Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Mahmoud Dariti

Erkennung von Verkehrszeichen mit Konvolutionalen Neuronalen Netzen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Mahmoud Dariti

Erkennung von Verkehrszeichen mit Konvolutionalen Neuronalen Netzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 8. April 2016

Mahmoud Dariti

Thema der Arbeit
Erkennung von Verkehrszeichen mit Konvolutionalen Neuronalen Netzen

Stichworte
Verkehrszeichenerkennung, Neuronale Netze, Objekterkennung, Bildverarbeitung, Konvolutionale Neuronale Netzwerke, Faltungsnetzwerke, Künstliche Intelligenz

Kurzzusammenfassung
Die Verkehrszeichenerkennungssysteme sind ein wesentlicher Bestandteil der zukünftigen Fahrzeuge. Diese Arbeit stellt das Thema vor, präsentiert die erfolgreichsten Verfahren und zeigt ein Realisierungsbeispiel mit Hilfe der Konvolutionalen Netze.

Mahmoud Dariti

Title of the paper
Traffic Sign Recognition with Convolutional Neural Network

Keywords
Traffic Sign Recognition, Neural Networks, Convolutional Neural Networks, Image recognition, Image processing, Machine learning, Artificial intelligence, Computer vision

Abstract
Traffic Sign Detection and Recognition system is becoming an essential component in the next generation of vehicles. This work presents the most successful recognition methods, and shows an implementation using convolutional neural network.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einführung

Die Automobilindustrie entwickelt sich weiter in Richtung der Fahrautomatisierung. Mit Produkten wie Google driverless-car[1] sind selbstfahrende Autos schon für Testfahrten im öffentlichen Staßenverkehr zugelassen. Für solche Autos sind Systeme nötig, die den Fahrer ersetzen. Um diesen Systemen vertrauen zu können, müssen sie sehr sorgfältig entwickelt werden, insbesondere was die Sicherheit betrifft.

Eine sehr wichtige Komponente von Verkehrssystemen sind die Verkehrszeichen, durch ihren Einsatz wird der Straßenverkehr reguliert. Ihre Beachtung ist ein sehr wichtiger Aspekt für die Sicherheit der Verkehrsteilnehmer. Die Verkehrszeichen sind Schilder, die gemacht wurden, um möglichst leicht von Menschen erkannt zu werden. Sei es im Dunkeln, bei Nebel, oder unter besonderen Lichtverhältnissen, durch ihre Form, ihre Farbkombinationen und ihren Kontrast sind sie meist leicht erkennbar. Diese Eigenschaften werden auch ausgenutzt, um ein System zu entwickeln, das die Verkehrszeichenerkennung durch den Computer möglich macht. Ob bei Verwendung von Bildverarbeitungsverfahren (HOG, SIFT..) oder künstlicher Intelligenz: das Ziel ist ein System zu entwickeln, das mindestens die menschliche Leistung in der Verkehrszeichenerkennung erreicht.

[1] Homepage: *https://www.google.com/selfdrivingcar*

1.1 Ziel der Arbeit

Verkehrszeichen aufzuspüren und zu erkennen durch einen Computer ist das Thema dieser Arbeit. Viele Verfahren wurden dazu entwickelt. Diese Arbeit stellt die erfolgreichsten davon vor und diskutiert die Grundlagen der Bilderkennungssysteme und die der Konvolutionalen Neuronalen Netze (CNN). Am Ende wird eine Realisierung durch ein CNN gezeigt.

Diese Arbeit verwendet das "Caffe" Framework Jia et al. (2014), eine sehr gelungene Framework für Neuronale Netze. Für das Training sind Datasets von der Webseite des German-benchmark-Team[2] Stallkamp et al. (2012) benutzt worden.

1.2 Aufbau der Arbeit

Im folgenden Kapitel 2.1 (S. 3) werden die Grundlagen der Objekterkennung diskutiert, ihre verschiedenen Phasen erklärt und die verwendeten Verfahren gezeigt. Der zweite Teil 2.2 (S. 6) des Kapitels befasst sich mit den Verkehrszeichen, Herausforderungen werden beschrieben und Lösungen vorgestellt. Kapitel 3 (S. 8) präsentiert die von Autoherstellern verwendeten Systeme und wirft einen Blick auf die jüngsten Forschungen zur Verkehrszeichenerkennung und stellt aktuelle CNN-Varianten vor.

In Kapitel 4 geht es um die CNN, im ersten Teil 4.1 (S. 11) werden die Grundlagen der Neuronalen Netze durch das Multilayer Perzeptron Netz (MLP) erläutert. Darauf aufbauend wird im zweiten Teil 4.2 (S. 18) das CNN detailliert erläutert. In 4.3 (S. 23) sind Anwendungen der CNN in der Verkehrszeichenerkennung durch Beispiele erklärt. Kapitel 5 (S. 27) zeigt eine Realisierung eines Verkehrszeichenerkennungssystems mit Hilfe der CNN. In 5.1 (S. 27) werden die dazu verwendeten Tools vorgestellt, danach die Realisierung in 5.2 (S. 33) präsentiert. Die Ergebnisse werden dann in Kapitel 6 analysiert und Kapitel 7 bietet eine Zusammenfassung.

[2] Homepage: http://benchmark.ini.rub.de

2 Verkehrszeichenerkennung im Allgemeinen

Das folgende Kapitel beschäftigt sich mit der Objekterkennung im allgemeinen. hier werden die verschiedene Schritte des Bilderkennungsprozess beschrieben und die benutzte Verfahren für jede Schritt gezeigt, schließlich wird die Anwendung der Bilderkennungsverfahren in Verkehrszeichenerkennung diskutiert.

2.1 Objekterkennung

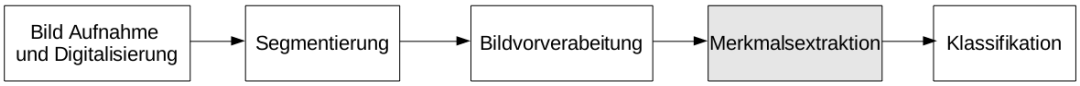


Abbildung 2.1: Bilderkennungsprozess

Im allgemeinen lässt sich der Objekterkennungsprozess in fünf Phasen durchführen: **Bild Aufnahme und Digitalisierung**, Dies wird in der Regel durch Hardware gemacht(z.B: Digitale Kamera). Das Ziel der **Bildvorverabereitung** ist es, die Farben und Kanten besser erkennbar zu machen und Störungen zu eliminieren sowie Kontrast und Lichtintensität zu korrigieren. Dadurch werden Objekte stärker vom Hintergrund unterschieden. In der **Bildsegmentierungs**-Phase werden die Positionen von den zu erkennenden Objekten im Bild gefunden. Diese Phase ist sehr wichtig, weil alles was nicht als potentielle Objekt in dieser Etappe anerkannt wird, wird gefiltert und

nicht mehr in Anspruch genommen. Die **Merkmalsextraktions**-Phase, hier werden symbolische Informationen (wie Form, Farbe...) aus den gefundenen Bildbereichen extrahiert. Diese Informationen werden in der Klassifikation verwendet. Die letzte Schritt ist die **Klassifikation**, hier werden die gefundene Merkmale vordefinierten Klassen zugewiesen und damit identifiziert.

2.1.1 Detektion

Das Eingangsbild ist in vielen Anwendungsfällen zu groß und enthält viele Informationen die nicht benötigt werden. Das Ziel der Detektion ist alle unnötige Komponenten des Bildes (Hintergrund, uninteressante Objekte...) zu entfernen, und potentielle Stellen im Bild zu finden, die das zu erkennende Objekt abbilden können.

Bildvorverarbeitung

Die Hauptaufgaben der Bildvorverarbeitung sind Kontrastverbesserung und Rauschminderung. Damit werden Kanten sichtbarer und Objekte einfacher zu finden. Die verwendete Methoden können in drei Kategorien unterteilt werden[1]:

- **Bildpunktoperationen:** Diese Operatoren beeinflußen einzelne Pixel ohne Beziehung der Umgebung (Bsp. Lineare Skalierung, Binarisierung eines Bildes...).
- **Lokale Operationen:** Diese Operationen verändern das Wert eines Pixels in Abhängigkeit vom Umgebung dieses Pixel (Bsp: Median, Faltung...).
- **Globale Operationen:** Diese Operationen verändern das Wert eines Pixel in Abhängigkeit vom gesamten Bild (Bsp: Tiefpass/Hochpass, GrauwertÃďqualisierung...).

[1] Sehr detaillierte Informationen sind in *http://www.christianbenjaminries.de/scientific/2010-04-13-Ries.pdf* zu finden

Segmenierung

Durch die Segmentierung werden gewünschte Merkmale hervorgehoben und der Rest ignoriert. Ein einfaches Verfahren ist zum Beispiel **Thresholding**. Dabei werden Pixelwerte von einem eindimensionalem Bild, mit einem Schwellwerts verglichen und dadurch die Zugehörigkeit dieses Pixels entschieden, das Ergebnis ist ein Binärbild mit:

$$P(x,y) = \begin{cases} 1 & \text{falls } P(x,y) \text{ gehört zum Objekt} \\ 0 & \text{sonst} \end{cases}$$

Abbildung 2.2: Thresholding

Ein effizienteres Verfahren aber Komplexeres Verfahren ist Efficient Graph-Based Image Segmentation Felzenszwalb and Huttenlocher.

Segmentierungsverfahren werden in fünf in fünf Kategorien unterteilt: Pixelorientiert, Kantenorientiert, Regionenorientiert, Modellbasierte und Texturbasierte Verfahren. Herr Bernd Jähne beschreibt sie ausfürlich in Kapitel 16 seines Buchs "Digital Image Processing".

Merkmalsextraktion

Aus der Resultierenden Bereichen der Segmentierung, werden jetzt Merkamle extrahiert. Merkmale wie Farbe, Form, Größe... werden für jedes Bereich in einem n-

dimensionalen Verktor (Mermalsvektor) zusammengefasst. dieser Vektor wird zu Klassifikator weitergegeben und wird als Basis der Klassifikation dienen.

2.1.2 klassifikation

Die letzte Schritt in der Bilderkennungsprozess ist die Klassifikation. Das ist die Einteilung der Objekten in Vordefinierten Klassen. Es gibt viele Methoden wie Objekte Klassifiziert werden. Zum Beispiel SVM Klassifikatoren durchführen die Klassifikation durch den Ansatz der Search Vector Machines Verfahren. die Idee dahinter ist Daten aus dem Merkmalsvektor nach vordefinierten Informationen über die Objekte zu filtern. Ein anderes Verfahren ist **die künstlische Neuronale Netzwerke**, diese Methode ähmt das Menschliche Gehirn in seinem Lernfähigkeit nach. Neuronale Netzen werden in Kapitel 4.1 (S. 11) erläutert. Ein spezifische Methode der Neuronale Netzen sind Die CNN. Diese haben sehr gute Ergebnisse für die Bilderkennungsaufgaben gezeigt. Im Kapitel 4.2 (S. 18) wird dieses Verfahren erklärt und am ende dieser Arbeit ist eine Implementierung zu finden.

