

Hochschule für Technik und Wirtschaft (FH)  
Fachbereich Informatik/Mathematik

Dokumentation zum Beleg

# Gesichtserkennung

im Fach Neuroinformationsverarbeitung

Bearbeitende Studenten : André Kempe  
Frank Kanschur

Betreuender Professor : Prof. Dr. rer. nat. habil. H. Iwe

7. Juli 2005

## **Kurzfassung**

Gesichter in Bildern sind unabkömmlich für moderne, intelligente Mensch-Maschine-Interfaces. Dies betrifft sowohl Fragen der Kommunikation (Gesichts-Ausdruck, Lippenverfolgung) als auch sicherheitsrelevante Anwendungen (Gesichtserkennung). Gegenwärtig geht ein Großteil der Forschung in Richtung Gesichtserkennung, Gesichtsverfolgung, Ausdruckerkennung und Lageabschätzung.

Diese Ansätze gehen bis jetzt zumeist davon aus, daß in dem Bild ein Gesicht vorhanden ist. Für ein Human-Computer-Interface ist jedoch ein robustes und effizientes Verfahren zur Gesichtsfindung notwendig - in einem gegebenen Bild soll es möglich sein, alle Gesichtsregionen unabhängig von ihrer räumlichen Lage, Drehung und Beleuchtungsbedingungen zu erkennen. Für diese Aufgabe wurden zahlreiche Ansätze und Methoden entwickelt und sind in diverssem Umfang im Internet auffindbar.

Ziel dieser Arbeit ist es, einen eigenen Ansatz zu entwickeln, der ein Bild in ein Gesicht oder Nicht-Gesicht klassifizieren kann. Diese Klassifizierung soll mit Hilfe eines Neuronalen Netzes verwirklicht werden, bei der auch die Bildvorverarbeitung entspr. Beachtung findet.

# Inhaltsverzeichnis

<b>1</b>	<b>Theoretische Betrachtungen</b>	<b>2</b>
1.1	Neuronale Netze . . . . .	3
1.2	Bildbearbeitung und Verbesserung . . . . .	11
<b>2</b>	<b>Projekt-Realisierung</b>	<b>15</b>
2.1	Vorbetrachtung . . . . .	16
2.2	Bildvorverarbeitung . . . . .	16
2.3	Der Netzwerk-Typ . . . . .	17
2.4	Das Update-Verfahren . . . . .	17
2.5	Parallelisierung . . . . .	18
<b>3</b>	<b>Installation und Nutzung</b>	<b>19</b>
3.1	Erforderliche Entwicklerwerkzeuge . . . . .	20
3.2	Übersetzung der erstellten Programme . . . . .	20
3.3	Die Nutzung der Programme . . . . .	22
3.4	Die Setting-Datei . . . . .	27
3.5	Ein Beispiel-Durchlauf . . . . .	28
<b>4</b>	<b>Gewonnene Erfahrungen</b>	<b>30</b>
4.1	Bildmaterial . . . . .	31
4.2	Bildvorverarbeitung . . . . .	31
4.3	Lern-Parameter . . . . .	33
4.4	Lernfortschritt und Ergebnisse . . . . .	33
4.5	Fortführende Aspekte . . . . .	35
<b>5</b>	<b>Entwicklungs-Internia</b>	<b>36</b>
5.1	Bibliotheken . . . . .	37
5.2	Entwicklungsumgebung und Compiler . . . . .	38
5.3	Klassenstruktur . . . . .	38
<b>A</b>	<b>Nutzungsrecht und -erklärung</b>	<b>41</b>

# Abbildungsverzeichnis

1.1	Sigmoiden-Funktion . . . . .	5
1.2	Eulersche Glockenkurve . . . . .	5
1.3	Das Back-Propagation-Verfahren . . . . .	9
3.1	Panel der Bildvorverarbeitung . . . . .	23
3.2	Der Matrix-Viewer . . . . .	24
3.3	Das Settings-Panel . . . . .	24
3.4	Parameter des Lernprogramms . . . . .	25
3.5	Die Oberfläche des Error-Viewers . . . . .	25
3.6	Die Oberfläche des grafischen Face-Checkers . . . . .	26
3.7	Parameter des Programms fchecker . . . . .	26
4.1	Praktikable Transformer-Einstellungen . . . . .	32
4.2	Praktikable Netzwerk-Einstellungen . . . . .	33
4.3	Ergebnisse bzgl. verschiedener Bild-Vorverarbeitungen . . . . .	34
5.1	Klassenstruktur des Neuronalen Netzes . . . . .	38
5.2	Übersicht der Hilfsklassen . . . . .	40

# Kapitel 1

## Theoretische Betrachtungen

### Inhaltsangabe

---

<b>1.1 Neuronale Netze . . . . .</b>	<b>3</b>
1.1.1 Aktivierungsfunktionen . . . . .	4
1.1.2 Diverse Netzwerkformen . . . . .	6
1.1.3 Das Lernen eines Netzes . . . . .	7
<b>1.2 Bildbearbeitung und Verbesserung . . . . .</b>	<b>11</b>
1.2.1 Rauschentfernung (Denoise) . . . . .	11
1.2.2 Kantenglättung (Smooth) . . . . .	11
1.2.3 Kontraststärkung . . . . .	12
1.2.4 Kantenfindung . . . . .	12
1.2.5 Mittelwert-Bilder . . . . .	13
1.2.6 Fast-Fourier-Transformation . . . . .	14
1.2.7 Weiteres . . . . .	14

---

In diesem Kapitel sollen zunächst einige theoretische Aspekte Neuronaler Netze sowie der Bildbearbeitung und -verbesserung behandelt werden. Da bzgl. dieser Thematik zahlreiche Literatur existiert, soll hier nicht allzu tiefgreifend darauf eingegangen werden.

## 1.1 Neuronale Netze

Ziel der Neuronalen Netze ist es, mittels Annäherung der Funktionsweise des menschlichen Gehirns eine flexible, auf algorithmischem Weg nicht formulierbare Problemlösung zu erreichen.

Ausgangspunkt ist eine mathematische Formulierung eines Neurons. Dieses besitzt eine Reihe von Eingangswerten, die es unterschiedlich stark berücksichtigt, um aus ihnen seinen eigenen Ausgangswert zu ermitteln. Aus diesem Netto-Ausgangswert berechnet sich durch Anwendung der sogenannten Aktivierungsfunktion der Ausgabewert des Neurons. Dieser Sachverhalt lässt sich mit Vektoroperationen wie folgt ausdrücken:

$$\boxed{n = \vec{o} \times \vec{w}^T} \quad (1.1)$$

$$\boxed{o = F_{act}(n)} \quad (1.2)$$

$\vec{o}$  ... Eingabe-Vektor  
 $\vec{w}$  ... Gewichts-Vektor  
 $F_{act}$  ... Aktivierungs-Funktion

Neuronale Netze entstehen durch die Gruppierung, d.h. das Neben- und Hintereinanderschalten von Neuronen. Abhängig von gewählter Form und vom Aufbau des Netzwerkes (siehe Seite 6) lässt sich dieser Aufbau mathematisch in anderer Form aufbereiten. Im Falle eines Feed-Forward-Netzes z.B., in dem die Neuronen in verbundenen Schichten angeordnet sind, lassen sich die Gewichts-Vektoren einer Neuronen-Schicht als Gewichtsmatrix  $\bar{W}$  angeben.

$$\bar{W} = (o_{j1}^{\vec{}} , \dots , o_{jn}^{\vec{}}) \quad (1.3)$$

$o_n^{\vec{}}$  ... Gewichtsvektor des  $n$ -ten Neurons der  $j$ -ten Schicht

Dadurch werden die Gleichungen 1.1 und 1.2 in die folgenden Formen überführt:

$$\vec{n} = \vec{o} \times \bar{W}^T \quad (1.4)$$

$$\vec{o} = F_{act}(\vec{n}) \quad (1.5)$$

$\vec{o}$  ... Eingabe-Vektor  
 $\bar{W}$  ... Gewichts-Matrix  
 $F_{act}$  ... Aktivierungs-Funktion

Als Resultat ergibt sich eine Reduktion des neuronalen Netzes auf eine Reihe von Gewichtsmatrizen, die sich mathematisch und rechentechnisch sehr effizient verarbeiten lassen.

### 1.1.1 Aktivierungsfunktionen

Die Wahl der Aktivierungsfunktion (auch Transferfunktion genannt) hat einen immensen Einfluss auf die Fähigkeiten des neuronalen Netzes, da sie letztendlich die Klassifizierung der Eingabe-Muster vornimmt. Historisch hat sich die Binäre Funktion als erste Aktivierungsfunktion entwickelt:

$$f_b(x) = \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x \leq 0 \end{cases} \quad (1.6)$$

Diese Funktion teilt die Eingabemuster an einer Grenze in zwei Klassen. Dieses Verfahren lässt sich durch Verschieben der Grenzen an die gegebene Problemstellung anpassen. Das Problem einer Binären Aktivierungsfunktion besteht jedoch darin, dass sie sich bei  $x = 0$  nicht differenzieren lässt, für ein erfolgreiches Lernverfahren jedoch unerlässlich ist. Aus diesem Grund geht man zu einer der Binären Funktion ähnlichen Aktivierungsfunktion über, der im folgenden erklärten Sigmoiden-Funktion.

#### 1.1.1.1 Sigmoiden-Funktion

Die Sigmoiden-Funktion ist eine klassische Aktivierungs-Funktion und findet in der Praxis zahlreiche Anwendung. In der in Gleichung 1.7 gezeigten Form lässt sie sich in Abszissen-Richtung verschieben und stauchen, ihre Ableitung ist in Gleichung 1.8 dargestellt.

$$f_s(x) = a * \frac{1}{1 + e^{-b*x}} + c \quad (1.7)$$

$$f'_s(x) = \frac{a * b * e^{-a*x}}{(1 + e^{-a*x})^2} \quad (1.8)$$

- $a$  ... Stauchung in Abszissen-Richtung
- $b$  ... Stauchung in Ordinaten-Richtung  $\rightarrow$  Anstieg
- $c$  ... Verschiebung in Abszissen-Richtung

Für  $b \rightarrow \infty$  geht die Sigmoiden-Funktion, beispielhaft dargestellt in Abbildung 1.1 auf der nächsten Seite, in die bereits erwähnte Binäre Funktion über.

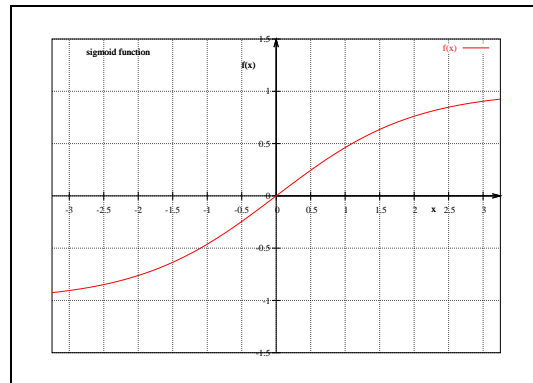


Abbildung 1.1: Sigmoiden-Funktion

### 1.1.1.2 Eulersche Glockenkurve

Neben der Sigmoiden-Funktion können auch andere Aktivierungs-Funktionen wie bspw. die Eulersche Glockenkurve zum Einsatz kommen. Sie ist durch Gleichung 1.9, ihre Ableitung durch Gleichung 1.10 beschrieben:

$$f_e(x) = \frac{1}{e^{a*x^2}} \quad (1.9)$$

$$f'_e(x) = 2 \frac{ax}{e^{a*x^2}} \quad (1.10)$$

Eine Eulersche Glockenkurve ist in Abbildung 1.2 beispielhaft dargestellt.

