RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Dominik Niedenzu

Aufbau eines binären Neocognitrons

Diplomarbeit

HD-KIP-03-11

KIRCHHOFF-INSTITUT FÜR PHYSIK

Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Diplomarbeit
im Studiengang Physik

vorgelegt von
Dominik Niedenzu
aus Saarbrücken
2003

Aufbau eines binären Neocognitrons

Die Diplomarbeit wurde von Dominik Niedenzu ausgeführt am
Kirchhoff-Institut für Physik
unter der Betreuung von
Herrn Prof. Dr. Karlheinz Meier

Aufbau eines binären Neocognitrons

Zielsetzung dieser Arbeit ist der Aufbau eines binären, auf den neuronalen Netzwerkchip HAGEN abgestimmten Neocognitrons zur Bildverarbeitung. Das Neocognitron ist eine 1988 von Kunihiko Fukushima vorgestellte hierarchische Netzwerktopologie. Die herausragende Eigenschaft dieses Netzes ist die schon in der Topologie verankerte Aufspaltung der Bilderkennung in voneinander unabhängige Teilaufgaben. Der parallele Einsatz mehrerer HAGEN-Chips ermöglicht somit die schnelle Bildverarbeitung auch großer Eingangsbilder. Das Neocognitron in seiner ursprünglichen Fassung ist nicht binär, entsprechende Modifikationen werden in vorliegender Arbeit ausgeführt und getestet. Ein Schwerpunkt liegt dabei auf der Erhaltung der datentypunabhängigen Funktionsweise des Originals. Zentraler Punkt ist die Entwicklung eines automatischen Trainings. Dieses beinhaltet sowohl eine Netzstrukturoptimierung des Neocognitrons als auch das Training des HAGEN-Chips. Da es bisher keine vergleichbaren Arbeiten gibt, muss vor allem die Frage geklärt werden, inwieweit die Abbildung der Neocognitronstruktur auf HAGEN möglich ist. Experimente zur Erkennung handgeschriebener Ziffern werden vorgestellt.

Setting up a binary Neocognitron

The presented thesis describes the development of a binary version of Fukushimas Neocognitron on the basis of the neural network chip HAGEN for image processing. The Neocognitron is a hierarchicaly organized topology introduced by Kunihiko Fukushima in 1988. The outstanding property of this network, already established in the topology, is that it divides the recognition process into functional parts working independently from each other. By letting several HAGEN chips run simultaneously, it enables fast image processing even with large inputs. The original version of the Neocognitron is not binary, corresponding modifications are being done and tested within this thesis. Hereby one main aspect is to preserve the independence of the functions of the Neocognitron from data types. The central point is the development of an automatic training, which contains maximising the efficiency of the network structure of the Neocognitron as well as the training of the HAGEN chip. Since there has not been any comparable previous work, the question must be answered, to what extent this mapping of the Neocognitron structure onto HAGEN is possible. First tests using handwritten digits are being carried out.

Inhaltsverzeichnis

1 Einleitung und Motivation **1**

2 Grundlagen **4**
2.1 Neuronale Netze 4
2.2 Die verwendete Hardware 6
2.2.1 Historischer Abriß 6
2.2.2 Der neuronale Netzwerk-Chip HAGEN 9
2.2.3 Darkwing, Die Verbindung zur Computerwelt 10
2.2.4 Ansteuerung via HANNEE 11
2.3 Gängige Netztopologien - Verschaltungsoptionen für die Hardware . 12
2.3.1 Der Musterassoziator 13
2.3.2 Der Autoassoziator 14
2.3.3 Das Wettbewerbsnetz 15
2.3.4 $\sigma\pi$-Netze 15
2.3.5 Das ART1 16
2.3.6 Das Neocognitron 17
2.4 Schrifterkennung 18
2.4.1 Vektor-, Punkt- und Objektgrafiken 19
2.4.2 Spezielle Probleme der Schrifterkennung 20
2.4.3 Gängige Lösungsansätze im Vergleich 20
2.4.4 Kenngrößen einer Lösung 22
2.4.5 Pro und Contra Neocognitron 22

3 Das Neocognitron **24**
3.1 Das Original von Kunihiko Fukushima 24
3.1.1 Fukushimas Idee 24
3.1.2 Die Grobstruktur 24
3.1.3 Die Ebenen im Einzelnen 27
3.1.3.1 S-Ebene 27
3.1.3.2 C-Ebene 28
3.1.3.3 V-Ebene 28
3.1.4 Das Training 31
3.1.4.1 Unüberwachtes Lernen 31

3.1.4.2 Überwachtes Lernen 31
3.2 Die Abbildung auf ein hybrides Netz wie HAGEN 32
3.2.1 F/S-Ebene 35
3.2.2 C-Ebene 36
3.2.3 L-Schicht 37

4 Konstruktion des binären Neocognitrons 38
4.1 Aufbau der Grundstruktur 38
4.1.1 Die Entwicklungsumgebung 38
4.1.2 Erste Abschätzung der Auflösung nach der C-Ebene 40
4.2 Training mittels Feature Selektoren 42
4.2.1 Notwendigkeit der Automatisierung 43
4.2.2 Der Verteilungsselektor 43
4.2.2.1 Der Algorithmus 44
4.2.2.2 Abschätzung der Tauglichkeit 45
4.2.3 Der Differenzselektor 46
4.2.3.1 Der Algorithmus 48
4.2.3.2 Abschätzung der Tauglichkeit 48
4.2.4 Der kombinierte Selektor 50
4.2.4.1 Abschätzung der Tauglichkeit 50
4.2.5 Funktionstests mit vereinfachten S-Ebenen 50
4.2.5.1 Hopfieldebenen 50
4.2.5.2 Vergleichsebenen 52
4.3 Die automatische Netzwerkstrukturoptimierung BONNEE 53
4.3.1 Die Parameterschleife 53
4.3.2 Das Softwarepaket SHAGEN 56
4.3.2.1 Optimierung der SHAGEN-Trainingsparameter . . 63
4.3.2.2 Übertragbarkeit auf die Hardware 65
4.3.2.3 Zusammenfassung 68
4.4 Die Vernetzung der Programme 71

5 Ergebnis 74
5.1 Das beste Individuum 74
5.2 Verbesserungsmöglichkeiten 77

6 Zusammenfassung und Ausblick 81

Literaturverzeichnis 83

A Abbildungen II

B Inhalt der CD XVIII

C Fachbegriffe XXI

1 Einleitung und Motivation

Schon Aristoteles beschrieb um 350 v.Chr. Gedächtnis als Verknüpfung einfacher Elemente, die zu komplexeren Strukturen kombiniert, logischer Schlüsse fähig sind [Koh88a][Med98]. In den folgenden zwei Jahrtausenden warteten zahlreiche prominente Persönlichkeiten aus den Gebieten Philosophie, Psychologie und Medizin mit ebenso zahlreichen Hypothesen zur Erklärung menschlichen Denkvermögens auf. Zu Beginn des letzten Jahrhunderts wurden erste, einfache Modelle direkt an Mensch und Tier überprüfbar. Zu dieser Zeit machte beispielsweise der russische Physiologe Iwan Pawlow seine berühmten Konditionierungsversuche mit Hunden [Gol97]. Obwohl sich die Grundlagenforschung dieses Gebiets rasant entwickelt hatte, war man noch weit davon entfernt, höhere Funktionen des Hirns auf biologische Prozesse zurückführen zu können. In der Folge entwickelte sich eine immer größer werdende interdisziplinäre Gemeinschaft, die nicht auf die Analyse biologischer Systeme setzte. Stattdessen wurden stark abstrahierte Modellsysteme entwickelt und diskutiert. Warren McCulloch und Walter Pitts entwarfen im Jahre 1943 die ersten künstlichen neuronalen Netze [Roj96].
Die Mehrzahl neuronaler Netze besteht aus einer Menge von Knoten, die durch gewichtbare erregende bzw. hemmende Leitungen, in Anlehnung an das biologische Vorbild Synapsen genannt, untereinander verbunden sind. Die Gewichte sind Signalmultiplikatoren. Hemmende Leitungen können dabei durch negative Gewichte dargestellt werden. Die Knoten fungieren als Schalter mit zumeist einfachen Schalteigenschaften. Die einfachste Variante sind Schwellwertneuronen. Sie addieren die Signale der Zuleitungen und feuern, falls die Summe größer einem Schwellwert ist. Feuert eine Zelle, so legt sie ein einheitliches Signal an ihre Ausgänge, die ihrerseits gewichtete Zuleitungen anderer Zellen sein können. Ein künstliches neuronales Netz hat in der Regel definierte Ein- und Ausgangsneuronen. Größere Netze lassen sich kaum ohne Computerhilfe berechnen. Das wenig später entwickelte Hodgkin-Huxley-Modell [Gol97] zur Beschreibung der Dynamik von Nervenimpulsen an biologischen Nervenzellen kann als ein künstliches neuronales Netz angesehen werden.
In den sechziger Jahren wurden durch die aufkommenden Computer andere Disziplinen auf neuronale Netze aufmerksam. Ingenieure, Physiker und Vertreter anderer Wissenschaften, deren primäres Interesse nicht der Erforschung des Menschen galt, entdeckten die neuronalen Netze als Automaten, die Probleme lösen können.

Mit einfachen Algorithmen zur Gewichtsanpassung lassen sich dabei, bei gleicher Netztopologie, verschiedene Netzverhaltensweisen erreichen, die durch herkömmliche Elektronikschaltungen nur äußerst aufwendig oder gar nicht konstruiert werden können. Mathematiker wie Turing, Hilbert und von Neumann hatten sich schon in den dreißiger und vierziger Jahren ausführlich mit dem Berechenbarkeitsproblem beschäftigt und für den Computerbau ein wertvolles Fundament geschaffen [Her88]. Die somit geborene Informationstheorie erreichte auch die Theorie neuronaler Netze und nicht zuletzt Physiker wie John Hopfield [Hop82] erarbeiteten in der Folgezeit ein ansehnliches mathematisches Gerüst zur Berechnung einiger Netzeigenschaften. Hopfield gelang es beispielsweise, eine spezielle Netztopologie, einen Assoziativspeicher, durch die in der theoretischen Physik schon länger bekannte Relaxationsmethode statistisch zu beschreiben. Der Boom um die neuronalen Netze hielt allerdings nur bis in die neunziger Jahre an. Zwar gab es bereits leistungsfähige Rechnersysteme, allerdings genügten diese nicht den hohen Erwartungen an die Universalität neuronaler Netze gerecht zu werden. In den letzten Jahren gewinnt dieses Forschungsfeld, nicht nur wegen der immer noch mit erstaunlicher Geschwindigkeit wachsenden Rechenleistung neuester Chips, wieder an Bedeutung. Die stetigen Fortschritte bei der Vernetzung herkömmlicher Computersysteme unterstützen den eigentlichen Vorteil neuronaler Netze gegenüber sequentieller Automaten. Obwohl die Schaltzeiten menschlicher Nervenzellen im Bereich von Millisekunden liegen und somit deutlich langsamer sind als schnelle elektronische Bauteile (ns), ist das menschliche Gehirn dennoch in vielen Bereichen überlegen. Maßgeblichen Anteil daran hat der hohe Vernetzungsgrad und die damit verbundene enorme Parallelisierung des Informationsflusses. Obwohl verteilte Softwarelösungen im Computerverbund ein hohes Maß an Parallelisierung erreichen, sind sie stets durch die Bandbreiten der Vernetzungstechnik beschränkt. Außerdem sind für größere Netze Platz- und Stromverbrauch, sowie Hitzeentwicklung keine trivialen Probleme.
Eine junge Entwicklung in der Geschichte neuronaler Netzwerke und nicht zuletzt deshalb, aus Sicht der Grundlagenforschung, ein spannendes Aufgabenfeld, ist die Implementation neuronaler Netzwerke auf dafür speziell entwickelten Chips, sogenannten ASICs[1] (Kap. 2.2.1). Die Gruppe Electronic Vision(s), innerhalb der diese Arbeit durchgeführt wurde, beschäftigt sich unter anderem mit der Entwicklung solcher Chips. Ein Ergebnis dieser Arbeit ist HAGEN[2] (Kap. 2.2.2). HAGEN beinhaltet vier Netzwerkblöcke mit jeweils 128 Ein- und 64 Ausgängen. Jeder dieser Blöcke stellt ein vollständig vernetztes einschichtiges Perzeptron dar. Ein- wie Ausgänge sind binär, die Gewichte hingegen analog. Diese hybride Technologie vereint die Vorteile analoger Implementationen mit der Skalierbarkeit digitaler Vernetzung. So können nahezu beliebig viele HAGEN-Chips direkt vernetzt werden. Die bei analoger Vernetzung problematischen Leitungsverluste fallen weg. Die

[1] **A**pplication **S**pecific **I**ntegrated **C**ircuit.
[2] **H**eidelberg **A**nalo**g** **E**volvable **N**eural Network.

Ausgangssignale können über programmierbare Signalrückführungen an die Eingangsneuronen rückgekoppelt werden. So sind mehrschichtige Netzwerktopologien schon mit einem Chip möglich.
Zielsetzung dieser Arbeit ist der Aufbau eines binären, auf den HAGEN-Chip abgestimmten Neocognitrons für die Bildverarbeitung. Einen Überblick über die dazu entwickelte Software und vorhandene Strukturen gibt Kap. 4.4. Ein HAGEN-Block hat 128 Eingänge, größere Bilder können dem Chip deshalb nur partitioniert zugeführt werden. Das Neocognitron ist eine 1988 von Kunihiko Fukushima vorgestellte, hierarchische Netzwerktopologie [Fuk88] (Kap. 3.1). Die herausragende Eigenschaft dieses Netzes ist die schon in der Topologie verankerte Aufspaltung der Bilderkennung in voneinander unabhängige Teilaufgaben. So lassen sich sogar mehrere Neocognitrons als Teile einer übergeordneten Neocognitron-Topologie parallel betreiben. Physiologisch inspiriert, wurden alle Netzwerkkomponenten analog entworfen und sind somit nicht direkt auf HAGEN abbildbar. In Kapitel 3.2 sind die notwendigen Modifikationen dargestellt. Die verwendeten Bilddaten stammen aus einer vom LeCunn bereitgestellten Datenbank handgeschriebener Ziffern [LeC03]. Obwohl der Aufbau des Neocognitrons anhand der Handschrifterkennung überprüft wird, ist es ein wesentliches Anliegen, keine spezifischen Eigenschaften der Testdaten auszunutzen und das Prinzip somit für beliebige Bilddaten nutzbar zu machen (Kap. 2.4.5). Wegen der äußerst komplexen Struktur dieser Topologie, spielt die Entwicklung eines Algorithmus zur automatischen Suche guter Netzwerkparameter eine zentrale Rolle (Kap. 4). Dieser beinhaltet ein automatisches Training - die trainierten Features (Kap. C) bestimmen die Netzstruktur maßgeblich mit. Da es bisher keine vergleichbaren Arbeiten gibt, muß durch diese Arbeit vor allem die Frage geklärt werden, inwieweit die Abbildung der Neocognitron-Struktur auf HAGEN möglich ist. Zum Neocognitron gibt es insgesamt nur wenige Publikationen. Zhengjun Pan et al. führen dies, ebenso wie D.R.Lovell et al., auf die große Komplexität dieser Topologie und ihre Empfindlichkeit gegenüber Parametervariationen zurück [PSAB99], [LDT95].
Für die Arbeit waren umfangreiche Kenntnisse aus den Gebieten neuronale Netze, Hardwareansteuerung, Softwareentwicklung und nicht zuletzt der Schrifterkennung vonnöten. Im Rahmen dieser Arbeit kann das einleitende Kapitel 2 daher nur die zentralen Erwägungen ansprechen. Aus didaktischen Gründen steigt die Informationsdichte nicht nur mit dem Verlauf der Arbeit, auch die einzelnen Kapitel verdichten sich stetig. Die Kapitel 2.3.6, 2.4.5, 3.1 und 3.2 nähern sich der Funktionsweise des Neocognitrons aus verschiedenen Blickwinkeln und mit steigender Präzision.

Kapitel 2

Grundlagen

2.1 Neuronale Netze

Das menschliche Gehirn besteht aus etwa 10^{10} Neuronen [Hor]. Neurone arbeiten als Schalter mit komplizierten Schalteigenschaften. Ihre starke Vernetzung untereinander - im Mittel besitzt jedes Neuron 10^4 Verknüpfungen zu anderen Neuronen - macht das Gehirn zu einem komplexen biologischen Computer. Die Informationsleitung basiert auf elektrischen und chemischen Signalen. Es existieren eine Reihe verschiedenartiger Neurone. Ihnen liegt jedoch eine gemeinsame Struktur zugrunde. Ein Neuron besteht aus einem Dendritenbaum, dem Zellkörper[1] und einem oder mehreren Nervenfasern[2] (Abb. 2.1). Nervenfasern anderer Neurone docken am Dendritenbaum via Synapsen an. Der an der Synapse eintreffende elektrische Impuls wird in der Regel auf chemischem Wege an den Dendritenbaum weitergeleitet. Im Dendritenbaum sammeln und summieren sich Impulse verschiedener Zellen. Die Leitfähigkeit im Dendritenbaum variiert. Zwei an verschiedenen Stellen eintreffende Signale werden damit unterschiedlich gewichtet. Nur wenn die Summe der am Zellkörper eintreffenden Signale eine gewisse Schwelle überschreitet, kann am Axonhügel[3] ein sogenanntes Aktionspotential initiiert werden. Aktionspotentiale sind dynamische, elektrische Signale. Sie sind Mittel der Informationsübertragung im Hirn. Natürlich sind Abweichungen von der hier sehr vereinfacht dargestellten Funktion eines Neurons möglich. Es existieren beispielsweise auch Axon-Axon Verbindungen.

Da die Entwicklung künstlicher neuronaler Netze ursprünglich der Überprüfung physiologischer sowie psychologischer Thesen diente, sind die meisten künstlichen neuronalen Netze stark am biologischen Vorbild orientiert. Wesentliche Merkmale natürlicher Nerven finden sich hier wieder. Sogar die atomaren

[1] Soma.
[2] Axone.
[3] Die Stelle am Zellkörper, an dem das Axon entspringt.

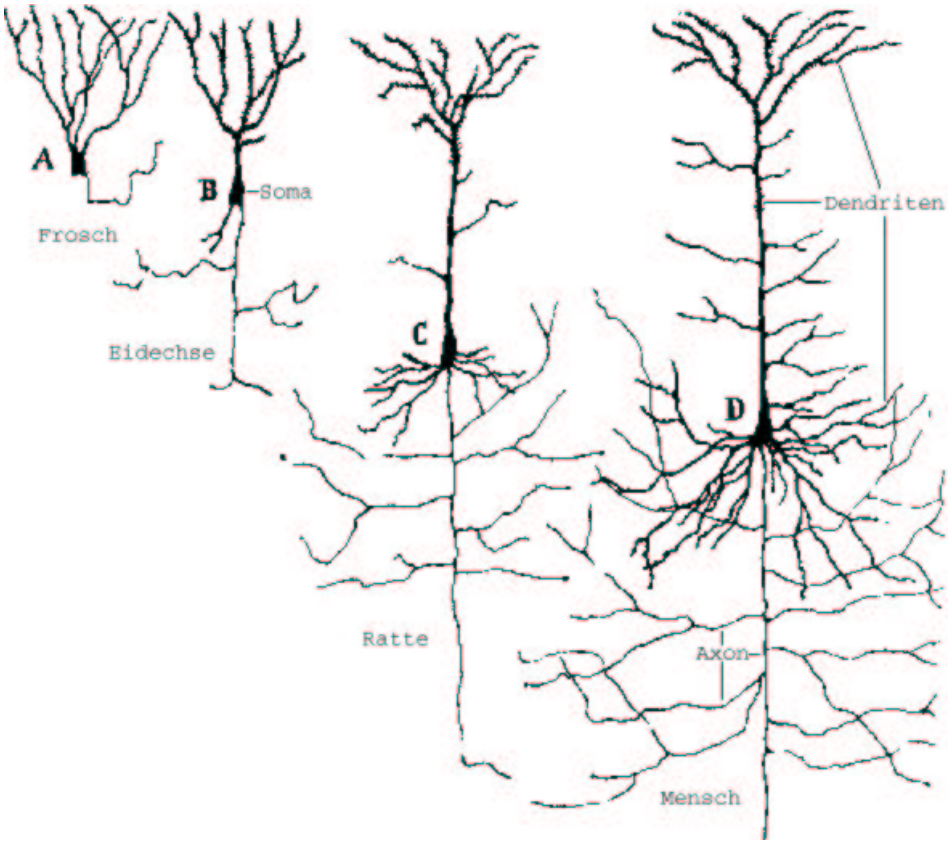

Abbildung 2.1: Pyramidenzellen aus der Großhirnrinde verschiedener Wirbeltiere, gezeichnet von Ramón y Cajal (1852-1934). Der Dendritenbaum setzt im Gegensatz zur Nervenfaser(Axon) an mehreren Stellen des Zellkörpers(Soma) an. Freundlicherweise überlassen von Arnd Roth, Max-Planck-Institut für Medizinische Forschung Heidelberg.

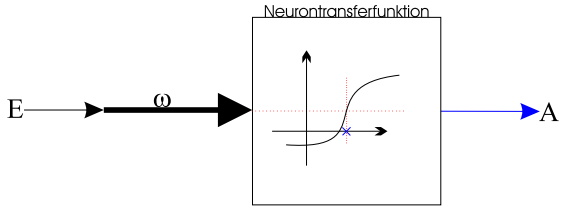

Abbildung 2.2: Schema eines künstlichen Neurons. Ein Eingangssignal E wird multipliziert mit dem Gewicht ω der Transferfunktion als Argument übergeben. Die Transferfunktion des Neurons ordnet diesem anschließend ein Ausgangssignal A zu. I.a. existieren mehrere gewichtete Eingangsleitungen, deren Signale summiert werden. Das Ausgangssignal A wird gleichermaßen auf alle Ausgänge dieses Neurons gelegt.

Einheiten erfolgsorientiert entworfener Netze haben überwiegend folgende simple Struktur: Eingangsleitungen (E) führen einem Knoten Signale zu, der diese in irgendeiner Form auswertet und gegebenenfalls ein Signal auf eine Reihe von Ausgangsleitungen (A) legt (Abb. 2.2). Dieses Signal ist im allgemeinen für alle Ausgangsleitungen des Knotens gleich. Den Eingangsleitungen hingegen ordnet man unterschiedliche Signalleitfähigkeiten in Form von Multiplikatoren, den Gewichten (ω), zu. Die Neurone werden in der Regel derart vernetzt, daß das Netz definierte Ein- und Ausgangsneurone hat. Werden die Netzgewichte so ausgewählt, daß bestimmte Eingangsmuster bestimmte Ausgangsmuster zur Folge haben, bezeichnet man diese Eingangs-/Ausgangsmusterkombinationen als gelernt. Die Suche passender Gewichte nennt man *Training*. Ein dem trainierten Eingangsmuster ähnliches Eingangsmuster bewirkt oft ein dem trainierten Ausgangsmuster ähnliches Ausgangsmuster, obwohl es dem Netz vorher nie vorgelegt wurde. Neuronale Netze lernen auf eine nicht einfach vorhersagbare Weise mehr, als ihnen beigebracht wurde. Diese Eigenschaft nennt man *Generalisierung*.Sie ist der entscheidende Vorteil gegenüber ablaufbestimmten Computeralgorithmen.

2.2 Die verwendete Hardware

2.2.1 Historischer Abriß

Die erste elektronische Realisierung eines neuronalen Netzes stellte Minsky 1954 unter dem Namen *Snark* vor [Min54]. Vier Jahre später folgte das *Perceptron Mark I* von Rosenblatt [Ros58]. Beide verwendeten motorgetriebene Potentiometer zur Implementation der Netzgewichte. In diesem Jahrzehnt wurde auch die *Memistor Corporation*, die erste Neurocomputer Firma, gegründet. Gründer war

Bernard Widrow, der Erfinder des ADALINE[4] - eines speziellen Musterassoziators [FB97],[Hof93]. Es ist kaum möglich, alle in der Folgezeit entstandenen Architekturen zu erwähnen. Ich beschränke mich daher auf Beispiele weit verbreiteter Architekturtypen.

Digitale Lösungen Eine der ersten digitalen Lösungen stellte *Mark IV* (1989) dar. Von der DARPA[5] finanziert, wurde dieses Projekt durch die TRW Inc. unter der Leitung von Hecht-Nielsen und Todd Gutschow verwirklicht [HN89]. Seine Recheneinheiten sind 16-bit Texas Instruments TMS32020 DSPs[6]. Mark IV kann 5,5 Millionen Synapsen bereitstellen. Fast ebensoviele Verbindungen können in der Sekunde ausgewertet werden. Mark IV hat eine Masse von 200kg und eine Leistungsaufnahme von 1,3kW.

• **Standard PC Lösungen** Zur Zeit des *MarkIV* waren herkömmliche Computer in der Lage, etwa 250 000 Verbindungen in der Sekunde - CPS[7] - zu simulieren. (Das Geschwindigkeitsmaß CPS lässt Größen wie die Präzision der Gewichte unberücksichtigt und ist deshalb nur ein grobes Maß für die Rechenleistung. Weil CPS, im Gegensatz zu komplizierteren Größen, für viele Systeme angegeben wird, soll diese Größe im Folgenden als Anhaltspunkt genügen.)
Die Konstruktion von PC[8]-Beschleunigerkarten bewirkte einen deutlichen Geschwindigkeitsschub. Eine solche Karte ist die *DFPP*[9] von SAIC[10] (1988) [Sou88]. Sie erreicht 10 MCPS und 1-2 MCUPS[11]. CUPS mißt die Geschwindigkeit, mit der Netzgewichte verändert werden können. Auch dies ist lediglich ein grobes Maß für die erzielbare Trainingsgeschwindigkeit.

• **Netze Paralleler Prozessoren** 1992 entwickelte Neuralogix den *NLX-420* Chip. Dieser besteht aus 16 parallel arbeitenden Prozessoren (PU[12]s), die eine Netzwerkschicht darstellen. Der einzige 16bit-Eingang ist allen diesen PUs gemeinsam. Gewichte werden extern gespeichert. Es sind interne Rückkopplungsleitungen vorhanden, außerdem können mehrere NLX-420 zu einem mehrschichtigen Netz zusammengesteckt werden. Derartige Konstruktionen nennt man Scheibenarchitekturen. Der NLX-420 bewältigt 300MCPS [Neu92].
Da die Simulation der meisten Neurontypen mit wenigen, einfachen Prozessorinstruktionen wie "addiere", "multipliziere" oder "lese aus einer Tabelle" aus-

[4] **A**daptive **L**inear **E**lement **L**inear **N**euron.
[5] **D**efense **A**dvanced **R**esearch **P**rojects **A**gency.
[6] **D**igital **S**ignal **P**rocessor.
[7] **C**onnections **P**er **S**econd.
[8] **P**ersonal **C**omputer.
[9] **D**elta **F**loating **P**oint **P**rocessor.
[10] **S**cience **A**pplication **I**nternational **C**orporation.
[11] **C**onnection **U**pdates **P**er **S**econd.
[12] **P**rocessing **U**nit.

kommen, sind Prozessoren mit komplexen Befehlssätzen von Nachteil. Sogenannte RISC[13]-Prozessoren wurden entwickelt. Sie verfügen über einen Minimalsatz von gut parallelisierbaren Instruktionen. Komplexere Befehle werden durch Sequenzen einfacher Instruktionen ersetzt. Diese Geschwindigkeitsvorteile gegenüber sogenannter CISC[14]s nutzten Hiraiwa et al. 1990 zum Bau einer 128 Intel 80860 Prozessoren umfassenden Architektur [HKAI90]. Für ein dreischichtiges Netz mit den Neuronenzahlen 256-80-32 und 5120 Trainingsmustern, bestehend aus Ein/Ausgabemusterkombinationen, gibt Hiraiwa 1000MCUPs an.
Ein alternativer Ansatz ist der Bau sogenannter SIMD-Netzwerke[15] [FB97]. Deren PUs sind nicht mit Controllereigenschaften ausgestattet und werden über eine zentrale CPU[16] gesteuert. Die von der CPU ausgehenden Instruktionen werden von allen PUs parallel bearbeitet.

Analoge Lösungen Die Grundschaltungen analoger Schaltungstechnik sind oft langsamer als ihre digitalen Gegenstücke [Kra96]. Weil analoge Schaltungen die physikalischen Charakteristiken ihrer Bausteine direkt nutzen können, ist die Anzahl benötigter Elemente für Operationen gleicher Präzision hingegen meist geringer. Dies geht einher mit geringerer Baugröße und höherer Geschwindigkeit der Gesamtschaltung; beispielsweise die Stromsummierung lässt sich digital nicht parallelisieren. Bei der Wahl analoger Lösungen muß allerdings bedacht werden, daß sie schlechter skalieren. Während die digitale Signalübertragung robust gegen relativ große Rausch-zu-Signalverhältnisse ist, ist die Aufrechterhaltung der Signalintegrität analoger Übertragung über große Strecken nicht trivial. Überdies wachsen beispielsweise Umladezeiten durch zusätzliche Leitungskapazitäten. 80170NW ETANN[17] von Intel ist ein Beispiel für ein Ende der neunziger Jahre entwickeltes analoges Netz [Int91]. Der Chip hat 64 analoge Eingänge. Die Neurone haben eine sigmoide Ausgangsfunktion deren Verstärkung wählbar ist. Für Rückkopplungen steht ein zweiter Satz Netzgewichte zur Verfügung. Eine 64-64-64 Topologie kann in $8\mu s$ evaluiert werden. Dies entspricht einer Geschwindigkeit von 2GCPS. Die Trainingsprozedur ist allerdings sehr aufwendig. Sie wird extern durch einen PC gesteuert.

Hybride Lösungen Hybride Lösungen sind Architekturen, die analoge mit digitalen Komponenten verbinden. Ein schneller Chip dieser Gattung ist der NeuroClassifier vom Mesa Research Institute der Universität Twente (1994) [MHW94]. 70 analoge Eingänge werden hier über eine sechs Neurone umfassende Zwischenschicht einem analogen Ausgang zugeführt. Die Gewichte sind digital, sie umfassen

[13] **R**educed **I**nstruction **S**et **C**omputer.
[14] **C**omplex **I**nstruction **S**et **C**omputer.
[15] **S**ingle **I**nstruction **M**ultiple **D**ata.
[16] **C**entral **P**rocessing **U**nit.
[17] **E**lectrically **T**rainable **A**nalog **N**eural **N**etwork.

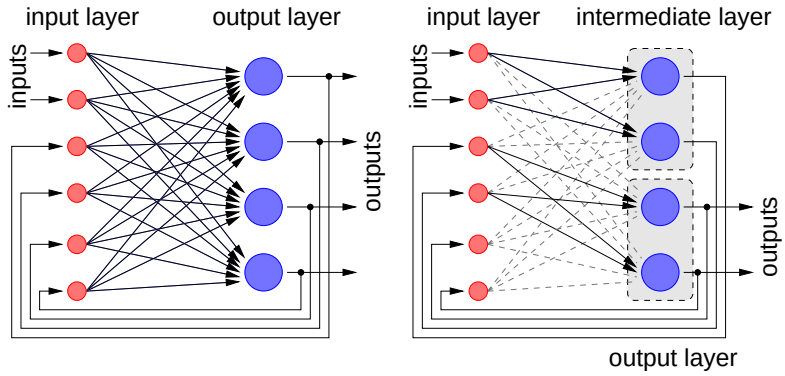

Abbildung 2.3: Der HAGEN-Chip stellt programmierbare Rückkopplungen zur Verfügung und implementiert somit ein mehrschichtiges Perzeptron.

jeweils 5 bit. Der Chip leistet 20GCPS.
Der im nächsten Kapitel ausführlich beschriebene HAGEN-Chip [SSHM02], gehört ebenfalls zur Klasse hybrider Chips. Seine Synapsen sind analog, Ein- und Ausgänge hingegen binär implementiert. Diese Struktur vereint die Vorteile analoger Gewichte mit der Skalierbarkeit digitaler Topologien. Wegen den parallel arbeitenden Synapsen, erreicht dieser Chip 1,64 TCPS [SHMS02] und 400 MCUPS.

2.2.2 Der neuronale Netzwerk-Chip HAGEN

HAGEN (**H**eidelberg **A**nalo**g** **E**volvable **N**eural Network) ist die Implementation eines Perzeptrons (S.13) in Hardware [SSHM02][SHMS02]. Dieser Chip wurde von der Gruppe *Electronic Vision(s)* um Prof. Dr. Karlheinz Meier und Dr. Johannes Schemmel entworfen. Seine Architektur ist hybrid: die Synapsen und dementsprechend die Netzwerkgewichte sind analog, Ein- und Ausgänge binär. HAGEN ist unterteilt in vier Netzwerkblöcke mit jeweils 128 Eingangs- und 64 Ausgangsneuronen. Jedes Eingangsneuron eines Blockes ist mit allen Ausgangsneuronen desselben verbunden. Da die Ein- und Ausgänge dieser Blöcke binär sind, sind die analogen Elemente durch die digitalen Schnittstellen gekapselt. Ausgangssignale können auf Eingänge des gleichen oder eines anderen Blockes rückgeführt werden. Mehrschichtige Netztopologien sind so schon mit einem Chip möglich. Aufgrund der digitalen Schnittstelle von HAGEN ist außerdem eine nahezu beliebige Anzahl HAGEN-Chips zu großen Netzen koppelbar. Ein HAGEN-Zyklus lässt sich durch eine geschlossene Formel beschreiben:

$$A_k(t+\Delta t) = \theta\left(\sum_{\text{keine Rückk. nach } l} E_l(t)\omega_{lk} + \sum_{\text{Rückk. } m \to n} A_m(t)\omega_{nk}\right) \tag{2.1}$$

Dabei sind E, A, ω Eingänge, Ausgänge bzw. Gewichte, $\frac{1}{\Delta t}$ die Taktung des HAGEN-Chips und θ die Heavysidefunktion. Die Synapsen sind, analog implemen-

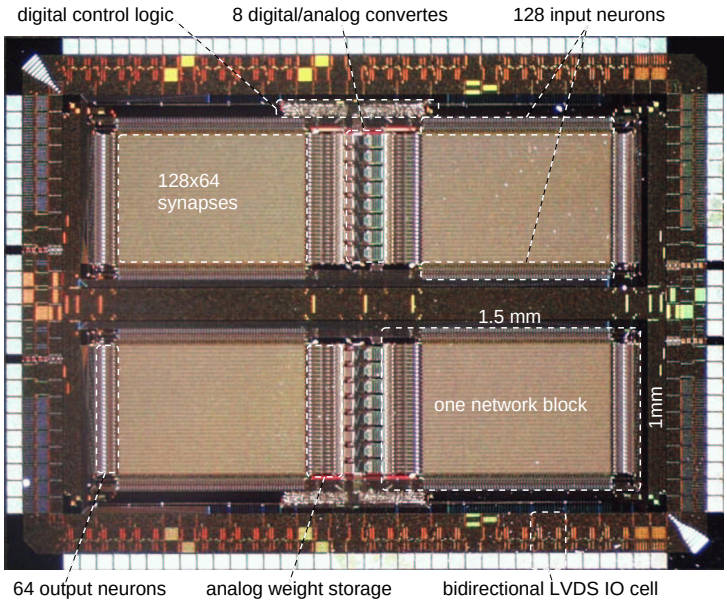

Abbildung 2.4: Photographie der vier Netzwerkblöcke eines HAGEN-Chips.

tiert, schneller als vergleichbare Digitallösungen. Wegen der binären Eingangssignale, reduziert sich die Synapsenfunktion auf eine einfache Summationsschaltung für Ströme. Ein weiterer Vorteil analoger Synapsen ist deren geringe Baugröße. HAGEN ist in $0.35\mu m$ CMOS[18]-Technik gefertigt und vereint 32768 Synapsen auf einer Fläche von 12,2 mm^2 (Abb. 2.4). Das Netz ist zeitdiskret mit einem Takt von etwa 50MHz. Dies führt zu einem Durchsatz von 1,64 TCPS.
Um das Training zu beschleunigen, werden die Gewichte nicht wie häufig in sogenannten EEPROM[19]s gespeichert. Sie werden stattdessen durch, über Kondensatoren einstellbare, Stromquellen repräsentiert. Wegen unvermeidbarer Leckströme werden die Kondensatoren alle 10-100ms aufgefrischt. Die Gewichte werden über sechzehn im Chip integrierte 10-bit DACs[20] dem Chip von außen digital zugeführt. Jeder dieser DACs benötigt $40\mu s$ pro Wandlung. So können 400 Millionen Gewichte in der Sekunde geladen werden (400 MCUPS). Gewichtskonstellationen können demnach mit maximal etwa 12,2kHz geladen werden.

2.2.3 Darkwing, Die Verbindung zur Computerwelt

Die Electronic Visison(s) Gruppe hat sich auf evolutionäre Trainingsalgorithmen spezialisiert. Es sind beliebig komplizierte evolutionäre Algorithmen denkbar. In der Prototypphase, in der sich HAGEN noch befindet, ist ein System sinnvoll,

[18] **C**omplementary **M**etal **O**xide **S**emiconductor.
[19] **E**lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory.
[20] **D**igital **A**nalog **C**onverter.

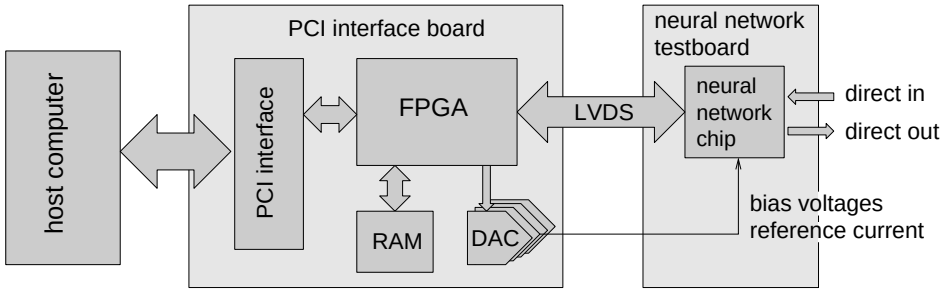

Abbildung 2.5: Die Darkwingplatine verbindet HAGEN über den PCI-Bus mit einem Standard-PC.

mit dem verschiedene Algorithmen getestet werden können. Deshalb wird das Training derzeit von einem Standard PC bewerkstelligt. Die vom Computer bereitgestellten Individuen werden über den PCI[21] Bus zu Darkwings übertragen (Abb. 2.5). Darkwing ist eine von Electronic Vision(s) entwickelte PCI-Steckkarte [SHSM02], die spezielle Adapterplatinen aufnehmen kann. Letzteren wiederum sind die HAGEN-Chips aufgesteckt. Darkwing beherbergt einen FPGA[22], dieser übernimmt die Details der Kommunikation zwischen Computer und HAGEN-Chip. Die Kommunikation zwischen FPGA und HAGEN erfolgt über einen LVDS[23]-Bus mit einer Übertragungsrate von 11,4 Gbit/s. Im derzeitigen Stadium wird also die Übertragungsrate zum Chip von der des PCI Busses bestimmt. Der PCI Bus in der benutzten Version 2.0 stellt eine Maximalrate von 132 MByte/s bereit. Es können Individuen dementsprechend mit einer maximalen Frequenz von ca. 2kHz übertragen werden, hierbei ist der Protokolloverhead nicht berücksichtigt.

2.2.4 Ansteuerung via HANNEE

Das Programm zur Steuerung des HAGEN-Chips heißt HANNEE[24]. Programmiert in C++ basiert seine Oberfläche auf Qt, einer plattformunabhängigen Grafikbibliothek der Firma Trolltech. Obwohl ausschließlich unter Linux entwickelt, stehen der Portierung auf andere Systeme deshalb keine grundsätzlichen Schwierigkeiten im Wege. Außer einer grafischen Benutzeroberfläche zum Training des HAGEN-Chips, bietet das Programm intern auch C++ Interface-Klassen zur Hardwareansteuerung an. HANNEE befindet sich noch in der Entwicklung. Um die von mir entwickelte Software in späteren Versionen trotzdem unverändert übernehmen zu können, habe ich auf HANNEEs Interfaceklassen[25] eine minimale Schnittstelle

[21] **P**eriphal **C**omponent **I**nterconnect.
[22] **F**ield **P**rogrammable **G**ate **Array**.
[23] **L**ow **V**oltage **D**ifferential **S**ignal.
[24] **H**eidelberg **A**nalog **N**eural **N**etwork **E**volution **E**nvironment.
[25] Objekte objektorientierter Programmiersprachen, die eine Schnittstelle definieren.

