$

Again, we start with a general approach: the one shown in Figure 5.14. Its linear differential equation is given in (5.24).

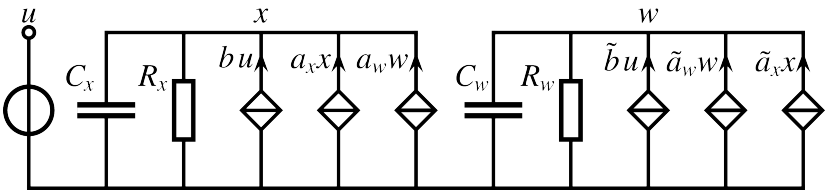


Figure 5.14: *General approach for the CNN realization of a band-pass filter*

$$
\begin{aligned}
&\ddot{w} + \\
&\left[\frac{1}{T_w} + \frac{1}{T_x} - \frac{a_x}{C_x} - \frac{\tilde{a}_w}{C_w}\right]\dot{w} + \\
\left[-\frac{\tilde{a}_w}{T_x C_w} + \frac{1}{T_x T_w} - \frac{a_x}{T_w C_x} + \frac{a_x \tilde{a}_w}{C_x C_w} - \frac{\tilde{a}_x a_w}{C_x C_w}\right] w \quad &= \\
\left[\frac{\tilde{b}}{C_w}\left(\frac{1}{T_x} - \frac{a_x}{C_x}\right) + \frac{\tilde{a}_x b}{C_x C_w}\right] u \quad + \quad &\frac{\tilde{b}}{C_w}\dot{u} \qquad (5.24)
\end{aligned}
$$

Compared to the network of Figure 5.11, the only component we have added is the VCCS labeled $\tilde{b}\,u$. From the left-hand side of (5.24) we see that, as expected, the determination of the poles is the same as for the low-pass filter. However, in order to force both right-hand sides of equations (5.23b) and (5.24) to be equal, condition (5.21c) is required.[15] Thus, we obtain the corresponding values for the poles from (5.22), and b and $\tilde{b}$ have to fulfill the following conditions:

$$b \stackrel{!}{=} 0 \qquad (5.25a)$$

$$\frac{\tilde{b}}{C_w} \stackrel{!}{=} K\omega_p \qquad (5.25b)$$

The templates of the solution for the CNN realization of a band-pass filter are given in (5.26).

$$
\begin{array}{lll}
A_{11} = \frac{1}{R_x} & A_{12} = \frac{-1}{R_x} & C_x = \frac{R_w}{R_x}\frac{1}{q\omega_p} \\
 & A_{21} = 1 \quad B_{21} = \frac{Kq}{R_w} & C_w = \frac{1}{R_w}\frac{q}{\omega_p}
\end{array} \qquad (5.26)
$$

5.2.4 The second-order term in the numerator of the transfer function

In the CNN structure all capacitances have a common potential point, namely the reference point or ground. With this structure, it is not possible to force both zeros of the biquad to be finite. Thus, the term s^2 in the numerator of the transfer function given in (5.9) cannot be realized, unless, as already

[15] This is the reason why we replaced (5.19c) by (5.21c) in the previous section. Relation (5.21b) is then a direct consequence of this substitution and the fact that $\tilde{a}_w = 0$.

indicated on page 72, we introduce nonlinear templates.[16] This is how, e.g., high-pass filters or frequency rejection networks (FRNs) [36] can be realized with CNN-compatible structures.

The values which determine the pole locations have been computed in the previous sections. The results are displayed in (5.27) as functions of σ_p and ω_p with parameters R_x and R_w. For convenience, we repeat the transfer function of a biquad (see (5.9)) in (5.28).

$$\begin{aligned} A_{11} = \tfrac{1}{R_x} \quad A_{12} = \tfrac{-1}{R_x} \quad C_x = \tfrac{R_w}{R_x}\,\tfrac{2\sigma_p}{\omega_p^2} \\ A_{21} = 1 \qquad\quad C_w = \tfrac{1}{R_w}\,\tfrac{1}{2\sigma_p} \end{aligned} \tag{5.27}$$

$$T(s) = K\frac{s^2 + 2\sigma_z s + \omega_z^2}{s^2 + 2\sigma_p s + \omega_p^2} \tag{5.28}$$

The only thing we now have to worry about is the second-order term of the numerator. It can be realized by introducing the VCCS labeled $\tilde{b}_1\dot{u}$ in Figure 5.10. For convenience, the network is again depicted in Figure 5.15. $\tilde{b}_1\dot{u}$ corresponds

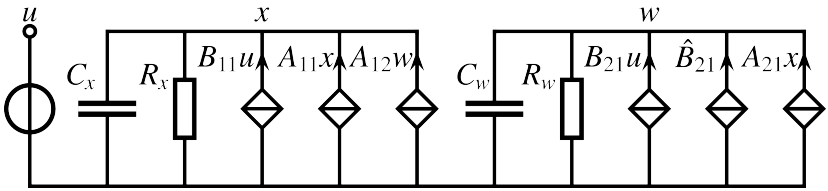


Figure 5.15: *Solution for a CNN realization of a biquad*

to $\hat{B}_{21}$ as we already know from (5.11). The linear differential equation which describes the dynamics of this CNN-compatible biquad is given in (5.29a). The corresponding transfer function is (5.29b), i.e., the solution we presented in Section 5.2.1.

$$\ddot{w} + \frac{1}{T_w}\dot{w} + \frac{1}{C_w T_x}w = \frac{b}{C_x C_w}u + \frac{\tilde{b}}{C_w}\dot{u} + \frac{\tilde{b}_1}{C_w}\ddot{u} \tag{5.29a}$$

$$T(s) = \frac{\tilde{b}_1}{C_w}\,\frac{s^2 + \frac{\tilde{b}}{\tilde{b}_1}s + \frac{b}{\tilde{b}_1 C_x}}{s^2 + \frac{1}{T_w}s + \frac{1}{T_x C_w}} \tag{5.29b}$$

[16] Nonlinear templates, of course, form part of the CNN structure.

The values for the control templates (see (5.30)) result immediately from a coefficient comparison between the numerator in (5.29b) (or (5.10)) and the numerator in (5.28) (or (5.9)).

$$\begin{aligned} B_{11} &= \frac{K}{R_x}\,\frac{\omega_z^2}{\omega_p^2} \\ B_{21} &= \frac{K}{R_w}\,\frac{\sigma_z}{\sigma_p} \quad \hat{B}_{21} = \frac{K}{R_w \cdot 2\sigma_p}\,\dot{u} \end{aligned} \tag{5.30}$$

Thus, equations (5.27) and (5.30) determine the location of the poles and zeros of the biquad, respectively. E.g., in order to realize a low-pass filter (see (5.12)) set $\omega_z = \omega_p$, $\sigma_z = 0$, and $\hat{B}_{21} = 0$, or, in the case of a high-pass filter with $T(s) = K\frac{s^2}{s^2+2\sigma_p s+\omega_p^2}$, set $\omega_z = \sigma_z = 0$.

5.3 Summary

In the first section of this chapter we have presented a 1-D to 2-D transformation which preserves phase information of the input signal. We have described the two parts of the so-called binaural chip, namely the filter-cascade model of the basilar membrane, and the sign correlation of the outputs of two identical filter cascades fed with the same input signal. For every acoustical input signal, a sequence of grey-scale images is generated. The images retain phase information in the acoustical signal. The generated sequence of images serves as input to the CNN.

In order to generate the image sequence with a CNN-like structure, therefore allowing the integration of the preprocessing device into a CNN universal machine, the filters in the cascades need to be replaced by CNNs with the appropriate templates. In Section 5.2 we have presented a CNN realization for filter biquads. We started by modelling low-pass filters because their zeros are at infinity and we could concentrate on the realization of the poles. As has been shown in Section 5.2.2, there are various possibilities to model the poles of a low-pass filter. Nevertheless, the values in (5.20) and (5.22) for C_x and C_w are the same.

The capacitance spread is proportional to q^2, and turns out to be the limiting factor for CNN-compatible structures: it would appear that CNNs cannot compete with capacitance-spread optimized filter realizations [37]. On the

other hand, if one needs a low-pass filter to pre-process input data to a CNN, the operation can be done with the same device,[17] or at least with a dedicated block based on the same integration technology as the rest of the CNN chip. The CNN realization offers advantages especially when linear filters have to be arranged in a matrix with local connections between the filters, e.g., in a filter cascade.

In the following sections the CNN solution for the low-pass filter was extended to higher-order terms in the numerator of the transfer function to realize band-pass and high-pass filters. Additional constraints had to be taken into account to achieve the required equality between the numerator of the band-pass filter and the numerator of its CNN realization.

In order to model the second-order term in the numerator of the biquadratic transfer function, a nonlinear template had to be introduced. It can be avoided if the time derivative of the input voltage is used as input to the second layer. Then the solution is fully compatible to the CNN structure presented in [1, 2]: a biquad can be realized by a 1×1 CNN of two layers without any nonlinear sources. This is a somewhat "degenerated" and simple CNN. Nevertheless, we have shown that the CNN universal machine still works for trivial operations like those performed by linear filters, and that its transient behaviour might be of interest as well.

[17] Provided one can tune the template values and the *RC* constants according to the required filter parameters.

Chapter 6

Processing Methods with CNNs

CNNs can be used to solve many different image-processing problems. In this chapter we present three methods of combining several image transformations to generate images characteristic for a given set of acoustical alarm signals. Each alarm signal is described by one image which ideally should have the following properties to achieve good classification results: it has to be similar to the characteristic images of signals from the same class, and it should be distinct from the characteristic images of signals from other classes. Thus, a simple classification device shall separate the acoustical alarm signals into distinct classes by using the characteristic images produced with CNNs. All three processing methods use as their input the image sequence produced by the binaural chip, and incorporate a time–to–space mapping in their last step to collect information spread over the duration of the acoustical signal into the required characteristic image.

6.1 Extracting Information from Image Sequences

The role of the CNN in our alarm-signal classification system, see Figure 4.1 on page 54, is to process each of the images from the binaural chip, and then to somehow characterize the entire image sequence, and hence the alarm signal itself. We avoid the following problem among others, when performing a time–to–space mapping on the image sequence, i.e., collecting information spread over the duration of the signal: if we were to continuously indicate which class the signal seems to belong to, we might give wrong decisions at the begining of the alarm signal, because the remaining parts of the signal may still contribute to the characterization. *Thus, the classification should preferably be done for the entire signal, and it is not useful to continuously indicate which class might be active.* The decision is taken after the signal has almost died out, when no more information about the signal is expected.

The last sentence addresses the question of where the starting and ending points of the signal are, i.e., the so-called segmentation problem. For our simulations, we took the entire duration into account,[1] i.e., we did not introduce any thresholds which determine whether a signal is present, as proposed in [35]. Two remarks have to be made at this point: first, in a real-life application there must be a segmentation of the signals coming, e.g. from a microphone, into silent periods, where only noise is present, and active time intervals, where a sound is detected.[2] This kind of segmentation has already implicitly be done in our case during the process of recording the signals on analogue tape, and also at the sampling procedure which led to our data base. Nevertheless, and this addresses the second remark, the signals of our data base are *non-stationary* signals since they contain silent periods, the start and the end of the acoustical alarm signals, other transitional segments, and the stationary parts. Thus, although the signals have already been roughly segmented, there is still enough non-stationarity in the signals that modelling them, e.g., via prediction, may result in an unsolvable problem [38].

For the two parts in which the CNN is used for the classification of acoustical alarm signals, namely the processing of the incoming images and the time–to–space mapping, we made the following assumptions:

[1] The entire signal means all samples stored in the corresponding files (see Figure 5.5).

[2] In real-life applications not only alarm signals may be heard but also other acoustical signals. We do not consider this additional difficulty in our classification task.

- The CNN is supposed to be implemented on a chip [5]. The convergence time of the network to reach the solution for an image-processing task can then be assumed to be in the range of μ-seconds. Thus, there is sufficient time to process the images which come out of the binaural chip at a rate in the range of milliseconds. There should even be enough time to process an image several times, performing different tasks one after the other, and combining intermediate results, to achieve more sophisticated image processing.[3]

- Because the CNN is supposed to be implemented in a chip, the processing steps must not rely on high-precision computations, nor on very sensitive template values. The precision of integrated analogue circuits hardly reaches 8 bits, as already mentioned on page 62.

- No multilayer CNNs nor delay-type templates[4] should be used since this would make the chip implementation nearly impossible for larger CNNs (see the next point). Nevertheless, we allow nonlinear templates.

- The size of the CNN, i.e., M and N (see page 5), corresponds to the size of the correlation matrix of the binaural chip (see Figure 5.4). In other words, the CNN size has to match the image resolution of the binaural chip whose precision as an integrated analogue circuit is also reduced to at most 8 bits. Thus, the limiting factor for the image resolution is the number of filters we can realize in the cascade which models the basilar membrane for the audible frequency bandwidth.

- The task of the classification device (see Figure 4.1) should be kept as simple as possible. This condition may be satisfied if the alarm-signal characteristics extracted with the CNN from the sequence of binaural-chip images are good descriptions of the individual classes.

In dealing with CNNs, one of the major problems is the synthesis of templates for special tasks. An exact design procedure has been presented in Chapter 2, but it only serves for binary input pictures, i.e., for black and white input images. Learning methods, in the sense of the well-known back-propagation algorithm for multilayer perceptrons [8, 39, 40], are still lacking for continuous-time CNNs. Nevertheless, the templates for many standard

[3] This is the basic idea behind the CNN universal machine.

[4] Delay-type templates [12] are impracticable for large delays and continuous-time CNNs because of the realization of the required delay-lines.

image-processing tasks have been catalogued [10, 11, 41]. New templates are often synthesised by experience and intuition.

Since we had to rely on the above qualities for template generation, we first have to define how to obtain a characteristic image, *one per signal*, from a sequence of images, like those in Figures 5.6 to 5.9. It is necessary to condense the characteristics into a single image, because the final output of a CNN is a single image, and also because this is nothing else than performing a time–to–space mapping. Processing acoustical signals with CNNs has been studied in [42] where simulations were used to obtain characteristic images. Several useful processing methods have been developed. All of them have the following common properties:

- The CNN marks the "active regions" within the images of the binaural-chip output sequence. These regions are patterns like those which can be observed in Figures 5.6 to 5.9.
- For historical reasons, the negative images of the binaural-chip output images are regarded. This does not impose any substantial changes since this corresponds to a notion definition: correlation and anticorrelation can be remapped to the values -1 and 1, respectively (see page 61). Figure 6.1 shows the inverted images of the fourth row of Figure 5.6.
- All processing methods are followed by the same time–to–space mapping: it is an OR-combination of the processed images. This simple mapping may have to be improved if more complex signals like, e.g., speech signals, need to be processed. However, for our well-defined task, the summation of all black pixels produced from the individual images of the binaural-chip output sequence yields the required, characteristic images with the desired properties: images from signals of the same class are similar, and images from signals of different classes are distinguishable.[5]

Figure 6.1: *Negative images of the fourth row of Figure 5.6*

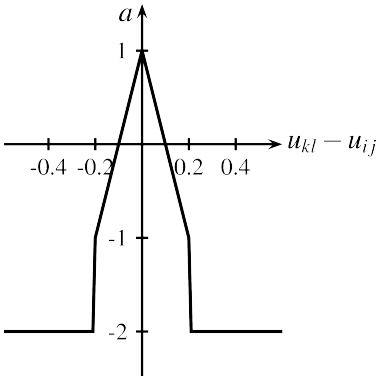
[5]These properties have to be valid from the point of view of the *classification device, not necessarily for an observer*.

In the following sections we will present three processing methods and the basic ideas behind them, and the corresponding characteristic images of the signals in our data base for the binaural-chip parameter values listed in Section 5.1.3.

6.1.1 Processing Algorithm 1

The first processing method (hereafter called Processing Algoritm 1), is the one with the highest classification error rates (see Chapter 8). However, the rates are already lower than the ones achieved in [30] and [31] with spectral techniques and (conventional) neural networks, respectively. Processing Algorithm 1 first uses the *extreme* template [10] with $I = -1.9$, which is a nonlinear template with neighbourhood radius $r = 1$ shown in Table 6.1. Essentially,

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} a & a & a \\ a & a & a \\ a & a & a \end{bmatrix} \qquad I = -1.9$$



$$\text{PWL-function } a = \begin{cases} -2 & , & & u_{kl}-u_{ij} & \leq & -0.21 \\ 100(u_{kl}-u_{ij}) + 19 & , & -0.21 < & u_{kl}-u_{ij} & \leq & -0.2 \\ 10(u_{kl}-u_{ij}) + 1 & , & -0.2 < & u_{kl}-u_{ij} & \leq & 0 \\ -10(u_{kl}-u_{ij}) + 1 & , & 0 < & u_{kl}-u_{ij} & \leq & 0.2 \\ -100(u_{kl}-u_{ij}) + 19 & , & 0.2 < & u_{kl}-u_{ij} & \leq & 0.21 \\ -2 & , & 0.21 < & u_{kl}-u_{ij} & & \end{cases}$$

Table 6.1: Extreme template: *a is a piecewise-linear* (PWL) *function of the argument* $(u_{kl} - u_{ij})$ *[12]. The initial state of the CNN is set equal to the input image.*

it compares the sum of the differences between the input pixels within the

neighbourhood with the threshold value I. The differences (input value of a cell in the neighbourhood minus the input value of the center cell) are first passed through a piecewise-linear function before they are summed up. Only small differences will contribute a positive value to the sum, i.e., the output of the (center) cell becomes black only if the cells around it have similar input values. Thus, regions of the input image which have similar values will become black after processing the image with the extreme template.

The results after the application of the extreme template to the negative[6] of the images in Figure 5.6 are shown in Figure 6.2. The extreme template produces noisy images, i.e., there are white and black isolated pixels in the images which may disturb further processing steps. To get rid of these artifacts, we apply the noise-removal template given in Table 6.2 to the output of the extreme template. Figure 6.3 shows the filtered images of Figure 6.2.[7]

A		
0	0	0
0	1.1	0
0	0	0

B		
0.2	0.2	0.2
0.2	0	0.2
0.2	0.2	0.2

$I = -0.7$

Table 6.2: *Noise-removal template*
The initial state of the CNN is set equal to the input image.

