hochschule mannheim

Bildklassifikation zur Navigation selbstfahrender Autos mittels Convolutional Autoencoder

Edward Alhanoun

Master-Thesis

Studiengang Informationstechnik

Fakultät für Informatik

Hochschule Mannheim

31.08.2019

Betreuer: Prof. Dr. Jörn Fischer, Hochschule Mannheim

Zweitkorrektor: Prof. Dr. Ivo Wolf, Hochschule Mannheim

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d.h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 31.08.2019

Edward Alhanoun

Abstract

Bildklassifizierung zur Navigation selbst fahrender Autos mittels Convolutional Autoencoder

Obwohl die Convolutional neuronalen Netzwerke (CNN) bei fast allen Aufgaben der Computer Vision auf dem neuesten Stand der Technik sind, bleibt ihr Training eine schwierige Aufgabe. Das Lernen unüberwachter Repräsentation mit einem Convolutional Autoencoder kann zur Initialisierung von Netzwerkgewichten verwendet werden und verbessert nachweislich die Testgenauigkeit nach dem Training. Die vorliegende Masterarbeit zeigt die Entwicklung eines Modells, das aus mehreren Convolutional Autoencodern (CAE) und einer Support Vector Machine (SVM) besteht und für Bildklassifizierung zur Navigation selbst fahrender Autos verwendet werden kann. Das Modell wird in Python nur mit Hilfe der Numpy Bibliothek programmiert und mit dem GTSRB (German Traffic Sign Recognition Benchmark) Dataset trainiert und getestet. Die Ergebnisse des Modells sind nicht ausreichend, könnten aber zukünftig als Grundlage dienen und verbessert werden. Der größte Vorteil dieses Models ist, dass das Modell flexibel ist und an jeder Stelle bearbeitet werden kann. Außerdem können die Parameter unkompliziert eingestellt werden. Das Modell könnte in Zukunft als Baustein für eine effiziente Architektur zur Klassifizierung von Verkehrszeichen dienen, die in der nächsten Fahrzeuggeneration implementiert werden kann.

Abstract

Image classification for the navigation of self-driving cars using of Convolutional Autoencoder

While Convolutional Neural Networks (CNN) are state of the art in almost all tasks of Computer Vision, their training remains a difficult task. Learning unsupervised representation with a convolutional autoencoder can be used to initialize network weights and has been proven to improve test accuracy after training. This master thesis shows the development of a model consisting of several Convolutional Autoencoders (CAE) and a Support Vector Machine (SVM) that can be used to classify images for navigation of self-driving cars. The model was programmed in Python using only the Numpy library and trained and tested with the GTSRB (German Traffic Sign Recognition Benchmark) dataset. The results of the model are not enough but can be used as a reference and improved in the future. The main advantage of this model is that it is flexible and can be edited at any point. Furthermore, the parameters can be easily adjusted. the model can serve in the future as an essential component for an efficient architecture for the classification of traffic signs, which can be implemented in the next vehicle generation.

Inhalt

1 Einleitung

Die Automobilindustrie entwickelt sich stetig weiter in Richtung Fahrautomatisierung. Für selbstfahrende Autos sind Systeme notwendig, die den Fahrer ersetzen. Um diesen Systemen vertrauen zu können, müssen sie sehr sorgfältig entwickelt werden, wobei Sicherheitsaspekte im Vordergrund stehen. Ein wichtiges Regulationsinstrument des Straßenverkehrs sind Verkehrszeichen.

1.1 Motivation

Die Erkennung und Klassifizierung von Verkehrszeichen haben realistische Anwendungen, wie z.B. bei der Fahrerassistenz und Fahrersicherheit sowie beim automatisierten Fahren. Die Verkehrsschilder sind eindeutig und für den Fahrer gut sichtbar [1, p. 1]. Dennoch gibt es verschiedene Herausforderungen bei der Verkehrszeichenerkennung und Klassifizierung. Beispielweise können Verkehrszeichen ähnlich aussehen. Außerdem kann eine unterschiedliche Beleuchtung dazu führen, dass sich die Farbe aufgrund der Wetterbedingungen allerdings ändert. Eine andere Möglichkeit ist, dass die Verkehrszeichen wegen Verblassen, Verschmutzung oder eines physischen Schadens nicht erkannt werden können [2, p. 1]. Die folgende Abbildung zeigt eine Vielzahl an Beispielen, die besondere Herausforderungen für die Verkehrszeichenerkennung sind (Abbildung 1.1).

Abbildung 1.1: Ein paar Beispiele der Verkehrszeichen, die wegen der Wetterbedingungen, physischen Schaden, Verschmutzung und Verblassen schwer zu erkennen sind. Quelle: [1, p. 1]

Es gibt viele Forschungen im Bereich des maschinellen Lernens, um diese Probleme zu lösen. Die Bilderklassifizierung wird durch verschiedene Methoden implementiert, wie z.B. die Verfahren der Bildverarbeitung (HOG, SIFT...), neuronale Netzwerke und Support Vector Machine (SVM). Aber die gefundenen Lösungen werden mit Hilfe von fertigen Modellen verwendet und auf verschiedenen Plattformen implementiert, da das Training der Modelle eine starke Rechenleistung und große Zeit erfordert. Das hat zur Folge, dass die Modelle nicht flexibel sind. Im Rahmen dieser Arbeit wird versucht, ein Model, welches das Klassifizierungsproblem ohne Verwendung von fertigen Modellen lösen kann. Hierdurch wird nicht nur die Trainingszeit verringert, sondern auch ein flexibles Modell eingeführt.

1.2 Zielsetzung

Der Ansatz basiert auf Convolutional neuronale Netzwerke. Es handelt sich um einen biologisch inspirierten mehrschichtigen Feed-Forward Architektur, welche mehrere Stufen invarianter Merkmale lernen kann. Anschließend wird der Ausgang der letzten Convolutional Schicht zum Eingang des Klassifikators zugeführt. Das erlaubt dem Klassifikator nicht nur die starken Merkmale zu verwenden, sondern auch die schwachen Merkmale, die üblicherweise weniger unveränderlich sind [1, p. 1]. In dieser Arbeit werden die wichtigsten Merkmale der Bilder durch mehrere Convolutional Schichten extrahiert, danach werden die extrahierten Merkmale durch die Support Vektor Maschine klassifiziert. Die folgende Abbildung stellt eine ähnliche Struktur dieser Arbeit dar.

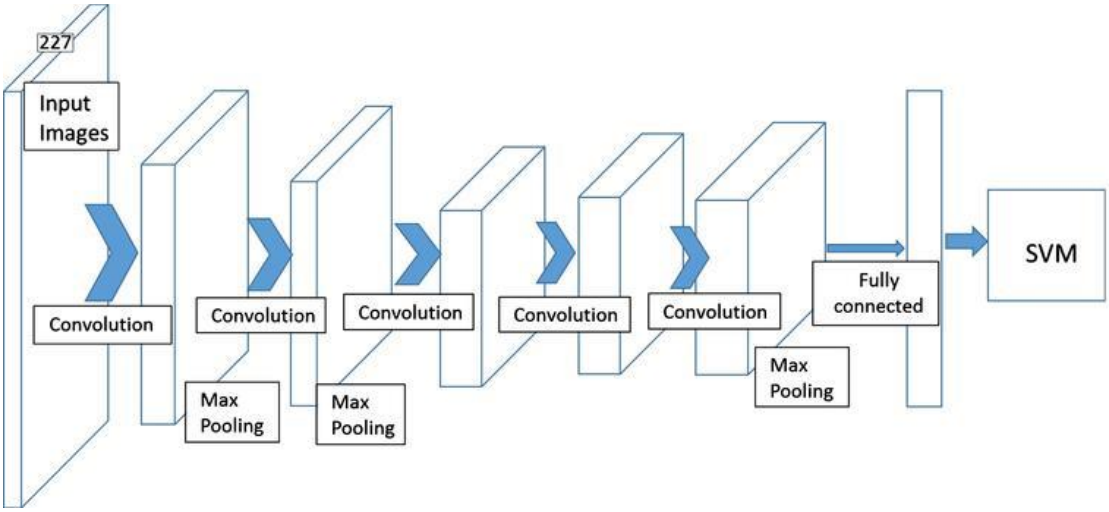


Abbildung 1.2: Eine Struktur eines Modells besteht aus mehrerer Convolution Schichten und einem SVM Klassifikator. Quelle: https://content.iospress.com/media/xst/2018/26-6/xst-26-6-xst18386/xst-26-xst18386-g002.jpg?width=755

1.3 Aufbau der Arbeit

In dieser Arbeit wird zuerst ein Einblick über die Bildklassifizierung mittels maschinellen Lernens und Deep Learnings gegeben. Danach werden die wesentlichen theoretischen Grundlagen im Kapitel drei erläutert. Dafür wird das neuronale Netzwerk allgemein und insbesondere Convolutional neuronales Netzwerk und Autoencoder behandelt. Im Anschluss gibt es einen Überblick über die Restricted Boltzmann Maschine (RBM) und ihren Lernalgorithmus. Danach wird die Klassifizierung durch die Support Vektor Maschine (SVM) erklärt. Am Ende des Kapitels werden die für diese Arbeit wesentlichen Trainingsalgorithmen behandelt. Der Kapitel 4 beschreibt den verwendeten Datensatz in dieser Arbeit und die Datenvorverarbeitung. Anschließend werden der Aufbau des Modells und die Funktionalität jedes Teil des Modells erklärt und welche Vorteile hat das Modell bei der Klassifizierungsproblem. Im 6 Kapitel werden die Teste und ihre Ergebnisse diskutiert und die besten Parameter ausgewählt. Am Ende stehen eine Zusammenfassung und ein Ausblick, wo diese Arbeit eingesetzt werden kann und wie sie für das Leben in der Zukunft nützlich und hilfreich kann.

2 Bildklassifizierung

Es wird versucht, das Bildklassifizierungsproblem durch viele Experimente zu lösen. Die alten Methoden basieren darauf, die wesentlichen Merkmale eines Bilds manuell zu definieren, dann an den Klassifikator weiterzugeben. Aber das hat nur für einfache Klassifizierungsaufgaben funktioniert wie das Finden eines Objekts, das eine bestimmte Form hat, in einem Bild [3, p. 21]. Die aktuellen Bildklassifizierungsverfahren werden nachfolgend erkläret.

2.1 Bildklassifizierung mittels maschinellen Lernens

Die Definition des maschinellen Lernens (ML) ist die Algorithmen, die bei der künstlichen Intelligenz (KI) verwendet werden, um die Probleme verstehen und die Entscheidungen treffen zu können. Beispiele sind die Logistik Regression und naive Bayes [4, p. 2]. Eine effiziente Methode für das Klassifizierungsproblem ist die Anwendung von maschinellen Lernalgorithmen. Das Konzept des maschinellen Lernens wird definiert als: „ein Computerprogramm lernt anhand von Wissen aus Erfahrungen und kann nach dem Lernen die Ergebnisse verallgemeinern“ [5, p. 1]. Das Ziel der Methoden des maschinellen Lernens ist die Entwicklung von Algorithmen, die anhand des Gelernten aus gegebenen Daten die Vorhersagen für neue Daten treffen zu können. Jedoch sind die maschinellen Lernalgorithmen nicht in der Lage, die Rohdaten zu verarbeiten. Deshalb erfordert das maschinelle Lernsystem ein Exzerpt, um die bedeutenden Merkmale aus den Rohdaten zu extrahieren und in Feature-Maps umzuwandeln. Anschließend werden die Feature-Maps durch den Klassifikator mit Klassen verknüpft [5, p. 1]. Die Extraktion der Merkmale lässt sich durch Drehung, Skalierung Transformation oder Normalisierung erreichen [6, p. 6].

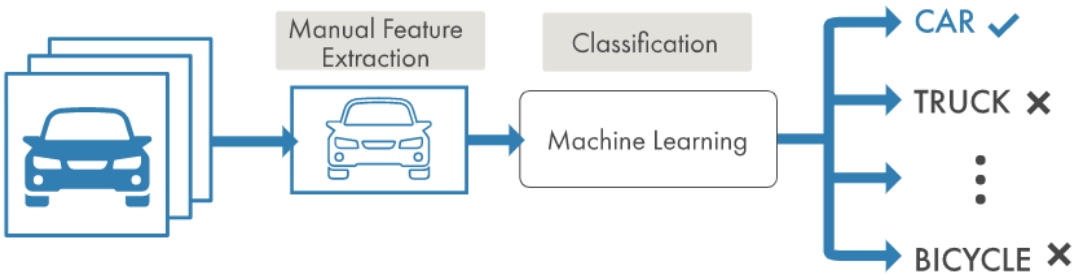


Abbildung 2.1: Manuales Extrahieren der Merkmale aus einem Bild, dann die Bildklassifizierung mittels eines maschinellen Lernalgorithmus. Quelle: https://de.mathworks.com/discovery/deep-learning.html

2.2 Bildklassifizierung mittels Deep Learning

Deep Learning (DL) ist ein Ansatz im maschinellen Lernen, um Wissen aus Erfahrung zu gewinnen und das zu lösende Problem mit Hilfe einer Hierarchie von Lösungskonzepten zu verstehen. Diese Konzepte sind jeweils durch einfachere Teillösungen definiert. Die Hierarchie von Lösungskonzepten ermöglicht dem Computer, komplexe Lösungsansätze aus einfacheren Ansätzen zu bilden [4, p. 8]. Ein Graph, der diese Konzepte aufeinander abbildet, besteht aus vielen Schichten und wird somit „tief“ genannt. Deshalb spricht man bei diesem Ansatz der künstlichen Intelligenz von „Deep Learning“. Zum besseren Verstehen von Deep Learning müssen die Grundlagen des maschinellen Lernens verstanden werden [4, p. 98]. Heutzutage wird Deep Learning zum Lösen des Bildklassifizierungsproblems am häufigsten verwendet, weil das Deep Learning Verfahren ermöglicht, die wichtigen Merkmale für die Bilderkennung und -klassifizierung aus einem Bild selbstständig auf verschiedene Ebenen zu extrahieren, dann zu einer Klasse zuordnen. Dieses Verfahren wird durch ein Convolutional neuronales Netzwerk geschaffen [3, p. 22].

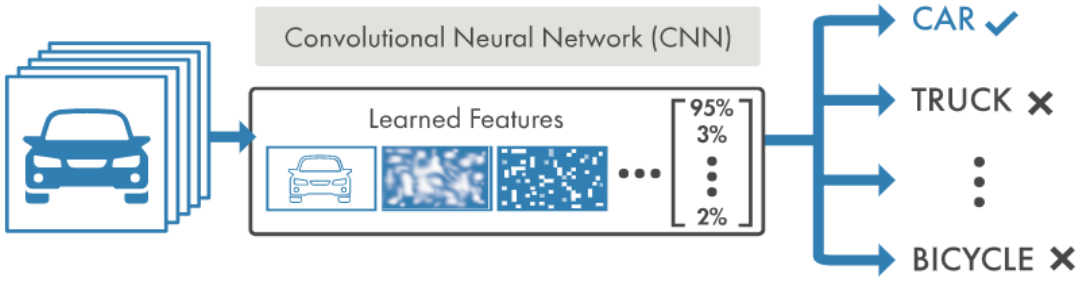


Abbildung 2.2: Automatisches Extrahieren der Merkmale aus einem Bild und Klassifizierung mittels eines Convolutional neuronalen Netzwerks. Quelle: https://de.mathworks.com/discovery/deep-learning.html

2.3 Beziehung zwischen Deep Learning, maschinellem Lernen und künstlichen Intelligenz

Um die Beziehung zwischen DL, ML und KI, die im Kapitel 2.1 und 2.2 erwähnt sind, besser zu verstehen, werden die Begriffe in diesem Kapitel kürz erklärt. Das Gebiet KI innerhalb der Informationstechnologie zielt auf die Entwicklung von Systemen ab, welche die Aufgaben ausführen, die normalerweise menschliche Intelligenz erfordern und in verschiedene Techniken verzweigen. Anderseits beschreibt ML ein Teilgebiet von KI und beinhaltet alle Ansätze, die es Computern ermöglichen, aus

Daten zu lernen, ohne explizit programmiert zu sein. Die ML Rechenmodelle und Algorithmen beinhaltet, die die Architektur der biologischen neuronalen Netzwerke im Gehirn nachahmen. Zu den Techniken, die unter dem ML stehen, hat sich DL als eine der vielversprechendsten herausgestellt. Tatsächlich ist DL eine Technik von ML, die sich wiederum auf eine breitere KI-Familie bezieht (Abbildung 2.3). Insbesondere gehören DL-Methoden zu den Repräsentationslernmethoden mit mehreren Repräsentationsebenen, die Rohdaten verarbeiten, um Klassifizierungs- oder Erkennungsaufgaben durchzuführen [7].

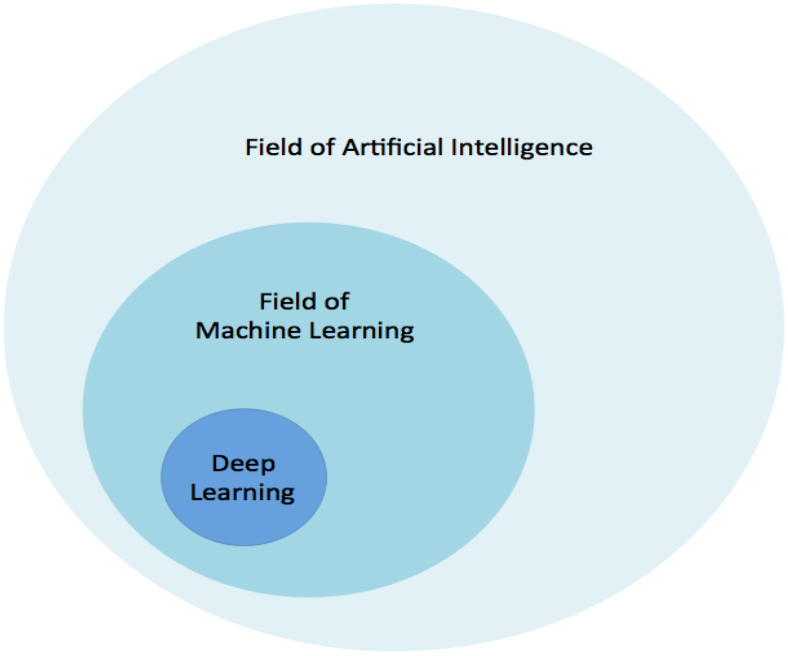


Abbildung 2.3: Ein Diagramm zeigt die Beziehung zwischen KI, ML und DL. Wobei DL eine Art von ML ist, das für viele Ansätze zur KI verwendet wird. Quelle: [8, p. 4]

3 Theoretische Grundlagen

3.1 Künstliches neuronales Netzwerk

Das biologische Neuron ist eine Nervenzelle, die die grundlegende Funktionalität für das Nervensystem darstellt. Neuronen existieren, um miteinander zu kommunizieren und elektrochemische Impulse über Synapsen von einer Zelle zur nächsten zu leiten. Der Impuls muss stark genug sein, um die Freisetzung von Chemikalien über einen synaptischen Spalt zu aktivieren. Die Nervenzelle besteht aus Dendriten, Soma, Axon und Synapsen (siehe Abbildung 3.1). Über die Dendriten empfangen die Nervenzellen Informationen von anderen Nervenzellen, die anschließend im Zellkern verarbeitet und über das Axon an weitere Nervenzellen weitergeleitet werden [8, pp. 43, 44].

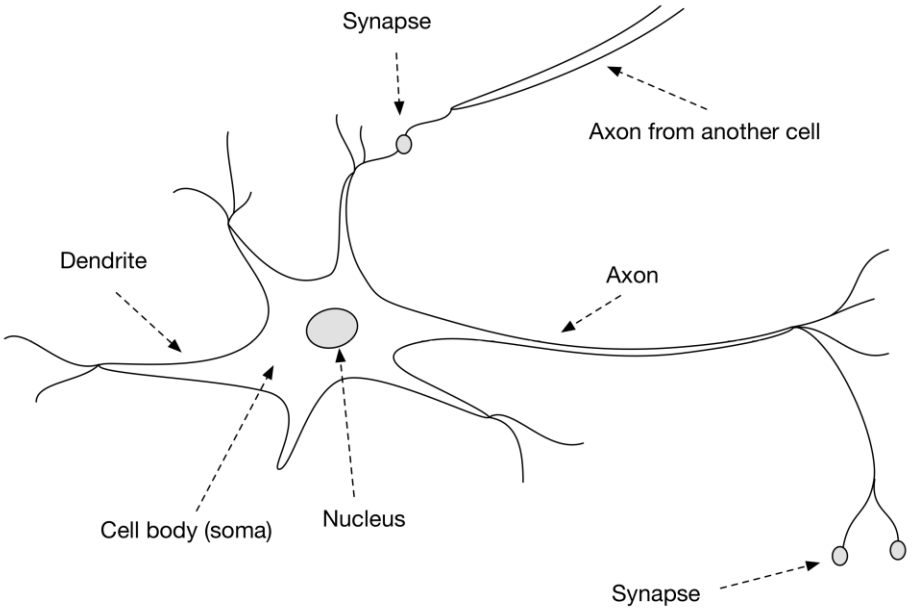


Abbildung 3.1: Schematische und stark vereinfachte Darstellung einer einzelnen Nervenzelle besteht aus Dendriten, Zellkörper, Post- und präsynaptische Verbindung am Ende der Dendriten und Axon. Quelle: [8, p. 44]

Künstliche neuronale Netze (KNN) haben viele Ähnlichkeiten zu natürlichen neuronalen Netzen. Auch im künstlichen neuronalen Netz nimmt jedes Neuron unterschiedliche Informationen auf und gibt anschließend ein Signal als Ausgabe aus (siehe Abbildung 3.2). Die Anordnung der Neuronen ist allerdings typischerweise in Ebenen aufgebaut. Die Verbindungen zwischen den Neuronen liegen in Form von Gewichten vor. Während des Trainings werden diese Gewichte angepasst, bis das

KNN die gewünschte Funktionsweise aufweist. Das Anpassen der Gewichte ist die Hauptmethode, mit der das neuronale Netz neue Informationen lernt. Der Aufbau eines neuronalen Netzwerks kann mit Anzahl der Neuronen, Anzahl der Schichten und die Art der Verbindung zwischen Schichten definiert [4, p. 41].

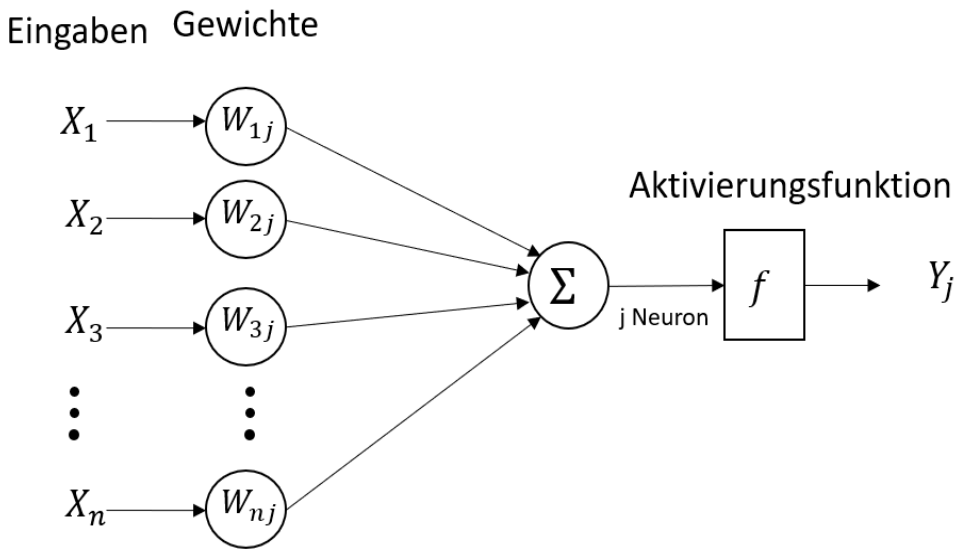


Abbildung 3.2: Schematische Darstellung eines künstlichen Neurons mit n Eingabeneuronen $(X_1, X_2, \dots, X_n)$, einem versteckten Neuron (j) und einer Ausgabe (Y_j), wobei (w_{ij}) die Verbindungen zwischen den Eingabeneuronen $(i = 1,2, \dots, n)$ und dem versteckten Neuron (j) sind. (einige Darstellung)

In der Abbildung 3.2 wird die Ausgabe eines Neurons durch die folgende Gleichung berechnet:

$$y_j = f(\sum_{i=0}^{n} w_{ij} x_i + b_j) \quad (1)$$

Hier ist w_{ij} die Gewichte des Neurons, x_i ist die Eingabewerte, y_j ist die Ausgabewert und f ist die Aktivierungsfunktion (wird anschließend erklärt). Das bekannteste und am einfachsten zu verstehendem neuronalem Netzwerk ist das Feed Forward mehrschichtige neuronale Netzwerk. Es besteht aus einer Eingabeschicht, eine oder mehrere versteckten Schichten und eine einzelne Ausgabeschicht. Jede Schicht kann eine unterschiedliche Anzahl von Neuronen haben und jede Schicht ist vollständig mit der benachbarten Schicht verbunden (siehe Abbildung 3.3) [8, p. 42].