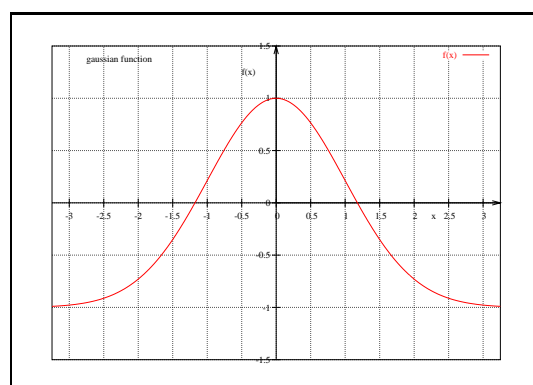


Abbildung 1.2: Eulersche Glockenkurve

Je nach Anwendungsfall lassen sich auch andere Aktivierungs-Funktionen einsetzen, an dieser Stelle soll jedoch nicht näher darauf eingegangen werden.

### 1.1.2 Diverse Netzwerkformen

Durch die geschickte Wahl der Netzwerkform kann bereits entscheidend über die Fähigkeiten eines neuronalen Netzes entschieden werden. Aufgrund der Vielfalt vorhandener Literatur wird hier jedoch nicht weiter tiefgründig auf diese Thematik eingegangen, sondern ausgewählte Formen nur kurz vorgestellt.

#### 1.1.2.1 Feed-Forward-Netzwerke

Das Feed-Forward-Netzwerk zählt zu den bekanntesten Netzwerkformen und zeichnet sich durch einen streng schichtenartigen Aufbau der Neuronen aus. Die Neuronen einer Schicht sind nur mit den Neuronen der direkt nachfolgenden Schicht verbunden. Die Neuronen einer gegebenen Schicht erzeugen einen Ausgabevektor  $\vec{o}_k$ . Die Gewichte der Neuronen einer Schicht können zu einer Gewichtsmatrix  $\vec{W}_j$  zusammengefasst werden. Die Multiplikation  $\vec{W}_j \times \vec{o}_k$  ergibt den Netto-Ausgabevektor  $o_{j \rightarrow net}$ , der durch die Aktivierungsfunktion  $F_{act}(o_{j \rightarrow net})$  in den Ausgabevektor  $\vec{o}_j$  überführt wird.

Das Feed-Forward ist ein General-Purpose-Netzwerk, das sich für fast alle Aufgabenbereiche einsetzen lässt. Es stellt dadurch aber auch höhere Ansprüche an das Training als andere Netzwerkformen.

#### 1.1.2.2 Kohonen-Netzwerke / Self-Organizing Maps

Selbstorganisierende Netzwerke bzw. Maps finden vorwiegend Anwendung, wenn gezielt zwischen diversen Mustern unterschieden werden soll. Ein typisches Beispiel stellt die Klassifizierung von Zeichen wie Ziffern und Buchstaben dar. Für jedes Muster bildet sich auf der Karte/Map eine bestimmte Region, die dieses Muster repräsentiert. Selbst bei abweichenden Mustern, wenn bspw. ein Buchstabe als verunreinigtes/gestörtes Muster anliegt, kann das Kohonen-Netz diese Muster bis zu einem gewissen Grad noch eindeutig zuordnen.

#### 1.1.2.3 Weitere Netzwerkformen

Neben den bereits genannten Netzwerkformen existieren für diverse Anwendungsfälle auch Formen wie Hopfield-Netze, Boltzmann-Maschine, Cascade Correlation Netze, Counterpropagation Netze, Probabilistische Neuronale Netze, Radiale Basisfunktions Netze, ART (kNN), Bidirektionaler Assoziativspeicher, Neocognitron oder PCNN (Pulsodierte Neuronale Netze), auf die hier jedoch nicht weiter eingegangen werden und stattdessen auf [ 8 ] verwiesen soll.

### 1.1.3 Das Lernen eines Netzes

Ziel des Lernprozesses ist es, eine Kombination von Gewichten zu finden, bei denen ein bestimmtes Abbruchskriterium erfüllt wird. Es existiert bisher keine bekannte direkte Berechnungsvorschrift für diese Gewichtekombination. Aus diesem Grund wird daher nachwievor auf ein iteratives Verfahren zur schrittweisen Annäherung der Gewichte zurückgegriffen.

Die Initialisierung dieses Lernprozesses erfolgt bei allen Verfahren durch die zufällige Verteilung der Gewichte auf die aktiven Verbindungen der einzelnen Neuronen, d.h. auf die Gewichtsmatrix. Die Geschwindigkeit des Lernverfahrens hängt neben anderen Faktoren sehr stark von dieser zufälligen Initialisierung ab, im ungünstigsten Fall erfolgt trotz vermeintlich guter Modellierung des Netzes kein Lernen und das Netz muss erneut zufällig initialisiert werden. Derzeit lässt sich diese Vorgehensweise durch keine andere Methode ersetzen.

#### 1.1.3.1 Lernen eines Feed-Forward-Netzes

In einem Feed-Forward-Netzwerk berechnet jedes Neuron eine gewichtete Summe seiner Inputs nach Gleichung 1.11:

$$a_j = \sum_i^N w_{ji} z_i \quad (1.11)$$

wobei  $z_i$  der Input,  $w_{ji}$  das diesem Input zugeordnete Gewicht und  $N$  alle Neuronen mit Verbindung zu diesem Neuron darstellen. Diese Summe wird durch eine nichtlineare Aktivierungs-Funktion  $g_{act}(\cdot)$  in den Output-Wert dieses Neurons nach Gleichung 1.12 überführt:

$$z_j = g_{act}(a_j) \quad (1.12)$$

Die zentrale Vorgehensweise des Lernprozesses besteht in der Minimierung einer Fehlerfunktion über eine Testmenge  $E$  mit  $N$  Elementen. Dabei wird davon ausgegangen, dass es sich bei  $E^n$  um eine differenzierbare Funktion des Netzwerks-Outputs handelt.

$$E = \sum_n^N E^n \quad (1.13)$$

$$E^n = E^n(y_1, \dots, y_c) \quad (1.14)$$

Ziel des Verfahrens ist die Bewertung der Ableitung der Fehlerfunktion  $E^n$  zu den Gewichten des Netzwerks. Da der Fehler von der Aktivierung  $a_j$  und diese von den Gewichten  $w_{ji}$  abhängt, gilt

$$\frac{\delta E^n}{\delta w_{ji}} = \frac{\delta E^n}{\delta a_j} \frac{\delta a_j}{\delta w_{ji}} \quad (1.15)$$

$$\frac{\delta a_j}{\delta w_{ji}} = z_i \quad (1.16)$$

Gleichung 1.16 kann direkt aus Gleichung 1.11 hergeleitet werden. Wird nun

$$\delta_j \equiv \frac{\delta E^n}{\delta a_j} \quad (1.17)$$

gesetzt, so folgt aus den Gleichungen 1.16 und 1.17 durch Einsetzen in Gleichung 1.15

$$\frac{\delta E^n}{\delta w_{ji}} = \delta_j z_i \quad (1.18)$$

$z_i$  stellt hierbei den Input in dieses Gewicht dar,  $\delta_j$  muß berechnet werden.

Für die Ausgabe-Neuronen ist dies recht einfach. Aus Gleichung 1.17 folgt

$$\delta_k \equiv \frac{\delta E^n}{\delta a_k} = g'(a_k) \frac{\delta E^n}{\delta y_k} \quad (1.19)$$

so dass sich aus Gleichung 1.19  $\delta_k$  für die Ausgabe-Neuronen berechnen lässt.

Um  $\delta$  für versteckte Einheiten zu berechnen, wird erneut die Kettenregel angewendet:

$$\delta_j \equiv \frac{\delta E^n}{\delta a_j} = \sum_k \frac{\delta E^n}{\delta a_k} \frac{\delta a_k}{\delta a_j} \quad (1.20)$$

wobei sich die Summe über alle dem Neuron  $j$  direkt nachgeschalteten Einheiten ergibt. Wird nunmehr Gleichung 1.16 in Gleichung 1.20 gesetzt und werden zusätzlich die Gleichungen 1.11 und 1.12 angewendet, so ergibt sich die folgende Formel

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \quad (1.21)$$

Da die  $\delta$ s der Ausgangs-Neuronen bereits in Gleichung 1.19 ermittelt werden, lassen sich

durch die rekursive Anwendung von Gleichung 1.21 die  $\delta$ s für das gesamte Netzwerk berechnen.

Die Ermittlung der Ableitung des Fehlers nach dem Output  $E^n$  lässt sich somit in die folgenden wesentlichen Schritte unterteilen:

1. Propagieren eines Musters  $\vec{x}^n$  durch das Netzwerk, um die Netto-Outputs jedes Neurons zu ermitteln (Gleichungen 1.11 und 1.12)
2. Ermittlung der  $\delta$ s für die Ausgabe-Neuronen (Gleichung 1.19)
3. rückwärts gerichtete Propagierung der  $\delta$ s durch das Netzwerk, um für jedes versteckte Neuron  $\delta_j$  zu berechnen (Gleichung 1.21)
4. Berechnung der erforderlichen Ableitungen (Gleichung 1.18)

Als Fehlerfunktion  $E^n$  wird oftmals die in Gleichung 1.22 dargestellte Standard-Fehlerfunktion genutzt. Abbildung 1.3 zeigt noch einmal grob wesentliche Bestandteile des Backpropagation-Verfahrens.

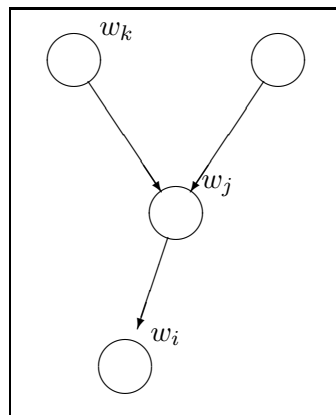


Abbildung 1.3: Das Back-Propagation-Verfahren

Für weitere Informationen sei an dieser Stelle auf die Skripte der Vorlesung “Neuroinformatik” bzw. einschlägige Literatur verwiesen.

### 1.1.3.2 Abbruchkriterien

Als Abbruchkriterien wird meist der absolute Fehler einer Testmenge bei einer gegebenen Gewichtskombination herangezogen. Dazu dient bspw. die Gleichung 1.22 auf der nächsten Seite dargestellte Standard-Fehlerfunktion.

$$E = \frac{1}{2} \sum_{i=0}^M \sum_{j=0}^N (t_{ij} - o_{ij})^2 \quad (1.22)$$

$M$  ... Größe der Testmenge  
 $N$  ... Anzahl der Elemente des Ausgabe-Vektors

Hierbei ist darauf zu achten, dass sich Test- und Lernmenge nicht überschneiden, d.h. es gilt  $T \not\subseteq L$  mit  $T \dots$  Lernmenge und  $L \dots$  Kontrollmenge. Durch diese eingeführte Trennung lässt sich erreichen, dass das Netz kein reiner Akzeptor für die angelegten Muster wird, sondern letztendlich generalisieren kann, also auch neue Muster gleicher Struktur korrekt erkennt.