Before performing the time–to–space mapping, the number of black pixels is reduced with the edge-extraction template given in Table 6.3 (see Table 1.3 on page 13 with $q = 1$). The resulting images, after the application of the

A		
0	0	0
0	1.25	0
0	0	0

B		
0	−1	0
−1	4	−1
0	−1	0

$I = -0.5$

Table 6.3: *Edge-extraction template*

edge-extraction template to the images of Figure 6.3, are shown in Figure 6.4.

[6] See above. The extreme template is not sensitive to the inversion of the input image.
[7] For lack of space, we cannot show the results for more alarm signals.

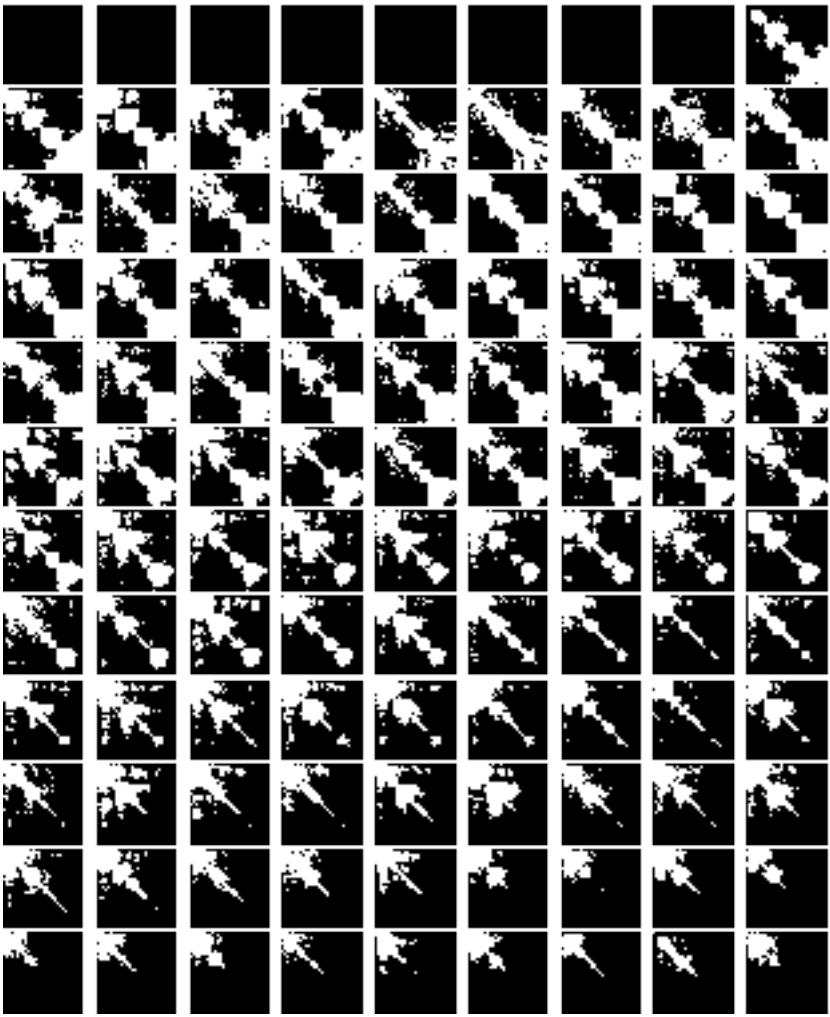


Figure 6.2: *Results after processing the negatives of the images in Figure 5.6 with the extreme template given in Table 6.1*

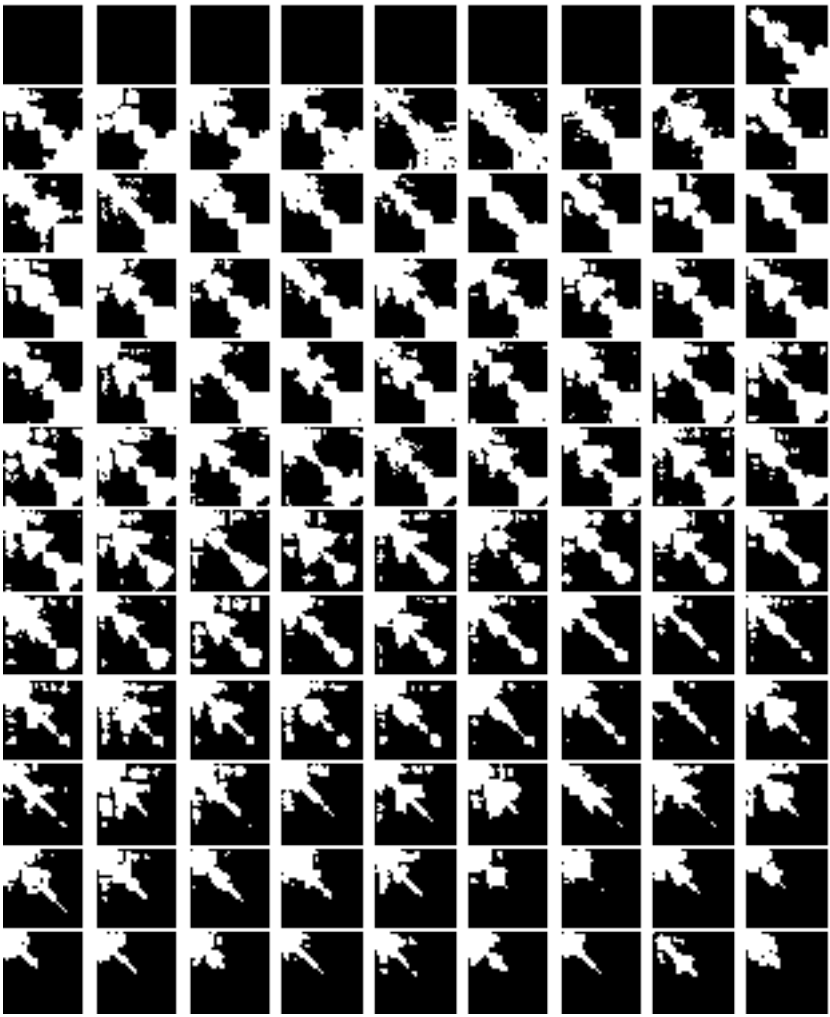


Figure 6.3: *Results after processing the images in Figure 6.2 with the noise-removal template given in Table 6.2*

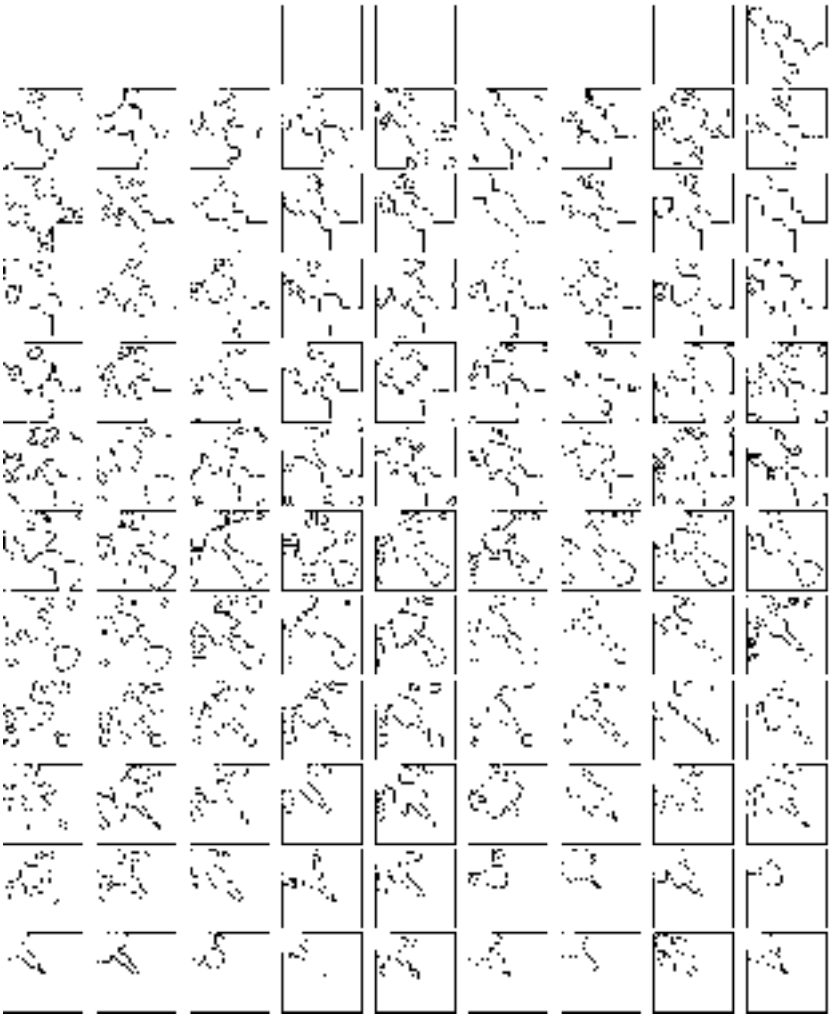


Figure 6.4: *Results after processing the images in Figure 6.3 with the edge-extraction template given in Table 6.3*

Finally, the time–to–space mapping is performed. For example, for the signal `bell02` we collect all the black pixels of the images in Figure 6.4 into one image.[8] Figures 6.5 to 6.8 show the results of Processing Algorithm 1 for all signals in our data base, i.e., these are the characteristic images generated with Processing Algorithm 1 for the acoustical alarm signals under consideration.[9]

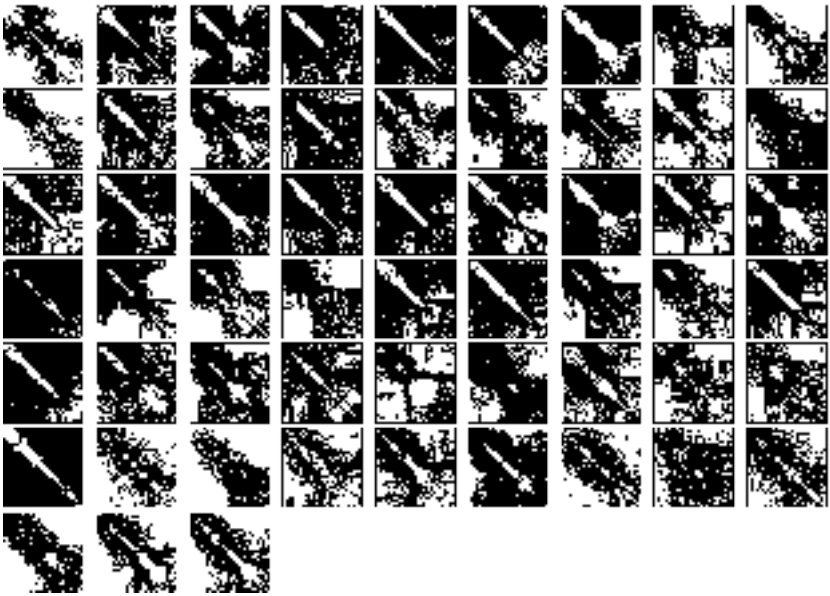


Figure 6.5: *Characteristic images of Processing Algorithm 1 for signals* `bell01` *to* `bell57` *(row wise from left to right)*

[8] It can be seen in Figure 6.5 on the top row, the second image from the left.

[9] We are giving the examples for the binaural-chip parameter values listed in Section 5.1.3.

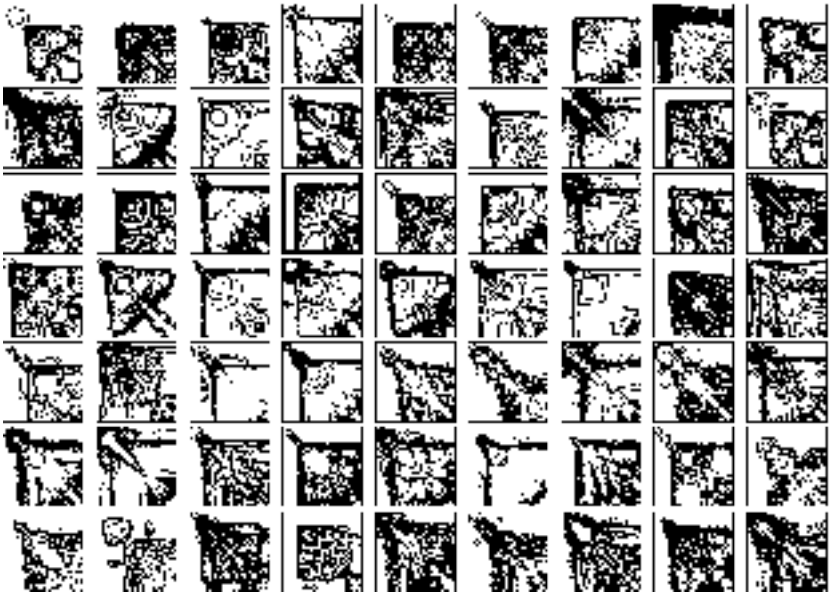


Figure 6.6: *Characteristic images of Processing Algorithm 1 for signals* `horn01` *to* `horn63` *(row wise from left to right)*

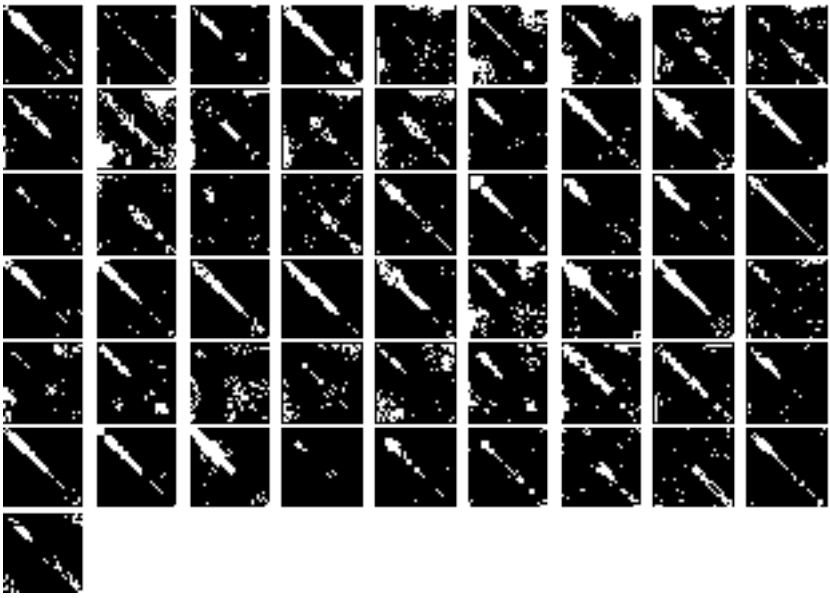


Figure 6.7: *Characteristic images of Processing Algorithm 1 for signals* `phone01` *to* `phone55` *(row wise from left to right)*

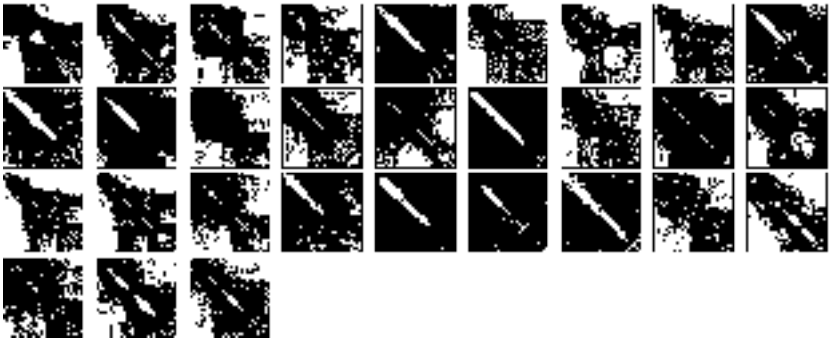
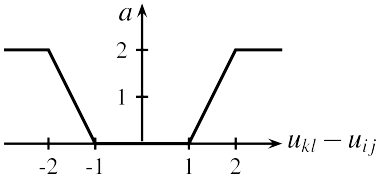


Figure 6.8: *Characteristic images of Processing Algorithm 1 for signals* `ring01` *to* `ring30` *(row wise from left to right)*

6.1.2 Processing Algorithm 2

Processing Algorithm 2 first applies the modified gradient template given in Table 6.4 [10,35]. It acts the same way as the extreme template in Table 6.1

$$A = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|}\hline a & a & a \\ \hline a & a & a \\ \hline a & a & a \\ \hline \end{array} \qquad I = -0.2$$



$$\text{PWL-function } a = \begin{cases} 2 & , \quad u_{kl} - u_{ij} \leq -2 \\ -2(u_{kl} - u_{ij}) - 2 & , \quad -2 < u_{kl} - u_{ij} \leq -1 \\ 0 & , \quad -1 < u_{kl} - u_{ij} \leq 1 \\ 2(u_{kl} - u_{ij}) - 2 & , \quad 1 < u_{kl} - u_{ij} \leq 2 \\ 2 & , \quad 2 < u_{kl} - u_{ij} \end{cases}$$

Table 6.4: Modified gradient template: *a is a piecewise-linear* (PWL) *function of the argument* $(u_{kl} - u_{ij})$ *[12]. The initial state of the CNN is set equal to the input image.*

(see page 87). The changes are in the threshold value I, and in the piecewise-linear function which is used to evaluate the differences of the input values within the neighbourhood. From the shape of the piecewise-linear function used in the gradient template it can be seen that only large differences between the input values within the neighbourhood contribute to the sum which is then compared to the threshold I. Thus, the gradient template marks regions in the input image which are inhomogeneous, rather than detecting small differences.

In Figure 6.9 we show the images which result from applying the gradient template to the negative of the images in Figure 5.6.[10]

[10]The modified gradient template, like the extreme template, produces similar results for the negative of an input image and for the image itself.