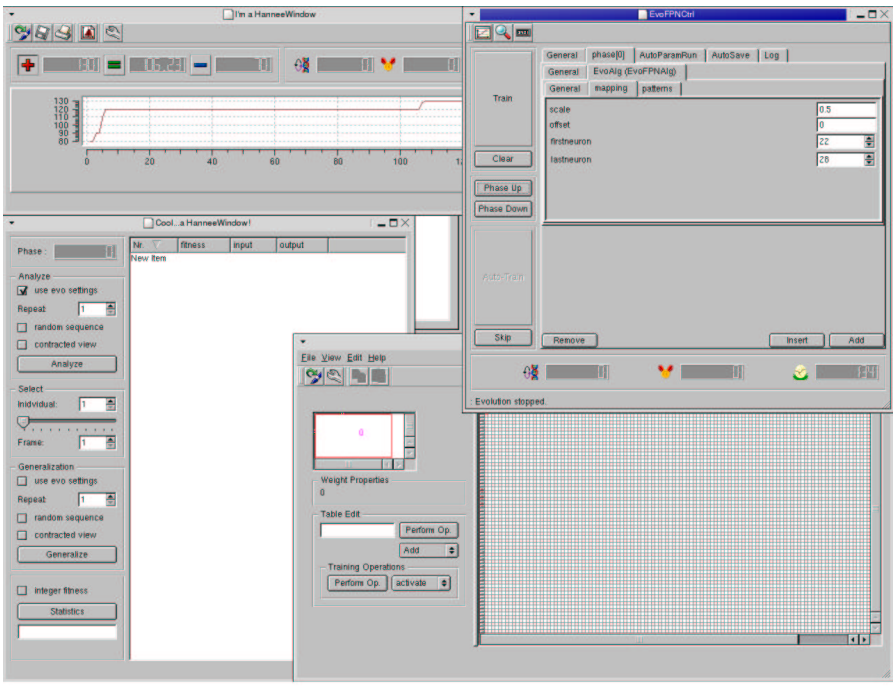

Abbildung 2.6: HANNEE bietet eine grafische Oberfläche zur HAGEN-Steuerung.

aufgesetzt. Diese bietet die Möglichkeit, HANNEE sowohl über eine MathLink-Schnittstelle[26], als auch über zukünftige C++ Programme fernzusteuern. Die binäre Implementation des Neocognitrons in Form des Mathematica-Programms BONNEE[27] baut auf dieser Schnittstelle auf (Kap. 4.1, 4.3, 4.4). Eine später entwickelte Software Simulation des HAGEN-Chips habe ich mit einer nahezu identischen Schnittstelle ausgerüstet, so lassen sich Hardware und Simulation leicht austauschen. Näheres hierzu in Kapitel 4.4.

2.3 Gängige Netztopologien - Verschaltungsoptionen für die Hardware

Bei der Suche nach einer geeigneten Netzstruktur für die Lösung eines Problems hat man nicht nur die Wahl zwischen verschiedenen Neurontypen, auch der Verschaltung der Neuronen sind kaum Grenzen gesetzt. Es gibt jedoch einige grundsätzliche Verschaltungsideen, die anhand prominenter Netztopologien im folgenden vorgestellt werden. Die hier beschriebenen Topologien sind Grundstock

[26] Ein von der Firma Wolfram zur Verfügung gestelltes TCP/IP-fähiges Protokoll.
[27] **B**asic **O**ptimizing **N**eural **N**etwork **S**tructure **E**volution **E**nvironment.

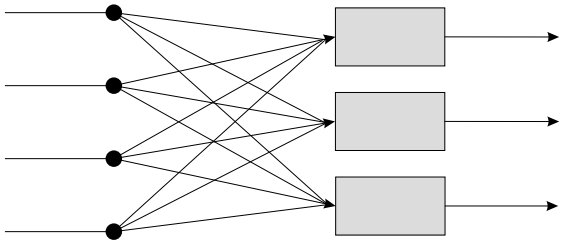

Abbildung 2.7: Beispiel eines einschichtigen Musterassoziators. Der Musterassoziator ist vorwärtsgerichtet.

im Handwerkszeug eines Netzwerktheoretikers; die Auflistung erhebt keinen Anspruch auf Vollständigkeit. Da die in Kapitel 2.2.2 vorgestellte Hardware nicht alle diese Topologien gleichermaßen unterstützt, wird hier die Wahl des Neocognitrons für unser Vorhaben der Bildverarbeitung mit HAGEN eine erste Begründung finden. Weiterführende Erläuterung zu den vorgestellten Netzwerken findet man in [Roj96], [Hof93].

2.3.1 Der Musterassoziator

Der Musterassoziator ist vorwärtsgerichtet - es gibt keine Rekursionen. Obwohl es auch mehrschichtige Musterassoziatoren gibt, ist der einschichtige Fall von grundsätzlichem Interesse. Im Falle einer linearen, monoton steigenden Ausgangsfunktion der Neurone kann man zeigen, daß diese Topologie lediglich voneinander linear unabhängige Muster lernen kann [Hof93]. Für Sätze linear unabhängiger Eingangsmuster konvergiert die sogenannte *Delta-Lernregel*: gruppiert man die Eingangssignale zu einem Vektor und tut selbiges mit den entsprechenden, einem Neuron zugeordneten, Netzgewichten (Abb. 2.2), so modifiziert die Delta-Lernregel diesen Gewichtsvektor mit einem zum Eingangsvektor parallelen Differenzvektor ($\delta\vec{\omega} = \eta(Soll - Ausgang)\overrightarrow{Eingänge}$). Es wird angenommen, daß jene Konvergenz auch für alle nicht linearen Ausgangsfunktionen gilt. Wie die allermeisten Netzwerktypen, unterliegt auch der Musterassoziator dem *Plastizitäts-Stabilitäts-dilemma*, d.h. das Netz kann nur eine bestimmte Anzahl Muster lernen. Wird diese Grenze überschritten, stören sich die gelernten Muster gegenseitig in ihrer Ausgabe. Ein spezieller, wohlbekannter Musterassoziator ist das einschichtige *Perzeptron*. Dieses beschränkt die Ausgangsfunktion auf eine Stufenfunktion mit den Ausgangssignalen 0 und 1. Ordnet man jedem Ausgang genau eine Klasse zu, so arbeitet dieses Netz als Klassifizierer. Die Eingangsmuster werden in Klassen aufgeteilt. Ein solcher Klassifizierer ist direkt auf HAGEN abbildbar und findet in unserer Version des Neocognitrons Verwendung (Kap. 3.2.1).

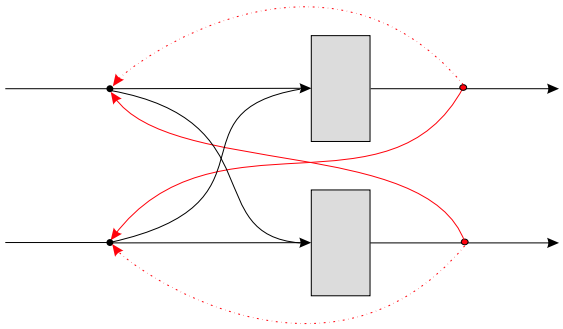

Abbildung 2.8: Beispiel eines einfachen Autoassoziators. Im Gegensatz zum Musterassoziator besitzt der Autoassoziator Rückkopplungen.

2.3.2 Der Autoassoziator

Der Autoassoziator assoziiert Muster mit sich selbst. Deshalb hat er stets genausoviele Ein- wie Ausgänge. Eingesetzt wird er z.B. zur Rekonstruktion von verrauschten Mustern. Eine vom Netz gelernte Selbstabbildung bildet auch verrauschte Eingangsbilder auf das Original ab - das Bild wird entrauscht. Dasselbe Prinzip wird verwandt, um einem Bild das ähnlichste Bild aus einer vorgegebenen Menge gelernter Daten zuordnen zu können. Ein interessanter Einsatzbereich solcher Netze ist die Vervollständigung von Bildfragmenten. Wegen seiner Rückkopplungsleitungen muß die Neuronausgangsfunktion beschränkt sein, andernfalls ist eine "Resonanzkatastrophe" möglich (Abb. 2.8). I.a. wird auf Selbstrückkopplung[28] verzichtet. So auch beim *Hopfieldnetz*. Die Gewichte aller Neuronenpaare des Hopfieldnetzes sind symmetrisch angelegt, d.h. eine Leitung vom Ausgang des Neurons a zum Eingang des Neurons b trägt dasselbe Gewicht wie jene von b nach a. Damit kann man das System durch eine Hamiltonfunktion beschreiben. Sie hat die Eigenschaft, mit der Zeit bloß abnehmen zu können. Im Gleichgewichtszustand ist die Hamiltonfunktion minimal[29]. Hopfieldneuronen sind binär, die Ausgangswerte sind also aus der Menge {0,1} - der Chip HAGEN besteht aus solchen Neuronen. Gelernt wird grundsätzlich mit der *Hebbschen Lernregel* ($\delta\omega_{kl} = Soll_k Eingang_l$). Es existiert eine geschlossene Lösung für den Lernvorgang. Diese funktioniert allerdings nur dann zufriedenstellend, wenn jedes Muster aus etwa derselben Zahl beider Binärwerte besteht. Die Obergrenze befriedigend lernbarer Muster in Hopfieldnetzen ist etwa $0,14 \cdot$Neuronenzahl. Diese Netztopologie habe ich in vorbereitenden Versuchen zur Simulation der Generalisierung eines Perzeptron Aufbaus verwendet (Kap. 4.2.5.1). Denkbar ist auch der Einsatz in der Vorverarbeitung von Bildausschnitten.

[28] Der Ausgang eines Neurons wird auf den eigenen Eingang rückgeführt.
[29] Möglicherweise lediglich lokal minimal.

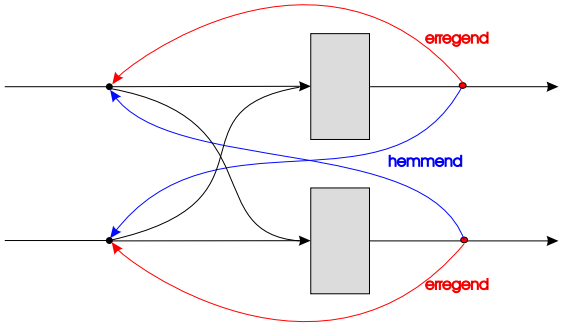

Abbildung 2.9: Beispiel einer Feature-Gruppe innerhalb eines Wettbewerbsnetzes. Nur die Selbstrückkopplungen sind erregender Natur.

2.3.3 Das Wettbewerbsnetz

Im Wettbewerbsnetz ist nur die Selbstrückkopplung erregender Natur. Alle Neurone hemmen sich gegenseitig[30]. Zur Klassifikation teilt man das Netz in vollständig verbundene Subnetze auf, die untereinander keine Verbindung haben. Innerhalb einer solchen *Feature-Gruppe* feuert immer genau ein Neuron. Dieses Neuron hat den Wettbewerb gewonnen. Ihm ist eine Klasse zugeordnet, das Eingangsmuster ist somit Teil dieser Klasse. Baut man die Klassifizierung in Form einer *Feature-Kaskade*, so nennt man das Netz hierarchisch. Feature-Kaskaden sind Entscheidungsbäume, deren Knoten auf bestimmte Features ansprechen. Ein Beispiel einer Feature-Kaskade ist das Neocognitron. Eine binäre Ausgangsfunktion macht aufgrund der gegenseitigen Hemmung der Neurone wenig Sinn. Diese Topologie ist also nur unter Einsatz sogenannter Integerneuronen auf HAGEN zu bewerkstelligen. Integerneurone sind Verbände von Neuronen mit binärem Ausgang, die so verschaltet und trainiert sind, daß eine Kombination binärer Ausgänge ein im Dualsystem kodiertes Integersignal darstellt. Zur Darstellung eines Integerneurons werden entsprechend viele Binärneuronen benötigt. Eine Wettbewerbsstruktur mit HAGEN zu bewerkstelligen, ist demnach nicht trivial, es würden insbesondere mehrere HAGEN-Blöcke, vielleicht sogar mehrere HAGEN-Chips nötig.

2.3.4 $\sigma\pi$-Netze

$\sigma\pi$-Neuronen sind Neuronen, deren Eingangsfunktion nicht (wie sonst üblich) das Skalarprodukt aus Eingangs- und Gewichtsvektor ist, also die Eingänge unabhängig voneinander gewichtet. Stattdessen werden höhere Ordnungen berücksichtigt. Das heißt, auch Produkte verschiedener Eingänge tragen, entsprechend gewichtet, zum effektiven Neuroneingang bei. Probleme, die nicht linear, aber beispielsweise quadratisch separierbar sind, können mit vergleichsweise wenigen

[30] Physiologischer Fachterminus: *laterale Inhibition*.

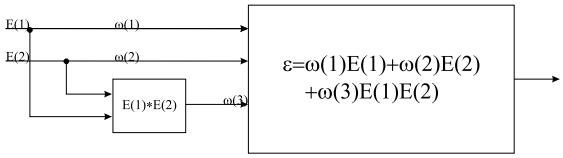

Abbildung 2.10: $\sigma\pi$-Neurone sind Neurone höherer Ordnung. Ein derartiges Neuron kann nicht linear separable Probleme lösen.

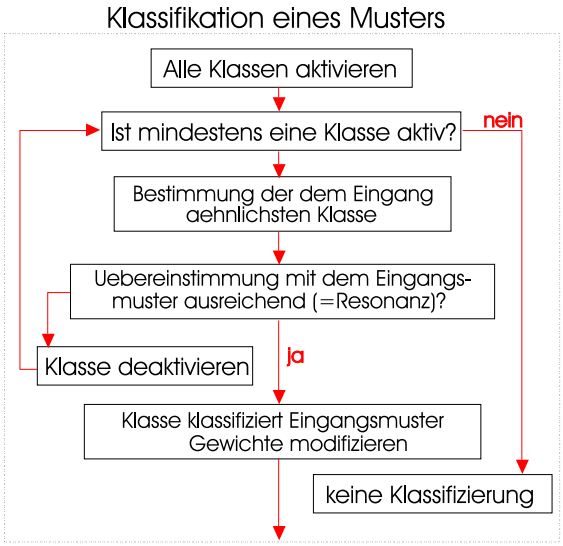

Abbildung 2.11: Das ART1 löst das Plastizitäts-Stabilitäts-Dilemma.

solcher Neurone gelöst werden. Klassisches Einsatzgebiet ist die invariante Mustererkennung. In simulierten Netzen wird der Vorteil geringer Neuronenzahl durch deren Komplexität gemindert. In HAGEN stehen Leitungen höherer Ordnung nicht zur Verfügung, $\sigma\pi$-Neuronen können deshalb ausgeschlossen werden.

2.3.5 Das ART1

Das ART1[31] ist ein höchst komplexes, vierschichtiges Netzwerk, innerhalb dessen auch rückgekoppelt wird. Die dem Netz zugrundeliegende Idee ist hingegen leicht in Form eines Ablaufdiagramms darstellbar (Abb. 2.11). In seiner ursprünglichen Fassung war das ART ein zeitkontinuierliches Netz. Es existieren aber auch zeitdiskrete ARTs, die der ebenfalls getakteten Computerwelt besser entsprechen. Das ART ist eines der wenigen Konzepte, die das *Stabilitäts-Plastizitäts-Dilemma* lösen. Es weist nämlich, falls die Lernkapazität des Netzes erschöpft ist, eigenständig solche Muster ab, die nicht in bestehende Klassen passen. Die Klassifikationsschicht

[31] **A**daptive **R**esonance **T**heory.

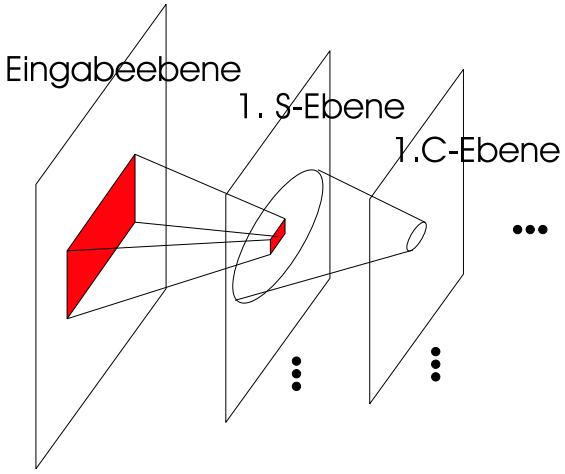

Abbildung 2.12: Stark vereinfachtes Schema einer Neocognitron-Stufe.

selber ist eine Wettbewerbsschicht - auch deshalb ist das ART schwerlich mit HAGEN darstellbar.

2.3.6 Das Neocognitron

Während der Betrachtung eines unbekannten Gegenstandes wandert der Fokusbereich des Auges scheinbar ziellos über dessen Oberfläche. Nach einem Grobeindruck erhöht das Auge Schritt für Schritt die Auflösung. Mit fortschreitender Betrachtungsdauer rücken immer kleinere Features, immer kleinere Details, in den Blickpunkt. Von ähnlichen Erkenntnissen wurde Kunihiko Fukushima zur Entwicklung einer *Cognitron* genannten Netzstruktur animiert, die von ihm später zum Neocognitron verbessert wurde (mehr hierzu in Kap. 3.1). Kehrt man diesen Prozeß der schrittweisen Erhöhung der Auflösung um, so kann man einen Gegenstand als Feature hoher Ordnung betrachten. Kleine Details werden zusammengesetzt zu größeren Eigenschaften, die ihrerseits zusammengesetzt werden, solange bis die Features die Größe des Originals annehmen. Die Ordnung der Features in dieser Hierarchie steigt mit jeder Stufe. Beispielsweise kann man aus zwei Kanten eine Ecke und aus vier Ecken ein Quadrat konstruieren. Ein Feature geringer Ordnung wäre dabei eine Kante, ein Feature höherer Ordnung eine Ecke. Dieses Prinzip nutzt das Neocognitron: in der ersten Stufe werden kleinste Bildausschnitte bekannten Features zugeordnet, also klassifiziert. In der nächsten Stufe werden mehrere dieser Features zu einem Feature höherer Ordnung zusammengesetzt und mit dem entsprechenden Satz bekannter Features verglichen. Dieser Vorgang erfolgt solange, bis die Größe der Features der aktuellen Stufe mit der Größe des gesamten Bildes übereinstimmt. Das Bild ist dann klassifiziert. Da nicht alle Gegenstände, die zur gleichen Klasse gehören, identisch sind, muß das Netz eine gewisse Toleranz

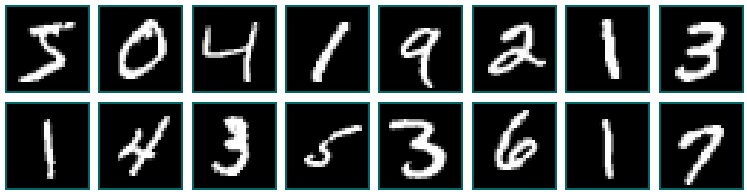

Abbildung 2.13: Auszug aus der MNIST Datenbank. MNIST stellt etwa 70000 handgeschriebene Ziffern in Form von Graustufenbildern bereit.

gegen Translation und Deformation von Features und ihrer Anordnung untereinander aufweisen. Die zwei wesentlichen Instanzen eines Neocognitrons sind in Abb. 2.12 vereinfacht dargestellt. Eine S-Ebene klassifiziert den aus der Eingabeebene extrahierten Bildausschnitt. Die anschließende C-Ebene sorgt für die Translationsinvarianz, indem sie die S-Ebene verschmiert wiedergibt. Die S-Zellen als atomare Bestandteile der S-Ebenen sind in ihrer Funktionsweise sehr kompliziert (s.h. Kap. 3.1) und durchweg analog. Eine direkte Abbildung auf HAGEN ist also für das Neocognitron in seiner ursprünglichen Form nicht möglich. In Kap. 3.2 wird gezeigt, daß unter Beibehaltung der grundsätzlichen Struktur eine Realisierung mit dem gemischt analog/digitalen Netzwerkchips HAGEN denkbar ist.

2.4 Schrifterkennung

Die elektronische Schrifterkennung ist durch das Aufkommen mobiler Systeme wie elektronischer Kalender oder Handys ein wirtschaftlich interessantes Gebiet geworden. Auf dem japanischen Elektronikmarkt gehört Handschrifterkennung bereits zum Alltag. Alle diese mobilen Systeme haben gegenüber konservativen Vorhaben, wie der Computerisierung von Handschriftarchiven, einen entscheidenden Vorteil: sie verfolgen die Eingabe. Somit können sie den Vektorzugcharakter eines Wortes optimal nutzen. Die sogenannten *offline*-Systeme müssen hingegen mit weniger Informationen auskommen. Grundlegende Dinge wie Schreibrichtung und Größe müssen vorab ermittelt werden. So ist nahezu allen kommerziellen Systemen eine umfangreiche Vorverarbeitung gemein. Systeme, die eine Drehinvarianz bieten, sind die Ausnahme. Da unser Vorhaben, mit einem binären Neocognitron ein System zu schaffen, dessen Prinzip soweit wie möglich datentypunabhängig ist, kein Interesse an der Lösung spezieller Probleme der Schrifterkennung hat, habe ich die MNIST-Zifferndatenbank genutzt [LeC03]. Diese Datenbank beinhaltet eine umfassende Sammlung handgeschriebener Ziffern. In der Summe werden hier 70000 Ziffern angeboten. Sie sind alle bereits vorverarbeitet, haben die gleiche Größe, sind zentriert und ausgerichtet (Abb. 2.13). Die Ziffern werden allerdings in Grauwertbildern geliefert und mußsen deshalb in Binärwertbilder umgerechnet werden (Abb. 2.14). Ihr Format 28x28-Punkte beinhaltet etwa sechsmal mehr Bildpunkte

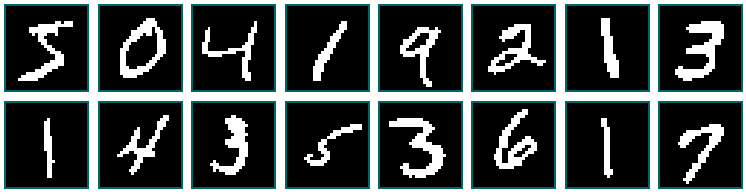

Abbildung 2.14: Die in dieser Arbeit benutzten Bilder sind schwarz/weiß.

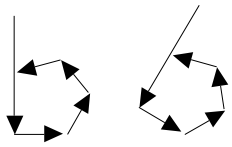

Abbildung 2.15: Vektorgrafiken sind dreh-,skalierungs- und verschiebungsinvariant.

als ein HAGEN-Block Eingänge vorweist. Deshalb und wegen der Vergleichbarkeit mit anderen Systemen, sind diese Daten geeignet für erste Versuche auf dem Gebiet der Bildpartitionierung.

2.4.1 Vektor-, Punkt- und Objektgrafiken

Grundsätzlich kann man zwischen Vektor-, Punkt- und Objektgrafiken unterscheiden. Vektorgrafiken sind gerichtet und überwiegend schwarz/weiß. Da Vektorgrafiken aus Relativvektoren (bzw. Kurven) aufgebaut sind, sind sie dreh-, verschiebungs- und skalierungsinvariant. Aus diesem Grunde benutzen alle modernen Grafik- bzw. Satzsysteme im Computerbereich sogenannte *TrueType*[32] oder *Type1*[33] Schriften. Es existieren zwar noch immer auch Punktgrafik[34]-Schriftsätze, diese gelten allerdings wegen ihrer schlechten Skaliereigenschaften als veraltet und beziehen ihre Existenzberechtigung vorwiegend aus Kompatibilitätsgründen. Vektorschriftzüge sind für die Schrifterkennung angenehmer, da sie Informationen wie Schreibrichtung, Buchstabengröße oder Relationen zwischen Buchstaben bereits implizit enthalten. Sogenannten *offline*-Systemen (s.h. S. 18) liegen Schriften aber meist als Punktgrafik vor - die Übersetzung in Vektorgrafiken ist ein nicht triviales Problem.
Punktgrafiken sind überwiegend mehrfarbig und ungerichtet. Der Bildrahmen gibt die Betrachtungsrichtung vor. Sie sind deshalb nicht drehinvariant. Skalierung ist nur bedingt möglich. Sie werden überwiegend zur Darstellung von Oberflächen verwendet. Diese sogenannten *Texturen* sind dann in Vektorgrafik eingebettet. Solche

[32] TrueType ist ein von Microsoft und Apple definiertes Verfahren für skalierbare Vektorschrift.
[33] Mit Type1 bezeichnet man skalierbare Postscript Schriftsätze [Vos01].
[34] Englisch: bitmap.

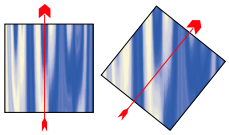

Abbildung 2.16: Der Rahmen einer Punktgrafik legt die Interpretationsrichtung des Bildes fest. Punktgrafiken sind nicht dreh- oder verschiebungsinvariant, Skalierung ist nur bedingt möglich.

aus Vektorgrafik und Texturen zusammengesetzten Bilder kann man als Objektbilder bezeichnen. In Computeranimationen wird massiv von dieser Kombination Gebrauch gemacht. Texturen sind dort durch auftretende Treppeneffekte beim Vergrössern an Gegenständen leicht entlarvbar.

2.4.2 Spezielle Probleme der Schrifterkennung

Allgemeine Probleme der Bilderkennung sind Verschiebungen, Drehungen, Skalierungen und Rauschen. Diese Herausforderungen bestehen unabhängig vom bereits gelernten Datensatz. Große Teile dieser Aufgaben werden von der Vorverarbeitung übernommen. In unserem Fall sind die Bilddaten bereits vorverarbeitet. Typische Probleme bei der Handschrifterkennung, aufgrund der Existenz verschiedener Schriftsätze aber auch im gedruckten Bereich, sind individuell verschiedene Abstraktionen, Verzierungen, Serifen und Ungleichmäßigkeiten. Hat das Schrifterkennungssystem einen Datensatz, also einen Schriftsatz, erfolgreich gelernt, so ist es nicht automatisch in der Lage, andere Schriftsätze zu erkennen. Noch schwieriger ist die Übertragung auf andere Datentypen - beispielsweise Lateinische Buchstaben verglichen mit chinesischen Schriftzeichen. Fast alle kommerziellen Systeme nutzen spezielle Eigenschaften des Datentyps und oft auch der Datensätze. So ist ein OCR-System[35], das zur Erkennung des Lateinischen Alphabets entwickelt wurde, nur schlecht oder gar nicht zur Erkennung chinesischer Schriftsätze geeignet. Die Verwendung von Punktgrafiken in unserem Aufbau stellt, verknüpft mit dem Verzicht auf Vektorisierung, ein Schritt in Richtung Datentypunabhängigkeit dar. Unterstützt wird dies durch das datentypunabhängige Funktionsprinzip des Neocognitrons. Datentypunabhängigkeit ist eines unserer primären Ziele.

2.4.3 Gängige Lösungsansätze im Vergleich

Hinter dem Begriff OCR versteckt sich eine Vielzahl verschiedener Schrifterkennungsalgorithmen [Mc.95]. Dennoch gibt es einige Gemeinsamkeiten. OCR-Algorithmen sind flache Verfahren. Im Gegensatz zu hierarchischen Ansätzen, ist

[35] **O**ptical **C**haracter **R**ecognition.

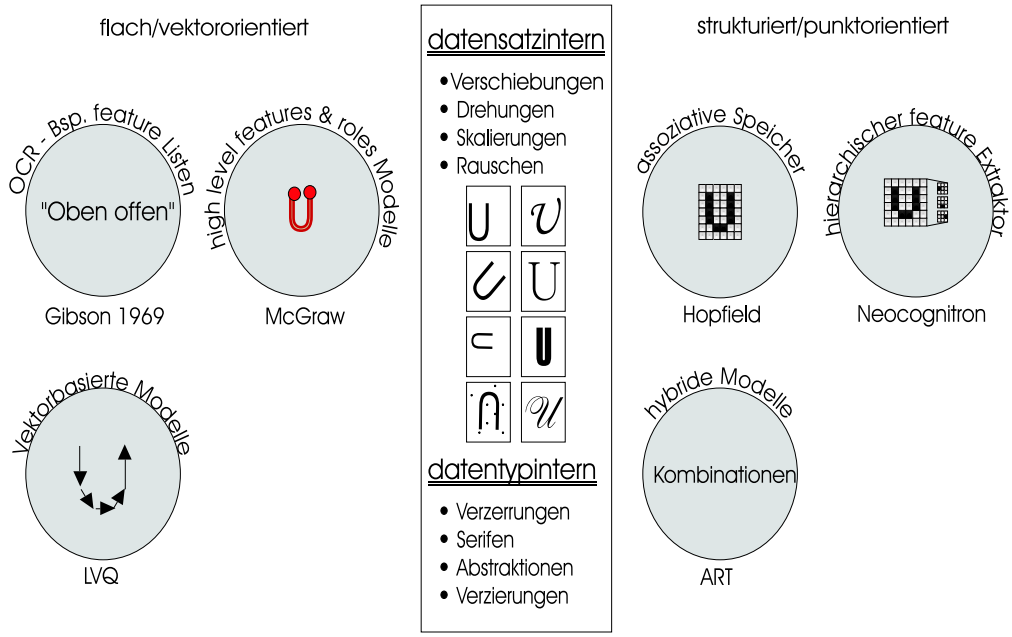

Abbildung 2.17: Verschiedene Lösungsansätze zur Schrifterkennung. Sie können in vektor- und punktorientierte Methoden eingeteilt werden und unterscheiden sich außerdem in den Verschachtelungstiefen ihrer Basisoperationen.

die Struktur der OCR-Algorithmen weniger komplex. Sie können zumeist in einfache, hintereinandergeschaltete Operationen zerlegt werden. Ziel ist die möglichst schnelle Erkennung bei hoher Erkennungsrate. Die Auswahl der von ihnen korrekt klassifizierbarer Datensätze ist speziell. Zu den OCR-Algorithmen gehören beispielsweise die sogenannten *Feature-Listen* [Gib69]. Feature Listen sind Listen von Eigenschaften, die die Buchstaben eines ausgewählten Datensatzes optimal trennen. Beispiel, im Falle eines lateinischen Alphabets, wäre "Lücke oben". Dieses simple Feature trennt (nach entsprechender Vorverarbeitung) die Buchstaben u,v,w,x,y vom Rest. Einige wenige dieser low level Features führen schnell zu hohen Erkennungsraten. Man erkennt leicht, daß ein solches System für die Erkennung chinesischer Schriftzeichen dann andere Feature Listen benötigt.
G.E.McGraw hat sich eines anderen Ansatzes bedient. Sein *high level features & roles* Modell nutzt Relationen unter den *low level features* [Mc.95]. Ein "u" könnte in diesem Modell beschrieben werden als eine Kurve, dessen beiden Endpunkte keine Verknüpfung zu anderen Elementen haben und eine Lücke oben bilden.
Weitere Möglichkeiten bietet die Verwendung sogenannter LVQ-Verfahren[36] [Koh88b]. LVQ klassifiziert Eingangsvektoren nicht mit Hilfe von Prototypen aus den Trainingsdaten. Es werden stattdessen Sätze statistisch günstigerer Vektoren zur Klassenrepräsentation aus der Abfolge der Trainingsvektoren gelernt. Der

[36] **L**earning**V**ector **Q**uantization.

Lernvorgang basiert auf Nachbarschaftsrelationen in den Klassendichtefunktionen des Eigenschaftsraumes. Die in diesem Abschnitt bisher genannten Verfahren basieren überwiegend auf Vektoreigenschaften der Schrift.
Assoziativspeicher (z.B. Abb. 2.7) sind Topologien neuronaler Netze, die in der Lage sind, Eingangsmustern Ausgangsmuster zuzuordnen. Sie können auch ähnliche, nicht gelernte Muster korrekt zuordnen. Um beispielsweise Handschrifterkennung bewältigen zu können, benötigen sie große Trainingsdatensätze. Dabei reagieren Assoziativspeicher empfindlich auf Verschiebungen und Rotationen. Als Teil eines übergeordneten Systems können sie sehr nützlich sein, beispielsweise in einem hierarchischen Modell oder als vorverarbeitende Instanzen (z.B. Hopfieldnetze S.14). Als Einzellösung scheiden sie schon wegen ihrer dann benötigten Abmessungen meist aus. Die in dieser Arbeit verwendeten Ziffern bestehen aus 784 Bildpunkten. Ein zugehöriger Assoziator müßte diese Anzahl an Eingangsneuronen aufweisen. Außerdem wird die Lerngrenze bei großen Trainingsdatensätzen schnell erreicht. Die Anzahl benötigter Zwischenebenen und damit die Gesamtzahl von Neuronen wäre damit noch um einiges größer.
Hierarchische Modelle partitionieren die Eingangsdaten in geeigneter Weise. Diese Bildfragmente werden dann in mehreren Stufen verarbeitet, die ihrerseits wieder partitionieren. Die Verknüpfungen unter den Bildfragmenten können beliebig komplex sein. Hierarchische Modelle sind die Antwort auf die oben genannten Dimensionsprobleme einfacher Assoziativspeicher.
Schließlich sind auch Kombinationen verschiedener Ansätze möglich. Man faßt sie unter dem Begriff *hybride Netzwerke* zusammen, als Beispiel sei hier das ART (S.16) genannt.

2.4.4 Kenngrößen einer Lösung

Bei der Beurteilung des Erfolges einer Lösung gibt es drei maßgebliche Größen: die Erkennungsrate, die Geschwindigkeit und die Datentypunabhängigkeit. Strukturierte Modelle sind in der Regel langsamer als flache Ansätze, bieten jedoch größere Datentypunabhängigkeit. Man stellt außerdem fest, daß die Größen Erkennungsrate und Datentypunabhängigkeit sich zumeist gegenläufig verhalten.

2.4.5 Pro und Contra Neocognitron

Da das Neocognitron als Eingangsdaten Punktgrafiken verwendet und ferner keine speziellen Eigenschaften der Eingangsdaten nutzt, ist es prinzipiell datentypunabhängig. Es gibt keine prinzipiellen Schwierigkeiten, die entwickelten Algorithmen zur Erkennung chinesischer Schriftzeichen zu nutzen. Zwar müssen dann einige Topologieparameter neu bestimmt werden; dieser Vorgang unterscheidet sich aber nicht von demjenigen, den ich zur Strukturoptimierung für arabische Ziffern aufgebaut habe. Diese Strukturoptimierung ist ein langwieriger und algorithmisch aufwendiger Prozeß, je Datentyp muß sie aber nur einmal vorgenommen werden. Ist

eine optimale Netzstruktur für einen Datentyp gefunden, ist die sequentielle Abarbeitung der Neocognitron-Stufen vergleichsweise langsam. Da die Arbeitsschritte innerhalb einer Neocognitron-Stufe jedoch sehr gut parallelisierbar sind, ist diese Komplexität mehr als ausgleichbar. Hierzu ist der HAGEN-Chip mit seiner fast beliebigen Skalierbarkeit bestens geeignet. Das Neocognitron ist nicht nur intern hierarchisch, sondern kann auch selbst Einheit einer übergeordneten Neocognitron-Struktur sein. Die Erweiterbarkeit des Neocognitrons ist eine der wichtigen Eigenschaften bei der Entscheidung für diese Struktur. In seiner Hierarchie findet sich außerdem bereits eine für beliebige Daten taugliche Bildpartitionierung.

Kapitel 3

Das Neocognitron

3.1 Das Original von Kunihiko Fukushima

3.1.1 Fukushimas Idee

Im visuellen Kortex des Großhirns gibt es Neurone, die selektiv auf lokale Features eines visuellen Musters reagieren [HW62]. Lokale Features sind dabei beispielsweise Linien und Ecken in bestimmten Orientierungen. Blakemore und Cooper konnten 1970 zeigen, daß die Verbindungen zwischen den entsprechenden Neuronen plastisch sind. Sie zogen Katzen in einer künstlichen Umgebung groß. Dort gab es lediglich schwarz/weiß Streifen einer Orientierung. Die Katzen hatten später keine Neurone, die sensitiv gegenüber Streifen anderer Orientierung sind [BC70]. Es gab jedoch noch keine evidenten Thesen, wie diese Selbstorganisation funktioniert. Kunihiko Fukushima schlug 1975 eine Hypothese vor, die die Selbstorganisation erklären sollte: "Die Synapse von Neuron x nach Neuron y wird verstärkt, falls x feuert und kein Neuron in der Nachbarschaft von y stärker feuert als y selbst." [Fuk75]. Gleichzeitig stellte er eine Netztopologie vor, die auf diese Art und Weise selbstorganisierend ist. Das sogenannte *Cognitron* baute bereits auf einer hierarchischen Extraktion von Features auf. Erst später wurde gezeigt, daß tatsächlich in höheren Hirnbereichen Zellen existieren, die selektiv auf Figuren wie Kreise, Dreiecke, Quadrate oder gar Gesichter - Features höherer Ordnung - ansprechen [BDG81], [SKI80]. Das Neocognitron ist die Weiterentwicklung des Cognitrons [Fuk88].

3.1.2 Die Grobstruktur

Das Neocognitron ist ein mehrstufiger Feature-Extraktor. Die nullte Stufe ist die Eingabeebene - das Bild selbst. Alle folgenden Stufen erhalten Ihre Eingaben von den Ausgaben der Vorgängerstufe. Jeder Stufe ist ein Feature-Satz zugeordnet. In der ersten Stufe der 1988 von Kunihiko Fukushima vorgestellten Topologie sind dies beispielsweise zwölf 3x3 Muster (Abb. 3.5) [Fuk88]. Aus jenen zwölf Mus-

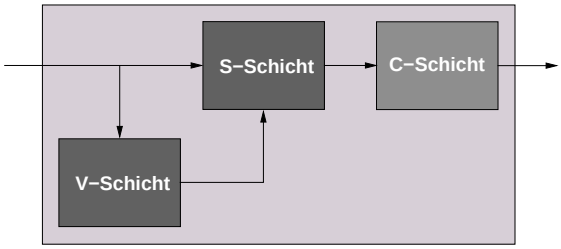

Abbildung 3.1: Blockschaltbild einer Neocognitron-Stufe.

tern lassen sich die von ihm verwendeten handgeschriebenen Ziffern approximativ konstruieren. Etwas ungenau, aber der Anschauung dienlich, kann dies wie folgt beschreiben werden: eine Neocognitron-Stufe kopiert die Eingabedaten so oft, wie die Anzahl der ihr zugewiesenen Features. Jede der Kopien ist einem dieser Features zugewiesen. Anschließend wird jedes Bild mit dem zugeordneten Feature gefiltert. Ein solcher Filter ersetzt alle Bildausschnitte der Größe des Features durch einen Wert, der angibt, wie gut der Ausschnitt mit dem Feature übereinstimmt. Die so entstandenen Grauwertbilder werden schließlich verwischt und an die nächste Stufe weitergegeben. Ab der zweiten Stufe haben die Stufen mehrere Eingangsbilder. Ein Fokusbereich bzw. ein Feature wird gebildet aus allen gleich großen Ausschnitten an gleicher Position in den verschiedenen Eingangsbildern. Man stellt ihnen deshalb das Wort "Hyper"voran. Jede Stufe besteht aus drei Schichten (Abb. 3.1). Die S-Schicht filtert die Eingangsdaten - sie ist die Klassifizierungsschicht. Die V-Schicht normiert die Ausschnitte - das ist notwendig, da sonst die Anzahl gesetzter Punkte in den Features unerwünschten Einfluß auf die Filterfunktion hätte. Die C-Schicht sorgt für die Verschmierung. Sie stellt damit eine gewisse Translationsinvarianz bereit. Ist ein Feature gegenüber dem Original ein wenig verschoben, wird es trotzdem erkannt - allerdings als weniger übereinstimmend klassifiziert.
Eine Schicht besteht aus mehreren Ebenen gleicher Größe (Abb. 3.2). Die Ebenen der S- und V-Schichten einer Stufe sind gleich dimensioniert. Eine Ebene besteht aus Zellen, deren analoge Ausgaben in Abb. 3.2 als Grauwerte dargestellt sind. Die Eingänge einer jeden Zelle sind verbunden mit einem Fokusbereich in einem anderen Schichttyp. Nur den S-Zellen wird ein zusätzliches Signal von einer ihr zugeordneten V-Zelle zugeführt. In Stufe Eins ist der Fokusbereich einfach ein Bildausschnitt. In den höheren Stufen wird er aus allen Ausschnitten zusammengesetzt, die sich an gleicher Position in den verschiedenen Ausgangsbildern der Vorgängerstufe befinden (die Ausschnittsgröße ist dabei für eine Schicht konstant) (Abb. 3.3). Ausnahme sind die C-Zellen. C-Ebenen sind direkt einer S-Ebene zugeordnet. Ihre Fokusbereiche sind Bildausschnitte aus genau einer S-Ebene der gleichen Stufe. S-Zellen erhalten ihre Eingabe aus dem Fokusbereich der C-Schicht der vorhergehenden Stufe. Dieser ist nur in der Stufe Eins ein zweidimensionaler Bildausschnitt - die C-Schicht der nullten Stufe besteht aus genau einer C-Ebene,

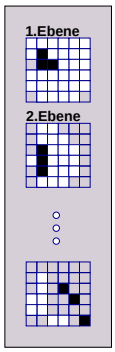

Abbildung 3.2: Schichten bestehen aus Ebenen; Ebenen aus Zellen mit analoger Ausgabe.