Der absolute Fehler kann durch einen gewichteten Fehler ersetzt werden. Dabei ist jedoch darauf zu achten, dass eventuell trotz fehlerhaft erkannter Datensätze erfolgreich abgebrochen wird. Als Abbruchkriterium lässt sich aber auch der Fehlerabstieg nutzen, um einen Stillstand des Lernverfahrens zu erkennen.

### 1.1.3.3 Weitere Maßnahmen

Neben den bereits benannten Vorgehensweisen kann das Lernverfahren auch durch Maßnahmen wie der Berücksichtigung des Momentums oder mittels einer Überwachung des Fehlerverlaufs im Allgemeinen beeinflusst werden. Das Momentum stellt den Fehlerabstieg während des Lernens dar und kann bei der Aktualisierung der Gewichte zusätzlich zu den Berechnungen hinzugezogen werden.

Darüber hinaus können weitere Detektoren eine Fehlerverlauf-Stagnation oder einen (unbeabsichtigten) Fehleranstieg erkennen und, z.B. durch geringfügige zufällige Modifikationen der Gewichte, gegensteuern.

## 1.2 Bildbearbeitung und Verbesserung

Im Regelfall ist für Bilder, die als Input für ein neuronales Netz dienen sollen, eine Stufe der Bildvorverarbeitung aus mehreren Gründen erforderlich. Die für Bilder repräsentativen Pixelwerte entsprechen nahezu nie dem für ein Netz optimalen Wertebereich, der wiederum von der gewählten Aktivierungsfunktion abhängig ist. Ausserdem sind die im Bild enthaltenen Informationen für gewöhnlich zu komplex, um von einem Neuronalen Netz erfolgreich erlernt zu werden. Dazu gesellt sich der Fakt, dass im Bild Daten vorhanden sind, die mit der eigentlichen Information keinerlei Korrelation haben und daher unnötige Informationen darstellen und bzgl. des Lernens eher hinderlich denn hilfreich sind.

Die Reduktion unnötiger Informationen erfolgt durch diverse Filter. Gängige und genutzte Verfahren werden in den nun folgenden Abschnitten etwas näher beschrieben.

### 1.2.1 Rauschentfernung (Denoise)

Rauschen stellt ein grosses Problem für neuronale Netze dar, da es statistisch unabhängig von den eigentlichen Bildinformationen ist. Es wird zwischen zwei verschiedenen Arten von Rauschen unterschieden:

- Weißes Rauschen ist statistisch gleichverteilt und daher "grau"
- Braunes Rauschen ist statistisch nicht gleichverteilt und hat damit eine dominante "Färbung"

Weißes Rauschen lässt sich bei digitalen Bildern vornehmlich in den niedrigen Bits der zu einem Pixel gehörenden Farbinformationen finden (siehe [ 7 ]). Ein Verfahren der Ent-rauschung nun besteht darin, diese niederen Bits, die keine relevanten Farbinformationen enthalten, zu löschen. Das Problem dieses Verfahrens besteht jedoch darin, dass natürlich auch "echte" Informationen gelöscht werden und sich aus der Umgebung des Pixels nicht restaurieren lassen.

### 1.2.2 Kantenglättung (Smooth)

Ein anderes Verfahren zur Rausch-Eliminierung besteht in einer lokalen Mittelwertbildung. Dabei wird ein Pixel durch den Mittelwert der Pixel seiner unmittelbaren Umgebung bestimmt und ersetzt. Für eine Mittelwertbildung über die benachbarten Pixel gilt hierbei:

$$g'(x, y) = \frac{1}{9} * \sum_{u=x-1}^{x+1} \sum_{v=y-1}^{v < y+1} g(u, v) \quad (1.23)$$

Dabei kann der herangezogene Bereich je nach Bedarf vergrößert werden. Die Ränder des Bildes können dabei entweder direkt übernommen werden, oder das Ausgangsbild wird an den Rändern zyklisch gedoppelt. Das Bild verliert jedoch je nach Stärke der Filterung an Schärfe und ggf. Informationen.

### 1.2.3 Kontrastspreizung

Für das neuronale Netz stellen ähnliche Informationen mit unterschiedlicher Bedeutung ein beträchtliches Problem dar, da es diese nicht ausreichend unterscheiden kann. Durch die Spreizung der vorhandenen Informationen über einen größeren Wertebereich lassen sich Merkmale besser herausarbeiten. Im Bild entspricht dieses Vorgehen einer Kontrastspreizung, so dass als Ergebnis ein Bild vorliegt, dessen minimaler Grauwert bei 0 ( $\hat{=}$  schwarz) und dessen maximaler Grauwert bei 255 ( $\hat{=}$  weiß) liegt.

Dem Vorteil dieses Verfahrens steht jedoch der Nachteil gegenüber, dass die Daten ohne Berücksichtigung ihrer Bedeutung gespreizt werden, d.h. auch eventuell vorhandenes Rauschen wird verstärkt. Daher sollte das Bild wenn nötig im Vorherein mit anderen qualitätsverbessernden Verfahren bearbeitet werden.

### 1.2.4 Kantenfindung

#### 1.2.4.1 Sobel- und Laplace-Operatoren

In bestimmten Situationen ist es von Vorteil, die in einem Bild vorhandenen Kanten zu extrahieren, z.B. um die Konturen eines Objektes herauszuarbeiten. Die Kanten-Extraktion erfolgt durch die Anwendung sogenannter Sobel-Operatoren auf die einzelnen Pixel des Bildes. Für die Faltung in horizontaler und vertikaler Richtung gibt es eigene Operatoren:

$$H_x = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad H_y = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \boxed{H_{lap} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}}$$

### 1.2.4.2 Canny-Operator

Der Canny-Algorithmus dient ebenfalls der Kantenfindung und vereint die Findung der Kanten in beide Richtungen. Im vorliegenden Fall lassen sich zusätzlich Schwellwerte (Farbwerte) bestimmen, die über eine Beachtung bzw. Nicht-Beachtung im Algorithmus entscheiden und eine gezielte Ausblendung bestimmter Farbwerte bei der Kantenfindung erlauben.

### 1.2.5 Mittelwert-Bilder

Wie unter 1.1.1 bereits beschrieben, hängt die Codierung der Eingangsdaten stark von der gewählten Aktivierungsfunktion und dem Soll-Output des Neuronalen Netzes ab. So ist der Wertebereich eines Pixels in einem Grauwert-Bild als Eingang für das Netzwerk bedingt durch den großen Wertebereich gänzlich ungeeignet. Auch durch die Stauchung des Grauwert-Bereiches 0...255 auf den Bereich -1...1 werden dem Netzwerk keine ausreichenden Möglichkeiten einer Unterscheidung geboten, da die Werte nunmehr zu dicht beieinander liegen.

Abhilfe schafft ein Mittelwert-Bild  $M'$ , das gemeinsam vorkommende Eigenschaften eines jeden Bildes einer Menge aus  $n$  Bildern vereint:

$$g'(x, y) = \frac{1}{n} * \sum_{u=0}^n g(x, y) \quad (1.24)$$

Durch die Bildung der Differenz eines Bildes zu diesem Mittelwert-Bild ergeben sich für das resultierende Bild zwei Eigenschaften:

- Die Grauwerte des Bildes liegen nunmehr im Intervall -128 ... 128. Dies ist das maximale Intervall, das resultierende Intervall ist in Abhängigkeit von den gegebenen Bildern jedoch kleiner
- Nur die Information sind enthalten, die in anderen Bildern nicht gefunden werden können, d.h. auch die Varianz der Werte wird kleiner

Das Resultat dieser Bearbeitung ist jedoch kein Bild mehr im herkömmlichen Sinne. Die weitere Behandlung muss daher rein mathematisch erfolgen. Für Bilder, die nur aus einem Farbwert bestehen, ergibt sich jedoch u.U. ein unbeabsichtigtes Resultat, dass der Berechnung eher hinderlich denn hilfreich ist. Die Mittelwert-Bildung kann daher nur als ein Teil einer Verarbeitungskette eingesetzt werden.

### 1.2.6 Fast-Fourier-Transformation

Durch die Möglichkeit, die in einem Bild enthaltenen Informationen in einen Frequenzraum mittels 2-dimensionaler Fast-Fourier-Transformationen zu überführen, erweitert sich das Vorverarbeitungs-Spektrum. Die Fast-Fourier-Transformation wird durch Gleichung 1.25 beschrieben:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^M \sum_{y=0}^N f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (1.25)$$

Das Resultat dieser Transformation kann analog zum Mittelwert-Bild ebenfalls nur noch rein mathematisch betrachtet werden und stellt kein Bild mehr im herkömmlichen Sinne dar. Zusammen mit der Fast-Fourier-Analyse eines Mittelwert-Bildes erweitert sich jedoch das Bearbeitungs-Spektrum, Merkmale können noch besser herausgearbeitet und extrahiert werden.

### 1.2.7 Weiteres

Für die Bildbearbeitung gibt es zahlreiche weitere Operatoren, die an dieser Stelle jedoch nicht weiter betrachtet werden sollen. Im Zusammenhang der vorliegenden Arbeit wurden jedoch noch einige Hilfs-Operatoren erarbeitet, die bzgl. der Bildmanipulation eingesetzt werden können. Hierzu zählen z.B. ein Verrauscher (Noiser), der ein Bild zusätzlich mit weißem Rauschen in unterschiedlichem Maße versehen kann, oder ein Randomizer, der vorhandene Bildwerte durch gänzlich zufällige Farbwerte ersetzt.

# Kapitel 2

## Projekt-Realisierung

### Inhaltsangabe

---

2.1	Vorbetrachtung . . . . .	16
2.2	Bildvorverarbeitung . . . . .	16
2.3	Der Netzwerk-Typ . . . . .	17
2.4	Das Update-Verfahren . . . . .	17
2.5	Parallelisierung . . . . .	18

---

Im vorigen Kapitel wurden die theoretischen Grundlagen eines Neuronalen Netzes kurz dargelegt. In diesem Kapitel soll nunmehr die konkrete Realisierung beschrieben werden.

## 2.1 Vorbetrachtung

Das Training eines Neuronalen Netzes kann nur anhand von Beispiel-Daten erfolgen. Im vorliegenden Projekt werden diese Daten durch Datensatz-Archive repräsentiert, die i.A. aus mehreren Bild-Matrizen bzw. Datensätzen besteht. Ein Datensatz ist gleichbedeutend mit einem einzelnen, vorverarbeiteten Bild. Die Bearbeitung der gestellten Aufgabe erfolgt in folgenden drei Schritten:

1. Bild-Vorverarbeitung, Erstellung von Datensätzen
2. Lernen des Neuronalen Netzes anhand der erstellten Datensätze
3. Nutzung des Neuronalen Netzes zum Prüfen diverser Bilder

Für diese einzelnen Schritte stehen entweder GUI-basierte oder rein kommandozeilen-orientierte Programme bereit. Die Bildvorverarbeitung erfolgt mit Hilfe eines Programms mit Graphical UserInterface (GUI), da die Vielzahl der Parameter und Einstellmöglichkeiten den Nutzer an der Kommandozeile überfordern würden.