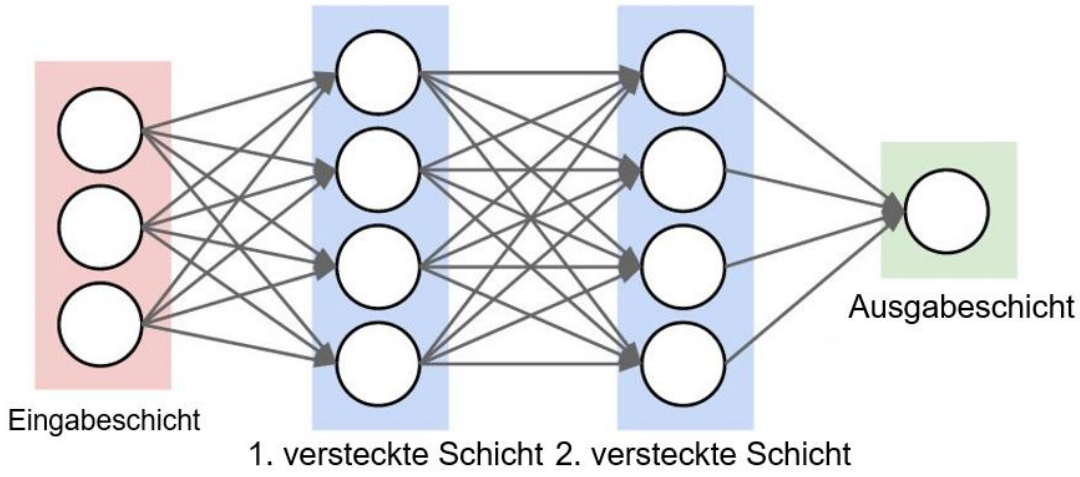


Abbildung 3.3: Schematische Darstellung eines Feed Forward mehrschichtigen neuronalen Netzwerk mit drei Input Neuronen, zwei versteckten Schichten und einer Output Neuron. Jede versteckte Schicht besteht aus vier Neuronen (einige Darstellung)

3.2 Aktivierungsfunktion

Die Verwendung einer Aktivierungsfunktion im neuronalen Netzwerk ermöglicht ein komplexes funktionales Mapping zwischen den Eingängen und Ausgängen eins Neurons zu erlenen. Außerdem wandelt sie ein Eingangssignal eines Neurons in ein Ausgangssignal um und ermöglicht dies Ausgangssignal als Eingabesignal der nächsten Schicht zu verwenden. Die Aktivierungsfunktion gibt das Netzwerk nichtlineare Eigenschaften, so dass das Netzwerk nicht nur auf lineare Operationen beschränkt ist [9]. Darüber hinaus werden die irrelevanten Informationen, die keinen maßgleichen Einfluss auf das Endergebnis haben, durch Aktivierungsfunktionen aussortiert [10]. Es gibt verschiedene Aktivierungsfunktionen wie Sigmoid, Tanh (Tangens hyperbolicus) und ReLu (engl. Rectified Linear Units). Im Folgenden wird jede Aktivierungsfunktion mit seiner Funktion und seinem Graph beschrieben:

Sigmoid
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

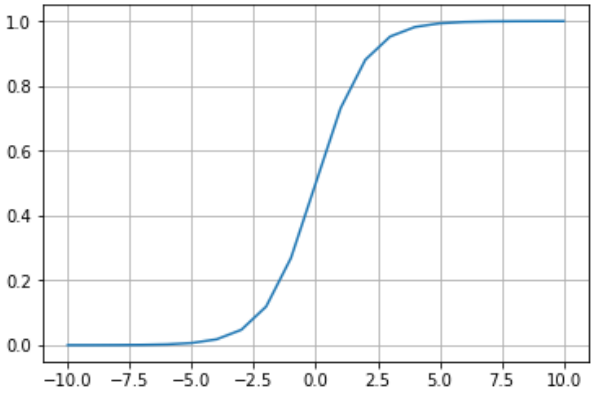


Abbildung 3.4: Ein Beispiel der Sigmoid Funktion in der Gleichung (2) mit Eingaben zwischen [-10,10] und liefert Ausgabe zwischen [0, 1]. (Spyder Konsole)

Tanh $$f(x) = 2sigmoid(2x) - 1 = \frac{1}{1 + e^{-2x}} - 1 \quad (3)$$

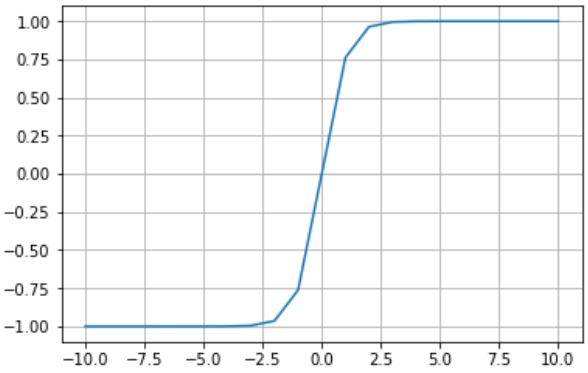


Abbildung 3.5: Ein Beispiel der Tanh Funktion in der Gleichung (3) mit Eingaben zwischen [-10, +10] und liefert Ausgabe zwischen [-1, +1] (Spyder Konsole)

ReLu $$f(x) = max\,(0, x) \quad (4)$$

Leaky ReLu $$f(x) = \begin{cases} ax;\ \text{wenn } x < 0 \\ x\ ;\ \text{wenn } x \geq 0 \end{cases} \quad (5)$$

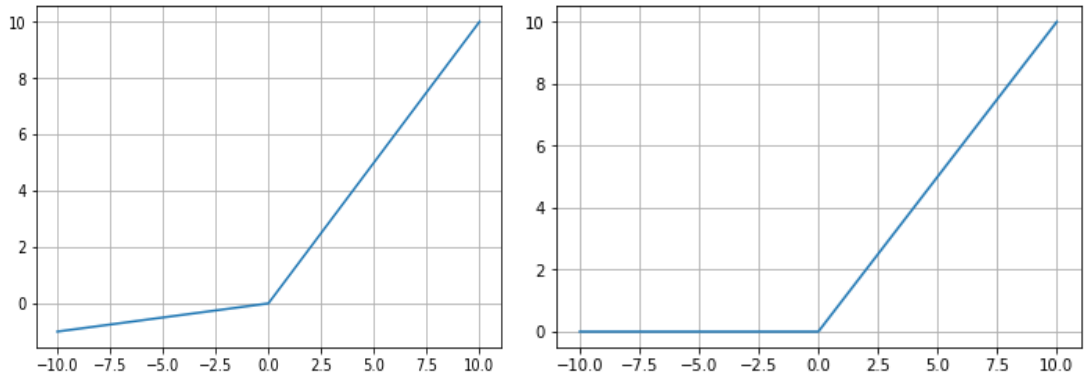


Abbildung 3.6: Ein Beispiel von ReLu (rechts) und Leaky ReLu (links) Funktionen mit Eingabe zwischen [-10, +10]. Die Leaky ReLu gibt die Ausgabe einen kleinen Wert, wenn die Eingabe weniger als Null. Im Gegenteil gibt ReLu die Ausgabe einen Nullwert, wenn die Eingabe weniger als Null. (Spyder Konsole)

In dieser Arbeit wird die rechnerisch effiziente Leaky ReLu Funktion verwendet. Vorteilhaft ist insbesondere, dass sie nicht sättigen kann und in der Praxis schneller als Sigmoid und Tanh Funktionen konvergiert. Der Unterschied zwischen ReLu und Leaky ReLu ist, dass mit Leaky ReLu das Neuron, das negative Werte als Eingabe hat, einen kleinen initialen positiven Wert am Ausgang bekommt. In Folge werden die mit dem Neuron verbundenen Gewichte immer aktualisiert und sterben nicht [11, pp. 32, 35].

3.3 Convolutional Neural Networks

Vollständig verbundene Feed-Forward neuronale Netze werden in der Regel zum gleichzeitigen Lernen von Merkmalen und zum Klassifizieren von Daten verwendet. Das Hauptproblem bei der Verwendung eines vollständig verbundenen neuronalen Netzwerks bei Bildern besteht darin, dass die Anzahl der Neuronen für Deep Learning Architekturen sehr hoch sein kann. Dies macht vollständig verbundene neuronale Netzwerke für die Anwendung auf Bildern unpraktisch macht. Zum Beispiel ist das in der Abbildung 3.7 dargestellte 16×16 Graustufenbild mit einer versteckten Schicht aus 7200 Neuronen verbunden. Wie in dieser Grafik angegeben wird, kann das Bild als ein 16 × 16 = 1024-dimensionaler Vektor betrachtet werden. Das erste Neuron in der verborgenen Schicht ist mit 1024 Einheiten in der Eingabe verbunden. In ähnlicher Weise sind auch andere Neuronen mit jeder Einheit des Eingabebildes verbunden. Folglich wird diese vollständig verbundene Schicht unter Verwendung von 1024

× 7200 = 7.372.800 verschiedenen Parametern formuliert. Zur Reduzierung der Anzahl der Parameter muss die Anzahl der Neuronen in der versteckten Schicht reduziert werden, infolgedessen wird die Klassifizierungsleistung schlechter [12, pp. 85, 86].

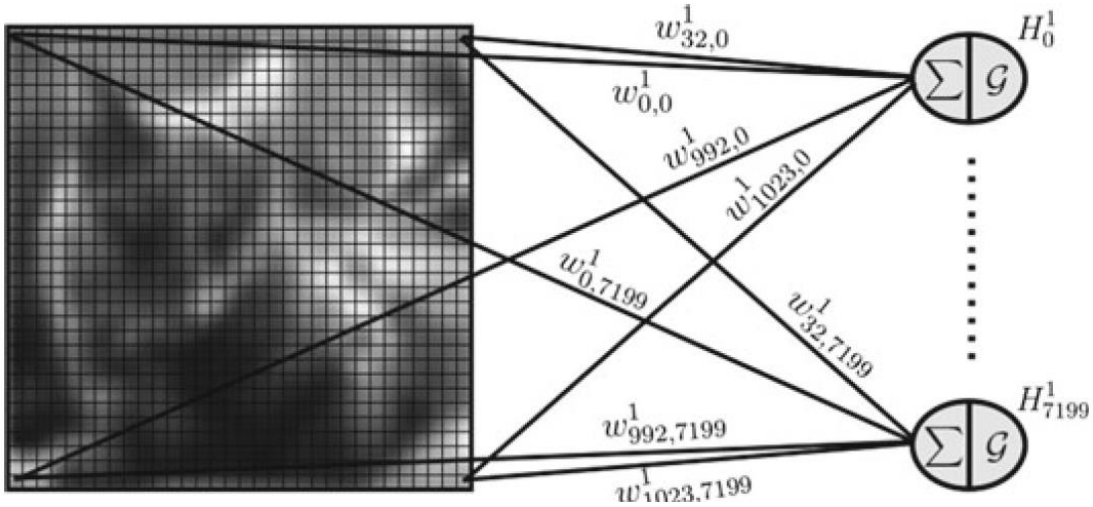


Abbildung 3.7: Ein Beispiel eines 16x16 Graustufenbilds mit einer versteckten Schicht, die aus 7200 Neuronen besteht, vollständig verbunden [12, p. 86]

Die Grundidee hinter dem Convolutional neuronalen Netzwerk (CNN) (engl. Convolutional Neural Networks) besteht darin, eine Lösung für die Reduzierung der Anzahl von Parametern zu entwickeln. Diese ermöglicht einem Netzwerk, mit viel weniger Parametern an Tiefe zu gewinnen. Die Faltung ist eine der wichtigsten Operationen in der Signal- und Bildverarbeitung. In einer Faltung werden die Eingabedaten mit einem sogenannten Faltungsfilter bearbeitet, damit in dem Resultat bestimmte Merkmale hervorgehoben oder unterdrückt werden [13, p. 27]. Mathematisch besteht die Faltung aus zwei Grundoperationen: Zum Ersten aus der Berechnung des Skalarprodukts zwischen den Filtern und dem überlappenden Teil des Bildes. Zum anderen aus der Ermittlung der höchsten Überlappung aus dem Skalarprodukt, die aus jeder Überlappung des Filters und des Eingangsbildabschnitts berechnet werden [14, pp. 182, 185]. Diese wird durch die folgende Gleichung berechnet

$$output = \sum input * Kernel \tag{6}$$

Das folgende Beispiel wird die Operation besser erklären. Wenn ein 4x3 Bild mit 2x2 Filter wie in Abbildung 3.8 gefaltet wird, wird zuerst das Skalarprodukt bei der ersten

Überlappung auf der linken Ecke berechnet. Anschließend wird es summiert und in einer neuen Matrix hinzugefügt und so weiter. Die Dimensionale der neuen Matrix werden durch die folgenden Gleichungen berechnet [15, p. 12]:

$$x_{neu} = \frac{x_{Bild} - x_{Filter}}{Stride} + 1 = \frac{4-2}{1} + 1 = 3$$

$$y_{neu} = \frac{y_{Bild} - y_{Filter}}{Stride} + 1 = \frac{3-2}{1} + 1 = 2$$

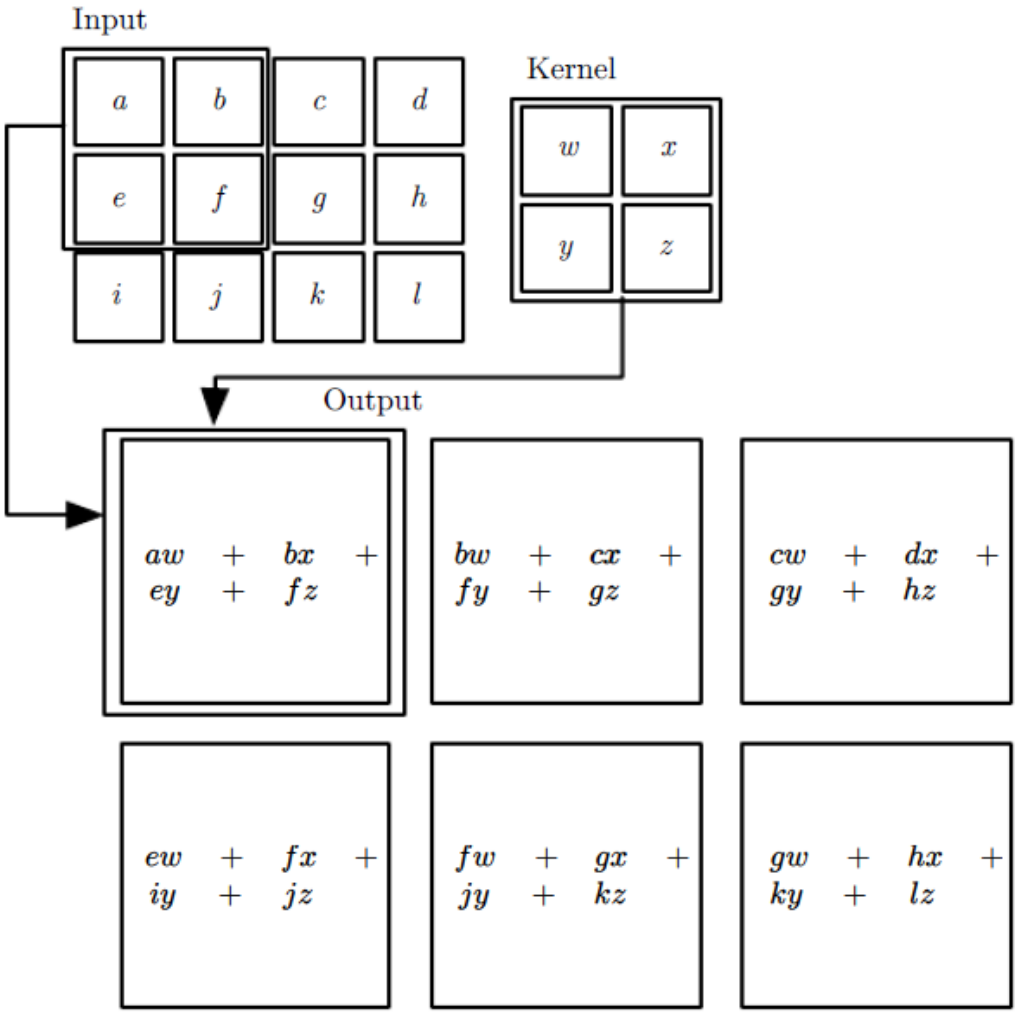


Abbildung 3.8: Ein Beispiel der Faltung von einem 4x3 Bild mit einem 2x2 Kernel und die Stride ist eins, sodass das Eingabebild durch den Kernel mit einem Stride-Wert gescannt wird und an jeder Stelle ein Skalarprodukt durchgeführt wird. Das Ergebnis der Faltung ist eine 2x3 Feature-Map. Quelle: [4, p. 334]

CNN besteht aus drei Schichten die Faltungsschicht, die ReLu Schicht und die Max-Pooling Schicht. Darüber hinaus stehen vollständig verbundenen Schichten am Ende,

die zur Klassifizierungsaufgabe dienen [16, p. 5]. Eine Faltungsschicht besteht aus einem Satz von erlernbaren Filtern. Jeder Filter hat die gleiche Tiefe der Eingabedaten. Zum Beispiel: Bei der ersten Schicht ist die Größe des Filters 5x5x3, hierbei 5 die Höhe und die Breite des Filters ist und 3 die Tiefe ist. Während des Vorwärtsdurchlaufs wird jeder Filter über die Breite und Höhe des Eingabebilder verschoben und das Skalarprodukt zwischen dem Filter und der Bildregionen berechnet. Im Folgenden wird eine 2D Aktivierungskarte erstellt, die die Response des Filters an jeder räumlichen Position angibt. Alle Neuronen in jedem Filter werden durch dieselben Gewichte mit Bildregionen verbunden, d.h. das Teilen der Gewischte (engl. weights sharing). Die Filter werden gelernt, wenn sie eine Art visuelles Merkmal auf der ersten Schicht bzw. komplexe Muster auf den höheren Schichten des Netzwerks erkennen. Bei Visuelle Merkmale kann es sich z.B. um eine Kante einer Orientierung oder einen Fleck einer Farbe handeln. Wenn ein Satz von Filtern (z.B. 50 Filter) verwendet wird, werden 50 Aktivierungskarten erzeugt, die die Ausgabe der Schicht sind [17]. Wenn es um das oben genannte Beispiel geht, wird die Anzahl der Parameter durch die Verwendung von 50 Filtern mit der Größe (5x5x1) stark reduziert, da jeder Filter 5x5x1 = 25 Gewichte verwendet und die gesamte Schicht durch 5x5x1x50 = 1250 Gewischte mit dem Bild verbunden wird [12, p. 87].

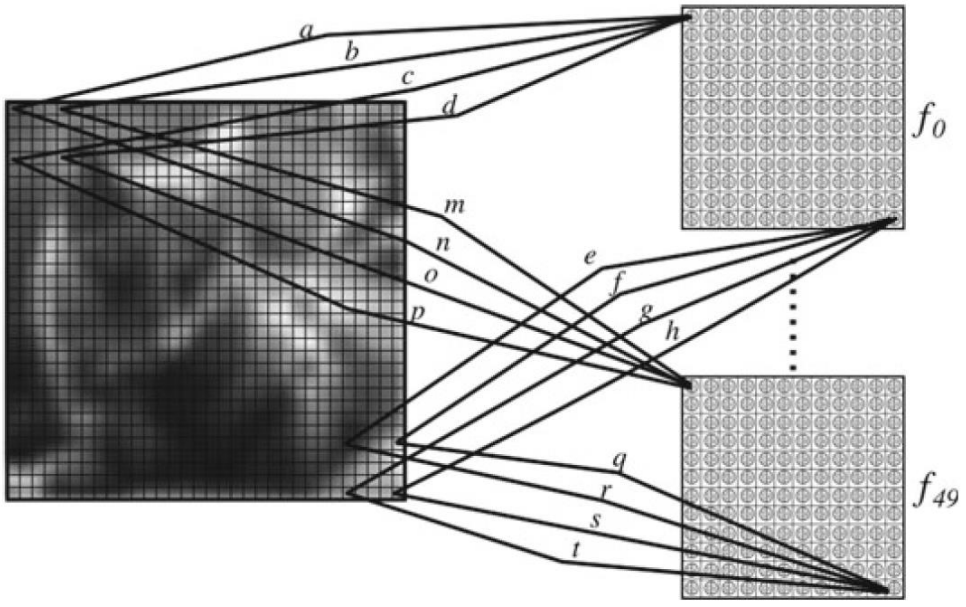


Abbildung 3.9: Ein Satz von 50 Filtern, die Neuronen beinhaltet, sodass jedes Neuron im Filter durch dieselben Gewichte mit allen Bildregionen verbunden wird. Quelle: [12, p. 87]

Die ReLu Schicht folgt auf die Faltungsschicht und ändert nicht die Größe der Eingabe. In dieser Schicht wird keine Parameter gelernt, sondern die ReLu Aktivierungsfunktion (3.2) implementiert. Das Ziel der Implementierung der Aktivierungsfunktion ist die Erhöhung der Nichtlinearität bei des Convolutional neuronalen Netzwerks [16, p. 11]. Die Max-Pooling Schicht wird zur Anpassung der Ausgabe von den Faltungs- und ReLu-Schichten verwendet. Eine Pooling-Funktion ersetzt die Ausgabe des Netzes an einem bestimmten Ort durch eine zusammenfassende Statistik der nahen gelegenen Ausgaben. Das hilft dabei, die Darstellung für kleine Eingaben ungefähr unveränderlich zu machen. Es gibt zwei bekannte Pooling-Funktionen, eine berechnet den maximalen Wert (Max-Pooling) und eine berechnet den durchschnittlichen Wert (Average-Pooling) [18, p. 4]. Es ist gut zu erkennen, dass die Pooling Schicht keine Parameter hat [16, p. 26].