2.2 Verkehrszeichenerkennung

Form und Farbe der Verkehrszeichen sind sorgfältig ausgesucht, um sie leicht erkennbar zu machen. Mit Hilfe der Bilderkennungsverfahren lassen sich Verkehrszeichen auch maschinell finden und identifizieren. Dazu müssen sie in einem Bild lokalisiert werden, viele Faktoren können dies jedoch verhindern, zum Beispiel:

- Tageszeit und Aufnahmequalität beeinflussen Farbe und Helligkeit des Bildes.
- Durch betriebsfremde Oberfläschenveränderungen werden manchmal Verkehrszeichen schwer identifizierbar (z.B. Moose, Aufkleber, Verbiegung...).
- Durch die Kameraaufnahme kann es zu unerwünschten Auswirkungen kommen (z.B. Blur, Lichtreflexe).

- Durch Verdeckung (z.B. durch Schnee, Bäume...).
- Tempolimit auf LKW sollen nicht als Verkehrszeichen erkannt werden (false-positive).

Abbildung 2.3: Beispiel der Identifizierungsproblematik bei Verkehrszeichen

Mit Berücksichtigung der vorgenannte Herausforderungen werden verschiedene Vorgehensweisen entworfen, z.B.

- Detektoren werden durch HOG oder Blob -Verfahren realisiert und Klassifikatoren durch SVM oder oder Neuronale Netze.
- Verkehrszeichen werden mit Selective Search lokalisiert und durch CNN werden Merkmale extrahiert und die Klassifikation durchgeführt.
- CNN für Detektion und Klassifikation.

3 Stand der Technik

Mit der Implementierung von Fahrerassistentsystemen in modernen Autos werden Autohersteller zugleich Verkehrszeichenerkennungssysteme integrieren. Zur Zeit ist dies noch auf Geschwindigkeitshinweise und Überholverbote beschränkt. Mercedes-Benz benutzt Bildverarbeitungsverfahren, um die Straßenschilder zu erkennen. Die Ergebnisse der Erkennung werden mit dem online-Verkehrsinformationsdienst abgeglichen und auf dem Bildschirm angezeigt[1] . Die Firma Bosch hat ihre Fahrassistenz myDriveAssist auch als Smartphone-Applikation zur Verfügung gestellt. Die erkannten Schilder werden zusammen mit der geographischen Position, die durch GPS ermittelt wird, in der Cloud gespeichert und mit Daten von anderen Benutzern verglichen[2]. Andere Hersteller wie BMW oder Volvo (usw...) verwenden ähnliche Verfahren (GPS und Bildverarbeitung).

Der Prozeß der Verkehrszeichenerkennungbesteht aus zwei Phasen: Detektion und Klassifikation.In der Detektionsphase werden die interessierenden Bereiche (Region Of Interest - ROI) ermittelt, das wird häufig mit Segmentierungsmethoden und Form/Farbeerkennungsverfahren durchgeführt. Gefundene ROIs werden dann entweder identifiziert durch ein Klassifikationsverfahren oder verworfen. In der Klassifikationsphase wird häufig Support Vektor Maschine (SVM) oder neuronale Netze verwendet.

Konvolutionale Neuronale Netzwerke (CNN) haben sind zur Zeit die besten Klassifikatoren. Bei einem Wettbewerb des German-Traffic-Sign-Recognition-Benchmark-

[1] Quelle: *http://techcenter.mercedes-benz.com/de_AT/traffic_sign_assist/detail.html* – März 2016
[2] Quelle *http://mydriveassist.bosch.com/mydriveassist* – März 2016

Team (GTSRB)[3] zur Verkehrszeichenerkennung haben sich folgende drei beste Ergebnisse gezeigt.

- Zaklouta et al. (2011) nutzen Random Forests mit HOG-Features zur Klassifikation und erreichen eine Klassifikationsrate von 96.14%
- Sermanet and LeCun (2011) haben durch den Einsatz von Multi-Scale Convolutional Networks eine Klassifikationsrate von 98.31% erreicht.
- Das beste Ergebnis dieses Benchmarks wurde von Cireşan et al. (2012) mittels einer Kombination von CNN und eines mit HOG-Features trainierten Multilayer Perceptrons (MZP) erreicht. Auf diese Weise wird eine Klassifikationsrate von 99.46% erzielt.

Es ist darauf hinzuweisen, dass die menschliche Erkennungsrate nur 98.84%[4] ist, was bedeutet, dass Verfahren wie Cireşan et al. (2012) schon die menschliche Leistung übetroffen haben.

Mathias et al. (2013) haben gezeigt, dass mit Verwendung der gleichen Verfahren für Digits- und Gesichtserkennung, eine Erkennungsrate von 95% bis 99% erreicht werden kann. Ihre beste Ergebnis haben sie erreicht durch Verwendung einer HOG-Variante für Detektion zusammen mit einem Sparse Wright et al. (2009) Klassifikator.

Es gibt Zahlreiche angepasste Varianten der CNN, die die Erkennung stark verbessern, zum Beispiel Zeng et al. (2015) zeigen, dass der Einsatz eines Extreme Learning Classifier (ELM - ein andere Art von Neuronale Netzen) anstelle des Perzeptron-Layers einem CNN zu einer deutlichen Steigerung der Erkennungsrate führt. Jaderberg et al. (2015) haben das Verfahren Spatial Transformer vorgestellt. Dieses kann zusätzlich in ein CNN integriert werden, um die räumliche Invarianz zu verbessern.

[3] Webseite: *http://benchmark.ini.rub.de/?section=gtsrb&subsection=results*

[4] Dieses Ergebnis ist entstanden durch ein Experiment mit 32 Personen, die ähnliche Aufgaben wie im Wettbewerb lösen sollten; mehr Details in Stallkamp et al. (2012) Kapitel 6.

Wu et al. (2013) benutzen Konvolutionale Neuronale Netzwerke[5] sowohl für Detektion als auch für Klassifikation. Die Idee hinter diesem Verfahren ist es, dass während der Konvolutionsphase die resultierenden Maps durch vordefinierte Filter aussortiert werden, so dass nur relevante Regionen weiter gelernt bzw. klassifiziert werden.

Mit Sermanet et al. (2013) konnten die Autoren den "Localisation and classification task" der ILSVRC2013[6] gewinnen. Das Verfahren ist ein CNN mit multi-scale und sliding window Verfahren.

[5] Das Konvolutionale Neuronale Netzwerk Verfahren ist in 4.2 (S. 18) beschrieben

[6] ILSVRC: ImageNet Large Scale Visual Recognition Challenge ist ein Wetbewerb für Algorithmen zur Objekterkennung und Bildklassifikation. Webseite: *http://image-net.org/challenges/LSVRC/2013/index*

4 Konvolutionale Neuronale Netze

In diesem Kapitel werden Neuronale Netzwerke und Konvolutionale Neuronale Netzwerke im Allgemeinen erklärt. Danach werden verschiedene CNN Konzepte gezeigt.

4.1 Neuronale Netze

Neuronale Netze sind Netzwerke aus Nervenzellen im Gehirn von Menschen oder Tieren. Künstliche Neuronale Netze in Informatik-Anwendungen sind, wie der Name andeutet, eine Simulation von neuronalen Netzen. Wie das biologische Netz, wird das künstliche Netz auf ein spezielles Verhalten hin trainiert und ist anschließend in der Lage, sowohl auf Situationen aus dem Training als auch auf neue Situationen zu reagieren. Das biologische Netz besteht aus Milliarden von Neuronen, jedes Neuron hat eine Aufgabe, die meistens als Aktivierungsfunkion simuliert werden kann. Der Ausgabewert dieser Funktion wird weiter an die nächsten Neuronen als Eingabewert gereicht. Die Künstlichen Neuronalen Netze sind genau so aufgebaut, eine große Anzahl von Funktionen, die die Arbeit eines biologischen Neuron simulieren, werden miteinander verbunden und daraus ein künstliches neuronales Netzwerk erstellt. Abhängig von der Funktion, der Anzahl der Neuronen, die Struktur in die die Neuronen eingeteilt sind und der Art und Weise wie sie miteinander kommunizieren werden unterschiedliche Netzen gebaut. In den folgenden Abschnitten wird jede dieser Komponenten erklärt.

4.1.1 Einzelne Neuronen:

Wie vorher erwähnt hat das Neuron nur die eine Aufgabe die Aktivierungsfunktion zu berechnen und das Ergebnis zurückzugeben. Eine grundlegende Klasse von Neuronen ist das Perzeptron, seine Funktion hat eine oder mehrere Eingaben und eine Ausgabe. Als erstes werden die Eingänge *(x1,x2,...,xn)* mit Gewichten *(w1,w1,...,wn)* multipliziert, und zusammen mit einem Schwellwert (*bias* θ) summiert, das Ergebnis wird dann von einer Aktivierungsfunktion ausgewertet.

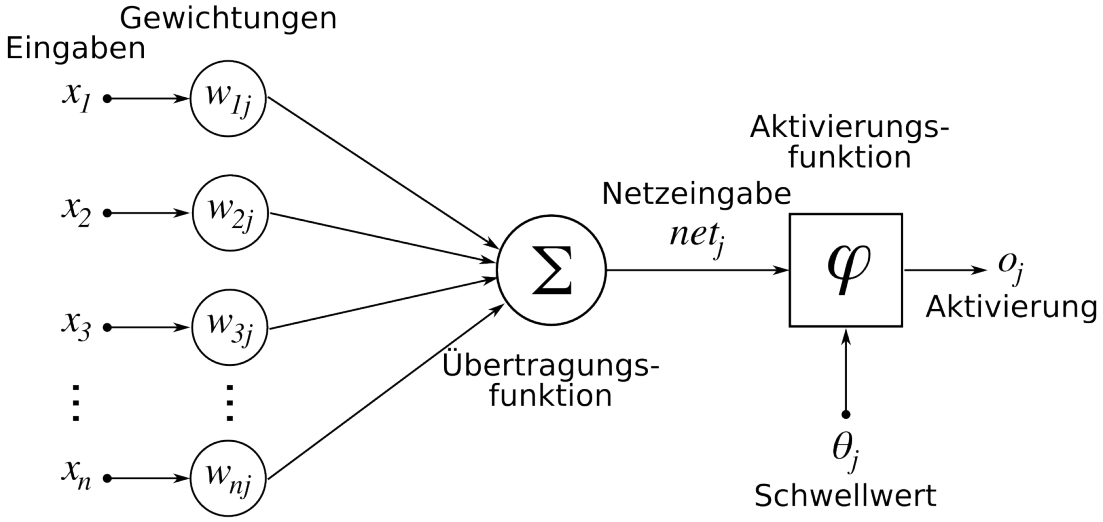


Abbildung 4.1: Aufbau eins Neurons -Quelle Wikipedia: Backproagation

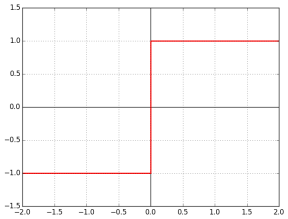
Die Mathematische Formel ist dann:

$$o_j = \phi(\sum_{i=1}^{n} x_i w_i + \theta)$$

Die Aktivierungsfunktion ϕ kann je nach Aufgabe des Neurons ausgewählt werden. Häufig werden folgende Funktionen benutzt:

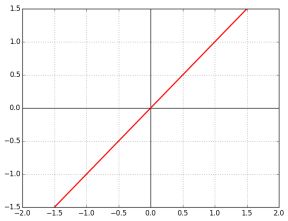
- *Stufenfunktion*:

$$\phi(x) = \begin{cases} 1 & x >= 0, \\ 0 & \text{sonst} \end{cases}$$



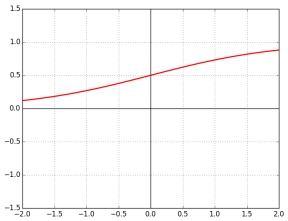
- *Linearfunktion*:

$$\phi(x) = x$$



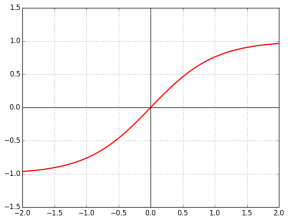
- Logistische Funktion

$$\phi(x) = \frac{1}{1+e^{-\alpha x}}$$



- Tangens Hyperbolicus

$$\phi(x) = tanh(x) = \frac{1-e^{-}x}{1+e^{-}x}$$



4.1.2 Neuronale Netze

Ein Neuronales Netz besteht aus mehreren Schichten von Neuronen. Es gibt drei Arten von Schichten: die **Eingabeschicht** (Input Layer) ist direkt mit den Eingabedaten

verbunden und versorgt das Netz mit Daten. Die **Ausgabeschicht** (Output Layer), teilt die gerechneten Ergebnisse in Klassen ein. Die **innere Schicht** (Hidden Layer) übernimmt weiteregehende Berechnungen bezüglich des Lernens und des Klassifizierens. Für einfachere Aufgaben ist die innere Schicht nicht nötig, für komplexere verfügt das Netz sogar über mehrere innere Schichten mit jeweils verschiedenen Aufgaben.

Die Schichten sind nacheinander angeordnet, so dass die Ausgabe der Neuronen einer Schicht als Eingabe der nächsten Schicht von Neuronen dienen.

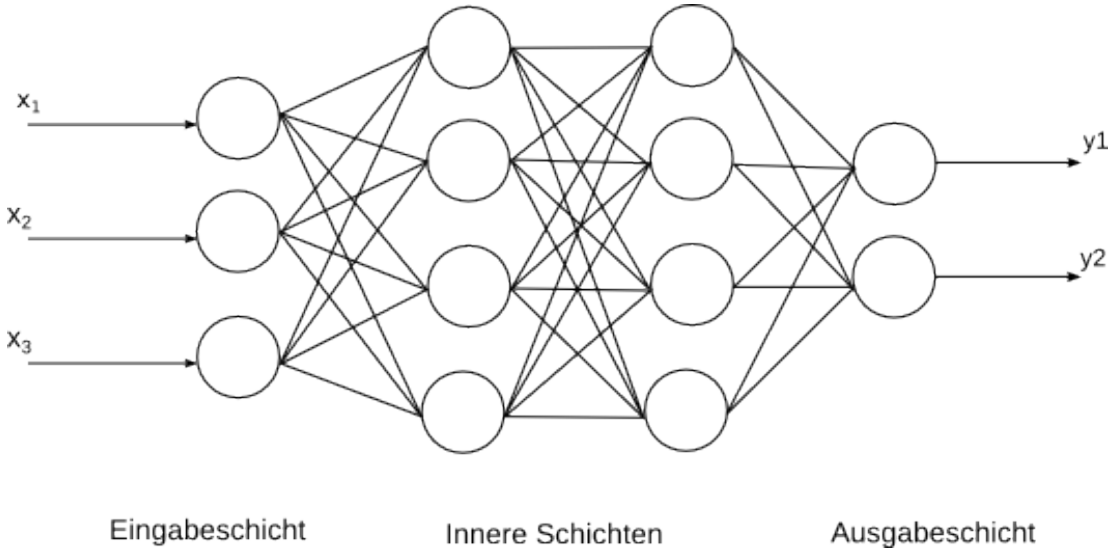


Abbildung 4.2: Beispiel eines Neuronalen Netzes mit zwei inneren Schichten

4.1.3 Lernen

Bei den Künstlichen Neuronalen Netzen sind zwei Arten von Lernen zu unterscheiden: **Überwachtes Lernen**: das Zielergebnis ist bekannt, und das System soll durch Versuch und Irrtum die Aufgabe zu lösen lernen. **Unüberwachtes Lernen**: das System soll selbst durch eine große Menge von Eingaben das Problemlösen lernen. Diese Arbeit beschäftigt sich mit dem überwachten Lernen.

Einzelne Neuronen: Das Neuron berechnet die Ergebnisse mit zufälligen Gewichten, das Ergebnis wird mit dem Zielwert verglichen. Die Abweichung zwischen Ist-Wert und Soll-Wert dient als Steuersignal, um den Gewichte zu korrigieren. Dieser Vorgang wird solange wiederholt, bis das Neuron akzeptable Ergebnisse zurückgibt.

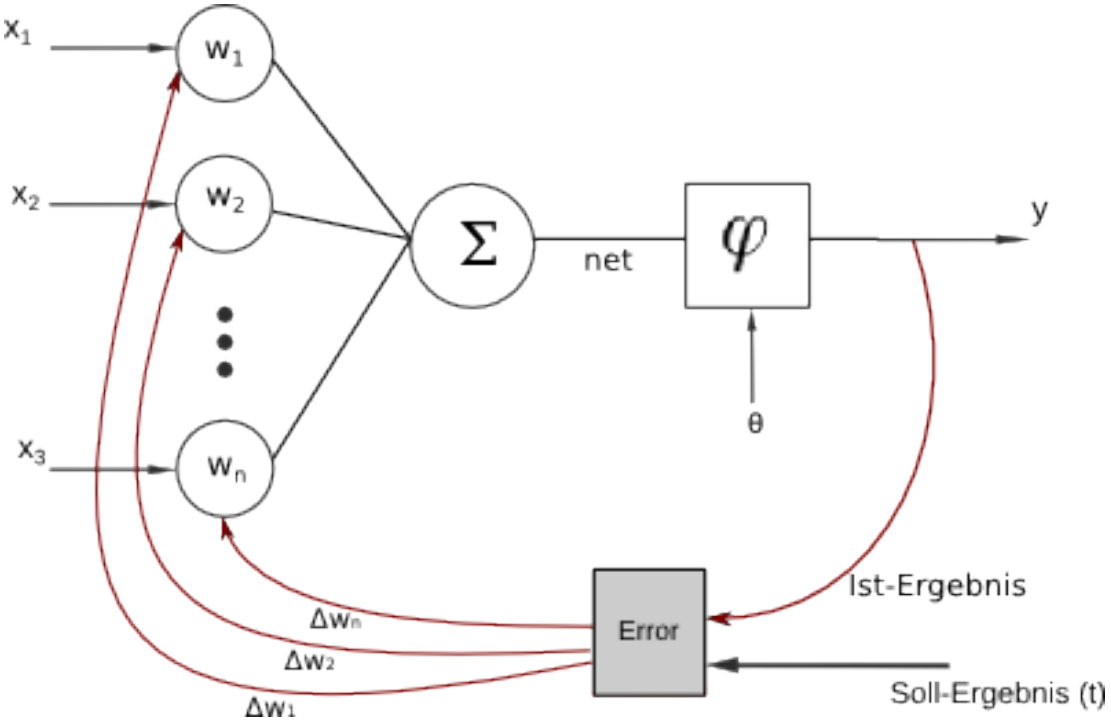


Abbildung 4.3: Lernen eines Neurons

Der Error Block (Abbildung **??**) ist eine externe Funktion, um die Gewichte neu zu kalibrieren. Die Änderung der Gewichte erfolgt nach folgender Formel:

$$\begin{aligned} w_i^{neu} &= w_i^{alt} + \Delta w_i \\ \Delta w_i &= \eta \phi'(net_j)(t-y)x_i \end{aligned} \tag{4.1}$$

w_i^{neu} neuer Wert des Gewichts.
w_i^{alt} alter Wert des Gewichts.
Δw_i Gewichtsänderung.

η Schrittweitenfaktor (learning rate), bestimmt die Größe der Schritte mit denen das Gewicht verändert wird.
t Soll-Wert des Neurons.
y Ist-Wert Ausgabe des Neurons.
x_i Eingangssignal am Eingang i.

Durch η kann die Grösse der Lernschritte beeinflusst werden, ein kleines η führt zu einem sehr langsamen Lernprozess, eine grosse η kann zu ungenauen Ergebnissen führen.

Mehrere Schichten: Das Korrigieren von Gewichten durch die Ausgabe heißt Backpropagation. Für Netze mit mehreren inneren Schichten ist die Soll-Ausgabe jedes einzelnen Neurons nicht bekannt. Deswegen wird ein Backpropagations-Algorithmus verwendet, um alle Gewichte in Form einer Matrix zu berechnen.