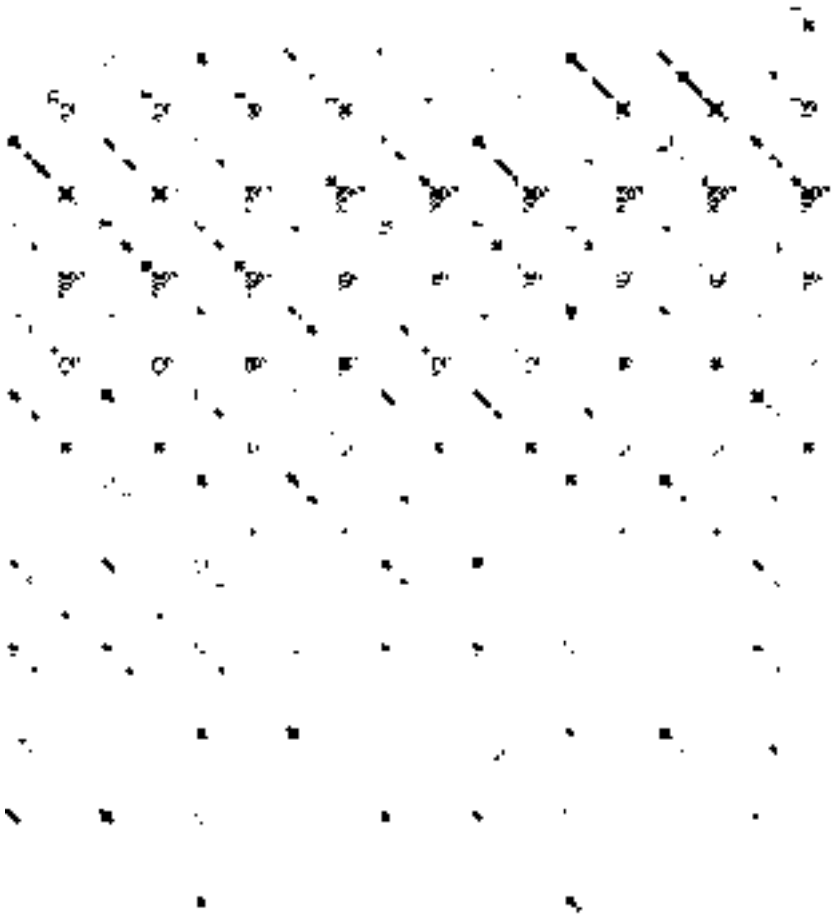


Figure 6.9: *Results after processing the negatives of the images in Figure 5.6* `(bell02)` *with the modified gradient template given in Table 6.4*

The second step of Processing Algorithm 2 (before the time–to–space mapping) is performed with the speed-detection template shown in Table 6.5. This

A		
0	0	0
0	2	0
0	0	0

B		
−0.25	−0.25	−0.25
−0.25	0.75	−0.25
−0.25	−0.25	−0.25

$I = -2$

Table 6.5: *Speed-detection template*
The initial state of the CNN is set equal to the input image.

linear template was used in [43] for speed detection, and checks, as stated in [43], if a black pixel of the input image has three or less black neighbours. We can prove this behaviour from the analysis presented in Section 1.4 for the linear template given in Table 1.2: the speed-detection template (Table 6.5) has the same form.

For the speed-detection template, and in the case of binary input pixels, i.e., $|u_{ij}| = 1$, $\forall\, i, j$, it can be easily shown from the inequalities in (1.14) that the output of the cell is 1 only if the input pixel of the center cell is black, i.e., $u_{ij} = 1$, *and* if it is surrounded by three or less black pixels which is what the cell was supposed to do in [43]. Hence, black input pixels do not appear in the output image unless they are sufficiently isolated, and therefore black regions are mostly deleted.

We show in Figure 6.10 the output produced with the speed-detection template for the input images given in Figure 6.9. Thus, in Figure 6.10 we see the binaural-chip output sequence of the alarm signal `bell02` after the application of the modified gradient template and the speed-detection template on every image.

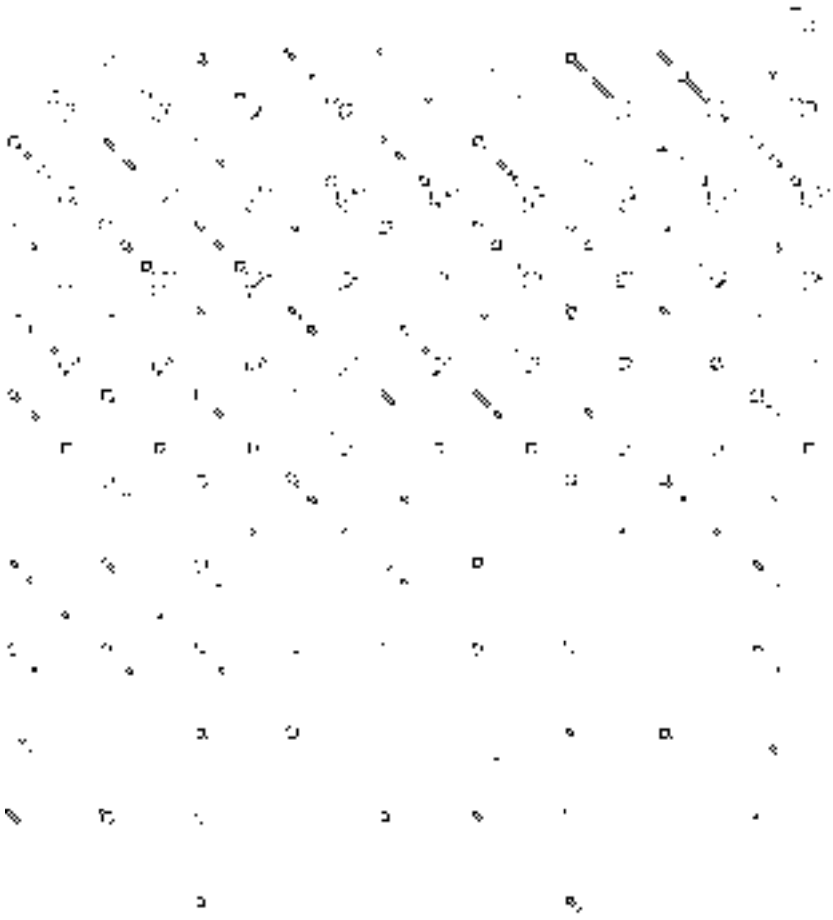


Figure 6.10: *Results after processing the images in Figure 6.9 with the speed–detection template given in Table 6.5*

Processing Algorithm 2 ends with the time–to–space mapping. The characteristic image for signal `bell02` results from collecting all pixels in Figure 6.10 with an OR-combination. As for Processing Algorithm 1, we show the characteristic images of all alarm signals (Figures 6.11 to 6.14).[11] (Again, our examples, and therefore the characteristic images shown below, were computed for the binaural-chip parameter values listed in Section 5.1.3).

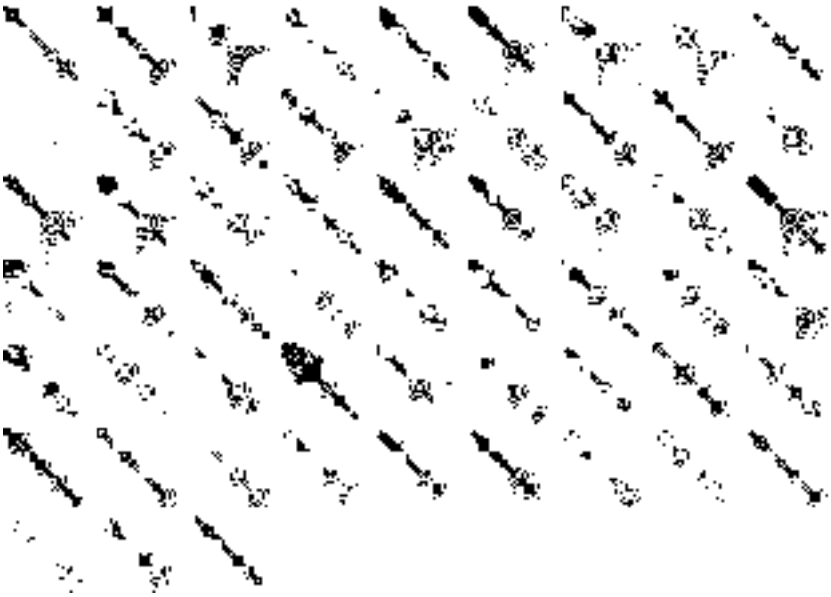


Figure 6.11: *Characteristic images of Processing Algorithm 2 for signals* `bell01` *to* `bell57` *(row wise from left to right)*

[11] The characteristic image of signal `bell02` for Processing Algorithm 2 can be found in Figure 6.11 on the top row, second image from the left.

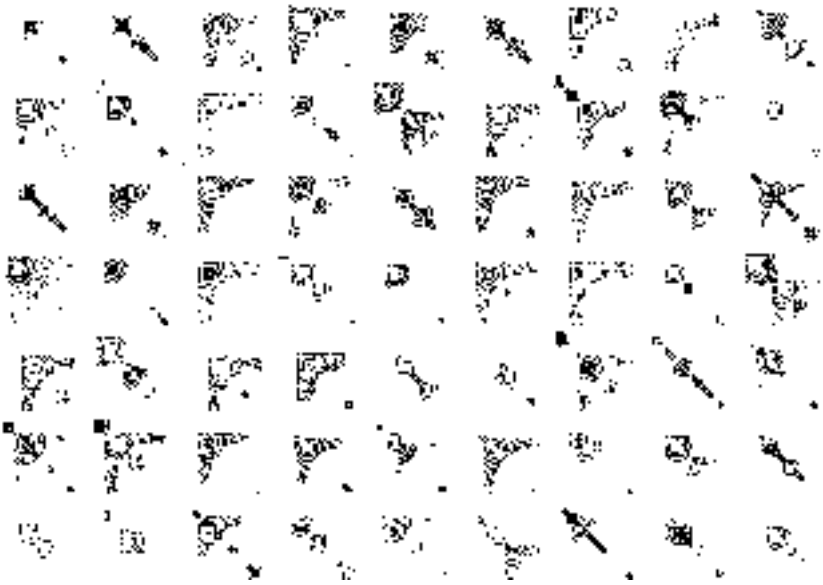


Figure 6.12: *Characteristic images of Processing Algorithm 2 for signals* `horn01` *to* `horn63` *(row wise from left to right)*

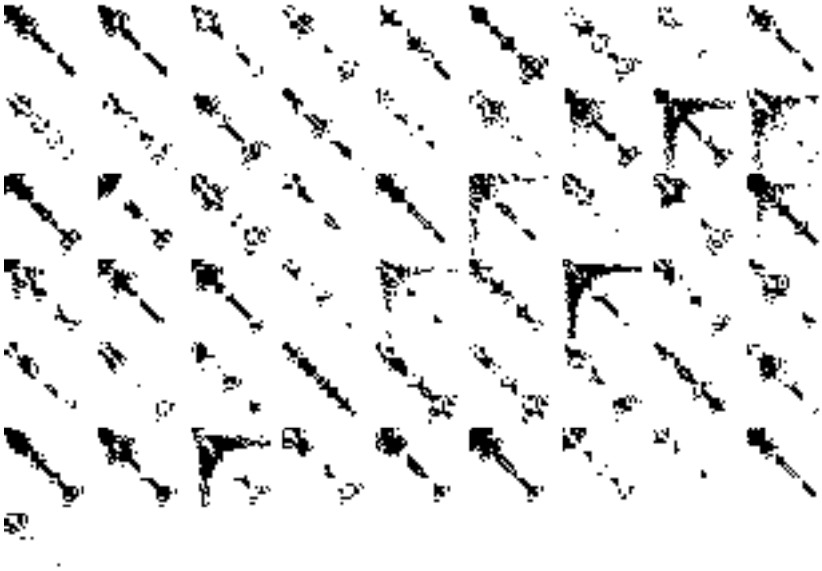


Figure 6.13: *Characteristic images of Processing Algorithm 2 for signals* `phone01` *to* `phone55` *(row wise from left to right)*

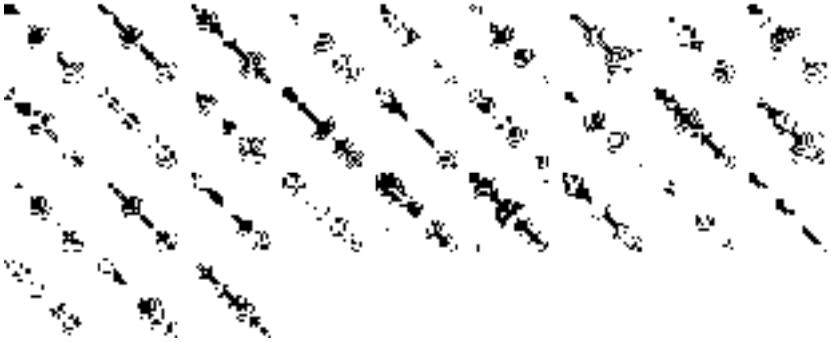


Figure 6.14: *Characteristic images of Processing Algorithm 2 for signals* `ring01` *to* `ring30` *(row wise from left to right)*

6.1.3 Processing Algorithm 3

Processing Algorithm 3 needs only one step before the time–to–space mapping is performed, i.e., the images coming from the binaural chip are processed only once before they are ORed. The linear template used, the so-called *square template* (see Table 6.6), is nearly of the form given in Table 1.2. The differ-

A		
0	0	0
0	2	0
0	0	0

B		
0	−0.25	0
−0.25	0.4	−0.25
0	−0.25	0

$I = -0.5$

Table 6.6: Square template: *The argument of template B is* $(u_{kl} + u_{ij})$ *[12]. The initial state of the CNN is set equal to the input image.*

ences are the following:

- The values at the corners of template B are set equal to zero.
- The argument of template B is $(u_{kl} + u_{ij})$. This means that the input value u_{ij} of the center cell is added to the input values u_{kl} of the cells in the neighbourhood before weighing with template B.

The cell dynamics for the square template is described by the following nonlinear differential equation (see page 9):

$$\dot{x}_{ij} = -x_{ij} + a_5 y_{ij} + \mathcal{U}_{ij} + (4b + 2b_5)\, u_{ij} + I \tag{6.1}$$

We can use the results of the template analysis given in Section 1.4 to see how the cell behaves. This time, $\mathcal{U}_{ij}$ contains only four terms in the brackets (equation (6.2)), and $\mathcal{C}$ (see (1.8)) is redefined in equation (6.3).

$$\mathcal{U}_{ij} = b\left(u_{i-1\,j} + u_{i\,j-1} + u_{i\,j+1} + u_{i+1\,j}\right) \tag{6.2}$$

$$\mathcal{C} = \mathcal{U} + (4b + 2b_5)\, u + I \tag{6.3}$$

Of course, in a real implementation one would realize template B with its usual argument u_{kl} and take the additional influence of u_{ij} with the parameter in the center of the template into account:

B		
0	b	0
b	$\tilde{b}_5$	b
0	b	0

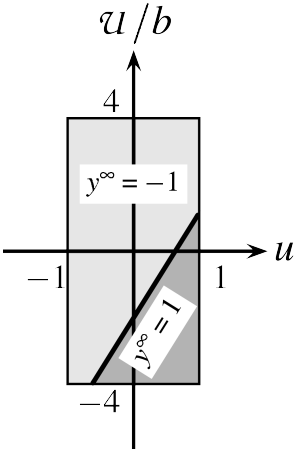
where $\tilde{b}_5 = 4b + 2b_5$

The behaviour of the cell is determined by the following inequalities (see (1.14)):

$$\mathcal{U} > -0.8u + 0.5 \qquad \Longrightarrow y^{\infty} = \quad 1 \tag{6.4a}$$

$$\mathcal{U} < -0.8u + 0.5 \qquad \Longrightarrow y^{\infty} = -1 \tag{6.4b}$$

Thus, the behaviour can be interpreted from Figure 6.15:



- Grey regions, i.e., when $\mathcal{U}/b$, $u \approx 0$, are set to white.
- Isolated dark pixels, i.e., when $u \approx 1$, $\mathcal{U}/b < 1.2$, are mapped to black.
- Pixels in a dark environment, i.e., when $\mathcal{U}/b > 1.2$, go to white.
- Bright pixels, i.e., when $u \approx -1$, are set white.
- Grey pixels in a bright environment go to black.

Figure 6.15: *Points above and below the straight line* $\mathcal{U}/b = 3.2u - 2$ *in the possible region* $-1 \leq u \leq 1$, $-4 \leq \mathcal{U}/b \leq 4$, *lead to* $y^{\infty} = -1$ *and* $y^{\infty} = 1$, *respectively.*

Figure 6.16 shows the images which result from applying the square template to the negative images of Figure 5.6, in order to enhance square-like patterns.[12]

[12] As in the previous processing methods, the images shown in Figure 5.6 are first inverted. The dividing line shown in Figure 6.15 was placed on that position of the plane to work with the negative images.