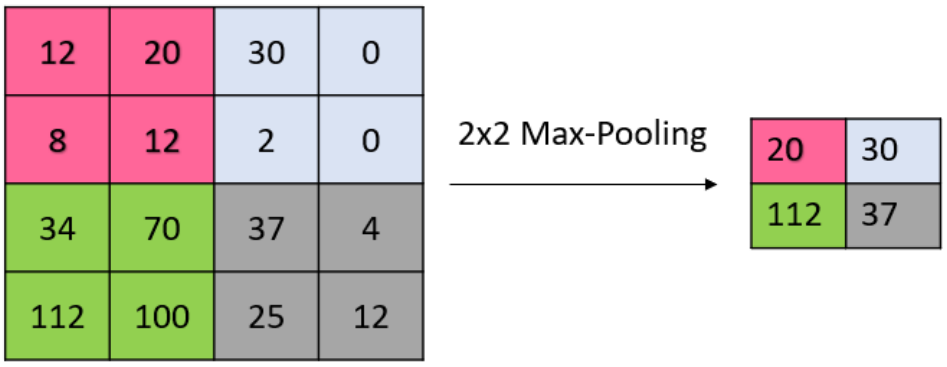


Abbildung 3.10: Max-pooling Operation auf einem Feature-Map (links) mit 2x2 Kernel. In jedem 2x2 Kernel von Feature-Map wird der maximale Wert ausgewählt und die Reste Werte werden weggelassen, sodass die Feature-Map-Größe reduziert wird und nur die maximalen Werten angezeigt werden. (einige Darstellung)

In der folgenden Abbildung wird die vollständige Architektur eines CNNs gezeigt.

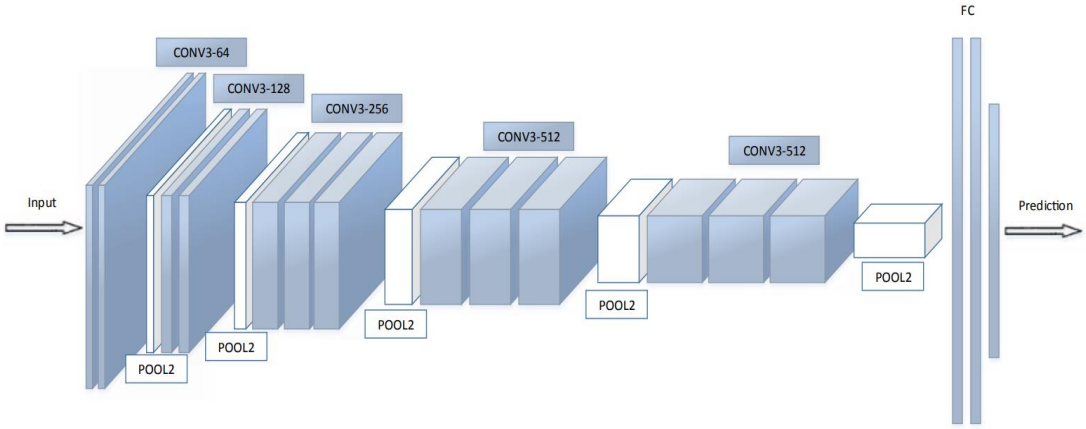


Abbildung 3.11: Architektur eines CNNs besteht aus fünf Faltungsschichten und drei Pooling Schichten und am Ende zwei vollständig verbundene Schichten. Die Faltungsschichten haben 64, 128, 256, 512 Filter aufeinanderfolgend. Die Pooling-Schichten reduzieren die Größe der Feature-Maps bis die Hälfte. Die vollständig verbundenen Schichten sind für die Klassifizierungsaufgabe verantwortlich. Quelle: https://www.omicsonline.org/open-access/face-verification-subject-to-varying-age-ethnicity-and-genderdemographics-using-deep-learning-2155-6180-1000323.pdf

3.4 Convolutional Autoencoder

Ein Autoencoder (AE) ist eine Lerntechnik, die eine unüberwachte vortrainierte Lernmethode zum Lernen benutzt. Eine Erklärung zu unüberwachten Lernmethode ist in Kapitel 3.6.2 zu finden. Er wird zur Reduzierung der Dimensionalen der Eingabedaten verwendet, um das überwachtes Lernen Stufe zu vereinfachen [19, p. 10]. Ein AE Modell besteht aus einem Encoder und einem Decoder. Im Encoder wird eine Eingabe in eine typische niedrig dimensionierte Darstellung umwandelt. Anschließend wird im Decoder die ursprüngliche Eingabe aus dieser Darstellung durch die Minimierung von Kostenfunktion rekonstruiert. Autoencoder werden in vielen Anwendungen verwendet. Dazu gehören unter anderem vortrainierte Netzwerke, das Extrahieren von Merkmalen und Gruppierungen [20, p. 1].

Beim Encoder: $$y = f(Wx + b) \quad (7)$$

Beim Decoder: $$z = f(W^T x + b') \quad (8)$$

Beim Decoder wird die transponierte Gewicht-Matrix beim Encoder zur Reduzierung der Anzahl der Parameter beim Lernen verwendet. W, b und b' werden durch die Minimierung der Verlustfunktion optimieret, um die rekonstruierten Daten ähnlich von

Eingabedaten zu sein. Die Verlustfunktion wird durch die folgende Gleichung berechnet:

$$L(x,z) = \frac{1}{2}(x-z)^2 \tag{9}$$

Nach dem Training wird der Decoder geschmissen und der Encoder wird als vorverarbeitungsschritt verwendet [19, p. 11]

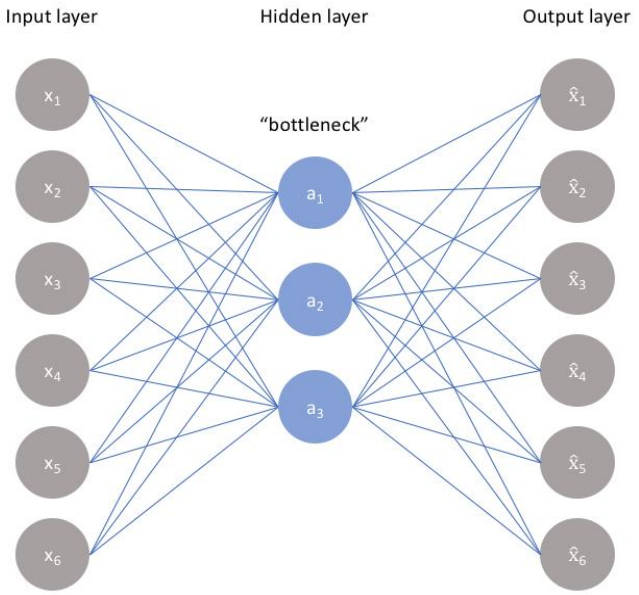


Abbildung 3.12: AE Struktur besteht aus Encoder und Decoder. Der Encoder besteht aus der Eingabeschicht $(x_1, x_2, .., x_6)$ und der versteckten Schicht (a_1, a_2, a_3), die zusammen vollständig verbunden sind. Der Decoder besteht aus der versteckten Schicht (a_1, a_2, a_3) und der Ausgabe $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_6)$, die zusammen vollständig verbunden sind. Die Ausgabeschicht hat die gleiche Größe der Eingabeschicht. Quelle: https://www.jeremyjordan.me/autoencoders/

Ein bekannter Typ von Autoencoder ist der Convolutional Autoencoder (CAE), der aus Faltungsschichten im Encoder und symmetrischen Entfaltungsschichten im Decoder besteht (siehe Abbildung 3.13). Es wird nach jeder Faltungsschicht und Entfaltungsschicht eine ReLu Aktivierungsfunktion implementiert. Die Eingabedaten und die Ausgabedaten von CAE haben die gleichen Dimensionalen. Die Grundidee beim CAE ist, dass der Encoder durch die Faltungsschichten die Merkmale des Bilds extrahiert und der Decoder durch die Entfaltungsschichten das Bild wieder rekonstruiert [21, p. 3].

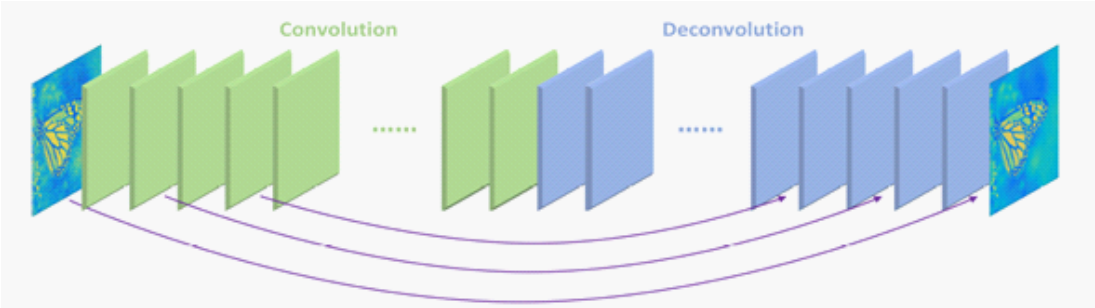


Abbildung 3.13: eine Darstellung einer CAE Architektur besteht aus Faltungsschichten beim Encoder und Entfaltungsschichten beim Decoder. Es wird Verbindungen zwischen den entsprechenden Faltungs- und Entfaltungsschichten im tiefen Netzwerk hinzufügt, um die verlorenen Information während der Faltung zu ersetzen und das Bild im tiefen Netzwerk gut zu rekonstruieren [21, p. 4]

In dieser Arbeit wird die Faltungsschicht zum Down-Sampling und die Entfaltungsschicht zum Up-Sampling verwendet. Es wird bemerkt, dass diese Architektur nur mit wenigen Schichtenanzahl arbeiten kann. Das liegt daran, dass im tiefen Netz viele Informationen durch die Faltung verloren werden und durch die Entfaltung nicht rekonstruiert werden können. Außerdem leiden tiefe Netzwerke oft unter dem Verschwinden von Gradienten und es wird schwieriger sie zu trainieren. Zum Lösen dieser Probleme wird Verbindungen-Überspringen (engl. Skip Connections) zwischen den entsprechenden Faltungs- und Entfaltungsschicht hinzufügt wie in der Abbildung 3.13 dargestellt. Hierbei enthält die Feature-Maps, die von Verbindungen-Überspringen übergeben werden, viele Bilddetails, die Entfaltung zur Wiederherstellung des Bildes unterstützt [21, p. 4].

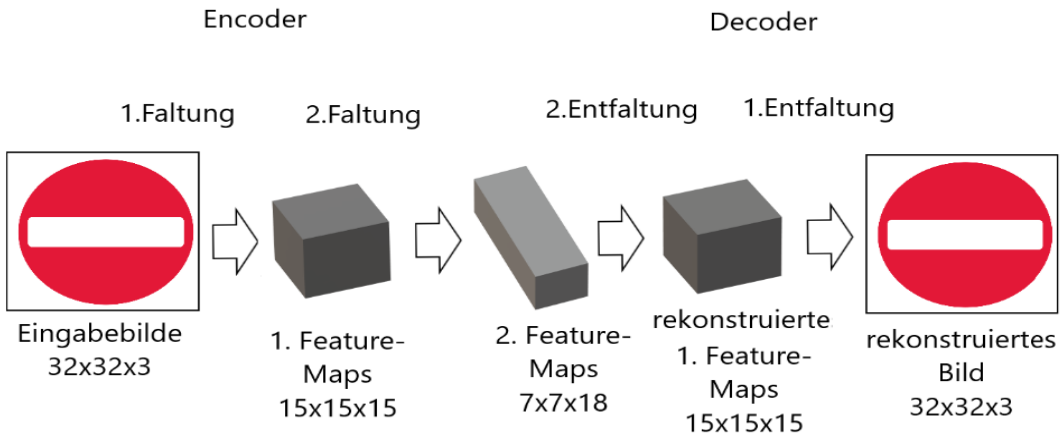


Abbildung 3.14: Ein Beispiel von zwei CAE Schichten (CAE01: Filter 3x3, Filteranzahl = 15, Stride = 2, CAE02: Filter 3x3, Filteranzahl = 18, Stride = 2). Die Eingabe ist ein 32x32 farbiges Bild. Nach der ersten Faltungsschicht liefert 15x15x15 Feature-Maps, die in einer zweiten Faltungsschicht eingeführt wird und liefert 7x7x18 Feature-Maps. Aus der zweiten resultierenden Feature-Maps wird die ersten Feature-Maps rekonstruiert, dann die ursprüngliche

Bild wieder rekonstruiert. Die Ausgabe des Decoders ist das rekonstruierte Bild (32x32x3). (eigene Darstellung)

3.5 Restricted Boltzmann Machine (RBM)

Boltzmann Machines (BM) werden entwickelt, um die komplexe Statistik beliebiger Systeme zu erfassen. Hierzu werden Neuronen in sichtbare (engl. visible) und versteckte (engl. hidden) Einheiten unterteilt. Die Einheiten der beiden Untergruppen werden anschließend vollständig miteinander verbunden. Die neuesten Merkmalsdetektoren und Klassifikatoren implementieren einen bestimmten Typ von BM, die Restricted Boltzmann Machine (RBM), aufgrund ihrer effizienten Lernalgorithmen [22, p. 2].

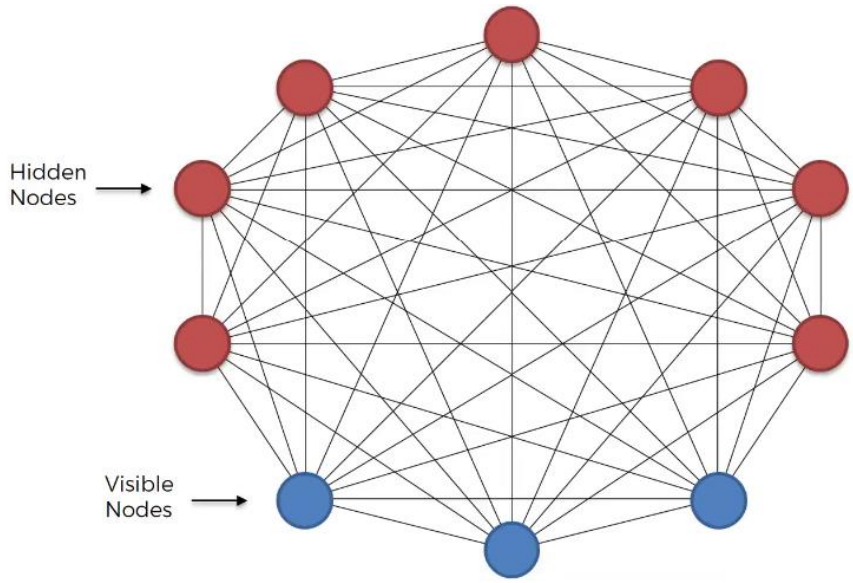


Abbildung 3.15: Boltzmann Maschine besteht aus versteckten und sichtbaren Einheiten. Die versteckten Einheiten (rot) sind symmetrisch miteinander und mit den sichtbaren Einheiten (blau) verbunden. Quelle: https://miro.medium.com/max/1200/1*Ere0a83PN-Rj7DF5_IVZdg.png

Die Restricted Boltzmann Machine (RBM) ist eine spezifische Art von neuronalen Netzwerken und wird zur Klassifizierung und Merkmalserkennung beim maschinellen Lernen eingesetzt. RBM hat eine einzelne Schicht von versteckten Einheiten, die nicht miteinander verbunden sind und symmetrische Verbindungen zu einer Schicht

von sichtbaren Einheiten haben. Diese Einheiten sind in der Lage, ein generatives Modell der Daten effizient zu erlernen [22, p. 1]. Die sichtbaren Einheiten codieren die Eingabedaten $x = (x_1, \dots, x_v)^T$ in einem Vektor und die versteckten Einheiten $y = (y_1, \dots, y_h)^T$haben immer eine binäre Werte zwischen [0,1] [23, p. 2]. RBM ist eine effektive Methode, um High-level-Features aus verschiedenen Daten zu extrahieren und wird bei der Spracherkennung verwendet [24, p. 1]. RBMs werden in der Regel mit dem kontrastiven Divergenz-Lernverfahren trainiert. Hier müssen die Parameter wie Lernrate, Momentum, Gewichtskosten, Sparsamkeitsziel, Anfangswerte der Gewichte, Anzahl der versteckten Einheit und Größe der einzelnen Mini-Batches eingestellt werden [25, p. 3].

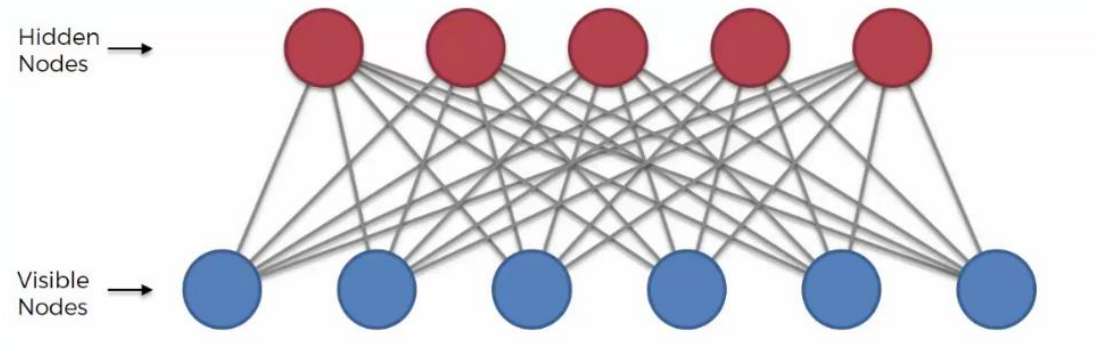


Abbildung 3.16: RBM besteht aus fünf versteckten Einheiten (rot) und sechs sichtbaren Einheiten (blau). Die Einheiten einer Schicht sind nicht miteinander verbunden und symmetrisch mit der anderen Schicht verbunden. Quelle: https://miro.medium.com/max/1200/1*LoeBW9Stm6HjK57yBp45sQ.png

Die gesamte Energie der sichtbaren, versteckten Schichten (v, h) wird wie folgend beschrieben:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \qquad (10)$$

v_i, h_j sind die binären Zustände der sichtbaren Einheit i und der versteckten Einheit j

a_i, b_j sind die Baises der sichtbaren Einheit i und der versteckten Einheit j

w_{ij} ist die Gewichte zwischen den Einheiten i und j

3.6 Lernverfahren

Ein maschinelles Lernverfahren ist ein Verfahren, welches aus Daten lernen kann. Lernen ist der Prozess, bei dem das Programm aus einer Erfahrung in Bezug auf Aufgabengruppen und ein Leistungsmaß lernt. Dabei steigert die Leistung des Programms, wenn seine Leistung sich bei den Aufgaben durch die Erfahrung anhand von Leistungsmaß verbessert [4, p. 100]. Es existieren verschiedene Methoden für das Lernen in neuronalen Netzen, z. B. das überwachte Lernen (engl. supervised Learning), das bestärkende Lernen (engl. reinforcement Learning) und das unüberwachte Lernen (engl. unsupervised Learning) [26, p. 34]. In dieser Arbeit wird am Anfang eine unüberwachtes Lernmethode durch Convolutional Autoencoder verwendet, wobei anschließend für die Klassifizierungsaufgabe eine überwachte Lernmethode verwendet wird.

3.6.1 Überwachtes Lernen

Überwachtes Lernverfahren basiert darauf, gelabelte Datensätze zu trainieren, in denen die korrekten Ergebnisse bereits zugeordnet sind. Das überwachte Lernverfahren wird in Feed-Forward und MLP (engl. Multilayer perceptron) Modellen angewendet. Das Netz wird trainiert, indem der Algorithmus auf Basis des Fehlersignals, also der Unterschied zwischen berechneten und gewünschten Ausgabewerten, die synaptischen Gewichte der Neuronen anpasst. Das synaptische Gewicht ist proportional zum Produkt des Fehlersignals und der Eingabeinstanz des synaptischen Gewichts [26, p. 35]. Dieses Lernverfahren hat zwei Hauptaufgaben, die Klassifizierung und die Regression.

3.6.2 Unüberwachtes Lernen

Im Gegensatz zum überwachten Lernen ist beim unüberwachten Lernen keine menschliche Arbeitskraft erforderlich, sodass im Lernprozess ungelabelte Datensätze verwendet werden. Dieses Lernverfahren wird am meisten mit Dichteschätzung, Lernen aus einer Datenverteilung oder Gruppieren der Daten in Gruppen verwendet [27, p. 1]. Außerdem kann die unbegrenzte Anzahl der ungelabelten Bilder und Videos genutzt werden, um gute Darstellungen von Merkmalen zu erlernen, die für Lernaufgaben wie Bildklassifizierung verwendet werden können. Beispiele vom unüberwachten Lernmethoden sind Autoencoder, RBM (engl. Restricted Boltzmann Machine), K-means Clustering und Gaussian mixtures [28, pp. 215, 217].

3.6.3 Bestärkendes Lernen

Das bestärkendes Lernen ist eine Lernverfahren, das auf einen Agenten in einer dynamischen Umgebung angewendet wird. Dieser Agent kann verschiedene Aktionen zum Erreichen seines Ziels durchführen. So werden nicht wie bei anderen Algorithmen eine Problemstellung und eine Reihe durchzuführender Aktionen definiert. Stattdessen muss das Netz diejenigen Aktionen bestimmen, die den besten Erfolg versprechen [29, p. 3].

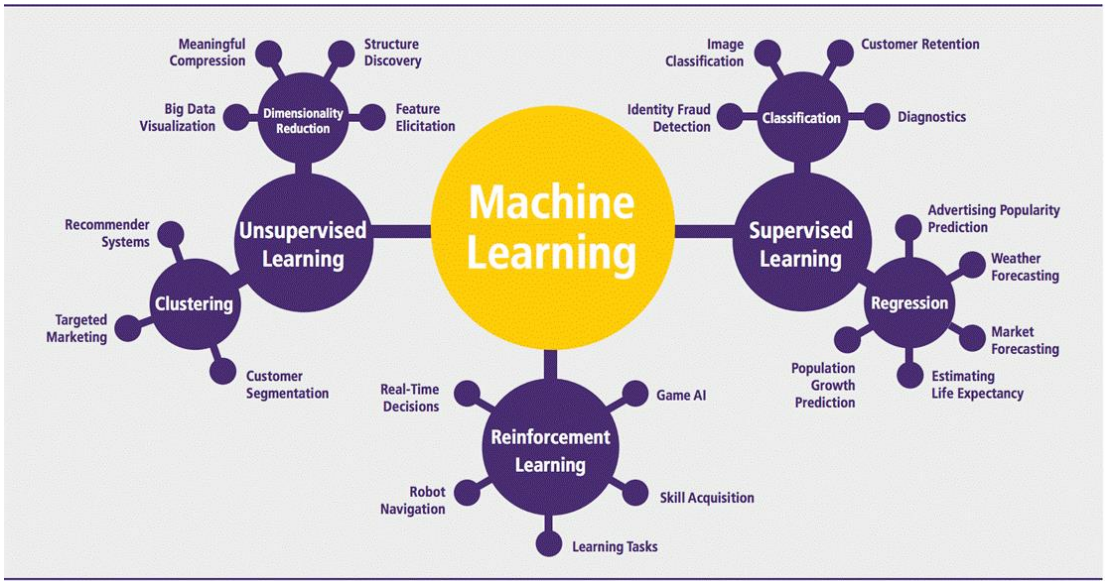


Abbildung 3.17: Eine Darstellung der Bereiche, bei denen ML verwendet werden kann. Die Bereiche werden in Bezug auf die verwendeten Lernalgorithmen gruppiert. Unter überwachten Lernalgorithmus stehen die Klassifizierungs- und die Regressionsaufgabe. Unter unüberwachten Lernalgorithmus stehen die Gruppierung (engl. Clustring) und das Dimensionalitätsreduktion. Unter bestärkenden Lernalgorithmus steht die Navigationsaufgabe des Robots. Quelle: https://www.guru99.com/machine-learning-tutorial.html

3.7 Klassifizierung mittels Support Vector Maschine (SVM)

3.7.1 Einleitung

Die Klassifizierung von Mustern hat das Ziel, ein Objekt in einer der angegebenen Klassen zu klassifizieren. Eingaben des Klassifikators sind die Merkmale, die die verschiedenen Klassen gut darstellen können [30, p. 1]. Die Klassifizierung besteht aus zwei Hauptkomponenten: einer Score-Funktion, die die Rohdaten auf Klassenwerte abbildet und durch die folgende Gleichung den Score jeder Klasse berechnet wird:

$$score = Wx + b \tag{11}$$

und einer Verlustfunktion, die den Unterschied zwischen den vorhergesagten Werten und den Echtheitslabels berechnet [31]. Für einen Datensatz $\{(x_i, y_i)\}_i^N$ kann der Verlust in verschiedenen Methoden berechnet werden:

Der quadratische Verlust
$$L_i = (s_j - s_{y_i})^2 \tag{12}$$

Der absolute Verlust
$$L_i = (s_j - s_{y_i}) \tag{13}$$

Der Hinge Verlust
$$L_i = \begin{cases} 0 \; wenn \; s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 \; ansonst \end{cases} \tag{14}$$

Wobei s_{y_i} ist die Score des richtigen Klasse und s_j ist die berechnete Score der aktuellen Eingabe [32, pp. 5-6]. Softmax und Support Vector Machine (SVM) sind die bekanntesten Klassifikatoren, die bei Klassifizierungsaufgabe verwendet werden. In dieser Arbeit wird SVM verwendet. Eine SVM ist ein mathematisches Verfahren der Mustererkennung und gehört zum überwachten Lernalgorithmus, der Daten in verschiedenen Klassen klassifiziert [30, p. 1]. SVM Klassifikatoren sind in vielen Erkennungsaufgaben sehr erfolgreich und besitzen eine ausgezeichnete Genauigkeit bei verschiedenen Klassifizierungsaufgaben [33, p. 1]. Eine SVM ist effektiv in hochdimensionalen Räumen, weil verschiedene Kernelfunktionen für die Entscheidungsfunktion verwendet werden können, und wirksam in Fällen, in denen die Anzahl der Dimensionen größer als die Anzahl der Samples ist. Darüber hinaus verwendet SVM eine Teilmenge von Trainingspunkten in der Entscheidungsfunktion (sogenannte Support-Vektoren), so dass sie auch speichereffizient sind [34]. Anderseits stellt SVM einige Schwierigkeiten beim Erlernen tieferer hierarchischer Merkmale dar, die durch Bilddatensätze dargestellt werden. Darüber hinaus kann die große Anzahl der Attribute, wenn das Bild aus einer großen Mange von Rohpixeln besteht, zu Skalierungsproblem führen [33, p. 1].