- Gewichte der Neuronen in der Ausgabeschicht werden direkt vom Soll-Ergebnis des Netzes beeinflusst.
- Gewichte der Neuronen aus der inneren Schichten werden von dem berechneten Fehler der nachfolgenden Neuronen und deren Gewichten beeinflusst.

$$\begin{aligned} w_{ij}^{neu} &= w_{ij}^{alt} + \Delta w_{ij} \\ \Delta w_{ij} &= \eta \delta_j x_i \end{aligned} \tag{4.2}$$

$$\delta_j = \begin{cases} \phi'(net_j)(t_j - o_j) & für\ Ausgabeschicht\ Neuronen \\ \phi'(net_j) \sum_k (\delta_k . w_{jk}) & für\ innereSchicht\ Neuronen \end{cases}$$

Δw_{ij} Gewichtsänderung der Neuron j.
η Schrittweitenfaktor (learning rate), bestimmt die Größe der Schritte mit denen das Gewicht verändert wird.
δ_j das Fehlersignal des Neurons j

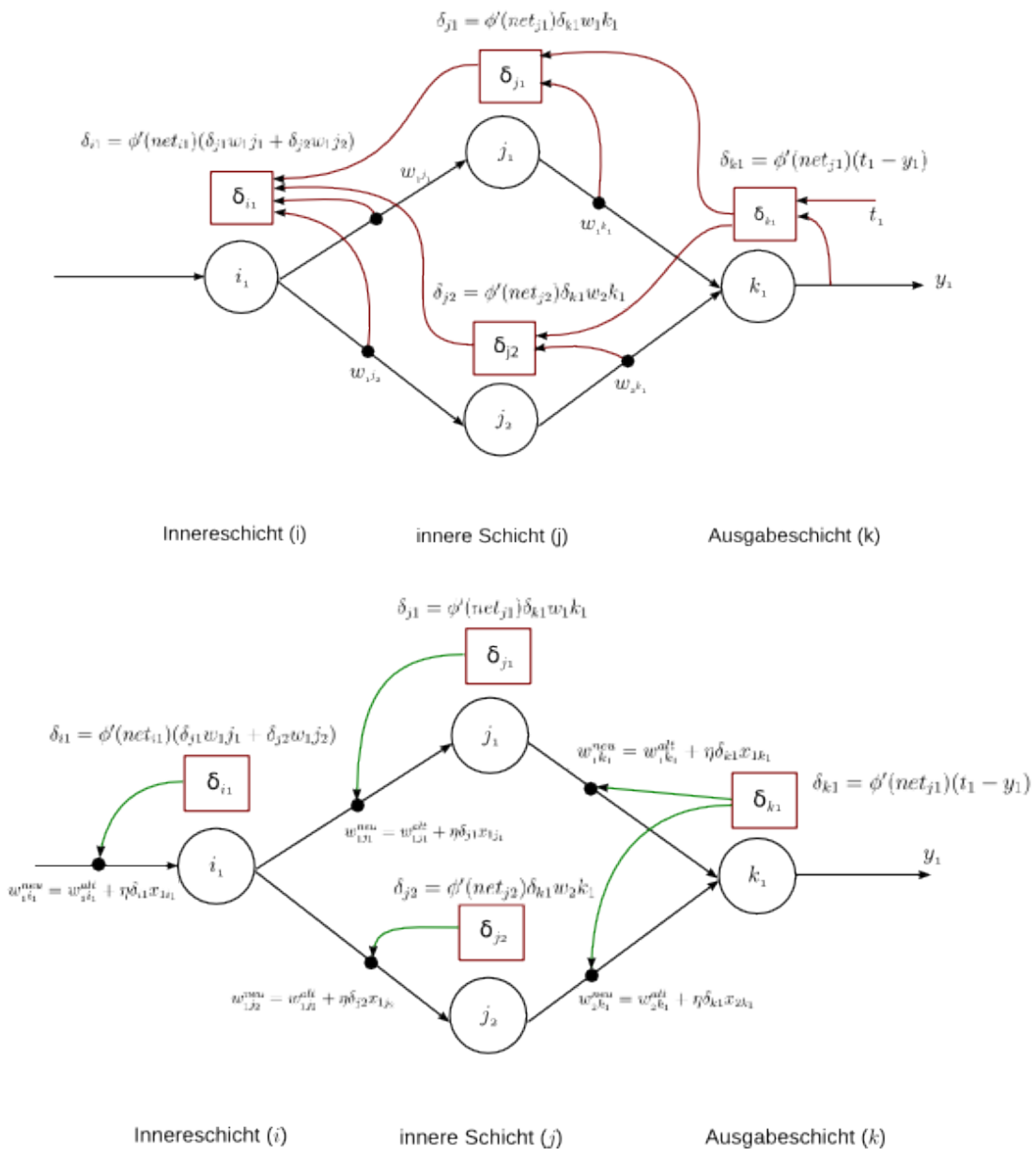


Abbildung 4.4: Die Gewichtenberechnung bei der Backpropagtion

x_i die Ausgabe des Neuron i = Eingabe des Neuron j

t_j die Soll-Ausgabe des Ausgabeneurons j

y_j die Ist-Ausgabe des Ausgabeneurons j

k der Index der nachfolgenden Neuronen von j

4.1.4 Kalassifizieren:

Nachdem das Training beendet ist, werden die Gewichte gespeichert. Das Netz ist hat den Lernprozess abgeschlossen, und ist nun bereit für das Klassifizieren. Die Klassifikation ist das Einführen von neuen Daten in das durch den vorangegangenen Lernprozess vorbereitete Netz. Im Prinzip die Klassifikation erfolgt durch das selbe Netz aber ohne Backpropagations-Phase.

4.2 Konvolutionale Neuronale Netzwerke

Konvolutionale Neuronale Netzwerke (Faltungsnetzwerke - CNN) sind Neuronale Netze mit zusätzlichen Schichten, die mehrere Faltungen der Eingabedaten durchführen. Dieses Verfahren ist von der Bildverarbeitung inspiriert und liefert sehr erfolgreiche Ergebnisse insbesondere bei Aufgaben der Objektklassifizierung.

Ein CNN besteht hauptsächlich aus drei Arten von Schichten:

- Konvolution-Schicht
- Pooling-Schicht
- Perzeptron-Schicht (MLP)

Die Konvolutions-Schicht und die Pooling-Schicht arbeiten paarweise, um relevante Daten aus der Eingabe zu extrahieren. Diese Daten werden dann durch die Perzeptron-Schicht klassifiziert.

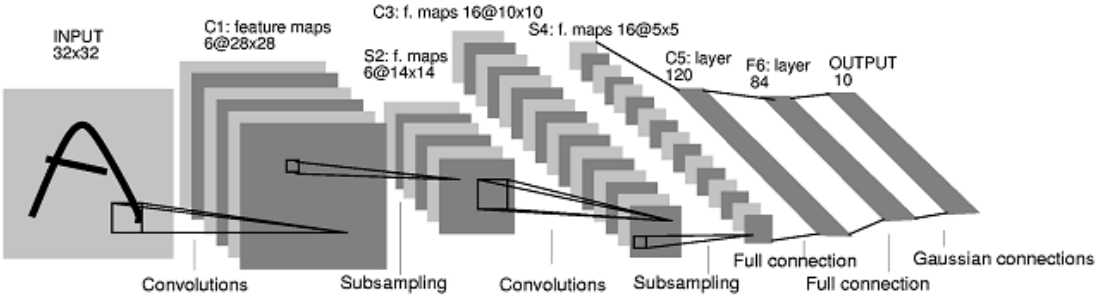


Abbildung 4.5: Struktur eines Konvolutionsnetzes

Konvolution Schicht:

Die Neuronen in dieser Schicht führen Faltungsfunktionen aus. Dies ist ein Verfahren aus dem Bereich der Signalbearbeitung und wird viel in der Bildverarbeitung benutzt. Eine Faltung ist die Anwendung einer Maske (auch Kernel genannt) auf eine Matrix. Üblicherweise wird ein quadratisches Kernel gemäß folgender Gleichung benutzt:

Faltung g einer Matrix M mit einer Maske h:

$$g = M * h \tag{4.3}$$

$$g_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} h(v,n) M(i+v-a, j+u-a) \tag{4.4}$$

g das resultierende Bild
M das originale Bild
h die Faltungsmask
a die Koordinate des Mittelpunkts in der quadratischen Faltungsmatrix

Das Beipiel 4.6 (S. 20) zeigt die erste Iteration der Faltungsfunktion. Für die Filterung des kompletten Bildes wird die Maske über alle Bildpositionen platziert, dabei muss ein Schritt Faktor (**stride**) definiert werden. Beispiel 4.7 (S. 20) zeigt die komplette Faltung. Abhängig von der Kernelgröße besteht ein Bildrand von mindestens einer Pixelbreite. Diesen Pixeln am Rande kann kein Wert zugewiesen werden. Dieser Rand

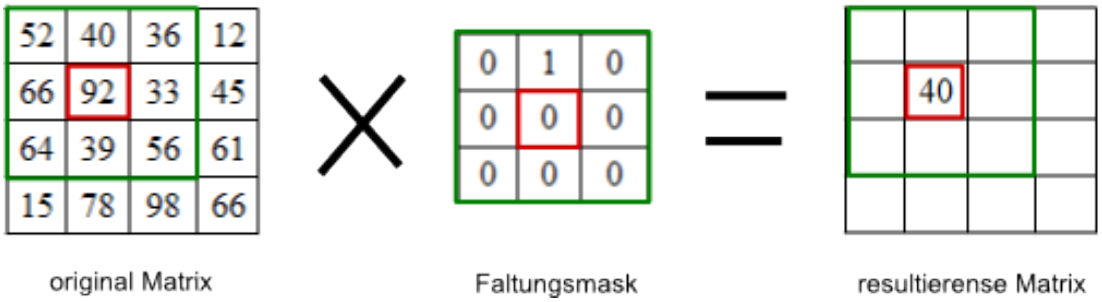


Abbildung 4.6: Beispiel einer Konvolution

kann mit verschiedenen Methoden behandelt werden, z.B kann der Rand mit Nullen oder mit den Werten benachbarter Pixel ausgefüllt werden (padding).

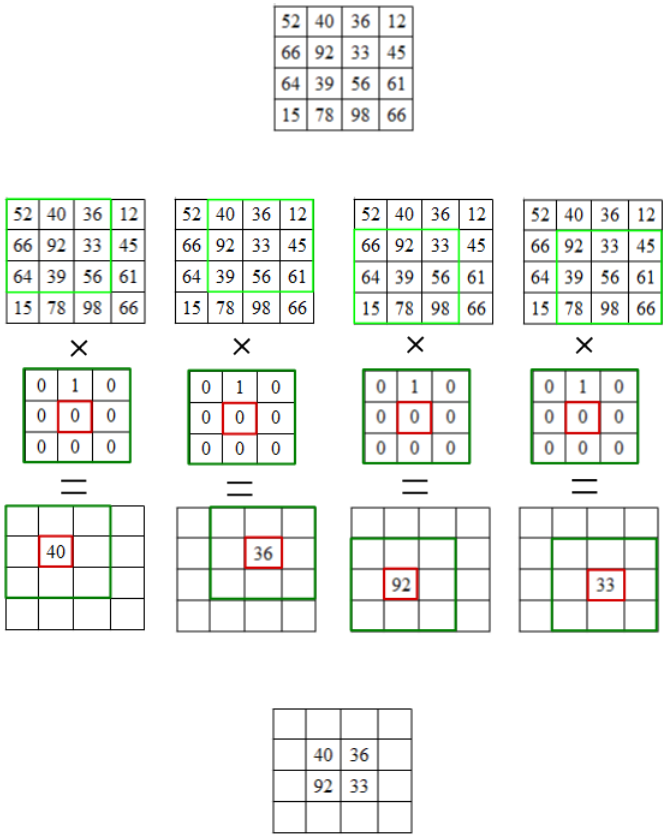


Abbildung 4.7: Beispiel einer kompletten Konvolution

Abbildung 4.8 (S. 21) zeigt verschiedene Effekte, die durch eine Konvolution entstehen können. In den Faltungsnetzen sind die Masken nicht vorgegeben, sondern werden mit zufälligen Zahlen initialisiert und durch das Training werden daraus passende Masken erzeugt.

Zusammengefasst besteht die Konvolutionsschicht aus einer Vernetzung von Faltungsoperationen, die unterschiedliche Masken verwenden. Durch dieses Verfahren entstehen mehrere Bildvarianten mit verschiedenen Effekten. Diese dienen als Eingabedaten für das Klassifikationsnetz.