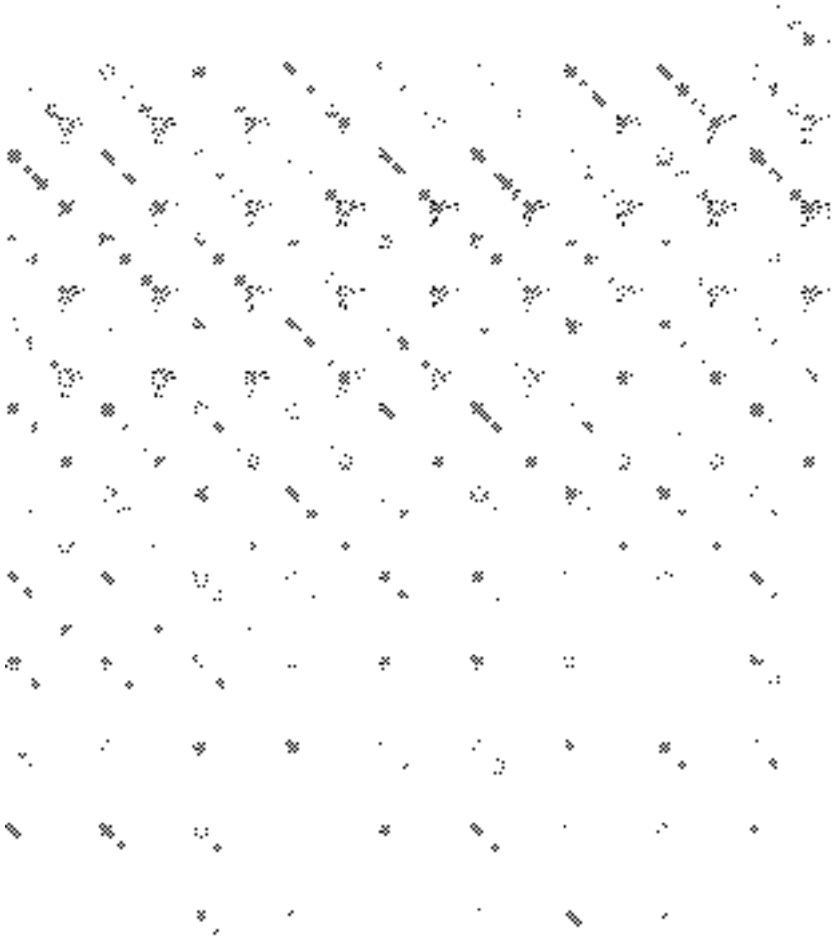


Figure 6.16: *Results after processing the negatives of the images in Figure 5.6* `(bell02)` *with the square template given in Table 6.6*

The black pixels produced with the square template are collected over the image sequence with the OR-combination. The characteristic images which result from Processing Algorithm 3 for the binaural-chip parameter values listed in Section 5.1.3, are shown in Figures 6.17 to 6.20. They are quite similar to the ones of Processing Algorithm 2 (Figures 6.11 to 6.14).

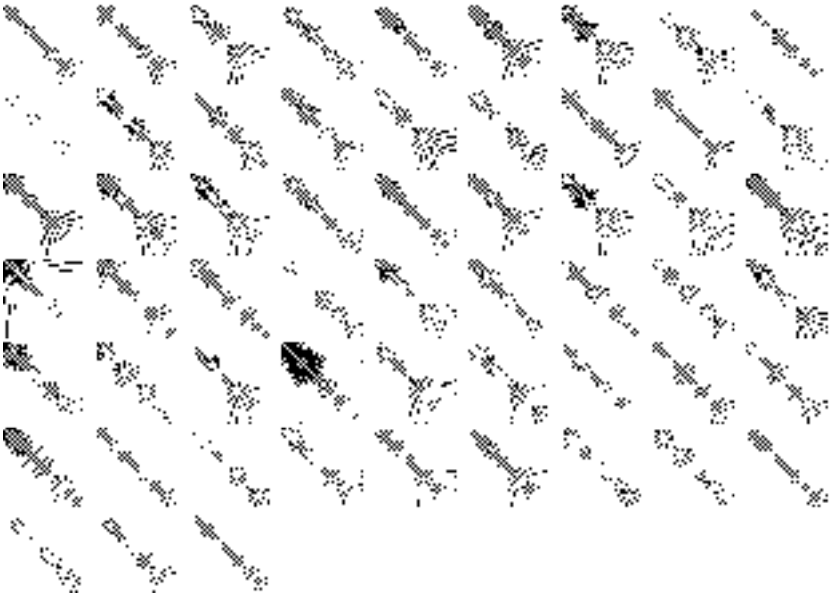


Figure 6.17: *Characteristic images of Processing Algorithm 3 for signals* `bell01` *to* `bell57` *(row wise from left to right)*

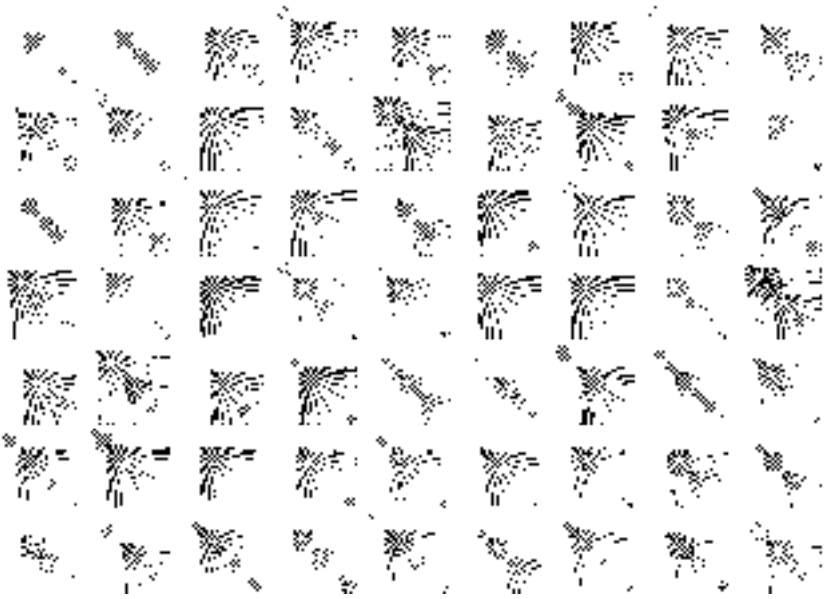


Figure 6.18: *Characteristic images of Processing Algorithm 3 for signals* `horn01` *to* `horn63` *(row wise from left to right)*

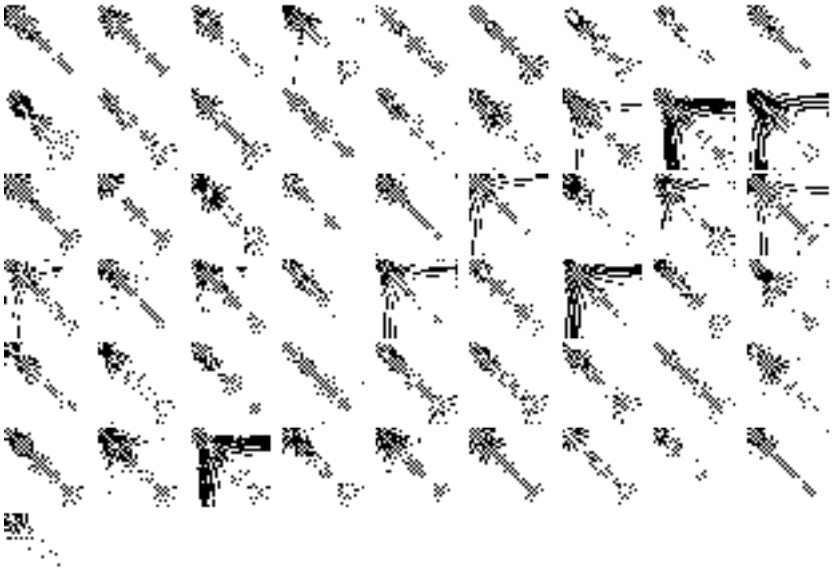


Figure 6.19: *Characteristic images of Processing Algorithm 3 for signals* `phone01` *to* `phone55` *(row wise from left to right)*

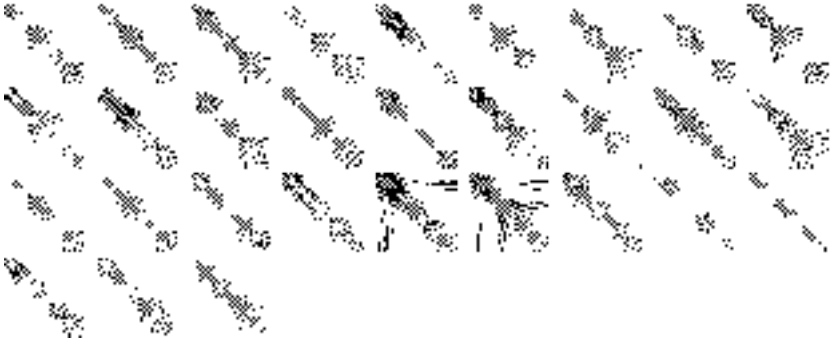


Figure 6.20: *Characteristic images of Processing Algorithm 3 for signals* `ring01` *to* `ring30` *(row wise from left to right)*

In Table 6.7 we list the templates used in Processing Algorithms 1 to 3.

Processing Algorithm 1	Extreme Table 6.1 page 87	Noise Removal Table 6.2 page 88	Edge Extraction Table 6.3 page 88	OR
Processing Algorithm 2	Gradient Table 6.4 page 95	Speed Detection Table 6.5 page 97		OR
Processing Algorithm 3	Square Table 6.6 page 102			OR

Table 6.7: *Templates used in Processing Algorithms 1 to 3*

6.2 Summary

We presented in this chapter three CNN processing methods which generate, from the binaural-chip output sequence, a characteristic image for the acoustical alarm signal fed into the binaural chip. All three methods use an OR-operation as the last step of an image-processing series, in order to perform a time–to–space mapping: information spread over the whole image *sequence* is collected into *one* image. *This image characterizes the alarm signal* and has the following properties: it is similar to characteristic images of signals which belong to the same class, and it is distinct from characteristic images of signals from other classes. Thus, the CNN performs a mapping of the binaural-chip output sequence onto a single image, which specifies the class to which the acoustical signal at the input of the binaural chip belongs.

The first processing method (Processing Algorithm 1) first applies the "extreme template" to the individual images of the binaural-chip output sequence to detect regions with similar values in the grey-scale input images. The output images of this first step have large black regions which contain isolated black and white pixels, i.e., they are "noisy". These artifacts are cancelled with a noise-removal template before the number of black pixels in compound regions is reduced applying an edge-extraction template. All the edges on the processed images of the binaural-chip output sequence are summed up with

an OR-combination, in order to produce a characteristic image of the alarm signal under consideration.

The second processing method (Processing Algorithm 2) uses the "gradient template" to mark non-homogeneous regions in the grey-scale images produced by the binaural chip. This template can be interpreted as the complementary of the extreme template. A direct summation of all black pixels produced during the whole sequence with the gradient template would blur the characteristic image too much. Therefore, the "speed-detection template"[13] is used before the OR-combination, in order to reduce the number of black pixels in the individual images.

The third processing method (Processing Algorithm 3) is a surprisingly simple one: it only needs to process the grey-scale images of the binaural-chip output sequence once with a linear template, before the pixels are summed up. The resulting characteristic images of the acoustical alarm signals are quite similar to those generated with the second method. In fact, we will see in Chapter 8 that the achieved classification error rates for both methods lie in the same range.

[13] The name comes from the first application the template was used for.

Chapter 7

Classification Device

This chapter presents the device used to classify the characteristic images produced by the CNN processing algorithms. The characteristic images can be interpreted as points in space. A simple one-layer perceptron is able to assign the points to the correct classes by separating them with hyperplanes.

7.1 Locating Classes in Space

The three CNN processing algorithms presented in the previous chapter produce characteristic images for the acoustical alarm signals to be classified: each algorithm uses its series of templates (see Table 6.7) and an OR-combination as time–to–space mapping, to transform a phase-preserving two-dimensional representation of a given acoustical alarm signal, i.e., the binaural-chip output sequence presented in Section 5.1, into a single image. *Thus, every processing algorithm attaches one image to every acoustical alarm signal.*

Now, the classification task can be reformulated in the following way: the classification of the given acoustical alarm signals is equivalent to the classification of the characteristic images produced, or, stated differently, we transform the problem of the classification of time-varying signals into an image classification problem. This procedure is valid if the characteristic images retain information about the class the corresponding acoustical alarm signals belong to, i.e., if the characteristic images of signals from the same class are similar, and if the characteristic images of signals from different classes are distinct, which were the requirements we made on the CNN in Section 6.1.

The characteristic images can be represented by vectors whose components are the pixel values of the images. Every vector defines a point in space which stands for the corresponding image. Similar images are mapped onto points close to one another in space, because similarity means that pixel values, and hence the vector components, do not differ "too much". On the other hand, if the images are distinct, the corresponding points in space are far away from each other. Thus, the different classes are clusters of points in space. Figure 7.1(a) illustrates the situation in the plane for points which belong to two different classes, and which are clearly arranged into two clusters. For this example, we can lay an infinite number of straight lines in the plane to divide it into two half-planes, in the course of which all points within one half-plane are said to belong to one class, and all points within the other half-plane to the second class.

The example shown in Figure 7.1(a) is well-behaved in the sense that points from different classes can readily be separated with a straight line. Figure 7.1(b) shows a more complex situation where the points which belong to the class of triangles appear in two clusters. The region in the plane we associate with the class of triangles may be an *assembly of intersections* of half-planes, i.e., both shaded areas in Figure 7.1(b). Thus, the triangles can still be

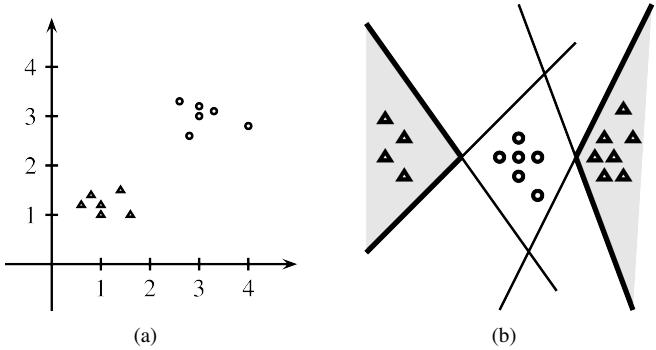


Figure 7.1: (a) *Two different classes of points on the plane arranged into two clusters*

(b) *Two different classes of points on the plane. Points within the shaded regions, i.e., the assembly of two intersections of half-planes, are associated with the class of triangles.*

separated by introducing straight lines in the plane, although with more effort than in the example of Figure 7.1(a), because the corresponding intersections have to be assembled.

The previous examples illustrate another interesting property of the classification performed by dividing the plane into regions we assign to the different classes: the capability of *generalization*. This means that *all* points within the assigned regions are said to belong to the corresponding classes, although only a finite number of points is used to determine the regions. Thus, further points which are not used to determine the regions are automatically assigned to the correct class if they fall into the right region.

What is the situation for the characteristic images of the acoustical alarm signals under consideration? How do the corresponding points lie in space? Is it possible to separate them with "straight lines"? And, how well do the regions attached to the corresponding classes generalize?

We can readily list the following properties in the case of the characteristic images:

- The dimension of the space is equal to the number of pixels in the images (or a subset of them, see below).

- The vector components only have values ± 1 which correspond to the black and white pixels of the output images of the CNN.

These two points answer the first and the second question, respectively: first, the dimension of the space the vectors lie in, is $\gg 1$, because every image has several thousand pixels. And, second, the points we map the characteristic images onto are located at the corners of the unit hyper-cube in that space, because the coordinates of the points have values ± 1.

From page 62 we know that the characteristic images are symmetric along the diagonal. Thus, the pixels under the diagonal are redundant, and the number n of vector components can be reduced to the value given in the next equation, where K is the image resolution along one edge of the square image, i.e., the number of filters in the cascade which models the basilar membrane (see page 62).

$$n = \frac{K(K+1)}{2}, \qquad \text{for } K = 1, 2, 3, \cdots \tag{7.1}$$

For $K \gg 1$, $n \approx K^2/2$. Thus, the dimension n of the space we map the characteristic images onto, is (approximately) half the number of pixels of one image.

The number n in equation (7.1) may be viewed as an upper bound. Since the pixels at the upper right corner of the characteristic images for Processing Algorithms 2 and 3 are mostly white (see Figures 6.11 to 6.14, and 6.17 to 6.20, respectively), they do not contribute class-specific information to the space, i.e., the values of the corresponding vector components are -1 for (almost) *all* images. We might take into consideration only $d-1$ pixels at the right-hand side of the diagonal. Then, the dimension $n_d \leq n$ of the space is given by the next equation:

$$n_d = \frac{1}{2}\left((2K+1)d - d^2\right), \qquad \text{for } 1 \leq d \leq K \tag{7.2}$$

Furthermore, if we ask for $n_d \approx n/2$, i.e., if we reduce the dimension n of the space by (approximately) a factor of two compared to equation (7.1), the width d is given by the following relation:

$$d = \left\lfloor K + \frac{1 - \sqrt{2K^2 + 2K + 1}}{2} \right\rfloor \tag{7.3}$$

where $\lfloor \cdot \rfloor$ denotes the greatest integer less than or equal to a number.[1] Figure 7.2 illustrates which pixels are taken into consideration with this method

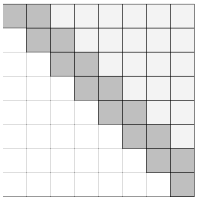


Figure 7.2: *Example for the reduction of the dimension of space: only pixels within a distance* $d-1=2-1=1$ *from the diagonal (at the right hand side) are considered in this* 8×8 *image in order to (approximately) halve the number of shaded pixels.*

in order to (approximately) halve the dimension of space in the case of $K = 8$: from equation (7.3) the width $d = 2$, and from equation (7.2) the dimension $n_2 = 15$, instead of $n = 36$ (from equation (7.1)), i.e., only the darker pixels are taken into consideration, instead of all shaded pixels.

One could use other methods to reduce the dimension of space. The *singular value decomposition* (SVD) [44] could be used to concentrate information spread over all vector components, on a vector of smaller dimension. However, the corresponding algorithms are founded upon numerical computations which can be performed with the required precision only by a digital microprocessor, and the classification system given in Figure 4.1 would then not be realizable entirely in analogue techniques of modest precision.

7.2 The One-Layer Perceptron

The one-layer perceptron lays hyperplanes in space [8]. The hyperplanes are determined by the network parameters, the so-called weights. The basic unit of a perceptron, the *neuron*, can be realized by analog techniques [45]. The structure of the neuron is shown in Figure 7.3(a). It weighs the components y_i, $1 \leq i \leq n$, of an input vector $\mathbf{y}$ with the parameters w_i, $1 \leq i \leq n$, respectively, introduces the constant input 1 and multiplies it with w_0, and

[1] The derivation of equations (7.1) to (7.3) is not shown here. The formulas can be determined from basic geometrical relations.