3.7.2 Binare SVM

Das Hauptziel der SVM ist, den gegebenen Datensatz mittels einer Hyperebene mit dem maximal möglichen Abstand zwischen Klassen zu trennen. Die Hyperebene wird mit dieser Gleichung beschrieben:

$$y = w^T x + b \tag{15}$$

wobei w die Gewichte und b Bias ist. Die Richtung der Hyperebene wird durch w bestimmt [35, pp. 11 - 13].

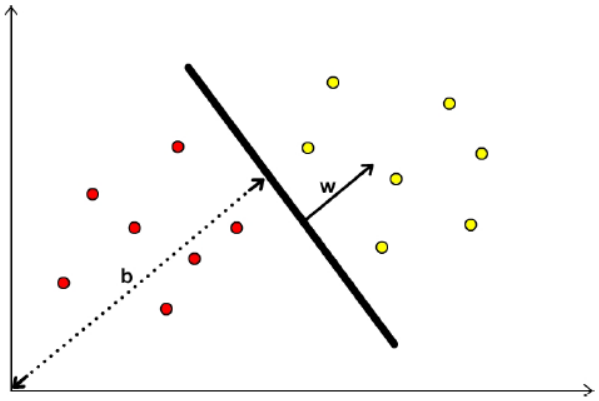


Abbildung 3.18: Eine Hyperebene im Fall der linearen Klassifizierung, sodass die Hyperebene zwischen den gelben markierten Mustervektoren und den roten trennt

Der Abstand von der Hyperebene zum nächsten Trainingsbeispiel ist die Marge des Klassifikators. Die Marge eines Datensatzes ist das Minimum der Marge aller seiner Samples [36, p. 6]. Für einen Datensatz (x_i, y_i), wobei $i = 1, \ldots, l$ und $y_i \in \{-1, +1\}$, kann der Klassifikator durch die folgende Gleichung beschrieben werden:

$$h_{w,b}(x) = f(w^T x_i + b) \tag{16}$$

f ist eine Entscheidungsfunktion und gibt für jede Sample einen Score pro Klasse an [35, p. 13].

Im Fall binären Klassifikator

$$f(z) = \begin{cases} +1 \; wenn \; z \geq 0 \\ -1 \; wenn \; z < 0 \end{cases} \tag{17}$$

Aus den Gleichungen (16) und (17) können die folgenden Gleichungen abgeleitet werden:

$$\begin{cases} w^T x_i + b \geq 0 \;\; wenn \;\; y_i = +1 \\ w^T x_i + b \leq 0 \; wenn \;\; y_i = -1 \end{cases} \tag{18}$$

Die Support-Vektoren sind die Datenpunkte, die der Hyperebene am nächsten liegen. Infolgedessen kann die Support-Vektoren mit dieser Gleichsetzung berechnet werden:

$$y_i(w^T x_i + b) \pm 1 = 0 \tag{19}$$

In der folgenden Abbildung werden die Support-Vektoren gezeigt, die der kleinste Abstand zur Hyperebene haben und die maximale Marge erreichen.

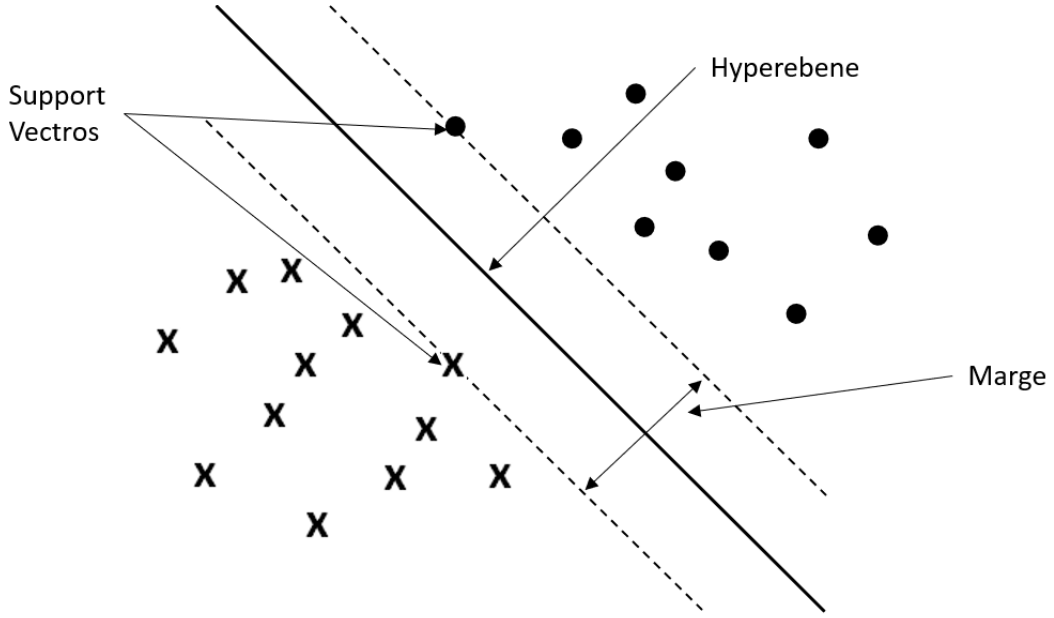


Abbildung 3.19: Eine Darstellung einer binaren SVM, die zwei Klassen (Malzeichen und Kreis) klassifiziert. Zwei Support Vektoren, die der kleinste Abstand zur Hyperebene haben und die maximale Marge erreichen. (einige Darstellung)

Zum Finden des optimalen Marge-Klassifikators müssen die optimalen w und b gefunden werden, sodass die Marge maximal ist und die Gleichung (19) mitberücksichtigt wird [37, pp. 8-10]. Wie vorher erwähnt ist, ist die Marge der Abstand zwischen die Support-Vektoren, d.h. Die Breite der Margen wird anhand der Gleichung (18) berechnet:

$$Bereit = ((w^T x_i + b + 1) - (w^T x_i + b - 1)).\frac{||w||}{w} = \frac{2}{w} \tag{20}$$

Um eine maximale Marge zu erreichen, muss die Breite maximal sein. D.h. die Gewichte müssen minimal sein, was in der Trainingsphase durch die Implementierung der folgenden Kriterien realisiert werden kann:

$$min_{w,b}\ \frac{1}{2}||w||^2$$
$$y_i(w^T x_i + b) \geq 1\ ; i = 1,..,m \tag{21}$$

Diese Optimierungskriterien können mittels der Lagrange-Methode in der Dual-Form umgewandelt werden, wobei α_i der Lagrange-Multiplikator und $\alpha_i \geq 0$ ist:

$$L(w,b,\alpha) = \frac{1}{2}\left||w|\right|^2 - \sum_{i=1}^{n} \alpha_i[\ y_i(w^T x_i + b) - 1] \tag{22}$$

Nach der Ableitung dieser Gleichung anhand von w und b wird die optimale w gefunden:

$$\frac{\partial L}{\partial w} = 0 \rightarrow\ w = \sum_{i=1}^{n} \alpha_i\ y_i x_i \tag{23}$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \alpha_i\ y_i = 0 \tag{24}$$

Dann ist der Lagrange aus der Gleichung (22) [35, pp. 20-22]:

$$L(w,b,\alpha) = \sum_{i}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i)^T x_j \tag{25}$$

Das führt zum Dual-Optimierungsproblem, das durch die folgenden Gleichungen beschrieben werden:

$$max_\alpha\ W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i)^T x_j$$
$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_\mathrm{i}\ \mathrm{y_i} = 0 \tag{26}$$
$$0 \leq \alpha_\mathrm{i}\ ; i = 1, \ldots., n$$

3.7.3 Soft Margin SVM

Wenn es Sonderfälle im Datensatz gibt, kann der Algorithmus entweder keine Hyperebene zum Trennen der Daten finden, oder er findet eine Hyperebene mit einer kleinen Marge. Dies führt zu einem harten Optimierungskriterium, weshalb die oben genannte Methode als Hart Margin SVM genannt wird. Wird jedoch ein Regularisierungsparameter am Optimierungskriterium hinzugefügt, kommt es zu einer Minimierung der Empfindlichkeit des Algorithmus gegenüber Sonderfällen und zum Erreichen einer maximalen Marge. Diese Vorgehensweise wird als Soft Margin SVM bezeichnet [30, p. 29]. Als Folge werden die Optimierungskriterium wie folgend beschrieben:

$$\begin{gathered} min_{w,b}\ \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ y_i(w^T x_i + b) \geq 1 - \xi_i\,; i = 1,..,m \\ \xi_i > 0; i = 1,..,m \end{gathered} \tag{27}$$

Wobei ξ_i die Slack-Variablen sind, die den Abstand des Punkts zu ihrer Marge-Hyperebene messen. Der Regularisierungsparameter C ist verantwortlich für das Ausgleichen zwischen der Minimierung von $\frac{1}{2}||w||^2$ und dem Trennen zwischen den meisten Beispielen mit einer Marge, die mindesten gleich 1 ist [35, pp. 24- 25].

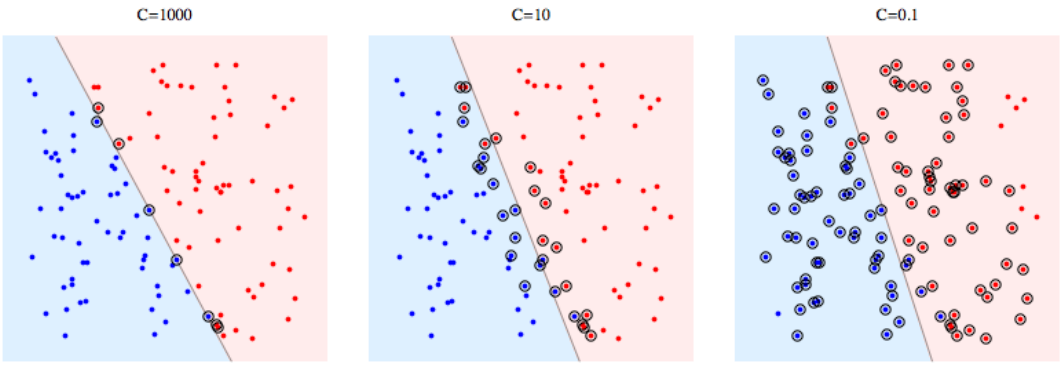


Abbildung 3.20: Beispiele von Soft Margin SVM mit verschiedenen Werten von C Parameter. Wenn C groß ist, ist das Optimierungskriterium hart und die Marge kleiner. Wenn C klein ist, ist das Optimierungskriterium soft und die Marge großer. Quelle: https://i.stack.imgur.com/0aYO8.png

3.7.4 Kernel SVM

Einige Fälle können mit der linearen Hyperebene nicht gelöst werden, wie in der Abbildung 3.21 links. In einer solchen Situation verwendet SVM einen Kernel-Trick, um die Eingabedaten in einem höherdimensionalen Raum zu transformieren, in dem sie dann wieder linear separierbar sind. Die Datenpunkte werden auf der x-Achse und der z-Achse dargestellt, wobei z die quadrierte Summe aus x und y ist ($z = x^2 + y^2$), dann können diese Punkte einfach durch lineare Hyperebene getrennt werden (Abbildung 3.21 rechts).

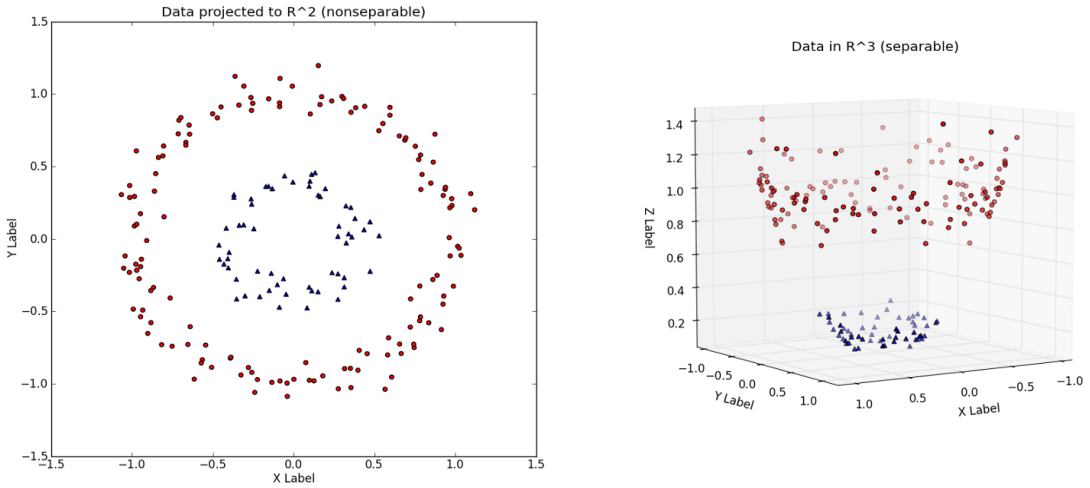


Abbildung 3.21: Ein Datensatz besteht aus zwei Klassen, die linear nicht separierbar. (Links) Die Darstellung des Datensatzes in 2-Deminsional (x,y). (Rechts) Die Darstellung des Datensatzes in 3-Deminsional (x, y, z). Quelle: https://medium.com/@vivek.yadav/why-is-gradient-descent-robust-to-non-linearly-separable-data-a50c543e8f4a

Ein Vorteil von SVM ist, dass sie einfach von Linearen zu Kernel wechseln kann, wobei die allgemeinen Formeln beibehalten werden. Aber anstatt der Arbeit an den Punkten $(x_1, x_2, \ldots, x_l)$ arbeitet sie an den Mapping-Funktionen $(\phi(x_1), \phi(x_2), \ldots, \phi(x_l))$, wobei ϕ Funktion die ursprünglichen p-dimensionalen Punkte X in einen m-dimensionalen Merkmalsraum X^* abbildet [38, p. 22]. Das innere Produkt der Mapping-Funktionen wird durch die Kernelfunktion definiert:

$$K(x_i, x_j) = \langle \emptyset(x_i), \emptyset(x_j) \rangle \tag{28}$$

Infolgedessen kann das Dual-Optimierungsproblem in der Gleichung (26) mit dem Kernel beschrieben:

$$max_{\alpha}\, W(\alpha) = \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\sum_{i=1}^{n} \alpha_i\, y_i = 0 \tag{29}$$

$$0 \leq \alpha_i \leq C$$

Der Kernel-Trick kann durch verschiedene Funktionen implementiert. Nachfolgend werden die bekannten Kernelfunktionen beschrieben:

Lineares Kernel $$K(x, x') = x^T . x' \tag{30}$$

Polynom Kernel $$K(x, x') = (x^T . x' + 1)^d \tag{31}$$

Radiale Basisfunktion Kernel (rbf) $$K(x, x') = exp(-\gamma . (x - x')^2) \tag{32}$$

Wobei d die Degree und γ ein positiver Parameter zur Steuerung des Radius ist [30, pp. 33-35]. Darüber hinaus wird der Klassifikator in der Gleichung (16) im Fall Kernel-SVM mit der folgenden Gleichung berechnet [35, p. 23]:

$$h_{w,b}(x) = f(\sum_{i=1}^{n} \alpha_i\, y_i\ K(x_i, x_j) + b) \tag{33}$$

3.7.5 Mehrklasse SVM

Der SVM Algorithmus wird nicht nur bei binären Klassifizierungsaufgaben verwendet, sondern auch bei Multiklasse-Klassifizierungsaufgaben. Dafür gibt es zwei unterschiedlichen Methoden. Zum einem One-Vs-All (OVA), zum anderem One-Vs-One (OVO). Für N-Klassen bei OVA, wobei $N > 2$ ist, werden N binäre Klassifikators verwendet und beim Training von der i SVM wird die i Klasse als positive Klasse zugeordnet und der Rest als negative Klasse zugeordnet. Dies Method hat den Nachteil, wenn die Anzahl der Trainingsbeispiele sehr groß ist, sodass das Training schwierig ist. Für N-Klasse bei OVO wird $N * (N - 1)\ /2$ binäre Klassifikator verwendet. Jeder binäre Klassifikator wird die positiven Beispiele als die erste Klasse und die negativen Beispiele als die zweite Klasse trainiert. Der Nachteil bei dieser Methode ist die große Anzahl der Klassifikatoren [39, p. 415]. In dieser Arbeit wird

die OVA Methode verwendet, sodass die richtige Klasse einen größeren Score als die anderen Klassen aufweist [6, p. 31].

$$y(x_j) = argmax(f(\sum_{i=1}^{n} \alpha_i \, y_i \; K(xi, xj) + b) \tag{34}$$

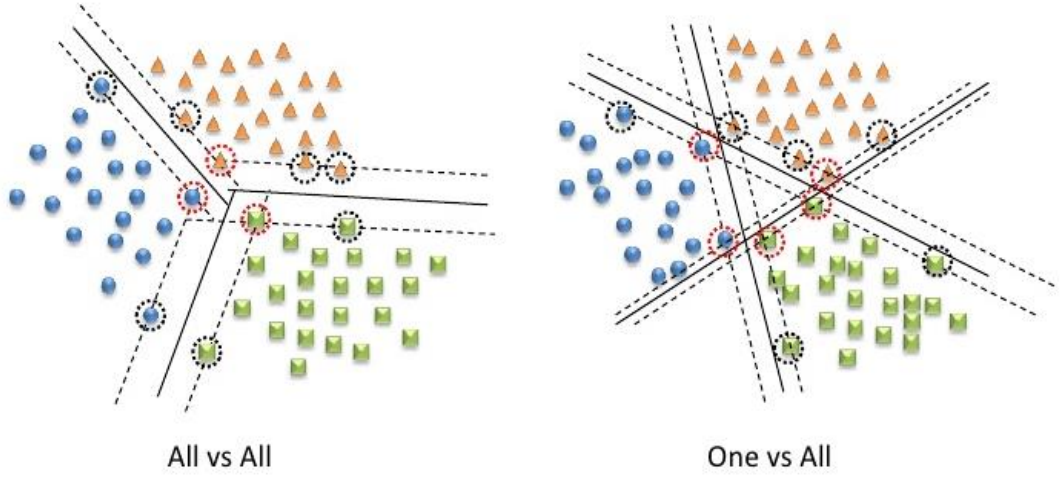


Abbildung 3.22: Mehrklasse SVM. (Links) OVO Verfahren, wobei jede Klasse gegen eine Klasse klassifiziert wird. Dies Prozess wiederholt sich N*(N-1) /2 Mal. (Rechts) OVA Verfahren, wobei jede Klasse gegen alle Klassen klassifiziert wird. Dies Prozess wiederholt sich N Mal. Hierbei ist N die Klassenanzahl. Quelle: https://image.slidesharecdn.com/gpgpu3slides-110104125237-phpapp02/95/multiclass-classification-using-massively-threaded-multiprocessors-31-728.jpg?cb=1294302662

3.8 Lernalgorithmus

3.8.1 Backpropagation

Backpropagation ist ein Algorithmus zum Trainieren von tiefen neuronalen Netzen beim überwachten Lernen. Der Algorithmus optimiert die Gewichte in einem neuronalen Netz schrittweise unter Verwendung von Trainingseingaben und den dazugehörigen gewünschten Ausgaben. In jeder Iteration wird dabei eine Trainingsprobe in das Netzwerk eingegeben und im ersten Schritt vorwärts durch das Netzwerk bis zur Ausgabeschicht propagiert. Ausgehend von einer Fehlerfunktion wird dann ein Gradientenabstieg entlang derselben durchgeführt und der Fehler minimiert. Als Fehlerfunktion kann beispielweise die quadratischer Fehlerfunktion in der Gleichung (35)

verwendet werden. Hierbei ist (x, y) eine Trainingsprobe und $(\hat{y})$ die Ausgabe des neuronalen Netzwerks, wenn (x) die Eingabe ist.