Das Lernen erfolgt rein kommandozeilen-orientiert, da hier keine weitere Interaktion mit dem Nutzer erforderlich ist. Für die Fehlerverfolgung steht jedoch ein kleines Programm bereit, das während des Trainings den Fehlerverlauf grafisch anzeigen kann.

Für die Nutzung des gelernten Netzwerkes stehen neben einer kommandozeilen-orientierten auch eine GUI-Version bereit. Im Gegensatz zur GUI-Variante können mit Hilfe der Kommandozeilen-Version auch Datensätze überprüft werden.

## 2.2 Bildvorverarbeitung

Die erforderliche Bildvorverarbeitung wird im unter Abschnitt 3.3.1 beschriebenen Programm *DSCC* durchgeführt. In Abschnitt 1.2 auf Seite 11 wurden bereits mehrere empfehlenswerte Methoden zur Bildverarbeitung beschrieben. Davon wurden die folgenden im *DSCC* realisiert:

- Rauschentfernung, Abschnitt 1.2.1
- Kantenglättung, Abschnitt 1.2.2
- Kontrastspaltung, Abschnitt 1.2.3
- Kantennfindung durch Faltung, Abschnitt 1.2.4.1

- Mittelwert-Bilder, Abschnitt 1.2.5
- 2-Dimensionale FFT, Abschnitt 1.2.6

Um die Vorverarbeitung möglichst flexibel zu gestalten, können die hier genannten *Transformer* genannten Bildmanipulationen in sogenannten Transformerketten hintereinander angeordnet werden. Diese Transformerketten werden dann sukzessive abgearbeitet. Das Programm *DSCC* erlaubt die einfache Konfiguration der einzelnen Transformer und der gesamten Kette. Aufgrund der Eigenheiten der beiden Transformatoren “Mittelwert-Bild” und “2-Dimensionale Fast-Fourier-Transformation” liegen die Bilder am Ende der Bearbeitung nicht mehr als Bilder im herkömmlichen Sinne, sondern als Matrizen vor.

## 2.3 Der Netzwerk-Typ

In der Frage des Netzwerk-Typs wurde sich für die Verwendung eines speziellen Feed-Forward-Netzes entschieden. Die vorhandenen Datensätze werden in mehrere Teilbilder segmentiert bzw. gesplittet. Diese Segmente werden dann an unabhängige Teilnetze angelegt. Am Ende bewertet ein Entscheidungsnetzwerk die Teilergebnisse. Diese Herangehensweise hat folgende Vorteile:

- Die Teilnetzwerke haben weniger Freiheitsgrade als ein einzelnes Netzwerk, der Lernvorgang wird hierdurch beschleunigt
- Die Teilnetze haben kleinere Teilbereiche zu bearbeiten und können sich somit besser auf eine jeweilige Region eines Gesichtes spezialisieren
- Die Entscheidung über das Gesicht wird erst in der letzten, dafür spezialisierten Stufe getroffen
- Das klassische Feed-Forward-Netz ist als Spezialfall in diesem Typ enthalten, auch wenn nur ein Entscheidungsnetzwerk existieren sollte

Weitere Netzwerk-Typen sind nicht implementiert.

## 2.4 Das Update-Verfahren

Zum Gewichts-Update kommt ein klassisches Error-Backpropagation-Verfahren mit Momentum zum Einsatz. Zusätzlich wurde eine Fehlerüberwachung realisiert, mit deren Hilfe eine Fehlerverlauf-Stagnation oder ein langfristiges Ansteigen des Fehlers verhindert werden soll. Sofern erforderlich, werden die Gewichte zusätzlich um einen zufälligen Anteil aktualisiert, um den Netz die Möglichkeit zu geben, erfolgreich weiterlernen zu können.

## 2.5 Parallelisierung

Zur Beschleunigung des Lernprozesses bieten sich zwei generelle Ansätze an. Eine Möglichkeit besteht in der Optimierung des Verfahrens selbst, indem bspw. die Freiheitsgrade des Netzes beschränkt oder die Aktivierungs- und Ableitungsfunktionen modifiziert werden.

Ein weiterer Ansatz versucht, die Rechenarbeit in einem Rechnernetz auf mehrere Prozessoren zu verteilen. Die ursprünglich vorgesehenen Möglichkeit zur Parallelisierung durch eine funktionale oder datenorientierte Zerlegung der Problemstellung haben sich jedoch als nicht praktikabel erwiesen und sind deshalb aus dem Projekt entfernt worden. Dabei offenbarte sich die Verbindung der Rechner als Problemstelle, welche den erforderlichen Datendurchsatz nicht erbringen konnte. Zur Verdeutlichung der Problematik soll das nachfolgend erläuterte Beispiel dienen.

In einem voll verbundenen Netzwerk mit 1000 Neuronen sind bereits  $1000^2 = 10^6$  Verbindungen nötig. Bei Verwendung von Gleitkomma-Zahlen mit doppelter Genauigkeit sind pro Verbindung 8 Byte erforderlich, für die gesamte Konnektionsmatrix fallen somit bereits insgesamt ca. 8-10 MByte an Daten an, deren Übertragung in einem 100MBit-Netzwerk wie dem der HTW ungefähr eine Sekunde in Anspruch nimmt. Bei einer datenorientierten Parallelisierung ergibt sich daraus folgendes Szenario:

- Ein Server sendet an  $N$  Rechner (bspw. 20) die Konnektionsmatrix und die zu bearbeitenden Datensätze, also insgesamt  $20 \times 10\text{MByte} = 200\text{MByte}$ , was in etwa 20 Sekunden in Anspruch nimmt. Bis die Daten eintreffen, liegt für die Rechner keine Arbeit an, der letzte Rechner vergeudet also 20 Sekunden Rechenzeit, ehe er mit seiner Arbeit beginnen kann.
- Das gleiche Bild ergibt sich beim Senden der Gewichtsänderungsmatrizen an den Server. In Abhängigkeit von der Rechengeschwindigkeit der Clients müssen entweder maximal 20 Konnektionsmatrizen gleichzeitig empfangen werden ( $\rightarrow$  Arbeitsspeicher und Netzwerk-Durchsatz), oder die 20 Clients müssen ihre Daten sukzessive an den Server senden. Im Fall der sukzessiven Übertragung muss nunmehr der erste Client auf den letzten warten, da der Server keine neue Konnektionsmatrix berechnen kann. Im Fall der parallelen Übertragung werden die Clients extrem ausgebremst, da die 200MByte auf einmal anfallen.

Für die funktionsorientierte Zerlegung sind die Probleme ähnlich gelagert, nur dass hier der Kommunikations- und Synchronisations-Overhead noch vermehrt wird, da die einzelnen Schichten ihre Ergebnisse untereinander austauschen und lokal Daten vorhalten müssen.

# Kapitel 3

## Installation und Nutzung

### Inhaltsangabe

---

<b>3.1</b>	<b>Erforderliche Entwicklerwerkzeuge . . . . .</b>	<b>20</b>
3.1.1	Bibliotheken . . . . .	20
3.1.2	Compiler . . . . .	20
<b>3.2</b>	<b>Übersetzung der erstellten Programme . . . . .</b>	<b>20</b>
3.2.1	Makefile . . . . .	21
3.2.2	make.inc . . . . .	21
3.2.3	Relevante Verzeichnisse . . . . .	21
<b>3.3</b>	<b>Die Nutzung der Programme . . . . .</b>	<b>22</b>
3.3.1	Bildvorverarbeitung mittels DSCC . . . . .	22
3.3.2	Lernen des Neuronalen Netzes mittels Datensätzen . . . . .	25
3.3.3	Der Face-Checker . . . . .	26
<b>3.4</b>	<b>Die Setting-Datei . . . . .</b>	<b>27</b>
<b>3.5</b>	<b>Ein Beispiel-Durchlauf . . . . .</b>	<b>28</b>
3.5.1	Erzeugung der Datensätze . . . . .	28
3.5.2	Konfiguration des Neuronalen Netzes . . . . .	29
3.5.3	Das Lernen . . . . .	29
3.5.4	Überprüfung des Ergebnisses . . . . .	29

---

Dieses Kapitel widmet sich der praktischen Anwendung der erstellten Programme. Erforderliche Bibliotheken und Entwicklerwerkzeuge für eine erfolgreiche Übersetzung werden benannt, anschließend erfolgt eine Übersicht notwendiger Schritte, um ein Neurnales Netz trainieren und anschließend nutzen zu können.

## 3.1 Erforderliche Entwicklerwerkzeuge

Die Installation erforderlicher Bibliotheken und Compiler ist in den jeweils zugehörigen Dokumentationen enthalten und soll an dieser Stelle nicht behandelt werden.

### 3.1.1 Bibliotheken

Um die enthaltenen Programme erfolgreich installieren zu können, sind folgende Bibliotheken erforderlich:

<b>MTL</b>	<b>Matrix Template Library</b>	<i><a href="http://www.osl.iu.edu/research/mtl/">http://www.osl.iu.edu/research/mtl/</a></i>
<b>BOOST</b>	<b>BOOST-Bibliotheken</b>	<i><a href="http://www.boost.org">http://www.boost.org</a></i>
<b>wxWidget</b>	<b>wxWidget-Bibliotheken</b>	<i><a href="http://www.wxwidget.org">http://www.wxwidget.org</a></i>

Diese Bibliotheken müssen als statische Bibliotheken kompiliert vorliegen und Linker-kompatibel zum eingesetzten Compiler sein. Ihre Header-Dateien müssen ebenfalls verfügbar sein. Während der Entwicklung traten bzgl. der MTL einige Probleme auf, die jedoch im Quelltext behoben werden konnten und diesen Programmen beiliegt. Die Bibliothek `wxWidget` wird in der Version 2.4.2 verwendet.

### 3.1.2 Compiler

Für die erfolgreiche Übersetzung der Bibliotheken und Programme muss ebenfalls ein C++-Compiler vorhanden und installiert sein, der den C++-Standard nach ISO 14418 beherrscht. Das können unter Linux z.B. der Intel High-Performance Compiler *ICC* oder der C++-Compiler der GNU-Compiler-Collection *GCC* sein. Im vorliegenden Fall wird der Compiler *GCC* in der Version 3.2 verwendet.

## 3.2 Übersetzung der erstellten Programme

Um die Programme korrekt übersetzen zu können, sind dem Projekt unter LINUX übliche Makefiles der einzelnen Bereiche mitgeliefert. Um den Vorgang zu starten, ist es ausreichend, im Wurzelverzeichnis des Projektes nach der Anpassung der Datei *make.inc* an die jeweilige System-Umgebung die Übersetzung mittels `make` zu starten.

### 3.2.1 Makefile

Eine Änderung der Makefiles ist nicht erforderlich. Stattdessen existiert die zentrale Datei *make.inc*, in der alle systemrelevanten Einstellungen vorgenommen werden.

### 3.2.2 make.inc

Für die erfolgreiche Übersetzung müssen die nachfolgend dargestellten Variablen u.U. an die jeweilige System-Umgebung angepasst werden.