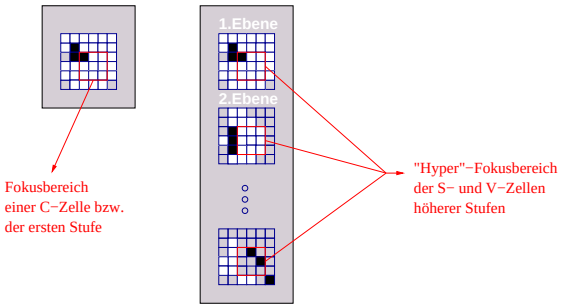

Abbildung 3.3: Fokusbereiche sind zwei- bzw. dreidimensional.

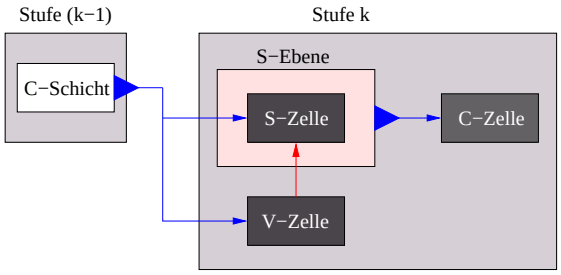

Abbildung 3.4: S- und V-Zellen erhalten ihre Eingabe von Hyper-Fokusbereichen (3D) der C-Schicht aus der Vorgängerstufe. Ausnahme ist Stufe 1. Die Ausgabe der V-Zellen gehen außerdem an die ihnen zugeordneten S-Zellen der gleichen Stufe. Der C-Zellen zugeordnete Fokusbereich ist zweidimensional, jede C-Ebene operiert auf genau einer S-Ebene der gleichen Stufe.

dem Originalbild. In höheren Stufen ist der Fokusbereich die geordnete Menge zweidimensionaler Bildausschnitte gleicher Position in den verschiedenen Ebenen der C-Schicht der vorhergehenden Stufe. Dieselbe Eingabe erhält auch die dieser S-Zelle zugeordnete V-Zelle. Ihre Ausgabe wiederum geht an die S-Zelle. *Die Position der Zellen innerhalb einer Ebene entspricht der Position der Fokusbereiche, von denen sie ihre Eingabe erhalten. Die Neocognitron-Stufen sind nachbarschaftserhaltend.* Eine Zelle in der rechten oberen Ecke einer Ebene erhält die Ausgabe eines Fokusbereiches der rechten oberen Ecke. Abbildung 3.5 zeigt diese Zusammenhänge am Beispiel der 1988 von Fukushima entworfenen Topologie zur Erkennung handschriftlicher Ziffern. Die V-Schichten sind der Einfachheit wegen weggelassen. Stufen sind nicht gekapselt dargestellt. Die Dimensionsangaben sind (Größe der Ebenen)x(Anzahl der Ebenen in der Schicht = Anzahl der Features der vorhergenden Stufe).

3.1.3 Die Ebenen im Einzelnen

3.1.3.1 S-Ebene

Eine Zelle einer S-Ebene ϵ der Stufe σ hat die Ausgabe $S_{\sigma,\epsilon}$. Innerhalb der Ebene wird sie durch einen zweidimensionalen Ortsvektor $\vec{r}$ lokalisiert; dieser bezieht sich allerdings auf das Zentrum des ihr zugeodneten Fokusbereichs[1]. Sie erhalten Signale aus den C-Ebenen der vorhergehenden Stufe ($l \in Schicht(C_{\sigma-1})$), allerdings nur von jenen C-Zellen, die zu einem Bereich mit Radius $FokusR(S_\sigma)$ um die Zentren $\vec{r}$ des Hyper-Fokusbereiches gehören. Die Gewichte dieser Verbindungen sind die ${}^{cs}\gamma$. Die Notation für die Verbindungen der V-Zellen ist gleich. Jede S-Zelle erhält aber nur den Ausgang genau einer ihr zugeordneten V-Zelle $V_{\sigma,\epsilon}(\vec{r})$. ${}^2k$ ist eine

[1]Erinnerung: die Ebenen im Neocognitron erhalten die Nachbarschaftsrelationen ihrer Quellen.

Konstante, die den Start des Sättigungsbereiches in der E/A-Charakteristik einer S-Zelle festlegt. Die Stärke des inhibitorischen Signals der V-Zellen wird durch ${}^1k$ bestimmt.

$$S_{\sigma,\epsilon}(\vec{r}) = {}^1k_\sigma \cdot \phi \left(\frac{{}^2k_\sigma + \sum_{l \in Schicht(C_{\sigma-1})} \sum_{\|\vec{d}\| \in [0, FokusR(S_\sigma)]} \left\{ {}^{cs}\gamma_{l,\sigma,\epsilon}(\vec{d}) \cdot C_{\sigma-1,l}(\vec{r}+\vec{d}) \right\}}{{}^2k_\sigma + \frac{{}^1k_\sigma}{1+{}^1k_\sigma} \cdot {}^{vs}\gamma_{\sigma,\epsilon}(\vec{r}) \cdot V_{\sigma,\epsilon}(\vec{r})} - 1 \right) \tag{3.1}$$

$$\phi(x) = \begin{cases} x & \text{falls } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{3.2}$$

3.1.3.2 C-Ebene

Sieht man von dem bisher aus den Betrachtungen ausgeschlossenen zweiten Term in Gl. 3.3 ab, so erhält eine C-Zelle ihre Eingaben ausschließlich aus der ihrer Ebene zugeordneten S-Ebene der gleichen Stufe. Jener Term eröffnet dem Trainer (für den Fall des überwachten Lernens) die Möglichkeit, die Detektion eines Features auf zwei S-Ebenen zu verteilen. Gibt es in dem Trainingssatz beispielsweise ein stark deformiertes Feature, das vom Netz nicht korrekt erkannt wird, so kann der Trainer dieses zusätzlich in die Netzstruktur aufnehmen. Deformiertes und nicht deformiertes Feature werden dann mittels ${}^{konv}\gamma$ einer gemeinsamen C-Ebene zugeführt. Sie werden in der nächsten Stufe dann nicht als verschiedene Features gewertet. Fast alle der ${}^{konv}\gamma$ sind dementsprechend Null. Die Gewichte ${}^{sc}k(\|\vec{r}\|)$ sind nicht veränderlich; sie sind Punkte einer streng monoton fallende Funktion von $\|\vec{r}\|$. Die Funktion Ψ spezifiziert den Verlauf der Sättigung der C-Zellen. Die übrige Notation wurde schon in Gl. 3.1.3.1 erläutert.

$$C_{\sigma,\epsilon}(\vec{r}) = \Psi \left(\sum_{\|\vec{d}\| \in [0, FokusR(C_\sigma)]} \left\{ {}^{sc}k_\sigma(\|\vec{d}\|) \cdot S_{\sigma,\epsilon}(\vec{r}+\vec{d}) \right\} + \sum_{\|\vec{d}\| \in [0, FokusR(C_\sigma)]} \sum_{l \in Schicht(S_\sigma)} \left\{ {}^{konv}\gamma_\sigma(\epsilon, l) \, {}^{sc}k_\sigma(\|\vec{d}\|) \cdot S_{\sigma,l}(\vec{r}+\vec{d}) \right\} \right)$$

$$\text{wobei} \qquad {}^{konv}\gamma_\sigma(\epsilon, \epsilon) = 0 \tag{3.3}$$

$$\Psi(x) = \frac{\phi(x)}{1+\phi(x)} \tag{3.4}$$

3.1.3.3 V-Ebene

Die V-Zellen Eingänge sind mit denselben C-Zellen verbunden, wie die der V-Zelle zugeordnete S-Zelle. V-Zellen senden ein inhibitorsiches Signal an S-Zellen, das

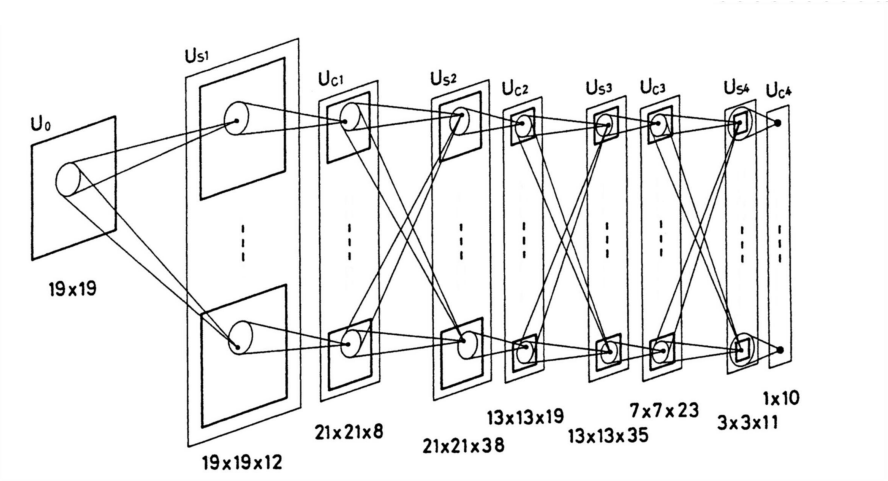

Abbildung 3.5: Topologie eines von Fukushima entworfenen Neocognitrons [Fuk88]. Die V-Schichten sind der Übersicht wegen weggelassen. Die Stufen sind anhand der Indizes zu erkennen.

dem gewichteten RMS[2] der eintreffenden C-Signale entspricht. Es hat Normierungsfunktion. Ihre Gewichte sind ebenfalls nicht variabel.

$$V_\sigma(\vec{r}) = \sqrt{\sum_{l \in Schicht(C_{\sigma-1})} \sum_{\|\vec{d}\| \in [0, FokusR(S_\sigma)]} {}^{cv}k_\sigma(\vec{d}) \cdot \left\{ C_{\sigma-1,l}(\vec{r}+\vec{d}) \right\}^2} \tag{3.5}$$

$$\text{mit} \qquad \sum_{l \in Schicht(C_{\sigma-1})} \sum_{\|\vec{d}\| \in [0, FokusR(S_\sigma)]} {}^{cv}k_\sigma(\vec{d}) = 1 \tag{3.6}$$

Zusammenfassung Das Neocognitron klassifiziert Eingangsdaten durch eine Abfolge von Stufen gleicher Funktionsweise. Jede Stufe extrahiert Features aus den Ausgangsdaten der vorhergehenden Stufe und bildet daraus eine komprimierte Darstellung. Diese Darstellung erhält die Nachbarschaftrelationen der extrahierten Features und wird an den Eingang der nächsten Stufe weitergegeben. Klassifikation wird erreicht durch die stetige Extraktion und Kompression solange, bis der Ausgang einer Stufe auf einen Vektor reduziert wurde. Dessen Elemente stellen Maße der Ähnlichkeit vom Eingabemuster zu den Repräsentanten der verschiedenen Klassen dar. Diese Kompression ist verlustbehaftet. Sie entspricht einer Abstraktion der Eingangsdaten bis hin zur bloßen Benennung. Diese Abstraktionsfähigkeit beinhaltet eine gewisse Unempfindlichkeit gegen Verschiebungen und Deformationen [Fuk88],[LDT95].

[2] **R**oot **M**ean **S**quare.

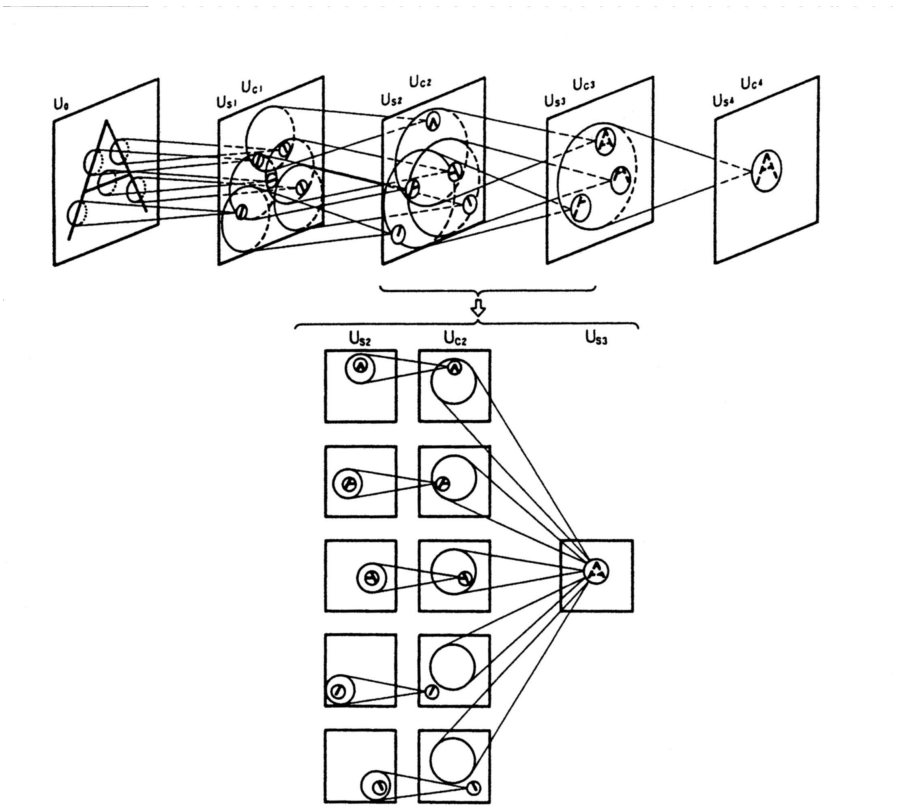

Abbildung 3.6: Die Abbildung zeigt die Funktionsweise des Neocognitrons [Fuk88]. Es werden in den höheren Stufen nicht die Fokusbereiche dargestellt. Gezeigt sind die diesen Bereichen logisch zugehörigen Bereiche aus dem Netzeingangsbild.

3.1.4 Das Training

Im Folgenden wird das Training bei fester Netzdimensionierung und übriger Parameter k betrachtet. Auf welche Art und Weise Herr Fukushima zu der von ihm gewählten Anzahl von Stufen etc. gelangte, ist meines Wissens nicht dokumentiert. Dieses Problem wird in Kap. 4.3 für unseren Ansatz gelöst. Es ist einer der wesentlichen Punkte auf dem Weg zu höherer Erkennungsrate [LDT95]. Nur die Eingänge der S-Zellen besitzen variable Gewichte; im Folgenden vorgestellte Trainingsmethoden modifizieren ausschließlich diese.

3.1.4.1 Unüberwachtes Lernen

Die Methode des unüberwachten Lernens entspricht der Methode der Selbstorganisation, die den Anreiz zur Entwicklung des Neocognitrons gab [Fuk75]. Dabei werden Verbindungen verstärkt, falls die Empfängerzelle an ihrer Position[3] am stärksten feuert und die Senderzelle ebenfalls aktiv ist. Die Modifikation ist proportional zur Aktivität der Senderzelle. Sowohl die erregenden Zuleitungen einer S-Zelle werden nach diesem Prinzip verändert, als auch die hemmenden Verbindungen, die von V-Zellen ausgehen. Die erregenden Leitungen wachsen hierbei derart, daß sie die räumliche Verteilung der Aktivität bei der Vorlage eines Bildes (oder Bildfragmentes - Feature) um so besser speichern, je öfter dieses vorgelegt wurde. Wird ein nicht relevantes Feature präsentiert, überwiegen die inhibitorische Signale der V-Zellen. *Die Feinabstimmung zwischen V- und S-Zellen ist also substantiell für den Erfolg des Neocognitrons* [LDT95]. Die Zelle mit dem stärksten Ausgangsignal bestimmt aber nicht nur über die Modifikation ihrer Zuleitungen, alle Zellen derselben Ebene erhalten die gleiche Behandlung. Dadurch wird sichergestellt, daß ein Feature an verschiedenen Positionen gleichermaßen erkannt wird. Die genaue Lernvorschrift zeigen Gl. 3.7 und 3.8. Dabei bezeichnen die fettgedruckten Größen die der am stärksten feuernden Zelle und ${}^3k_\sigma$ eine positive Konstante - sie entscheidet über die Geschwindigkeit der Gewichtsanpassungen.

$$\Delta^{cs}\gamma_{e,\boldsymbol{\sigma},\boldsymbol{\epsilon}}(\vec{d}) = {}^3k_{\boldsymbol{\sigma}} \cdot {}^{cv}\gamma_{\boldsymbol{\sigma}}(\vec{d}) \cdot C_{\boldsymbol{\sigma}-1,e}(\vec{\boldsymbol{r}} + \vec{d}) \tag{3.7}$$

$$\Delta^{vs}\gamma_{\boldsymbol{\sigma},\boldsymbol{\epsilon}} = {}^3k_{\boldsymbol{\sigma}} \cdot V_{\boldsymbol{\sigma}}(\vec{\boldsymbol{r}}) \tag{3.8}$$

3.1.4.2 Überwachtes Lernen

Beim überwachten Lernen entscheidet der Trainer bei jedem Eingangsmuster darüber, welche Zellen die Gewinnerzellen sein sollen. Der übrige Lernvorgang ist mit der unüberwachten Methode identisch (Kap. 3.1.4.1). Trainiert wird stufenweise.

[3] Genauer: es gibt keine aktivere Zelle an ihrer Position in den verschiedenen Ebenen Ihrer Schicht **und** keine Zelle Ihrer Ebene feuert stärker und erfüllt ebenfalls vorgenannte Bedingung.

3.2 Die Abbildung auf ein hybrides Netz wie HAGEN

Das Prinzip des Neocognitrons, wie es durch die Gleichungen 3.1 bis 3.8 beschrieben wurde, basiert auf der Verwendung analoger (bzw. nicht binärer) Neuroneneinund ausgänge. So sind beispielsweise das Verhältniss erregender C-Zellen zur hemmenden V-Zelle und die damit verbundene Einstellung der Parameter ${}^1k$ und ${}^2k$ wesentlich für die Erkennungsleistung des Netzes [LDT95]. Lovell stellt in diesem Artikel SHOP[4] vor. SHOP ist ein Algorithmus zur automatischen Suche obengenannter Parameter. Gegenüber Fukushimas handoptimierter Version [FW91] steigert Lovell die Erkennungrate damit um ca. 20%. Ein binäres inhibitorisches Ausgangssignal der V-Zellen macht keinen Sinn - eine Normierung hiermit ist nicht möglich. Auch der Wertebereiche der S-Zellen-Ausgänge ist nicht ohne Bedacht kontinuierlich gewählt. Sie sollen die Güte der Übereinstimmung zwischen Fokusbereich und Feature kontinuierlich wiedergeben. Werden beispielsweise die kontinuierlichen Ausgänge der letzten S-Ebenen einem mehrschichtigen Perzeptron zugeführt und dieses auf reale Daten trainiert, so steigt die Zuverlässigkeit des Netzes um etwa 10% [LDT95]. Der HAGEN-Chip bietet allerdings lediglich binäre Ein- wie auch Ausgänge.

Integerneurone Vorstellbar ist deshalb die näherungsweise Darstellung kontinuierlicher Werte im Dualsystem. Bei beispielsweise 8-bit Auflösung stehen mit einem HAGEN-Block effektiv sechzehn 8-bit Integer-Eingänge zur Verfügung (die Rechnung vernachlässigt die mindere Qualität der Randneurone und eventuell benötigte Rückkopplungsleitungen). Mit einer Fokusbereichsgröße von $n_{Ebenen} \cdot 3 \times 3$ könnte ein HAGEN-Block dann lediglich die erste Stufe mit der Ebenenanzahl eins bearbeiten. In höheren Stufen ist die Anzahl der Ebenen gleich der Anzahl der Features der Vorgängerstufe. Sogar eine 2-bit Auflösung liesse maximal lediglich sieben Features in der niedrigsten Stufe zu - verglichen mit Fukushimas Entwurf von bis zu 35 Features (Abb. 3.5) sicher zu wenig. Die Kopplung mehrerer Hagen-Blöcke ist zur Zeit aber noch nicht möglich.
Grundsätzlich muß man überdies bedenken, daß die Simulation kontinuierlicher Eingänge der Hardware HAGEN nicht angemessen ist. Der daraus resultierende zusätzliche Ressourcenverbrauch hebt jeden Vorteil gegenüber Simulationen via Standard-PC auf. Prozessoren herkömmlicher Computer arbeiten immerhin schon auf 32-bit Basis und mit Taktraten im GHz-Bereich. So leistet der HAGEN-Chip 400MCUPs (S.10), die Netzgewichte können jedoch nur blockweise zum Chip übertragen werden - eine Gewichtskonstellation kann somit mit maximal etwa 12,2kHz geschrieben werden. Die Anzahl der pro Trainingsschritt modifizierten Gewichte ist typischerweise im Bereich von 5-20%, außerdem kann ein HAGEN-Block maximal sechzehn 8-bit Eingänge und acht 8-bit Ausgänge (die im Neocognitron

[4] **S**electivity Hunting to **O**ptimize **P**erformance.

mit einem HAGEN-Block nicht einmal genutzt werden könnten s.o.) darstellen. Eine PC-Simulation gleicher Auflösung müsste in jedem Trainingsschritt dementsprechend etwa 26 Gewichte ändern. Ein PentiumIV kann dies einige Millionen mal in der Sekunde. Das Training auf diese Weise genutzter HAGEN-Blöcke wäre somit vergleichsweise ineffizient.
Gleiches gilt für die Simulation eines HAGEN-Zyklus in dieser Situation. Diese würde sich auf 16·8 Additionen und Multiplikationen beschränken - dies entspricht etwa $16 \cdot 8 \cdot 4 = 512$ Taktzyklen eines PentiumPro-Klasse Prozessors. Aktuelle PentiumIV Prozessoren sind etwa 40fach höher getaktet als HAGEN und bearbeiten im Mittel etwa drei RISC[5]-Befehle gleichzeitig. Die Netzfrequenz von HAGEN wäre so nur marginal größer als die der Simulation.
Schon ein $(n_{Ebenen} = 1) \cdot 3 \times 3$ Fokusbereich in der ersten Stufe des Neocognitrons benötigte außerdem 72 Eingänge. Mit der derzeitigen Begrenzung auf einen HAGEN-Block wäre nur ein Feature je Neocognitron-Stufe möglich.
Um der Leistungsfähigkeit des HAGEN-Chips gerecht zu werden, sind somit binäre Ein- und Ausgaben unumgänglich. Der Versuch, die Neocognitron-Struktur derart anzupassen, birgt das Risiko, die Funktionsweise zu beeinträchtigen oder gänzlich zu zerstören. Lovell hat beispielsweise 1992 die Auswirkungen verschiedener Transferfunktionen für S-Zellen untersucht. Ersetzte er die für das Neocognitron übliche Schwelle-Linear-Funktion (Gl. 3.2) durch eine Stufenfunktion, wie sie den Neurone des HAGEN-Chips entspricht, verlor das Netz etwa 20% Erkennungsrate [LT92]. Seine Ausführungen lassen alles in allem Erkennungsraten weit über 30% als unwahrscheinlich erscheinen. Als eigenständiges Handschrifterkennungssystem wird das digitale Neocognitron kaum konkurrenzfähig sein, seine Stärken in Verbindung mit dem HAGEN-Chip liegen in der Verarbeitung großer Datenmengen. Beispielsweise die schnelle Vorselektion oder die hierarchische Echtzeitglättung großer Bilder könnten sinnvolle Einsatzgebiete sein. An den in diesem Abschnitt angedeuteten Erwägungen orientiert sich die im folgenden vorgestellte Abbildung des Neocognitrons auf den HAGEN-Chip.

Die gewählte Abbildung auf HAGEN Die zur Abbildung des Neocognitrons auf HAGEN notwendigen Modifikationen werden im Folgenden beschrieben.
C-Ebenen erhalten ihren Eingabe aus Fokusbereichen der zugeordneten S-Ebene vorhergehender Stufe (die Zusammenfassung von Ebenen zu Stufen ist umgeordnet, die Reihenfolge von C- und F/S-Ebenen ändert sich dadurch nicht, s.h. Abb. 3.7). Weil C-Zellen nur entscheiden, ob Aktivität in dem zugehörigen Fokusbereich vorhanden ist, sind ihre Zuleitungen ungewichtet. C-Zellen sind Schwellwertneurone mit der Schwelle eins. Von der Idee Fukushimas weichen sie insofern ab, als daß sie die Güte der Aktivität nicht berücksichtigen. Aktivität im Zentrum des Fokusbereiches wird genauso bewertet wie Randaktivität, geringe Intensität genauso wie maximale Intensität. Entweder ein versetztes Feature wird voll oder

[5] **R**educed **I**nstruction **S**et **C**omputer.

gar nicht erkannt. Inwieweit dadurch die Erkennungsleistung herabgesetzt wird, kann an dieser Stelle nur gemutmaßt werden. In Kap. 4.1.2 wird überprüft, ob Beeinträchtigungen nach der ersten Stufe erkennbar sind. Die C-Ebenen werden innerhalb dieser Arbeit in Software implementiert. Die Abbildbarkeit auf HAGEN ist offensichtlich, eine Überprüfung nicht notwendig - sie würde die Komplexität des Aufbaus unnötig belasten.
Die S-und V-Ebenen werden ersetzt durch ein dreischichtiges Perzeptron (F/S-Schicht genannt). Da es verschiedene Konventionen zur Schichtnummerierung gibt, sei hier erwähnt, daß es sich um eine Topologie mit genau einer Zwischenschicht handelt. Die Verwendung dreier Schichten ist begründet durch die bereits vorhandene Infrastruktur für einen zwei Zyklenbetrieb des HAGEN-Chips. Alle vorab gemachten Tests, ein dreischichtiges Perzeptron auf Feature Sätze zu trainieren, waren ausreichend erfolgreich. Die Verwendung zusätzlicher Schichten ist jedoch grundsätzlich ohne Abwandlung des vorgestellten Prinzips möglich.
Im ersten Zyklus ist ein Teil der HAGEN-Ausgänge der zweiten Perzeptronschicht zugeordnet. Ihre Aktivitäten werden im ersten Zyklus evaluiert. Sie werden anschließend auf unbelegte HAGEN-Eingänge zurückgeführt. Im zweiten Zyklus werden alle benutzten HAGEN-Eingänge - sowohl die Perzeptroneingänge, wie auch die Ausgänge der Perzeptronschicht zwei (die rückgekoppelten Eingänge) - den Ausgängen des HAGEN-Chips zugeleitet. Im zweiten Zyklus stellen letztere die Perzeptronausgänge dar. Es existieren demnach Direktverbindungen von Schicht eins (den Perzeptroneingängen) zur Schicht drei (den Perzeptronausgängen) und indirekte Verbindungen über die Schicht zwei (zu deren Berechnung HAGEN-Rückkopplungsleitungen benutzt werden). Die Direktverbindungen werden in beiden Zyklen gleichermaßen behandelt, jedoch nur die Ausgänge des zweiten Zyklusses werden als Perzeptronausgänge verwendet.
Die Perzeptroneingänge sind mit den C-Zellen im aktuellen Hyper-Fokusbereich der gleichen Stufe verbunden. Der Fokusbereich wird schrittweise verschoben. In Abb. 3.7 wird dies durch das Pfeilkreuz am HAGEN-Block angedeutet. Betrachtet man den Hagen-Block und seine Zu- und Ableitungen als starres Gerüst, so deuten die Pfeilrichtungen seine Verschiebung an. Sie sind parallel zur Verschiebung des Hyper-Fokusbereiches in der C-Schicht der Vorgängerstufe. Diese Verschiebung geschieht in dem vorgestellten Aufbau via Software. Dabei ändern sich zwar die Eingangssignale, die Perzeptrongewichte jedoch nicht. Somit können einem mit entsprechenden Gewichten geladenen HAGEN-Block sequentiell die Eingangssignale verschiedener Hyper-Fokusbereiche vorgelegt werden. Die HAGEN-Ausgangssignale gehen an die entsprechenden S-Zellen derselben Stufe (Abb. 3.7). Die in dieser Arbeit umgesetzte sequentielle Präsentation der Fokusbereichsdaten kann später durch den Einsatz mehrerer Chips gleicher Gewichtskonstellation parallelisiert werden. Das durch HAGEN vertretene dreischichtige Perzeptron wird auf die Klassifikation der Fokusbereichseingangsdaten trainiert. Dazu wird jedem Feature (Eingangsdaten) eine Klasse in Form genau eines aktiven Ausgangsneu-

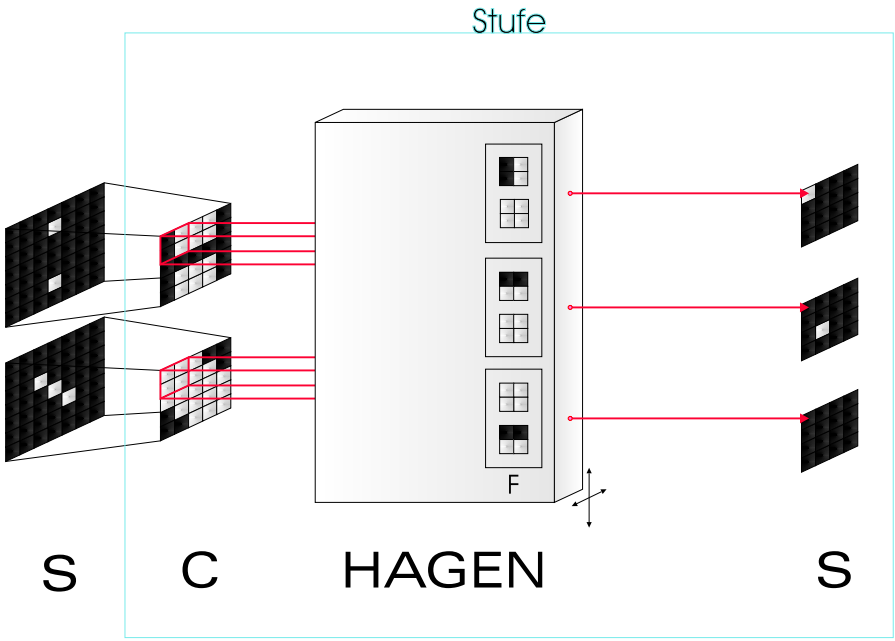

Abbildung 3.7: Eine Stufe des in dieser Arbeit aufgebauten digitalen Neocognitrons.

rons (Ausgangsdaten) zugeordnet. Wird später Feature Nummer Eins erkannt, soll nur das erste Ausgangsneuron aktiv sein. Werden Feature zwei und drei erkannt, werden S-Zellen aus den folgenden S-Ebenen zwei und drei mit Signalen gefüttert. Die Features sind entsprechend den Hyper-Fokusbereichen Hyperfeatures. In dem abgebildeten Beispiel ist das Netz trainiert auf die Erkennung dreier Hyperfeatures. Jedes dieser Features besteht aus zwei Ausschnitten - ebenso wie auch ein Hyperfokusbereich Daten aus zwei vorhergehenden C-Ebenen erhält.

3.2.1 F/S-Ebene

Die Indizierung in den Gleichungen 3.9–3.13 unterscheiden sich von denen der Gleichungen 3.1–3.8. Die Stufen wurden umgeordnet. Eine nullte Stufe gibt es nicht mehr. Die erste C-Schicht gehört nun zur Stufe eins, entsprechend die zweite zur Stufe zwei. Eine Stufe besteht aus C- und F/S-Schichten in dieser Reihenfolge. Die Umordnung wurde aus Konsistenzgründen mit dem Programm vorgenommen. Der Teil des Programmes, der die Netzstruktur bestimmt ist, in Mathematica[6] geschrieben - in Mathematica beginnen die Indizes bei Eins (Kap. 4.1).
Die F/S-Schichten sind durch dreischichtige Perzeptrons realisiert, die auf bestimmte Sätze Features trainiert wurden. Die Ausgabe $S_{\sigma,\epsilon}(\vec{r})$ der S-Zelle der Schicht σ in der Ebene ϵ am Ort $\vec{r}$ ist wegen der Stufenfunktion der HAGEN-

[6]Eine Entwicklungsumgebung der Firma Wolfram.

Neurone binär. Die Anzahl $\#\mathcal{Z}$ der Neurone der Zwischenschicht $\mathcal{Z}$ ist beliebig wählbar.
Die Verwendung von vier Rückkopplungen hat sich aber als vernünftig erwiesen. Jede Rückkopplung belegt HAGEN-Eingänge. Eine größere Anzahl von Zwischenneuronen bedingt demnach eine stärkere Begrenzung der Anzahl verwendbarer Features - jedem Feature der Vorgängerstufe ist genau eine S-Ebene zugeordnet. Die Menge aller Ausschnitte gleichen Ortes in den verschiedenen S-Ebenen einer S-Schicht bilden die Fokusbereiche - die Eingaben in den HAGEN-Chip. Zusätzlich ist es wünschenswert, die Verwendung der Randneurone des HAGEN-Chips zu vermeiden. Diese sind minderer Quälität. Da ein HAGEN-Block nur 128 Eingänge besitzt, ist die Anzahl Features sowieso schon unangenehm beschränkt, zusätzliche Beschränkungen sind demnach möglichst klein zu wählen.
Die Notation ist im gesamten Kapitel 3.2 konsistent gehalten und wurde bereits erklärt. Ich möchte aber an dieser Stelle noch darauf hinweisen, daß die Gesamtheit aller Gewichte γ_σ eine HAGEN-Gewichtskonstellation bildet, im Kontext genetischer Algorithmen bezeichnet man sie auch als *Individuum*.

$$S_{\sigma,\epsilon}(\vec{r}) = \theta\left(\mathcal{Z}_{\sigma,\epsilon}(\vec{r}) + \sum_{l \in Schicht(C_\sigma)} \sum_{\|\vec{d}\| \in [0,FokusR(S_\sigma)]} \left\{ {}^{cs}\gamma_{\sigma,\epsilon}(\vec{r}+\vec{d}) \cdot C_{\sigma,l}(\vec{r}+\vec{d}) \right\}\right) \tag{3.9}$$

$$\mathcal{Z}_{\sigma,\epsilon}(\vec{r}) = \sum_{m=1}^{\#\mathcal{Z}} \left\{ {}^{zs}\gamma_{m,\sigma,\epsilon} \cdot \theta\left(\sum_{l \in Schicht(C_\sigma)} \sum_{\|\vec{d}\| \in [0,FokusR(S_\sigma)]} {}^{cz}\gamma_{\sigma,l,m}(\vec{d}) \cdot C_{\sigma,l}(\vec{r}+\vec{d}) \right) \right\} \tag{3.10}$$

$$\theta(x) = \begin{cases} 1 & \text{falls } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{3.11}$$

3.2.2 C-Ebene

Der $\vec{r}$ der C-Zellen $C_{\sigma,\epsilon}$ in der Ebene ϵ der Stufe σ wird, wie auch für die S-Zellen gültig, aus der Position der linken oberen Ecke des ihr zugeordneten Fokusbereiches bestimmt. Der C-Zelle $C(\binom{2}{1})$ ist demnach der um eine Position nach rechts verschobene Fokusbereich, ausgehend von der linken oberen Ecke, zugeordnet. Ist mindestens eine S-Zelle in diesem S-Ebenen-Ausschnitt aktiv, so feuert $C(\binom{2}{1})$.

$$C_{\sigma,\epsilon}(\vec{r}) = \theta_2\left(\sum_{\|\vec{d}\| \in [0,FokusR(C_\sigma)]} S_{\sigma-1,\epsilon}(\vec{r}+\vec{d}) \right) \tag{3.12}$$

$$\theta_2(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{sonst} \end{cases} \tag{3.13}$$

3.2.3 L-Schicht

Die Menge aller Bilder, die die Ziffer Eins darstellen, enthält sowohl paarweise ähnliche wie auch paarweise konträre Bilder. Beispielsweise mögen die Einsen mit Rechtsneigung untereinander sehr ähnlich sein, eine linksgeneigte Eins wird sich hingegen schlecht in die Menge rechtsgeneigter Einsen einordnen. Die letzte Schicht des Neocognitrons (L-Schicht) baut im Gegensatz zu den S-Schichten auf einem Feature-Satz auf, der in zusammengehörige Features, sogenannte Feature-Gruppen, eingeteilt ist. Alle Features einer Feature-Gruppe repräsentieren dabei dieselbe Klasse. Eine Feature-Gruppe könnte beispielsweise aus zwei Features bestehen: jeweils einem Repräsentant linksgeneigter und rechtsgeneigter Einsen. Beide Repräsentanten reagieren zwar auf verschiedene Eingangsbilder, ihre Antwort muß jedoch diesselbe Klassifikation zur Folge haben. Eine Möglichkeit eine solche Klassifikationsschicht zu bewerkstelligen, wäre jedem Bild eine Klasse zuzuordnen und die zusammengehörigen Klassen anschließend zu Bündeln. Der Trainingserfolg dieser Variante ist jedoch bescheiden. Zweien nahezu identischen Eingangsbildern wird ein könträres Ausgangsmuster zugewiesen. Diese Bilder können nicht befriedigend gelernt werden. Das Konzept der S-Schichten ist für die letzte Stufe ungünstig.
Neben den eindeutig einem Feature aus der Feature-Gruppe zuordbaren Eingangsbilder wird es auch Bilder geben, die mehreren Features ein wenig entsprechen. Die Abgrenzung der Features einer Feature-Gruppe untereinander sollte daher verschwimmen. Wird vorausgesetzt, daß es maximal dieselbe Anzahl konträrer Features innerhalb einer Feature-Gruppe gibt wie Trainingsbilder vorgelegt werden, scheint es sinnvoll, diese Zahl Ausgangsbits zur Repräsentation der entsprechenden Klasse anzulegen. Jedes dieser Bits entspricht dann einer Eigenschaft des einer Feature-Gruppe zugehörigen Features. Dabei sind Redundanzen erlaubt. Mehrere Bits dürfen gemeinsam aktiv sein. Eine nicht geneigte Eins könnte beispielsweise sowohl als links- wie auch als rechtsgeneigt eingestuft werden. Um so mehr Bits einer Feature-Gruppe aktiv sind, um so mehr Features dieser Gruppe ähneln dem Eingangsbild. Die Abgrenzung gegenüber anderen Klassen erfordert dementsprechend, daß möglichst wenige Bits fremder Feature-Gruppen aktiv sind.
Das in dieser Arbeit verwendete Kriterium welcher Klasse ein Eingangsbild zugeordnet wird, erwartet, daß die Anzahl gesetzter Bits der gewünschten Feature-Gruppe gegenüber aller anderen Feature-Gruppen um mindestens eins größer ist. Sobald eine fremde Feature-Gruppe mehr Aktivität aufweist, gilt das Bild als falsch klassifiziert. Sind diese beiden Bedingungen nicht erfüllt, wird das Eingangsbild nicht klassifiziert. Andere Kriterien sind möglich, konnten im zeitlichen Rahmen jedoch nicht genauer untersucht werden.