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \tag{35}$$

Der Gradientenabstieg basiert auf der Idee, dass der Wert der Fehlerfunktion sich verringert, wenn die Gewichte in Richtung des negativen Gradienten der Fehlerfunktion angepasst werden. Die negative Steigung der Fehlerfunktion wird gefolgt, um ein Minimum zu erreichen. Die Gewichte in jeder Schicht werden dabei jeweils mithilfe des zurückpropagierten Fehlers aus der vorhergehenden Schicht angepasst. Die Abbildung 3.23 zeigt eine einfache Darstellung des Algorithmus. Zur Berechnung der neuen Gewichte wird also die abgeleitete Fehlerfunktion mit der Lernrate α gewichtet und dann von den alten Gewichten subtrahiert Gleichung (36), wobei die Lernrate dabei die Geschwindigkeit des Gradientenabstiegs steuert. Wobei kann die Lernrate zur Überanpassung (engl. Over Fitting) führen, wenn sie so klein ist [4, pp. 204-224].

$$w_{ij} = w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}} \tag{36}$$

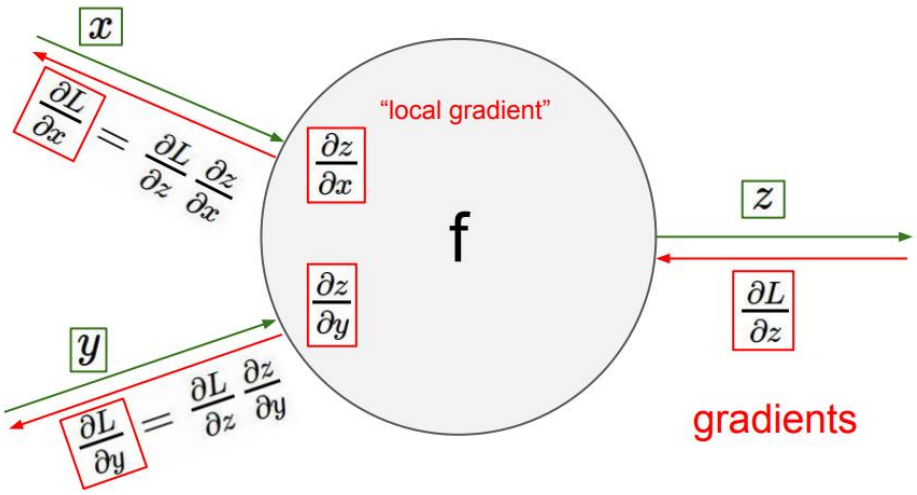


Abbildung 3.23: Einfache Darstellung des Backpropagation Algorithmus mit zwei Eingaben x und y, wobei zuerst ein Forward Propagation ausgeführt wird, um den Wert von (z) zu berechnen. Dann wird die Fehlerfunktion L berechnet. Anschließend wird der Gradient entlang L durchgeführt, um den Fehler zu minimieren. Quelle: https://cdn-images-1.medium.com/max/1600/1*FceBJSJ7j8jHjb4TmLV0Ew.png

3.8.2 Contrastive Divergence

Der Contrastive Divergence (CD) Algorithmus ist ein Vorwärts-Lernalgorithmus und wird häufig auf Restricted Boltzmann Machines (RBMs) angewendet, der Baustein von Deep Believe Netzwerken (DBN). Er basiert auf Gibbs Sampling, da in jedem Schritt aus zwei Propagation (positive und negative) besteht. Die positive Propagation berechnet h_j aus v_i und die negative Propagation berechnet v_{i+1} aus h_j. Die Propagationen basieren auf den folgenden Berechnungen der bedingten Wahrscheinlichkeiten:

$$p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{37}$$

$$p(v_i = 1|h) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{38}$$

Wobei σ die Sigmoid-Funktion ist [24, p. 2]. CD ordnet für jede Eingabe einen Eingabevektor zu den sichtbaren Einheiten und führt k Schritte des Gibbs-Samplings durch. Dann wird die Änderung der Gewichte mit der folgenden Gleichung berechnet:

$$\Delta w_{ij} = \varepsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon} \right) \tag{39}$$

Für diese Algorithmen gibt es zwei Hauptparameter: Die Lernrate ε und die Anzahl der Gibbs-Sampling-Schritte k [25, pp. 4-5]. Die Abbildung 3.24 zeigt der CD Algorithmus

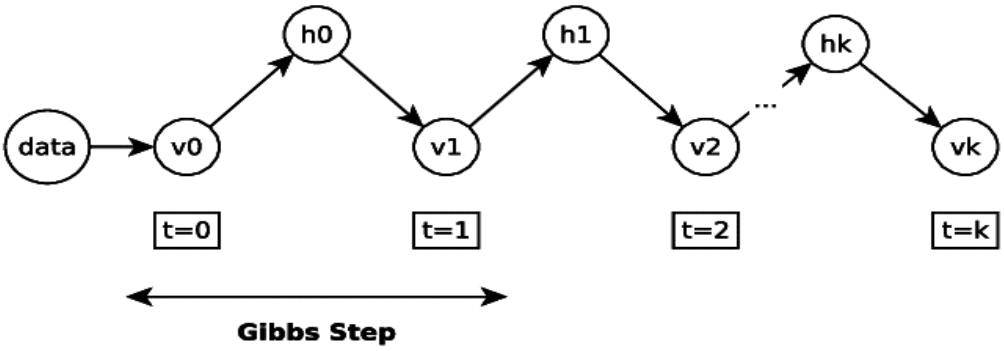


Abbildung 3.24: Eine grafische Darstellung des CD Algorithmus auf RBM. Ein Eingabevektor wird zu den sichtbaren Einheiten v zugeordnet, dann werden k Gibbs Schritte durchgeführt. Quelle: [40, p. 2]

Hier ist eine kurze Beschreibung des CD Algorithmus auf einen Mini-Batch [40, p. 5]:

Standard Contrastive Divergence Algorithmus (auf Mini-Batch)

Schleife für alle $v_0 \in$ Mini-Batch

h_0 versteckte Probe wird durch v_0 aktiviert

v_1 sichtbare Probe wird durch h_0 aktiviert

h_1 versteckte Probe wird durch v_1 aktiviert

$W_{pos} += v_0 \otimes h_0$

$W_{neg} += v_1 \otimes h_1$

Ende der Schleife

$$\nabla W = \frac{\epsilon}{B}(W_{pos} - W_{neg})$$

$$\nabla b = \frac{\epsilon}{B}(v_0 - v_1)$$

$$\nabla c = \frac{\epsilon}{B}(h_0 - h_1)$$

Ein Nachteil von CD ist, dass CD bei tiefen, mehrschichtigen Netzwerken mit unterschiedlichen Gewichten auf jeder Ebene scheitert, weil diese Netzwerke zu lange brauchen, um selbst mit einem geklemmten Datenvektor ein bedingtes Gleichgewicht zu erreichen [41, p. 1535].

3.8.3 Greedy Layer-Wise

Das Training einer tiefen Struktur kann schwierig sein, da es hohe Abhängigkeiten zwischen den Parametern der Schichten geben kann. Das Greedy Layer-Wise Training ist ein unüberwachter vortrainierter Lernalgorithmus von Geoffrey Hinton für Deep Belief Netzwerke (DBN) (Abbildung 3.25) und zielt darauf ab, jede Schicht eines DBN sequentiell zu trainieren und die Ergebnisse der unteren Schichten an die oberen Schichten weiterzugeben während die unteren Schichte fixiert bleiben. Zuerst werden einfache Konzepte gelernt und darauf wird aufgebaut, um so starke Konzepte zu lernen. Dies ermöglicht die Optimierung eines Netzwerks im Vergleich zu herkömmlichen Trainingsalgorithmen [42, pp. 2-3].

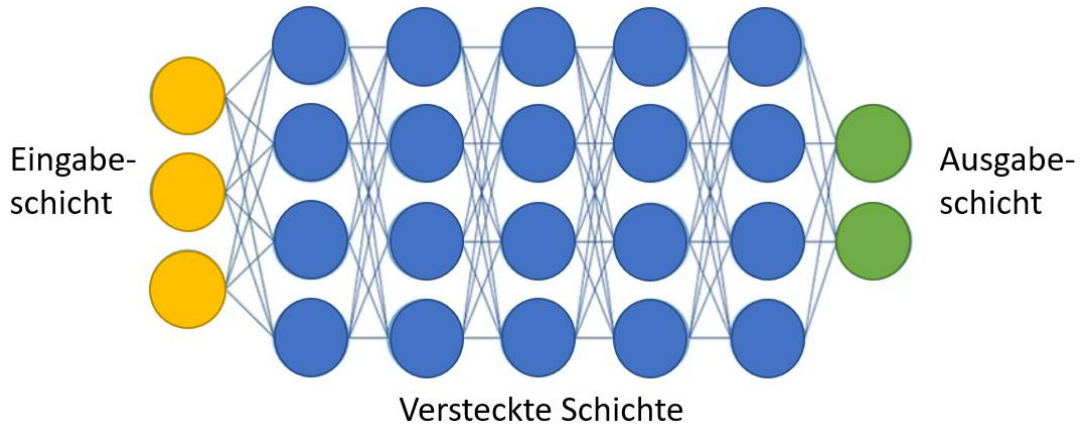


Abbildung 3.25: Deep Beliefe Netzwerk mit einer Eingabeschicht (gelb), fünf versteckten Schichten (blau) und einer Ausgabeschicht (grün).Alle Schichten sind vollständig miteinander verbunden. (einige Darstellung)

Wenn jede Schicht eine RBM (Abbildung 3.26) ist, werden die Schichten sequentiell beginnend mit der ersten Schicht trainiert und jede Schicht lernt eine höherwertige Darstellung von der letzten Schicht. Der Algorithmus führt die folgenden Schritte aus: Zuerst wird die erste Schicht mit einem unüberwachten Lernalgorithmus, wie RBM trainiert, und es wird ein initialer Satz von Parametern für die erste Schicht erzeugt. Dann wird die Ausgabe der ersten Schicht als Eingabe der nächsten Schicht zugeordnet und diese Schicht wird auch wie die erste Schicht initialisiert. Nachdem den Schichten initialisiert werden, wird das gesamte neuronale Netzwerk mit einem überwachten Lernalgorithmus trainiert werden. Ein Vorteil des Algorithmus ist, dass der Lernalgorithmus, der mit dem Konzept „Schicht für Schicht“ arbeitet, einen guten Satz von Modellparametern schnell finden kann, auch für Modelle, die viele Schichten von Nichtlinearitäten und viele Parameter enthalten. Außerdem kann der Lernalgorithmus eine sehr große Menge von ungelabelten Daten beim Erstellen eines vortrainieren, unüberwachten Modells effizient nutzen [43, pp. 1552-1554].

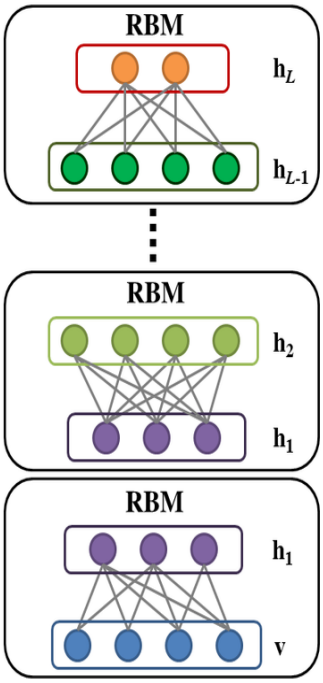


Abbildung 3.26: Greedy Layer-Weis Lernen in einem Deep Believe Network (DBN). Das Netzwerk besteht aus mehrerer RBM Netzwerke. Quelle: https://www.researchgate.net/publication/305078729_Maximum_Entropy_Learning_with_Deep_Belief_Networks/figures?lo=1

3.8.4 Sequential Minimal Optimization

Das Training einer SVM erfordert die Lösung eines Optimierungsproblems der quadratischen Programmierung (QP), die in der Gleichung (29) beschrieben ist. Die Karush-Kuhn-Tucker (KKT) Bedingungen sind notwendige Bedingungen zum Finden einer optimalen Lösung des QP-Problems. Beim QP-Problem des SVM Algorithmus sind die KKT Bedingungen:

$$\begin{aligned} \alpha_i = 0 &\leftrightarrow y_i u_i \geq 1 \\ 0 < \alpha_i < C &\leftrightarrow y_i u_i = 1 \\ \alpha_i = C &\leftrightarrow y_i u_i \leq 1 \end{aligned} \tag{40}$$

Wobei $u_i = w^T x_i + b$ der Ausgang der SVM ist [44, p. 4]. Sequential Minimal Optimization (SMO) ist ein einfacher Algorithmus, der das SVM QP-Problem ohne zusätzlichen Matrixspeicher und ohne numerische QP-Optimierungsschritte schnell lösen kann. SMO teilt das große QP-Problem in eine Reihe von kleinstmöglichen QP-Problemen auf, dann werden diese kleinen QP-Probleme analytisch gelöst, so dass eine zeitaufwändige numerische QP-Optimierung als innere Schleife entfällt. Das

kleinstmögliche QP-Problem beinhaltet zwei Lagrange-Multiplikatoren, wobei SMO bei jedem Schritt zwei Lagrange-Multiplikatoren α_i, α_j zur Optimierung auswählt, danach findet der SMO die optimalen Multiplikatoren und aktualisiert das SVM entsprechend den neuen optimalen Werten. Wegen der linearen KKT Bedingungen liegen die zwei Lagrange-Multiplikatoren auf einer diagonalen Linie, wie die folgende Abbildung.

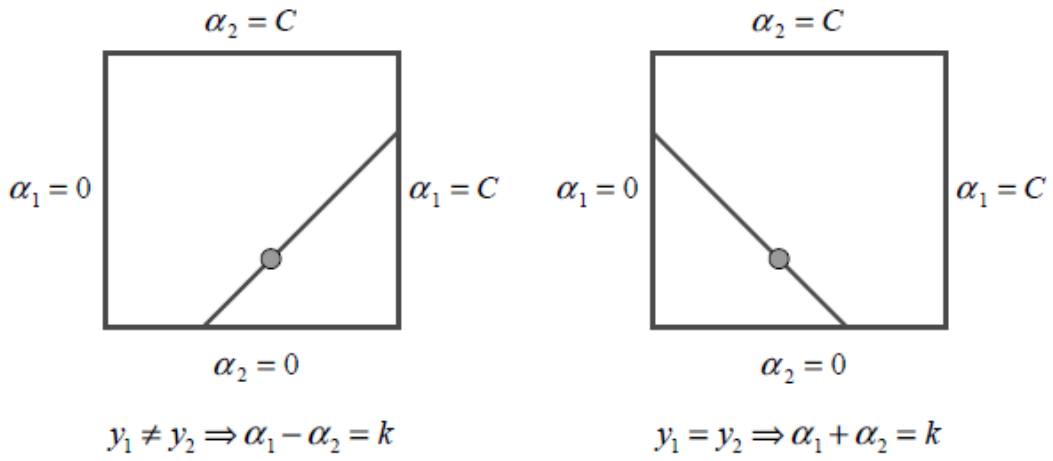


Abbildung 3.27: Die zwei Lagrange Multiplikatoren $\alpha_1$und $\boldsymbol{\alpha_2}$ und die Grenzen L und H. Der Lagrange Multiplikator bewegt sich entlang die diagonale Linie und hat immer einen Wert zwischen Null und C. Quelle: [44, p. 6]

Zunächst berechnet der Algorithmus den α_j und die Enden der diagonalen Linie in Bezug auf den α_j.

Wenn $y_i \neq y_j$ $\quad L = max(0, \alpha_i - \alpha_j)$ & $H = min(C, C + \alpha_j - \alpha_i)$ (42)

Wenn $y_i = y_j$ $\quad L = max(0, \alpha_i - \alpha_j)$ & $H = min(C, C + \alpha_j - \alpha_i)$ (43)

Das hat zur Folge, dass der neue α_j durch die folgende Gleichung berechnet werden kann:

$$\alpha_j := \alpha_j - \frac{y_j(E_i - E_j)}{\eta} \tag{44}$$

Hierbei ist $\eta = K(\vec{x}_i, \vec{x}_i) + K(\vec{x}_j, \vec{x}_j) - 2K(\vec{x}_i, \vec{x}_j)$ die zweite Ableitung der Lagrange in der Gleichung (29) entlang der diagonalen Line und $E_\mathrm{i} = u_i - y_i$ die Kostfunktion ist. Aus den Gleichungen (42), (43) und (44) ergibt sich α_j :

$$\alpha_j := \begin{cases} H & wenn\ \alpha_j \geq H \\ \alpha_j & wenn\ L < \alpha_j < H \\ L & wenn\ \alpha_j \leq L \end{cases} \quad (44)$$

Anschließend kann α_i durch die folgende Gleichung berechnet werden:

$$\alpha_i = \alpha_i + y_i y_j (\alpha_j{}^{alte} - \alpha_j) \quad (45)$$

Danach kann die Lagrange mit den neuen Lagrange-Multiplikatoren berechnet werden. Wenn die Lagrange nicht optimal ist, berechnet der SMO neue Lagrange-Multiplikatoren bis die Lagrange minimal ist [45, pp. 2-3]. Nach jedem Schritt wird b berechnet, sodass die Berechnung von b von der α_i und α_j Werten abhängt:

$$0 < \alpha_i < C \quad b_1 = b - E_i + y_i(\alpha_i - \alpha_i{}^{alte})\langle x_i, x_i\rangle - y_j\left(\alpha_j - \alpha_j{}^{alte}\right)\langle x_j, x_j\rangle$$

$$0 < \alpha_j < C \quad b_2 = b - E_j + y_i(\alpha_i - \alpha_i{}^{alte})\langle x_i, x_j\rangle - y_j\left(\alpha_j - \alpha_j{}^{alte}\right)\langle x_i, x_j\rangle \quad (46)$$

Ansonsten $$b = \frac{b_1 + b_2}{2}$$

Hier ist ein Überblick der SMO Algorithmus beim Training der SVM [46, p. 17]:

SMO Algorithmus zum Training SVM

Initiale Koeffizienten $\alpha_k = 0, k \in \{1, n\}$

Initiale Gradient $g_k = 1, k \in \{1, n\}$

Schleife

$if\ y_i\alpha_i < B_i \rightarrow i = argmax_i(y_i g_i)$

$if\ y_i\alpha_i > A_j \rightarrow j = argmax_j\left(y_j g_j\right)$

Optimale Kriterium $if\ y_j g_j \geq y_i g_i$

Lambda Suchen $\lambda = Min\left\{B_i - y_i\alpha_i, y_j\alpha_j - A_j, y_i g_i - \frac{y_j g_j}{K_{ii}+K_{jj}-2K_{ij}}\right\}$

Gradient Aktualisieren $g_k = g_k - \lambda_{yk}K_{ik} + \lambda_{yk}K_{jk}$

Koeffizienten Aktualisieren $\alpha_j = \alpha_j - y_j\lambda, \alpha_i = a_i + y_i\lambda$

Ende der Schleife

Der Vorteil von SMO liegt darin, dass die Lösung für zwei Lagrange-Multiplikatoren analytisch erfolgen kann. Auf diese Weise wird eine numerische QP-Optimierung

vollständig vermieden. Darüber hinaus benötigt SMO keinen zusätzlichen Matrixspeicher. Die Rechenzeit von SMO wird dominiert durch SVM-Auswertung, daher ist SMO am schnellsten für lineare SVMs und spärliche Datensätze [44, p. 6].

4 Datensatz

4.1 Überblick über den Datensatz

In dieser Arbeit wird der Datensatz des GTSRBs verwendet. GTSRB „The German Traffic Sign Recognition Benchmark“ besteht aus mehr als 50,000 Bildern von ausgeschnitten einzelnen Verkehrsschildern. Der Datensatz ist in Ordner aufgeteilt und in jedem Ordner gibt es Bilder und eine CSV-Datei mit den Anmerkungen der Bilder. Der GTSRB ist in ein Trainingsdataset und ein Testdataset aufgeteilt. Das Trainingsdataset besteht aus fast 40,000 Bildern, die in 43 Klassen geordnet sind und das Testdataset besteht aus mehr als 12,000 Bildern [47]. In der folgenden Abbildung sind ein paar Beispiele der Klassen im Datensatz.

Abbildung 4.1: Beispiele für die 43 Klassen von Verkehrszeichen im GTSRB Datensatz. Quelle: [47]

4.2 Bildeigenschaften

Jedes Bild hat nur ein Verkehrszeichen und die Größe der Bilder variiert zwischen 15x15 und 250x250 Pixeln. In diesem Datensatz sind die Bilder im PPM (Portable Pixmap) Format gespeichert. Die Bilder haben einen Rand von 10% um das Verkehrszeichen um die Kantenerkennung zu ermöglichen. Die CSV Datei stellt die folgenden Informationen bereit: Dateiname, Bereit, Höhe, ROI.x1, ROI.y1, ROI.x2, ROI.y2. ROI.x1 und ROI.y1 sind die Koordinate der oberen linken Ecke von Verkehrszeichen. ROI.x2 und ROI.y2 sind die Koordinate der unteren rechten Ecke von Verkehrszeichen [47]. Die folgende Abbildung zeigt die Koordinaten des Bildes.

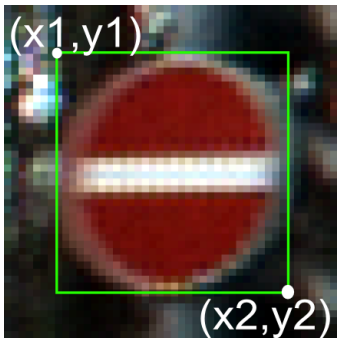


Abbildung 4.2: Ein Beispiel vom Datensatz, das die Koordinaten ROI.x1, ROI.y1, ROI.x2 und ROI.y2 zeigt [47]

4.3 Datenvorverarbeitung

Da die Bildergröße zwischen 15x15 und 250x250 Pixeln variiert, werden die Bilder verarbeitet, um die gleiche Größe zu bekommen. Dann werden die Trainingsdaten in Trainingsdaten und Validierungsdaten aufgeteilt, so dass während des Trainings die Ergebnisse gegen neue Daten getestet werden. Es wird für die Validierung ein kleiner Teil 1/10 der gesamten Trainingsdaten benutzt. Am Ende werden die Daten in Arrays gespeichert. Die Klassen haben im Datensatz nur Klasse-ID, die zwischen 0 und 43 variiert, aber sie werden mit einem CSV Datei „Schildernamen“ verknüpft, die die Labels und die Schildernamen beinhaltet. Hier wird die readTrafficSigns Funktion aus [47] verwendet, um die folgenden Aufgaben bei der Trainings- und Testdaten zu erfüllen:

- Datenlesen aus dem Trainingsordner und dem Testordner und Speicher in List.
- Geben die Bilder eine einheitliche Größe (32x32) Pixel.

- Rückgabe: Bilder und ihre Labels in einer Liste

Dann werden die Trainingsdaten in Trainings- und Validierungsdaten durch die train_test_spilt Funktion aus Sklearn Bibliothek aufgeteilt, wobei die Validierungsdaten 10% von den Trainingsdaten sind. Am Ende werden die Daten in Arrays gespeichert, um sie in den Main Code zu laden. In der folgenden Abbildung wird ein Histogramm der Verteilung der Klassen im Trainingsdaten angezeigt.

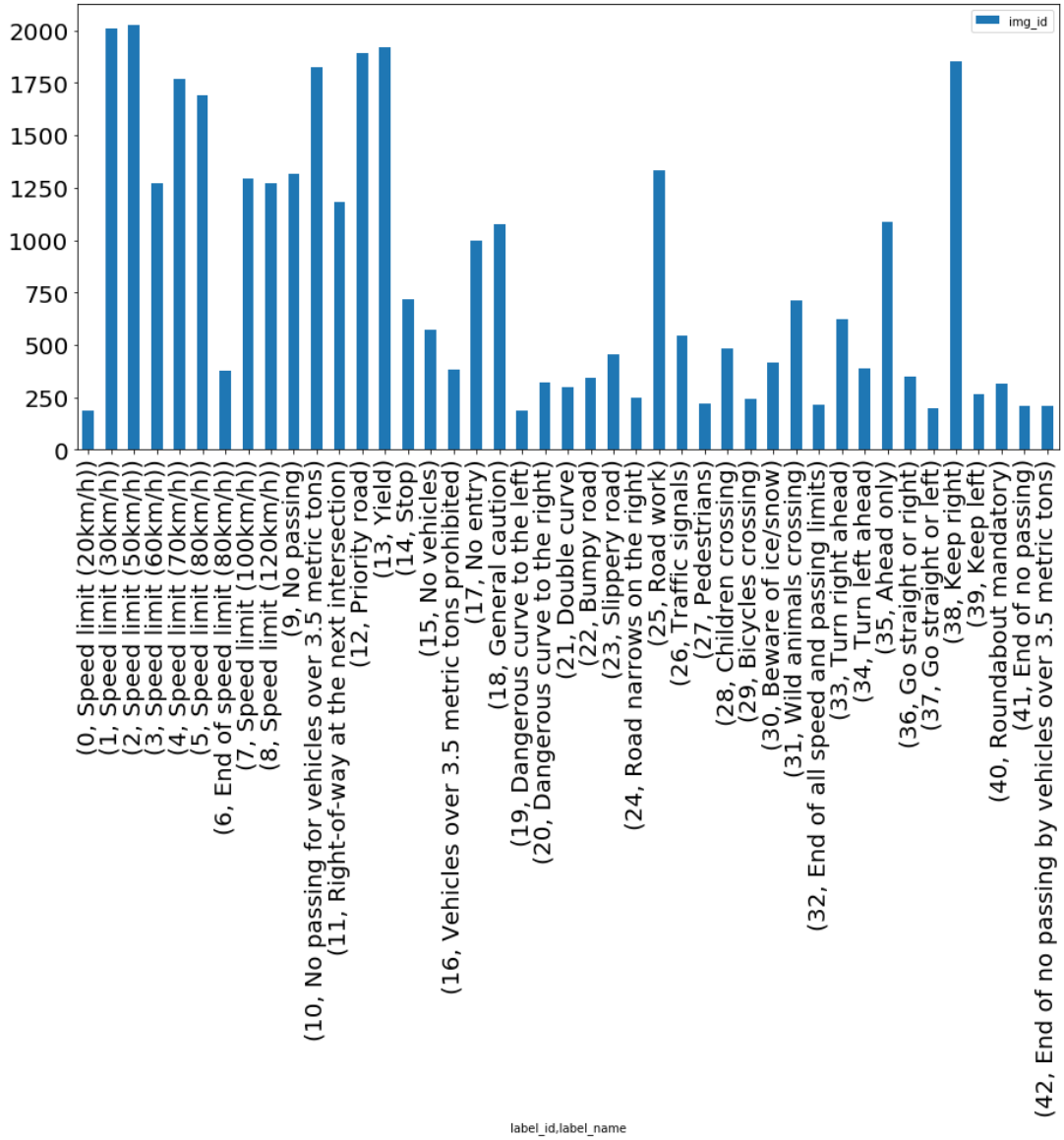


Abbildung 4.3: Ein Histogramm zeigt der Verteilung der Klassen im Trainingsdaten an, wobei einige Klassen weniger als 200 Trainingsproben und andere Klassen fast 2000 Trainingsproben haben (eigene Darstellung)

Es gibt ein deutliches Ungleichgewicht zwischen den Klassenverteilung in den Trainingsdaten, wie das Histogramm in der Abbildung 4.3 zeigt. Einige Klassen haben weniger als 200 Bilder, während andere über 2000 Bilder haben. Das bedeutet, dass das Modell auf überrepräsentierte Klassen ausgerichtet sein könnte, insbesondere wenn es in seinen Vorhersagen unsicher ist. Wegen der unterschiedlichen Verteilung der Klassen im Datensatz werden die 10 am meisten aufgetretenen Klassen gesammelt und als einen neuen Datensatz betrachtet. Dafür müssen die Ordner von diesen Klassen im Trainings- und Testordner in einem neuen Trainings- und Testordner kopiert und umbenannt werden, damit die vorherige Funktion die Bilder aus diesen Ordnern lesen und zuordnen kann. Außerdem müssen die CSV Datei in jedem Ordner und die Labels in jeder CSV Datei umbenannt werden. Der erstellte Datensatz besteht aus:

- 16254 Trainingsmustern
- 1806 Validierungsmustern
- 5940 Testmustern

In der Abbildung 4.4 wird die Verteilung der Klassen im neuen Datensatz dargestellt.