ROOTPATH	Projekt-Wurzelverzeichnis
COMPILERPATH	Wurzelverzeichnis des zu verwendenden Compilers
CC	zu verwendender C-Compiler
CXX	zu verwendender C++-Compiler
MTLINCLUDEPATH	Wurzelverzeichnis der zu verwendenden MTL
BOOSTINCLUDEPATH	Wurzelverzeichnis der zu verwendenden Boost-Bibliothek
WXWIN_PATH	Wurzelverzeichnis der wxWidget-Bibliothek
VERBOSE	Grad der Informationspreisgabe während der Übersetzung 0 .. minimal, 1 .. ausführlich
CXXFLAGS	Einstellungen für den C++-Compiler
CXXOPTIMIZEFLAGS	Einstellungen, um den erzeugten Code zu optimieren

Nach der Compilierung liegen die nutzbaren Programme im Verzeichnis `build/`. Auf Sie kann über das Verzeichnis `bin/` zugegriffen werden, in denen symbolische Links zu Shellskripten enthalten sind, die wiederum jeweils spezielle Umgebungsvariablen setzen und anschließend die eigentlichen Binaries aufrufen.

### 3.2.3 Relevante Verzeichnisse

Die nachfolgend dargestellten Verzeichnisse sind vorrangig von Bedeutung:

<code>bin/</code>	Links zu Skripten bzw. den eigentlichen Binaries
<code>build/</code>	erstellte Binaries und Skripte
<code>data/</code>	Bilder und Datensätze
<code>doc/</code>	Projekt-Dokumentation
<code>libs/</code>	Bibliotheken, die für dieses Projekt relevant sind
<code>preconfigured/</code>	vorgefertigte Datensätze und bereits gelernte Netzwerke
<code>tools/</code>	Hilfsprogramme wie <i>DSCC</i> , <i>ErrorViewer</i> usw.

Die Quelltexte der Projekt-Software sind in den Verzeichnissen `include/`, `shared/` und `src/` zu finden. Die erforderlichen externen Bibliotheken befinden sich unter `/opt/`.

### 3.3 Die Nutzung der Programme

Wie bereits in Abschnitt 2.1 auf Seite 16 benannt, erfolgt die Handhabung der vorliegenden Software in mehreren Schritten. Ausgehend von der Bildvorverarbeitung und Erstellung der Datensätze über das Lernen des Neuronalen Netzes können anschließend beliebige Bilder auf Gesicht bzw. Nicht-Gesicht getestet werden.

#### 3.3.1 Bildvorverarbeitung mittels DSCC

Das **DataSet Control Center** stellt eine zentrale Möglichkeit der Datensatz-Erzeugung, -Zusammenfügung und -Betrachtung dar. Es bietet eine graphische Oberfläche, in der alle Einstellungen vorgenommen werden können. Hierzu existieren mehrere Karteikarten, die nachfolgend erklärt werden. Gestartet wird das Programm *DSCC* durch `./dscc` im `bin`-Verzeichnis.

##### 3.3.1.1 Image Preprocessing

Unter dieser Karteikarte werden die für das spätere Lernen benötigten Datensätze mittels Konfiguration der Vorverarbeitung erstellt. Dazu sind folgende Schritte erforderlich:

- Laden von Gesichtern und Nicht-Gesichtern
- Konfiguration der Transformer bzgl.
  - zu verwendender Transformer
  - Transformer-Reihenfolge
  - und ihren Einstellungen
- weitere mathematische Operationen
  - Mittelwert-Bild von Gesichtern
  - Art der Differenzbildung
  - Fast-Fourier-Transformation
- Angaben, welche die zu generierenden Bild-Matrizen betreffen
  - die Grösse der Ausgabe-Bildmatrizen
  - ihren Wertebereich
  - vertikales und horizontales Image-Splitting sowie den Overlap-Bereich

In Abbildung 3.1 auf der nächsten Seite ist die Karteikarte der Bildvorverarbeitung als Screenshot dargestellt.

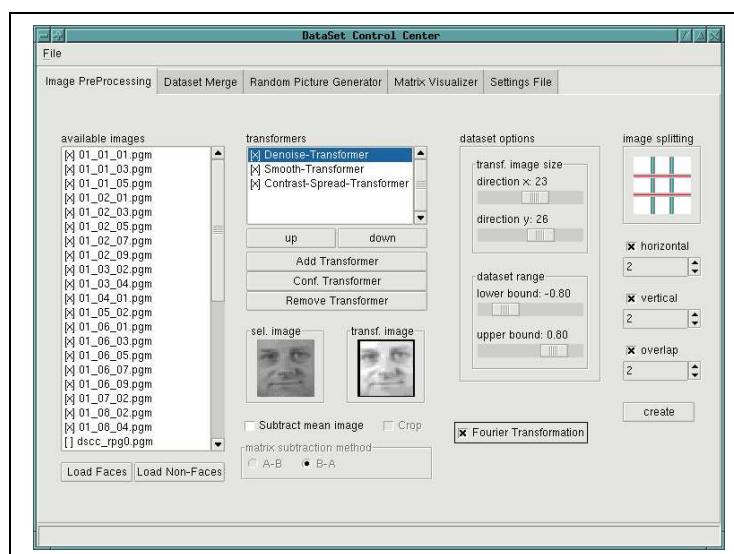


Abbildung 3.1: Panel der Bildvorverarbeitung

An dieser Stelle werden die einflussreichsten Einstellungen vorgenommen, da die Bildvorverarbeitung eine immanent wichtige Rolle bzgl. des Trainings des Neuronalen Netzes trägt.

### 3.3.1.2 Dataset Merge

Unter diesem Panel können bereits vorhandene Datensätze zu einem neuen Datensatz zusammengefasst werden. Die folgenden Operationen sind möglich:

- Auswahl der zu bearbeitenden Datensätze
- Einstellen bzgl. Matrizen-Grösse, Splitting etc.
- Zusammenfassen und Erstellen eines neuen Datensatzes

**ACHTUNG** Sollten auf den Datensätzen unterschiedliche Transformatoren o.ä. vorgenommen worden sein, so werden diese NICHT abgeglichen!

### 3.3.1.3 Random Noise Generator

Im *Random Noise Generator* können zum Test Bilder mit rein zufälligem Inhalt erzeugt werden. Als Vorgaben werden die Anzahl zu erzeugender Bilder, ein Dateinamen-Prefix und ein Verzeichnis benötigt, in das die zu erzeugenden Bilder gespeichert werden sollen.

### 3.3.1.4 Matrix Viewer

Dieses Panel dient zum Betrachten einzelner Bild-Matrizen aus einem Datensatz-Archiv. Durch den `matrix`-Button wird ein Archiv geladen, die darin vorhandenen Bilder werden anschließend in einer Liste mit Index, Namen und Größe aufgelistet. Zusätzlich erhält man die Information, ob es sich bei dem Bild um Gesicht oder ein Nicht-Gesicht handelt. Durch Auswahl eines bestimmten Datensatzes wird dieser im rechten Feld graphisch angezeigt. Die Möglichkeit der Abspeicherung des dargestellten Bildes ist ebenfalls gegeben. Abbildung 3.2 zeigt einen Screenshot dieses Panels.

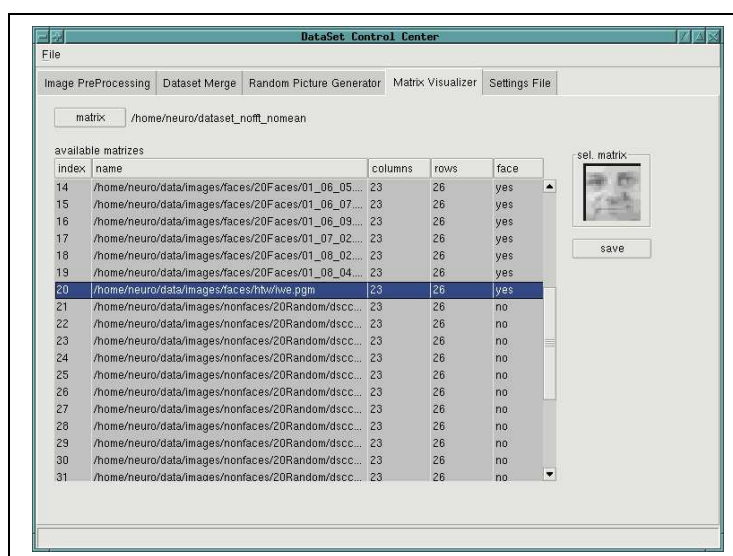


Abbildung 3.2: Der Matrix-Viewer

### 3.3.1.5 Settings File

In diesem Panel werden die Einstellungen der unter Abschnitt 3.4 beschriebenen Settings-Datei getroffen, Abbildung 3.3 zeigt den zugehörigen Screenshot.

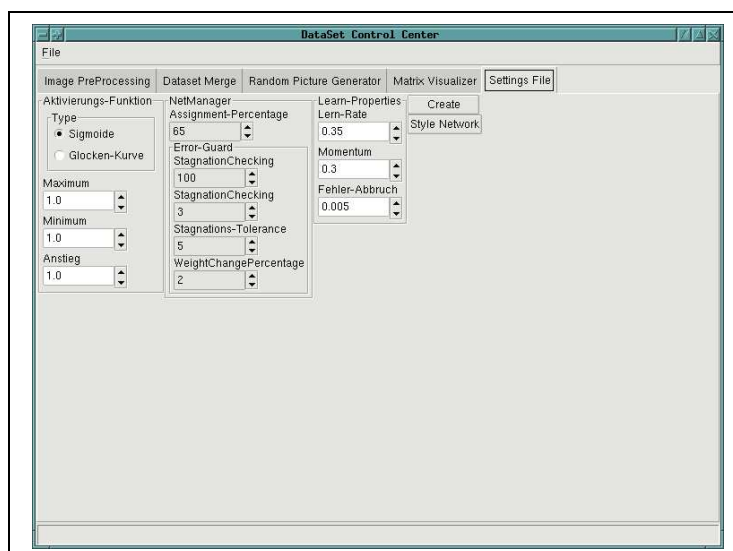


Abbildung 3.3: Das Settings-Panel

## 3.3.2 Lernen des Neuronalen Netzes mittels Datensätzen

### 3.3.2.1 Das Lernprogramm

Das Programm **network\_learner** ist, wie der Name sagt, für das Lernen eines Neuronalen Netzes verantwortlich. Es ist ein reines Kommandozeilen-Programm, die möglichen Einstellungen werden ihm per Kommandozeilen-Parameter übergeben:

```
network-learner --settings=dateiname
                --dataset=dateiname
                [--network=dateiname]
                [--savefile=dateiname]
                [--help]
```

Abbildung 3.4: Parameter des Lernprogramms

Sofern beim Aufruf des **network\_learners** keine Datei zum Speichern des erlernten Neuronalen Netzes angegeben wurde, wird bei Abbruch des Lernens nach einem Dateinamen gefragt. Durch Drücken der ENTER-Taste wird das Netzwerk nicht gespeichert.

Eine Zusammenfassung zeigt nach Darstellung der Soll- und Ist-Outputs eines jeden einzeln durchpropagierten Datensatzes die genutzte reine Rechenzeit des jeweiligen Lern-Durchlaufs.

### 3.3.2.2 Der ErrorViewer

Der ErrorViewer (Aufruf mittels `./bin/error-viewer`) wird parallel zum Lernen des neuronalen Netzes eingesetzt und dient zur graphischen Darstellung des Fehlerverlaufes. Er kann NICHT korrigierend eingesetzt werden, Abbildung 3.5 zeigt die Oberfläche des ErrorViewers.