Kapitel 4

Konstruktion des binären Neocognitrons

4.1 Aufbau der Grundstruktur

4.1.1 Die Entwicklungsumgebung

Für den schnellen Aufbau eines Erstsystems mit Prototypcharakter sind folgende an die Entwicklungsumgebung gestellte Anforderungen prioritär:

- Bereitstellung von *high-level*[1]-Befehlen, um die Lösung von Detailfragen auf die Schlussphase zu verlagern (dies ermöglicht die Vermischung induktiver mit deduktiver Entwicklungsmethoden und verkürzt die Planungsphase drastisch).

- Flexibilität in der Entwicklungsphase.

  - Eine unkomplizierte Testumgebung.

    * Bereitstellung von Visualisierungshilfmitteln.
    * Kurze Testzeiten, beispielsweise Vermeidung langer Übersetzungszeiten.
    * Umfangreiche Fehlerdiagnosen seitens der Entwicklungsumgebung.

  - Einfache Programmstruktur mit weitestmöglicher Unabhängigkeit der Programmelemente untereinander - schnelle Änderungen werden so möglich.

- Vertrautheit mit dem System.

[1] Befehle, die komplexe Operationen wie beispielsweise *Datei einlesen* ausführen.

Mathematica ist eine Skriptsprache der Firma *Wolfram Research* [Wol]. Skriptsprachen[2] sind Sprachen, die zur schnellen Bewältigung kleinerer Aufgaben entwickelt wurden. Sie werden zumeist interpretiert, also zur Laufzeit übersetzt, und erlauben deshalb Befehlskonstrukte, die ein Compiler[3] vor Laufzeit nicht zuordnen kann. Im Kern ist Mathematica ein Computeralgebrasystem mit starken symbolischen Methoden zur analytischen Lösung von Gleichungen, überdies beinhaltet es eine umfangreiche Numerikbibliothek. Heute ist Mathematica weit mehr: es bietet eine vollständige Programmierumgebung, Visualisierungshilfmittel, Anbindung an andere Hochsprachen wie C und zahlreiche Pakete mit Lösungen spezieller Aufgaben. Der hohe Abstraktionsgrad und Umfang dieses Systems erfordern eine lange Einarbeitungszeit. Da mir dieses System bereits bekannt war und es obengenannte Anforderungen an die für diese Arbeit benötigte Entwicklungsgeschwindigkeit weitestgehend erfüllt, habe ich mich entschieden, große Teile des Systems in Mathematica zu verfassen. Zur fairen Beurteilung dieser Wahl, möchte ich nun auch die Nachteile nennen. Es sind die enorme Komplexität des Basissystems (Mathematica vereint Eigenschaften aus allen gängigen Hochsprachen, beispielsweise Listen, Operatorüberladung oder funktionale Programmierung mit einer interaktiven Oberfläche. Die Sprache ist Kernel-basiert[4]), die Fehlerdiagnosemöglichkeiten sind unzureichend und mit der Möglichkeit der kryptischen Programmierung werden große Projekte unübersichtlich. Der Ressourcenverbrauch Mathematicas ist überdies ebenso gewaltig wie die Abstraktion diesen Systems. Resummierend lässt sich bestätigen, daß der Umfang dieser Arbeit für mich und in der Kürze der Zeit mit keinem anderen System zu bewältigen gewesen wäre. Dies lässt sich bei einem Blick auf Abbildung 4.18 erahnen.
Der Programmierteil dieser Arbeit lässt sich in die Entwicklung folgender Programmstrukturen einteilen:

- Einer übergeordneten Instanz, die Daten extrahiert und den übrigen Instanzen zuführt, außerdem deren Steuerung übernimmt; ihr Name ist BONNEE (Basic Optimizing Neural Network Evolution Environment). Sie enthält hauptsächlich folgende Algorithmen:
  - In ihr sind vier selbst entwickelte Bibliotheken eingebunden, die beispielsweise den Zugriff auf die MNIST-Zifferndatenbank ermöglichen (Kap. B).
  - Die grundlegenden Neocognitron-Operationen wie beispielsweise die Extraktion der Fokusbereichsdaten.
  - Die Mathematica-Seite der MathLink-Steuerung von HAGEN sowie SHAGEN (s.h. unten).

[2] Der Begriff ist eng verknüpft mit besonders kryptischen Befehlskonstrukten.
[3] Übersetzerprogramm, das Hochsprachenbefehle in Maschinensprache übersetzt.
[4] Gegenteil von monolithisch.

  - Ein wesentlicher Teil von BONNEE ist die Parameterschleife (Kap. 4.3.1) zur automatischen Netzstrukturoptimierung.
  - Experimentelle Algorithmen wie beispielsweise die Vergleichsebenen (Kap. 4.2.5).
  - Funktionen zur Messung der Erkennungsrate und zur Darstellung der Resultate.

- Der Ansteuerung des HAGEN-Chips; sie besteht aus dem bereits bestehenden C++ Programm HANNEE (Kap. 2.6) und einer diesem aufgesetztem MathLink[5]-fähigen Schnittstelle zur Kommunikation mit BONNEE, aber auch mit anderen C-Programmen.

- Dem SHAGEN-Paket (Soft HAGEN) (Kap. 4.3.2); es umfasst eine C-Softwaresimulation des Chips, einen in Assembler implementierten evolutionären Trainingsalgorithmus und eine MathLink-Schnittstelle zu BONNEE.

- Den Verteilungs- und Differenzselektoren (Kap. 4.2.2 und 4.2.3) zur automatischen Extraktion und Selektion von geeigneten Feature-Sätzen - diese stellen die eigentlichen Trainingseingangsdaten für SHAGEN/HAGEN dar.

Die Reihenfolge der in den folgenden Sektionen beschriebenen Programmteile orientiert sich weniger an didaktischen Überlegungen, als an der tatsächlichen Aufbaureihenfolge. Diese Reihenfolge wurde gewählt, um die mit dem Aufbau verbundenen Probleme in den richtigen Kontext zu setzen. Am Ende des Kapitels findet sich jedoch eine geordnete Zusammenfassung der Vernetzung entstandener Programmteile (Kap. 4.4).

4.1.2 Erste Abschätzung der Auflösung nach der C-Ebene

Die C-Zellen in der hier vorgestellten Neocognitron-Modifikation erhalten binäre Eingangssignale und besitzen nur zwei Ausgangszustände: 0 und 1. Die stetige Verschmierungseigenschaft der C-Schichten des Originals wird ersetzt durch ein Alles-oder-Nichts Prinzip. Verschiedene Eingangsmuster können somit als identisch erkannt werden. Diese Möglichkeit ist in Fukushimas Neocognitron stark reduziert. Ein versetztes Feature wird zwar erkannt, allerdings wird es als weniger übereinstimmend klassifiziert. Ein kontinuierliches Maß der Übereinstimmung ist jedoch mit binären Signalen nicht möglich. Inwieweit diese Einschränkung die Funktionsweise des Neocognitrons beeinträchtigt, ist nicht klar.
Nach der Implementation der grundlegenden Neocognitron-Operationen wurde deshalb der im Folgenden beschriebene einfache, qualitative Test durchgeführt (Abb. 4.1). Einem zweistufigen Neocognitron wurden die ersten 17000 der 65536 möglichen 4x4 Eingangsbilder nacheinander vorgelegt. Dabei war die

[5]Schnittstelle zu Mathematica.

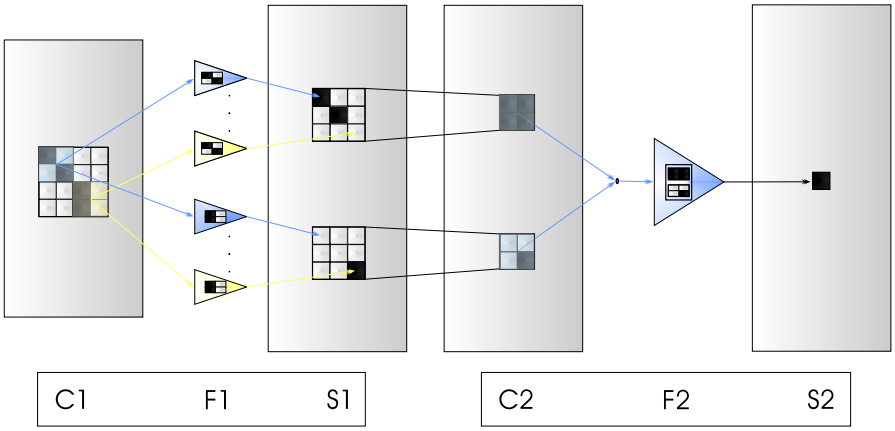

Abbildung 4.1: Der zweistufige Testaufbau zur Überprüfung möglicher Auflösungsverluste in den C-Schichten. Die Eingangsbilder haben das Format 4x4, die zur S1/C2-Schicht gehörenden Fokusbereiche 2x2 und die S2-Hyperfokusbereiche bestehen aus 2·2x2 Bildpunkten. Sind letztere für zwei verschiedene Eingangsbilder identisch, sind diese nach der C2-Schicht nicht mehr trennbar.

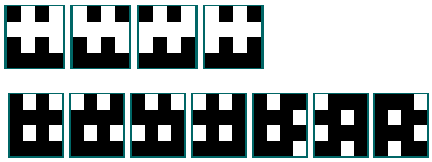

Abbildung 4.2: Die zeilenweise gruppierten 4x4 Eingangsbilder eines minimalen Testaufbaus (Abb. 4.1) besitzen ab der Schicht C2 diesselben Aktivitäten und sind somit vom Netz nicht mehr trennbar.

Neocognitron-Struktur derart, daß die Fokusbereiche der ersten S-Schicht 2x2 Bildausschnitte aus den Eingangsbildern (C1) darstellten. Die C-Schicht der Stufe zwei erhielt ihre Eingangsdaten aus den 2x2 Fokusbereichen der Schicht S1. Die erste F-Schicht (Teil der F/S-Schicht) wurde mit dem Satz aller 16 möglichen 2x2 Features ausgestattet. Die C1-Schicht wurde demnach direkt auf die folgende C2-Schicht durchgeleitet. Da in diesem Aufbau auf die Generalisierungseigenschaft der F/S-Schicht verzichtet wird, ist es einerseits möglich, die Auswirkungen der C-Schicht isoliert zu betrachten, andererseits besitzt das Ergebniss nur qualitativen Charakter. In Abbildung 4.2 sind diejenigen Eingangsmuster gruppiert dargestellt, die ab der C2-Schicht nicht mehr trennbar sind. Eine größere Auswahl findet sich in Abbildung A.1. Die Gruppierungen geben einen ersten Eindruck von der maximalen Auflösung eines Neocognitrons mit binären C-Zellen. Die Gruppierungen erscheinen sinnvoll. Es existieren keine gruppierten Muster, die für das menschliche Auge nicht ausreichende Ähnlichkeit vorweisen. Darüberhinaus zeigen die

Gruppierungen deutlich die durch die C-Schicht bereitgestellte Translationsinvarianz der Features untereinander - kleine Deformationen stören die Erkennung nicht. Ob sich diese Auflösungsverluste jedoch über mehrere Stufen ungünstig summieren und die Trennung unterschiedlicher Eingangsklassen unmöglich machen, lässt sich an dieser Stelle nicht beantworten. Die Verbesserung der Erkennungsleistung liegt ebenfalls im Bereich des Möglichen, deformierte Features werden schließlich ohne Wertminderung erkannt - das Fukushimasche Neocognitron hingegen mindert den Wert deformierter Features.

4.2 Training mittels Feature Selektoren

Die F/S-Schichten des binären Neocognitrons sind die eigentlichen Klassifikationsschichten. Sie ordnen Bildausschnitten Klassen zu, die durch eine Menge zuvor gelernter Features repräsentiert werden. Jede dieser Schichten wird dargestellt durch einen Feature-Satz (F-Schicht) und den auf die Klassifikation dessen Features trainierten HAGEN-Block (S-Schicht). Jedem Feature eines solchen Feature-Satzes wird ein Ausgangsneuron zugeordnet, das feuert, sobald die Eingangsdaten diesem Feature ausreichend ähneln. Eingangsdaten können demnach auch mehreren Feature-Klassen gleichzeitig zugeordnet werden.
Das Training der F/S-Schichten benötigt somit zwei Schritte:

- Die Auswahl geeigneter Feature-Sätze.
- Das Training der HAGEN-Blöcke auf die korrekte Klassifikation der Features.

Die Gruppe *Electronic Vision(s)* hat sich auf evolutionäre Algorithmen spezialisiert. Derartige Algorithmen orientieren sich an dem erstmals von Charles Darwin (1809–1882) erkannten Selektionsschema in der Natur [Met92]. Nur jene Arten überleben, die ihren Daseinsbedingungen am besten angepaßt sind. Von Generation zu Generation werden die Stärksten selektiert. Die Selektion ist ungerichtet, es gibt außer den Umwelteinflüssen keine Entscheidungsinstanz. Schwächere unterliegen im direkten Vergleich mit ihren Konkurrenten oder sind sonstigen Lebensbedingungen nicht gewachsen. Evolutionäre Algorithmen schaffen immer eine Konkurrenzsituation zwischen verschiedenen Lösungsansätzen, den sogenannten *Individuen*. Ein HAGEN Individuum wird repräsentiert durch die Gesamtheit der Netzgewichte; diese bestimmt sein Verhalten vollständig (neben fixen Parametern wie Neuronenschwellen etc.). Entsprechen die übrigen Individuen einer *Generation* eher den Vorgaben, den Eingangsmuster-Ausgangssollmuster-Kombinationen, so fällt dieses Individuum aus der *Population* heraus. Die Population verändert sich dabei von Generation zu Generation. Die Individuen unterliegen Mutationen und anderen genetischen Operationen. Mutationen sind zufällige Änderungen der *Gene* - in unserem Falle der Netzgewichte. Sie sorgen in Verbindung mit der Selektion für die Evolution, die Weiterentwicklung der Population.
Der verwandte genetische Algorithmus wird im Detail beschrieben in Kapitel 4.3.2.

4.2.1 Notwendigkeit der Automatisierung

Lovell macht deutlich, daß die Erkennungsleistung des Neocognitrons entscheidend von der Einstellung der Selektivitätsparameter ${}^1k$ aus Gleichung 3.1 abhängt [LT93]. Der in dieser Veröffentlichung vorgestellte Algorithmus SHOP[6] zur Anpassung dieser Parameter, orientiert sich an den mit den jeweiligen Parametersätzen erzielten Erkennungsraten. Es wird also die Kenntnis der Feature-Sätze bereits vorausgesetzt. Außerdem benötigt Shop eine fest vorgegebene Netztopologie - genau vier Stufen. Weiterhin wird ${}^1k_1 = 1,7$ vorausgesetzt.
Im binären Neocognitron hingegen existieren keine direkten Selektivitätsparameter. Die Selektivität wird indirekt über die Wahl der Feature-Sätze, auf die die S-Schichten trainiert werden, festgelegt. Ohne die Kenntnis der Feature-Sätze ist die Netztopologie eines binären Neocognitrons jedoch nicht festgelegt - sie wird durch die Fokusbereichsgrößen und die Anzahl der Features in den Feature-Sätzen der Stufen gewählt. Die ohnehin wichtige Auswahl der verwendeten Features erhält hier zusätzliche Bedeutung. Ferner sind HAGEN Fokusbereichsgrößen oberhalb des n·4x4-Formats kaum sinnvoll (die Anzahl n der möglichen Features in der vorhergehenden Stufe wäre dann kleiner 5, außerdem ist eine C-Kompression oberhalb von n·3x3 zu verlustreich); da die Ebenen bei konstanten n·4x4-Fokusbereichen pro Stufe um lediglich 2·3 Punkte in beiden Abmessungsrichtungen schrumpfen, wären bei zwei Features in der letzten Stufe bereits vier Stufen notwendig (bei Verwendung des in dieser Arbeit benutzten MNIST Datensatzes). In der Praxis werden etwa sieben Stufen benötigt. Abgesehen davon, daß es unmöglich scheint, im zeitlichen Rahmen dieser Arbeit die Dimensionsparameter und Feature-Sätze für ein siebenstufiges Neocognitron manuell einzurichten, widerspräche dies auch entscheidend dem Ansatz der Datentypunabhängigkeit - diese Prozedur wäre für jeden neuen Datentyp notwendig.

4.2.2 Der Verteilungsselektor

Der Verteilungsselektor ist ein Algorithmus zur automatischen Selektion geeigneter Features aus den Trainingsdaten. Die ihm zugrunde liegenden Ideen sind statistischer Natur:

- Je öfter ein Feature in den Trainingsdaten vorkommt, je besser ist es geeignet diese zu beschreiben.
- Features, die nur in wenigen Bildern einer bestimmten Ziffer vertreten sind, sind zur Beschreibung dieser Ziffer weniger wichtig.
- Für bestimmte Ziffern spezifische Features sind günstig, um diese von den Übrigen zu trennen.

[6] **S**electivity **H**unting to **O**ptimize **P**erformance.

Der Verteilungsselektor wurde in Mathematica implementiert. Er ist Teil der Mathematica-Bibliothek *FeatureExtraction-lib-V2.9.nb* zur Extraktion von Features. Sein Funktionsname ist *selectBestFeaturesOfFList*.

4.2.2.1 Der Algorithmus

Alle zum Trainingssatz gehörigen Bilder werden zuerst nach Zugehörigkeit zu den Ziffern getrennt (Abb. 4.3). Dazu wird Gebrauch gemacht von *Listen*, einer elementaren Mathematicastruktur. Listen im Mathematica Sinne sind geordnete Mengen beliebiger Elemente. Einer sogenannten *Zifferliste* werden alle Bilder einer bestimmten Ziffer zugeordnet. Die *Gesamtliste* umfasst alle diese Zifferlisten. Die Zifferbilder liegen in einem proprietären[7] Format vor [LeC03]. Die hierfür entwickelte Mathematica-Bibliothek trägt den Namen *NIST-lib-V1.5.nb* und stellt Einlese-, Transformations- und Darstellungsroutinen für die Bilder der verwendeten MNIST-Datenbank zu Verfügung. Bilder können hiermit als Listen von Listen von Binärwerten gespeichert werden. Jedes Bild der Gesamtliste wird nun ersetzt durch eine Liste der Inhalte aller seiner Fokusbereiche, der Features (Abb. 4.3). Diese Listen werden *Bildlisten* genannt. Innerhalb der Selektorfunktion sind die Features geordnete Listen binärer Werte, also Vektoren.

Schritt 1 Im ersten Schritt wird die Funktion *dropUnevenlyDistributedLL3* aufgerufen. Sie benötigt zwei Parameter: p_1 und p_2. Beide Parameter beinhalten eine Prozentangabe und sind somit aus dem Wertebereich [0,1]. Die Funktion verwirft nun jene Features, die nur in wenigen Bildern einer bestimmten Ziffer enthalten sind. Ist ein Feature l in $n_{vBilder}(l)$ verschiedenen Bildern enthalten, so wird es verworfen falls:

$$n_{vBilder}(l) \leq Min\{n_{vBilder}\} + [Max\{n_{vBilder}\} - Min\{n_{vBilder}\}] \cdot p_1. \qquad (4.1)$$

Außerdem werden alle Features verworfen, die in der jeweiligen Zifferliste vergleichsweise selten vorkommen ($n_{BgZiffer}$ gibt dabei an, wie oft ein Feature insgesamt, d.h. in allen Bildern einer Ziffer, enthalten ist):

$$n_{BgZiffer}(l) < Min\{n_{BgZiffer}\} + [Max\{n_{BgZiffer}\} - Min\{n_{BgZiffer}\}] \cdot p_2. \quad (4.2)$$

Schritt 2 Der zweite Schritt wird durch die Funktion *dropEvenlyDistributedLL4* dargestellt. Sie erhält einen Parameter p_3. Auch p_3 ist eine Prozentangabe und aus dem Wertebereich [0,1]. dropEvenlyDistributedLL4 verwirft Features, die in vergleichsweise vielen verschiedenen Ziffern benötigt werden, n_{Ziffer} gibt dabei an, in wieviel verschiedenen Zifferlisten das Feature entdeckt wurde:

$$n_{Ziffer}(l) \geq Max\{n_{Ziffer}\} - p_3 \cdot [Max\{n_{Ziffer}\} - Min\{n_{Ziffer}\}] \qquad (4.3)$$

Bei grösseren Parametern p_1, p_2 und p_3 liefert selectBestFeaturesOfFList weniger Features.

[7] Für spezielle Zwecke entwickelte, nicht standardisierte Verfahren.

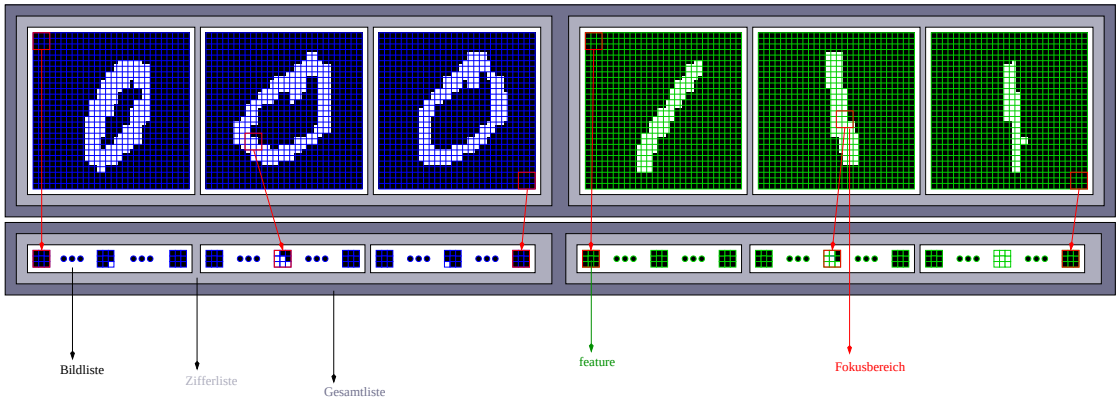

Abbildung 4.3: Der Verteilungsselektor wertet die Vorkommen aller Features statistisch aus. Hierzu werden die Features nach Ziffern geordnet. Die Gesamtliste umfasst alle Listen gleicher Ziffer, die Zifferlisten. Zifferlisten bestehen aus Bildlisten. Deren Elemente sind Features in Form von Vektoren binärer Elemente. In der Abbildung sind Listen durch Rechtecke verschiedener Grautöne markiert. Die Abbildung zeigt die Sortierung am Beispiel einer ersten Neocognitron-Stufe.

4.2.2.2 Abschätzung der Tauglichkeit

Weil die verwendeten Feature-Sätze die Netzstruktur des Neocognitrons bestimmen, ist die Überprüfung des Verteilungsselektors und die Einstellung seiner Parameter letztendlich nur durch den Erfolg der vollständig konstruierten Neocognitrons möglich. Die Leistungsfähigkeit letzterer hängt jedoch auch von anderen Parametern ab, wie beispielsweise den Abfolgen der gewählten Fokusbereichsgrößen oder den vorgegebenen Grenzen für die Zahl der Features in den Stufen. Da die benötigte Zeit zur Konstruktion eines speziellen Neocognitrons im Bereich von Stunden liegt, scheidet dieser indirekte Leistungstest für die begrenzte Zeit dieser Arbeit aus. Zwar ist es schwierig, Aussagen zu treffen, welche Kriterien über die Güte erhaltener Feature-Sätze entscheiden können, jedoch gibt es eine einfache, grobe Möglichkeit zur Abschätzung der Tauglichkeit solcher Selektoren: das menschliche Gehirn. Es ist das derzeit beste neuronale Netz zur Handschrifterkennung. Für die erste Neocognitron-Stufe bilden die Feature-Sätze die Grundbausteine der Ziffern. Das menschliche Hirn sollte dementsprechend hier noch gut in der Lage sein, tendenziell die Frage nach der Güte der Feature-Sätze beantworten zu können. Für höhere Stufen ist dies unmöglich - deren Features sind Hyper-Features, entsprechen somit keinen gewohnten Bildern. Abbildung 4.4 zeigt die vom Verteilungsselektor ausgewählten Features der Stufe Eins bei 30 Trainingsbildern. Für jede Ziffer wurden hierbei die ersten drei Bilder aus der MNIST-Datenbank verwendet. Die Fokusbereichsgröße ist 3x3 Bildpunkte. Die vom Selektor getroffene Wahl scheint plausibel. Es werden alle horizontalen wie auch vertikalen Linien abgedeckt, außerdem die leere und gefüllte Fläche. Die vierzehn Features sind sym-

Abbildung 4.4: Die vom Verteilungsselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgröße ist gewählt als 3x3 Bildpunkte und die Selektorparameter sind: $p_1 = 0,02$; $p_2 = 0,02$; $p_3 = 0$.

metrisch, d.h. zu jedem Feature gibt es ein gespiegeltes Feature. Sechs Features unterscheiden sich allerdings von mindestens einem anderen durch lediglich einen invertierten Bildpunkt. Dieser Feature-Satz lässt sich deshalb nicht vollständig von HAGEN lernen.
Bei der Betrachtung der entsprechenden Feature-Sätze bei Fokusbereichsgrößen von 4x4, 5x5 und 6x6 fällt auf, daß sie überwiegend schwarz umrahmte 3x3 Features beinhalten (Abb. 4.5). Neben der Tatsache, daß offenbar 3x3 Features zur Beschreibung der ersten Stufe ausreichen, unterstützt dies die Annahme, daß der Verteilungsselektor grundsätzlich die an ihn gestellten Erwartungen erfüllt. Andererseits wird auch deutlich, daß zusätzliche Selektionskriterien nötig sind. Die Feature-Sätze umfassen einige Feature-Paare, die sich kaum unterscheiden und HAGEN somit vor Lernprobleme stellen. Der Parameter p_3 scheint wenig Einfluss auf die Art der selektierten Features zu haben, man vergleiche hierzu Abbildung A.2 im Anhang.

4.2.3 Der Differenzselektor

Die Motivation zur Programmierung des Verteilungsselektors war, diejenigen Features auszuwählen, die für den Erkennungsprozeß besonders zuträglich sind. Der Differenzselektor folgt einer anderen Direktive. Er soll Sätze von Features finden, die möglichst gut von HAGEN lernbar sind. Vernachlässigt man die wenigen benutzten Rückkopplungen, so wird HAGEN als einschichtiger Musterassoziator verwendet. Ein einschichtiger Musterassoziator kann befriedigend nur linear unabhängige Muster lernen [Hof93]. Auch unter Berücksichtigung der in dieser Arbeit verwendeten vier Zwischenneurone, gilt dies wahrscheinlich - allerdings bisher nicht beweisbar (S.13). Andererseits sollten trainierte Features möglichst ähnlich zu allen Features sein, deren Klasse sie vertreten - die Wahrscheinlichkeit der richtigen Zuordnung nicht gelernter Features steigt damit. Folgende zwei Kriterien erscheinen sinnvoll:

- Die Features eines Feature-Satzes sollten paarweise möglichst verschieden sein.

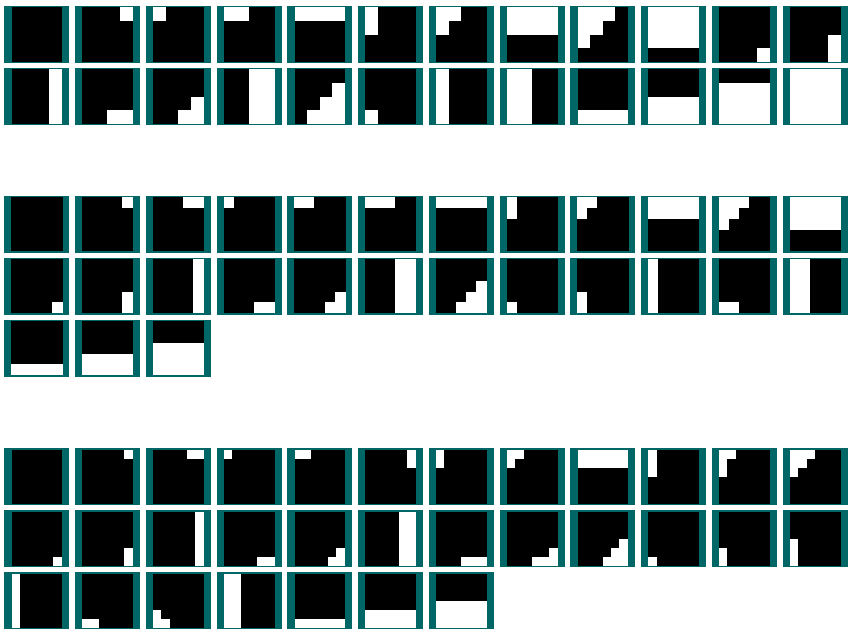

Abbildung 4.5: Die vom Verteilungsselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgrößen sind 4x4, 5x5 bzw. 6x6 Bildpunkte und die Selektorparameter sind: $p_1 = 0,01$; $p_2 = 0,01$; $p_3 = 0$.

- Jedes Feature eines Feature-Satzes sollte möglichst vielen nicht im Satz enthaltenen Features möglichst ähnlich sein.

4.2.3.1 Der Algorithmus

Der Differenzselektor arbeitet ebenfalls in zwei Schritten. Die zugehörigen Funktionen sind *sortFeaturesForHopfieldFitness* und *pileUpSimilarFeatures*.

Schritt 1 Den aus den Trainingsdaten extrahierten Features l wird ein Gütewert $\mathcal{G}(l)$ zugewiesen:

$$\mathcal{G}(l) = \sum_{\delta(k,l) \geq \mathcal{S} \cdot p_4} \delta(k,l) \; + \sum_{\delta(k,l) < \mathcal{S} \cdot p_4} p_5 \cdot (\mathcal{S} \cdot p_4), \tag{4.4}$$

dabei steht $\delta(k,l)$ für die Anzahl der in Feature k gegenüber Feature l invertierten Bildpunkte. $\mathcal{S}$ ist die Gesamtzahl der Bildpunkte eines Features - für einen Fokusbereich der Größe 3x3 bei fünf Features in der Vorgängerstufe also beispielsweise 45. p_4 gibt eine prozentuale Schwelle für die Anzahl verschiedener Bildpunkte bezogen auf die Gesamtpunktzahl an, unterhalb der zwei Features als ähnlich gelten. Die Features werden nun nach ihrer Güte sortiert; die größte Güte führt die Liste an. Der Parameter p_5 bestimmt dabei, welches der beiden obengenannten Kriterien bei der Sortierung überwiegt. Wählt man p_5 negativ, so werden Features selektiert, die möglichst wenigen Features ähneln, sowohl jenen des gewählten Feature-Satzes, als auch den Übrigen.

Schritt 2 Anschließend werden, am Ende der sortierten Liste beginnend, diejenigen Features verworfen, die sich von einem anderen Feature der Liste maximal um $p_6 \cdot \mathcal{S}$ Bildpunkte unterscheiden. Der Parameter p_6 bestimmt somit direkt die Größe des Feature-Satzes. Ist $p_6 = 1$ ist der Feature-Satz leer.

4.2.3.2 Abschätzung der Tauglichkeit

Bei der Abschätzung der Tauglichkeit wird ebenso vorgegangen wie in Kap. 4.2.2.2. Das Augenmerk gilt in diesem Falle allerdings der Lernbarkeit der Feature-Sätze. Vereinfacht sollten die Features eines Satzes paarweise möglichst viele unterschiedliche Bildpunkte aufweisen.
Es wurden wiederum Features aus 30 Eingangsbildern der Ziffern 0 bis 9 extrahiert und sortiert (Abb. 4.6). Jede Ziffer wurde dabei dreifach bedacht. Die Differenzselektorparameter sind: $p_4 = 0,3$; $p_5 = 5$; $p_6 = 0,15$. Etwa die Hälfte aller Features für die Fokusgrößen 3x3 und 4x4 entsprechen denen des Verteilungsselektors. Es muß dabei betont werden, daß die Selektorparameter diesbezüglich nicht kritisch sind - diese Tatsache gilt für einige andere Parametersätze ebenfalls (Abb. A.3). Auffällig ist hingegen, daß sich die Features des Differenzselektors mit zunehmender Fokusbereichsgröße dem menschlichen Auge entfremden - ganz im Gegensatz

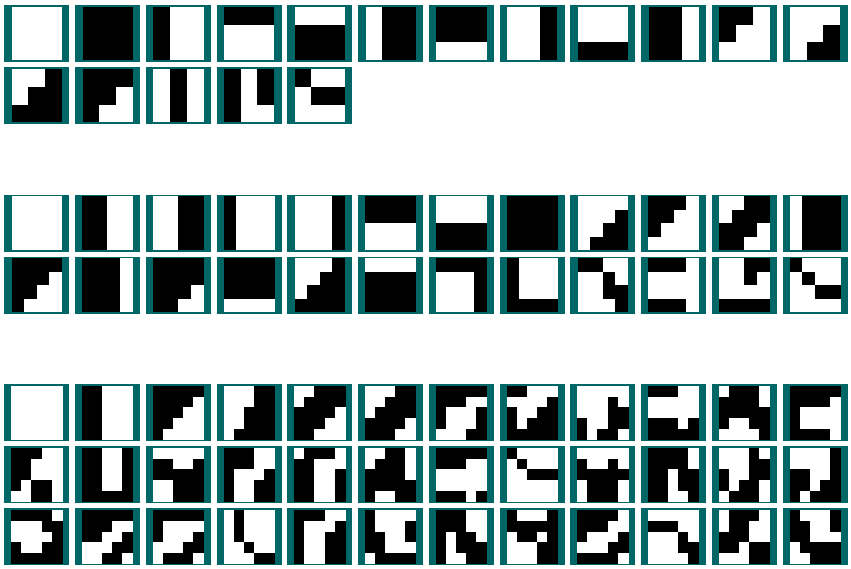

Abbildung 4.6: Die vom Differenzselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgrößen sind 3x3, 4x4 bzw. 5x5 Bildpunkte und die Selektorparameter sind: $p_4 = 0,3$; $p_5 = 5$; $p_6 = 0,15$.

zu jenen des Verteilungsselektors (s.h. auch Abb. A.4). Seine Aufgabe erfüllt der Differenzselektor: Es existiert in dem Beispiel nicht ein einziges Feature Paar, das sich nur in einem Bildpunkt unterscheidet. Die Sätze des Differenzselektors sind somit besser von HAGEN lernbar als die des Verteilungsselektors.

4.2.4 Der kombinierte Selektor

Die in den vorangegangenen Kapiteln beschriebenen Selektoren ergänzen sich in ihrer Funktionsweise. Der Verteilungsselektor wählt die zum Erkennungsprozeß günstigsten Features aus. Der Differenzselektor selektiert Feature-Sätze, die bestmöglich von einem neuronalen Netz wie HAGEN lernbar sind. Beide Eigenschaften werden vom binären Neocognitron benötigt. Es liegt deshalb nahe, beide Selektoren zu verbinden. Der kombinierte Selektor leistet dies. Die Funktion *combinedSelector* in der Bibliothek *FeatureExtraction-lib-V2.9.nb* führt zuerst den Verteilungsselektor und anschließend den Differenzselektor aus. Sie erhält dementsprechend die Parameter p_1 bis p_6 wie sie in den vorangegangenen Kapiteln beschrieben wurden.

4.2.4.1 Abschätzung der Tauglichkeit

In Abbildung 4.7 sind die Feature-Sätze für Fokusbereichsgrößen von 3x3 bis 6x6 Bildpunkten bei gleichen Selektorparametern dargestellt. Es fällt auf, daß die gewählten Features im Gegensatz zu den beiden einzelnen Selektoren für alle Fokusbereichsgrößen im wesentlichen skaliert zu sein scheinen. Die Feature-Sätze unterscheiden sich im wesentlichen in der Anzahl der Features je Satz - eine gute Voraussetzung für die spätere automatische Justage der Selektorparameter über die Anzahl gewünschter Features (Kap. 4.3.1).

4.2.5 Funktionstests mit vereinfachten S-Ebenen

4.2.5.1 Hopfieldebenen

Hopfieldnetze sind spezielle Autoassoziatoren (s.h. auch S.14) [Hof93]. Sie sind einschichtig und rückgekoppelt. Die Netzwerkzyklen eines zeitdiskreten Hopfieldnetzes lassen sich durch eine Matrixmultiplikation mit den Eingangsvektoren darstellen. Das Training des Hopfieldnetzes reduziert sich auf die Bestimmung der Matrixelemente. Hierfür existiert eine geschlossene Lösung:

$$\omega_{kk} = 0 \tag{4.5}$$

$$\omega_{kl} = \sum_{\mu=1}^{p} \vec{E}_k^{\mu} \cdot \vec{E}_l^{\mu} \qquad \forall\ k \neq l \tag{4.6}$$

$$\theta_k = 0. \tag{4.7}$$

Abbildung 4.7: Die vom kombinierten Selektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgrößen sind 3x3, 4x4, 5x5 und 6x6 Bildpunkte und die Selektorparameter sind: $p_1 = 0,006$; $p_2 = 0,006$; $p_3 = 0,00006$; $p_4 = 0,3$; $p_5 = 2,5$; $p_6 = 0,16$.

Hierbei bezeichnen die ω_{kl} die Matrixelemente und $\vec{E}^{\mu}$ das μ-te Feature aus dem Trainingssatz. Die binären Eingangsdaten beziehen dabei ihre Werte aus der Menge {-1,1}. Die Neuronenschwellen θ sind Null; das ist nur dann legitim, wenn jedes Eingangsmuster etwa diesselbe Anzahl negativer wie positiver Bildpunkte aufweist. Im nullten Netzwerkzyklus werden die Eingangsdaten direkt auf die Ausgänge gelegt - diese sind rückgekoppelt und fliessen somit im nächsten Zyklus in die Netzberechnung ein. Damit sind alle Netzzyklen ansonsten gleich zu behandeln:

$$\epsilon_k = \sum_{l=1}^{N} \omega_{kl} \cdot A_l \tag{4.8}$$

$$A_k(t+1) = \begin{cases} -1 & \text{für } \epsilon_k < \theta_k \\ A_k(t) & \text{für } \epsilon_k = \theta_k \\ +1 & \text{für } \epsilon_k > \theta_k. \end{cases} \tag{4.9}$$

Die ϵ bezeichnen dabei die effektiven Neuroneneingänge und A_k ist die Aktivität am k-ten Ausgang. Da die Neuronenschwellen nur dann nicht mitgelernt werden müssen, wenn alle Eingangsmuster etwa diesselbe Zahl negativer wie positiver Bildpunkte aufweisen und außerdem die Anzahl zerstörungsfrei lernbarer Muster durch die Neuronenanzahl (sprich die Größe der Eingangsmuster) beschränkt ist, wurde zu folgendem Trick gegriffen: Die Eingangsmuster werden durch die Funktion *blowUp* in *NDH-ABC-V7.0* auf die vierfache Größe gebracht. Dem Muster wird sein Inverses angehängt, außerdem zwei teilinvertierte (und dabei auch gegeneinander vollständig inverse) Varianten, so daß nicht nur die Anzahl belegter Eingangsneuronen vervierfacht wird, sondern auch die Anzahl der positiven Eingänge immer exakt denen der negativen Eingänge entspricht.
Das Hopfieldnetz wurde in Vorversuchen vor allem zur Kontrolle der Lernbarkeit bestimmter Feature-Sätze verwendet und um einen ersten Eindruck über die Folgen der Generalisierungseigenschaft eines neuronalen Netzes zu erhalten. In Verbindung mit Vergleichsebenen können sie aber auch effizient als Klassifizierungsschicht eingesetzt werden. Ihre Größe wächst bei dem hier verwendeten Verfahren allerdings recht schnell über die von Mathematica in angemessener Zeit berechenbare Grenze hinaus. Abbildung 4.8 zeigt ein Beispiel eines Feature-Satzes, für den die Reaktion des zugehörigen Hopfieldnetzes getestet wurde. Hopfieldnetze sind rückgekoppelt und benötigen deshalb mehrere Zyklen, um ihren Endzustand zu erreichen. Diese Zyklen sind in Abbildung 4.9 für leicht veränderte Features aus dem Trainingssatz aufgezeichnet. Man kann sozusagen der Generalisierung bei der Arbeit zuschauen.