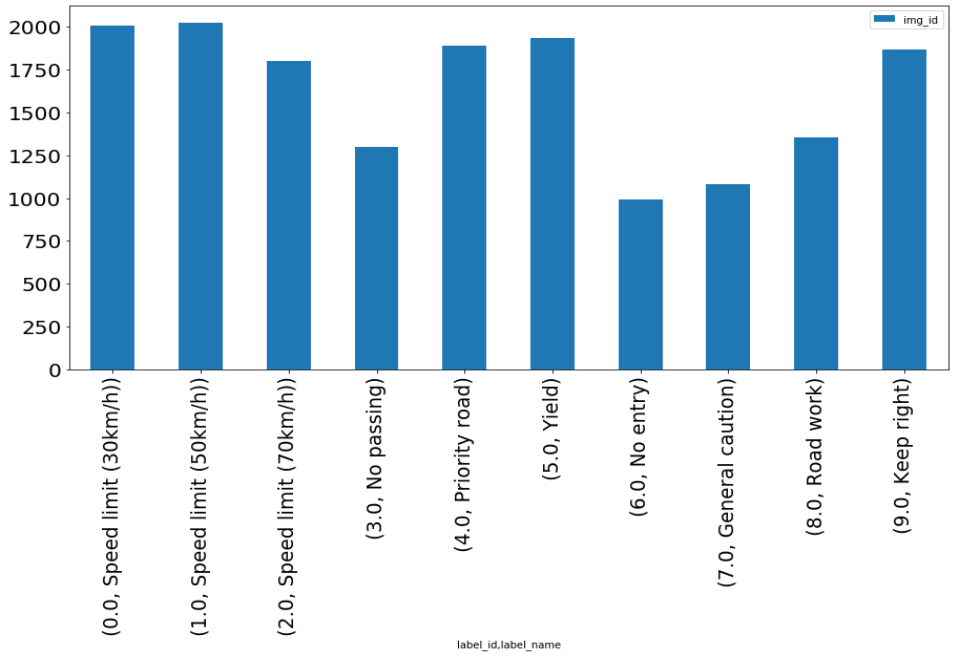


Abbildung 4.4: Ein Histogramm zeigt der Verteilung der Klassen im neuen Trainingsdaten an, wobei alle Klassen ähnliche Trainingsprobenanzahl haben. Die Probenanzahl der Klassen variiert zwischen 1000 und 2000 (eigene Darstellung)

5 Aufbau und Implementierung des Modells

In dieser Arbeit wird ein Modell entwickelt, das aus zwei Bestandteilen besteht. Der erste Teil extrahiert und erlernt die wesentlichen Merkmale aus der Eingabe, mit denen die Eingabe gut repräsentiert werden kann. In diesem Fall wird von einem Convolutional Neural Netzwerk und Convolutional Autoencoder gesprochen. Der zweite Teil stellt den Klassifikator dar, sodass der zweite Teil die extrahierten Merkmale vom ersten Teil als Eingabe erhält, dann kann die Klassifizierungsaufgabe anhand der eingegebenen Merkmale erfüllt werden. Für Klassifikationsaufgaben verwenden die meisten Deep-Learning-Modelle die Softmax-Aktivierungsfunktion zur Vorhersage und Minimierung von Cross-Entropie-Verlusten. In dieser Arbeit wird SVM als Klassifikator verwendet.

5.1 Convolutional Autoencoder

Die verwendete Architektur des Convolutional Autoencoders kombiniert die Eigenschaften des Convolutional Autoencoders und Restricted Boltzmann Machine. Infolgedessen basiert ein tiefes Netzwerk von CAE auf dem Konzept von DBN, das aus mehreren RBM-Schichten hintereinander besteht. Wie im Kapitel 3.4 beschrieben, besteht CAE aus einer Faltungs- und einer Entfaltungsschicht, sodass bei der Faltungsschicht ein Skalarprodukt zwischen einer Bildregion und den n Filtern ausgeführt wird, wobei n die Anzahl der Filter darstellt. Daraus folgt, dass n Feature-Map-Regionen erzeugt werden (Abbildung 5.1). Die Bildregion, deren Größe der Filtergröße entspricht, wird als eine sichtbare Schicht betrachtet und die Feature-Maps-Region als eine versteckte Schicht, wobei die sichtbare Schicht vollständig mit der versteckten Schicht verbunden ist. Die Anzahl der Einheiten in der sichtbaren Schicht entspricht der Filtergröße, während die Anzahl der Einheiten in der versteckten Schicht die Anzahl der Filter entspricht (siehe Abbildung 3.16). Das Training einer RBM erfolgt durch den Contrastive Divergence Lernalgorithmus (Kapitel 3.8.2). Dafür müssen eine positive Propagation und eine negative Propagation berechnet werden, anschließend wird der Contrastive Divergence implementiert. Die positive Propagation in CAE entspricht der Faltung zwischen einer Bildregion und den n Filtern. Die negative Propagation in CAE repräsentiert die Entfaltung der n Feature-Map-Regionen mit den n Filtern. Die Contrastive Divergence wird anschließend die Gewichte anpassen, sodass die Differenz zwischen der Eingabe und Rekonstruktion minimiert wird. Wenn das Training der Gewichte fertig ist, werden die Ergebnisse abge-

speichert und die Convolutional Autoencoder nur als Faltungsschicht verwendet, sodass die Feature-Maps berechnet und an die nächsten CAE weitergegeben werden. Die Verbindung zwischen den Schichten basiert auf dem Greedy-Layer-Wise Algorithmus (im Kapitel 3.8.3). Zur Reduktion der Bilddimension wird in dieser Arbeit die Stride anstatt die Pooling-Schicht verwendet.

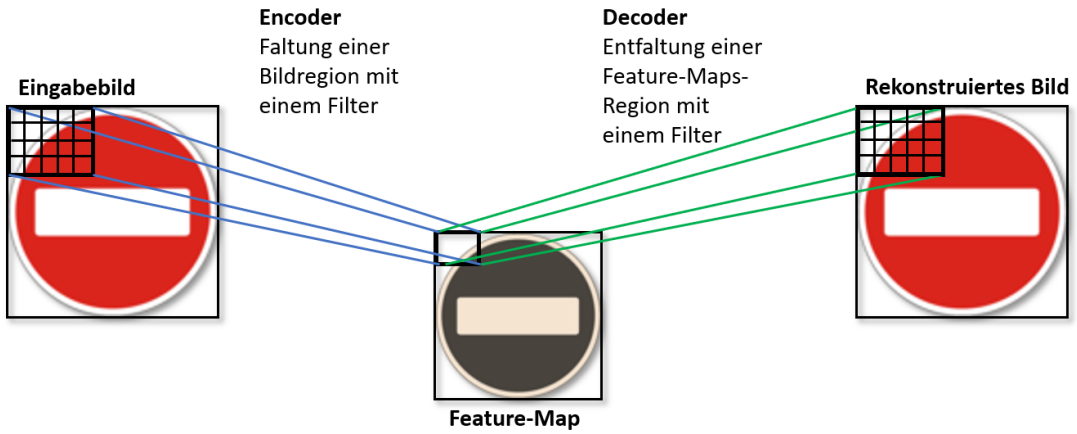


Abbildung 5.1: Convolutional Autoencoder basiert auf dem RBM Konzept, sodass jede Bildregion, die der Filtergröße entspricht, als eine sichtbare Schicht betrachtet wird und jede Feature-Map-Region als eine versteckte Einheit betrachtet wird. Die Anzahl der Feature-Maps entspricht der Anzahl der versteckten Einheiten. In diesem Beispiel ist die Filtergröße 5x4. Das entspricht 20 sichtbaren Einheiten. Es gibt ein Feature-Map, d.h. eine versteckte Einheit (eigene Darstellung)

5.2 SVM

In diesem Modell wird die letzte Schicht in CNN mittels SVM ersetzt. SVM gilt dank seiner Flexibilität und seiner Vorhersagefähigkeit als der modernste Algorithmus zur Lösung linearer und nichtlinearer Klassifizierungsprobleme [48, p. 1]. Zunächst hat SVM eine maximale Generalisierungsfähigkeit, sodass der SVM beim Training die maximale Marge zwischen den Klassen sucht, was sich nicht stark auf die Generalisierungsfähigkeit auswirkt. Im Gegenteil dazu, minimiert KNN beim Training den quadratischen Fehler zwischen den Ausgängen und den gewünschten Klassen, sodass die Klassengrenzen sich mit der Änderung der Gewichte ändert, was sich auf die Generalisierungsfähigkeit auswirkt [30, p. 58]. SVM bildet die Daten mit Hilfe der Kernelfunktionen in einem neuen, höher dimensionalen Raum ab, in dem ein linearer Separator in der Lage ist, zwischen den verschiedenen Klassen zu unterscheiden. Das hat zur Folge, dass SVM mit der richtigen Auswahl der Kernelfunktion jeden Trainingssatz erlernen kann [49, p. 1]. SVM ist robust gegen die Sonderfälle im Datensatz, wobei der Regularisierungsparameter die Sensitivität des Algorithmus gegen die

falschen Klassifizierungen steuert. Im Vergleich zu KNN, das anfällig für die Sonderfälle sind, da es die Summe der quadratischen Fehler berechnet [30, p. 59]. Ein anderer Vorteil ist, dass nach dem Training nur die Support Vektoren, welche die maximale Marge definieren, gespeichert werden. Wobei SVM nur die Support Vektoren für die Vorhersagephase erfordert, was zu einem niedrigeren Speicherplatz führt [50, p. 42]. Die Lösung des quadratischen Programmierproblems von SVM wird durch den SMO Trainingsalgorithmus erreicht, um die optimale Hyperebene zu finden.

5.3 Vorteile des Modells

- Die Faltungsschichten können getrennt von der Klassifizierungsschicht trainiert werden, sodass die Convolutional Autoencoder Schichten als ein vortrainiertes Netzwerk betrachtet werden. D.h., wenn neue Daten zum Datensatz hinzufügt werden, müssen die Filter nicht neue trainiert, aber die Klassifizierungsschicht muss die neuen Daten lernen. Im Vergleich zu CNN muss bei Hinzufügen neuer Daten das ganze Netzwerk neue trainiert, wenn neue Daten an den Datensatz hinzufügt werden.
- Die Filter jeder Convolutional Autoencoder Schicht werden mit dem Contrastive Divergence Lernalgorithmus trainiert und an die nächste Schicht anhand Greedy Layer Wise Algorithmus übertragen und fixiert bleiben, sodass die Filter bei jeder Schicht nur einmal trainiert werden müssen und nicht wie dem Backpropagation Algorithmus. Hierbei ist jedoch zu beachten, dass alle Filter für jeden Batch angepasst werden müssen.
- Der SVM Klassifikator kann mittels Kernelfunktionen die Daten in einem hochdimensionalen Raum transformieren und dadurch effizienter klassifizieren.
- Die Schwierigkeiten beim Erlernen tieferer hierarchischer Merkmale beim SVM wird in diesem Modell durch die Faltungsschichten vermieden, sodass die tiefen Merkmale extrahiert werden, dann an den SVM weitergegeben werden.

5.4 Architektur der Implementierung

Im folgenden Klassendiagramm wird die Architektur des Quellcodes dargestellt.

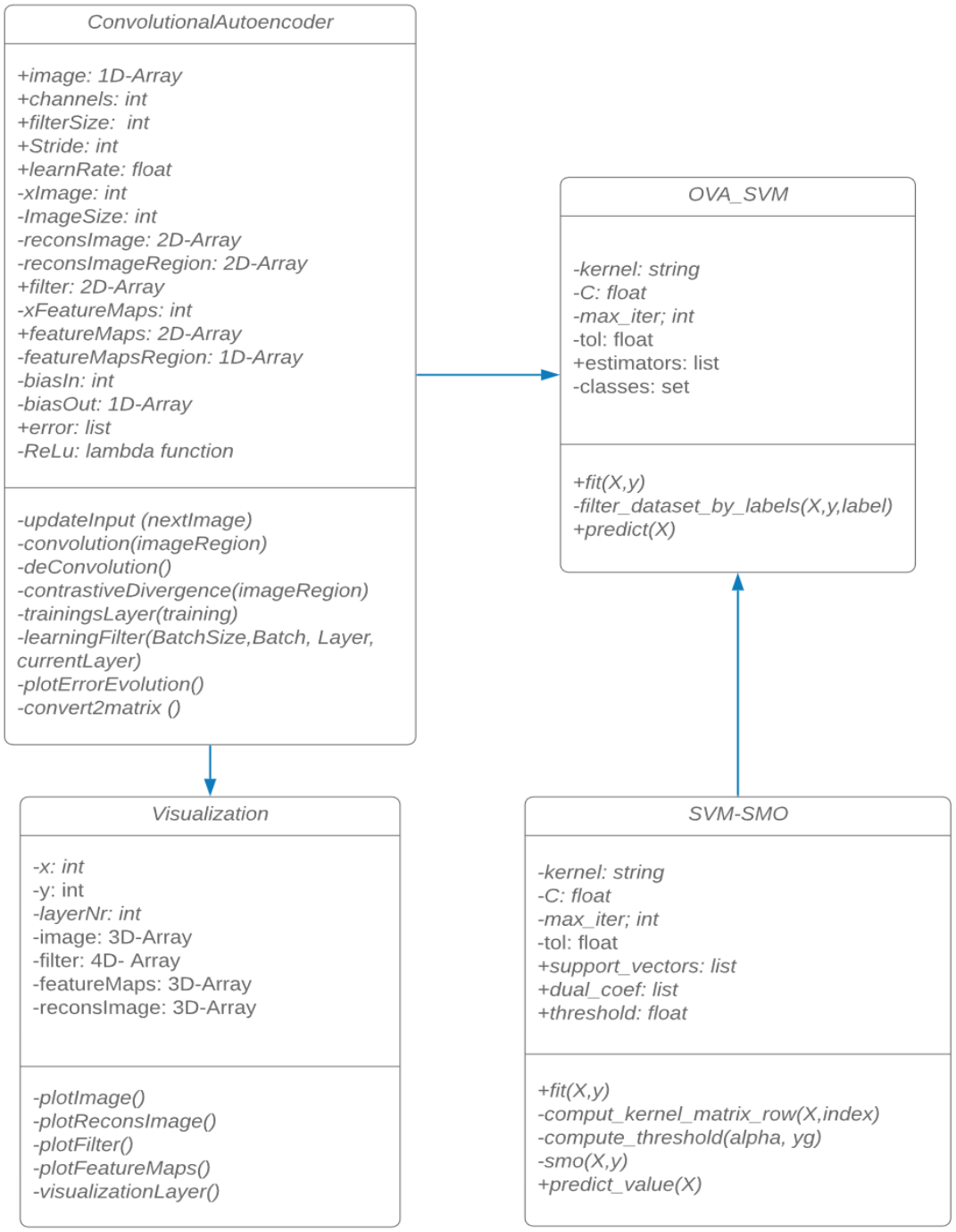


Abbildung 5.2: a) Der ConvolutionalAutoencoder Klasse stellt die Arbeitsweise des Convolutional Autoencoder dar. b) Visualization Klasse ist dafür verantwortlich, eine Visualisierung der Convolution Autoencoder Ergebnisse darzustellen. c) SVM_SMO Klasse stellt einen binaren SVM Klassifikator dar, der mit SMO Lernalgorithmus trainiert wird. d) OVA_SVM Klasse bildet die Mehrklassen SVM Klassifikator ab (eigene Darstellung)

Der Quellecode des Modells: https://github.com/Edwardha92/Image-Classification-For-Self-Driving-Cars-Through-Convolution-Autoencoder

5.4.1 Convolution Autoencoder Klasse

Die **ConvolutionAutoencoder** Klasse stellt die Arbeitsweise des Convolutional Autoencoder dar. Sie besteht aus mehreren Methoden, die die Aufgabe des CAE erfüllen können. Das Ziel des CAEs ist, die Filter bzw. die Gewichte zu trainieren, so dass das Eingabebild aus den Feature-Maps gut rekonstruiert werden kann. Das erfolgt durch die Anwendung die learningFilter Methode. Diese Methode hat zwei wesentlichen Aufgaben: Die Gewichte in jeder Schicht auf den Bereich der Batchgröße mit dem Batch als die Eingabe zu trainieren und die Verbindungen zwischen den Schichten anhand des Greedy-Layer-Wise Algorithmus (im Kapitel 3.8.3) herzustellen. Für jedes Eingabebild vom Batchgröße-Bereich wird kontrolliert, ob die aktuelle Schicht die erste Schicht ist oder nicht. Wenn die aktuelle Schicht die erste Schicht ist, wird der Batch als die Eingabe der aktuellen Schicht verwendet, ansonsten werden die flachen Feature-Maps der vorherigen Schicht als die Eingabe der aktuellen Schicht verwendet. Die Aktualisierung der Eingabe wird durch die updateInput Methode erfolgt. Anschließend wird die trainingsLayer Methode auf die aktuelle Schicht und die vorherigen Schichten ausgeführt. Die trainingsLayer-Methode hat einen boolean Parameter. Während der Trainingsphase ist dieser Parameter „True“, sodass diese Methode die Filter auf dem ganzen Bild mit einem Stride-Wert verschiebt und die Faltung, die Entfaltung und den Contrastive Divergence Algorithmus auf einer Bildregion mit der Filtergröße anwendet. Außerdem wird der quadratischen Fehler durch diese Methode berechnet und durch die plotErrorEvolution Methode grafisch dargestellt. Wenn der Parameter „False“ ist, werden die Faltungs- und Entfaltungsschritte ohne Lernen durchgeführt. Die Faltung wird durch die convolution Methode implementiert, wobei diese Methode die Faltungsoperation in der Gleichung (6) beschreibt, sodass das Skalarprodukt zwischen der Bildregion und den Filtern plus die Bias aus der Eingabe zu den Feature-Maps berechnet wird. Darüber hinaus wird die Leaky-ReLu-Aktivierungsfunktion auf das Ergebnis angewendet und resultiert die Feature-Maps.

$$\text{featureMapsRegion} = \text{ReLu}(\text{np.dot}(\text{filter}, \text{imageRegion}) + \text{biasIn})$$

Die Aktivierungsfunktion filtert die Werte in den Feature-Maps, sodass alle Werte, die größer als Null sind, gleichbleiben und die, die kleiner als Null sind, mit Faktor 0.01 multipliziert werden. Die Aktivierungsfunktion kann mit der folgenden Gleichung beschrieben werden:

$$f(x) = \begin{cases} 0.01 * x & wenn \ x < 0 \\ x & wenn \ x \geq 0 \end{cases}$$

Für weniger Komplexität wird die Bias aus Eingabe zu Feature-Maps immer auf 1 eingesetzt.

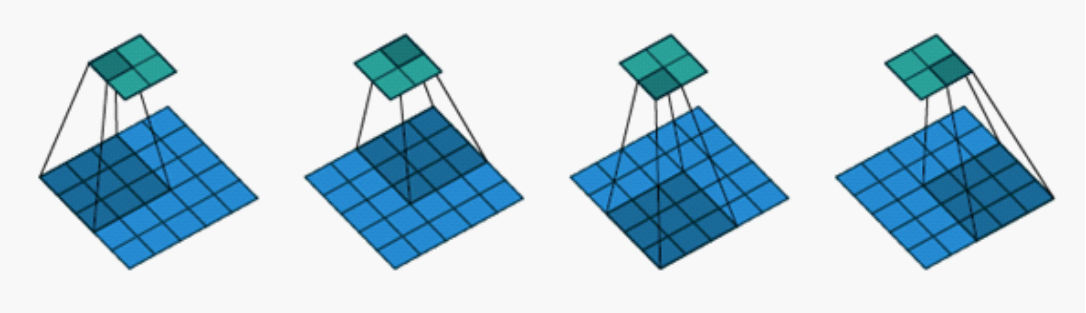


Abbildung 5.3: Die Faltungsschritte zwischen einem 5x5 Bild (Blau) und einem 3x3 Filter (Dunkelblau) und das Feature Map (Grün) mit Stride = 2 und kein Padding. Quelle: [51, p. 16]

Die Entfaltung wird durch die deConvolution Methode implementiert und sie beschreibt die Entfaltungsoperation zwischen der Feature-Maps-Region und dem inversen Filter und liefert die rekonstruierte Bildregion plus die Bias aus Feature-Maps zum rekonstruierten Bild. Außerdem wird die Leaky-ReLu Aktivierungsfunktion auf das Ergebnis angewendet. Hier werden die Bias aus Feature-Maps zum rekonstruierten Bild auf 0 eingesetzt für weniger Komplexität.

$$\text{reconsImagRegion} = \text{ReLu}(\text{np.dot}(\text{filter.T}, \text{featureMapRegion}) + \text{biasOut})$$

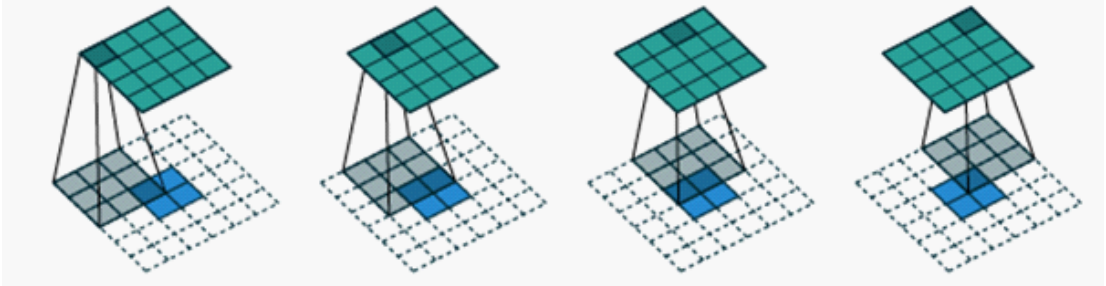


Abbildung 5.4: Die Entfaltungsschritte zwischen einem Feature-Map (Blau) und einem 3x3 Filter (Weiß) und das rekonstruierte Bild (Grün) mit Stride = 2 und kein Padding. Quelle: [51, p. 22]

Die Contrastive Divergence Methode optimiert den Filter bzw. die Gewichte durch die Anwendung des Contrastive-Divergence Lernalgorithmus, der in der Gleichung (39) beschrieben ist, sodass das Skalarprodukt zwischen der Feature-Maps-Region

und der Differenz zwischen der Eingabebildregion und der rekonstruierten Bildregion berechnet wird. Dann wird es mit der Lernrate zum Berechnen der Änderung der Gewichte multipliziert. Am Ende wird die Subtraktion des Ergebnisses von den alten Gewichten die neuen Gewichte gegeben.

$$\text{filter} -= \text{learnRate} * (\text{featureMapsRegion}, (\text{reconsImagRegion} - \text{imageRegion}).\text{T})$$

Um das Eingabebild, den Filter, die Feature-Maps und das rekonstruierte Bild zu visualisieren, müssen sie im Array durch convert2matrix Methode umgewandelt werden. Im nächsten Abschnitt wird die Visualisierungsklasse erklärt.