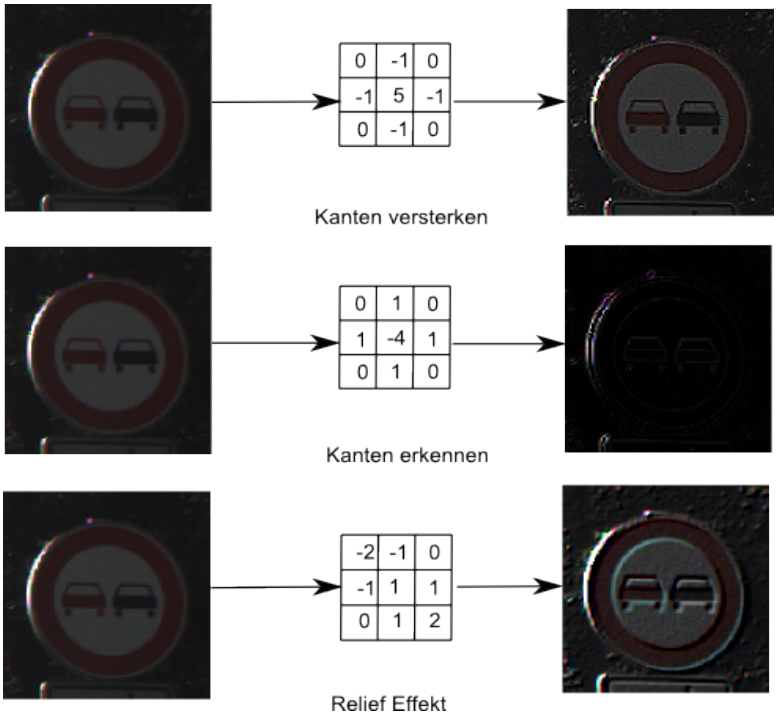


Abbildung 4.8: Beispiele von Konvulution Effekte

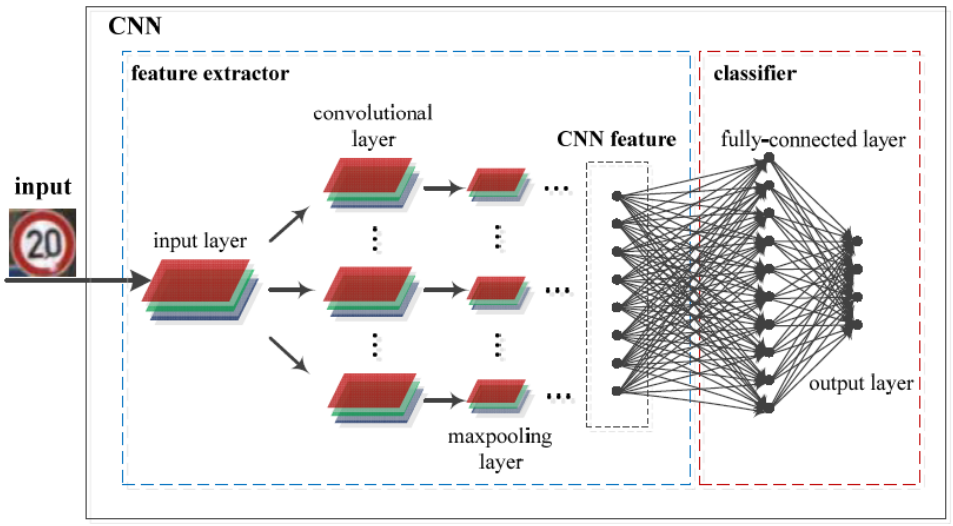


Abbildung 4.9: Struktur der CNN -Quelle: Zeng et al. (2015)

Pooling Schicht:

Die Pooling Schicht ist üblicherweise nach der Faltungsschicht einzufügen. Ihre Aufgabe ist es, die räumliche Auflösung der Eingaben zu reduzieren, um die Anzahl der Gewichte und damit den Rechenaufwand zu reduzieren.

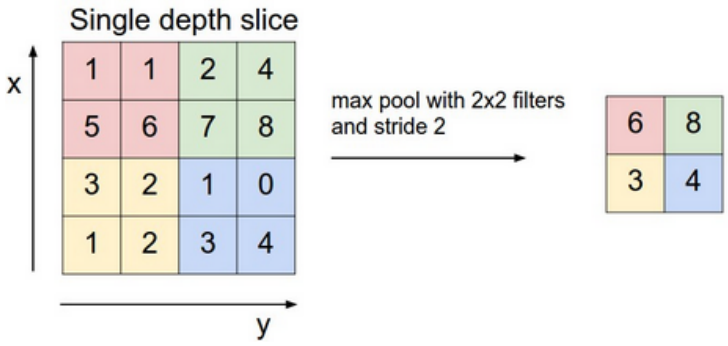


Abbildung 4.10: Arbeitsweise des Max-pooling Verfahrens

In der Regel wird das maximum pooling Verfahren mit einem Filter der Grösse 2x2 ohne Überlappung verwendet. Das bedeutet, dass das Bild in jeder Dimension um

den Faktor 2 verkleinert wird, dabei wird jede 2x2 Teilmatrix durch ihren maximalen Eintrag ersetzt. Abbildung 4.10 (S. 22) zeigt Ein Beispiel[1].

Perzeptron Schicht:

Diese Schicht ist aus Perzeptronen aufgebaut (siehe 4.1 (S. 11)). Ihre Aufgabe ist es, die resultierenden Features zu klassifizieren.

4.3 Verkehrszeichenerkennung mit CNN

Die Verwendung von CNN im Bereich der Verkehrszeichenerkennung kommt in zwei Arten vor. Zum Einen können sie als Klassifikatoren benutzt werden. Der Prozessablauf in diesem Fall fängt mit der Verwendung von Segmentierungs- und Lokalisationsverfahren an, um die Verkehrszeichen im Bild zu finden. Die Verkehrszeichen werden dann als Teilbilder zum Klassifikator geschickt. Der Klassifikator ist ein CNN, das die gefundenen Bilder sortiert. Zum Anderen können die CNN zur Detektion und Klassifikation verwendet werden. Im Folgenden werden diese beiden Techniken anhand von Beispielen erklärt.

4.3.1 CNN als Klassifikatoren

Im Folgenden soll das von Cireşan et al. (2012) benutzte Netz (Abbildung 4.11 (S. 24)) betrachtet werden:

- die Eingabeschicht ($Layer\ 0$) besteht aus drei Maps, das sind die drei Farbkanäle (z.B. RGB).
- Für die erste Konvolutionale Schicht ($Layer\ 1$) ist ein Kernel von 7x7 Pixeln und 100 Feature Maps ausgewählt worden. Diese Parameter sind frei wählbar, dabei

[1] Bildquelle: *http://cs231n.github.io/convolutional-networks/#pool*

Layer	Type	# maps & neurons	kernel
0	input	3 maps of 48x48 neurons	
1	convolutional	100 maps of 42x42 neurons	7x7
2	max pooling	100 maps of 21x21 neurons	2x2
3	convolutional	150 maps of 18x18 neurons	4x4
4	max pooling	150 maps of 9x9 neurons	2x2
5	convolutional	250 maps of 6x6 neurons	4x4
6	max pooling	250 maps of 3x3 neurons	2x2
7	fully connected	300 neurons	1x1
8	fully connected	43 neurons	1x1

Abbildung 4.11: Model eines Konvolutionale Neuronale Netzwerk

sollen die Erkennungsrate und die Bearbeitungszeit optimiert werden. Die Anzahl der Neuronen (42x42) entspricht der Anzahl der Pixel. Bei der Anwendung einer Konvolution mit einem (7x7)-Kernel entstehen Ränder von 3 Pixeln an jeder Seite (daher 42x42).

- Die nächste Schicht ist eine Max-pooling mit einem (2x2)-Kernel und einem *stride* von 2. Die Anzahl der Maps ändert sich nicht, jedoch die Anzahl der Pixel. Somit halbiert sich die Anzahl der Neuronen.
- Die nächsten vier Schichten sind ähnlich zu den beiden vorherigen.
- Schicht 7 und 8 sind ein Perzeptron Netz. Die Anzahl der Neuronen (43) in der Ausgabeschicht ($Layer\ 8$) entspricht der Anzahl der Klassen.

4.3.2 CNN als Detektoren

Als Beispiel von Detektion mit CNN wird hier zwei Verfahren gezeigt:

Die in Sermanet et al., 2013 beschriebene Methode, benutzt eine Sliding Window als Eingabe der Neuronale Netze um ROIs zu finden. Abbildung 4.12 (S. 25) zeigt das verwendete Modell[2].

» input 3x221x221
» stage 1: convo: 7×7 stride 2×2; ReLU; maxpool: 3×3 stride 3×3; output (layer 3): 96x36x36
» stage 2: convo: 7×7 stride 1×1; ReLU; maxpool: 2×2 stride 2×2; output (layer 6): 256x15x15
» stage 3: convo: 3×3 stride 1×1 0-padded; ReLU; output (layer 9) 512x15x15
» stage 4: convo: 3×3 stride 1×1 0-padded; ReLU; output (layer 12) 512x15x15
» stage 5: convo: 3×3 stride 1×1 0-padded; ReLU; output (layer 15) 1024x15x15
» stage 6: convo: 3×3 stride 1×1 0-padded; ReLU; maxpool: 3×3 stride 3×3; output (layer 19) 1024x5x5
» stage 7: convo: 5×5 stride 1×1; ReLU; output (layer 21) 4096x1x1
» stage 8: full; ReLU; output (layer 23) 4096x1x1
» stage 9: full; output (layer 24) 1000x1x1
» output stage: softmax; output (layer 25) 1000x1x1

Abbildung 4.12: Overfeat Neuronale Netze Modell

Eine andere Methode ist die R-CNN (Region Proposals and Convolutional Neural Network) Girshick et al. (2013) Die Methode benutzt Region Proposals Verfahren anstatt sliding window. Die Funktionsweise ist wie in Abbildung 4.13 (S. 25) dargestellt[3]:

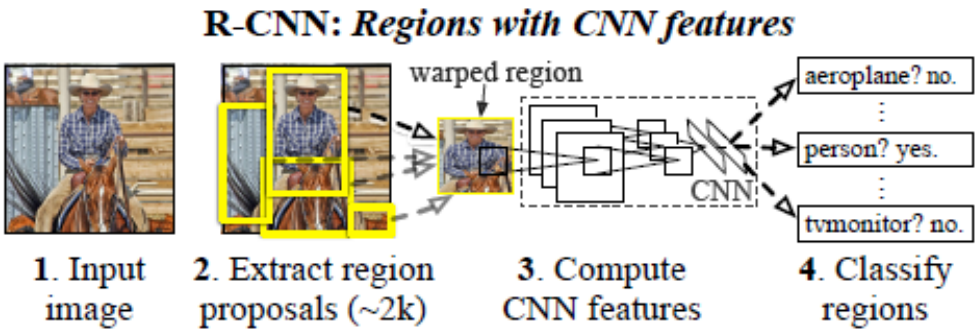


Abbildung 4.13: Objekterkennung mit R-CNN

- Region Proposals extrahieren, dafür gibt es viele Methoden (z.B. selective search Uijlings et al. (2013)).