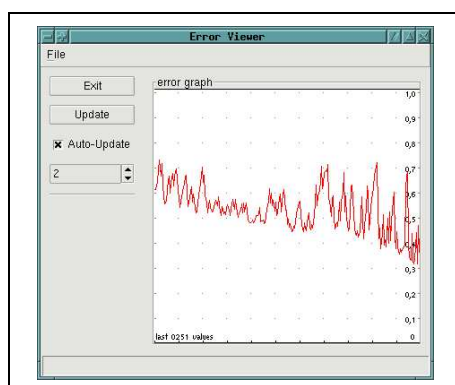


Abbildung 3.5: Die Oberfläche des Error-Viewers

### 3.3.3 Der Face-Checker

Der **Face-Checker** ist das finale Programm zum Prüfen eines Bildes auf Gesicht bzw. Nicht-Gesicht. Er existiert in einer grafischen und einer Kommandozeilen-Version.

#### 3.3.3.1 GUI-basiert

Die graphische Variante wird durch den Aufruf `./bin/face-checker` aus dem Projekt-Wurzelverzeichnis gestartet. Es existieren keinerlei Kommandozeilenparameter, alle Einstellungen werden im Programm selbst vorgenommen. Abbildung 3.6 zeigt einen Screenshot der GUI-Variante des Face-Checkers.



Abbildung 3.6: Die Oberfläche des grafischen Face-Checkers

Durch den `image`-Button wird das zu prüfende Bild gewählt. Mit dem `network`-Button wird das zu ladende Netzwerk eingestellt, mit dem die Prüfung erfolgen soll. Der Button `check` propagiert letztendlich das Muster durch das neuronale Netz und liefert das Ergebnis in einer Meldung zurück.

#### 3.3.3.2 Kommandozeilen-orientiert

Diese Version des **Face-Checkers** wird durch das Programm `fchecker` gestartet. Es bietet die gleiche Funktionalität wie sein graphisches Pendant, erlaubt aber zusätzlich, ein komplettes Datensatz-Archiv zu testen.

```
fchecker --network=dateiname
         --dataset=dateiname
         <image> [images]
         [--help]
```

Abbildung 3.7: Parameter des Programms `fchecker`

Der Parameter `--network` gibt das neuronale Netz an, mit dem der Test durchzuführen ist. Mit `--dataset` lässt sich ein komplettes Datenarchiv mit dem Netzwerk testen, durch Angabe einzelner Bilder werden auch diese getestet. Im Gegensatz zur grafischen Version lassen sich hier beide Möglichkeiten zugleich nutzen.

### 3.4 Die Setting-Datei

Die Setting-Datei liefert Informationen über den Aufbau des neuronalen Netzes, aber auch über das zu verwendende Lernverfahren etc. Da sie eine zentrale Rolle im Projekt spielt, soll sie hier detailliert beschrieben werden.

Es handelt sich hierbei um eine reine Textdatei, in der mit einem Texteditor manuell Einstellungen vorgenommen werden können. Im `bin`-Verzeichnis befindet sich eine Beispiel-Datei, in der alle relevanten Einträge bereits vorhanden sind. Folgende Einstellungen können in der `settings`-Datei vorgenommen werden:

- die genauen Parameter der Aktivierungs-Funktion (Minimum, Maximum, Anstieg)
- der Aufbau des Neuronalen Netzes (parallele Teilnetze, Breite der einzelnen Schichten etc.)
- Angaben zum Lernverfahren wie Lernrate, Momentum und prozentuale Verteilung von Lern- und Kontrollmenge
- Die Anzahl von Iterationen, die insgesamt und zwischen den einzelnen Update-Schritten zurückzulegen sind

**Angaben zur Aktivierungs-Funktion** In der Konfigurations-Datei befindet sich ein Schlüssel mit der Bezeichnung `ActivationFunction`. Darunter befinden sich alle nötigen Einstellungen zur Aktivierungsfunktion. Die Werte der Ableitungsfunktion werden ebenfalls aus diesen Parametern gewonnen. Für weitergehende Informationen zur verwendeten Aktivierungsfunktion sei auf Abschnitt 1.1.1 verwiesen.

**Aufbau des neuronalen Netzes** Die Beschreibung der Netzwerks-Struktur befindet sich unter dem Schlüssel `NetworkStructure`. Jedes Teilnetz wird durch eine eigene Gruppe `PartialNetwork_#` beschrieben. Innerhalb der Gruppen wiederum existiert für jede Schicht ein eigener Eintrag, der ihre Breite angibt. Der letzte Gruppen-Eintrag in der Netzwerk-Struktur ist für das Entscheidungsnetzwerk reserviert. Ein Netzwerk ohne parallele Teilnetze besteht dementsprechend nur aus einem Entscheidungsnetzwerk.

Soll das Netzwerk komplett neu erzeugt werden, werden die Gewichte mittels Zufallszahlengenerator initialisiert. Dabei werden die unter den Schlüsseln `Max` und `MinWeightValue` gegebenen Werte für die obere bzw. die untere Grenze genutzt.

**Angaben zum Lernverfahren** In der Konfigurations-Datei befinden sich Schlüssel mit Namen `LearnRate`, `Momentum`, `ErrorBreak` und `AssignmentPercentage`. `LearnRate` und `Momentum` sind selbsterklärend. `AssignmentPercentage` regelt die prozentuale Verteilung der Eingangsdaten zwischen Lern- und Kontrollmenge, `ErrorBreak` gibt den minimal erlaubten Fehler für einen Abbruch an.

**Anzahl von Iterationen** Mit Hilfe des Schlüssels `MainLoopCount` lässt sich die maximale Anzahl von Iterationen angeben, die der Lernprozess zurücklegen soll. Unter dem Schlüssel `ClientLStep` befindet sich die Anzahl der durchzuführenden Update-Schritte, ehe die Gewichtsänderungsmatrizen in die Gewichtsmatrizen integriert werden.

## 3.5 Ein Beispiel-Durchlauf

Um die Anwendung der einzelnen Programme besser illustrieren zu können, soll hier exemplarisch ein Durchlauf des gesamten Prozederes gezeigt werden. Es lässt sich in vier Teile gliedern:

1. Erzeugung des Datensatz-Archivs
2. Konfigurierung des Neuronalen Netzes
3. der eigentliche Lernvorgang
4. Überprüfung des gelernten Neuronalen Netzes

Die Abspeicherung der selbst erstellten Datensätze, Konfigurationseinstellungen und gelernter Neuronaler Netze kann bspw. im Verzeichnis `/tmp/` erfolgen.

### 3.5.1 Erzeugung der Datensätze

Im `DSCC` lassen sich unter den Panels “Image-PreProcessing”, “Dataset Merge” und “Random Noise Generator” Datensätze erzeugen und manipulieren. Dem Panel “Image-PreProcessing” ist hierbei vorrangig Beachtung zu schenken, da hier umfangreiche Manipulationen der Daten vorgenommen werden können. Durch Drücken des `create`-Buttons

werden die gewählten Bilder entspr. der konfigurierten Transformer bearbeitet und anschließend in einem Datensatz-Archiv zusammengefasst.

### 3.5.2 Konfiguration des Neuronalen Netzes

Hier wird das Aussehen bzw. die Form des Neuronalen Netzes bestimmt. Auch Einstellungen für das Lernverfahren wie Momentum, Lernrate etc. werden hier getroffen. Dafür wird das Panel “Settings File” im *DSCC* bemüht, alternativ kann die Textdatei auch direkt mit einem Text-Editor bearbeitet werden. Mit Ausnahme der Netzwerk-Struktur sind alle anderen Einstellungen mit Standardwerten voreingestellt, die Netzwerkstruktur kann und muss individuell angelegt werden.

### 3.5.3 Das Lernen

Das Lernen erfolgt mit dem Programm **network\_lerner**. Zum Starten genügen die Angabe des zu lernenden Datensatzes sowie die zu nutzende *Settings*-Datei. Für die Verfolgung der Fehlerentwicklung kann während des Lernens das Programm **ErrorViewer** herangezogen werden.

### 3.5.4 Überprüfung des Ergebnisses

Die Prüfung der Ergebnisse bzw. die Nutzung des erlernten Netzwerkes erfolgt mit dem Programm **Face-Checker**. Nun können auch Bilder geprüft werden, die nicht im Datensatz-Archiv des gelernten Neuronalen Netzes enthalten sind.

Für detailliertere Informationen sei auf Kapitel 5 bzw. die Projekt-Quelltexte verwiesen.

# Kapitel 4

## Gewonnene Erfahrungen

### Inhaltsangabe

---

<b>4.1</b>	<b>Bildmaterial . . . . .</b>	<b>31</b>
<b>4.2</b>	<b>Bildvorverarbeitung . . . . .</b>	<b>31</b>
4.2.1	Transformer und ihre Einstellungen . . . . .	31
4.2.2	Allgemeine Datensatz-Einstellungen . . . . .	32
<b>4.3</b>	<b>Lern-Parameter . . . . .</b>	<b>33</b>
<b>4.4</b>	<b>Lernfortschritt und Ergebnisse . . . . .</b>	<b>33</b>
<b>4.5</b>	<b>Fortführende Aspekte . . . . .</b>	<b>35</b>

---

Für die erfolgreiche Bewältigung der gestellten Aufgabe haben sich bestimmte Vorgehensweisen und Einstellungen herauskristallisiert, die in den nun folgenden Abschnitten näher dargestellt werden sollen.

## 4.1 Bildmaterial

Die Bearbeitung der Bilder erfolgt in Graustufen, um den Informationsgehalt bzgl. des Neuronalen Netzes nicht allzu hoch anzusetzen. Für die Bildvorverarbeitung sollten alle Gesichter in einer einigermaßen einheitlichen Auflösung, gleichen Lichtverhältnissen und ähnlich gewählten Bildausschnitten bzw. Gesichts-Regionen vorliegen. Für die Auflösung der unbearbeiteten Gesichts-Bilder empfiehlt sich eine Bildgröße von ca. 30x34 Pixeln.

Sowohl die Bilder der Gesichter als auch die Bilder der Nicht-Gesichter müssen im Dateiformat `.pgm` vorliegen, ggf. müssen mit externen Programmen Bilder dieses Dateiformats erzeugt werden.

## 4.2 Bildvorverarbeitung

Das Lernverhalten und der damit verbundene Erfolg des Netzwerk-Trainings hängen sehr stark von der Komplexität der dem Neuronalen Netzwerk präsentierten Daten ab. Den Aspekten der Informations-Reduktion und Herausarbeitung markanter Merkmale kommt demnach eine zentrale Rolle zu. Die im Zusammenhang der Gesichtserkennung erforderlichen und sich als praktikabel erwiesenen Manipulationen sind in den nachfolgenden Abschnitten kurz dargestellt.

### 4.2.1 Transformer und ihre Einstellungen

Grundsätzlich lassen sich alle Transformer mehrfach und in jeder gewünschten Reihenfolge einsetzen, sofern sie reine Bild-Transformatoren sind. Ziel der Verwendung der Transformer ist es, mit möglichst wenig Rechenaufwand die Datensätze derart vorzuverarbeiten, dass nach Möglichkeit allein durch die Transformatoren signifikante Unterschiede zwischen Gesichtern und Nicht-Gesichtern entstehen.