4.2.5.2 Vergleichsebenen

Eine Vergleichsebene wird durch einen Algorithmus dargestellt, der den Inhalt eines Fokusbereichs mit allen, jeweils eine Klasse vertretenden, Features vergleicht und ihn den Klassen zuordnet, deren Referenz-Feature von ihm um nicht mehr als

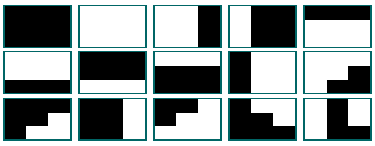

Abbildung 4.8: Der zu Abbildung 4.9 gehörende Trainingssatz von 3x3 Features des verwendeten Hopfieldnetzes.

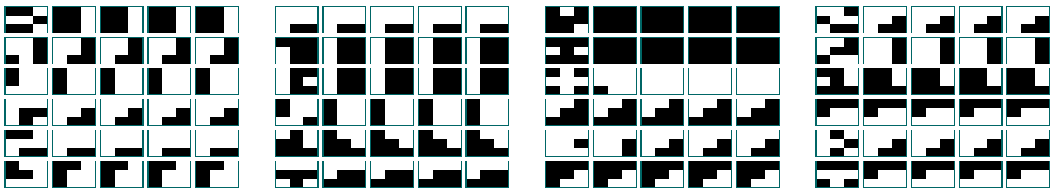

Abbildung 4.9: Um die Generalisierungseigenschaften eines Hopfieldnetzes sichtbar zu machen, wurden zufällig veränderte Muster aus dem Trainingssatz (Abb. 4.8) dem Netz vorgelegt und die Ausgaben der Netzwerkschritte ausgegeben. Das erste Bild einer Reihe ist das Eingangsbild und Bild fünf die Netzwerkausgabe.

eine bestimmte Anzahl von Bildpunkten abweicht. Vergleichsebenen werden aktiviert durch Übergabe der Zeichenkette "deviation" als Parameter an die Funktion *classifyByFeatureList* im BONNEE-Programm. Da sie einfach implementiert, leicht zu kontrollieren und schnell sind, bilden sie ein gutes Werkzeug zur Überprüfung der Neocognitron-Algorithmen und zum Vergleich mit den HAGEN-F/S-Ebenen. Überdies sind sie, einer Hopfieldebene nachgeschaltet, ein gutes Mittel zur Simulation einer generalisierenden Klassifizierungsschicht.

4.3 Die automatische Netzwerkstrukturoptimierung BONNEE

4.3.1 Die Parameterschleife

Das Neocognitron ist ein mehrstufiger Feature-Extraktor. Jede seiner Stufen ist in ihrer Funktionsweise bestimmt durch die gewählten C-,S-Fokusbereichsabmessungen und den ihr zugeordneten Satz von Features. Die korrekte Klassifikation der Features dieses Feature-Satzes wird durch den entsprechend trainierten HAGEN-Block bewerkstelligt. Die Güte der von ihm bereitgestellten Generalisierung wird neben der Wahl des Feature-Satzes von HAGEN spezifischen Parametern wie Neuronenschwelle oder dem Trainingsalgorithmus mitbestimmt. Da die Dimensionierung einer Stufe nicht unabhängig von der vorhergehenden und nachfolgenden

	p_1	p_2	p_3	p_4	p_5	p_6
Stufe 1	0,005	0,005	0,0	0,3	1,0	0,15
Stufe 2	0,00743	0,00547	0,00008	0,29123	2,52	0,15683
Stufe 3	0,005504	0,00609	0,00004	0,278847	2,58	0,19086
Stufe 4	0,00589	0,00586	0,000059	0,324404	2,52	0,15456
Stufe 5	0,006725	0,00628	0,000069	0,306902	2,4	0,15597
Stufe 6	0,0068	0,007525	0,00007	0,275844	2,76	0,12887
Stufe 7	0,005	0,005	0,0	0,3	2,0	0,2
Stufe 8	0,005	0,005	0,0	0,3	2,0	0,2

Tabelle 4.1: Die gemittelten Parameter des kombinierten Selektors der Neocognitron-Stufen Eins bis Acht. Gemittelt wurde über 50 Neocognitron-Topologien zur Erkennung der Ziffern 0 und 1 bei gleichen Fokusdimensionierungen. Die Tabelle zeigt, daß die von der Parameterschleife gefundenen Optima der Parametersätze aller Stufen in unmittelbarer Umgebung des Start-Parametersatzes (Stufe 7,8 zeigen diesen unverändert) liegen.

Stufe ist, erfolgt die Parameterwahl während der Aufbauphase sequentiell. Beginnend mit der ersten Stufe werden Abmessungen und Feature-Sätze Stufe für Stufe festgelegt. Die angestrebte Datentypunabhängigkeit des Aufbaus verlangt die Entwicklung eines automatischen Verfahrens zur Suche und Optimierung der Netzwerkparameter und somit der Neocognitron-Topologien für verschiedene Eingangsdaten.
Die HAGEN bzw. SHAGEN spezifischen Parameter: Anzahl Rückkopplungen (4), Neuronenschwelle (SHAGEN: -1) und die Trainingsparameter: Populationsgröße (30 Individuen), obere Grenze für die Anzahl der Ersetzungen schlechter Individuen durch das aktuell beste (25), Mutationsrate (7,8%), maximale Anzahl Generationen (200.000) sowie die Gewichtung fehlender Aktivität gegenüber überflüssiger Aktivität bei der Bewertung der Gewichtskonstellationstauglichkeit ($1 \leftrightarrow 1$ + **1**=doppelte Gewichtung), sind im Rahmen dieser Arbeit für alle Stufen gleich. Die für die Optimierung dieser Parameter zusätzlich anfallende Konstruktionszeit lässt sich durch die geringen Unterschiede ihrer Optima für verschiedene Eingangsmustertypen nicht rechtfertigen. Alle Vorversuche haben dies bestätigt - es muß allerdings erwähnt werden, daß deren Statistik nicht ausreichend ist, um dies allgemein zu belegen. Der in diesem Kontext interessanteste Parameter dürfte dabei die Anzahl der Rückkopplungen sein. Die Anzahl verwendeter Rückkopplungen hat direkt Einfluß auf den Lernerfolg und die Güte der Generalisierung. Die Wahl der hier angegebenen Parameter wird in Kapitel 4.3.2 ausführlich diskutiert.
Trotz der fixierten spezifischen HAGEN/SHAGEN-Parameter muß das Neocognitron-Training noch immer acht Parameter je Stufe optimieren. Es sind neben den Fokusdimensionen (und damit der vorgegebenen maximalen Anzahl verwend-

barer Features in den Stufen) die sechs Parameter des kombinierten Selektors (Kap. 4.2.4). Der Selektor sorgt dann in Verbindung mit den gewählten Trainingsdaten für die Extraktion und Selektion der Features der Feature-Sätze jeder Stufe. Weil die Selektorparameter Relativgrößen sind, hat es sich bewährt, in allen Stufen (ausgenommen der ersten) mit dem gleichen Parametersatz zu starten. Geringe Variationen zeigen große Effekte. Es bestätigt sich aber, daß sich zumeist in unmittelbarer Nähe der Startparameter ein Parametersatz finden lässt, der die gewünschte Anzahl Features selektiert. Ob der in dieser Arbeit experimentell ermittelte Startparametersatz der günstigste ist, bleibt ungeklärt. Die erschöpfende Suche war im zeitlichen Rahmen nicht möglich. Tabelle 4.1 zeigt die über 50 Neocognitron-Topologien gemittelten Parametersätze. Die Parametersätze der Stufen sieben und acht wurden durch den Suchalgorithmus unverändert übernommen - dies sind die Startparameter.
Innerhalb des Mathematica-Programms *bonneeV2.6.nb* wird die Konstruktion einer bestimmten Neocognitron-Topologie von der Funktion *constructNet* übernommen. Diese beinhaltet die sogenannte Parameterschleife zur automatischen Suche geeigneter Selektorparameter. Die Variation der Parameter wird dabei innerhalb festgelegter Grenzen zufällig vorgenommen. Die erfolgreichste Parameterkombination wird ständig zwischengespeichert und alle zehn Schleifendurchgänge zur Ausgangskombination für die folgenden Variationen erhoben. Die Parameter werden nicht einzeln variiert, sondern in folgenden Verbunden: $\{p_1, p_2, p_3\}$ und $\{p_4, p_5\}$ und $\{p_6\}$. Diese Gruppierungen entsprechen den logisch zusammengehörenden Schritten innerhalb des kombinierten Selektors (Kap. 4.2.2,4.2.3). Die Abbruchsbedingung für die Suche eines Parametersatzes wird durch den gewünschten Bereich der Anzahl Features im Feature-Satz vorgegeben. Im Mittel sind die benötigten Variationsschritte derzeit in der Größenordnung einer Dekade - es ist dementsprechend nicht zu erwarten, daß kompliziertere Algorithmen hier wesentlich schneller konvergieren. Ein systematischer Fehler ist allerdings durch die Wahl der Parametergrenzen möglich. Diese sind wie folgt gewählt: $p_1, p_2 \in [0; 0,015]$, $p_3 \in [0; 0,0002]$, $p_4 \in [0; 0,7]$, $p_5 \in [1; 5]$ und $p_6 \in [0; 0,35]$. Die Fokusdimensionierung muß vorab manuell festgelegt werden und wird von der Parameterschleife nicht verändert. Alle Netzwerkparameter eines gefundenen Neocognitron-Individuums, einschliesslich der HAGEN/SHAGEN-Netzgewichte, werden in automatisch generierten Dateien mit der Endung ".syms" gespeichert. Sie enthalten die Definitionen in direkt von Mathematica lesbarer Form. Die Auswahl, welche Parameter aus einer solchen Datei gelesen werden sollen, trifft die globale Liste *loadSymsL*. Sie wird vor einem entsprechenden Get[...]-Aufruf mit den Namen aller gewünschten Parameter gefüllt. Eine auf diese Weise eingelesene Neocognitron-Topologie kann durch die Funktion *processImageUpToStage* für beliebige Eingangsdaten bis zu einer beliebigen Stufe abgearbeitet werden. Details sind den Programmen selbst zu entnehmen - sie sind sehr ausführlich dokumentiert. Die Benennung der Funktionen und Variablen ist außerdem nahezu in Klartext gehalten (s.h auch S. XVIII). Das ebenfalls

innerhalb *constructNet* initiierte Training der HAGEN-Blöcke auf die Klassifikation der Features gefundener Feature-Sätze wird im folgenden Kapitel detailliert beschrieben.

4.3.2 Das Softwarepaket SHAGEN

Das sich in der Entwicklung befindliche Programm HANNEE bietet eine Graphische Benutzeroberfläche zum Training eines HAGEN-Blockes (Kap. 2.2.4). Erste einfache Trainingsalgorithmen sind implementiert. Das Training von fünfzehn 64-Bit-Eingangsmuster/15-Bit-Ausgangsmusterkombinationen bei praxistauglichen Parametern (sie entsprechen in etwa den in einigen Zeilen ausführlich genannten Parametern für das SHAGEN-Training) benötigt für 10.000 Generationen etwa 30 Minuten ($\star$). Es zeigt sich jedoch, daß 10.000 Generationen nicht genügen, um diese Kombinationen ausreichend fehlerfrei zu lernen. Die daraus resultierenden Trainingszeiten sind für die Parametersuche nicht akzeptabel, schliesslich müssten sie für jede Stufe aufgebracht werden. Einfache Veränderungen in den vorhandenen Lernalgorithmen beschleunigen zwar die Konvergenz des Trainings gegen eine gute Gewichtskonstellation ein wenig - für die Lösung zuvorgenannter Aufgabe genügen sie jedoch nicht (jedenfalls nicht den im Rahmen dieser Arbeit gemachten Vorversuchen zu Folge).
Es ist möglich, daß evolutionäre Algorithmen existieren, die für die Lösung der Lernaufgaben jeder Neocognitron-Stufe gleichermaßen ausreichend schnell gegen eine gute Lösung konvergieren. Es ist jedoch fraglich, ob in der kurzen Zeit, die für die Bewältigung dieser Aufgabe bereitstand, auf diese Art und Weise ein Faktor in der Größenordnung von 40 gegenüber dem bestehenden System erreicht hätte werden können (dieser wurde mit der im Folgenden beschriebenen Methode innerhalb kürzester Entwicklungszeit erzielt), ohne die Datentypunabhängigkeit des Neocognitrons zu gefährden.
Es liegt nahe, die Komponenten des HANNEE-Trainings auf Geschwindigkeit zu optimieren. Dazu muß zuerst festgestellt werden, in welchen Trainingsschritten die Engpässe zu finden sind und inwieweit eine Beschleunigung durch die Hardware begrenzt ist. Die grundsätzlichen Geschwindigkeitsgrenzen für das Softwaretraining möchte ich anhand des im Rahmen der Arbeit entwickelten geschwindigkeitsoptimierten Softwarepakets SHAGEN erläutern. Dessen Trainingsroutinen wurden in Assembler implementiert und bieten deswegen einen Einblick in die durch den Algorithmus und den Computer selbst gesteckten Obergrenzen. Der Vergleichbarkeit wegen werden im Folgenden die Größen Individuenfrequenz und Generationenfrequenz gebraucht. Eine Individuenfrequenz von 1Hz bedeutet hierbei, daß genau eine HAGEN- bzw. SHAGEN-Gewichtskonstellation in der Sekunde bearbeitet wird. Können hingegen alle Individuen einer Generation (s.h. Populationsgröße) einmal in der Sekunde entsprechend bearbeitet werden, leistet das System 1Hz Generationenfrequenz.
Das Softwarepaket SHAGEN (**S**oft **HAGEN**) besteht aus:

- Einem i486-Assembler kodiertem Trainingsalgorithmus.
- Einer C-basierter HAGEN-Chip Simulation.
- Einer Schnittstelle zur Ansteuerung via Mathematica (TCP/IP-fähig).

Die Motivationen zur Entwicklung dieses Pakets waren:

- Durch den Einsatz optimierter Assemblerroutinen ist SHAGEN für gängige Trainingsaufgaben etwa um den Faktor 40 schneller als die in der Entwicklung befindlichen HANNEE-Routinen. Die HANNEE-Routinen sind primär um Flexibilität bemüht.
- SHAGEN kann auf mehreren Computern gleichzeitig arbeiten - im Rahmen dieser Arbeit stand mir hingegen nur ein HAGEN-Chip zur Verfügung.
- Die Ansteuerung von SHAGEN ist einfacher, beispielsweise die Programmierung der automatischen Wahl von Rückkopplungen für verschiedene Lernaufgaben ist bei Verwendung HAGENs aufwendig.
- Die Übertragung von SHAGEN-Gewichten auf den HAGEN-Chip ermöglichen einen direkten Test HAGENs auf Übereinstimmung mit der Theorie.

Der Engpass im Softwaretraining Der von mir entwickelte genetische Algorithmus ist sehr einfach. Eine Population besteht hierbei aus $\mathcal{N}$ Individuen - das sind Gewichtskonstellationen für HAGEN (bzw. SHAGEN). Alle diese Individuen werden innerhalb einer Generation mutiert, ausgenommen das Individuum, welches die Erwartungen in Form der Eingabe-Ausgabe-Kombinationen bisher am besten erfüllte. Die Mutation eines Individuums ersetzt dabei einen prozentualen Anteil $\mathcal{M}$ (Mutationsrate) der Gewichte einer Konstellation durch zufällig gewählte neue Gewichtswerte. Alle Individuen einer Population sind in einer ungeordneten Liste gespeichert. Ihre Leistungen werden nun erneut bewertet. Beginnend bei der schlechtesten Gewichtskonstellation werden rückwärts in dieser Liste maximal $\mathcal{R}$ Individuen durch das derzeit beste Individuum ersetzt und zwar höchstens solange, bis der Beginn der Liste erreicht wurde. Diese Ersetzungsstrategie hat einige Vorteile:

- Sie ist leicht zu implementieren - wichtig für eine Assemblerimplementation bei knapper Entwicklungszeit.
- Durch die von Generation zu Generation variierende Anzahl von Ersetzungen wird die Gratwanderung zwischen Konvergenz der Population gegen das derzeit erfolgreichste Individuum und dem zu geringem Einfluß dieses Individuums automatisiert - beide Effekte wechseln sich ab.

- Gegenüber einigen komplizierteren Algorithmen verwendet dieser Algorithmus keine Kenntnis über das spezifische Problem und entspricht somit der geforderten Datentypunabhängigkeit des Neocognitrons.
- Die gegenüber komplizierten Algorithmen langsamere Konvergenz gegen ein optimales Individuum wird ausgeglichen durch die massive Geschwindigkeit der Implementation.

Das Training lässt sich in drei Phasen aufspalten:

- Die Mutationsphase; sie wird zeitlich dominiert durch die Erzeugung der Zufallszahlen und ist somit abhängig von den Parametern $\mathcal{M}, \mathcal{N}$.
- Die Bewertungsphase; sie wird zeitlich dominiert durch die Dauer eines Netzzyklus - für SHAGEN durch die Größe der Eingabe/Ausgabevektoren und die Anzahl $\mathcal{F}$ der benötigten Rückkopplungen.
- Die Ersetzungsphase; ihre Dauer bestimmt sich aus dem Zeitaufwand für die Kopie aller benötigten Gewichte einer Konstellation - also durch die Dimensionen der Eingabe/Ausgabevektoren und durch $\mathcal{R}, \mathcal{N}$.

Um dem Leser einen Eindruck zu vermitteln, um welche zeitlichen Größenordnungen es sich handelt, werden die benötigten Zeiten für die Phasen im Folgenden überschlagen. Es handelt sich dabei um eine grobe Rechnung, die Realität ist bedeutend komplizierter. Es gilt allerdings die Faustregel, daß die errechneten PentiumPro-Taktzyklen eine Obergrenze für die in Assembler implementierten Phasen darstellen - auf die Einbeziehung komplizierter Optimierungsmethoden ist im Folgenden verzichtet worden:

Die Mutationsphase Es existieren mindestens zwei grundlegend verschiedene Möglichkeiten, die zu mutierenden Gewichte einer Netzkonstellation auszuwählen:

1. Die Netzgewichte können in einem Vektor angeordnet werden; eine Zufallszahl entscheidet nun über den Index des zu verändernden Eintrages.
2. Jedem Gewicht wird eine Zufallszahl zugeordnet. Überschreitet diese eine Schwelle, wird das zugehörige Gewicht mutiert.

Um die Vergleichbarkeit mit den bisherigen Algorithmen zu gewährleisten und da bei variierenden Eingangs/Ausgangsmustergrößen die Optimierung der ersten Methode überaus komplex ist, habe ich Methode Nummer zwei gewählt. Die Mutationsentscheidungen werden dabei durch 8-Bit Zufallszahlen getroffen.
Für alle nun folgenden Überschlagsrechnungen wird eine Mutationsrate $\mathcal{M}$ von 10%, eine HAGEN-Eingangsmustergröße von 64-Bit, ein 10-Bit Ausgang (entspricht zehn Neocognitron-Features), eine Populationsgröße $\mathcal{N}$ von 30 Individuen und eine maximale Zahl von Ersetzungen pro Generation $\mathcal{R}$ von 24 Individuen

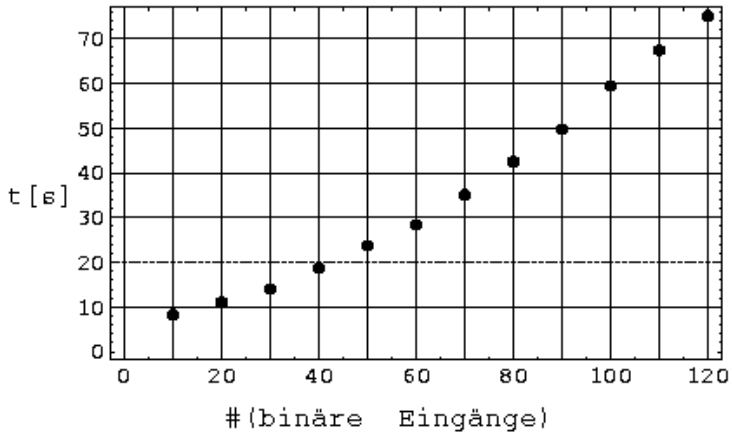

Abbildung 4.10: Aufgetragen sind die für das gesamte Training benötigten Zeiten in Abhängigkeit der Anzahl verwendeter Eingänge. Die Ausführungszeiten der Mutations-, Bewertungs- und Ersetzungsphasen sind ungefähr proportional zur Anzahl verwendeter Netzgewichte (die eine Folge benötigter Ein- und Ausgänge, sowie Rückkopplungen sind). In die Bewertungsphasenbegrenzung fliesst außerdem die Anzahl aktiver Bildpunkte im Eingabemuster ein. Die übrigen Parameter sind: 4 Rückkopplungen, 10 Ausgangsbits, 30 Individuen in der Population, 7,8% Mutationsrate, maximal 25 Kopien des besten Individuums in jeder Generation, 10000 Generationen, 2,4GHz PentiumIV. Außerdem ist jeder Messwert über 10 zufällige Eingangsmuster gemittelt (Einzelmessungen wurden nicht gemacht).

vorausgesetzt. Weiterhin gehe ich davon aus, daß die Eingangsmuster etwa genausoviele aktive wie nicht aktive Punkte aufweisen. Diese Größen haben sich in der Praxis bestätigt.
Die Anzahl benötigter Zufallszahlenbits pro Generation ist somit

$$(\mathcal{N}-1)(64\cdot 10)\cdot\{8\text{Bit}+\mathcal{M}\cdot 16\text{Bit}\}\approx 180.000\text{Bit}, \tag{4.10}$$

etwa 11300 Integer-Worte[8]. Der Standard-C-Zufallszahlengenerator benötigt für die Erzeugung einer Integerzufallszahl ca. 120 PentiumPro-Klasse Taktzyklen. Die Mutationsphase einer Generation würde somit etwa 1.360.000 Taktzyklen benötigen. Bei Einsatz eines 2GHz PentiumIV-Prozessors entspräche dies etwa einer durch die Mutationsphase auf 1 kHz begrenzten Generationenfrequenz.

Meine Assemblerimplementation eines Pseudo-DES[9] [PTVF99] Algorithmus benötigt (ohne die Zufälligkeit verschlechternde Tricks) etwa 20 Taktzyklen[10] zur

[8] Ein Integer-Wort besteht aus 16-Bit.
[9] **D**ata **E**ncryption **S**tandard
[10] Für PentiumPro-Klasse Prozessoren.

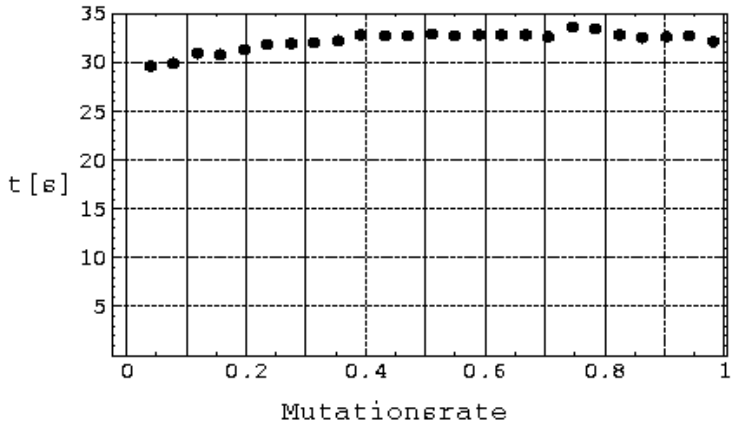

Abbildung 4.11: Der entwickelte Zufallszahlenalgorithmus ist relativ stabil gegenüber Veränderungen der Mutationsrate. Die Parameter sind: 64 binäre Eingänge, 4 Rückkopplungen, 10 Ausgangsbits, 30 Individuen in der Population, maximal 25 Kopien des besten Individuums in jeder Generation, 10000 Generationen, 2,4GHz PentiumIV. Außerdem ist jeder Messwert über 10 zufällige Eingangsmuster gemittelt.

Erzeugung einer 16-Bit Zahl. Außerdem werden durch Kreuzung aus diesen innerhalb weiterer 15 Taktzyklen zusätzlich acht 8-Bit Zahlen erzeugt. Da der Algorithmus bei geringer Mutationsrate nicht alle erzeugten Zufallszahlen auch verwertet[11], ist deren Zufälligkeit dadurch nicht gefährdet. Bei hoher Mutationsrate muß die Gewährleistung ausreichender Zufälligkeit geprüft werden. Ein weiterer Vorteil des Algorithmus ist hingegen, daß die Begrenzung der Mutationsphasenfrequenz kaum von der Mutationsrate abhängt (Abb. 4.11). Im Schnitt lässt sich hiermit die Begrenzung der Generationenfrequenz durch die Mutationsphase auf ca. 6kHz anheben.

Die Bewertungsphase Ist in den Eingangsmustern etwa die Hälfte der Eingänge aktiv, so reduziert sich - bei Vernachlässigung möglicher Rückkopplungen (in anbetracht der in der Arbeit verwendeten vier Rückkopplungen ist dies legitim) - die Anzahl der zur HAGEN-Simulation benötigten Operationen im wesentlichen auf $32 \cdot 10$ Gewichtsadditionen. In einer Schleife benötigen diese ca. $32 \cdot 10 \cdot 3 = 960$ Taktzyklen pro Zyklus, also 9600 Taktzyklen pro Individuum (bei zehn Klassen und entsprechend zehn Trainingsmusterkombinationen). Bei 29 zu bewertenden Individuen benötigt eine Bewertungsphase etwa 280.000 Taktzyklen, beschränkt die Generationenfrequenz demnach auf ungefähr 7kHz. Die Simulation

[11] Der interne Zufallszahlenpuffer wird nach spätestens acht Mutationsversuchen verworfen.

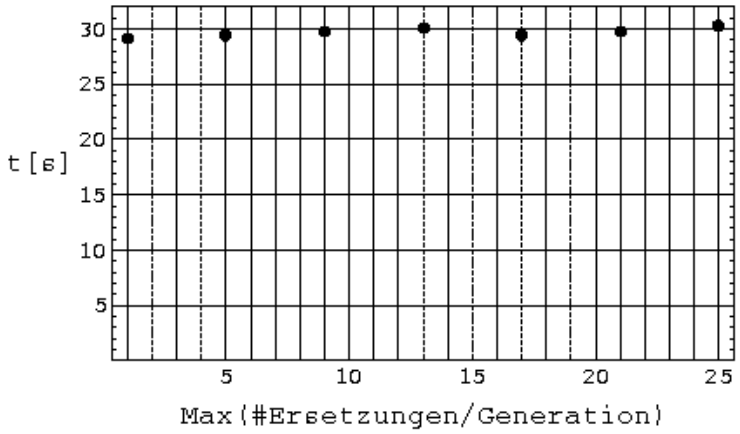

Abbildung 4.12: Die Anzahl der Ersetzungen schlechter Individuen durch das aktuell beste Individuum hat kaum Einfluß auf den Zeitbedarf einer Generation. Die verwendeten Parameter sind: 64 Eingangsbits, 4 Rückkopplungen, 10 Ausgangsbits, 30 Individuen in der Population, 7,8% Mutationsrate, 10000 Generationen, 2,4GHz PentiumIV, außerdem ist jeder Messwert über 10 zufällige Eingangsmuster gemittelt.

des HAGEN-Chips ist aus Entwicklungszeitgründen in C geschrieben, dies dürfte erfahrungsgemäss die Begrenzung der Generationenfrequenz ein wenig unter die der Mutationsphase herabsetzen und lässt weiteren Optimierspielraum offen. Es gibt zwar schon einen äußerst schnellen Assemblerprototyp, dieser kommt wegen spezieller Optimierungen bisweilen jedoch mit dem Prozessorcache[12] in Konflikt. Er müsste in den Linux-Kernel[13] kompiliert werden, um den Cache sauber steuern zu können. DerTrainingsalgorithmus stellte sich übrigens bei Tests auf die Cache-Fluktuationen so präzise ein, daß das Cache-Problem nur durch direkten Gewichtsvergleich mit der C-Simulation auffiel! Da die Assemblerimplementation an dieser Stelle nicht notwendig war, wurde auf die C-Simulation zurückgegriffen.

Die Ersetzungsphase Die Geschwindigkeit dieser Phase lässt sich genauer angeben als die der beiden anderen. Es existiert hier außer den üblichen Tricks (aligning, cache-günstige Befehlsgruppierung, etc. **??**) kaum Optimierspielraum (außer der Verwendung von MMX[14]-Befehlen, diese sind jedoch nicht Teil des Standard-i486-Befehlssatz). Die Kopie eines 32-Bit Wortes benötigt innerhalb einer Schleife etwa 3 Taktzyklen. Weil pro Individuum $64 \cdot 10$ Netzgewichte mit jeweils 16-Bit kopiert werden müssen, benötigt die Ersetzungsphase etwa 12.000

[12] Zwischenpuffer aus gegenüber dem Hauptspeicher schnellen RAM-Bausteinen.
[13] Kern des Betriebssystems Linux.
[14] **M**ulti **M**edia **Ex**tension

Taktzyklen, wobei davon ausgegangen wird, daß im Schnitt ungefähr die Hälfte der maximalen Ersetzungen $\mathcal{R}$ tatsächlich ausgeführt werden. Die resultierende Frequenzbegrenzung von ca. 160kHz stellt kein Problem dar (Abb. 4.12). An ihr lässt sich aber erahnen, welche Generationenfrequenzen mit genetischen Algorithmen überhaupt möglich sind. Frequenzen im Megahertzbereich sind wohl kaum erreichbar.

Es sei nochmals ausdrücklich erwähnt, daß die Beispielrechnungen der Vorführung von Rechenzeittendenzen dienen. Sie erheben keinen Anspruch auf Exaktheit. Reale Rechenzeiten eines PentiumIV-Prozessors lassen sich nicht direkt aus der PentiumPro-Taktzyklentabelle übertragen. Der Geschwindigkeitszuwachs einer neuen Prozessorgeneration ist nicht direkt durch die Erhöhung der Taktfrequenz gegeben, außerdem sind Caching und Pipelining aktueller Prozessoren äußerst komplex [AMD02]!

Der Engpass in der Kommunikation mit der Hardware Für das Training eines neuronalen Netzes durch genetische Algorithmen ist der geschwindigkeitsbegrenzende Faktor die Erzeugung der Zufallszahlen zwecks Mutation. Bei optimaler Hardwareimplementation des HAGEN-Trainings via FPGA[15] der Darkwing-Platine (Kap. 2.2.2) ist die obere Grenze für die Mutation der Netzgewichte die Transfergeschwindigkeit des LVDS[16]-Busses zum HAGEN-Chip. Dieser bewältigt 11.4 Gbit, unter Vernachlässigung des Transferprotokolloverheads also etwa 10^9 Gewichte in der Sekunde. Allerdings können die Gewichte zum HAGEN-Chip nur blockweise übertragen werden, d.h. der Bus kann gleichermaßen ein Gewicht wie auch alle Gewichte eines Blockes mit maximal etwa 122kHz übertragen. Diese Frequenz wird durch die auf HAGEN befindlichen DACs[17] auf 12,2kHz begrenzt (Kap. 2.2.2). Die Verbindung von Darkwing zur Computerwelt wird mittels PCI[18]-Bus bewerkstelligt. Dieser setzt die Individuenfrequenz, also die Frequenz, mit der Gewichtskonstellationen zum HAGEN-Chip übertragen werden können (ebenso wie die Übertragungsfrequenz einzelner Gewichte), auf etwa 2kHz herab (Kap. 2.5). Diese PCI-Begrenzung fällt für das zukünftige Training mittels des NATHAN-Systems, entwickelt von A.Grübl, weg [Gru03]. NATHAN ermöglicht die Ansteuerung des HAGEN-Chips auf voller LVDS-Bandbreite und bietet die Umgebung für ein eingebettetes Linux System auf FPGA-Basis. Dieses wird zukünftig das Training vollständig übernehmen können. Unter Vernachlässigung der übrigen Phasen eines Trainings und aller Protokolloverheads bietet der HAGEN-Chip bei einer Populationsgröße von 30 Individuen (von denen 29 mutiert werden) derzeit eine Obergrenze für die Generationenfrequenz von

[15] **F**ield **P**rogrammable **G**ate **Array**.
[16] **L**ow **V**oltage **D**ifferential **S**ignal.
[17] **D**igital to **A**nalog **C**onverter.
[18] **P**eriphal **C**omponent **I**nterconnect.

$\frac{2}{29}kHz \approx 70Hz$. Damit wären 10.000 Generationen in etwa 150 Sekunden möglich (vorausgesetzt Training und Übertragung sind vollständig parallelisierbar). Da dieses System während dieser Arbeit nicht zur Verfügung stand, wurde SHAGEN entwickelt.

4.3.2.1 Optimierung der SHAGEN-Trainingsparameter

Der Zeitbedarf des SHAGEN-Trainingsalgorithmus wird neben der Geschwindigkeit seiner Komponenten auch durch die Anzahl benötigter Generationen zur Lösung der Aufgabe beeinflusst. Die wichtigsten Parameter zur Steuerung seines Konvergenzverhaltens werden im Folgenden untersucht. Die Trainingsaufgabe wurde im Hinblick auf die für das Neocognitron gegebene Problemstellung ausgewählt und ist diesselbe wie im vorherigen Abschnitt. Eine allgemeine Aussage lässt sich hieraus nicht ableiten.

Mutationsrate $\mathcal{M}$, Ersetzungsobergenze $\mathcal{R}$ In Kapitel 4.3.2 wurde die Abhängigkeit der Geschwindigkeit des SHAGEN-Trainingsalgorithmus von den Parametern Mutationsrate $\mathcal{M}$ und Ersetzungsobergrenze $\mathcal{R}$ untersucht. Abbildung 4.13 hingegen zeigt das Konvergenzverhalten des Algorithmus für verschiedene $\mathcal{M}, \mathcal{R}$. Die Messwerte sind dabei über fünf Durchläufe bei gleichen Trainingsmusterkombinationen gemittelt. Die Eingangsmuster bestehen aus dreizehn, von einem frühen Differenzselektor gefundenen, Satz 3x3 Features. Zwölf der Features sind auch in Abbildung 4.8 zu finden. Die zugehörigen Ausgangssollmuster sind die fürs Neocognitron benötigten dualen Darstellungen der Potenzen der Zahl Zwei. Die Aufgabenstellung ist demnach real. Die Abbildung zeigt, daß eine geringe Mutationsrate im Bereich um $\mathcal{M} \approx 8\%$ maßgeblich für die Trainingsleistung des Algorithmus ist. Entsprechende Ergebnisse zu dem ersten HANNEE-Algorithmus sind vergleichbar und unter [HSSM02] nachzulesen. Oberhalb von $\mathcal{R} \approx 6\frac{Kopien}{Generation}$ scheint die maximale Anzahl von Ersetzungen schlechter Individuen durch das aktuell erfolgreichste Individuum geringen Einfluß zu haben. Anhand des gewählten Ersetzungsschemas (s.h. S.57) lässt sich dies erklären: Mit der Position des schlechtesten Individuums schwankt die Anzahl der Ersetzungen statistisch. Wie bereits erwähnt, soll dies Konvergenz gegen lokale Minima bzw. zu gering eingestellte Konvergenzgeschwindigkeiten verhindern. Die übrigen SHAGEN-Parameter finden sich in der Erläuterung der Abbildung. Das hier präsentierte Ergebnis bestätigt sich auch für geringere Generationenzahlen, siehe dazu beispielsweise Abbildung A.5.

Auflösung der Gewichte Der HAGEN-Chip wandelt die von außen gelieferten 11-Bit Integerzahlen via integrierter DACs in analoge Netzgewichte. Das Verhalten einer einzelnen Synapse ist etwa mit einer Auflösung von 10-Bit reproduzierbar. Unterschiede zwischen den Synapsen verhindern diesselbe Auflösung

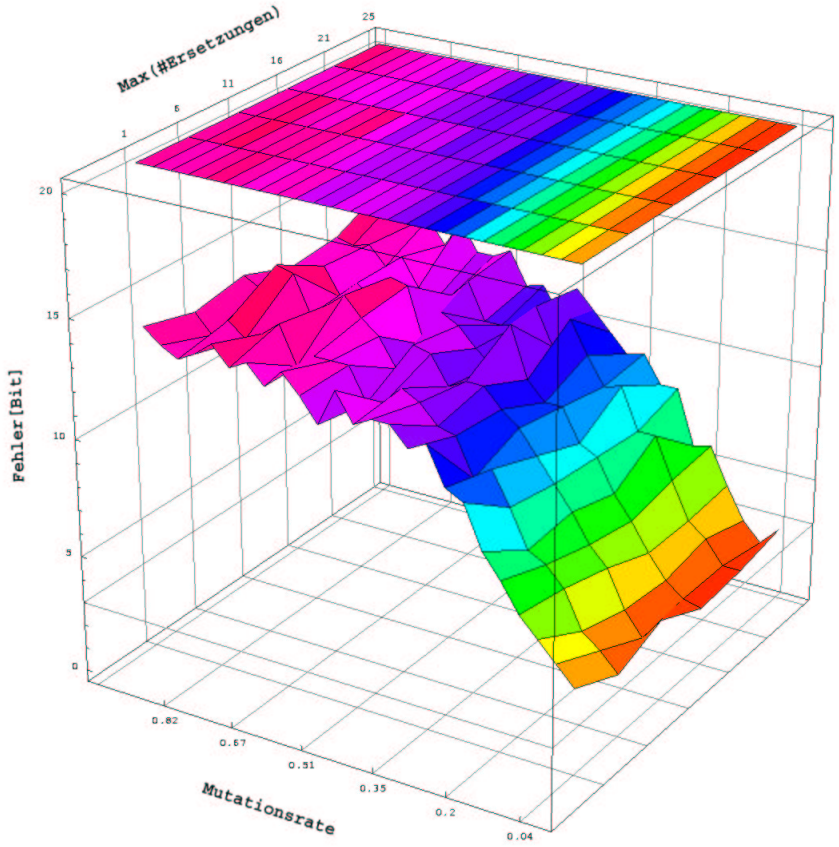

Abbildung 4.13: Der Einfluss der Mutationsrate und der maximalen Anzahl Ersetzungen schlechter Individuen durch das aktuell erfolgreichste Individuum (je Generation) auf den SHAGEN-Algorithmus wurde untersucht. Die dafür verwendete Trainingsaufgabe entspricht der einer ersten Neocognitron-Stufe. Der Fehler bezeichnet hierbei die Anzahl invertierter Punkte der erhaltenen Ausgangsmuster gegenüber den Ausgangssollmustern (nach Abschluss des Trainings). Die Messpunkte sind gemittelt über fünf Durchläufe und die Generationsobergrenze ist konstant 200.000. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Gewichtsauflösung=3Bit und eine zusätzliche Gewichtung fehlender gegenüber überflüssiger Aktivität von eins. Abbildung A.5 zeigt diesselbe Konstellation bei einer Generationsobergrenze von 5.000.

beim Austausch der Gewichtswerte unter ihnen. Durch eine Kalibration können zumindest die Gewichtsoffsets festgestellt und rechnerisch ausgeglichen werden. Trotz Kalibration gehen etwa zwei oder drei Bits Auflösung verloren. Die Frage, inwieweit eine hohe Gewichtsauflösung nötig ist, ist aus drei Gründen interessant:

- Eine geringere Auflösung hat eine schnellere Zufallszahlerzeugung sowie eventuell Zeitgewinne bei der Datenübertragung zur Folge.
- Die Begrenzung der möglichen Gewichtskonstellationen verkleinert den Suchraum erheblich.
- Die innerhalb des SHAGEN-Trainings ermittelten Netzgewichte können bei der Übertragung auf den HAGEN-Chip stärker skaliert werden. Dies erhöht die Wahrscheinlichkeit Fabrikations- oder Umgebungsveränderungen kompensieren zu können.