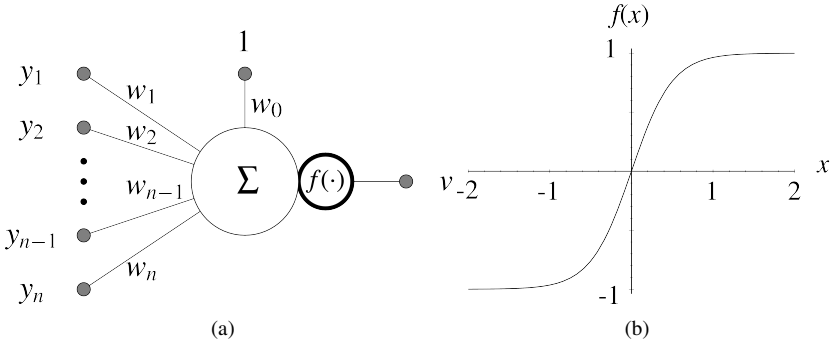


Figure 7.3: (a) *The basic unit of a perceptron, the so-called neuron, builds the scalar product of an input vector* **y** *with a weight vector* **w***, and passes the result through a nonlinear function* $f(\cdot)$*.*

(b) *The nonlinear function* $f(\cdot)$ *we use in our application is the sigmoid function* $f(x) = \tanh(2x)$

passes the overall sum through a nonlinear function $f(\cdot)$. The next equation formulates the operation of the neuron:

$$v = f(\mathbf{w}^T \mathbf{y}), \qquad \text{for } \mathbf{y}, \mathbf{w} \in \mathbb{R}^{n+1} \tag{7.4}$$

Vector **y** can be interpreted as a point in $\mathbb{R}^{n+1}$, where its first component $y_0 = 1$, and the weight vector **w** as a hyperplane of dimension n in $\mathbb{R}^{n+1}$. If the scalar product of **y** and **w** is zero, the point lies on the hyperplane. If the scalar product is negative, the point **y** lies on one side of the hyperplane **w**, and if it is positive, it lies on the other side. Thus, the hyperplane defined by the weight vector **w** can be used to separate points in space. The nonlinear function shown in Figure 7.3(b) bounds the output value v of the neuron to be within ± 1, i.e., $-1 < v < 1$.

For the classification of the acoustical alarm signals, we use four independent neurons, one for every class (bell, horn, phone, ring, see Table 4.1 on page 55). All neurons share the same inputs. This structure is called a one-layer perceptron with four output units (see Figure 7.4).[2] Every neuron lays a hyperplane in space in order to separate the points which belong to "its" class,

[2]For convenience, in Figure 7.4, we dropped the input with constant value 1 shown in Figure 7.3(a).

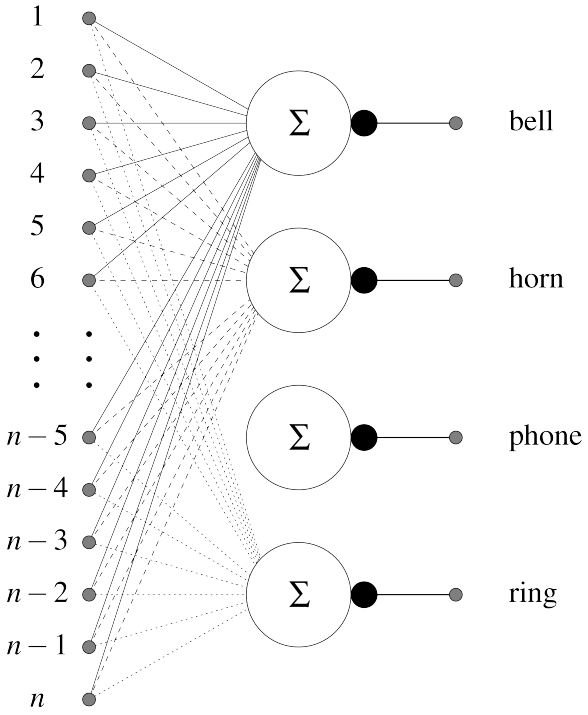


Figure 7.4: *We use a one-layer perceptron with four output units to classify the given acoustical alarm signals.*

from other points. An input vector which, e.g., belongs to the class of bells, is correctly detected if the output of the first neuron is positive, and all other neurons produce negative outputs. Thus, the regions in space assigned to the corresponding classes, each results from the intersection of four half-spaces, one half-space for every hyperplane.[3]

The question now arises, how to place the hyperplanes in space in order to separate the classes in an "optimal" way, given the points in space, or, stated differently, how can the one-layer perceptron learn to adjust the weights of the individual neurons in order to correctly separate the different classes with hyperplanes. We cite a paragraph of [8, p. 10], where the notion of perceptron

[3] If a point falls outside all interesting intersections, e.g., if for a given input two neurons produce a positive output value, the one with the highest value is selected.

is equivalent to the neuron introduced above:[4]

There are numerous learning algorithms for the perceptron. Most of them were developed in the 1960s. They include the perceptron learning algorithm [118] ([46] in our list)*, the Least Mean Squares (LMS) learning algorithm [149]* ([47] in our list)*, and many others [28, 134]* ([48, 49] in our list). *For the most part, the details of these algorithms are beyond the scope of this article. The LMS algorithm, however, is a special case of the* backpropagation *learning algorithm (for multilayer perceptrons), and will be discussed shortly.*

Hush and Horne give an excellent review of the back-propagation algorithm in [8, pp. 12–14]. We use this algorithm to train the network parameters for our classification task. The algorithm is implemented in the NeuroBasic simulator, and runs on the MUSIC parallel computer [50] (see also Appendix A).

At this point, we can answer the two last questions posed on page 113. From simulations (see the classification error rates listed in Chapter 8), we can state that it is possible to separate the points in space with "straight lines", i.e., with hyperplanes. Furthermore, the clusters, or, clouds of points, are well-behaved like the example in Figure 7.1(a): the points which belong to one class are in the same cloud such that no clusters need to be assembled like in Figure 7.1(b). Thus, the four hyperplanes placed by the back-propagation algorithm in space, one for every neuron in the one-layer perceptron of Figure 7.4, are sufficient to separate the four classes of acoustical alarm signals.

Actually, one could think that it might be sufficient to introduce only two hyperplanes in space to divide it into four regions, one region for every class.[5] From simulations it can be shown that the classification error rates are then increased by approximately a factor of two compared to the results achieved with four output units. This happens because the resulting four regions are too wide, i.e., two hyperplanes do not sufficiently restrict the regions assigned to the four classes.

[4] An algorithm which determines the parameter values of the neuron can also be used to set the weights of the four neurons in Figure 7.4, because they work independently of each other, and therefore the weights can be determined by applying the learning algorithm to the individual neurons.

[5] See Chapter 3: if the number of hyperplanes (in general position) is less than or equal to the dimension of the space, the number of regions the space is divided into is two to the power of the number of hyperplanes.

Hence, we wish that the classification device generalizes as well as possible, but not too "generously". The back-propagation algorithm adjusts the weights of the perceptron in order to place the hyperplanes "around" a given set of points, the so-called training set, accordingly to a squared-error criterion [8, 40], *and* given these specific points which are used to train the network. The regions which are determined by the algorithm should, on the one hand, separate the given points without any misclassification, or, at least, with few errors, and, on the other, be as wide as possible in order to allow further points to fall into the correct regions. Thus, there is a trade-off between the exact specification of the region which is assigned to a class given the training set, and the openness to further points which do not form part of the training set. Again, from simulations, it can be shown that, on the one hand, the subset of the characteristic images used to train the perceptron (see Chapter 8) can be perfectly separated with the simple structure shown in Figure 7.4, and that, on the other hand, the introduction of further hyperplanes, e.g., with a two-layer perceptron [8], deteriorates the generalization capability of the classification device.

7.3 Summary

The task of classifying the acoustical alarm signals under consideration has been transferred into an image classification task by the binaural-chip transformation and the CNN processing algorithms. The characteristic images of the alarm signals can be interpreted as points in space, where the (binary) pixel values determine the coordinates of the points. This mapping has the property that similar images are mapped onto close points in space. Thus, the classification of the signals is performed by separating clusters of points in space.

From simulations, it can be shown that four hyperplanes are sufficient to separate the four clusters, i.e., the classes bell, horn, phone and ring. The hyperplanes are realized by a one-layer perceptron with four output units: the weights of the four neurons determine the hyperplanes, and they can be adjusted with the back-propagation algorithm in order to fulfill a given error criterion in an optimal way for the training set, i.e., for a given set of points. The corresponding error can be made smaller by introducing more hyperplanes in space with a more complicated classification device, but then the generalization capability deteriorates: the regions attached to the given classes become too narrow so that points which were not used to place the hyperplanes in

space may fall too far away from the correct regions.

Chapter 8

Results

In this chapter, we list the error rates achieved for the classification of the acoustical alarm signals under consideration. The results were computed for different parameter settings of the binaural-chip model, and for the three CNN processing algorithms presented in Chapter 6. We used a one-layer perceptron with four output units (see Chapter 7) as the classification device.

8.1 Classification Skills of People

Before we give the classification error rates achieved by our system, we address the question of the classification skills of people with normal hearing [42]. The test conditions for people and for the classifier were not the same, and, therefore, the test results may not be compared one–to–one. For example, the acoustical alarm signals under consideration are real-life signals, and it is difficult to quantify the amount of time which the test subjects had been trained to distinguish the signals: not all may have lived for the same length of time in Zurich or be equally familiar with Zurich's tram bells.[1] Additionally, long tests would be an unreasonable demand on a single test subject. Therefore, the classification skills of people were averaged over 14 volunteers which made the tests under the same conditions:

- Every person had to classify 32 randomly chosen signals. 16 signals had been sampled in CD-quality,[2] i.e., with a sampling frequency $f_s = 44.1\,\text{kHz}$ and 16-bit resolution, and the other 16 signals with the lower sampling frequency $f_s = 8192\,\text{Hz}$, and stored as 8-bit μ-law encoded data (see Section 4.2).
- A signal could not be replayed, and there was no classification option "unknown".

Table 8.1 lists the results for the classification error rates of the 14 volunteers. From these results we can state the following: people did not classify

sampling data in	bell	horn	phone	ring	total
CD-quality	4%	0%	0%	5%	2%
lower quality	2%	0%	0%	45%	12%

Table 8.1: *Classification error rates of healthy people [42]*

with a zero error rate. The quality of the signals, i.e., the sampling quality, affected the classification. Apparently, the recognition of car horns and phones did not seem to pose any problems for people.

One could think that tram bells and tram rings are mixed up, because the source of these two classes of acoustical alarm signals is the same, and the

[1] The acoustical alarm signals under consideration were recorded in Zurich.
[2] CD stands for Compact Disc.

only difference is in the rapid repetition of the "same" sound in the class of rings. Yet, misclassifications of ring signals were also due to wrong detections of phone signals, i.e., misclassifications cannot be explained only by isolating the classes where trams are involved.

8.2 Unifying the Test Conditions

In order to be able to compare the classification error rates achieved by CNNs with the error rates published in [30] and [31], we decided to train our classification device with a training set of the same size as in [31]. Table 8.2 lists how many signals of every class were used in order to determine the

class	# signals for training	# signals for testing	total # signals
tram bells	52	5	57
car horns	57	6	63
phones	50	5	55
tram rings	27	3	30
all classes	186	19	205

Table 8.2: *Number of signals used to train the classification device: approximately 90% of the signals of every class were used to train the one-layer perceptron. The remaining signals formed the test set.*

weights of the one-layer perceptron introduced in Chapter 7.[3] The signals which were not used in the training set, i.e., $\approx 10\%$ of the signals, served as test signals for the classification task. The really interesting results are the classification error rates achieved for the test signals, because they tell us how well the classification works for "unknown" samples of the acoustical alarm signals.

The 186 signals of the training set were chosen randomly out of the total set of 205 acoustical alarm signals. As in [31], we averaged the classification error rates over 30 runs, where in every run a new training set was randomly chosen. However, we could see that 30 runs were not sufficient to produce

[3]The 1-D to 2-D transformation performed by the binaural chip, and the CNN processing algorithms, did not need to be trained to solve their tasks. They used the *a priori* knowledge needed for their design.

reliable results, i.e., the number of misclassified test signals varied approximately 4% ($\approx 23/(30 \cdot 19)$) depending on the $30 \cdot 186$ training signals which were chosen during the 30 runs. In order to compare the classification error rates achieved for different parameter settings of the binaural-chip model (see Section 5.1), and for the three CNN processing algorithms presented in Chapter 6, we fixed 30 training/test pairs, and averaged the performance of the three CNN processing algorithms for different parameter settings of the binaural-chip model over these 30 "unified", or, normalized, training/test pairs. Since other classification methods may be compared with the one presented in this thesis, we list in Table 8.3 the signals which we randomly chose as test signals in the 30 runs. Thus, all results given hereafter can be compared with each other, because they have been computed using the same training/test pairs.

run	bell					horn					
1	9	10	12	20	23	5	23	31	36	47	58
2	10	30	36	47	56	9	35	38	52	54	56
3	4	10	27	39	50	17	33	34	35	54	57
4	4	18	27	28	48	20	21	48	50	53	60
5	16	30	34	37	44	8	35	42	51	54	61
6	1	7	15	28	31	6	31	33	53	59	62
7	1	23	25	26	33	4	10	16	27	31	53
8	6	8	13	22	24	12	14	24	27	50	62
9	1	4	11	16	45	2	4	26	27	44	60
10	12	15	42	49	55	2	5	13	26	38	42
11	10	12	37	42	53	2	37	43	47	60	63
12	21	24	27	31	44	15	19	32	38	57	61
13	5	9	10	23	48	4	19	40	43	52	57
14	3	14	16	20	39	2	4	9	11	16	41
15	5	6	7	13	14	24	26	29	35	36	54
16	7	11	25	32	37	9	12	49	51	57	59
17	19	22	25	31	44	24	31	33	34	39	43
18	20	23	24	41	45	2	11	19	31	41	58
19	5	9	10	21	26	15	27	33	57	61	62
20	8	9	12	14	27	20	23	36	51	59	60
21	7	32	36	41	52	7	17	18	21	38	48
22	12	14	31	35	38	26	41	43	50	61	63
23	13	24	33	40	53	15	18	28	34	54	63
24	1	12	28	48	49	13	14	17	26	32	39
25	1	15	21	42	51	12	14	31	39	45	47
26	21	36	42	43	53	19	32	39	52	54	60
27	26	48	51	54	57	13	16	23	35	39	40
28	10	18	37	43	52	3	31	45	55	61	62
29	17	23	25	26	28	14	15	24	37	45	63
30	37	38	40	44	57	1	10	15	27	31	48

Table 8.3: *Labels of the test signals used in the 30 runs, classes bell and horn (see next page for classes phone and ring)*

run	phone					ring		
1	2	12	23	30	39	15	17	20
2	18	23	48	53	55	19	22	25
3	8	16	17	40	42	2	7	16
4	4	12	14	21	35	6	9	14
5	2	17	18	26	42	8	10	20
6	3	5	7	43	44	4	5	23
7	3	11	16	35	50	4	12	25
8	14	20	23	26	28	23	27	30
9	6	19	22	29	48	4	19	25
10	1	4	29	50	53	18	21	24
11	8	15	24	29	46	20	21	25
12	3	4	8	24	40	10	29	30
13	5	8	13	37	53	9	24	26
14	20	23	43	49	52	1	14	18
15	16	21	33	36	38	8	26	29
16	16	20	24	27	40	13	18	22
17	9	21	22	23	28	1	19	24
18	5	20	26	34	40	13	20	22
19	14	21	27	34	37	5	13	24
20	13	35	36	50	52	22	23	26
21	1	3	10	14	45	2	6	27
22	8	28	32	37	50	14	22	28
23	13	17	25	44	51	18	19	26
24	13	14	24	36	54	24	25	29
25	29	30	31	39	54	9	16	25
26	25	32	39	44	50	17	20	22
27	9	34	35	44	51	6	7	28
28	7	8	14	36	41	4	6	22
29	2	7	20	27	41	3	10	16
30	20	29	39	43	50	1	8	11

Table 8.3: (Continued) *Labels of the test signals used in the 30 runs, classes phone and ring*

8.3 Parameter Settings of the Binaural-Chip Model

For the simulations whose results we present, we used the acoustical alarm signals under consideration, sampled at a rate of $f_s = 8$ kHz with 8-bit μ-law encoded resolution. The signal quality which results from this sampling procedure is the one which can be retained in a future analogue realization of the classification system. The different parameter settings we used for the binaural-chip model are given in Table 8.4. Every column of Table 8.4 con-

parameter	Setting 1	Setting 2	Setting 3
`filters`	64	64	128
`fu` [Hz]	100	100	100
`fo` [Hz]	3750	3750	3750
`q`	1	1	1
`fs` [Hz]	8192	8192	8192
`r`	32	64	32
`threshold`	0.05	0.05	0.05

Table 8.4: *Different parameter settings of the binaural-chip model (see also Table 5.1 on page 63)*

tains the parameter values of one setting. Setting 1 corresponds to the values given on page 64, i.e., these are the parameter values which were used to produce the output images shown in Section 5.1. Setting 2, compared to Setting 1, doubles the rate at which the images are produced in order to exploit the frequency/time trade-off discussed in Section 5.1.2. Lastly, Setting 3 exploits the frequency/time trade-off by doubling the number of filters in the model of the basilar membrane, instead of increasing the rate r.