Hierfür stehen sowohl reine Bild-Transformatoren, deren Ergebnisse wiederum Bilder ergeben, als auch Transformatoren mit mathematischem Hintergrund zur Verfügung, deren Ergebnisse keine Bilder mehr im herkömmlichen Sinne sind und nur noch rein mathematisch weiterverarbeitet sind.

Die in der Auflistung auf der nächsten Seite dargestellten reinen Bild-Transformatoren haben sich in der angegebenen Reihenfolge als nützlich erwiesen.

1. Resizer (vorübergehende Erweiterung des Operationsraumes)
2. Smoother (Glättung)
3. Contrast-Spreader (Ausgleich von Helligkeits-Unterschieden)
4. Resizer (Reduktion der Bildgrößen auf verarbeitbares Maß)

Hierbei sind folgende Einstellungen als praktikabel anzusehen:

Resizer	:	Direction x	=	46
	:	Direction y	=	52
Smoother	:	Sigma	=	0.5
Resizer	:	Direction x	=	23
	:	Direction y	=	26

Abbildung 4.1: Praktikable Transformer-Einstellungen

Für die Bildvorverarbeitung haben sich folgende Transformatoren mit mathematischem Hintergrund als nützlich erwiesen:

1. ggf. Mittelwert-Bildung bzgl. Gesichtern (gemeinsame Merkmale aller Gesichter)
2. Fast-Fourier-Transformation

Für die anderen aufgeführten Transformatoren sind keine Konfigurations-Einstellungen vorgesehen. Die Art der Differenzbildung bzgl. des Mittelwert-Bildes bzw. seiner Fast-Fourier-Transformation und der Fast-Fourier-Transformation eines jeden Bildes (Datensatzes) kann zwar im *DSCC* festgelegt werden, hat jedoch bzgl. des Lernens im vorliegenden Fall kaum Auswirkungen auf das erlernte Neuronale Netz und ist neben weiteren Dingen hauptsächlich für Testzwecke vorgesehen.

#### 4.2.2 Allgemeine Datensatz-Einstellungen

Die Datensätze werden einheitlich in einer Größe von 23x26 Pixeln bzw. Matrix-Spalten und -Zeilen angelegt. Eine anschließende Normalisierung in Abhängigkeit der minimalen und maximalen "Farbwerte" auf den Bereich von -0.8 ... 0.8 erfolgt für alle Datensätze. Die Normalisierung auf den Bereich von -1 ... 1 ist nicht sinnvoll, da diese Werte, bedingt durch die Aktivierungsfunktion eher schwierig denn ohne weiteres erreicht werden können. Das Intervall von -0.8 ... 0.8 erweist sich im vorliegenden Fall als solide Basis für die weiteren Berechnungen.

Die Datensätze sind zusätzlich gesplittet, um den Teilnetzen eine Spezialisierung bestimmter Gesichtsmkmale zu ermöglichen. Als praktikable Einstellungen haben sich je ein bis zwei Aufteilungen in jeder Richtung sowie eine Überlappung von zwei “Pixeln” herauskristallisiert.

### 4.3 Lern-Parameter

Die nachfolgend dargestellten Parameter des Neuronalen Netzes ergeben praktikable Werte. Sie sind in der *Settings*-Datei enthalten.

ActivationFunction	1, Gaußsche Glockenkurve
NetManagerLStep	100
LearnRate	0.35
ErrorBreak	0.0025 - 0.0075, je nach Fehlerverlauf
Momentum	0.3
MainLoopCount	2500 - 5000, je nach Aufbereitung der Datensätze und/oder Lerngeschwindigkeit
MaxLearnCount	5
AssignmentPercentage	65
StepOfStagnationChecking	100
RStepsStagnationChecking	3
StagnationTolerancePercentage	5
WeightChangePercentage	2

Abbildung 4.2: Praktikable Netzwerk-Einstellungen

Die jeweiligen Einstellungen sind einer Standard-*Settings*-Datei etwas näher durch einen entspr. Kommentar beschrieben und sollen an dieser Stelle nicht weiter betrachtet werden.

### 4.4 Lernfortschritt und Ergebnisse

Bevor das Training bzw. Lernen überhaupt beginnen kann, muss die Gewichtsmatrix initialisiert werden. Wie erwartet, hängt die Geschwindigkeit des Lernens erheblich von der zufälligen Initialisierung ab, gleiche Datensätze mit gleichen Einstellungen können bei verschiedenen Aufrufen teils erhebliche unterschiedliche Ergebnisse zu Tage fördern. Zur Zeit existiert kein bekanntes zuverlässiges Verfahren, dass bessere Ergebnisse in Verbindung mit der Initialisierung herbeiführt.

Die Auswahl der Transformatoren hat erheblichen Einfluss auf die Lern-Geschwindigkeit und den Lern-Erfolg des Neuronalen Netzes, Abbildung 4.3 gibt darüber Aufschluss.

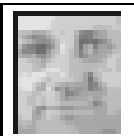
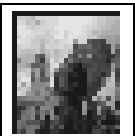

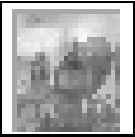
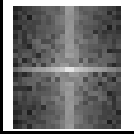
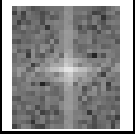
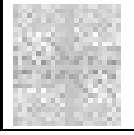
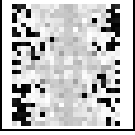
Mittelw.-Bild	FastFourier	Gesicht	Nicht-Gesicht	Eignung/Erfolg
–	–			•
✓	–			•
–	✓			○ / •
✓	✓			•
Eignung/Erfolg: • ≐ weniger gut    ○ ≐ mäßig bis gut    ● ≐ gut bis sehr gut				

Abbildung 4.3: Ergebnisse bzgl. verschiedener Bild-Vorverarbeitungen

Neben der Bildvorverarbeitung und der damit verbundenen Informations-Reduktion bei gleichzeitiger Merkmals-Hervorhebung ist ein weiterer Aspekt zu beachten. Um dem Netzwerk das Lernen einer Unterscheidung nach Gesicht bzw. Nicht-Gesicht zu ermöglichen, müssen neben den Positiv-Datensätzen (Gesichter) auch Negativ-Beispiele (Nicht-Gesichter) in den Lernprozess eingebracht werden. Die Auswahl der Negativ-Beispiele ist hierbei problematisch, denn die Menge der Nicht-Gesichter ist bei weitem größer als die der Gesichter und eine diesbzgl. Generalisierung entspr. schwierig.

Der Vorverarbeitung der Bilder bzw. Datensätze mit reinen Bild-Transformatoren sind jedoch Grenzen gesetzt, da die Merkmals-Extraktion nur bis zu einem gewissen Grade erfolgen kann. Vor allem die Transformatoren mit mathematischem Ansatz haben daher entspr. Relevanz. Wie sich zeigt, genügt die Herausrechnung des Mittelwert-Bildes eines Gesichts allein jedoch nicht. Bei einfarbigen Bildern kehrt sich der Sachverhalt sogar um, weil durch die Differenzbildung als Resultat ein Bild mit einem Gesicht, nämlich dem des Mittelwert-Bildes entsteht. Die Fast-Fourier-Transformation hat hier eine große Bedeutung.

Im vorliegenden Projekt genügen die vorhandenen Transformer bzw. die durch ihre Verwendung letztendlich hervorgerufenen Ergebnisse den selbstgestellten Anforderungen, die

Bilder werden hinreichend gut klassifiziert. Für Bilder jedoch, die ein ähnliches Frequenz-Spektrum aufweisen, reichen auch diese Transformatoren nicht aus. Ihnen kann sich in zukünftigen Projekten gewidmet werden.

Die durchschnittliche Rechendauer eines Lern-Prozederes beträgt mit einer Datensatz-Archiv-Größe von 40 Datensätzen unter den gegebenen Einstellungen und Parametern bei 3000 durchlaufenden Schritten etwa 17 Minuten. Hierbei kommt ein x86-kompatibler PC mit AMD Athlon 1400MHz CPU und 384MB Arbeitsspeicher zum Einsatz.

## 4.5 Fortführende Aspekte

Für eine weitere Behandlung der Problematik ergeben sich diverse Betätigungsfelder. So können bspw. weitere Transformatoren erarbeitet werden, die markante Merkmale eines Gesichtes noch besser herausarbeiten und bspw. den Farbraum eines Gesichtes (bei Farbbildern) berücksichtigen.

Darüber hinaus kann eine Modellierung des Netzwerks hinsichtlich verteilten Rechnens das Einsatz-Spektrum der Problemstellung erweitern. Wie bereits in Abschnitt 2.5 gezeigt, ist dies jedoch keine triviale Angelegenheit.

Durch die Erstellung weiterer Netzwerk-Typen lässt sich das Einsatzgebiet der vorliegenden Software ebenfalls erweitern. Die Basis für diese Erweiterungen ist bereits vorhanden. Um diese Basis besser verstehen zu können, sollen im anschließenden Kapitel 5 einige Einblicke in die Entwicklungs-Interna gegeben werden.

# Kapitel 5

## Entwicklungs-Interna

### Inhaltsangabe

---

5.1	Bibliotheken . . . . .	37
5.2	Entwicklungsumgebung und Compiler . . . . .	38
5.3	Klassenstruktur . . . . .	38

---

In diesem Kapitel wird ein grober Einblick in die Entwicklungs-Interna gegeben. Die wesentlichen und grundlegenden Ideen und Konzepte sind hier dargestellt. Aufgrund der hohen Komplexität der entstandenen Software wird hier jedoch nicht jedes Detail behandelt, stattdessen sei auf den dokumentierten Quelltext verwiesen.

## 5.1 Bibliotheken

Für die Entwicklung wurde auf eine Reihe unter der **GNU GPL** frei erhältlichen Bibliotheken für verschiedene Aufgabenbereiche zurückgegriffen. Dazu zählen unter anderem Dinge wie Matrizenrechnung, Bildbearbeitung, das Lesen und Schreiben von Konfigurations-Dateien sowie SmartPointer und Zufalls-Generatoren.

Matrizen	MTL
Bildbearbeitung	IMG-Lib / wxIMG-Lib
Config-Dateien	CFG-Lib
SmartPointer, Zufalls-Generatoren	BOOST
GUI-Framework	wxWidget

Im Quellcode des Projektes sind die Bibliotheken IMG-Lib, wxIMG-Lib und CFG-Lib bereits enthalten. Die anderen Bibliotheken können unter den angegebenen Internet-Adressen (siehe Abschnitt 3.1.1 auf Seite 20) heruntergeladen werden.

**MTL** Bei der **Matrix Template Library** handelt es sich um eine ausgeklügelte Bibliothek zur Matrizenrechnung. Neben einfachen Operation wie der Multiplikation mit einem Skalar und anderen Matrizen oder dem Transponieren stellt sie unter anderem auch Inverse,  $L \times R$ -Zerlegung, die Arbeit mit Submatrizen u.ä. zur Verfügung.