5.4.2 Visualisierung-klasse

Die **Visualization** Klasse ist dafür verantwortlich, die Eingabe, die Filter, die Feature-Maps und die Rekonstruktion pro Schicht grafisch darzustellen, um die Funktionalität und die Ergebnisse des CAEs besser zu verstehen. Dafür wird die Pyplot Funktion aus der Matplotlib Bibliothek verwendet. Die plotImage und plotReconsImag Methode haben die gleiche Funktionalität aber eine andere Eingabe. Die plotImage Methode stellt auf der ersten Schicht das Eingabebild im Fall farbiger oder grauer Bilder dar, sodass das vom Datensatz abhängig ist. In der nächsten Schicht ist das Eingabebild die Feature-Maps der vorherigen Schicht, deshalb funktioniert die plotImage Methode wie die plotFeatureMaps Methode, sodass sie die Feature-Maps darstellt. Der Filter sind in der Regel am besten auf der ersten Schicht interpretierbar, weil sie direkt auf die Rohpixeldaten trainiert. Die Visualisierung der Filter ist nützlich, da gut trainierte Netzwerke in der Regel schöne und glatte Filter ohne verrauschte Muster anzeigen. Verrauschte Muster können wegen des nicht langen Trainings verursacht werden, was auch zu einer Überanpassung führen kann [52]. Die plotFilter Methode ist verantwortlich für die Filtervisualisierung, sodass die Kanäle eines Filters aufsummiert und in Graustufen dargestellt werden. Um die Visualisierung der ganzen Schicht wird die visualizationLayer Methode durchgeführt, die alle Visualisierungsmethoden beinhaltet.

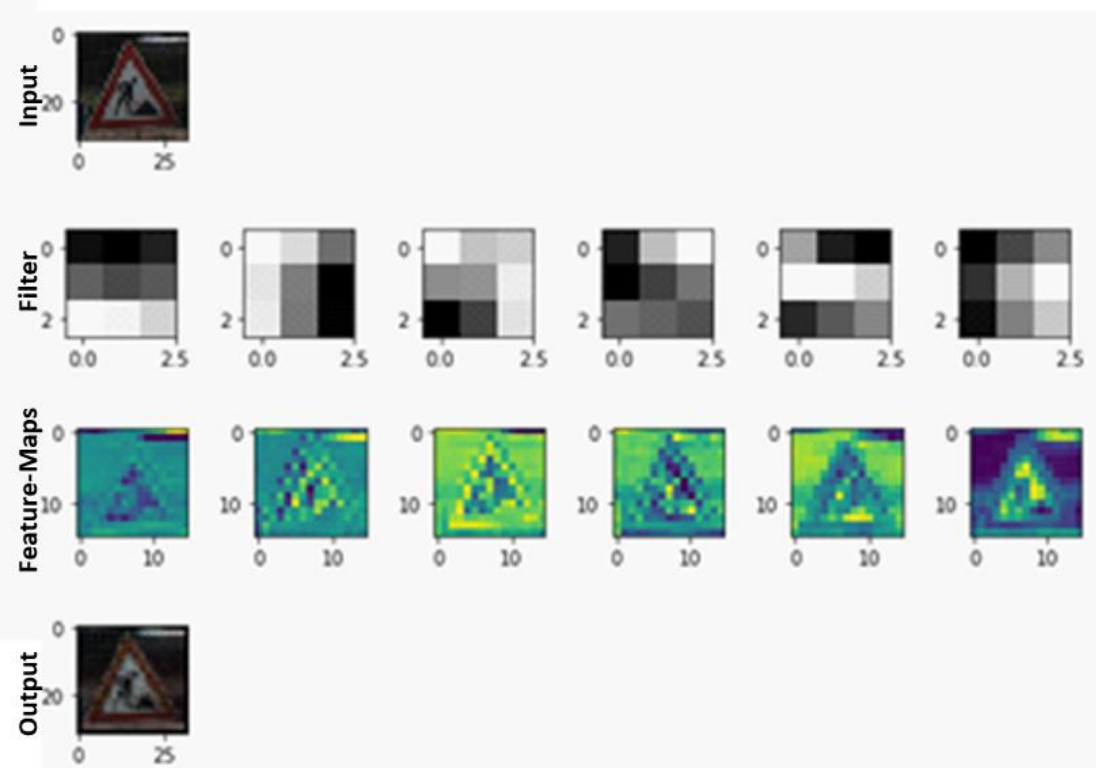


Abbildung 5.5: Beispiel der Visualisierung der ersten Convolutional-Autoencoder-Schicht mit einem farbigen Bild (32x32x3) als eine Eingabe und 6 Filtern (Spyder Konsole)

5.4.3 Binäre SVM Klasse

Der binäre SVM-Klassifikator wird durch die **SVM_SMO** Klasse implementiert und verwendet, um zwei Klassen zu klassifizieren. Die binäre SVM wird trainiert, um die optimale Trennungshyperebene zwischen zwei Klassen {-1, +1} zu finden. Dafür muss der SMO Algorithmus, der im Kapitel 3.8.4 beschrieben ist, zum Lösen des SVM-Optimierungsproblems angewendet werden, sodass die Lagrange-Multiplikatoren und den Schwellenwert gefunden werden. Danach werden die Dual-Koeffizienten (in der Gleichung 29) und die Support-Vektoren (in den Gleichung 19 und Gleichung 40) berechnet, sodass alle Trainingsdaten, die einen Lagrange-Multiplikator großer als Null haben, die Support Vektoren sind und die Dual Koeffizienten die Summe der Multiplikation zwischen den Lagrange-Multiplikatoren und dem Label sind, wenn der Lagrange-Multiplikator α_i großer als Null ist.

$$\text{Support Vektoren} = \text{Trainingsdaten (wenn } \alpha_i > 0)$$

$$\text{Dual Koeffizienten} = \sum \text{Lagrange\_Multiplikatoren} * \text{Label} \ \ (\text{wenn } \alpha_i > 0)$$

Nach dem Training werden die Testdaten in SVM eingeführt und ihre Labels werden vorhergesagt. Die Vorhersage wird erfolgreich durch die Berechnung von der Gleichung (33), sodass das vorhergesagte Label das Ergebnis der Entscheidungsfunktion von der Summe der Multiplikation zwischen den berechneten Dual-Koeffizienten und dem Kernel von den eingegebenen Daten und den Support Vektoren plus den Schwellenwert beträgt.

$$\text{Vorhersage} = f\left(\sum_{i=1}^{n} \text{Dual Koeffizienten} * K(x_i, \text{support Vektoren}) + b\right)$$

5.4.4 Mehrklassen SVM Klasse

Die **OVA_SVM** Klasse stellt den Mehrklassen SVM Algorithmus zur Datenklassifizierung dar. In dieser Arbeit wird die OVA Methode verwendet, die im Kapitel 3.7.1.4 beschrieben ist. Wie erwähnt ist, besteht die OVA Methode aus N binären SVM-Klassifikatoren, wobei N die Anzahl der Klassen im Datensatz ist. Deshalb wird die **SVM_SMO** Klasse innerhalb der **OVA_SVM** aufgerufen. Zunächst werden die Trainingsdaten für jede i Klasse in zwei Klassen {-1, +1} gefiltert, wobei die i Klasse die Klasse +1 und die restlichen Klassen den Wert -1 entsprechen. Danach werden die Daten in der binäre SVM Klassifikator eingeführt und wie im Kapitel 5.4.4 werden der binäre SVM die Support-Vektoren und die Dual-Koeffizienten für die i Klasse berechnet. Während der Testphase werden die Testdaten in allen binären SVM eingeführt und die Klasse, die größte Score hat, ist die vorhergesagte Klasse Gleichung (34).

$$\text{Vorhersage} = \text{argmax}(\text{der vorhergesagte Score der Klassen})$$

6 Tests

In der Praxis wird das Modell mit einem Teil des GTRSB Datensatzes, das aus den 10 am häufigsten aufgetretenen Klassen besteht, getestet, weil wie im Kapitel 4.3 erwähnt ist, wegen des großen Unterschieds der Klassenverteilung das Modell auf überrepräsentierte Klassen ausgerichtet sein könnte. Das Modell wird nicht nur mit den farbigen Bildern getestet, sondern auch mit Graustufenbilder, sodass die farbigen Bilder in Graustufenbilder durch die **rgb2gray** Funktion (im Anhang) zum Erstellen eines neuen Datensatzes umgewandelt werden. Am Ende werden die Ergebnisse des **OVA_SVM** Klassifikators mit den Ergebnissen der **SVC** Funktion aus der Scikit-Learn Bibliothek verglichen. Das Modell wird auf einer Plattform getestet:

- HP 15-bs114ng
  - Windows 10 Education 64 Bit
  - Prozessor: Intel(R) Core i5 8250 CPU @1.6 GHz (8 CPUs)
  - Arbeitsspeicher: 8192MB
  - Programme: Anaconda 1.9.7- Spyder 3.3.3 – Python 3.6.8

6.1 Hyperparameter

Die Hyperparameter jeder Schicht müssen vor der Trainingsphase eingestellt werden. Um die richtigen Hyperparameter zu finden, die die besten Generalisierungsergebnisse ohne Überanpassung liefern, müssen verschiedene Hyperparameter auf dem Modell getestet werden. Die Hyperparameter der Convolutional-Autoencoder-Schicht sind die Filtergröße, die Filteranzahl, Stride und die Lernrate. Es wird durch die Visualisierung der Convolutional-Autoencoder-Ergebnisse erkannt, ob die eingestellten Hyperparameter zur Schicht passen oder nicht, sodass die Auswirkungen jedes Hyperparameters auf die Feature-Maps angezeigt werden. Die Lernrate ist der wichtigste Hyperparameter. Wenn die Lernrate zu hoch ist, erfolgt das Training schnell aber kann der Trainingsfehler erhöht werden und nicht mehr verringern. Im Gegenteil, wenn die Lernrate zu niedrig ist, ist die Trainingszeit langsamer und erfordert mehr Daten zu lernen [4, p. 429]. Die kleinste effektive Filtergröße ist 3x3, mit der der Filter alle Informationen von links, rechts, oben, unten und die Mitte erfassen kann [53, p. 2]. Die Verwendung einer großen Filtergröße führt zu einer kleinen Feature-Maps-Größe und erfordert mehr Speicher für die Parameter [4, p. 431]. Außerdem hängt die

Filtergröße von der Bildgröße im Datensatz ab, sodass mit einer Filtergröße relativ groß zur Bildgröße kein tiefes Netzwerk erstellt werden kann. Je kleiner die Stride bei der Faltung ist, desto mehr Informationen sind in den Feature-Maps zu extrahieren. Außerdem führt das zu einer besseren Darstellung der Informationen. Im Gegenteil zu einer kleinen Stride liefert eine große Stride weniger Recherchieren aber auch weniger Feature-Maps. D.h. viele Informationen werden bei der Verwendung einer großen Stride verloren. Darüber hinaus spielt die Klassenanzahl im Datensatz eine wichtige Rolle beim Auswählen der Filteranzahl, weil im Fall einen Datensatz mit vielen Klassen das Netzwerk mehr Features lernen muss, um die Klassen zu unterscheiden. Wenn eine gute Konfiguration gefunden wird, wird auf die zweiten Convolutional-Autoencoder-Schicht übertragen, dann werden die beiden Schichten verknüpft und die Hyperparameter der zweiten Schicht gesucht werden. Zu den SVM Hyperparameter gehören die Kernelart, der Regularisierungsparameter, die maximale Iteration und die Toleranz. Das Modell wird mit verschiedenen Kernelarten, Toleranzwerten und Batchgrößen getestet. Anhand der Erkennungsrate werden die Hyperparameter umgestellt. Der Kernelart spielt eine große Rolle bei der Suche nach einer separierenden Hyperebene. Wenn die Daten durch eine lineare Hyperebene getrennt werden können, wird der lineare Kernel verwendet. Aber sind im Fall die Daten nicht linear separierbar, wird der rbf- Kernel empfohlen. Die folgende Abbildung zeigt, wie der rbf-Kernel die Daten trennt.

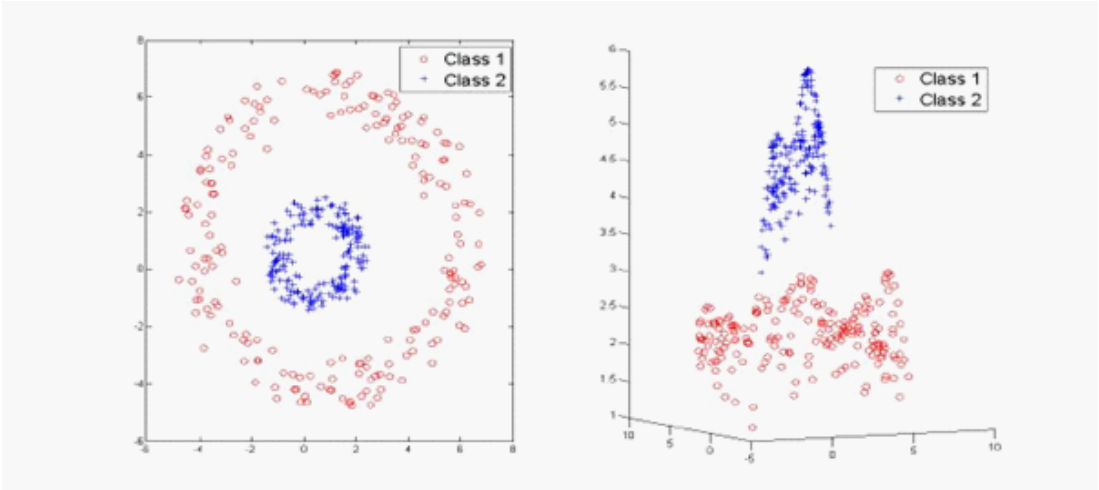


Abbildung 6.1: Beispiel für die Kernel-Mapping-Daten der Radialen Basisfunktion (RBF) vom nichtlinearen trennbaren Raum in den hochdimensionalen trennbaren Raum. Quelle:

https://www.researchgate.net/profile/Trong-Ton_Pham/publication/281602651/figure/fig12/AS:284558864994317@1444855528796/Example-of-Radial-Basis-Function-RBF-kernel-mapping-data-from-non-linear-separable.png

Der Regularisierungsparameter C ist standardmäßig auf 1 eingestellt und ziel darauf ab, die Marge zwischen den Klassen zu maximieren und die Anzahl der falsch klassifizierten Daten zu minimieren. Wenn C einen großen Wert hat, sucht der Algorithmus nach einer Hyperebene mit einer kleinen Marge aber liefert gute Ergebnisse. Umgekehrt führt ein sehr kleiner Wert von C dazu, dass der Algorithmus nach einer Hyperebene mit einer großen Marge sucht aber schlechte Ergebnisse liefert. Es ist auch notwendig auch zu wissen, dass C nicht sehr groß ist, weil das eine Überanpassung und eine schlechte Verallgemeinerung verursacht.

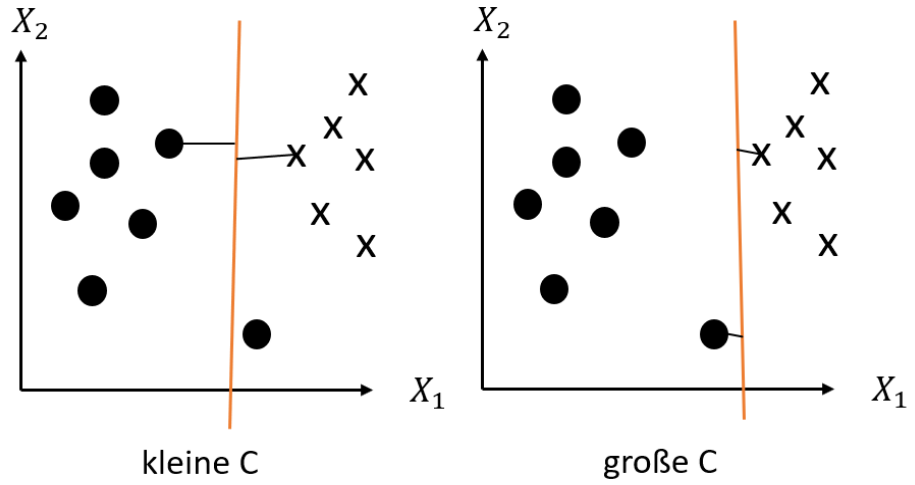


Abbildung 6.2: Ein Beispiel des SVM Klassifikators zwischen zwei Klassen (Kreis und X) mit einem kleinen Wert von C (Links) und einem großen Wert von C (Recht). (einige Darstellung)

Die maximale Iteration bezieht sich darauf, wie häufig der Algorithmus die Parameter aktualisiert. Der Toleranz Parameter wird für das Stopp-Kriterium verwendet, sodass der Algorithmus aufhört, nach einer optimalen Hyperebene zu suchen, wenn dieser Parameter erreicht ist. Darüber hinaus hat die Batchgröße eine große Bedeutung beim Training des Klassifikator, sodass sie stark auf die Erkennungsrate beeinflusst.

6.2 Vorgehensweise

Vor dem Training der Convolutional-Autoencoder-Schichten müssen die Trainings-, Validierungs- und Testdaten auf 255 zur Normalisierung geteilt werden, dann in Vek-

toren umgewandelt werden. Danach beginnt die Trainingsphase, sodass jede Schicht anhand des Contrastive-Divergence Trainingsalgorithmus (Kapitel 3.8.2) mit einem bestimmten Batchgröße nacheinander trainiert wird. Darüber hinaus werden die Schichten durch den Greedy-Layer-Wise Algorithmus (Kapitel 3.8.3) miteinander verknüpft, sodass die Feature-Maps der vorherigen vortrainierten Schicht zur Eingabe der nächsten Schicht zugeordnet wird und die Filter der vorherigen Schicht fixiert bleiben. Nach dem Training der Filter wird der quadratische Fehler von jeder Schicht abhängig von der Trainingsbatchgröße dargestellt, sodass der quadratische Fehler den Unterschied zwischen dem Eingabebild und dem rekonstruierten Bild anzeigt. Um die Filtertrainingsergebnisse zu sehen, wird eine Visualisierung für das Eingabebild, die Filter, die Feature-Maps und das rekonstruierte Bild von jeder Schicht dargestellt. Die nächste Phase ist die Verknüpfung der trainierten Convolutional Autoencoder Schichten mit dem Klassifikator. Dafür muss ein Batch der Trainingsdaten in die Convolutional Autoencoder ohne Training eingeführt werden, sodass der vortrainierte CAE als eine Faltungsschicht verwendet wird, um die Daten zu extrahieren. Dann werden die Feature-Maps der letzten Schicht flach gemacht und in einem Array hinzufügt, um die Eingabe des Klassifikators zu erstellen. In dieser Phase müssen keine Parameter eingestellt werden. Anschließend beginnt die Trainingsphase des SVM-Klassifikators. Hier wird der SVM mit dem erstellten Array trainiert, sodass er nach den optimalen Hyperebenen mit der maximalen Marge sucht, die die Daten trennen können. Dafür muss das Optimierungsproblem (im Kapitel 3.7.1.3) gelöst werden. Anschließend wird der SVM mit den Validierungsdaten validiert. Am Ende werden die Testdaten in der SVM eingeführt, um die Erkennungsrate des Modells zu zeigen. Zur Darstellung der Klassifizierungsergebnisse wird eine sog. Konfusionsmatrix verwendet. In dieser Matrix repräsentiert jede Zeile die vorhergesagte Klasse, während jede Spalte die tatsächliche Klasse repräsentiert. Zum Erstellen der Konfusionsmatrix wird die confusion_matirx Funktion aus sklearn.metrics aufgerufen, die die vorhergesagten Labels und die eingegebenen Labels als Parameter braucht.

6.3 Struktur des Modells

Das Modell besteht aus 3 Convolutional Autoencodern Schichten und einer SVM Schicht. Zum Training des Modells müssen zunächst die Hyperparameter der Schichten eingestellt werden. Hier ist eine Tabelle mit den verwendeten Hyperparameter der Convolutional Autoencoder Schichten.

Tabelle 6.1: Die verwendeten Hyperparameter der Convolutional Autoencoder Schichten im Modell

Hyperparameter	**CAE01**	**CAE02**	**CAE03**
Filtergröße	3x3	3x3	3x3
Filteranzahl	15	18	25
Stride	2	1	2
Lernrate	0.01	0.0005	0.00005

Beim Training des SVM Klassifikators werden die Hyperparameter in der Tabelle 6.2 verwendet:

Tabelle 6.2: Die verwendeten Hyperparameter im SVM Klassifikator

Hyperparameter	**SVM**
Kernel	Linear
Regularisierungsparameter	1
maximale Iteration	4000
Toleranz	0.01

Die folgende Abbildung zeigt, wie das Modell am Ende aussieht.