8.4 Classification Error Rates of the Training Set

Tables 8.5 to 8.7 list the *classification error rates of the training set* achieved with Processing Algorithms 1 to 3 (see Sections 6.1.1 to 6.1.3, respectively) for Settings 1 to 3 (see previous section). The results were computed taking into consideration $d-1$ pixels at the right-hand side of the diag-

d	CNN Processing Algorithm	bell	horn	phone	ring	total
18	1	0%	0%	0%	0%	0%
	2	0%	0%	0%	0%	0%
	3	0%	0%	0%	0%	0%
K	1	0%	0%	0%	0%	0%
	2	0%	0%	0%	0%	0%
	3	0%	0%	0%	0%	0%

Table 8.5: *Classification error rates of the training set for Processing Algorithms 1 to 3, Setting 1*

d	CNN Processing Algorithm	bell	horn	phone	ring	total
18	1	0%	0%	0%	0%	0%
	2	0%	0%	0%	0%	0%
	3	0%	0%	0%	0%	0%
K	1	0%	0%	0%	0%	0%
	2	0%	0%	0%	0%	0%
	3	0%	0%	0%	0%	0%

Table 8.6: *Classification error rates of the training set for Processing Algorithms 1 to 3, Setting 2*

d	CNN Processing Algorithm	bell	horn	phone	ring	total
37	1	0%	0%	0%	2%	0%
	2	0%	0%	0%	1%	0%
	3	0%	0%	0%	1%	0%
K	1	0%	0%	0%	0%	0%
	2	0%	0%	0%	0%	0%
	3	0%	0%	0%	1%	0%

Table 8.7: *Classification error rates of the training set for Processing Algorithms 1 to 3, Setting 3*

onal (see page 114).[4] Thus, in the case of $d \neq K$, the classification device, i.e., the one-layer perceptron, was trained using only (approximately) half of the non-repeated pixels of the characteristic images (see Figure 7.2).

From the results listed in Tables 8.5 to 8.7, we can state that the one-layer perceptron with four output units (see Section 7.2) is able to separate the points of the training set with a negligible number of errors.

8.5 Classification Error Rates of the Test Set

In this section, we list the *classification error rates of the test set*.[5] These are the results which tell us how good the performance of our alarm-signal classification system with CNNs is, i.e., how well it works for new acoustical alarm signals. Thus, analogously to the previous section, we list in Tables 8.8 to 8.10 the classification error rates of the test set achieved with Processing Algorithms 1 to 3 for Settings 1 to 3, i.e., for the three parameter settings of the binaural-chip model given in Table 8.4. Again, d was chosen such that either (approximately) half of all,[6] or, in the case of $d = K$, all, non-repeated pixels of the characteristic images were considered.

d	CNN Processing Algorithm	bell	horn	phone	ring	total
18	1	22%	3%	5%	40%	15%
	2	21%	2%	3%	24%	11%
	3	15%	1%	1%	32%	10%
K	1	21%	6%	1%	48%	15%
	2	21%	2%	3%	24%	11%
	3	17%	1%	1%	32%	10%

Table 8.8: *Classification error rates of the test set for Processing Algorithms 1 to 3, Setting 1*

[4] Actually, $d = K$ means that *all* non-repeated pixels of the characteristic images were taken into consideration.

[5] Again, see Section 8.2, the results presented in this chapter are the classification error rates averaged over 30 runs.

[6] See Figure 7.2

d	CNN Processing Algorithm	bell	horn	phone	ring	total
18	1	32%	13%	11%	39%	22%
	2	13%	0%	1%	28%	8%
	3	17%	1%	5%	30%	11%
K	1	26%	10%	5%	36%	17%
	2	14%	0%	1%	27%	8%
	3	15%	1%	5%	33%	11%

Table 8.9: *Classification error rates of the test set for Processing Algorithms 1 to 3, Setting 2*

d	CNN Processing Algorithm	bell	horn	phone	ring	total
37	1	21%	14%	1%	31%	15%
	2	11%	3%	4%	26%	9%
	3	9%	1%	4%	26%	8%
K	1	19%	14%	2%	29%	15%
	2	11%	2%	4%	23%	8%
	3	10%	1%	4%	26%	8%

Table 8.10: *Classification error rates of the test set for Processing Algorithms 1 to 3, Setting 3*

8.6 Summary

In order to be able to evaluate the error rates achieved with our system, we presented some results concerning the classification skills of people with normal hearing in the first section of this chapter. One cannot compare one–to–one the classification error rates averaged over the 14 test subjects with the results of the simulations, because the test conditions were not equivalent. Nevertheless, we can make the following qualitative statements: people *did* misclassify a part of the signals, the number of classification errors grew for lower signal quality, and all car horns and phones were recognized as such.

The second section of this chapter lists the randomly chosen test signals which were used to average the classification error rates over 30 runs. The resulting unified training/test pairs allow the comparison of other classification methods with the one presented in this thesis.

In the next section, we listed three different parameter settings of the binaural-chip model. The classification error rates given in the last sections of this chapter were computed for these three parameter settings.

The classification error rates of the training set were nearly 0%. This means that a simple one-layer perceptron with four output units managed to separate the points of the training set without making (hardly) any errors.

The classification error rates of the test set give us a measure, how well our classification system works, because it has to prove that it recognizes signals it has never been confronted with before. The results will be commented on in the next chapter.

Chapter 9

Conclusions

In the last chapter of this thesis, we comment on the results achieved with our system for the classification of the given acoustical alarm signals. We compare the performance with that of other schemes, and list some concluding remarks.

9.1 CNNs for the Classification of Acoustical Alarm Signals

In the second part of this thesis, we investigated the following question:

Are CNNs generally suitable for the processing of non-stationary signals?

In order to be able to answer this question, we decided to solve an acoustical alarm-signal classification problem by using CNNs. The non-stationary acoustical alarm signals belong to four different sets (see Table 4.1 on page 55).

Since the signals under consideration are acoustical signals, they first had to be transformed into images in order to process them with CNNs. We chose a phase-preserving transformation which can be performed in real-time by an existing analogue device, the so-called binaural chip (see Section 5.1). Furthermore, the transformation performed by the binaural chip can be realized by an analogue device based on CNNs: the part of the binaural chip which models the basilar membrane, i.e., the filter cascade of second-order low-pass filters, can be realized by CNNs which are operated as dynamical systems, and not as mapping devices as in image processing (see Section 5.2). Thus, the preprocessing stage needed to transform the acoustical alarm signals into images may well be realized using CNNs.

The transformation of the acoustical signals into images provided a series of images for every signal. The image-processing capabilities of CNNs were exploited in order to extract meaningful patterns from every image of the sequence for a given signal. In Chapter 6, three CNN processing algorithms were presented, which, by performing a simple time–to–space mapping with an OR-combination, generated a single characteristic image for every given acoustical alarm signal. Thus, the idea of using CNNs for the classification of the given non-stationary signals was to transform the problem into one of image classification.

The classification of the characteristic images produced by the CNN processing algorithms was performed by a one-layer perceptron with four output units (see Chapter 7). This simple classification device solved the task, because the images fulfilled the following requirements: characteristic images of signals from the same class were similar, and characteristic images of signals from different classes were distinct from the point of view of the classifier.

Thus, the CNN was able to process the images provided by the 1-D to 2-D transformation such that a one-layer perceptron was sufficient to solve the resulting image-classification problem.

9.2 Commenting Our Results

In this section, we list some comments on the results given in Section 8.5, i.e., on the classification error rates of the test set achieved for different parameter settings of the binaural-chip model, different CNN processing algorithms, and considering either (approximately) half, or all, non-repeated pixels of the characteristic images.

- Generally, signals from the classes horn and phone were better recognized than signals from the classes bell and ring. The worst recognition rates were those for the class of rings. This corresponds to the behaviour observed for healthy people (see Section 8.1). Thus, our system encountered the same difficulties as people. Especially, signals from the class of rings lead to most errors. For our system one might argue that this is due to the smaller set, i.e., only 27 signals from the class of rings were available to train the classification device, approximately half the number for the other classes (see Table 8.2). For people, this argument may not be valid, since there is no obvious reason for tram rings to be less known than other alarm signals. Thus, for the given system, the reason given above, i.e., that the training set for the class of rings might not be sufficiently large, is not a satisfactory answer to the question why rings are harder to classify than the other acoustical alarm signals. We believe our system is not particularly ill-suited to classify signals from the class of rings, but rather that the nature of these signals leads to unclear class-specific features.

  The similar behaviour of our system and the classification skills of people might not be very surprising since our system uses as a preprocessing device a model of the basilar membrane (see Section 5.1).

- We presented in Sections 6.1.1 to 6.1.3 three CNN processing algorithms. Our simulations showed that the most complicated (Processing Algorithm 1) is the one that gave the worst classification error rates. From the point of view of a system realization this is a positive result, since the problem should be solved with as little effort as

possible. For certain parameter settings of the binaural-chip model, Processing Algorithm 2 gave the best results, at the cost of needing two image-processing steps, one of which uses the nonlinear gradient template given in Table 6.4. On the other hand, there were cases where the simple Processing Algorithm 3, which needs only one image-processing step with an easily realizable linear template (see Table 6.6 on page 102), achieved the lowest classification error rates. *Thus, we could show that the use of nonlinear templates does not necessarily lead to higher recognition rates.*

- The choice of considering only a reduced number of pixels for the image-classification task (see Figure 7.2 on page 115) was motivated by the fact that for Processing Algorithms 2 and 3 most black pixels of the characteristic images lie along the diagonal (see for example Figures 6.11 and 6.17, respectively). With only one exception (Table 8.10, Processing Algorithm 2), the total classification error rates of Processing Algorithms 2 and 3 did not increase by reducing the number of considered pixels by (approximately) a factor of two.[1]

  Since the black pixels of the images produced by Processing Algorithm 1 are generally not concentrated along the diagonal (see Figure 6.5), one would expect a clearly worse behaviour in the case of a reduced number of pixels. This can be stated only for Setting 2 (Table 8.9), where the performance decreased by 5%. Thus, in general, the given reduction of the number of considered pixels did not lead to higher classification error rates.

- Setting 2 differs from Setting 1 in that the rate r at which the binaural-chip model produces the series of images is twice as high. This leads to a deterioration of the classification performance for Processing Algorithms 1 and 3 (up to 7%, and 1%, respectively). *Thus, increasing the rate of image flow does not necessarily produce better results for these CNN processing algorithms.* On the other hand, the results for Processing Algorithm 2 could be improved in order to reach a classification error rate of 8%.

  Setting 3 increases the amount of data which is processed with the CNN by increasing the image resolution, i.e., by doubling the number of filters in the cascade which models the basilar membrane. The

[1] The error rate of Processing Algorithm 2, Setting 3, increased from 8% to 9%, which, in our opinion, is a negligible difference.

128×128 pixel images are at the upper realization limit for an analogue implementation (see page 85). The classification rates could be improved for Processing Algorithms 2 and 3 in order to achieve, again, a classification error rate of 8%.

In summary, one is faced with a compromise: our system achieves an error rate of 8% for the classification of the given acoustical signals either by increasing the rate at which the binaural-chip model produces the two-dimensional representation of the acoustical alarm signals, and using a more complicated CNN processing algorithm (Processing Algorithm 2), or by increasing the image resolution, where then a simple template can be applied (Processing Algorithm 3). Since the processing speed of an analogue CNN chip is sufficiently high, and the image resolution of Setting 2 is 64×64, i.e., nearly the one of the existing binaural chip,[2] the first solution might be chosen for a chip implementation of our classification system. Of course, the best would be to reach a classification performance of 8% or less errors with a simple linear template for this low image resolution: the results for Processing Algorithm 3 might be improved by tuning the square template in order to better extract the patterns of the image series produced with Settings 1 and 2 (see below).

9.3 Comparison with the Performance of Other Schemes

In order to evaluate the performance of our classification system, we presented in Section 8.1 some classification results for people with normal hearing, and we saw that the test subjects did not classify with a zero error rate.

It is also interesting to compare the performance of our system with the results achieved by other schemes. Table 9.1 summarizes the classification error rates achieved in [30] and [31]. The classification error rates achieved with the system presented in this thesis could clearly improve these results.

Recently, the classification problem was also solved by using Hidden Markov Models (HMMs) and a maximum likelihood classifier [51]. The ob-

[2] With a resolution of 56×56.

Scheme	Classification Method	Results for the Test Set
Spectral Techniques [30]	Minimum-Distance	22%
Neural Networks [31]	Perceptron	22%
	Masking Field	19%

Table 9.1: *Classification error rates of other schemes*

servation probability density function of each class was modelled by a four-state HMM. The results listed in Table 9.2 can be compared one–to–one to the

Scheme	bell	horn	phone	ring	total
HMM (4 states)	7%	11%	0%	12%	7%

Table 9.2: *Classification error rates achieved with HMMs [51]*

results given in Section 8.5, because all were obtained by using the unified test signals listed in Table 8.3.

Two remarks should be made with regard to these results:

- Compared to our results, the better classification achieved with HMMs for signals from the class of rings is deteriorated by a worse performance in the classification of horns.

- The good overall performance achieved with HMMs gives the present benchmark for the classification error rate: 7%. We achieve with our system 8% in five cases (see Tables 8.9 and 8.10). Thus, the classification of the given alarm signals can be performed by CNNs without any serious performance loss. Therefore, we could demonstrate that CNNs are suited to process non-stationary signals, especially acoustical signals. This may smooth the way for further investigations in the field of speech recognition with CNNs.

9.4 Suggestions for Further Work

Our simulations require high storage capacities and high computation power, since a large amount of images is produced and has to be processed. We did not run simulations for all possible combinations of the parameter values of the binaural-chip model. Apart from the settings given in Table 8.4, we investigated other reasonable values for the parameters of the binaural-chip model, but no better classification error rates could be found. Another possibility is to tune the template values for the CNN processing algorithms, e.g. the shape of the nonlinearity of the gradient template, or the location of the straight line which defines the behaviour of the square template (see Figure 6.15). As disk space becomes cheaper and computers faster, a more complete test series may be run, and better results might be observed.

In our opinion, it is more important to realize the image classification, i.e., the task performed by the one-layer perceptron, with CNNs. This would allow to incorporate the whole system into an analogue chip based on CNNs. Discrete-time CNNs [9] have been applied to texture classification [52]. The difficulty of realizing an image-classification device based on *analogue* CNNs is to determine the appropriate templates.

9.5 Summary

We have shown that it is possible to perform the given classification task by using CNNs. Nonetheless, an appropriate transformation of the acoustical alarm signals had to be found in order to provide images as input to the CNN. The idea of using CNNs for the classification of the given acoustical alarm signals was to transform the problem into one of image classification. The CNN processed a series of images in order to produce a characteristic image for every acoustical alarm signal. The classification of the characteristic images was performed by a one-layer perceptron with four output units. It was shown that the phase-preserving transformation of the acoustical signals to a 2-D representation might well be performed with a CNN-like structure by realizing second-order low-pass filters with CNNs for the model of the basilar membrane. The whole classification system cannot yet be realized entirely with CNNs, because the image classification is performed with a perceptron.

The solution presented in the second part of this thesis might be realized

as an analogue chip based on the CNN structure when the first programmable CNN chips with a sufficient number of cells appear. Then, the advantages of analogue techniques, i.e., low power consumption at high processing speeds, might be used to provide sophisticated aids for the hearing impaired.

Appendix A

Simulation Tools

This appendix is split into two parts. In the first, we present the MATLAB *model of the binaural chip. The second part deals with our simulator of CNNs which runs on a multiprocessor system, i.e., on a parallel computer.*

A.1 The Model of the Binaural Chip

We built a simulator for the binaural chip with the MATLAB package for numeric computation [53]. This package has given us a flexible and easy–to–use tool for a fast implementation of the binaural-chip model. The sampled acoustical alarm signals serve as input to the model which, in turn, produces the required input images for the CNN. Thus, the MATLAB model implements the two basic building blocks of the binaural chip: the model of the basilar membrane, i.e., the filter cascade (see page 58), and the two-dimensional correlation (see page 60).

The following parameters control the behaviour of the binaural-chip model (see also Table 5.1 on page 63):

- `filters` specifies the number of filters the cascade consists of
- `fu` and `fo` set the lower and upper pole frequencies f_l and f_u, respectively, of the filter cascade (see equations (5.2) on page 59) [1]
- `q` is the quality factor q_p of the low-pass filters (see equation (5.1))
- `fs` specifies the sampling frequency, i.e., the frequency the alarm signals were sampled with
- `r` sets the image rate (in images per second)
- `threshold` controls the sensitivity of the basilar membrane model: excitations on the basilar membrane which are smaller than this threshold value are cancelled

The MATLAB program is split into two independent parts. The first models the binaural chip, and the second transforms the output data of the model into `GIF` images (Graphics Interchange Format).

[1] `fu` and `fo` stand for lower and upper pole frequencies, respectively, from German *untere und obere Polfrequenz*.

A.1.1 The programs of the binaural-chip model

The MATLAB model of the binaural chip consists of six short files. Table A.1 lists them and gives a short description of their contents. The code is given on the next pages.