Das Ergebnis ist erstaunlich deutlich: die minimale Auflösung (oberhalb der wenig sinnvollen 1-Bit-Grenze: hier gibt es keine inhibitorischen Leitungen, kein Vorzeichenbit) zeigt den größten Trainingserfolg für hiesige Problemstellung (Abb. 4.14). Die Trainingsaufgabe ist dabei dieselbe wie für den vorhergehenden Abschnitt. Für die in dieser Arbeit verwendete Generationenzahl ist der Erfolg der 3-Bit-Variante nicht wesentlich geringer. Außerdem ist nicht klar, wie sich Veränderungen der Eingangs/Ausgangsmustergrößen etc. auswirken. Damit der Lösungsraum nicht versehentlich zu stark beschränkt wird, findet die 3-Bit Variante in dieser Arbeit Verwendung.

Gewichtung fehlender gegenüber überflüssiger Aktivität Da die typischen Ausgangssollmuster im Neocognitron lediglich einen aktiven Punkt beinhalten, ist eine Gewichtung fehlender gegenüber überflüssiger Aktivität sinnvoll. Andernfalls neigt der SHAGEN-Trainingsalgorithmus dazu, die Netzgewichte auf Null zu halten, die entscheidende Aktivität gegenüber der Übermacht an Inaktivität zu unterdrücken. Die Funktion, die die Abweichungen der erhaltenen Ausgangsmuster eines Individuum von den Ausgangssollmustern bewertet, kann entsprechend angepasst werden. Die Auswirkung zeigt Abbildung 4.15, die Anzahl zusätzlich vergebener Fehlerpunkte ist gegen die Generation aufgetragen. Doppelte Gewichtung der gewünschten Einsen bringt den größten Erfolg - dies gilt auch über die Generation 50200 hinaus (Abb. A.6). Die genauen SHAGEN-Parameter finden sich in der Erläuterung der Abbildung.

4.3.2.2 Übertragbarkeit auf die Hardware

Um die Übertragbarkeit der innerhalb des SHAGEN-Pakets gelernten Gewichtskonstellation zur Lösung einer HAGEN-Aufgabe auf den HAGEN-Chip zu über-

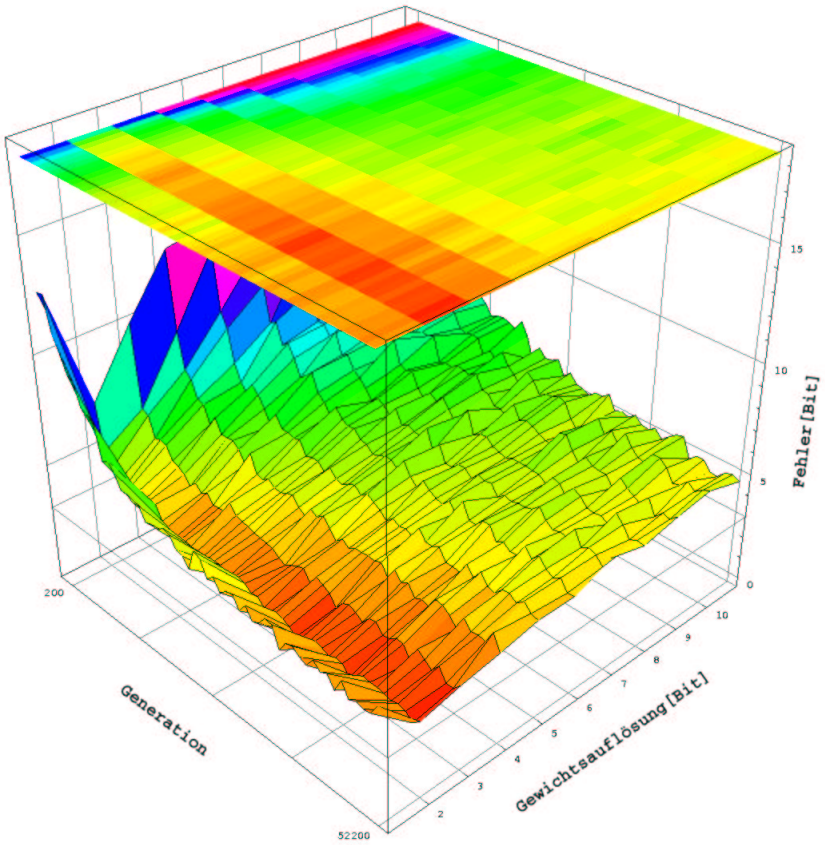

Abbildung 4.14: Eine Begrenzung der Bit-Länge der während der Mutationsphase erzeugten Zufallszahlen bewirkt die entsprechende Herabsetzung der Gewichtsauflösung. Dies schränkt den Suchraum ein - existiert eine Lösung in diesem Raum, wird sie im Mittel vom Trainingsalgorithmus schneller entdeckt als bei voller 11-Bit (HAGEN-) Auflösung. Die Aktivitätssummation erfolgt innerhalb der SHAGEN-Simulation mit 16-Bit. Überläufe sind demnach nicht möglich. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Ersetzungsobergrenze=25 und eine Mutationsrate von $\frac{30}{255}$.

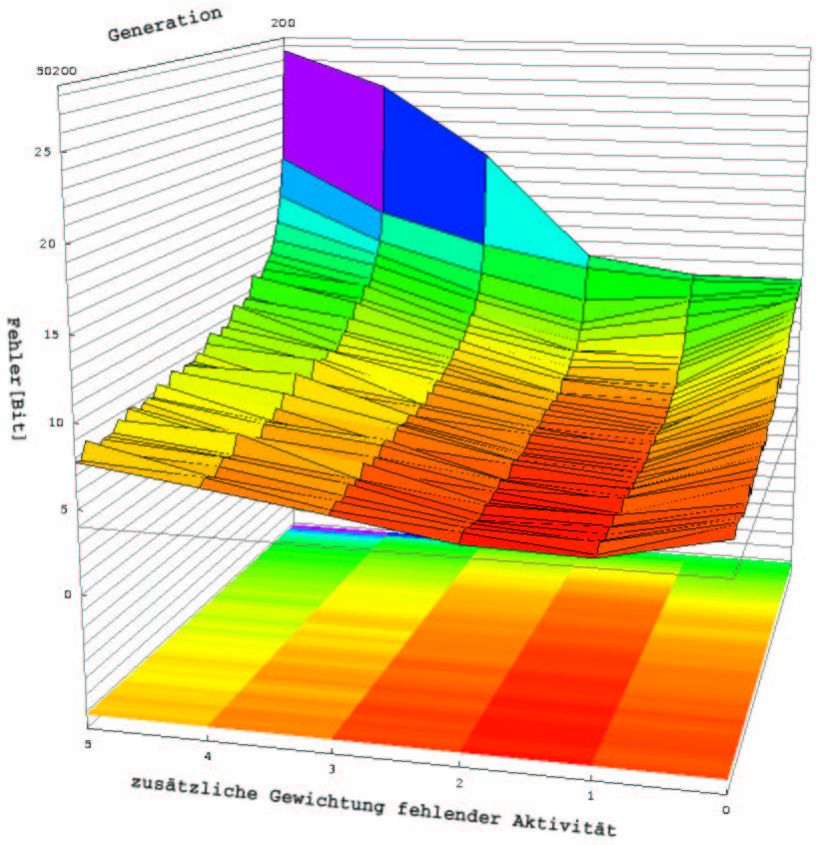

Abbildung 4.15: Da die typischen Ausgangssollmuster im Neocognitron lediglich einen aktiven Punkt beinhalten, ist eine Gewichtung fehlender gegenüber überflüssiger Aktivität sinnvoll. Null entspricht dabei keiner besonderen Gewichtung. Ist die zusätzliche Gewichtung fehlender Aktivität hingegen eins, so wird jeder fehlende Punkt im Ausgangsmuster mit einer (gegenüber den überflüssigen Punkten) zusätzlichen Fehlereinheit gewichtet. "Fehler" bezeichnet hier die Abweichung der trainierten Ausgaben von den Sollausgaben. Ein geringer Fehler entspricht einer guten Lernleistung. Die Lernaufgabe entspricht einer typischen ersten Neocognitron-Stufe, und die Messwerte sind gemittelt über 50 Durchläufe. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Ersetzungsobergrenze=25, Gewichtsauflösung=7Bit und eine Mutationsrate von $\frac{30}{255}$. Einen Einblick in den Bereich oberhalb 50.000 Generationen zeigt Abbildung A.6.

prüfen, wurden die Gewichte nach dem SHAGEN-Training über die MathLink-Schnittstellen direkt zur Hardware übertragen (Abb. 4.16). Die Gewichtsauflösung für SHAGEN betrug für das Training 3-Bit. Nach erfolgtem Training wurden die Gewichte auf 11-Bit skaliert und via MathLink an das HANNEE-Programm weitergereicht (Abb. 4.18). Innerhalb HANNEEs ist es möglich, den Spannungsbereich auszuwählen, auf den es diese Gewichtswerte anschließend skaliert. Der Skalierungsfaktor Eins in der Abbildung bedeutet folglich, daß das SHAGEN-Gewicht *111* die größtmögliche Spannung zur Steuerung der HAGEN-Gewichtsladung bedient. Mit zunehmender Skalierung der Gewichtswerte werden die Differenzen zwischen den Strömen (Kap. 2.4) auf dem HAGEN-Chip größer - das Signal-zu-Rauschverhältnis verbessert sich. Die Lösung wird dann entsprechend stabiler (Abb. 4.17). Der Übertragungstest besteht aus folgenden Schritten:

- Training von SHAGEN auf die Klassifikation von sechs zufällig gewählten 9-Bit Eingangsmustern; für die Einteilung der sechs Eingangsmuster in verschiedene Klassen werden ihnen 6-Bit Ausgangsmuster zugewiesen - jedes Bit signalisiert eine Klasse.

- Die SHAGEN-Gewichte werden skaliert in den HAGEN-Chip geschrieben.

- Den sechs Eingangsmustern werden sechs zusätzliche, ebenfalls zufällige, Eingangsmuster hinzugefügt - um die Übereinstimmung der Generalisierung ebenfalls zu prüfen.

- Die zwölf Eingangsmuster werden SHAGEN und HAGEN präsentiert, die jeweiligen Antworten verglichen.

- Die Anzahl verschiedener Ausgangsbits ist in Abbildung 4.16 aufgetragen; drei unterschiedliche Bits entsprechen dabei etwa $\frac{3}{72} \cdot 100\% \approx 4,2\%$ Fehler.

4.3.2.3 Zusammenfassung

Die Abschätzung der theoretischen Geschwindigkeitsbeschränkungen der Trainingskomponenten ergab, daß der Optimierung der in der Entwicklung befindlichen HANNEE-Routinen seitens der Hardware nichts im Wege steht. In der Folge beziehe ich mich auf die anfangs des Kapitels erwähnte Lernaufgabe (s.h. $\star$). Diese besteht aus 15 Eingangs/Ausgangsmusterkombinationen bei der Populationsgröße von 30 Individuen mit einer Generationsobergrenze von 10.000 Generationen. Die HAGEN-Taktung von etwa 50MHz genügt, um ca. $50.000.000/(29 \cdot 15 \cdot 2 \cdot 10) \approx 5800$ Generationen in der Sekunde zu bewerten. Der Faktor Zwei im Nenner steht für die zwei benötigten HAGEN-Zyklen zur Bereitstellung eines zweischichtigen Perzeptrons. Der Faktor zehn gibt nur die Größenordnung der zur Hardware-ausgabenstabilisation ("Aufwärmphase") benötigten Leerdurchläufe an - er mag

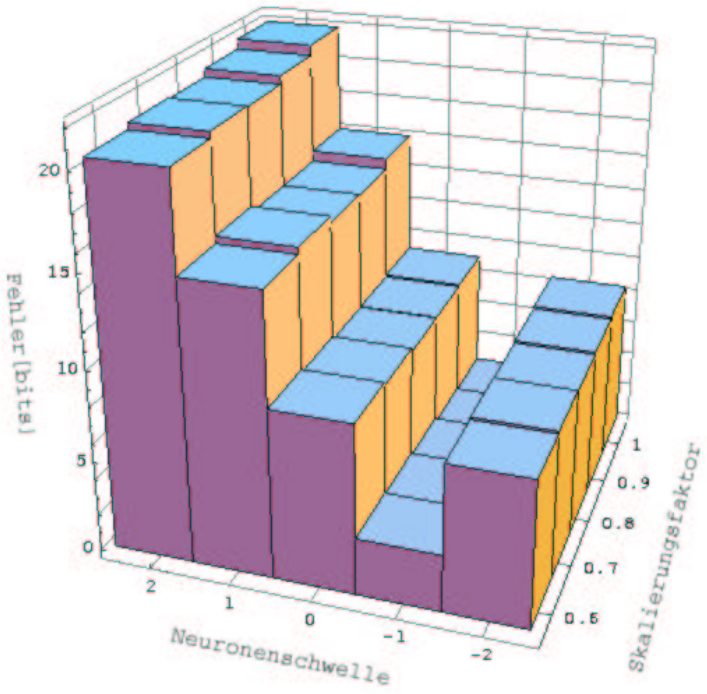

Abbildung 4.16: Werden die innerhalb SHAGEN gelernten Gewichte skaliert auf die Hardware HAGEN übertragen, so ist der Unterschied zwischen den SHAGEN-Ausgaben und den HAGEN-Ausgaben bei einer SHAGEN-Neuronenschwelle (bereits während des Trainings) von -1 minimal. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=3, Ausgänge=6, Generationsobergrenze=10000, Populationsgröße=30, Ersetzungsobergrenze=25, eine zusätzliche Gewichtung fehlender Aktivität von Eins und eine Mutationsrate von $\frac{20}{255}$. Für den Vergleich wurde neben den sechs Trainingsmustern ein Satz von sechs zusätzlichen Eingangsmustern verwendet. Alle Muster wurden zufällig generiert.

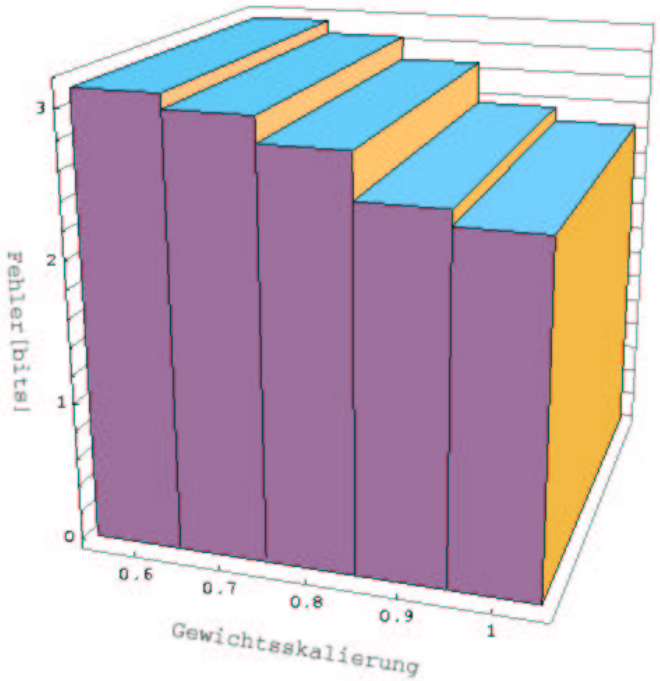

Abbildung 4.17: Werden die innerhalb SHAGEN gelernten Gewichte skaliert auf die Hardware HAGEN übertragen, so ist der Unterschied zwischen den SHAGEN-Ausgaben und den HAGEN-Ausgaben bei einer SHAGEN-Neuronenschwelle (bereits während des Trainings) von -1 minimal. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=3, Ausgänge=6, Generationsobergrenze=10000, Populationsgröße=30, Ersetzungsobergrenze=25, eine zusätzliche Gewichtung fehlender Aktivität von Eins und eine Mutationsrate von $\frac{20}{255}$. D.h. eine Reduzierung der Gewichtsauflösung innerhalb des SHAGEN-Trainings auf einige wenige Bits sind günstig für die spätere Übertragung auf den HAGEN-Chip.

bei geschickter Hardwareansteuerung wesentlich kleiner ausfallen. Die theoretische Obergrenze für die Mutationsphasenfrequenz auf HAGEN ist zwar mit etwa $(\frac{12,2}{29} \approx 0.42)kHz$ (DACs) deutlich ungünstiger, die am Kapitelanfang genannte Lernaufgabe (s.h. $\star$) ließe sich so aber noch immer in ungefähr 24 Sekunden (gegenüber derzeit 1800s) bewältigen. Der PCI-Bus hebt die benötigte Zeit auf etwa 150s und bewahrt trotzdem einen Faktor oberhalb von zehn gegenüber dem HANNEE-Training. Die SHAGEN-Vergleichsmessung benötigte für das gesamte Training obengenannter Lernaufgabe 46 Sekunden. Dieser Wert ist im Gegensatz zur theoretischen DAC-Begrenzung gemessen. Überdies ist eine weitere Optimierung des SHAGEN-Trainings möglich (interne Verwendung von 4-Bit Zufallszahlen→Faktor 4, die Umsetzung der HAGEN-Simulation in Assembler, die Verwendung neuerer PC-Prozessoren, ...). Somit ist das SHAGEN-Paket in der Lage, die mit HAGEN maximal möglichen Generationenfrequenzen zu erreichen (die Größenordnung stimmt bereits jetzt überein). Die Übertragung der SHAGEN-Trainingsergebnisse zum HAGEN-Chip scheint auch in Anbetracht der guten Übereinstimmung zwischen SHAGEN und HAGEN-Ergebnissen (etwa 4% Fehler) reizvoll. Ein Nachtraining auf dem Chip könnte parallel zum Softwaretraining laufen. Nicht nur aus diesem Grunde sind die derzeitigen Bestrebungen der *Electronic Vision(s)* Gruppe weg von dem Engpass PCI-Bus hin zu einem FPGA-basierten Training mit LVDS-Anbindung zum HAGEN-Chip ein Schritt in die richtige Richtung [Gru03]. Ein Geschwindigkeitsschub in der theoretischen Begrenzung durch die Hardware um etwa den Faktor sechs ist damit möglich.

4.4 Die Vernetzung der Programme

Die Implementation des in dieser Arbeit vorgestellten binären Neocognitrons und die zugehörige Testumgebung bestehen aus drei weitestgehend voneinander unabhängigen Programmen: BONNEE, SHAGEN und HANNEE(erweitert um eine entsprechende Schnittstelle). Im Kontext objektorientierter Entwicklung sind diese Programme Objekte mit klar definierten Schnittstellen. Die Schnittstellen bieten sogenannte Interfaceklassen zur direkten Ansteuerung innerhalb der jeweiligen Entwicklungsumgebung und außerdem eine MathLink-Schnittstelle an. MathLink ist ein von der Firma Wolfram entwickeltes TCP/IP[19]-fähiges, proprietäres[20] Protokoll zur Kommunikation zwischen Mathematica[21]-Programmen und externem Kode. Beispielsweise C-Routinen können so aus Mathematica heraus angesprochen werden. Umgekehrt lässt sich Mathematica-Kode mittels MathLink fernsteuern - die Mathematica-GUI[22] ist beispielsweise auf diese Art und Weise mit dem Mathematica-Kernel verbunden. Durch die Verwendung von MathLink können al-

[19] **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol.
[20] Ein nicht standardisiertes Verfahren zur Bewältigung spezieller Aufgaben.
[21] Eine Entwicklungsumgebung der Firma Wolfram.
[22] **G**raphical **U**ser **I**nterface.

le drei Programme auf verschiedenen Rechnern ausgeführt und über das Internet verbunden werden. Außerdem ist es damit leicht möglich, die Programme Schritt für Schritt gegen neuere Versionen auszutauschen. Im allgemeinen wird jedoch die Verbindung zum SHAGEN-Paket automatisch aus Mathematica heraus und somit auf dem selben Computer installiert (so auch in Abbildung 4.18 dargestellt). Der Benutzer wird um das Eintippen von Portnummern[23] erleichtert. Soweit es die Neocognitron-Struktur zulässt, steht grundsätzlich der Parallelisierung von Mathematica-Kode mit den externen Komponenten, und damit der Verteilung von Rechenlast auf verschiedene Rechner, nichts im Wege.
HANNEE ist der Name der bereits bestehenden Steuerung des HAGEN-Chips via GUI. Ihr wurde eine minimale MathLink-Schnittstelle aufgesetzt. Sie macht diese Arbeit weitestgehend unabhängig von den täglichen Veränderungen durch die Weiterentwicklung des HANNEE-Projekts und bietet durch die Möglichkeit der Fernsteuerung eine einfache Automatisierung der HAGEN-Ansteuerung durch Mathematica-Kode. Die Schnittstelle beinhaltet außerdem einen Puffer zur Zwischenspeicherung bereits gelernter Trainingskombinationen und stellt den vom Training unabhängigen Betrieb des Hagen-Chips bereit.
SHAGEN ist ein Softwarepaket bestehend aus HAGEN-Simulation, Simulationsbasiertem Training und MathLink-Schnittstelle. Dieses Paket beschleunigt das Training Neocognitron-typischer Aufgabenstellungen etwa um den Faktor 40. Über die Math-Link Schnittstellen können seine Netzgewichtskonstellationen direkt auf den HAGEN-Chip übertragen werden.
Die zentrale Steuereinheit BONNEE verbindet die Komponenten, bietet seinerseits Basisoperationen des binären Neocognitrons und stellt ein umfangreiches Auswertesystem zur Verfügung. BONNEE ist im Gegensatz zu den übrigen Komponenten Mathematica-basiert (s.h. Abbildung 4.18).

[23] Verbindungsadressen der Interprozeßkommunikation.

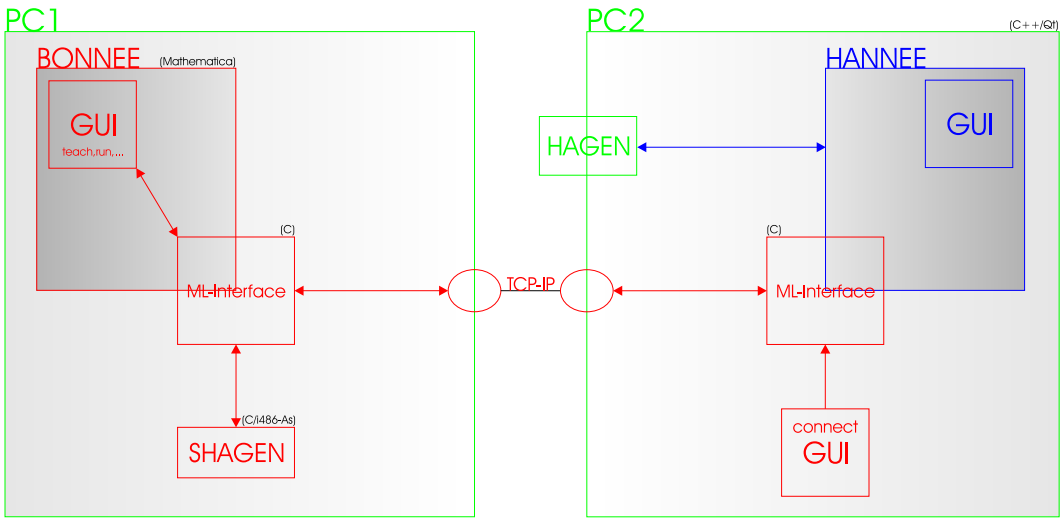

Abbildung 4.18: Schematische Darstellung der Vernetzung der in dieser Arbeit benötigten Strukturen. Grün dargestellt ist die Hardware, blau die bereits bestehende Programmstruktur und rot die für diese Arbeit entwickelten Programmteile. GUI steht für **G**raphical **U**ser **I**nterface. An den jeweiligen rechten oberen Ecken sind die Entwicklungsumgebungen des zugehörigen Paketes angegeben.

Kapitel 5

Ergebnis

5.1 Das beste Individuum

Die Wahl der Eingangsdaten zur Überprüfung der Funktion des binären Neocognitrons fiel auf handgeschriebe Ziffern. Sie stammen aus einer MNIST genannten, via Internet zugänglichen Datenbank von Yann LeCunn [LeC03]. Unter Verwendung der automatischen Netzstrukturoptimierung innerhalb BONNEEs konnten 155 verschiedene Neocognitron-Topologien kreiert werden. Dabei wurde hauptsächlich nach guten Parameter für den *kombinierten Selektor* gesucht. Die Dimensionierung der Fokusbereichsgrößen der Stufen wurden manuell nur geringfügig verändert. Anhand der Datumsangaben in den Namen der automatisch erzeugten Individuen ist zu ersehen, daß die ersten tauglichen Netzkonstellationen kurz vor Beendigung dieser Arbeit erzeugt wurden. Der Rechenaufwand zur Konstruktion eines Neocognitron-Individuums, nicht zuletzt durch den hohen Ressourcenverbrauch des Mathematica-Umfeldes, ist enorm - ebenso wie die damit verbundenen Testzeiten. Kleine Programmkorrekturen und die Suche nach einem initialen Parametersatz mit dem die Parameterschleife ohne Komplikationen bis zur letzten Stufe durchlaufen kann, hat bis kurz vor Abschluss der Arbeit angedauert. 102 der Individuen konnten im zeitlichen Rahmen der Arbeit anhand der ihnen zugrundegelegten Trainingsdaten auf ihren möglichen Erfolg überprüft werden. Die Erkennungsrate einer Auswahl von 33 Individuen wurde unter Verwendung von 100 Bildern pro gelernter Ziffer ausgewertet. Sechs Topologien konnten mit 300 Bildern je Ziffer bewertet werden. Obwohl die Suche nach einer optimalen Netzstruktur für das binäre Neocognitron somit am Anfang steht, konnten einige Parametersätze ermittelt werden, die mit bis zu 84% korrekter Klassifikation die Ziffern Null und Eins trennen können (Tab. 5.1).
Die vorgegebenen (Hyper-) Fokusbereichsgrößen für die S-Schichten sind dabei bis einschließlich zur dritten Stufe Vielfache der Größe (3x3), die der Stufen Vier, Fünf und Sechs Vielfache der Größe (4x4). Die siebte und letzte Stufe besteht aus einer L-Schicht deren Fokusbereich alle Ausgänge der vorhergehenden

Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-10.5.2003-15:51.31	0 1	1	300	0,8433	0,125	0,03167
V2.5-10.5.2003-5:12.12	0 1	1	300	0,7167	0,2283	0,055
V2.5-10.5.2003-6:59.51	0 1	1	300	0,7083	0,255	0,03667

Tabelle 5.1: Die besten Neocognitrontopologien zur Trennung von Nullen und Einsen erreichen etwa 80% Erkennungsrate. Sie wurde mit 300 Nullen sowie 300 Einsen überprüft (beginnend mit dem ersten Datenbankeintrag). Die zugehörigen Dateien finden sich im Unterverzeichnis Evolver10 auf der beigefügten CD(s.h. Kap. B).

Stufe abdeckt. Die Fokusbereichsgrößen der C2- und C3-Schicht sind Vielfache von (3x3), die der folgenden C-Schichten Vielfache von (2x2). Die mit den Fokusbereichsgrößen korrespondierenden Begrenzungen der Zahlen von Features der Stufen sind $fConstraints = \{[10,13],[7,13],[4,7],[3,7],[3,7],[3,4]\}$ für die ersten sechs Stufen. In der sechsten Stufe besteht der Feature-Satz demnach aus drei oder vier Features. Die SHAGEN-Parameter sind: Anzahl Rückkopplungen=4, Neuronenschwelle=-1, Mutationsrate=$\frac{20}{255}$, Generationsgröße=30, Generationsobergrenze=200.000 und die maximale Anzahl Ersetzungen ist 25. Die Trainingsdaten bestanden aus zwanzig Eingangsbildern, zehn Nullen und zehn Einsen. Die L-Stufe einiger Individuen, darunter auch jene des bereits erwähnten erfolgreichsten, wurden mehrfach trainiert. Die vier Auswertungen des Besten brachten nahezu identische Ergebnisse (s.h. Tabelle 5.1, A.1 und A.2).

Die Selektorparameter für das beste Individuum sind:

$Stufe\ 1:$	0.005	0.005	0	0.3	1	0.15
$Stufe\ 2:$	0.005	0.005	0	0.558677	2	0.107554
$Stufe\ 3:$	0.00298067	0.0137149	0.0000236689	0.3	2	0.0982769
$Stufe\ 4:$	0.00857937	0.010089	0.000120387	0.656222	4	0.0889593
$Stufe\ 5:$	0.0110611	0.00884603	0.00004894	0.42086	4	0.0378201
$Stufe\ 6:$	0.00317424	0.00280185	0.000117896	0.033268	5	0.0950864.

Genaue Daten der übrigen Individuen sind in den entsprechenden ASCII[1]-Dateien via Texteditor nachzulesen. Ein Teil ihres Namens ist in den Tabellen angegeben. Ihre Endung ist *.syms* (s.h. Kap. B).
In Abbildung 5.3 (sowie A.9, A.10 und A.11) sind die Ausgaben aller Zellen des zur besten Konfiguration gehörenden Neocognitrons dargestellt. Das Bild in der ersten Zeile stellt die C1-Schicht dar. Sie besteht aus einer Ebene - dem Eingangsbild. Die zweite Zeile zeigt die S1-Schicht. Ihre neun Ebenen korrespondieren mit den neun Features der ersten Stufe. Da die S1-Schicht ihre Eingangsdaten nicht aus Hyperfokusbereichen bezieht, sondern vielmehr direkt aus Ausschnitten des Eingangsbildes, kann man in dieser Schicht (sowie der nachfolgenden C-Schicht)

[1] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

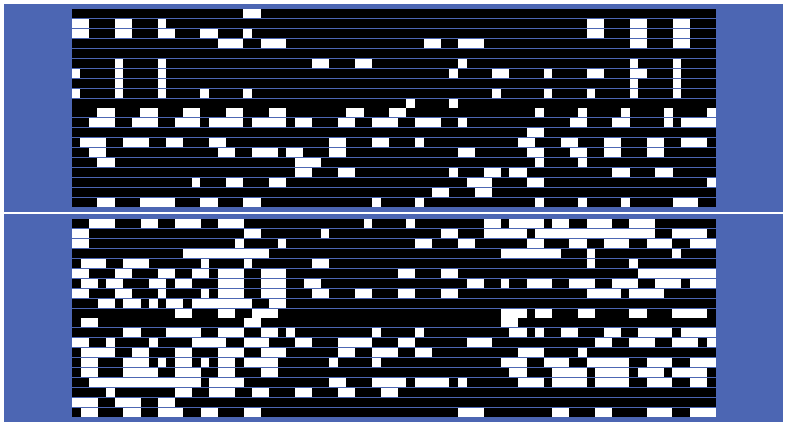

Abbildung 5.1: Die Ausgaben der letzten C-Schicht (vor der Klassifizierungsschicht L) des besten Individuums aus Tabelle 5.1 und eines mittelmässigen Individuums (AllvLindies/bonneeV2.5-10.5.2003-0:14.19.200351695625.vLindy s.h. letzte der Tabellen A.2). Die obere Grafik stellt das bessere Individuum dar. Die Ausgaben innerhalb der Grafiken sind zeilenweise geordnet. Die zugehörigen Eingabebilder sind die Trainingsdaten selbst, d.h. die ersten zehn Zeilen beider Grafiken gehören zu den verschiedenen Bildern einer Ziffer und die zweiten zehn zu jenen der anderen Ziffer. Im Vergleich der oberen mit der unteren Hälfte einer Grafik, sollten für ein gutes Individuum demnach Systematiken erkennbar sein. Für das bessere der beiden Individuen scheint dies eher der Fall zu sein.

erahnen, weshalb an jeweiliger Position ein bestimmtes Feature entdeckt wurde. Die folgenden Zeilen sind abwechselnd C- und S-Schichten. Die letzte Zeile allerdings ist eine L-Schicht. Ist die Aktivität in ihrer linken Hälfte größer als die der rechten Hälfte, so wird das Eingangsbild als Null klassifiziert. Überwiegt die rechte Hälfte, wird eine Eins erkannt. Im vorliegenden Falle ist die Klassifikation demnach korrekt erfolgt. In den Abbildungen A.9 und A.11 sind Fehlklassifikationen dieses Individuums gezeigt.
Es fällt auf, daß einige Ebenen bei keiner der Ziffern Aktivität zeigen. Ein Anzeichen für die Imperfektion des gefundenen Parametersatzes. Abbildung 5.1 zeigt die Aktivitäten der letzten C-Schicht des besten und eines mittelmässigen Individuums. Die ersten zwanzig Zeilen gehören zu dem erfolgreichsten Individuum, die zweite Abbildungshälfte zur schlechteren Topologie. Jede Zeile entspricht dabei der letzten C-Schicht bei Eingabe eines bestimmten Bildes aus dem Trainingsdatensatz. Das oberste Viertel in der Abbildung ist denselben Ziffern zugeordnet wie das dritte Viertel, für die übrigen beiden Viertel gilt selbiges (für Bilder einer anderen Ziffer). Anders ausgedrückt: Die oberen Hälften der beiden Grafiken in der Abbildung korrespondieren miteinander ebenso, wie die beiden unteren Hälften. Umso größer der Unterschied zwischen den Hälften der Grafik eines Individuums und umso geringer die Unterschiede der Zeilen einer Hälfte,

umso einfacher wird der Klassifikationsprozeß für die folgende L-Schicht. In der Tat scheint das obere Individuum diese Kriterien besser zu erfüllen.
Bis hierher sind alle Ergebnisse ausschließlich mittels des SHAGEN-Pakets entstanden. Wie in diesem Kontext die Hardware HAGEN angesprochen werden kann, wurde in Kapitel 4.3.2 bereits erläutert. Die in SHAGEN gefundenen Individuen werden dazu einfach via MathLink zu HAGEN übertragen (Abb. 4.18). Die Schnittstellen von SHAGEN und HAGEN sind nahezu identisch und die Einbindung beider in BONNEE entsprechend ähnlich. Die Prozedur zur Evaluation eines Individuums ist bis auf die zusätzliche, manuelle Einstellung des HANNEE-Programms gleich. Einen auf diese Weise gesteuerten Hardwaredurchlauf zeigt Abbildung A.12. Es fällt ins Auge, daß das Ergebnis deutlich schlechter ist als bei der SHAGEN-Evaluation. Wie Kapitel 4.3.2.2 darlegt, ist die Fehlerrate bei den dort überprüften Lernaufgaben in der Größenordnung von 4%. Sie kann durch ein kurzes Nachtraining auf der Hardware behoben werden. Dieses konnte allerdings im Rahmen der Arbeit aus Zeitgründen nicht implementiert werden. Grundsätzlich sind hier keine Schwierigkeiten zu Erwarten. Lernfehler treten bei Einsatz des HAGEN-Chips somit derzeit in jeder Stufe auf - in der Summe genügt dies um die Ausgabe der letzten C-Ebene praktisch unbrauchbar zu machen. Obwohl die Erkennung direkt mit HAGEN somit statistisch nicht nachgewiesen werden konnte, so konnte doch ausreichend belegt werden, daß die in vorliegender Arbeit entwickelten Strukturen dazu fähig sind (bei zukünftiger Einbindung eines HAGEN-Nachtrainings).

5.2 Verbesserungsmöglichkeiten

Da die in dieser Arbeit aufgebauten Strukturen ein Erstsystem mit Prototypcharakter darstellen und für keine der Komponenten Vorarbeiten existieren, ist viel Spielraum für Verbesserungen vorhanden. Einige Verbesserungsvorschläge werden nun genannt und es wird deutlich, daß die Leistung des Systems sicherlich steigerungsfähig ist:

- Aufhebung der Begrenzung der Anzahl möglicher Features durch die Kopplung mehrerer Hagen-Blöcke (s.h. auch Kap. 3.2).

- Systematische Parametersuche für alle Kombinationen aus Fokusbereichsgrößen und Featurezahlbegrenzungen der Stufen.

- Repräsentation der Klassen durch mehrere Feature-Gruppen schon in den ersten Stufen (Kap. 3.2.3).

- Statistische Optimierung der der letzten Stufe angehängten Vergleichs-Schwelle, die über die Klassifikation entscheidet (Kap. 3.2.3).

- Verwendung eines größeren Satzes Trainingsdaten. In dieser Arbeit war eine Beschränkung auf etwa 30 Trainingsbilder wegen der benötigten Rechenzeit unumgänglich.
- Verbesserung des SHAGEN/HAGEN-Trainingserfolges durch z.B. Auswahl der Features im Hinblick auf echte lineare Unabhängigkeit (Kap. 4.2.3).
- Einführung einer den V-Zellen ähnlichen Instanz.
- Rückkopplungen über die Neocognitronstufen hinweg.

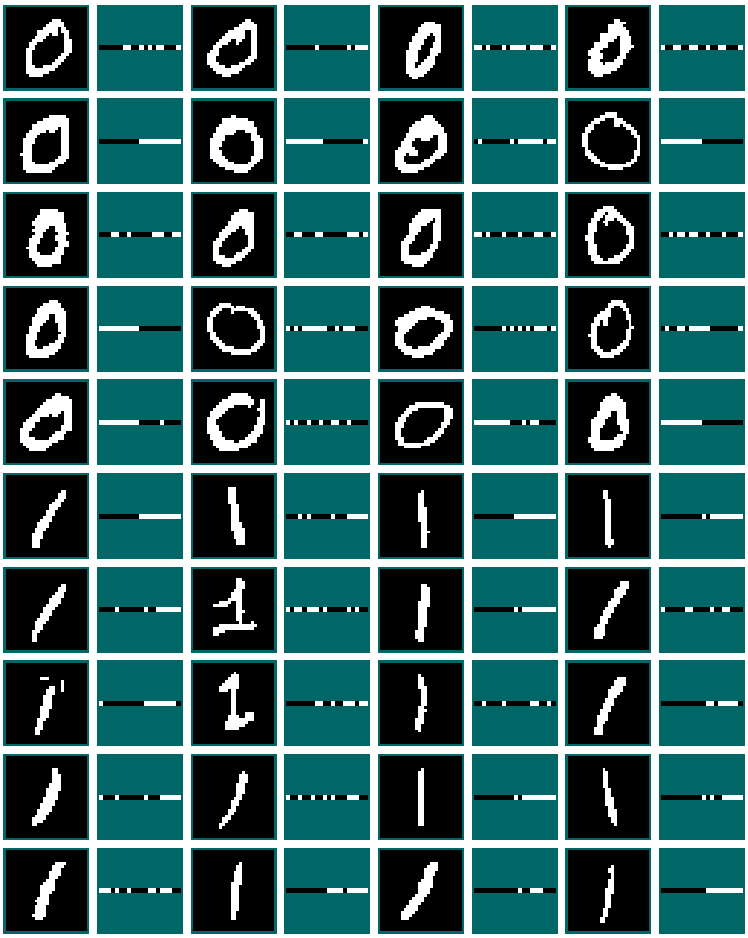

Abbildung 5.2: Die ersten 20 Bilder der beiden Ziffern Null und Eins - ausgewertet mit dem besten Individuum aus Tabelle 5.1. Die horizontalen Balken neben den Bildern sind die Ausgaben der L-Schicht. Ihre linke Hälfte steht für die Feature-Gruppe der Nullen wohingegen ihre rechte Hälfte die der Einsen darstellt. Überwiegt die Aktivität in einer der beiden Gruppen, gilt das Bild als entsprechend klassifiziert 3.2.3.