[2]Quelle: *http://cilvr.nyu.edu/doku.php?id=software:overfeat:start* die Webseite bietet auch eine Implementierung der Verfahren und einen Vortrainierten Netz mist Daten Der ILSVRV2013.
[3]Quelle: Girshick et al. (2013)

- Features für jede Region durch CNN erzeugen.
- Die erzeugten Features durch einen SVM klassifizieren.

5 Realisierung

In Diesem Kapitel werden drei CNN Netze implementiert und miteinander verglichen. Vorher werden zuerst die verwendete mitteln vorgestellt.

5.1 Verwendete Tools

5.1.1 Dataset

Diese Arbeit benutzt Verkehrsbilder aus den Öffentlichen Datasets des GTSRB[1]. Sie bietet zwei Arten von Datasets.

- The German Traffic Sign Recognition Benchmark (GTSRB) Stallkamp et al. (2012): besteht aus mehr als 50.000 Bilder von ausgeschnittene einzelne Verkehrsschilder, etwa 40.000 davon sind in 43 Klassen zugeteilt. Diese Bilder sind für das Training geeignet. 12.569 sind für das Test. Die Verkehrsbilder sind aus unterschiedlichen Perspektiven und unter unterschiedliche Belichtungsverhältnissen aufgenommen, viele Bilder haben Störungen die im echten Leben auftreten können. Abbildung 5.1 (S. 28) zeigt Beispiele von allen Klassen der Dataset.

- German Traffic Sign Detection Benchmark (GTSDB) Houben et al. (2013): enthält 900 (600 Training und 300 Validierung) Bilder von echten Leben Situationen, sowie sie von, in einem Auto, montierten Kamera aufgenommen werden. Dieses Database ist für die Detektion gedacht.

[1]Homepage: *http://benchmark.ini.rub.de/?section=home&subsection=news*

Abbildung 5.1: Beispielbilder aus der GTSRB Dataset

Abbildung 5.2: Beispielbilder aus der GTSTB Dataset Bildquelle: GTSB webseite

In diese Arbeit wird das erste Dataset für Training und Klassifikation benutzt. Die Zweite Dataset wird nur für Detektion verwendet.

5.1.2 Bildverarbeitung

Die opencv[2] Bradski (2000) wurde für Bildverarbeitungsverfahren benutzt. Opencv (Open Source Computer Vision Library) ist eine freie Bibliothek für Bildverarbeitung

[2]Homepage: *http://opencv.org*

und maschinelles Sehen Algorithmus. Es enthält einfache Funktionen wie Bilder laden, speichern und Farbraum ändern, sowie komplexe Algorithmen, zum Beispiel für Filterung oder Detektion. Opencv ist ursprünglich in C/C++ geschrieben, hat aber auch Python, Java und MATLAB Schnittstellen, und unterstützt Windows, Linux, Android und Mac OS.

5.1.3 Neuronale Netzwerke

Es gibt viele Bibliotheken und Frameworks die Neuronale Netzwerke implementieren. In dieser Arbeit wird das Caffe-Framwork verwendet. Caffe Jia et al. (2014) ist ein open source modular Deep-Learning-Framework, entwickelt von Berkeley Vision and Learning Center (BVLC), und mit Hilfe der Community weiterentwickelt. Caffe bietet eine einfache und modifizierbare Framework für Training der CNN und andere Deep-Learning Algorithmen. Es ist geschrieben in C++ und hat Python und MATLAB Schnittstellen, Caffe unterstützt berechnungen mit CPU und mit GPU durch NVIDIA CUDA Technologie, was die Trainingzeiten deutlich verbessert.

Arbeitsweise mit Caffe

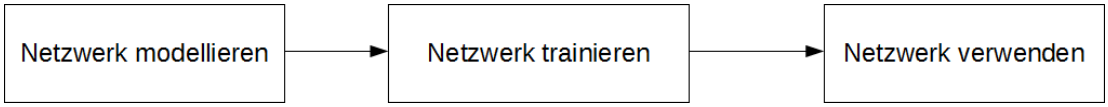


Abbildung 5.3: Caffe Arbeitsweise

In der Netzwerk Modellierungsphase werden Schichten (Layers) und ihre Parametern definiert. Caffe benutzt eine vertikale Orientierung, wie in Abbildung 5.4 gezeigt ist. Die Untere Stein repräsentiert die Eingabeschicht. Daten werden an den nächsten Schicht geschickt (in Beispiel Konvolutionschicht). Die Informationen in Caffe sind

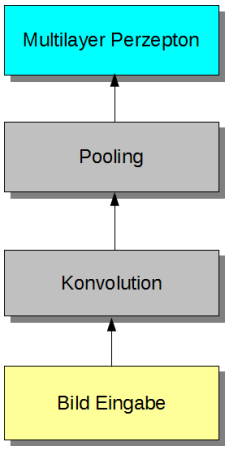


Abbildung 5.4: Aufbau des Netzes

gespeichert und ausgetauscht in Blobs[3]. Schichten Definition sind in Protobuf [4] Model Format geschrieben.

Aufbau einer Schicht

Die Abbildung 5.5 (S. 31) zeigt ein Beispiel einer Konvolutionsschicht.

name	name der Schicht.
type	type der Schicht (z.B Convolution, pooling...).
bottom	die vorherige Schicht(en) falls vorhanden.
top	die aktuelle Oberste(n) Schicht(en).

Mehr ausführliche Informationen über die Schichten und ihere Parametrn sind in Caffe[5] Webseite zu finden.

[3] Blobs sind n-dimensionale Arrays benutzt als einheitliche Speicherschnittstelle um Daten (wie Bilder und Parametern) zu speichern und kommunizieren, ihre Verwendung sichert auch die Synchronisationsfähigkeit zwischen der CPU und der GPU.

[4] Protocol Buffer (protobuf) ist ein von Google entwickelte Datenformat. Es wird in Caffe benutzt um Netzwerke, Schichten und Konfigurationsdateien zu definieren.

[5] cafee webseite Layer Catalogue: *http://caffe.berkeleyvision.org/tutorial/layers.html*

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Abbildung 5.5: Beispiel einer Konvolutionschicht -Quelle Caffe Webseite

Mit Hilfe dieser Datei wird eine Konvolution Klasse generiert, und mit gegebene Parameter initialisiert. Caffe kommt mit viele vordefinierten Schichten die üblicherweise in Deep-Learning Algorithmen verwendet werden, es gibt sonnst die möglichkeit selbst eine Schicht zu implementieren und in Caffe zu integrieren. In dieser Arbeit wird keine Schicht selbst geschrieben, sondern die vorhandene benutzt.

Netzwerk Trainieren

Das gesamte Netz (alle Schichten) werden in den Protobuf Datei geschrieben, und damit ist das Model bereit zum Training. Um die Trainingsphase zu starten wird ein Solver benötigt. Das ist ein in protobuf geschriebene Konfigurationsdatei, wo der Trainingsprozess definiert wird und die notwendige Parametern (Lernrate, Training Iterationen...) festgelegt sind[6]. Beispiel 5.6 (S. 32) zeigt den benutzten Solver zum Training der MNIST[7] Dataset.

[6] *http://caffe.berkeleyvision.org/tutorial/solver.html*

[7] MNIST ist eine Dataset von Handgeschriebene Zahlen. Caffe bietet ein Beispiel für MNIST Training mit Hilfe von Lenet Verfahren.

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"

# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100

# Carry out testing every 500 training iterations.
test_interval: 500

# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005

# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75

# Display every 100 iterations
display: 100

# The maximum number of iterations
max_iter: 10000

# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"

# solver mode: CPU or GPU
solver_mode: GPU
```

Abbildung 5.6: Beispiel eines Solver -Quelle: Caffe Framework

Mit dem Model und dem Solver vorhanden, das Training kann z.B. mit dem Batch 5.1 gestartet werden.

```
1 #!/usr/bin/env sh
2
3 ./build/tools/caffe train \
4         --solver=examples/tsr/solver.prototxt \
5         --log_dir=examples/tsr/
```

Listing 5.1: Script zum Starten der Trainig

Während des Trainings wird der Prozess auf die Konsole gezeigt, da kann die learning rate, Error (loss), und Accuracy überwacht werden.

```
I0320 17:32:16.975612  5551 solver.cpp:346] Iteration 15000, Testing net (#0)
I0320 17:32:31.859738  5551 solver.cpp:414]     Test net output #0: accuracy = 0.959166
I0320 17:32:31.859786  5551 solver.cpp:414]     Test net output #1: loss = 0.204403 (* 1 = 0.204403 loss)
I0320 17:32:31.902731  5551 solver.cpp:242] Iteration 15000, loss = 0.0362016
I0320 17:32:31.902762  5551 solver.cpp:258]     Train net output #0: loss = 0.0362016 (* 1 = 0.0362016 loss)
I0320 17:32:31.902783  5551 solver.cpp:571] Iteration 15000, lr = 5.02973e-05
I0320 17:32:40.305827  5551 solver.cpp:242] Iteration 15100, loss = 0.329318
I0320 17:32:40.305874  5551 solver.cpp:258]     Train net output #0: loss = 0.329318 (* 1 = 0.329318 loss)
I0320 17:32:40.305893  5551 solver.cpp:571] Iteration 15100, lr = 5.0147e-05
I0320 17:32:48.707928  5551 solver.cpp:242] Iteration 15200, loss = 0.40382
I0320 17:32:48.708101  5551 solver.cpp:258]     Train net output #0: loss = 0.40382 (* 1 = 0.40382 loss)
I0320 17:32:48.708119  5551 solver.cpp:571] Iteration 15200, lr = 4.99976e-05
I0320 17:32:57.113178  5551 solver.cpp:242] Iteration 15300, loss = 0.0676279
I0320 17:32:57.113227  5551 solver.cpp:258]     Train net output #0: loss = 0.0676279 (* 1 = 0.0676279 loss)
I0320 17:32:57.113243  5551 solver.cpp:571] Iteration 15300, lr = 4.98494e-05
I0320 17:33:05.516649  5551 solver.cpp:242] Iteration 15400, loss = 0.134246
I0320 17:33:05.516700  5551 solver.cpp:258]     Train net output #0: loss = 0.134246 (* 1 = 0.134246 loss)
I0320 17:33:05.516716  5551 solver.cpp:571] Iteration 15400, lr = 4.97021e-05
I0320 17:33:13.837975  5551 solver.cpp:346] Iteration 15500, Testing net (#0)
I0320 17:33:28.722203  5551 solver.cpp:414]     Test net output #0: accuracy = 0.959772
I0320 17:33:28.722373  5551 solver.cpp:414]     Test net output #1: loss = 0.203878 (* 1 = 0.203878 loss)
I0320 17:33:28.765553  5551 solver.cpp:242] Iteration 15500, loss = 0.0809671
I0320 17:33:28.765585  5551 solver.cpp:258]     Train net output #0: loss = 0.080967 (* 1 = 0.080967 loss)
I0320 17:33:28.765604  5551 solver.cpp:571] Iteration 15500, lr = 4.95558e-05
I0320 17:33:37.168812  5551 solver.cpp:242] Iteration 15600, loss = 0.0727989
I0320 17:33:37.168864  5551 solver.cpp:258]     Train net output #0: loss = 0.0727988 (* 1 = 0.0727988 loss)
I0320 17:33:37.168880  5551 solver.cpp:571] Iteration 15600, lr = 4.94106e-05
```

Abbildung 5.7: Beispiel der Training Ausgabe

5.2 Implementierung

5.2.1 Dataset Vorbereitung

Folgende änderungen sind dem GTSRB Dataset zugeführt

- Das hier verwendete Trainingsverfahren benutzt Trainingsdaten und Validierungsdaten, so dass während das Training die Ergebnisse gegen neue Daten getestet werden. Es ist zwar möglich die gleiche Daten für Training und Validierung zu benutzen, eine Trennung ergebt jedoch bessere Ergebnisse. Deswegen werden die Daten getrennt so dass ein kleines Teil ($1/6$ der gesamten Training Dataset) wird als Validierungsdaten benutzt.

- GTSRB Bilder haben verschiedene größen (zwischen 15x15 und 250x250) und sind nicht immer Quadrat, sie werden konvertiert zu einer Einheitliche Größe von 48x48.

- Caffe akzeptiert mehrere Eingabedaten Formate, es ist empfohlen für große Datasets *lmdb* format zu benutzen. Caffe bietet ein Tool um die Daten zu konvertieren.