`prodbin.m`	**main program** calls `readalarm8kHz`, `binauralchip`, and `writebcsequence` The input signal is split into several parts to avoid large matrices and swapping or other memory problems on the computer.
`readalarm8kHz.m`	reads files which contain 8-bit μ-law encoded acoustical (alarm) signals sampled at 8 kHz
`binauralchip.m`	**produces the output of the binaural chip** This file calls `travelingwave.m` which, in turn, calls `cascade.m`. These files model the first building block of the binaural chip, namely the basilar membrane. Finally, `binauralchip.m` performs the correlation after noise cancellation with a threshold value (see above).
`travelingwave.m`	computes the traveling wave on the basilar membrane (see Figure 5.2 on page 60)
`cascade.m`	generates a filter cascade to model the basilar membrane
`writebcsequence.m`	writes the images which come out of the binaural-chip model to a file in binary format

Table A.1: MATLAB*'s* `m`*-files for the binaural-chip model*

prodbin.m

```
%
%                  Matlab script for the model of the
%
%                       B I N A U R A L  C H I P
%
%               audio base: Uvacek's alarm signals
%                           (57 bell, 63 horn,
%                            55 phone, and 30 ring signals)
%                           sampling frequency fs=8192Hz
%
%       This main program produces files which contain the output
%       images of the binaural chip in binary format.
%
%       See also readalarm8kHz, binauralchip, writebcsequence.
%
%       J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% constants

  dirsound = '/disks/isibee17/osuna/cnn/SoundsCD/';
  dirbin = '/disks/isibee17/osuna/cnn/matlab/bin/';

  eightkilohertz=1;     % boolean, used to choose the sampling
                        %          frequency fs

% initializations

  if eightkilohertz
    dirsound = '/disks/isibee17/osuna/cnn/Sounds/';

    fs =  8192;         % sampling rate 8 kHz
    r = 32;
  end

%-----------------------------------------------------------------------

% main program

  for i=1:30
    for k=1:4

      if k == 1
        class = 'horn';
        name = 'h';
      elseif k == 2
        class = 'bell';
        name = 'b';
      elseif k == 3
        class = 'phone';
        name = 'p';
      else
        class = 'ring';
        name = 'r';
      end
```

```
      if eightkilohertz
        u=readalarm8kHz([dirsound class ...
                         '/y' class num2str(i) '.sound']);
      end
      lengthu = length(u);
      maxlengthpartu = min([fs/gcd(fs,r) lengthu]');

      partu = ceil(lengthu/maxlengthpartu);
      partubegin = zeros(partu,1);
      partuend   = zeros(partu,1);
      for m=1:(partu-1)
        partubegin(m) = (m-1)*maxlengthpartu+1;
        partuend(m)   = m*maxlengthpartu;
      end
      partubegin(partu) = (partu-1)*maxlengthpartu+1;
      partuend(partu)   = lengthu;

      for m=1:partu
        if partu==1
          seq = binauralchip(u(partubegin(m):partuend(m)),...
                             fs,r,0,[],[]);
        else
          if m==1
            [seq,Zf,h] = ...
              binauralchip(u(partubegin(m):partuend(m)),fs,r,m,[],[]);
          else
            [seq,Zf,h] = ...
              binauralchip(u(partubegin(m):partuend(m)),fs,r,m,Zf,h);
          end
        end
        writebcsequence(seq,...
                [dirbin class '/' class num2dig(i,2) '.bin'],fs/r);
        clear seq
      end
    end
  end

% end of program "prodbin"
```

readalarm8kHz.m

```
function y = readalarm8kHz(filename)
%
% y = readalarm8kHz('filename')
%
% readalarm8kHz reads the binary file "filename" which contains
%               audio data and produces vector y which contains the
%               sampled values
%
%               the audio files are Uvacek's four classes of
%               alarm signals (phones, rings, bells and horns)
%
%               audio specifications:
%
%                       8kHz sampling rate (8192 Hz)
%                       8-bit mu-law encoded (12 bits --> 8 bits)
%
%               See also FOPEN, MU2LIN.
%
% J.A. Osuna, 16.12.1994

%-------------------------------------------------------------------------

% initializations

  FID = fopen(filename);

% function body

  y8bit = fread(FID);                   % 8-bit compressed values
  y     = mu2lin(y8bit);                % decompressed values

  y = y/max(abs(y));

  fclose(FID);

% end of function "readalarm8kHz"
```

binauralchip.m

```
function [seq,Zf,h] = binauralchip(u,fs,r,part,Zf,h)
%
% [seq,Zf,h] = binauralchip(u,fs,r,part,Zf,h)
%
% binauralchip  produces the output of the binaural chip as a
%               sequence of images (default: r images / sec )
%
%               u       is (a segment of) the input signal
%
%               fs      sampling rate
%
%               r       images per second
%
%               part    segment # of the input signal
%                       if part=0, then u is the whole input signal
%
%               seq     is a matrix containing values
%                       between -1 (green) and 1 (red)
%
%                       - each column of seq corresponds to an image
%                         (integration of outputs produced at every
%                         sampling step during the time window of
%                         one image)
%                       - the columns of a binaural-chip image
%                         (KxK pixels, where K is the number of
%                         filters in the cascade which models the
%                         basilar membrane) form a column of seq
%
%               Zf      is the final condition (see FILTER)
%               h       the i-th row of h is the i-th digital filter
%                       (see cascade)
%
%               See also travelingwave, cascade.
%
% J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% initializations

  ti = fs/r;                    % # pictures in 1/r seconds
  lastframe = 0;                % "frame"="image"

  if part==0
    y = travelingwave(u,fs,part,[],[]);
  elseif part==1
    [y,Zf,h] = travelingwave(u,fs,part,[],[]);
  else
    [y,Zf] = travelingwave(u,fs,part,Zf,h);
  end
  [filters,t] = size(y);
  t = t/fs;                     % length of sound sequence [s]

  numbframes = floor(t*r);

  threshold    = 0.05;
```

```
% function body

  if threshold==0
    s = sign(y);
    clear y                  % to avoid swapping for large y-matrices
  else
    s = sign(y);
    A = abs(y)-threshold;
    clear y                  % to avoid swapping for large y-matrices
    s = sign( (A>0).*A ).*s;
    clear A                  % to avoid swapping for large A-matrices
  end                        %
                             % the following initialization comes
                             % here to avoid swapping
  seq = zeros(filters^2,numbframes);
                                % each column of seq contains an
                                % image of the binaural-chip output
                                % sequence

  for j=1:numbframes
    frame = ti*j;
    fig = s(:,(lastframe+1):frame)*s(:,(lastframe+1):frame)';
    lastframe = frame;
    seq(:,j) = fig(:);
  end

% end of function "binauralchip"
```

travelingwave.m

```
function [y,Zf,h] = travelingwave(u,fs,part,Zi,h)
%
% [y,Zf,h] = travelingwave(u,fs,part,Zi,h)
%
% travelingwave  stores the traveling wave along the basilar
%                membrane in matrix y
%
%                Row y_i contains the filter output for filter i,
%                where i=1 is the low-pass filter with the highest
%                pole frequency.
%
%                u      is (a segment of) the input signal
%
%                fs     is the sampling frequency
%
%                part   is the segment # of the input signal
%                       if part=0, then u is the whole input signal
%
%                Zf     is the final condition   (see FILTER)
%                Zi     is the initial condition
%
%                h      the i-th row of h is the i-th digital filter
%                       (see cascade)
%
%                See also cascade, FILTER.
%
% J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% constants

  filters = 128;                % for comments see "cascade.m"
  fu = 100;                     %
  fo = 3750;                    %
  q  = 1.0;                     % quality factor

% initializations

  lengthu = length(u);
  y = zeros(filters,lengthu);
  input = u';

  if part==0
    h = cascade(fu,fo,filters,q,fs);
  elseif part==1
    h = cascade(fu,fo,filters,q,fs);
    Zf = zeros(2,filters);
  end

% function body

  for i=1:filters
    if part==0
      y(i,:) = filter(h(filters+1-i,1:3),h(filters+1-i,4:6),input);
    elseif part==1
      [y(i,:),Zf(:,i)] = ...
```

```
        filter(h(filters+1-i,1:3),h(filters+1-i,4:6),input);
    else
      [y(i,:),Zf(:,i)] = ...
        filter(h(filters+1-i,1:3),h(filters+1-i,4:6),input,Zi(:,i));
    end
    input = y(i,:);
  end

% end of function "travelingwave"
```

writebcsequence.m

```
function writebcsequence(A,filename,maxabs)
%
% writebcsequence(A,'filename',maxabs)
%
% writebcsequence       writes the sequence of binaural-chip output
%                       images stored in matrix A on "filename"
%
%                       - each column of A corresponds to an image
%                       - the columns of a binaural-chip output
%                         image form a column of A
%
%                       maxabs  corresponds to the # samples in
%                               the integration time interval
%
%                       See also binauralchip.
%
% J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% initializations

  FID = fopen(filename,'a');

% function body

  vals = round((A/maxabs+1)*127.5)-128; % -1 <= A/maxabs <= 1
  fwrite(FID,vals,'int8');      % binary values stored as 8-bit
                                % integers

  fclose(FID);

% end of function "writebcsequence"
```

cascade.m

```
function hd = cascade(fu,fo,filters,q,fs)
%
% hd = cascade(fu,fo,filters,q,fs)
%
% cascade       generates a filter bank of second-order filters which
%               are equally spaced in log scale.
%
%               fu determines the lower frequency [Hz]
%               fo determines the upper frequency [Hz]
%
%               filters is the number of equally-spaced filters
%                       within fu and fo
%
%               q  is the quality factor (the same for all filters)
%
%               fs is the sampling frequency of the digital system
%
%               hd is a matrix whose rows y_i = [num_i den_i] contain
%                  the numerator and denominator of the i-th digital
%                  filter
%                  numerator:   [b1 b2 b3], b1+b2*z^(-1)+b3*z^(-2)
%                  denominator: [a1 a2 a3], a1+a2*z^(-1)+a3*z^(-2)
%
%               See also BILINEAR.
%
% J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% initializations

  Ts = 1/fs;               % fs: sampling frequency

  numd = zeros(filters,3); % numd: numerator of the digital filters
  dend = zeros(filters,3); % dend: denominator of the digital filters
                           % numd & dend are both matrices:
                           % #rows    = #filters (one row per filter)
                           % #columns = filter order = 3

  logdistance = ( log10(fo) - log10(fu) )/filters;

% function body

  fi = 10.^( [0:filters]*logdistance + log10(fu)*ones(1,filters+1) );

% fi is a row vector of size filters+1 containing all points in the
%    f-axis [Hz] which delimit the in log scale equally-spaced
%    regions (number of regions = filters = K)
%
% T(s) = 1/( s^2/op^2 + 1/(op*qp)*s + 1 )       ; qp = quality factor
%                                                 op = pole frequency
%
% The second-order low-pass filters have their pole frequencies at
% the points fi (s-plain).
% Note: we take only K filters at the first K points specified in fi

  op = 2*pi*fi(1:filters)';     % op stands for omega_p
```

```
  TPnum = [0 0 1];              % all filters in the s-domain have
                                % the same numerator
  TPden = [1./op.^2 1./(op*q) ones(size(op),1)];

% Now, let's apply the bilinear transformation:
%   - same frequency response as in the s-plain (distortion only in
%     the frequency axis)

  od = 2/Ts*tan(Ts/2*op);       % omega_d is the vector with the
                                %         distorted pole frequencies
                                %         for the bilinear transform

  TPdend= [1./od.^2 1./(od*q) ones(size(od),1)];

  for i=1:filters
    [numd(i,:),dend(i,:)] = bilinear(TPnum,TPdend(i,:),fs);
  end

  hd=[numd dend];

% end of function "cascade"
```

A.1.2 Transformation to GIF images

The second part of the binaural-chip simulator transforms the output data of the model into GIF images. This is performed by the four MATLAB files listed in Table A.2. Their code is given in the following pages.

`bin2gif.m`	**main program** This file prepares the data for `fram2gif.m`.
`readpbcseq.m`	reads the coded output data of the binaural-chip model
`fram2gif.m`	**converts the coded images to GIF files** calls `num2dig.m` and uses the function `GIFWRITE` from the image-processing toolbox for the conversion to GIF files
`num2dig.m`	formatting function for file names

Table A.2: MATLAB *files to convert the binaural-chip output to* GIF *images*

bin2gif.m

```
%
%               Matlab script to generate GIF files from the
%       output files in binary format produced by the model of the
%
%                       B I N A U R A L   C H I P
%
%               audio base: Uvacek's alarm signals
%                           (57 bell, 63 horn,
%                            55 phone, and 30 ring signals)
%
%       This main program produces GIF files from the binary files
%       which contain the output images of the binaural chip.
%
%       See also readpbcseq, fram2gif, prodbin.
%
%       J.A. Osuna, 16.12.1994

%-----------------------------------------------------------------------

% constants

  maxpics = 2;

  dirbin = '/disks/isibee17/osuna/cnn/matlab/bin/';
  dirgif = '/disks/isibee17/osuna/cnn/matlab/gif/';

%-----------------------------------------------------------------------

% main program

  for i=1:30
    for k=1:4

      if k == 1
        class = 'horn';
        name = 'h';
      elseif k == 2
        class = 'bell';
        name = 'b';
      elseif k == 3
        class = 'phone';
        name = 'p';
      else
        class = 'ring';
        name = 'r';
      end

      % first read the number of frames stored in the file
      columns= 0;
      seqnormtmp=1;
      FID = fopen([dirbin class '/' class num2dig(i,2) '.bin']);
      while seqnormtmp ~= []
        seqnormtmp = readpbcseq(FID,maxpics);
        [seqrows,seqcolumns] = size(seqnormtmp);
        columns = columns+seqcolumns;
      end
      fclose(FID);
```

```
      count=0;
      seqnorm=1;
      FID = fopen([dirbin class '/' class num2dig(i,2) '.bin']);
      while seqnorm ~= []
        seqnorm = readpbcseq(FID,maxpics);
        [seqrows,seqcolumns] = size(seqnorm);
        fram2gif(seqnorm,[dirgif class '/' name num2dig(i,2)],...
                 (1:seqcolumns)+count*maxpics,ceil(log10(columns+1)));
        count = count + 1;
      end
      fclose(FID);

    end
  end

% end of program "bin2gif"
```

readpbcseq.m

```
function A = readpbcseq(FID,numbpics)
%
% A = readpbcseq(FID,numbpics)
%
% readpbcseq    reads numbpics images of the binaural-chip
%               output from the file given by FID
%
%               See also bin2gif, FOPEN.
%
% J.A. Osuna, 27.12.1994

%-------------------------------------------------------------------------

% initializations

  filters = 128;
  bcpixels = filters^2;

%-------------------------------------------------------------------------

% function body

  A = fread(FID, [bcpixels numbpics], 'int8');
                            % binary values stored as 8-bit integers

% end of function "readpbcseq"
```

fram2gif.m

```
function [numbfiles,numbframes] = ...
                        fram2gif(seqnorm,filename,frames,digs)
%
% [numbfiles,numbframes] = fram2gif(seqnorm,filename,frames,digs)
%
% fram2gif      stores all frames of sequence "seqnorm" listed in
%               vector "frames" into separate GIF files
%
%               - "seqnorm" is the sequence produced by function
%                 'binauralchip'
%                 (normalized values -128 <= seqnorm <= 127)
%               - the frames are stored each in a different file
%                 named 'filename-number.gif', where "number" is an
%                 element of vector "frames" (expanded to "digs"
%                 digits)
%
%               numbfiles  is the number of stored files, i.e., the
%                          length of vector "frames"
%               numbframes is the number of frames contained in
%                          the sequence "seqnorm"
%
%               uses the function GIFWRITE from the
%               image-processing toolbox
%
%               See also bin2gif, binauralchip, num2dig, GIFWRITE.
%
% J.A. Osuna, 16.12.1994

%-------------------------------------------------------------------

% constants

  G = ((0:255)/255)';           % Green colour (for colour map)
  R = flipud(G);                % Red
  B = zeros(256,1);             % Blue

% initializations

  [bcpixels,numbframes] = size(seqnorm);

  filters = sqrt(bcpixels);

  numbfiles = 0;

  mycolmap = [R G B];

% function body

  seq = 256-(seqnorm+128);

  for i=frames
    frame = reshape(seq(:,numbfiles+1), filters, filters);

    frame = fliplr(frame);      % flipping for correct representation
    frame = flipud(frame);      % within the simulator of CNNs

    filenamegif = [filename '-' num2dig(i,digs) '.gif'];
```

```
    gifwrite(frame,mycolmap,filenamegif) % function from the
                                         % image-processing toolbox

    numbfiles = numbfiles + 1;
  end

% end of function "fram2gif"
```

num2dig.m

```
function strg = num2dig(integer,digits)
%
% 'strg' = num2dig(integer,digits)
%
% num2dig       converts an "integer" to a string (strg) filling
%               the number with zeros to the left to a total
%               of "digits" decimal digits
%
% Example:
%
%  strg = num2dig(15,3);        % produces the string '015'
%
%
% J.A. Osuna, 23.4.1993, Budapest

%-----------------------------------------------------------------------

% initializations

  strint = num2str(integer);
  lengthint = length(strint);           % #digits of "integer"

  zeros = '';

% function body

  for i=1:(digits-lengthint)
    zeros = [zeros '0'];
  end

  strg = [zeros strint];

% end of function "num2dig"
```

A.2 Simulation of CNNs on a Multi-Signal-Processor System

CNNs consist of repeated cells which work in parallel and which, together with the property of local connectivity (see Section 1.1), make them well suited for integration. Since a fully-programmable CNN chip is not yet available, the CNNs must be simulated on digital computers. The implementation of the CNN architecture on a *parallel* computer exploits the parallel signal-processing capability of CNNs. We use a Single Process Multiple Data (SPMD) architecture with distributed memory called MUSIC (MUlti Signal processor system with Intelligent Communication) [54].[2] The numerical integration of the differential equations (1.3) (page 7) which describe the dynamical behaviour of the CNN can be processed by the MUSIC to provide a powerful CNN simulator which allows real-time signal processing.

A.2.1 The MUSIC

The parallel computer MUSIC has a modular architecture (see Figure A.1). A single Processor Element (PE) consists of a DSP 96002 from Motorola (60 MFlops), program and data memory and a fast, independent communication interface; all communication interfaces are connected through a communication ring. One node with 3 PEs and the node manager fit on a double-height Eurocard (8.6×9.2 inch). The system can be extended modularly, e.g., 20 boards (nodes) can be placed in a 19 inch rack resulting in a system with 60 PEs, a peak performance of 3.6 GFlops, and an electrical power consumption of less than 800 Watts (including forced air cooling) [55].

To estimate the performance of MUSIC, the following equation can be given [56] to compute the speed-up factor $s(n)$ of an n-processor system, i.e., for n PEs:

$$s(n) = \frac{t_e + t_o}{t_e + \frac{t_o}{n} + t_c + n \cdot t_{su}} \tag{A.1}$$

[2] The MUSIC has been developed at the Electronics Laboratory (IfE) of the Swiss Federal Institute of Technology, ETH, in Zurich.