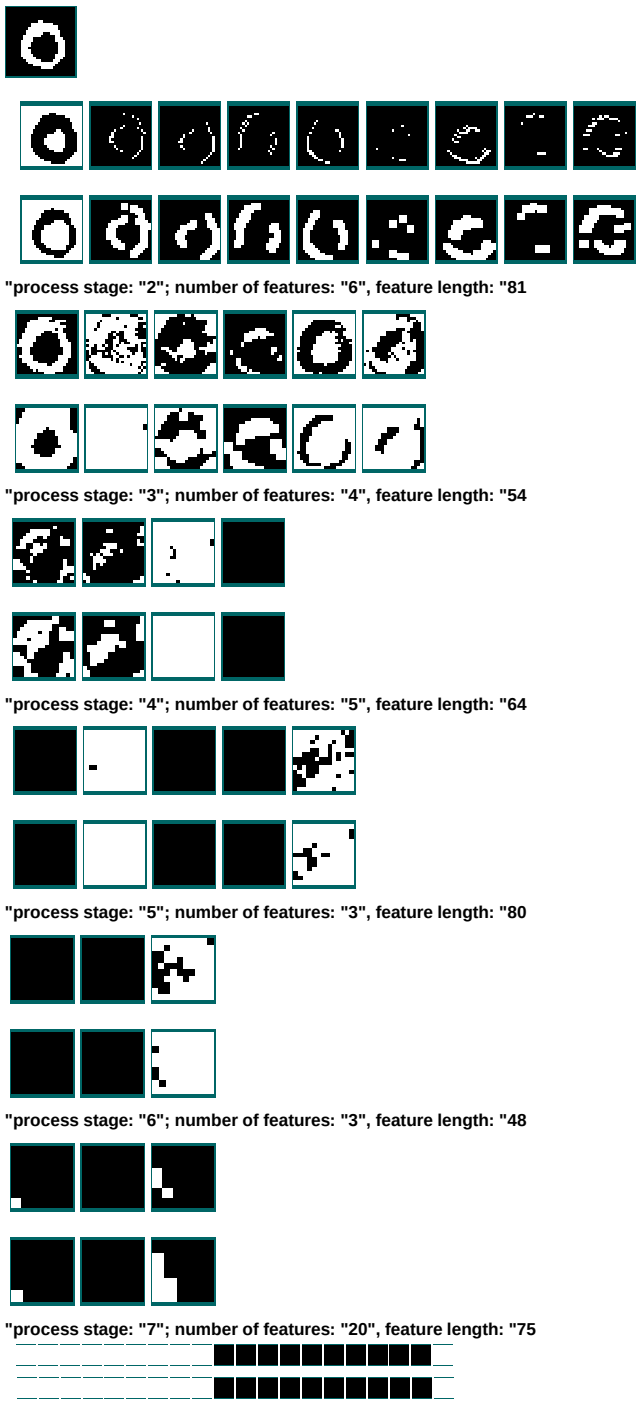

Abbildung 5.3: Die Ausgaben aller Stufen des besten Individuums aus Tabelle 5.1. Das Eingangsbild ist eine Null. Die erste Stufe besitzt neun Features. Auf das Eingangsbild folgt die erste S-Schicht. Ihre Aktivität zeigt an welchen Positionen das einer Ebene zugeordnete Feature entdeckt wurde. Die dritte Zeile zeigt die C2-Schicht und so fort. Streng genommen sind die Ausgaben "process stage" jeweils eine Zeile zu tief - sie beziehen sich lediglich auf die S-Schichten und ihre Position hat historische Gründe. Die Ausgabe der letzten Schicht, der L-Schicht, teilt sich in zwei Hälften. Die ersten zehn Punkte stehen für Features aus der Feature-Gruppe *Null*, die zweiten zehn Punkte sind der Feature-Gruppe *Eins* zugeordnet. Die Hälfte mit größerer Aktivität bestimmt die Klassifikation.

Kapitel 6

Zusammenfassung und Ausblick

Ziel der Arbeit war der Aufbau eines binären, auf den neuronalen Netzwerkchip HAGEN abgestimmten Neocognitrons. Neben der Bereitstellung der Neocognitron-Basisoperationen und der Hardwareansteuerung waren die Entwicklung eines datentypunabhängigen Trainings und einer automatischen Netzstrukturoptimierung von zentraler Bedeutung. Das Training spaltet sich dabei auf in die Suche geeigneter Features für die einzelnen Neocognitronstufen und das Training des HAGEN-Chips auf deren korrekte Klassifikation.
Neben der geforderten Datentypunabhängigkeit des binären Neocognitrons erforderte auch die Tiefe seiner Netzstruktur die Konzeption eines automatischen Feature-Selektors. Die Feature-Selektion ist für größere Aufgabenstellungen manuell nicht durchführbar. Der in Kapitel 4.2.4 vorgestellte *kombinierte Selektor* erfüllt die an ihn gestellten Forderungen: Seine Auswahl der Feature-Sätze soll einerseits die Trainingsdaten möglichst verlustfrei abstrahieren, andererseits muß sie vom HAGEN-Chip ausreichend gut lernbar sein. Die Übereinstimmung der von ihm selektierten Features für die erste Stufe eines Neocognitrons mit manuell zusammengestellten Feature-Sätzen ist bemerkenswert (Abb. 4.7).
Da mit dem HAGEN-Chip ein mehrstufiges Perzeptron implementiert wird, können gängige Lernalgorithmen eingesetzt werden. Die bereits bestehenden evolutionären Trainingsalgorithmen des sich in der Entwicklung befindlichen HANNEE-Programms konnten nicht angewendet werden - ihr Zeitbedarf ist zu hoch. Das im Rahmen der Arbeit entwickelte SHAGEN-Paket löst dieses Problem mittels eines geschwindigkeitsoptimierten Algorithmus auf Basis einer HAGEN-Simulation. Durch den Einsatz von Assemblerroutinen konnte die Anzahl der in einer vorgegebenen Zeit erreichbaren Generationen um den Faktor 40 erhöht werden. In Kapitel 4.3.2.3 konnte überdies gezeigt werden, daß die Übertragung der mittels SHAGEN gelernten Netzgewichtskonstellationen zum HAGEN-Chip auch über die Prototypphase hinaus reizvoll ist. Die theoretische Geschwindikeitsbegrenzung des Trainings auf der Hardware HAGEN durch dessen DACs wird von SHAGEN schon jetzt nahezu erreicht. Ferner stimmt der HAGEN-Chip mit der Simulation gut überein (Kap. 4.3.2.2). Eine Parallelisierung des SHAGEN-Trainings mit einem

Nachtraining auf der Hardware scheint erfolgsversprechend. Die Übertragungswege zwischen SHAGEN und HAGEN sind entwickelt und getestet (Kap. 4.4).
Die Funktionalität des Aufbaus konnte am Beispiel der Erkennung handgeschriebener Ziffern demonstriert werden. Unter Verwendung der automatischen Netzstrukturoptimierung des BONNEE-Programms (Kap. 4.3) konnten insgesamt 155 verschiedene Netztopologien auf SHAGEN kreiert werden. 102 dieser Individuen konnten dabei im zeitlichen Rahmen der Arbeit anhand der ihnen zugrundegelegten Trainingsdaten auf ihren möglichen Erfolg überprüft werden. Die Erkennungsrate einer Auswahl von 33 Individuen wurde unter Verwendung von 100 Bildern pro gelernter Ziffer ausgewertet. Sechs Topologien konnten mit 300 Bildern je Ziffer bewertet werden. Die erfolgreichste Kombination ist ein auf die Klassifikation der Ziffern Null und Eins trainiertes Neocognitron (Tab. 5.1). Seine Erkennungsrate von 84% übertrifft die Erwartungen ebenso wie die geringen 12% Fehl-Klassifikationen (Kap. 3.2). Erste Versuche mit größeren Ziffernmengen lassen hoffen, sind jedoch nicht in ausreichendem Umfang möglich gewesen. Weil die entwickelten Programme Prototypcharakter haben, bleibt genügend Verbesserungsspielraum (Kap. 5.2).
Die Aufhebung der Begrenzung auf einen HAGEN-Block und die damit verbundene Begrenzung auf maximal dreizehn 3x3-Features muß diesbezüglich hervorgehoben werden. Es konnte außerdem gezeigt werden, daß die hier mit SHAGEN gewonnenen Ergebnisse ohne grundsätzliche Schwierigkeiten auf HAGEN übertragen werden können (Kap, 4.3.2.2).
Aufgrund der Skalierbarkeit HAGENs über die Grenzen einzelner Chips hinaus und die damit verbundene massive Parallelisierbarkeit sehe ich das binäre Neocognitron zukünftig vor allem in der Verarbeitung großer Datenmengen angesiedelt. Die Vorselektion großer Datenbestände oder die hierarchische Bildglättung können zukünftig sinnvolle Einsatzgebiete sein.

Literaturverzeichnis

[AMD02] AMD. *AMD Athlon Processor x86 Code Optimization Guide.* http://www.amd.com. 2002

[BC70] Blakemore, C. ; Cooper, G.F.: Development of the brain depends on the visual environment. In: *Nature* (1970), S. 477–478

[BDG81] Bruce, C. ; Desimone, R. ; Gross, C.G.: Visual properties of neurons in a polysensory area in superior temporal sulcus of the macaque. In: *Journal of Neurophysiology* (1981), S. 369–384

[FB97] Fiesler, E. ; Beale, R.: *Handbook of Neural Computation.* Oxford University Press, 1997

[Fuk75] Fukushima, Kunihiko: Cognitron: A Self-organizing Multilayered Neural Network. In: *Biol.Cybernetics* (1975), S. 121–136

[Fuk88] Fukushima, Kunihiko: Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition. In: *Neural Networks* (1988), Vol. I, S. 119–130

[FW91] Fukushima, K. ; Wake, N.: Handwritten Alphanumeric Character Recognition by the Neocognitron. In: *IEEE Trans. Neural Networks* (1991), S. 355–365

[Gib69] Gibson, E. *Principles of Perceptual Learning and Development.* Meredith Corporation. 1969

[Gol97] Golenhofen, K.: *Physiologie Heute.* 2. Urban&Fischer, 1997

[Gru03] Gruebl, Andreas. *Eine FPGA-basierte Plattform für neuronale Netze.* 2003

[Her88] Herken, R.: *The Universal Turing Machine: A Half Century Survey.* Hamburg : Kammerer&Unverzagt, 1988

[HKAI90] Hiraiwa, A. ; Kurosu, S. ; Arisawa, S. ; Inoue, M.: A two level pipeline RISC processor array for ANN. In: *Proc. Int. Joint Conf. on Neural Networks* (1990), S. 137–140

[HN89] Hecht-Nielsen, R. *Reading.* Ma: Addison Wesley. 1989

[Hof93] Hoffmann, Norbert: *Kleines Handbuch Neuronale Netze.* Vieweg, 1993

[Hop82] Hopfield, John: Neural Networks and physical systems with emergent collective computational abilities. In: *Proceedings of the National Academy of Sciences* (1982), Vol. 79, S. 2554–2558

[Hor] Horner, H. *Neuronale Netze*

[HSSM02] Hohmann, S. ; Schemmel, J. ; Schürmann, F. ; Meier, K.: Exploring the Parameter Space of a Genetic Algorithm for Training an Analog Neural Network. In: *Genetic and Evolutionary Computing Conference* (2002)

[HW62] Hubel, D.H. ; Wiesel, T.N.: Receptive fields, binocular interaction and functional architecture in the cats visual cortex. In: *Journal de Physiologie* (1962), S. 106–154

[Int91] Intel, Corp. *80170NX Electrically Trainable Analog Neural Network, Data Sheet.* 1991

[Koh88a] Kohonen, Teuvo: An Introduction to Neural Computing. In: *Neural Networks* (1988), Vol. I, S. 3–16

[Koh88b] Kohonen, Teuvo: *Self-Organization and Associative Memory.* 2. Springer-Verlag, 1988

[Kra96] Kramer, Alan H.: Array-Based Analog Computation. In: *Proc. MicroNeuro* (October 1996), S. 22–29

[LDT95] Lovell, D.R. ; Downs, T. ; Tsoi, A.C.: Is the Neocognitron Capable of State-of-the-art Digit Recognition? In: *IEEE Trans. Neural Networks* (1995), July

[LeC03] LeCunn, Yann. *MNIST.* http://yann.lecun.com/exdb/mnist. 2003

[LT92] Lovell, D.R. ; Tsoi, A.C.: The Performance of the Neocognitron with Various S-Cell and C-Cell Transfer Functions. (1992)

[LT93] Lovell, D.R ; Tsoi, A.C.: Determining Selectivities in the Neocognitron. In: *Proceedings of the World Congress on Neural Networks Vol.3* (1993), S. 695–699

[Mc.95] Mc.Graw, G.E.: *Letter Spirit.* Bloomington, USA, Indiana University, Diplomarbeit, September 1995. – Phd-Thesis, http://-goosie.cogsie.cogsci.indiana.edu/farg/mcgrawg/thesis.html

[Med98] Medler, David A.: A Brief History of Connectionism. In: *Neural Computing Surveys 1* (1998), S. 61–101

[Met92] Metzler, J.B.: *Linder Biologie.* Stuttgart : Verlagsbuchhandlung Metzler, 1992

[MHW94] Masa, P. ; Hoen, K. ; Wallinga, H.: A 20 Input, 20 Nanosecond Pattern Classifier. In: *Proc. Int. Joint Conf. on Neural Networks* (1994), July

[Min54] Minsky, M.L. *Neural nets and the brain-model problem - Phd.* Princeton University Press. 1954

[Neu92] Neurologix, Inc. *NLX420 Data Sheet.* June 1992

[PSAB99] Pan, Z. ; Sabisch, T. ; Adams, R. ; Bolouri, H.: Staged Training of Neocognitron by Evolutionary Algorithms. In: *Proceeding of IEEE Congress on Evolutionary Computation (CEC'99)* (1999), July, S. 1965–1972

[PTVF99] Press, W.H. ; Teukolsky, S.A. ; Vetterling, W.T. ; Flannery, B.P.: *Numerical Recipes in C.* 2. Cambridge University Press, 1999

[Roj96] Rojas, R.: *Theorie der neuronalen Netze.* 4. Springer, 1996

[Ros58] Rosenblatt, F.: The perceptron a probabilistic model for information storage and organization in the brain. In: *Psychychological* (1958), Revue 62, S. 386–408

[SHMS02] Schemmel, J. ; Hohmann, S. ; Meier, K. ; Schürmann, F.: A Mixed-Mode Analog Neural Network using Current-Steering Synapses. In: *Kluwer Academic Publishers, Niederlande* (2002)

[SHSM02] Schürmann, F. ; Hohmann, S. ; Schemmel, J. ; Meier, K. *Towards an Artificial Neural Network Framework.* NASA/DoD Conference on Evolvable Hardware. 2002

[SKI80] Sato, T. ; Kawamura, T. ; Iwai, E.: Responsiveness of inferotemporal single units to visual pattern stimuli in monkeys performing discrimination. In: *Experimental Brain Research* (1980), S. 313–319

[Sou88] Souček, B. und M.: *Neural and Massively Parallel Computers—the Sixth Generation.* New York : Wiley, 1988

[SSHM02] Schemmel, J. ; Schürmann, F. ; Hohmann, S. ; Meier, K.: An Integrated Mixed-Mode Neural Network Architecture for Megasynapse ANNs. In: *World Congress of Computational Intelligence (WCCI20002)* (2002)

[Vos01] Voss, A.: *Das große PC & Internet Lexikon 2001/02.* Data Becker, 2001

[Wol] Wolfram, Stephen. *Mathematica.* http://www.wolfram.com

Anhang A

Abbildungen

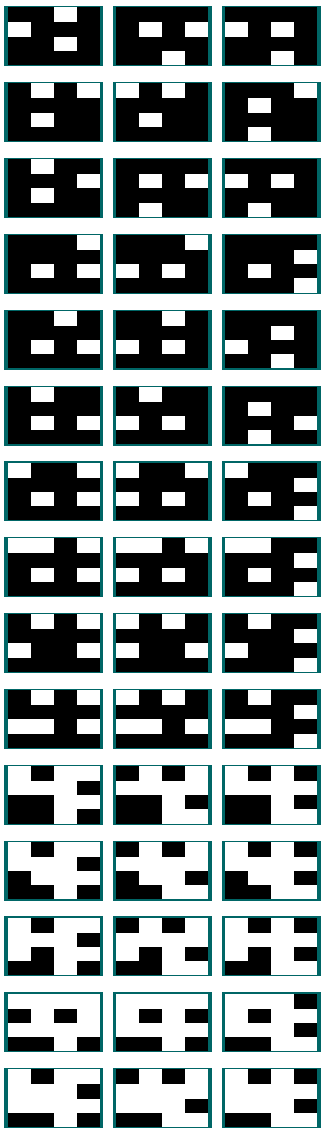

Abbildung A.1: Die zeilenweise gruppierten 4x4 Eingangsbilder eines minimalen Testaufbaus (Abb. 4.1) besitzen ab der Schicht C2 dieselben Aktivitäten und sind somit vom Netz nicht mehr trennbar.

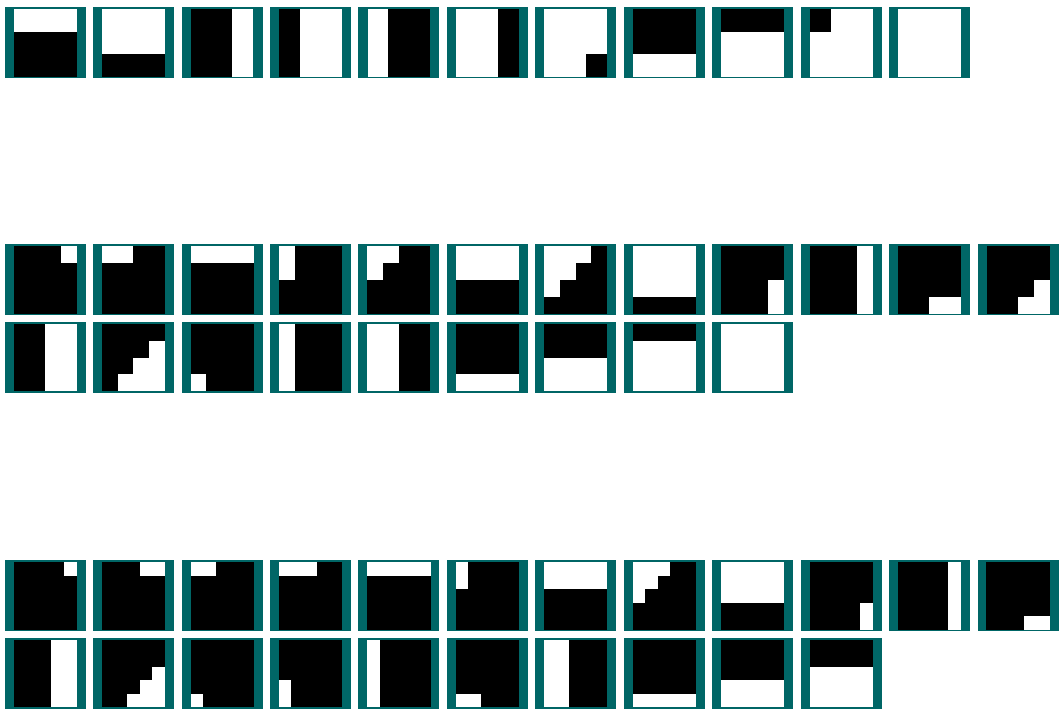

Abbildung A.2: Die vom Verteilungsselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgrößen sind 3x3,4x4, 5x5 Bildpunkte und die Selektorparameter sind: $p_1 = 0,01$; $p_2 = 0,01$; $p_3 = 0,5$. Für die Fokusbereichsgröße 3x3 sind die Selektorparameter analog zu Abbildung 4.4 $p_1 = 0,02$; $p_2 = 0,02$. Der Vergleich mit dieser Abbildung zeigt, daß der Parameter p_3 wenig Einfluss auf die Art der selektierten Features hat. Er beschränkt sie lediglich zusätzlich.

Abbildung A.3: Die vom Differenzselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgröße ist 3x3. Die Selektorparameter sind von oben nach unten: $(p_4 = 0,3\,;\, p_5 = 1\,;\, p_6 = 0,15)$, $(p_4 = 0,3\,;\, p_5 = 3\,;\, p_6 = 0,15)$, $(p_4 = 0,3\,;\, p_5 = 5\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 1\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 3\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 5\,;\, p_6 = 0,15)$, $(p_4 = 0,7\,;\, p_5 = 1\,;\, p_6 = 0,15)$.

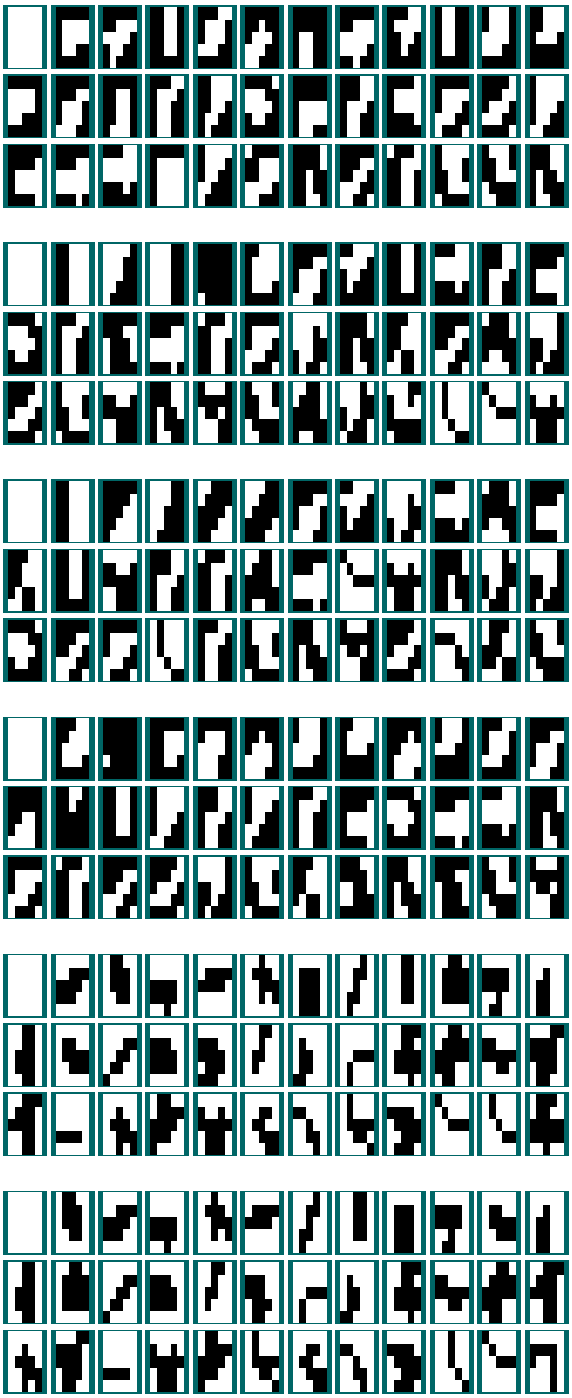

Abbildung A.4: Die vom Differenzselektor getroffene Auswahl für die Features der ersten Stufe. Der Trainingsdatensatz besteht dabei aus den ersten drei Bildern jeder Ziffer innerhalb der MNIST-Datenbank. Die Fokusbereichsgröße ist 5x5. Die Selektorparameter sind von oben nach unten: $(p_4 = 0,3\,;\, p_5 = 1\,;\, p_6 = 0,15)$, $(p_4 = 0,3\,;\, p_5 = 3\,;\, p_6 = 0,15)$, $(p_4 = 0,3\,;\, p_5 = 5\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 1\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 3\,;\, p_6 = 0,15)$, $(p_4 = 0,5\,;\, p_5 = 5\,;\, p_6 = 0,15)$.

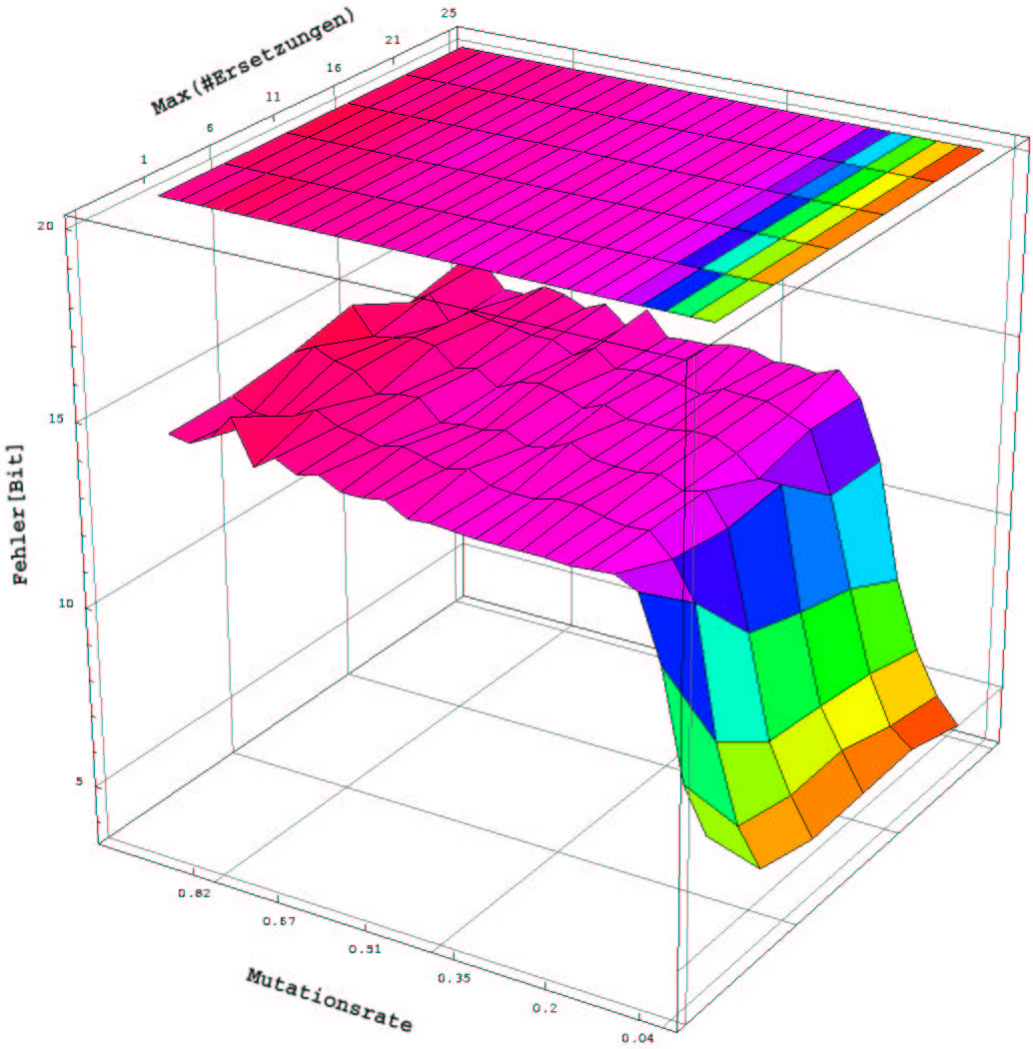

Abbildung A.5: Der Einfluss der Mutationsrate und der maximalen Anzahl Ersetzungen schlechter Individuen durch das aktuell erfolgreichste Individuum (je Generation) auf den SHAGEN-Algorithmus wurde untersucht. Die dafür verwendete Trainingsaufgabe entspricht der einer ersten Neocognitron-Stufe. Der Fehler bezeichnet hierbei die Anzahl invertierter Punkte der erhaltenen Ausgangsmuster gegenüber den Ausgangssollmustern. Die Messpunkte sind gemittelt über fünf Durchläufe, und die Generationsobergrenze ist konstant 5.000. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Gewichtsauflösung=3Bit und eine zusätzliche Gewichtung fehlender gegenüber überflüssiger Aktivität von eins.

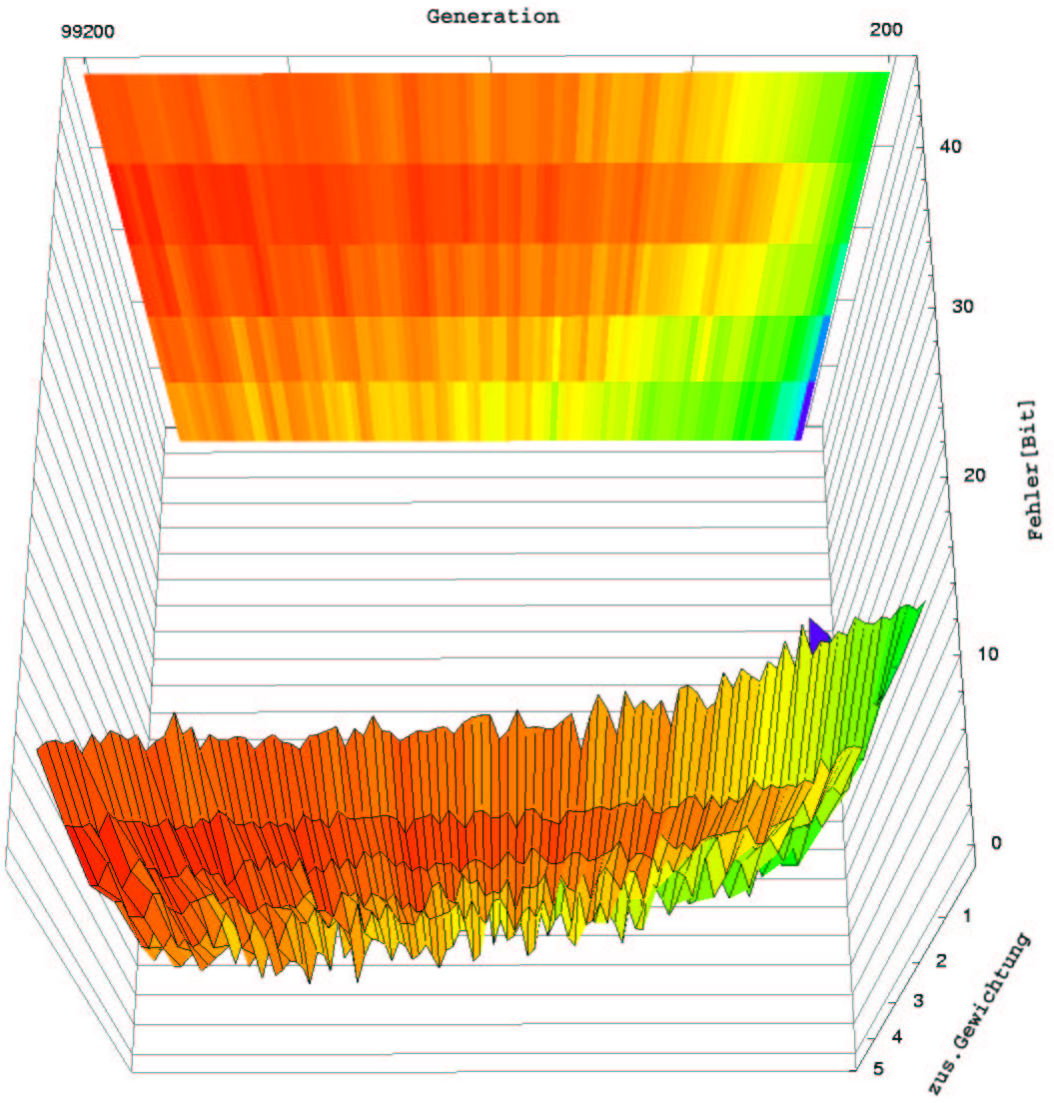

Abbildung A.6: Da die typischen Ausgangssollmuster im Neocognitron lediglich einen aktiven Punkt beinhalten, ist eine Gewichtung fehlender gegenüber überflüssiger Aktivität sinnvoll. Null entspricht dabei keiner besonderen Gewichtung. Ist die zusätzliche Gewichtung fehlender Aktivität hingegen eins, so wird jeder fehlende Punkt im Ausgangsmuster mit einer (gegenüber den überflüssigen Punkten) zusätzlichen Fehlereinheit gewichtet. *Fehler* bezeichnet hier die Abweichung der trainierten Ausgaben von den Sollausgaben. Ein geringer Fehler entspricht einer guten Lernleistung. Die Lernaufgabe entspricht einer typischen ersten Neocognitron-Stufe und die Messwerte sind gemittelt über 10 Durchläufe. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Ersetzungsobergrenze=25, Gewichtsauflösung=7Bit und eine Mutationsrate von $\frac{30}{255}$.

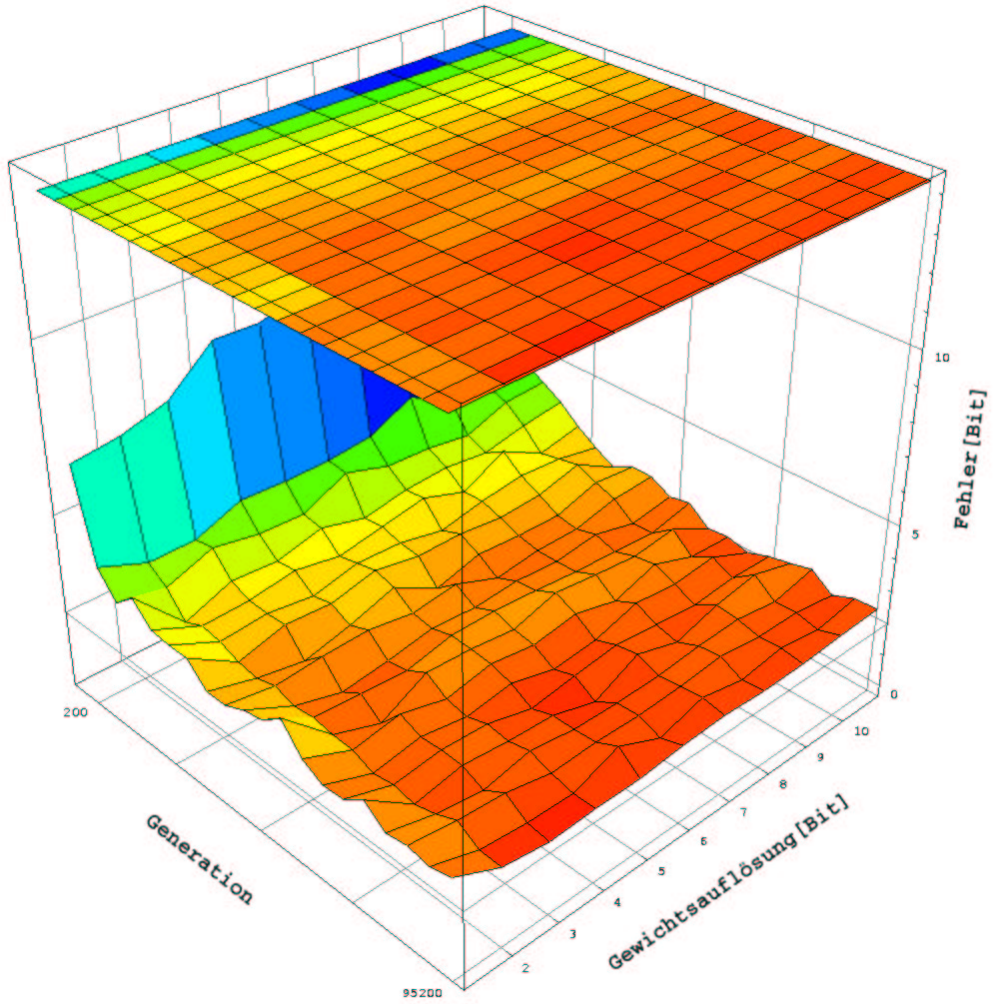

Abbildung A.7: Eine Begrenzung der Bit-Länge der während der Mutationsphase erzeugten Zufallszahlen bewirkt die Herabsetzung der Gewichtsauflösung. Dies schränkt den Suchraum ein - existiert eine Lösung in diesem Raum, wird sie im Mittel vom Trainingsalgorithmus schneller entdeckt als bei voller 11-Bit (HAGEN-) Auflösung. Die Aktivitätssummation erfolgt innerhalb der SHAGEN-Simulation mit 16-Bit, Überläufe sind demnach nicht möglich. Die übrigen Parameter sind: Eingänge=9, Anzahl Rückkopplungen=4, Ausgänge=13, Neuronenschwelle=-1, Populationsgröße=30, Ersetzungsobergrenze=25 und eine Mutationsrate von $\frac{20}{255}$. Die imVergleich zu Abbildung 4.14 herabgesetzte Mutationsrate verbessert den Trainingserfolg für alle Auflösungen deutlich. Unterschiede zwischen den verschiedenen Auflösungen sind deswegen und wegen der höheren Anzahl Generationen nicht so deutlich erkennbar. Offensichtlich ist jedoch, daß eine geringe Gewichtsauflösung nicht von Nachteil ist.

Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-9.5.2003-8:1.54	0 1	1066	10	1	0	0
V2.5-7.5.2003-4:48.8	0 1	2948	10	1	0	0
V2.5-7.5.2003-19:45.9	0 1	1004	10	1	0	0
V2.5-6.5.2003-7:58.39	0 1	1561	10	1	0	0
V2.5-6.5.2003-20:48.39	0 1	263	10	1	0	0
V2.5-10.5.2003-6:59.51	0 1	2275	10	1	0	0
V2.5-10.5.2003-5:12.12	0 1	2553	10	1	0	0
V2.5-10.5.2003-15:51.31	0 1	149	10	1	0	0
V2.5-9.5.2003-4:36.15	0 1	2891	10	0,95	0,05	0
V2.5-8.5.2003-17:19.45	0 1	215	10	0,95	0,05	0
V2.5-7.5.2003-23:35.23	0 1	1872	10	0,95	0	0,05
V2.5-6.5.2003-5:32.44	0 1	2982	10	0,95	0,05	0
V2.5-6.5.2003-2:19.6	0 1	2525	10	0,95	0,05	0
V2.5-11.5.2003-1:40.24	0 1	860	10	0,95	0,05	0
V2.5-10.5.2003-2:25.27	0 1	1421	10	0,95	0	0,05
V2.5-8.5.2003-19:4.38	0 1	629	10	0,9	0,1	0
V2.5-8.5.2003-0:19.15	0 1	1804	10	0,9	0,1	0
V2.5-6.5.2003-9:48.0	0 1	84	10	0,9	0,1	0
V2.5-6.5.2003-21:17.25	0 1	100	10	0,9	0,05	0,05
V2.5-6.5.2003-12:43.29	0 1	1573	10	0,9	0,1	0
V2.5-5.5.2003-23:32.57	0 1	274	10	0,9	0,1	0
V2.5-8.5.2003-16:39.54	0 1	634	10	0,85	0,1	0,05
V2.5-8.5.2003-1:1.0	0 1	595	10	0,85	0,15	0
V2.5-5.5.2003-22:29.32	0 1	2379	10	0,85	0,15	0
V2.5-5.5.2003-20:39.54	0 1	2435	10	0,8	0,2	0
V2.5-6.5.2003-22:48.53	0 1	1757	10	0,75	0,25	0
V2.5-9.5.2003-6:26.40	1 7 6 8	2428	9	0,7222	0,1111	0,1667
V2.5-9.5.2003-2:21.16	1 7 6 8	1737	9	0,7222	0,1667	0,1111
V2.5-10.5.2003-6:29.58	1 7 6 8	350	9	0,7222	0,1944	0,08333
V2.5-9.5.2003-8:36.44	1 7 6 8	773	9	0,6667	0,05556	0,2778
V2.5-9.5.2003-2:20.53	2 3 5 6 8 0	1665	4	0,6667	0,04167	0,2917
V2.5-10.5.2003-5:44.19	1 7 6 8	2563	9	0,6389	0,2778	0,08333
V2.5-9.5.2003-1:16.3	1 7 6 8	2919	9	0,6111	0,1667	0,2222
V2.5-8.5.2003-22:40.2	2 3 5 6 8 0	2658	4	0,5833	0,1667	0,25
V2.5-10.5.2003-8:6.12	1 7 6 8	2392	9	0,5833	0,1944	0,2222
V2.5-8.5.2003-23:37.44	1 7 6 8	1768	9	0,5556	0,2222	0,2222
V2.5-9.5.2003-3:20.4	2 3 5 6 8 0	1590	4	0,5417	0,08333	0,375
V2.5-10.5.2003-7:29.18	1 7 6 8	2488	9	0,4722	0,3611	0,1667
V2.5-10.5.2003-9:6.14	1 7 6 8	2876	9	0,4444	0,4444	0,1111
V2.5-9.5.2003-3:54.32	1 7 6 8	2028	9	0,3889	0,1944	0,4167
V2.5-7.5.2003-23:55.3	0 1	1132	10	0,2	0	0,8
V2.5-9.5.2003-9:37.51	0 1	2882	10	0	0	1
V2.5-9.5.2003-8:42.23	0 1	405	10	0	0	1
V2.5-9.5.2003-7:13.5	0 1	2340	10	0	0	1
V2.5-8.5.2003-6:56.33	0 1	823	10	0	0	1
V2.5-8.5.2003-6:22.12	0 1	5	10	0	0	1
V2.5-8.5.2003-3:7.45	0 1	300	10	0	0	1
V2.5-8.5.2003-22:10.19	0 1	2635	10	0	0	1
V2.5-8.5.2003-21:50.9	0 1	2697	10	0	0	1
V2.5-8.5.2003-20:22.14	0 1	1954	10	0	0	1
V2.5-8.5.2003-19:39.36	0 1	670	10	0	0	1
V2.5-8.5.2003-18:36.16	0 1	2321	10	0	0	1
V2.5-8.5.2003-1:46.43	0 1	1651	10	0	0	1
V2.5-8.5.2003-14:49.40	0 1	2778	10	0	0	1
V2.5-8.5.2003-13:5.25	0 1	1554	10	0	0	1
V2.5-8.5.2003-11:21.4	0 1	2725	10	0	0	1
V2.5-8.5.2003-0:47.54	0 1	1767	10	0	0	1
V2.5-7.5.2003-6:20.23	0 1	2640	10	0	0	1
V2.5-7.5.2003-5:38.53	0 1	720	10	0	0	1
V2.5-7.5.2003-5:10.34	0 1	847	10	0	0	1
V2.5-7.5.2003-3:53.3	0 1	1324	10	0	0	1
V2.5-7.5.2003-2:39.42	0 1	1367	10	0	0	1
V2.5-7.5.2003-22:37.26	0 1	2560	10	0	0	1
V2.5-7.5.2003-22:13.30	0 1	2539	10	0	0	1
V2.5-7.5.2003-21:29.31	0 1	1122	10	0	0	1
V2.5-7.5.2003-20:52.44	0 1	918	10	0	0	1
V2.5-7.5.2003-0:51.24	0 1	390	10	0	0	1
V2.5-7.5.2003-0:13.11	0 1	1744	10	0	0	1
V2.5-6.5.2003-8:39.55	0 1	1117	10	0	0	1
V2.5-6.5.2003-6:58.5	0 1	788	10	0	0	1
V2.5-6.5.2003-4:1.1	0 1	1628	10	0	0	1
V2.5-6.5.2003-3:27.35	0 1	1981	10	0	0	1
V2.5-6.5.2003-21:46.13	0 1	2097	10	0	0	1
V2.5-6.5.2003-19:51.58	0 1	722	10	0	0	1
V2.5-6.5.2003-11:39.45	0 1	478	10	0	0	1
V2.5-5.5.2003-22:59.47	0 1	1400	10	0	0	1
V2.5-11.5.2003-4:28.38	0 1	2787	10	0	0	1
V2.5-11.5.2003-3:31.26	0 1	148	10	0	0	1
V2.5-10.5.2003-9:55.25	0 1	2094	10	0	0	1
V2.5-10.5.2003-6:0.51	0 1	1618	10	0	0	1
V2.5-10.5.2003-3:29.6	0 1	1340	10	0	0	1
V2.5-10.5.2003-16:46.7	0 1	1713	10	0	0	1
V2.5-10.5.2003-1:52.28	0 1	2083	10	0	0	1
V2.5-10.5.2003-14:55.33	0 1	183	10	0	0	1
V2.5-10.5.2003-13:55.10	0 1	1709	10	0	0	1
V2.5-10.5.2003-12:43.26	0 1	2796	10	0	0	1
V2.5-10.5.2003-11:57.16	0 1	2201	10	0	0	1

Tabelle A.1: Überprüfung aller Topologien auf die Erkennung der Trainigsdaten selbst.

Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-10.5.2003-11:57.16	0 1	1	300	0	0	1
V2.5-10.5.2003-0:58.38	0 1	1	300	0	0	1
V2.5-10.5.2003-0:14.19	0 1	1	300	0	0	1
Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-9.5.2003-2:20.53	2 3 5 6 8 0	500	100	0,1117	0,6083	0,28
V2.5-8.5.2003-22:40.2	2 3 5 6 8 0	500	100	0,08833	0,5733	0,3383
V2.5-9.5.2003-3:20.4	2 3 5 6 8 0	500	100	0,05167	0,47	0,4783
Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V1.1-22.4.2003-22:57.50	0 1	500	100	0,61	0,39	0
V1.5-26.4.2003-15:57.13	1 7 6 8	500	100	0,36	0,4275	0,2125
V1.5-25.4.2003-1:10.2	1 7 6 8	500	100	0,355	0,5725	0,0725
V2.5-10.5.2003-9:6.14	1 7 6 8	500	100	0,345	0,565	0,09
V1.5-28.4.2003-4:29.52	1 7 6 8	500	100	0,275	0,515	0,21
V2.5-10.5.2003-5:44.19	1 7 6 8	500	100	0,25	0,6225	0,1275
V2.5-10.5.2003-8:6.12	1 7 6 8	500	100	0,245	0,5575	0,1975
V1.5-27.4.2003-14:34.7	1 7 6 8	500	100	0,2275	0,55	0,2225
V2.5-10.5.2003-6:29.58	1 7 6 8	500	100	0,225	0,605	0,17
V1.5-25.4.2003-16:43.18	1 7 6 8	500	100	0,2	0,5425	0,2575
V2.5-10.5.2003-7:29.18	1 7 6 8	500	100	0,195	0,5075	0,2975
V2.5-8.5.2003-23:37.44	1 7 6 8	500	100	0,14	0,525	0,335
Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-10.5.2003-15:51.31	0 1	500	100	0,83	0,095	0,075
V2.5-10.5.2003-16:46.7	0 1	500	100	0,26	0,22	0,52
V2.5-10.5.2003-1:52.28	0 1	500	100	0	0	1
V2.5-10.5.2003-14:55.33	0 1	500	100	0	0	1
V2.5-10.5.2003-13:55.10	0 1	500	100	0	0	1
V2.5-10.5.2003-12:43.26	0 1	500	100	0	0	1
V2.5-10.5.2003-11:57.16	0 1	500	100	0	0	1
V2.5-10.5.2003-0:58.38	0 1	500	100	0	0	1
V2.5-10.5.2003-0:14.19	0 1	500	100	0	0	1
Dateiname(Teil)	Ziffern	Start	#/Ziffer	Richtig	Falsch	Weder/Noch
V2.5-10.5.2003-15:51.31	0 1	500	100	0,84	0,13	0,03
V2.5-10.5.2003-0:14.19	0 1	500	100	0,58	0,335	0,085
V2.5-10.5.2003-1:52.28	0 1	500	100	0,57	0,265	0,165
V2.5-10.5.2003-0:58.38	0 1	500	100	0,56	0,405	0,035
V2.5-10.5.2003-13:55.10	0 1	500	100	0,525	0,31	0,165
V2.5-10.5.2003-16:46.7	0 1	500	100	0,51	0,42	0,07
V2.5-10.5.2003-14:55.33	0 1	500	100	0,51	0,345	0,145
V2.5-10.5.2003-11:57.16	0 1	500	100	0,42	0,49	0,09
V2.5-10.5.2003-12:43.26	0 1	500	100	0,41	0,355	0,235

Tabelle A.2: Einige der gefundenen Neocognitron-Topologien konnten ausführlich auf ihre Erkennungsleistung getestet werden. Der Dateiname in der ersten Spalte muß um die Kürzel *bonnee* (vorne) und $\cdots$*.syms* (hinten) ergänzt werden. Spalte drei gibt die Anzahl der Testbilder je Ziffer an, im ersten Fall wurde das Individuum also auf die Erkennung von 600 Eingangsbildern überprüft. Die zugehörigen letzten Stufen der Individuen sind zu finden in den Unterverzeichnissen: Evolver16, Evolver13, Evolver14, Evolver16 und StanFromEvolver16 in dieser Reihenfolge. Sie tragen das Kürzel *.vLindy*. Siehe Kapitel B.

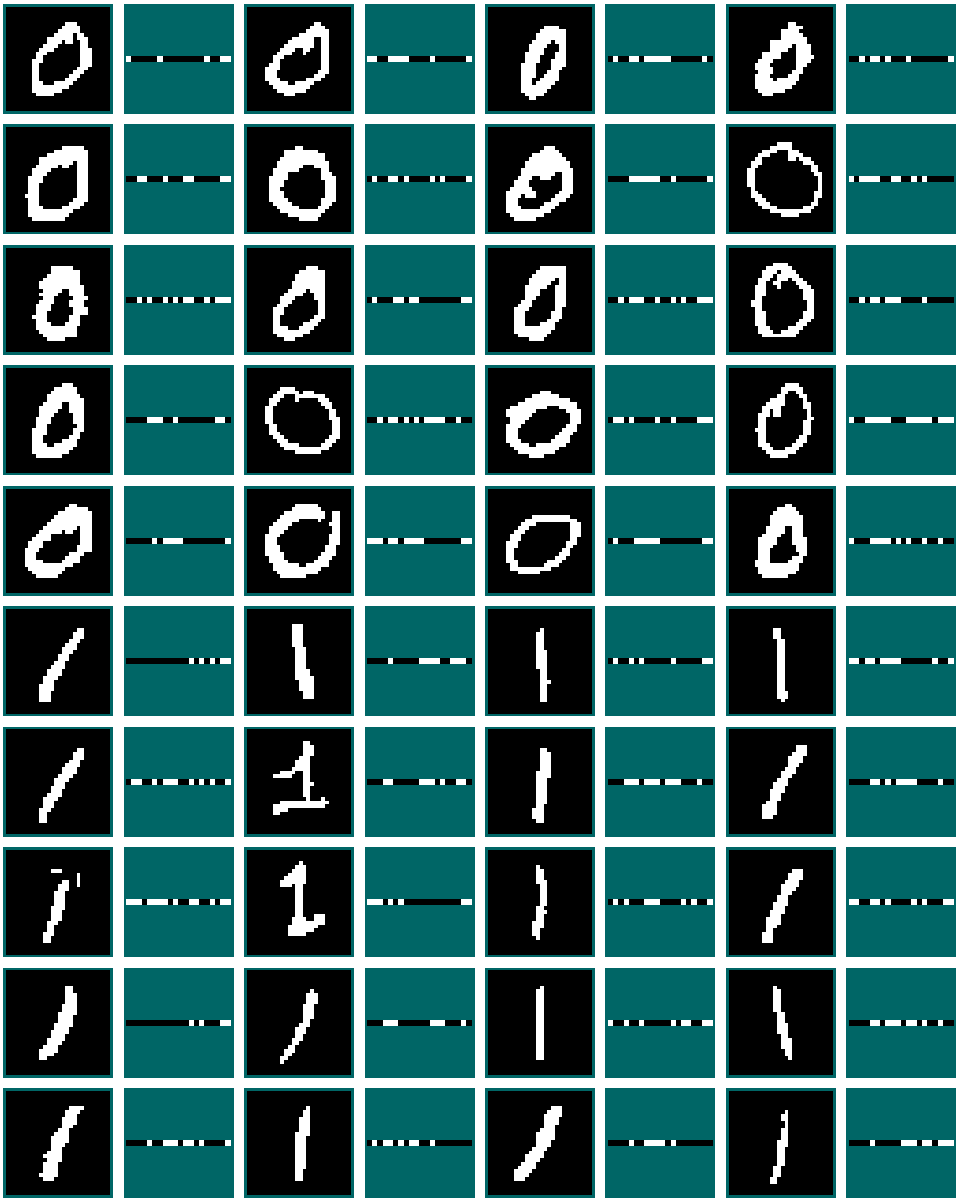

Abbildung A.8: Die ersten 20 Bilder der beiden Ziffern Null und Eins - ausgewertet mit einem mittelmässigen Individuum (AllvLindies/bonneeV2.5-10.5.2003-0:14.19.200351695625.vLindy s.h. letzte(!) Tabelle A.2). Die horizontalen Balken neben den Bildern sind die Ausgaben der L-Schicht. Ihre linke Hälfte steht für die Feature-Gruppe der Nullen wohingegen ihre rechte Hälfte die der Einsen darstellt. Überwiegt die Aktivität in einer der beiden Gruppen, gilt das Bild als entsprechend klassifiziert (Kap. 3.2.3).

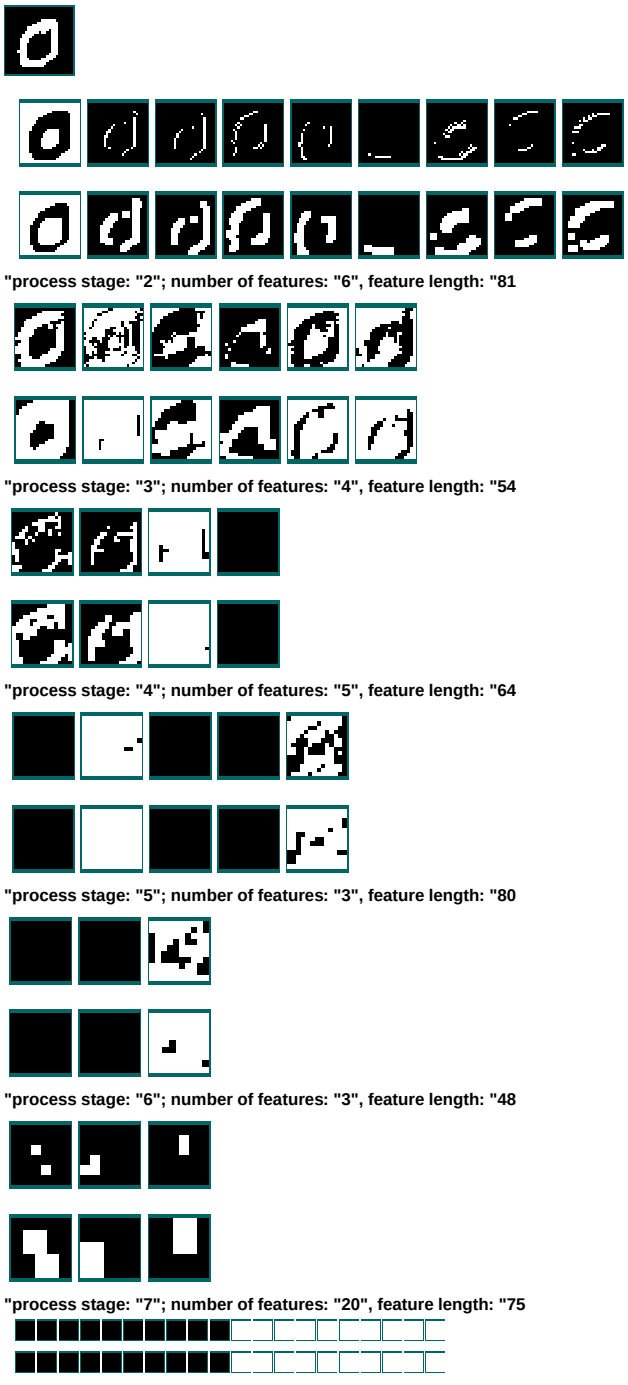

Abbildung A.9: Die Ausgaben aller Stufen des besten Individuums aus Tabelle 5.1. Das Eingangsbild ist eine Null. Die erste Stufe besitzt neun Features. Auf das Eingangsbild folgt die erste S-Schicht. Ihre Aktivität zeigt, an welchen Positionen das einer Ebene zugeordnete Feature entdeckt wurde. Die dritte Zeile zeigt die C2-Schicht und so fort. Streng genommen sind die Ausgaben "process stage" jeweils eine Zeile zu tief - sie beziehen sich lediglich auf die S-Schichten und ihre Position hat historische Gründe. Die Ausgabe der letzten Schicht, der L-Schicht, teilt sich in zwei Hälften. Die ersten zehn Punkte stehen für Features aus der Feature-Gruppe *Null*, die zweiten zehn Punkte sind der Feature-Gruppe *Eins* zugeordnet. Die Hälfte mit größerer Aktivität bestimmt die Klassifikation.

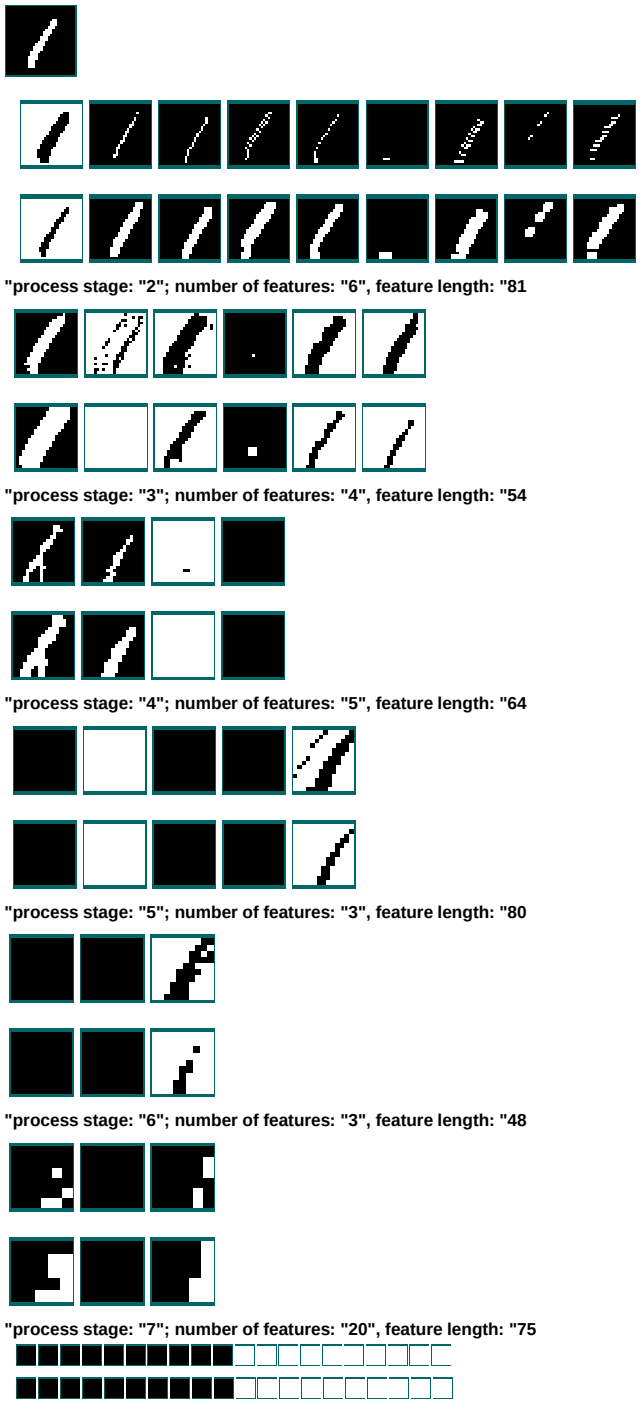

Abbildung A.10: Die Ausgaben aller Stufen des besten Individuums aus Tabelle 5.1. Das Eingangsbild ist eine Eins. Die erste Stufe besitzt neun Features. Auf das Eingangsbild folgt die erste S-Schicht. Ihre Aktivität zeigt, an welchen Positionen das einer Ebene zugeordnete Feature entdeckt wurde. Die dritte Zeile zeigt die C2-Schicht und so fort. Streng genommen sind die Ausgaben "process stage" jeweils eine Zeile zu tief - sie beziehen sich lediglich auf die S-Schichten und ihre Position hat historische Gründe. Die Ausgabe der letzten Schicht, der L-Schicht, teilt sich in zwei Hälften. Die ersten zehn Punkte stehen für Features aus der Feature-Gruppe *Null*, die zweiten zehn Punkte sind der Feature-Gruppe *Eins* zugeordnet. Die Hälfte mit größerer Aktivität bestimmt die Klassifikation.

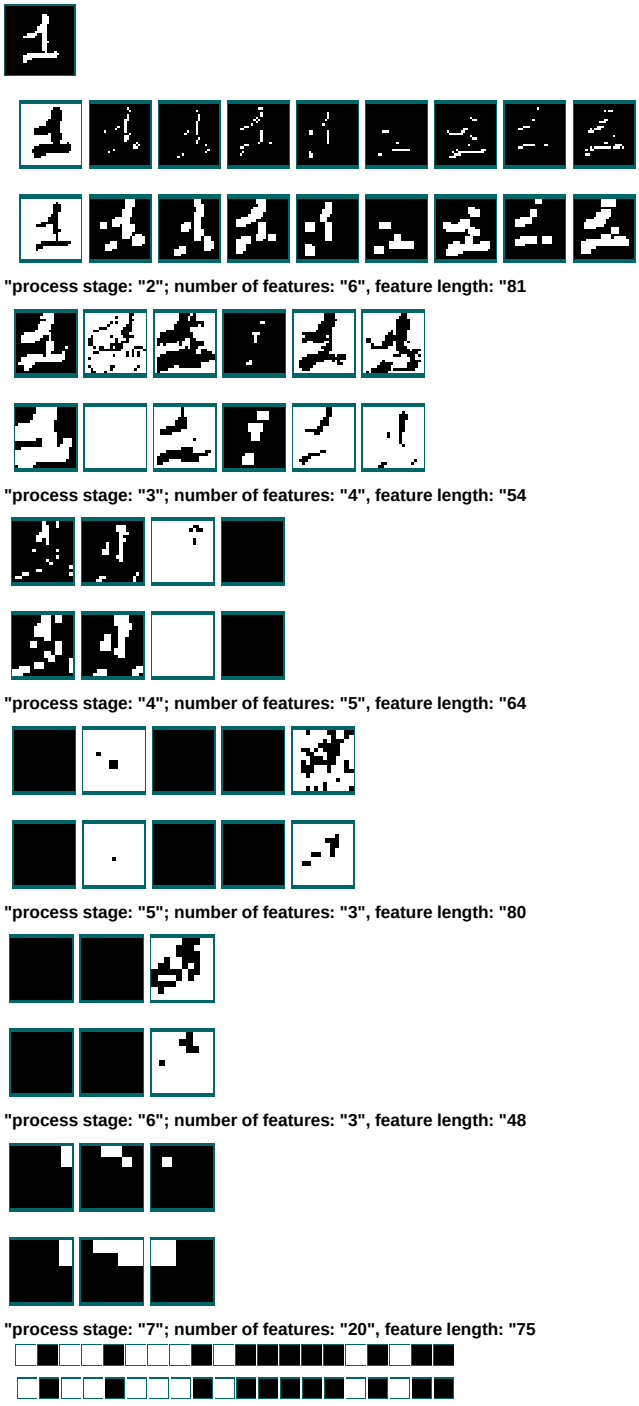

Abbildung A.11: Die Ausgaben aller Stufen des besten Individuums aus Tabelle 5.1. Das Eingangsbild ist eine Eins. Die erste Stufe besitzt neun Features. Auf das Eingangsbild folgt die erste S-Schicht. Ihre Aktivität zeigt, an welchen Positionen das einer Ebene zugeordnete Feature entdeckt wurde. Die dritte Zeile zeigt die C2-Schicht und so fort. Streng genommen sind die Ausgaben "process stage" jeweils eine Zeile zu tief - sie beziehen sich lediglich auf die S-Schichten und ihre Position hat historische Gründe. Die Ausgabe der letzten Schicht, der L-Schicht, teilt sich in zwei Hälften. Die ersten zehn Punkte stehen für Features aus der Feature-Gruppe *Null*, die zweiten zehn Punkte sind der Feature-Gruppe *Eins* zugeordnet. Die Hälfte mit größerer Aktivität bestimmt die Klassifikation.

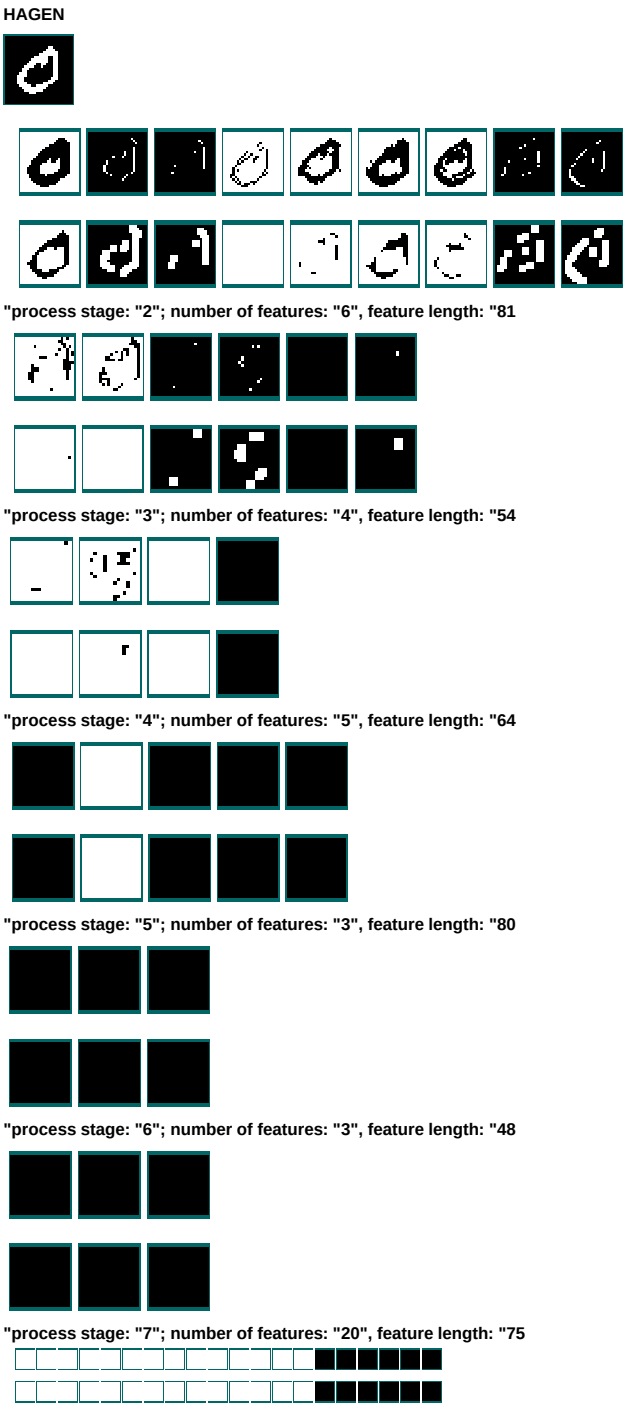

Abbildung A.12: Die Übertragung der Netzgewichte von SHAGEN auf HAGEN und die anschließende Evaluation dieses Individuums mittels HAGEN ist schon möglich. Allerdings ist ein Nachtraining zur Behebung geringfügiger Unterschiede zwischen Simulation und Hardware ($\approx 4\%$) noch nicht implementiert. Die Summierung kleiner Abweichungen über sieben Stufen genügt offenbar, um die Ausgabe der letzten Stufe zu nihilieren.

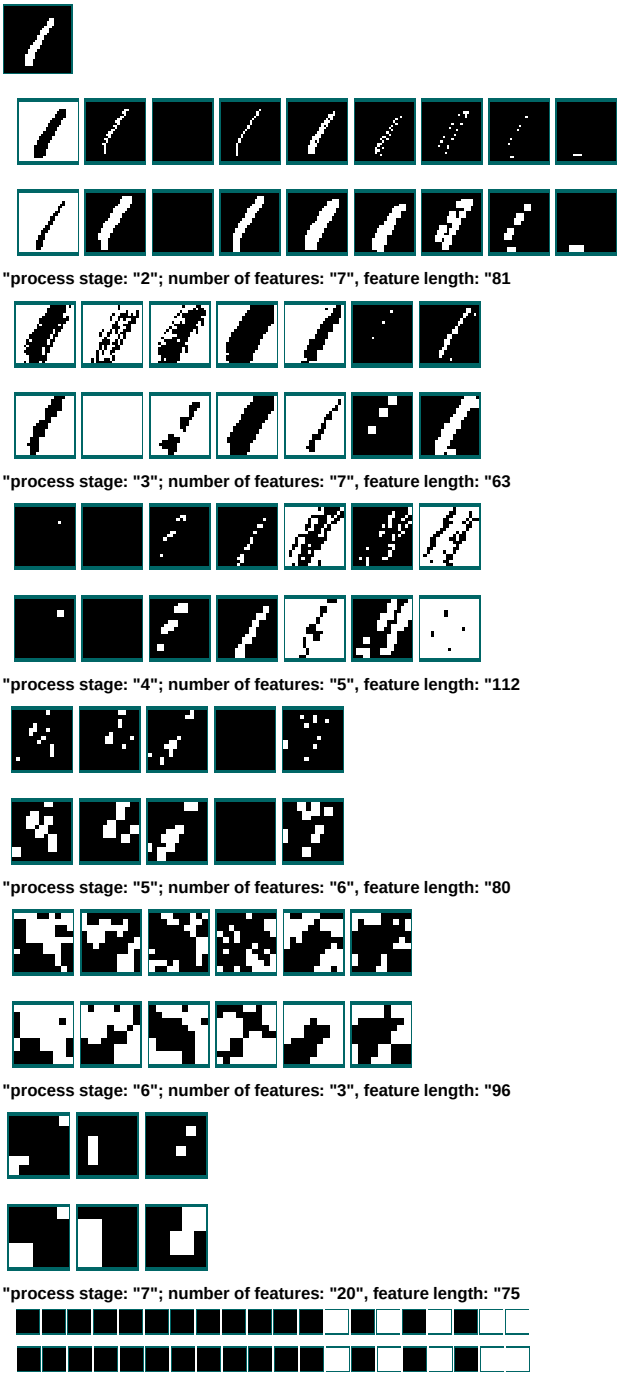

Abbildung A.13: Die Abbildung zeigt die Aktivitäten aller Ebenen des mittelmässigen Individuums *bonneeV2.5-10.5.2003-0:14.19.200351695625.vLindy* (s.h. letzte der Tabellen A.2). Es fällt auf, daß die Aktivität deutlich gleichmässiger über die Ebenen verteilt ist als in Abbildung 5.3,A.9,A.10 oder A.11.

Anhang B

Inhalt der CD

Die CD enthält die entwickelten Programmstrukturen ebenso wie die zugehörigen Daten. In mehr als 70 Verzeichnissen sind über 3100 Dateien zu finden (Abb. B.1). Zwar sind alle diese Dateien mit der vorliegenden Arbeit verknüpft, es muß im Folgenden dennoch eine kurze Beschreibung der wichtigsten Einsprungpunkte in das Projekt genügen. Sie wird ausreichend durch die den Teilprojekten beigestellten Dokumentationen ergänzt. Jedes der Hauptverzeichnisse enthält überdies einen minimalen README.txt-Text. Diesem können die wichtigsten Einsprungpunkte der Arbeit entnommen werden. Ich erlaube mir in diesem Kapitel das in der Gruppe vorhandene Wissen zu den Projekten vorauszusetzen.

MathematicaLibraries Das BONNEE Programm, verkörpert durch das Mathematica-Notebook *Datei bonneeV2.5.nb*, lädt die in diesem Verzeichnis enthaltenen Mathematica-Bibliotheken. Vier Bibliotheken wurden entwickelt, ihre Dateien sind an der Endung *.m* zu erkennen. Zwei zentrale Beispiele der Verwendung der BONNEE-Routinen sind innerhalb *bonneeV2.5.nb* rot hinterlegt. Sie setzen die vorherige Ausführung aller *Programm*-Zellen via *Selektion&Shift-Return* voraus. Nahezu alle Mathematica-Funktionen sind sowohl mit einer Kurzbeschreibung, wie auch mit einem Beispiel versehen. Die Kurzbeschreibung kann entweder dem Funktionsdefinitionsblock selbst entnommen oder auch über die Mathematica-übliche Hilfefunktion *?...* abgerufen werden. Der Befehl *Install[softHagen$Path]* verbindet die Mathematica-Seite mit dem automatisch initiierten SHAGEN-Paket. Dieses befindet sich in dem Unterverzeichnis *softHagen* und heißt *sHagen*. *sHagen* kann auch manuell aufgerufen und verlinkt werden. Der Benutzer wird dann nach seinem Start nach der gewünschten port-Nummer gefragt. Die verwendeten Auszüge aus der *MNIST*-Datenbank sind in *train-images-sset* enthalten. Die ihnen zugeordneten Ziffern können *train-labels-idx1-ubyte* entnommen werden - Details finden sich in der entsprechenden Bibliothek (*NIST-lib-V1.5*).

Arbeit Das Verzeichnis *Arbeit* enthält die *.tex*-Dateien dieser Diplomarbeit. Überdies finden sich in dem Unterverzeichnis *Pictures* die Abbildungen sowohl als

Abbildung B.1: Verzeichnisstruktur der beigefügten CD.

.eps-Dateien, wie auch deren Ursprünge wieder. Im Verzeichniss *Arbeit/Pictures/MiniNBs* sind beispielsweise die Mathematica-Notebooks gespeichert, die zur Erzeugung einiger Abbildungen entwickelt wurden.

CentralPark Alle im Rahmen der Arbeit kreierten Neocognitron-Individuen wurden in *CentralPark/AllIndies* untergebracht. Ihre Dateinamen enden auf *.syms*. Innerhalb eines Mathematica-Notebooks werden sie eingelesen mittels *Get[...]*. Die einzulesenden Variablennamen müssen zuvor in eine Liste mit dem Namen *loadSymsL* eingetragen werden. Gleiches gilt für die ihnen zugehörigen letzten Stufen (L-Schichten). Sie befinden sich im Unterverzeichnis *AllvLindies*, und enden mit *.vLindy*. Die Auswertungsschritte sind in verschiedene Unterverzeichnisse aufgespalten, deren Namensgebung selbsterklärend ist.

Hannee-evolver10 Dieses Verzeichnis beherbergt die zur Ansteuerung des HAGEN-Chips notwendigen Programmstrukturen und MathLink-Schnittstellen. Die Bedeutung ihrer Programme sind im HANNEE-Projekt dokumentiert. Zugriff auf diese Dokumentation, auf deren Einführung ich ganz besonders stolz bin :), erhält man beispielsweise via *konqueror Hannee-evolver10/project/hannee/hannee-api/index.html&* . Soll HANNEE aus einem Mathematica-Notebook ferngesteuert werden, so wird der HANNEE-Aufruf durch das Skript *Hannee-evolver10/project/hannee/source/callThisSourcedBeforeMathematicasInstallIsCalled* erledigt. Dieses initiiert neben dem HANNEE-Aufruf eine kleine GUI zur Verbindung HANNEEs mit der MathLink-Schnittstelle. Die Verbindung wird mittels drücken des Knopfes *connect* etabliert. Die port-Nummer kann anschließend aus der ASCII-Datei *portForTheHagenMathematicaConnection* im selben Verzeichnis ausgelesen werden. Sie wird für die Mathematica-seitige Verbindung benötigt.

Scripts Das Skript *diffa* automatisiert die Versions-Kontrolle. Die Unterschiede zweier Verzeichnisse können hiermit automatisch in *tkdiff* geladen werden. Dieses Hilfmittel ist auf die *kdevelop*-Umgebung zugeschnitten und ignoriert beispielsweise alle *.moc*-Dateien. Ist das zeitgenaue *Loggen* spezieller Prozesse gewünscht, hilft das Skript *topper*.

Anhang C

Fachbegriffe

Folgende Fachbegriffe werden innerhalb der Arbeit eingeführt. Diese Stichwortliste soll einige Begriffe, die für Verwirrung sorgen können, leichter nachschlagbar machen. Sie ist als Erinnerung gedacht und erhebt keinen Anspruch auf Vollständigkeit.

Bit Stelle im Dualsystem. Ein Bit kann die Werte 0 oder 1 annehmen.

Byte 8-Bits.

Feature Ein Feature ist eine Eigenschaft eines Bildes in Form eines markanten Bildausschnittes.

Fokusbereich Ein Fokusbereich ist die Begrenzung eines Bildausschnittes. Er wird durch seine Abmessungen und Position innerhalb des Bildes bzw. der Ebene vollständig bestimmt. Die Neocognitron-Zellen erhalten ihre Eingabe aus Fokusbereichen der Vorgängerschicht.

Generalisierung Neuronale Netze können beispielsweise auch Muster, die nicht im Trainingssatz enthalten waren, korrekt klassifizieren. Sie generalisieren.

Generation Die Selektion evolutionärer Algorithmen wird zeitdiskret vorgenommen. Ein Selektions-Zyklus wird Generation genannt.

Hyper Die Neocognitron-C-Schichten bestehen i.a. aus mehreren Ebenen. Ein Fokusbereich der folgenden S-Schicht setzt sich aus den "Unter"-Fokusbereichen gleicher Position innerhalb aller dieser Ebenen zusammen. Zur Betonung dieser Tatsache findet die Silbe *Hyper* manchmal Verwendung. Da Features die Inhalte entsprechender Fokusbereiche darstellen, ist der Begriff Hyper-Feature eng mit dem Begriff Hyper-Fokusbereich verknüpft.

Individuum Meint im Rahmen der Arbeit entweder eine vollständige Gewichtskonstellation des SHAGEN bzw. HAGEN-Netzes oder auch alle Informationen, die zur Festlegung eines Neocognitrons notwendig sind.

Interfaceklassen Begriff objektorientierter Programmiersprachen. Sie bilden die Schnittstelle zum Benutzer bzw. Programmierer und verbergen die internen Details eines Objektes.

Population Eine Menge Individuen, die zur selben Zeit in einem Konkurrenzverhältnis existieren.

Protokolloverhead Bei der Datenübertragung werden neben den Daten auch Steuerinformationen wie beispielsweise Absender, Zieladressen oder Prüfsummen transferiert. Stehen für diese keine zusätzlichen Steuerleitungen zur Verfügung, erhöhen sie die effektive Datenmenge.

Topologie Die Art und Weise der Vernetzung aller Zellen eines Netzwerkes. Diese hängt im Falle des Neocognitrons allerdings von allen übrigen Parametern ab: die Anzahl der Features einer Stufe bestimmt beispielsweise die Anzahl der Ebenen derselben und die Fokusbereichsgröße nachfolgender S-Schicht mit. Die Begriffe Netztopologie, Netzstruktur und vollständiger Netzparametersatz lassen sich nicht trennen und werden in der Arbeit i.a. ähnlich gebraucht.

Danksagung

Ich danke:

- Herrn Prof. Dr. Karlheinz Meier für die unkomplizierte Betreuung und die angenehme Atmosphäre in den Montagsmeetings.
- Herrn Prof. Dr. Heinz Horner für die freundliche Übernahme der Zweitkorrektur.
- Dr. Johannes Schemmel für die visionäre Idee und heisse Diskussionen.
- Felix Schürmann für die stete Hilfsbereitschaft bei der Beantwortung dummer Fragen.
- Meinem Anstandswowow Stevo Hoeman fürs Wow, die aufopfernde Unterstützung und die netten MensaClub-Pausen.
- Hannee Fieres für objektorientiertes Denken und Sprachgewalt.
- Mirko Sedlacek und Alex Sinsel für einige Gemeinsamkeiten.
- Tillmann Schmitz für die Teilnahme an Rechenspielen.
- Anne-Catherine ehemals Schuch für die Verschönerung der Gruppe und die Erkenntnis, daß wir alle einmal heiraten.
- Brandy Grümmel für die erfrischende Normalität und das partiell gemeinsame Interesse an frischer Luft.
- Stefan Philipp für den Inbegriff der Bastelei.
- Andreas Breidenassel für das Gefühl, zu unmöglichen Tageszeiten doch nicht der einzige Mensch im Institut zu sein.
- Thorsten Maucher für die Ablenkung zwischendurch.
- Jörg Langeheine fürs Gehör und die wiederholte Bereitschaft zur Rettung mißglückter CD-Fackelei.
- Martin Trefzer für die leider nie zustandegekommene Ziegelhäuser Party.
- Dr. Robert Weis für die unerschütterliche Hilfsbereitschaft.
- Prof. Dr B. Sakmann, Klaus Bauer und Arnd Roth für die schöne Zeit am MPIMF.
- Das MPI für Kernphysik und meinen Daddy für die reichliche Überlassung von Knete.
- Dibbe,Hannee,Schorsch, Eva, meiner Mama und Zuzana fürs Korrekturlesen.
- Kathy und dem Drachenkaiser für die allwochenendliche Erheiterung morgens um sechs in der Frühe.
- Radio Sunshine, die mir des öfteren den Takt vorgaben.
- Allen die ich mag für die bloße Anwesenheit und den ganzen Rest.

und ganz besonders meiner besseren Hälfte Bon, ohne die diese Arbeit sicher nicht zustandegekommen wäre.

Erklärung:

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 26.05.2003

.....
(Unterschrift)