```
#!/usr/bin/env sh
# Resize and convert the Dataset to lmdb inputs

EXAMPLE=data/GTSRB/GTSRB_Trianing
DATA=data/TSR
TOOLS=build/tools

TRAIN_DATA_ROOT=date/GTSRB_Trianing/Train_Images/
VAL_DATA_ROOT=date/GTSRB/GTSRB_Trianing/Test_Images/

RESIZE_HEIGHT=48
RESIZE_WIDTH=48

BACKEND="lmdb"

echo "Creating tran lmdb..."
$TOOLS/convert_imageset \
                        --resize_height=$RESIZE_HEIGHT \
                        --resize_width=$RESIZE_WIDTH \
                        --shuffle \
                        $TRAIN_DATA_ROOT \
                        $DATA/train.txt \
                        $EXAMPLE/tsr_gray_train_lmdb

echo "Creating val lmdb..."
$TOOLS/convert_imageset \
                        --resize_height=$RESIZE_HEIGHT \
                        --resize_width=$RESIZE_WIDTH \
                        --shuffle \
                        $VAL_DATA_ROOT \
                        $DATA/val.txt \
                        $EXAMPLE/tsr_gray_val_lmd

echo "Done."
```

Listing 5.2: Script zum Konvertieren des Datasets

Die *val.txt* und *train.txt* enthalten der relativen Pfad zu den tatsächlichen Bilder zusammen mit ihre Klasse.

```
00000/00003_00024.ppm 0
00000/00003_00028.ppm 0
00001/00000_00004.ppm 1
```

```
4 00001/00000_00010.ppm 1
```

Listing 5.3: Ausschnitt aus der Datei val.txt

5.2.2 Das Modell

Verschiedene Modellen wurden implementiert und miteinander verglichen. Hier wird das Modell in Abbildung 5.1 (S. 35) diskutiert.

Schicht	Type	Anzahl der Neuronen	Anzahl der Maps	Kernel größe	Stride
1	Eingabe	48x48	3	_	_
2	Konvolution	42x42	100	7	1
3	Pooling	21x21	100	2	2
4	Konvolution	16x16	150	6	1
5	Pooling	8x8	150	2	2
6	Konvolutione	6x6	200	3	1
7	Pooling	3x3	200	2	2
8	MLP	300	_	_	_
9	MLP	43	_	_	_

Tabelle 5.1: Das verwendete Modell

Abbildung 5.8: Das verwendete Netz Modell

Schicht 1 : Die Eingabedaten sind Verkehrszeichen der Größe 48x48 die Farbkanäle werden einzeln bearbeitet (3 Maps). Das Modell ist für Training und Validierung benutzt, deshalb verfügt es über zwei Eingabe Phasen (Lern-Phase und Validierungs-Phase).

Schicht 2 : Die erste Konvolution ist auf die drei 48x48 Maps angewendet.Wenn der 7x7 Faltungskernel auf ein 48x48 Bild angewendet wird, entstehen Ränder mit 3 Pixelbreite auf jeder Seite. Es werden 100 (42x42)-Maps erzeugt. Abbildung 5.11 (S. 36) zeigt Beispiele Aus den Ausgabedaten der zweiten Schicht und die der Abbildung 5.10 (S. 36) sie die benutzte 7x7 Kernel zu sehen.

Abbildung 5.9: Eingabeschicht Maps

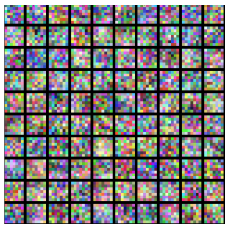


Abbildung 5.10: Einige trainierte Konvolutionsmaske

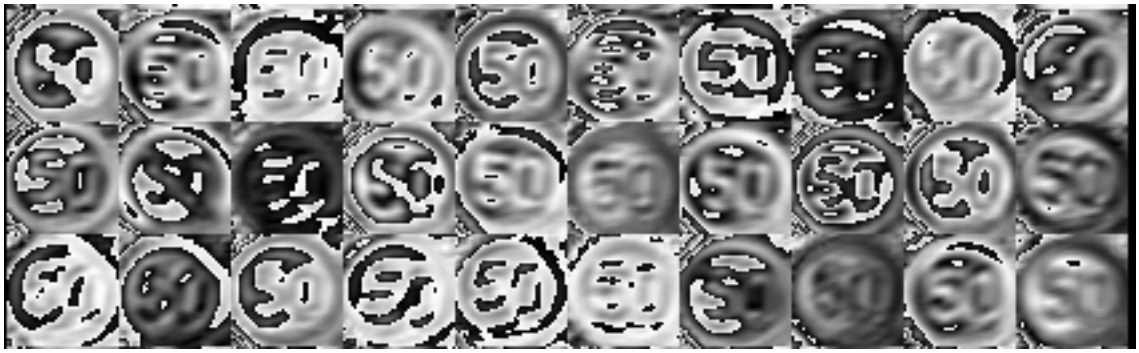


Abbildung 5.11: Beispiele Für resultierende Feature Maps der erste Konvolution

$Schicht\ 3$: 2x2 Maximum Pooling wird auf die Maps angewendet. Die Anzahl der Maps wird nicht geändert. Abbildung 5.12 (S. 37) zeigt Beispiele Aus den Ausgabedaten der Pooling Schicht.

$Schicht\ 4-7$: Da werden Schicht 2 und 3 wiederholt um mehr Feature Maps zu erzeugen.

$Schicht\ 8-9$: ist Ein Multilayer Perzeptron Netz, mit mit zwei Schichten.

Abbildung 5.12: Die Feature Maps nach Ausführung des Max-Pooling Verfahrens

5.2.3 Training

In Training wird folgenden Solver verwendet.

```
net: "examples/tsr/test6/train_val.prototxt"

test_iter: 100
test_interval: 500

# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005

# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75

# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 100000
# snapshot intermediate results
snapshot: 10000
snapshot_prefix: "examples/tsr/test6/tsr"
# solver mode: CPU or GPU
solver_mode: GPU
```

Listing 5.4: Verwendete Solver

$base\_lr$ ist die Schrittweiten Faktor, Die *learning rete policy* bestimmt wie und wann die Gewichten geändert werden. aus der caffe source code:

```
// The learning rate decay policy. The currently implemented learning rate
// policies are as follows:
//    - fixed: always return base_lr.
//    - step: return base_lr * gamma ^ (floor(iter / step))
```

```
//    - exp: return base_lr * gamma ^ iter
//    - inv: return base_lr * (1 + gamma * iter) ^ (- power)
//    - multistep: similar to step but it allows non uniform steps defined by
//      stepvalue
//    - poly: the effective learning rate follows a polynomial decay, to be
//      zero by the max_iter. return base_lr (1 - iter/max_iter) ^ (power)
//    - sigmoid: the effective learning rate follows a sigmod decay
//      return base_lr ( 1/(1 + exp(-gamma * (iter - stepsize))))
//
// where base_lr, max_iter, gamma, step, stepvalue and power are defined
// in the solver parameter protocol buffer, and iter is the current iteration.
```

Listing 5.5: Learninig rate policies Berechnung

5.2.4 Detektion

Es wurde mit Hilfe von OpenCV ein einfachen Blob Detektor implementiert. Der BLOB-Detektor: trennt Objekte vom Hintergrund basierend auf ihre Helligkeitswerten. Diese Objekte können anschließend anhand ihrer Größe, Geometrie oder Lage unterschieden. Opencv bietet ein BLOB-Algorithmus Implementierung.

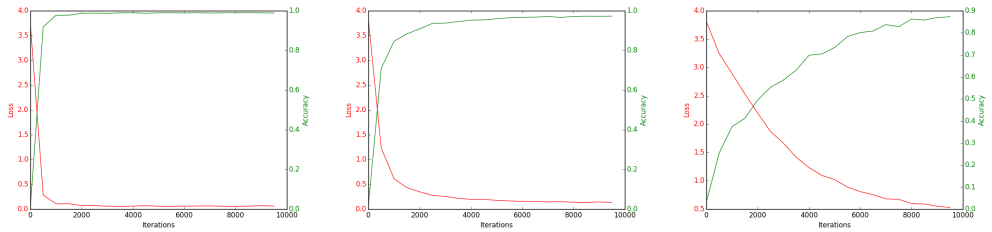
```
import cv2

image_path = 'test_image.ppm'
im = cv2.imread(image_path)

# convert to HSV
hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)

# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()

# Filter by Circularity
params.filterByCircularity = True
params.minCircularity = 0.75

# Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector(params)