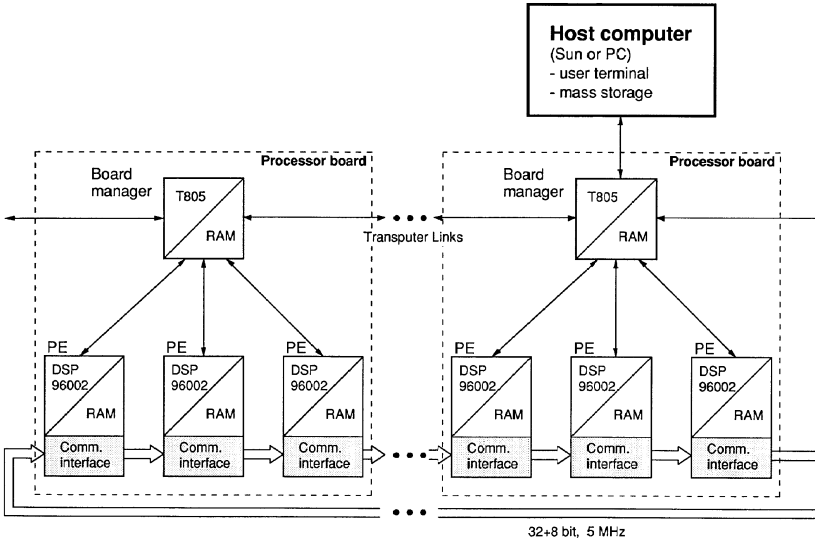


Figure A.1: *Architecture of the MUSIC*

t_e	:	time needed to process the program overhead
t_o	:	computation time required *without* parallel processing
t_c	:	communication time for the produced data
t_{su}	:	sequential setup time for the communication of a data block

Assuming $t_e + t_o \approx t_o$, equation (A.1) can be modified as follows:

$$s(n) = \frac{1}{n \cdot d + \frac{1}{n} + \frac{1}{c} + e} \tag{A.2}$$

$d = t_{su}/t_o$: ratio of the sequential setup time to the computing time using only one PE
$c = t_o/t_c$: complexity factor
$e = t_e/t_o$: serial factor

The function $s(n)$ given in (A.2) is shown in Figure A.2 for typical constant values d, e and c.[3] As can be seen from Figure 1.2, the speed-up reaches its

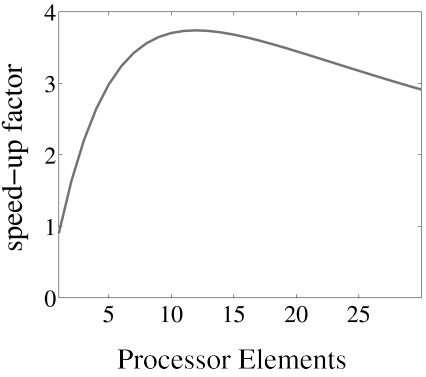


Figure A.2: *Speed-up factor given in (A.2) with* $d = 7 \cdot 10^{-3}$, $e = 10^{-5}$ *and* $c = 10$

maximum $s_{\max}$ at a certain number n_o of PEs. The results are given in the next equations:

$$n_o = 1/\sqrt{d} \tag{A.3a}$$

$$s_{\max} = s(\frac{1}{\sqrt{d}}) = \frac{1}{2\sqrt{d} + \frac{1}{c} + e} \tag{A.3b}$$

A.2.2 CNNs on MUSIC

The MUSIC is a general purpose parallel computer and was not designed to implement the CNN architecture in an optimal way as, e.g., in terms of

[3] The values of d, e and c have to be estimated from the program code which implements the specific application on MUSIC.

speed. Nevertheless, it has been shown from simulations that for larger CNNs, i.e. for $M,N \gg 1$, the parallel processing works in an ideal manner: the speed-up factor equals the number of processors, or, mathematically, $s(n) \approx n$. Thus, for $M,N \gg 1$, the constants d, e and c in (A.2) take on the following values: $d,e \approx 0$, $c \to \infty$.

The partitioning of the $M \times N$ CNN in n regions of (approximately) the same size occurs along the longer side. Without loss of generality, we can suppose that $M \geq N$. Then, a single PE has to integrate $\frac{M}{n} \cdot N$ nonlinear differential equations and needs to communicate only the first and the last row[4] of cells with its upper and lower neighbours, respectively. Moreover, the communication of these $2 \cdot N$ output values is managed by the communication interface of the PE independently of the DSP. Thus, the data can be interchanged during the computation of the remaining cell outputs in the inner part of the assigned region.

A.2.3 Iteration speed

The numerical integration of a 512×512 CNN with non-zero feedback and control templates[5] using Eulers' method with step length $h = 0.01$ [57, p. 1063] takes 25.9 seconds[6] on a 31-processor system[7] for the integration interval of $0 \ldots 10$.[8] This translates to an iteration time of $0.1\mu s/\text{cell}/\text{iteration}$.

A.2.4 CNNs in the NeuroBasic simulation environment

It is not the purpose of this section to give an introduction to the NeuroBasic simulation environment. Instead, we will reproduce on the next page the announcement of the simulator for CNNs which we developed in collaboration with the Electronics Laboratory (IfE) [58], and which we made public as free software on 7 December 1994. The simulator, together with its manual,

[4] For a neighbourhood radius $r = 1$

[5] Edge extraction, see Table 6.3 on page 88

[6] With *C*-program code

[7] Ten processor boards, each containing three PEs, plus one video board for additional on-line representation of the computed data on a monitor. With the on-line representation switched on and displaying the resulting 512×512 picture after each iteration, the integration time for the same task increases to 89.8 seconds, i.e., for 512×512 pictures, on-line displaying is possible at a rate of 11 images$/s$.

[8] The step length h and the integration interval are normalized time values.

useful utilities and several example programs, can be fetched by anonymous ftp from the server `ife.ethz.ch` at IfE.

```
Simulator of Cellular Neural Networks   Announcement    Dec 7, 1994    Os
-------------------------------------

You can retrieve from

        anonymous ftp://ife.ethz.ch/pub/NeuroBasic

free copies of our simulator for Cellular Neural Networks (CNNs). In
subdirectory cnn/ you will find versions for the following platforms:
        sun/    Sun-4 workstations
        hp/     HP7000 series workstations
        linux/  Linux PCs
        msdos/  MS-DOS PCs

A manual in PostScript format which describes the use and the
functionality of the simulator is included in every subdirectory. You
will also find several useful readme files and many example programs.


Features of CNN-simulator
-------------------------

Originally, the program was written for the parallel supercomputer MUSIC.
But the simulator also runs with *full* functionality on the platforms
given above. The simulator is  u n i v e r s a l  in the following sense:
        - simulation of multilayer CNNs
        - definition of your own nonlinear templates
        - delayed templates
        - you can choose from several numerical-integration algorithms

The simulator can be controlled by an easy-to-use Basic syntax
interactively from a shell, or from a loadable program. You have several
display options on multitasking operating systems:
        - animation of image sequences with the program animate from the
          ImageMagick package
        - display of images at the input/state/output of the CNN for all
          layers with the program xv

Within the same simulation environment which we call NeuroBasic V2.0, you
may make use of functions to simulate (conventional) neural networks:
Perceptrons and Convolutional Nets. The subdirectory all/ contains all
available packages of the NeuroBasic environment.

There are no memory limitations other than the physical ones of your
computer. The code was written in ANSI-C and compiled with the GNU-C
compiler.


For further information or problem reports, please contact

J.A. Osuna
ISI                     e-mail: osuna@isi.ee.ethz.ch
ETH-Zentrum             phone:  +41 1 63 23620
CH-8092 Zurich          fax:    +41 1 63 21208
```

Bibliography

[1] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 1257–1272, Oct. 1988.

[2] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 1273–1290, Oct. 1988.

[3] K. Preston, Jr. and M. J. B. Duff, *Modern Cellular Automata — Theory and Applications*. Advanced applications in pattern recognition, New York: Plenum Press, 1984.

[4] J. M. Cruz and L. O. Chua, "A CNN Chip for Connected Component Detection," *IEEE Transactions on Circuits and Systems*, vol. 38, pp. 812–817, July 1991.

[5] D. Lím and G. S. Moschytz, "A Programmable, Modular CNN Cell," in *Third IEEE International Workshop on Cellular Neural Networks and their Applications*, (Rome), pp. 79–84, Dec. 1994.

[6] E. A. Vittoz, "Analog VLSI Signal Processing: Why, Where and How?," *Journal of VLSI Signal Processing*, vol. 8, pp. 27–44, July 1994.

[7] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, vol. 4, pp. 4–22, Apr. 1987.

[8] D. R. Hush and B. G. Horne, "Progress in Supervised Neural Networks," *IEEE SP Magazine*, pp. 8–39, Jan. 1993.

[9] H. Harrer and J. A. Nossek, "Discrete-Time Cellular Neural Networks," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 453–467, September-October 1992.

[10] P. L. Venetianer, "CNN Analogic (Dual) Software Library, Version 4.1," Report DNS-1-1993, Dual & Neural Computing Systems Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Jan. 1993.

[11] J. A. Osuna and G. S. Moschytz, "CNN Templates for Image Processing Tasks," Report 9206, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, ETH, Zurich, Nov. 1992.

[12] T. Roska and L. O. Chua, "Cellular Neural Networks with Non-Linear and Delay-Type Template Elements and Non-Uniform Grids," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 469–481, September-October 1992.

[13] L. O. Chua and T. Roska, "CNN Universal Machine and Supercomputer." Patent Disclosure, July 1992.

[14] T. Roska and L. O. Chua, "The CNN Universal Machine: An Analogic Array Computer," *IEEE Transactions on Circuits and Systems–II*, vol. 40, pp. 163–173, Mar. 1993.

[15] G. Seiler and M. Hasler, "Convergence of Reciprocal Cellular Neural Networks," Report TUM-LNS-TR-91-12, Institute for Network Theory and Circuit Design, Technical University Munich, June 1991.

[16] F. Zou and J. A. Nossek, "Design method for Cellular Neural Network with linear relaxation," in *IEEE International Symposium On Circuits And Systems*, (Singapore), pp. 1323–1326, June 1991.

[17] L. O. Chua and P. Thiran, "An Analytic Method for Designing Simple Cellular Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. 38, pp. 1332–1341, Nov. 1991.

[18] A. Gees and T. Putzi, "Cellular Neural Networks for Signal Processing," Semester project 9215, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, ETH, Zurich, July 1992. (in German).

[19] B. Mirzai and G. S. Moschytz, "Design of Robust CNN Templates." in preparation.

[20] T. Matsumoto, L. O. Chua, and H. Suzuki, "CNN Cloning Template: Shadow Detector," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 1070–1073, Aug. 1990.

[21] L. O. Chua and T. Roska, "Stability of a Class of Nonreciprocal Cellular Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 1520–1527, Dec. 1990.

[22] T. Matsumoto, L. O. Chua, and R. Furukawa, "CNN Cloning Template: Hole-Filler," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 635–638, May 1990.

[23] T. Matsumoto, L. O. Chua, and H. Suzuki, "CNN Cloning Template: Connected Component Detector," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 633–635, May 1990.

[24] J. A. Osuna and G. S. Moschytz, "Exact Design of Reciprocal Cellular Neural Networks," in *IEEE International Workshop on Cellular Neural Networks and their Applications*, (Munich), pp. 11–16, Oct. 1992.

[25] G. M. Georgiou, "Comments on 'On Hidden Nodes in Neural Nets'," *IEEE Transactions on Circuits and Systems*, vol. 38, p. 1410, Nov. 1991.

[26] L. Schläfli, *Collected Mathematical Treaties I*, pp. 209–212. Basel, Switzerland: Verlag Birkhauser, 1950. (in German).

[27] L. Schläfli, "Theorie der vielfachen Kontinuität (Theory of Multidimensional Spaces)," in *Neue Denkschriften der allgemeinen schweizerischen Gesellschaft für die gesammten Naturwissenschaften* (J. H. Graf, ed.), vol. 38, (Zurich, Switzerland), pp. 1–237, Denkschriften-Kommission der Schweizerischen Naturforschenden Gesellschaft, Zürcher & Furrer, 1901. (in German).

[28] T. M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transactions on Electronic Computers*, vol. 14, pp. 326–334, June 1965.

[29] L. O. Chua and T. Roska, "The CNN Paradigm," *IEEE Transactions on Circuits and Systems–I*, vol. 40, pp. 147–156, Mar. 1993.

[30] B. Uváček, *Electro-Tactile Recognition of Preclassified Alarm Signals for Profoundly Deaf*. PhD thesis, Swiss Federal Institute of Technology, ETH, No. 8649, Zurich, 1988. (in German).

[31] J.-F. Leber, *The Recognition of Acoustical Signals Using Neural Networks and an Open Simulator*. PhD thesis, Swiss Federal Institute of Technology, ETH, No. 10016, Zurich, 1992.

[32] C. A. Mead, X. Arreguit, and J. Lazzaro, "Analog VLSI Model of Binaural Hearing," *IEEE Transactions on Neural Networks*, vol. 2, pp. 230–236, Mar. 1991.

[33] R. F. Lyon and C. Mead, "An Analog Electronic Cochlea," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1119–1134, July 1988.

[34] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*. Signal Processing Series, Englewood Cliffs, New Jersey: Prentice Hall, 1983.

[35] J. A. Osuna, G. S. Moschytz, and T. Roska, "A Framework for the Classification of Auditory Signals with Cellular Neural Networks," in *European Conference on Circuit Theory and Design*, vol. 1, (Davos, Switzerland), pp. 51–56, Elsevier, 30 Aug.– 3 Sept. 1993.

[36] G. S. Moschytz and P. Horn, *Active Filter Design Handbook — For Use with Programmable Pocket Calculators and Minicomputers*. John Wiley & Sons, 1981.

[37] R. Unbehauen and A. Cichocki, *MOS Switched-Capacitor and Continuous-Time Integrated Circuits and Systems*. Communications and Control Engineering Series, Berlin: Springer-Verlag, 1989.

[38] D. Muster, "Modelling Acoustical Alarm Signals by Prediction," Postdiploma thesis 94.1, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, ETH, Zurich, Spring 1994. (in German).

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, vol. 1 of *Parallel distributed processing*, ch. 8, pp. 318–362. Cambridge: MA: MIT Press, 1986.

[40] S. Haykin, *Neural Networks — A Comprehensive Foundation*. New York: Macmillan College Publishing Company, 1994.

[41] T. Roska and L. Kék, "Analogic CNN Program Library, Version 6.1," Report DNS-5-1994, Analogical and Neural Computing Laboratory, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Dec. 1994.

[42] J. Bernold and M. Hänggi, "RATSCELL, Recognition of AcousTical alarm Signals using CELLular neural networks," Semester project 9417, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, ETH, Zurich, Summer 1994. (in German).

[43] T. Roska, T. Boros, A. Radványi, P. Thiran, and L. O. Chua, "Detecting Moving and Standing Objects Using Cellular Neural Networks," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 613–628, September-October 1992.

[44] G. H. Golub and C. F. van Loan, *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences, Baltimore and London: Johns Hopkins, second ed., 1989.

[45] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley Computation and Neural Systems Series, Addison-Wesley, 1989.

[46] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[47] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 IRE WESCON Convention Record*, (New York, NY), pp. 96–104, 1960.

[48] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York, NY: Wiley, 1973.

[49] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.

[50] U. A. Müller, A. Gunzinger, and W. Guggenbühl, "Fast Neural Net Simulation with a DSP Processor Array," *IEEE Transactions on Neural Networks*, vol. 6, pp. 203–213, Jan. 1995.

[51] S. Oberle and A. Kaelin, "Recognition of Acoustical Alarm Signals for the Profoundly Deaf Using Hidden Markov Models," in *IEEE International Symposium On Circuits And Systems*, vol. 3, (Seattle, Washington), pp. 2285–2288, 1995.

[52] A. Kellner, H. Magnussen, and J. A. Nossek, "Texture Classification, Texture Segmentation and Text Segmentation with Discrete-Time Cellular Neural Networks," in *Third IEEE International Workshop on Cellular Neural Networks and their Applications*, (Rome), pp. 243–248, Dec. 1994.

[53] The MathWorks, Inc. e-mail: `info@mathworks.com`.

[54] A. Gunzinger, U. Müller, W. Scott, B. Bäumle, P. Kohler, and W. Guggenbühl, "Architecture and Realization of a Multi Signalprocessor System," in *Proc. Int. Conference on Application-Specific Array Processors*, (Berkeley, California), Aug. 1992.

[55] U. A. Müller, M. Kocheisen, and A. Gunzinger, "High Performance Neural Net Simulation on a Multiprocessor System with "Intelligent" Communication," in *Advances in Neural Information Processing Systems (NIPS-6)* (J. Coowan, G. Tesauro, and J. Alspector, eds.), Morgan Kaufmann Publishers, 1994.

[56] U. A. Müller, *Simulation of Neural Networks on Parallel Computers*. PhD thesis, Swiss Federal Institute of Technology, ETH, No. 10188, Zurich, 1993.

[57] E. Kreyszig, *Advanced Engineering Mathematics*. New York: John Wiley & Sons, sixth ed., 1988.

[58] I. Hasler and B. Tiemann, "Simulation of a Cellular Neural Network Universal Machine on MUSIC," Semester project 9309, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, ETH, Zurich, July 1993. (in German).

Index

Curriculum Vitae

José A. Osuna
Born on 6 May 1965 in Winterthur, Switzerland

1972–1980	Attended elementary and intermediate schools in Winterthur
1980–1984	Attended secondary school (Gymnasium) in Winterthur; graduated in September 1984 with a mathematics/science diploma (Maturitätsprüfung Typus C), and the Spanish secondary school diploma Bachiller Unificado Polivalente (B.U.P.); graduated in May 1985 with the Spanish secondary school diploma Curso de Orientación Universitaria (C.O.U.)
1984–1989	Attended the Swiss Federal Institute of Technology (ETH) in Zurich; received the equivalent of an MS degree (Diplom) in Electrical Engineering in February 1989
1989–1995	Employed as Research Assistant (wissenschaftlicher Mitarbeiter) at the Signal and Information Processing Laboratory (Institut für Signal- und Informationsverarbeitung), Swiss Federal Institute of Technology (ETH), Zurich; from March 1989 to April 1990 mainly envolved in educational tasks at the laboratory