Die MTL ist in C++ implementiert und basiert auf dem Template-Mechanismus von C++. Die zu verwendenden Datentypen müssen zum Compilieren bekannt sein, wenn die Größen der Matrizen bekannt sind können auch diese angegeben werden. Durch dieses Wissen um verwendete Datentypen und Größen sind hochoptimierenden Compilern wie dem *ICC* alle Möglichkeiten offen, den erzeugten Code zu verbessern. Download und weitere Informationen siehe bspw. unter [Bib4].

**IMG / wxIMG** Die Bibliothek IMG-Lib erlaubt das Lesen, Bearbeiten und Speichern von Bildern. Neben der Möglichkeit zur pixelweisen Bildmanipulation werden sogenannte Transformatoren zur Verfügung gestellt, die bestimmte Aufgaben wie Bildfaltungen, Kantenglättungen u.ä. erledigen.

Die Bibliothek wxIMG-Lib stellt hierbei eine Schnittstelle zwischen den Transformern der IMG-Lib und dem GUI-Framework wxWidget dar. Über diese Schnittstelle lassen sich die Transformatoren durch entspr. Menüs konfigurieren.

**BOOST** Hierbei handelt es sich um eine Sammlung von Hilfsklassen, die unter anderem SmartPointer, Zufallsgeneratoren u.v.m. umfasst. Für weitere Informationen und Download-Möglichkeiten sei auf [Bib3] verwiesen.

## 5.2 Entwicklungsumgebung und Compiler

Die Entwicklung erfolgte unter Linux mit gängigen Tools wie *make*, *vim*, *mc* und weiteren. Als Compiler kamen neben dem Intel-C-Compiler *ICC* hauptsächlich der C-/C++-Compiler der GNU-Compiler-Collection *GCC* zum Einsatz. Es sollte aber auch jeder andere ISO 14418 kompatible C++-Compiler funktionieren.

## 5.3 Klassenstruktur

Ziel der Klassenstruktur ist es, zum einen ein modulares und erweiterbares Netz zu ermöglichen, dessen Parameter schnell und flexibel geändert werden können. Die Modularität soll es ermöglichen, das Netz auch über mehrere Computer verteilt in einem Rechnerverbund lernen zu lassen. Sie soll aber auch möglichst performant sein und dem Compiler Ansätze zum Optimieren liefern. Abbildung 5.1 zeigt die resultierende Struktur.

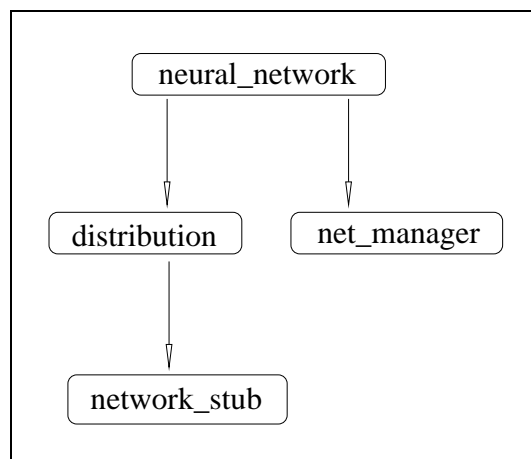


Abbildung 5.1: Klassenstruktur des Neuronalen Netzes

**neural\_network** ist hierbei nur eine Sammlung von Objekten, an die es die einzelnen Funktionen durchreicht. Die Klasse selbst definiert keine eigene Funktionalität.

**distribution** regelt die logische Anordnung und physikalische Verteilung der einzelnen Netzwerk-Schichten und ihre Kommunikation untereinander. Es existiert im Augenblick nur eine lokale Verteilung, einer Erweiterung zu einer echten physikalischen Verteilung in einem Netzwerk steht aber nichts im Wege. Die Probleme bzgl. einer physikalischen Verteilung auf mehrere Rechenanlagen wurden bereits in Abschnitt 2.5 auf Seite 18 dargestellt.

**net\_manager** ist für die Update-Strategie und die Überwachung des Lernverfahrens verantwortlich. Er veranlasst das Update der Gewichte aus den Änderungs-Matrizen, er verteilt die Datensätze auf Lern- und Kontrollmenge und ist für die Überwachung des Fehlerabstiegs verantwortlich, um notfalls korrigierend einzugreifen.

Zur Korrektur ergeben sich zwei Möglichkeiten. Einerseits kann eine Änderung der Lernrate erfolgen, um die Gewichtsänderung herbeizuführen. Die zweite Möglichkeit besteht in der direkten Änderung der Gewichte, z.B. um einen geringen, zufallsgesteuerten Betrag. Ein letzter Ausweg besteht in der Änderung des verwendeten Momentums, um den neu berechneten Werten mehr Gewicht zu verleihen. Ist der maximal erlaubte Fehler des Netzwerks vom Lernvorgang erreicht worden, oder wurde die maximale erlaubte Anzahl von Lernschritten überstiegen, wird vom **net\_manager** der Abbruch veranlasst.

**network\_stub** repräsentiert eine einzelne Schicht des Netzes. Sie erhält ihre Daten von der **distribution** und speichert erforderliche Daten lokal ab. Dadurch ist sie unabhängig von anderen Schichten und kann mit minimalem Kommunikationsaufwand physikalisch in einem Netzwerk verteilt werden. Sie propagiert einen Input-Wert durch sich und speichert die dabei auftretende Aktivierung für einen späteren Update-Schritt. Somit ist sie auch für das verwendete Lernverfahren verantwortlich, entscheidet also über Back-Backpropagation, eventuell mit Momentum.

Zu dieser Klassenübersicht kommen noch eine Reihe von Hilfsklassen hinzu, bspw.:

CmdLineParser	parst die Kommandozeile und sucht nach Werten der Form $-param [= value]$
ConfReader	zum Parsen der Settings-Datei (Abschnitt 3.4)
DatasetLoader	lädt die Datensätze eines Archivs und erledigt das Splitten/Aufteilen der Bilder
EventLogger	zum Logging
matrix_string	speichert eine Matrix in serialisierter Form bzw. lädt sie daraus
matrixstream_of	speichert eine Matrix oder <code>matrix_string</code> in einer Datei
matrixstream_if	liest eine Matrix aus einer Datei
RandomNumberGenerator	Globaler Zufallszahlengenerator
ring_buffer	ein einfacher Ring-Puffer fester Größe, der als Template-Parameter gegeben ist
basic_string	Klassen-Template für einen binären String
STC	setzt die C++- <code>unexpected</code> und <code>terminate</code> -Handler

SHMSegment	Shared-Memory-Segment zum Datenaustausch zwischen Network-Learner und ErrorViewer
vectorstream_if	Einlesen eines Vektors aus einer Datei
vectorstream_of	Speichern eines Vektors in einer Datei

Abbildung 5.2: Übersicht der Hilfsklassen

Für eine weitergehende Erklärung der Klassen wird an dieser Stelle auf den Quell-Text und die darin befindlichen Kommentare verwiesen. Dort sind alle relevanten Klassen und Funktionen beschrieben und detailliert erklärt.

## Anhang A

# Nutzungsrecht und -erklärung

### Nutzungsrecht

Jede Form der Nutzung, insbesondere Vervielfältigung, Veröffentlichung und/oder Nutzung der Quellcodes und/oder Ergebnissen dieser Arbeit bzw. dieses Projektes oder Teilen davon bedürfen der ausdrücklichen und schriftlichen Zustimmung BEIDER Autoren dieser Arbeit bzw. dieses Projektes sowie der Hochschule für Technik und Wirtschaft Dresden, Deutschland.

Für Forschungszwecke innerhalb der Hochschule für Technik und Wirtschaft Dresden, Deutschland, sind die schriftlichen Zustimmungen BEIDER Autoren einzuholen.

### Nutzungserklärung

Die Nutzung der Software des Projektes “Gesichtserkennung” geschieht auf eigene Gefahr und Verantwortung. Trotz sorgfältiger Konzeptionierung und Programmierung kann keinerlei Garantie für evtl. auftretende Datenverluste und/oder Schäden an Hard- und/oder Software gegeben werden. Jegliche Gewährleistungs- und/oder Haftungsansprüche sind ausgeschlossen. Mit der Nutzung der Software zur Gesichtserkennung mittels Neuronaler Netze bzw. Teilen davon werden Nutzungsrecht und -erklärung zugestimmt und anerkannt. In jedem anderen Fall darf die Software nicht genutzt werden.

# Literaturverzeichnis

- [ 1 ] Ming-Hsuan Yang, David Kriegman, Narendra Ahuja : *Detecting Faces in Images : A Survey*, (IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 24, №1 , Jan. 2002)
- [ 2 ] Rein-Lien Hsu, Mohammed Abdel-Mottaleb, Anil K. Jain : *Face Detection in Color Images*, (IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 24, №5, May 2002)
- [ 3 ] W. Kinnebrock : *Neuronale Netze - Grundlagen, Anwendungen, Beispiele*, (Oldenbourg, 1994)
- [ 4 ] C. M. Bishop : *Neural Networks for Pattern Recognition*, ( Clarendon Press, Oxford, 1995 )
- [ 5 ] Prof. Dr.-Ing. Karl-Friedrich Kraiss : *Identifikations- und Klassifikationsverfahren - Seminarreihe am Lehrstuhl fuer technische Informatik*, (Rheinisch-Westfaelische Technische Hochschule Aachen, Juli 2001)
- [ 6 ] Charles Poynton : The Color FAQ,  
(<http://www.inforamp.net/~poynton/ColorFAQ.html>)
- [ 7 ] K. Holtorf : *Das Handbuch der Grafikformate, 2. Auflage*. (Franzis` , 1994)
- [ 7 ] Peter Haberäcker : *Digitale Bildbearbeitung*. (Hanser, 1989)
- [ 8 ] Wikipedia : Künstliches neuronales Netz,  
([http://de.wikipedia.org/wiki/Künstliches\\_neuronales\\_Netz](http://de.wikipedia.org/wiki/Künstliches_neuronales_Netz))
- [Bib1] MPI : ([www-unix.mcs.anl.gov/mpi/](http://www-unix.mcs.anl.gov/mpi/)) The Message Passing Interface Standard
- [Bib1] MPI : ([www-unix.mcs.anl.gov/mpi/mpich](http://www-unix.mcs.anl.gov/mpi/mpich)) MPICH - A Portable Implementation Of MPI
- [Bib2] LAM-MPI : ([www.lammpi.org](http://www.lammpi.org)) Eine Implementierung des MPI-Standards

- [Bib3] C++ Boost : ([www.boost.org](http://www.boost.org)) C++-Bibliothek für verschiedene algorithmische Probleme
- [Bib4] Matrix Template Library : ([www.osl.iu.edu/research/mtl/](http://www.osl.iu.edu/research/mtl/)) Bibliothek für Matrizen-Operationen unterschiedlichster Art
- [Img1] MIT Database ( <ftp://whitechapel.media.mit.edu/pub/images> ) : Faces of 16 people, 27 of each person under various illumination conditions, scale and head orientation.
- [Img2] FERET Database ( <http://www.nist.gov/humanid/feret> ) : A large collection of male and female faces. Each image contains a single person with certain expression.
- [Img3] Yale Database ( <http://cvc.yale.edu> ) : Face images with expressions, glasses under different illumination conditions.
- [Img4] AT&T (Olivetti) Database ( <http://www.uk.research.att.com> ) : 40 subjects, 10 images per subject.
- [Img5] Rowley et al. ( <http://www.cs.cme.edu/har/faces.html> )