## Detect blobs.
keypoints = detector.detect(hsv[:,:,1])
```

Listing 5.6: BLOB-Detektor Beispiel

6 Diskussion der Ergebnisse

6.1 Training

Das Modell 5.1 (S. 35) wurde mit verschiedenen learn-policies trainiert. Die Abbildung 6.1 (S. 39) zeigt das Training Verhalten wärend die ersten 10.000 Iterationen. Es ist zu sehen dass das learn-rate ein sehr große Rolle bei der Lernzeit spielt.



(a) $lr\_rate = 0.01$ (b) $lr\_rate = 0.001$ (c) $lr\_rate = 0.0001$

Abbildung 6.1: Der Lernprozess bei verschiedene *learning-rate*

6.2 Klassifikation

Es wurden 3 Netzwerke implementiert und mit den gleichen Dataset und Solver für jeweils 10.000 iterationen trainiert. Sie werden mit den gleichen Klassifikator gegen zwei Bild Sammlungen getestet: Die GTSRB Test Dataset, und Bilder aus dem GTSTB Dataset. Tabelle 6.1 (S. 40) zeigt die verwendeten Netzen und die Ergebnis der Klassifikation.

Netze	Phase 1		Phase 2		Phase 3		Phase 4	Ergebnisse	
	conv	pool	conv	pool	conv	pool	conv	Rate 1	Rate 2
Netz 1	7x7-1	2x2-2	4x4-1	2x2-2	4x4-1	2x2-2	_	93,58%	86,81%
Netz 2	7x7-1	2x2-2	6x6-1	2x2-2	3x3-1	2x2-2	_	93,06%	87,85%
Netz 3	4x4-1	3x3-3	4x4-1	2x2-2	4x4-1	_	3x3-1	91,33%	83,50%

Tabelle 6.1: Klassifikationsergebnisse

$Netz\ 1$ Das Cireşan et al. (2012) Modell

$Netz\ 2$ basiert sich auf $Netz\ 1$ und benutzt größere Kernels in der ersten Phasen.

$Netz\ 3$ durch die letzte Konvolution wurde die Größe der Feature Maps zu 1 Pixel reduziert und es wird direkt mit einem 43 Neuronen Schicht klassifiziert

$Rate\ 1$ Erkennungsrate der Klassifikation auf 12.657 bilder der Online GTSRB-Dataset

$Rate\ 2$ Erkennungsrate der Klassifikation auf 766 bilder der GTSTB-Dataset

Abbildung 6.2: Einige Beispiele für richtig erkannten Verkehrszeichen

Abbildung 6.3: Einige Beispiele für falsch erkannten Verkehrszeichen

Das Ergebnis kann sich variieren Je nach verwendete Datasets und die ausgeführte Bild Vorverarbeitung der Trainingsdaten und die Klassifikationsverfahren.

6.3 Detektion

Daten der GTSTB Dataset wurden benutzt um Die Detektionsrate zu ermitteln. Mit einem BLOB-Detektor und das in 5.1 (S. 35) beschriebenen Netz wurde eine Erkennungsrate von 42% erreicht. Das liegt daran Das der Detektor nicht alle Schilder lokalisieren konnte. Man kann also zum Schluss kommen, dass der BLOB-Detektor alleine bietet keine ausreichende Detektion bietet.

Abbildung 6.4: Beispiele für richtig identifizierte Verkehrszeichen

Abbildung 6.5: Beispiele von Bilder, die nicht vom Detektor erfasst wurden

Die in Abbildung 6.6 (S. 42) gezeigten Verkehrszeichen wurden richtig detektiert. Die Klassifikation wurde aber fehlgeschlagen. Obwohl der Klassifikator sehr gute Ergebnisse bei der Klassifikationsaufgabe erzielt hat. Im Bild (b) z.B. wurde der Klassifikator nicht auf das Fußgängerüberweg Schild trainiert, (es gehört nicht zum GTSRB Dataset). Im Bild (a) wurden Ampeln als Verkehrszeichen erkannt. Eine Lösung für dieses Problem ist den Klassifikator mit einer zusätzlichen Klasse zu trainieren. Diese Klasse

(a) (b)

Abbildung 6.6: Beispiele von Bilder, die richtig detektiert wurden aber falsch klassifiziert

besteht aus Bilder von Objekten die üblicherweise von der Kamera aufgenommen werden aber nicht zur Verkehrszeichen grhören (z,B Bäume, Gebäude, Himmel...).

(a) (b)

Abbildung 6.7: In Bild (b) wurde der Klassifikator mit einer zusätzlichen Hintergrund Klasse (43) trainiert. d.h alle gefundene Objekte im Bild sind richtig identifiziert, anders als Bild (a).

7 Fazit und Ausblick

In dieser Arbeit wurde die Problematik bei der Verkehrszeichenerkennung diskutiert und verschiedene Lösungen vorgestellt. Ein Überblick über das Thema wird in Kapitel 2.2 (S. 6) gegeben. Grundlagen und Anwendungen der Konvolutionalen Neuronalen Netzwerke insbesondere in der Verkehrszeichenerkennung sind im Kapitel 4 (S. 11) erläutert. Eine Realisierung mit CNN und einem klassischen Detektor ist im Kapitel 5.2 (S. 33) implementiert.

Es hat sich gezeigt, dass Klassifikationsaufgaben einfach und effektiv mit CNN implementiert werden können. Die Detektion ist dabei immer noch die schwierigste Aufgabe. Die klassischen Bildverarbeitungsmethoden liefern gute Ergebnisse. Die Verwendung der CNN in der Objekterkennung ist ein sehr aktuelles Thema und die vorhandenen Lösungen versprechen bessere Ergebnisse in der Zukunft.

Literaturverzeichnis

G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000.

Dan Cireşan, Ueli Meier, Jonathan Masci, and JÃijrgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333 – 338, 2012. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2012.02.023. URL http://www.sciencedirect.com/science/article/pii/S0893608012000524. Selected Papers from {IJCNN} 2011.

Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181. ISSN 1573-1405. doi: 10.1023/B:VISI.0000022288.19776.77. URL http://dx.doi.org/10.1023/B:VISI.0000022288.19776.77.

Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. URL http://arxiv.org/abs/1506.02025.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL http://arxiv.org/abs/1411.4038.

M. Mathias, R. Timofte, R. Benenson, and L. Van Gool. Traffic sign recognition x2014; how far are we from the solution? In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8, Aug 2013. doi: 10.1109/IJCNN.2013.6707049.

P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813, July 2011. doi: 10.1109/IJCNN.2011.6033589.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. URL http://arxiv.org/abs/1312.6229.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323 – 332, 2012. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2012.02.016. URL http://www.sciencedirect.com/science/article/pii/S0893608012000457. Selected Papers from {IJCNN} 2011.

J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. URL https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013.

J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, Feb 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.79.

Yihui Wu, Yulong Liu, Jianmin Li, Huaping Liu, and Xiaolin Hu. Traffic sign detection based on convolutional neural networks. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7, Aug 2013. doi: 10.1109/IJCNN.2013.6706811.

F. Zaklouta, B. Stanciulescu, and O. Hamdoun. Traffic sign classification using k-d trees and random forests. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2151–2155, July 2011. doi: 10.1109/IJCNN.2011.6033494.

Yujun Zeng, Xin Xu, Yuqiang Fang, and Kun Zhao. Traffic sign recognition using extreme learning classifier with deep convolutional features. In *The 2015 international conference on intelligence science and big data engineering (IScIDE 2015), Suzhou, China*, 2015.

Symbolverzeichnis

ELM Extreme Learning Machine, Seite 36

HOG Histogram of oriented gradient, Seite 36

MLP MultiLayer Perceptron, Seite 36

MSER maximally stable extremal regions, Seite 36

POI Region Of Iterrest, Seite 36

SVM support vector machines, Seite 36

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 8. April 2016 Mahmoud Dariti

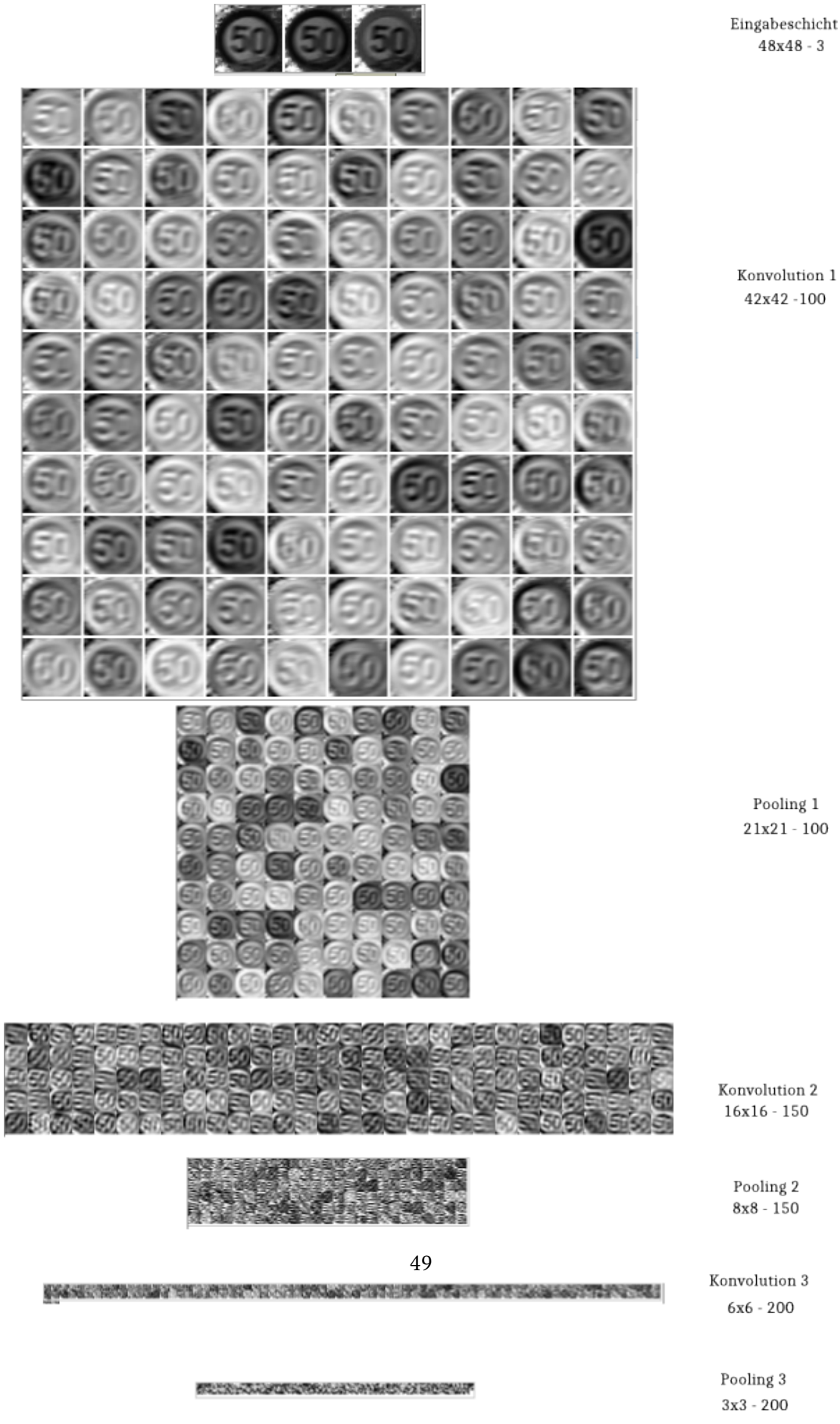


Abbildung 1: Konvolutionale Neuronale Netzwerk