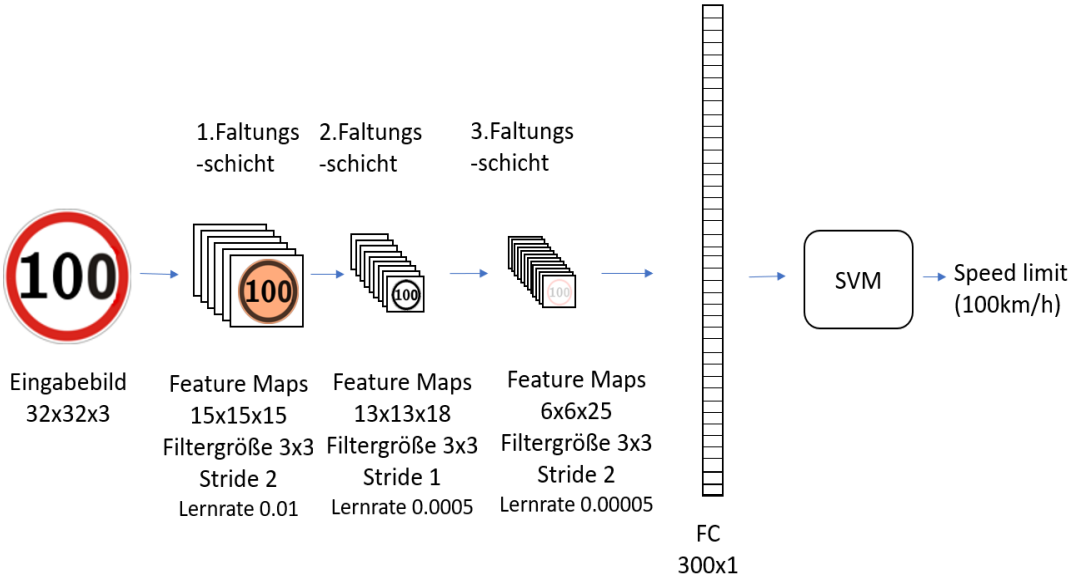


Abbildung 6.3: Eine Darstellung der Architektur des verwendeten Modells. Das Modell besteht aus drei CAE, einer vollständig verbundenen Schicht und einer SVM. Die Eingabe des Modells sind Bilder mit der Größe 32x32x3 (im Fall farbiger Bilder) und 32x32 (im Fall Graustufenbilder). Die Parameter der ersten CAE Schicht: 3x3 Filtergröße, Stride 2 und Lernrate 0.01. Die Parameter der zweiten CAE Schicht: 3x3 Filtergröße, Stride 2 und Lernrate 0.0005. Die Parameter der dritten CAE Schicht: 3x3 Filtergröße, Stride 1 und Lernrate 0.00005 (eigene Darstellung)

6.4 Ergebnisse des Datensatzes mit farbigen Bildern

Im Folgenden wird das Modell mit den oben genannten Einstellungen der Hyperparameter der Convolutional Autoencoder Schichten und der SVM Schicht und mit verschiedenen Batchgrößen vom Datensatz mit farbigen Bildern getestet. Die folgende Abbildung zeigt die quadratischen Fehler der drei CAE-Schichten mit zwei verschiedenen Batchgrößen. In Abbildung 6.2 (links) wird eine kleine Batchgröße bei jeder Schicht (CAE01: 50, CAE02: 250, CAE03: 750) verwendet und (rechts) wird eine große Batchgröße (CAE01: 100, CAE02: 1000, CAE03: 2500) verwendet. Mit kleinen Batchgrößen liefern die CAE-Schichten gute Ergebnisse, sodass die Filter in kurzer Zeit 7,5 Sekunden trainiert werden und die Eingabebilder wieder rekonstruieren können, aber der quadratische Fehler ist hoch, was auf die Erkennungsrate beim Klassifikator in der zweiten Phasen auswirken kann. Deswegen wird eine große Batchgröße beim Training verwendet, sodass das Modell bei der Klassifizierungsaufgabe eine gute Erkennungsrate auf Kosten der Trainingszeit erreichen kann, sodass das Training 26.7 Sekunden dauert.

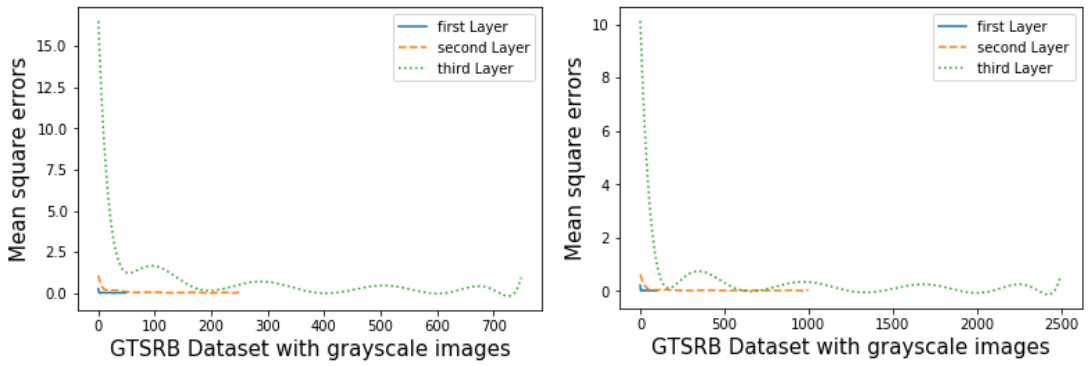


Abbildung 6.4: Die Evolution des quadratischen Fehlers bei der Verwendung drei Convolutional Autoencoder mit kleinen Batchgrößen (50,250,750) (links) und großen Batchgröße (100,1000,2500) (rechts) von farbigen Bildern. Die quadratischen Fehler der Schichten (links) ist hoher als die quadratischen Fehler (rechts). Die Trainingszeit ist (links) 7.5 Sekunden und (rechts) 26.7 Sekunden (Spyder Konsole)

Die Tabelle 6.3 repräsentiert die Trainingszeit und die Werte des quadratischen Fehlers jeder Schicht mit der verwendeten Batchgröße. Es ist auffällig, dass die tiefen Schichten langer Trainingszeit als die erste Schicht brauchen. Das liegt daran, dass die Lernrate kleiner ist und die Batchgröße und die Filteranzahl größer ist.

Tabelle 6.3: Die Trainingszeit und der maximale und minimale quadratische Fehler bei jeder CAE Schicht. In den tiefen Schichten ist die Lernrate kleiner als die vorhergien Schichten, deshalb erfordert das Training des CAEs mehr Proben zu lernen. Das hat zur Folge, dass die Trainingszeit dieser Schichten steigt.

Trainingsparameter	**CAE01**		**CAE02**		**CAE03**	
Batchgröße	100		1000		2500	
Trainingszeit	0.47 s		7,79 s		18,45 s	
Quadratische Fehler	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
	0.0003	0.201	- 0.0002	0.621	- 0.014	10.12

Die nachfolgende Abbildung zeigt die Entwicklung des quadratischen Fehlers bei jeder Schicht, sodass in jeder Schicht der quadratische Fehler bei einem hohen Wert beginnt, dann nähret sich mit der Zeit von Nullwert an. Aber je das Netzwerk tiefer ist, desto schwerer ist es, der quadratische Fehler der tiefen Schichten sich von Nullwert anzunähern.

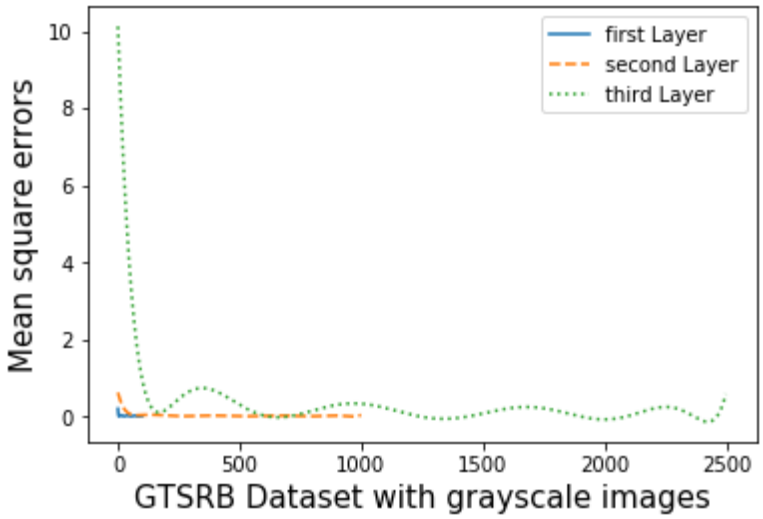
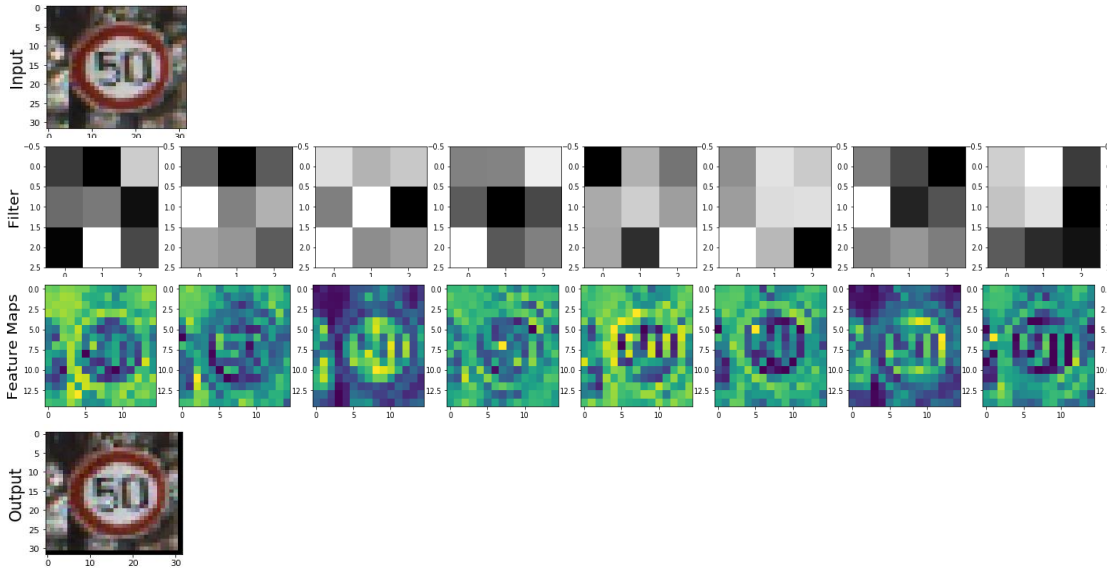
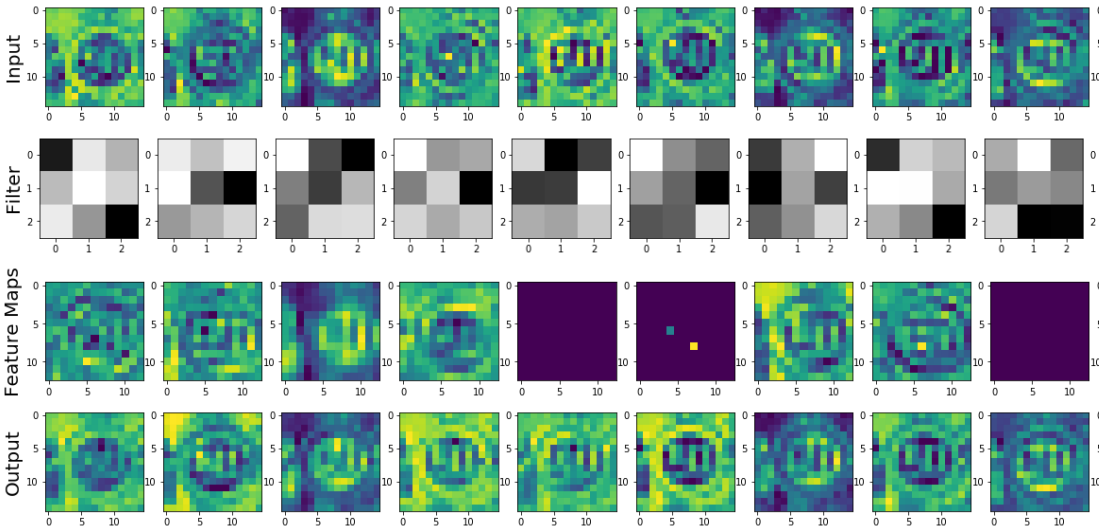


Abbildung 6.5: Die Evolution des quadratischen Fehlers bei der Verwendung drei Convolutional Autoencoder mit farbigen Bildern. Bei der ersten Schicht ist der quadratische Fehler nah von Null. Bei der zweiten Schicht fängt der quadratische Fehler von einem Wert nah von eins an, dann passt sich mit der Zeit an. Bei der dritten Schicht beginnt der quadratische Fehler von einem hohen Wert, dann entwickelt sich bis Wert nah von Null, aber er schwingt immer zwischen 0 und 1 (Spyder Konsole)

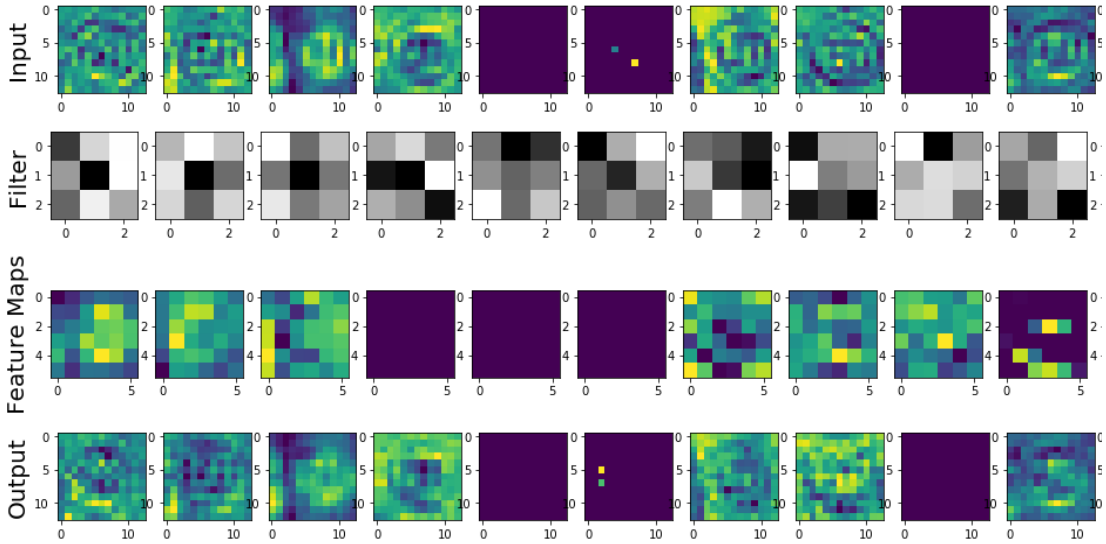
Nach dem Training wird eine Visualisierung der Ergebnisse der CAE-Schichten dargestellt. Die folgenden Abbildungen repräsentieren die Eingabebilder, die gelernten Filter, die Feature-Maps und die rekonstruierten Bilder. Es fällt auf, dass die Features in der ersten Schicht (Abbildung 6.4.a) sehr gut extrahiert werden und das Eingabe aus diesen Features wieder rekonstruiert wird. Bei der zweiten Schicht (Abbildung 6.4.b) treten ein paar Feature-Maps auf, die so wenig Informationen oder keine Informationen haben. Aber trotzdem kann die Eingabe aus diesen Feature-Maps wieder gut rekonstruiert werden. Bei der dritten Schicht haben einige Eingaben Nullwert. Das hat zur Folge, dass mehrere Feature-Maps keine Informationen beinhalten. Wenn mehr Feature-Maps keine Informationen haben, wirkt sich das negativ auf die Erkennungsrate aus.



a) Die Visualisierung des Eingabebilds (32x32x3), der Filter (3x3x15), der Feature-Maps (15x15x15) und des rekonstruierten Bildes (32x32x3) der ersten Schicht. Die Filter werden nach 100 Trainingsproben in 0.47 Sekunden gut gelernt und werden das Eingabebild gut rekonstruiert. Es wird nur 8 Filter und Feature-Maps ausgeschnitten, um die Ergebnisse besser zu zeigen. (Spyder Konsole)



b) Die Visualisierung der Eingabebilder (15x15x15), der Filter (3x3x18), der Feature-Maps (13x13x18) und der rekonstruierten Bilder (15x15x15) der zweiten Schicht. Die Filter werden nach 1000 Trainingsproben in 7.79 Sekunden gut gelernt und werden die Eingaben gut rekonstruiert. Es ist auffällig, dass ein paar Feature-Maps Nullwert haben oder fast null haben. Es wird nur 9 Eingaben, Filter, Feature-Maps und Rekonstruktion ausgeschnitten, um die Ergebnisse besser zu zeigen. (Spyder Konsole)



c) Die Visualisierung der Eingabebilder (13x13x18), der Filter (3x3x25), der Feature-Maps (6x6x25) und der rekonstruierten Bilder (13x13x18) der dritten Schicht. Die Filter werden nach 2500 Trainingsproben in 18,45 Sekunden gelernt und werden die Eingaben gut rekonstruiert. Es ist auffällig, dass in dieser Schicht mehr Feature-Maps Nullwert haben oder fast null haben. Es wird nur 10 Eingaben, Filter, Feature-Maps und Rekonstruktion ausgeschnitten, um die Ergebnisse besser zu zeigen (Spyder Konsole)

Abbildung 6.6: Die Visualisierung der Ergebnisse der drei CAE-Schichte, Hierbei ist die erste Zeile: Die Eingabe, die zweite Zeile: Die Filter, die dritte Zeile: Die Feature-Maps und die vierte Zeile: Die Rekonstruktion der Eingabe.

Bei der Klassifizierungsaufgabe werden zwei Klassifikatoren **OVA_SVM** und **SVC** getestet, wobei die beide SVM sind. Die Ergebnisse in der Tabelle 6.4 zeigen, dass der **OVA_SVM** Klassifikator mit weniger Trainingsdaten eine gute Erkennungsrate im Vergleich mit vielen Trainingsdaten erreichen kann. Es ist bemerkbar, dass der Klassifikator ein Trainingszeitproblem hat. Das liegt daran, dass der verwendete Lernalgorithmus in diesem Klassifikator viele Recherchen macht, um die optimalen Trennungshyperebenen mit maximalen Margen zu finden.

Tabelle 6.4: Die Trainingszeit, die Validierungs- und Erkennungsrate von OVA_SVM mit den obengenannten Hyperparametern anhand der Batchgröße mit farbigen Bildern

Batchgröße	**Zeit des Features Extrahieren**	**Trainingszeit**	**Validierungsrate**	**Erkennugsrate**
1K	6.73 s	154.24 s	76.02%	79.79%
2K	14.78 s	309.84 s	75.02%	76.85%
5K	34.73 s	835.01 s	77.63%	78.10%
10K	88.15 s	1912.07 s	65.64%	66.85%

Im Folgenden wird die Konfusionsmatrix des **OVA_SVM** Klassifikators mit der besten Erkennungsrate 79.79% für 1K Trainingsproben des Datensatz mit farbigen Bildern dargestellt, sodass die Spalten der Konfusionsmatrix die vorhergesagten Klassen repräsentiert und die Ziele der Konfusionsmatrix die realen Klassen repräsentiert. Es ist bemerkbar, dass die meisten falschen Klassen bei der „Speed limit 50km/h“ Klasse zugeordnet sind. Die „Keep right“ Klasse hat den besten Score in der Konfusionsmatrix. Die beste zugeordnete Klasse ist die „No entry“ Klasse, sodass sie nur fünf Mal falsch klassifiziert wird.

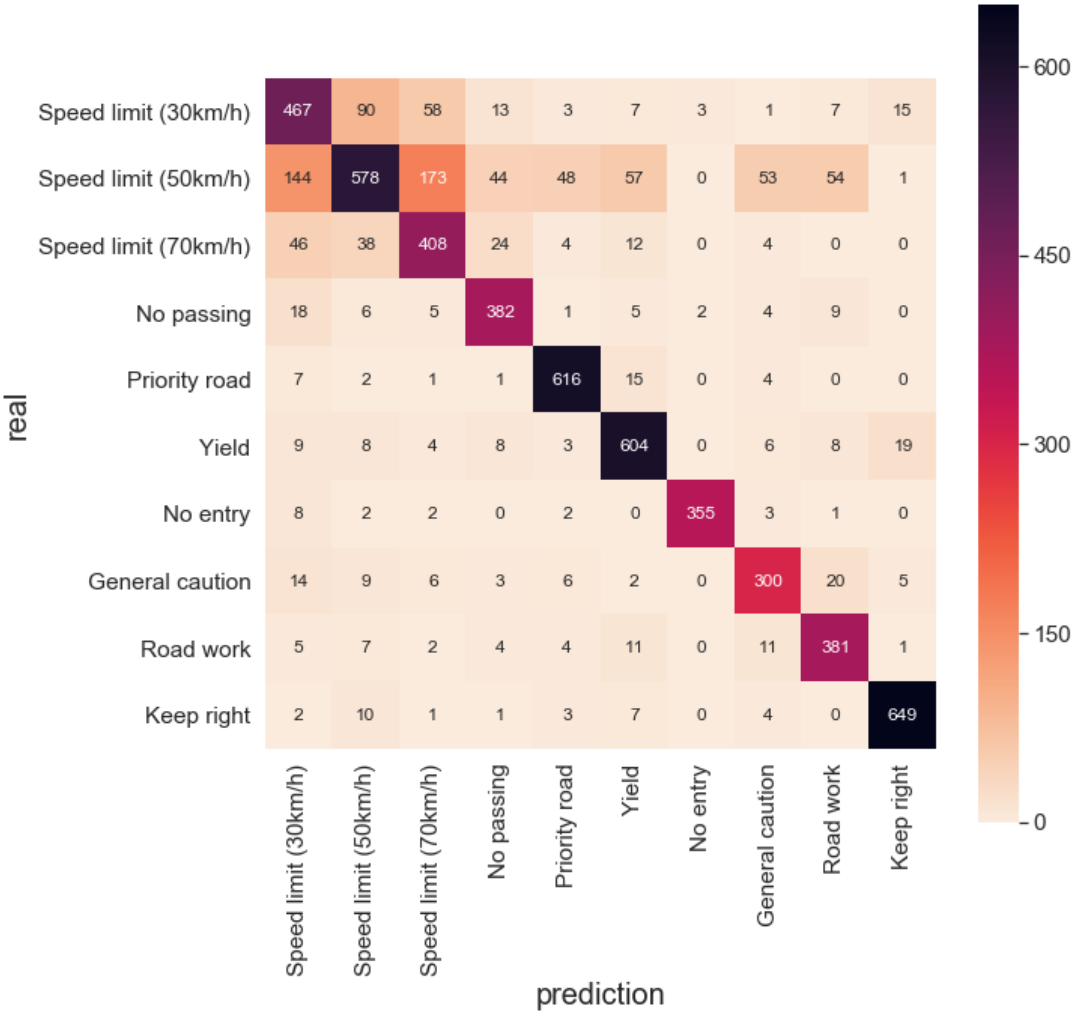


Abbildung 6.7: Die Konfusionsmatrix des OVA_SVM Klassifikators mit der besten Erkennungsrate 79.79% für 1K Trainingsproben des Datensatz mit farbigen Bildern. Die beste klassifizierte Klasse ist „Not entry“. Der beste Score gehört zur „Keep right“ Klasse. Die meisten Klassen sind zu den „Speed limit 50km/h“ zugeordnet (Spyder Konsole)

Die Tabelle 6.5 zeigt das Gegenteil der Tabelle 6.4, sodass je die Batchgröße größer ist, desto einer bessere Erkennungsrate geliefert werden kann. Andererseits ist der **SVC** zu schnell im Vergleich zum **OVA_SVM**.

Tabelle 6.5: Die Trainingszeit, die Validierungs- und Erkennungsrate von SVC mit den obengenannten Hyperparametern anhand der Batchgröße mit farbigen Bildern

Batchgröße	Zeit des Features Extrahieren	Trainingszeit	Validierungsrate	Erkennugsrate
1K	6.73 s	0.534 s	80.01%	81.55%
2K	14.78 s	1.556 s	86.32%	87.53%
5K	34.73 s	7.368 s	88.54%	90.96%
10K	88.15 s	26.04 s	91.42%	94.56%

Im Folgenden wird die Konfusionsmatrix des **SVC** Klassifikators mit der besten Erkennungsrate 94.56% für 10K Trainingsproben angezeigt. Aus der Konfusionsmatrix wird erkannt, dass die meisten falschen Vorhersagungen zwischen den ersten drei Klassen „Speed limit 30km/h, Speed limit 50km/h und Speed limit 70km/h“ sind. Das führt auf die Ähnlichkeit der drei Schilder zurück. Beispielweise hat die „Yield“ Klasse den höchsten Score in der Konfusionsmatrix, obwohl sie 14 Mal falsch klassifiziert. Anderseits hat die „No entry“ Klasse den geringsten Score, obwohl sie nur einmal falsch klassifiziert. Das liegt daran, dass die Verteilung der Klassen zwischen 1000 und 2000 Trainingsprobe pro Klasse variiert. Wegen des Trainingszeitproblems des Klassifikators wird das Zeil der Arbeit durch **OVA_SVM** nicht erreicht, sondern durch **SVC.**

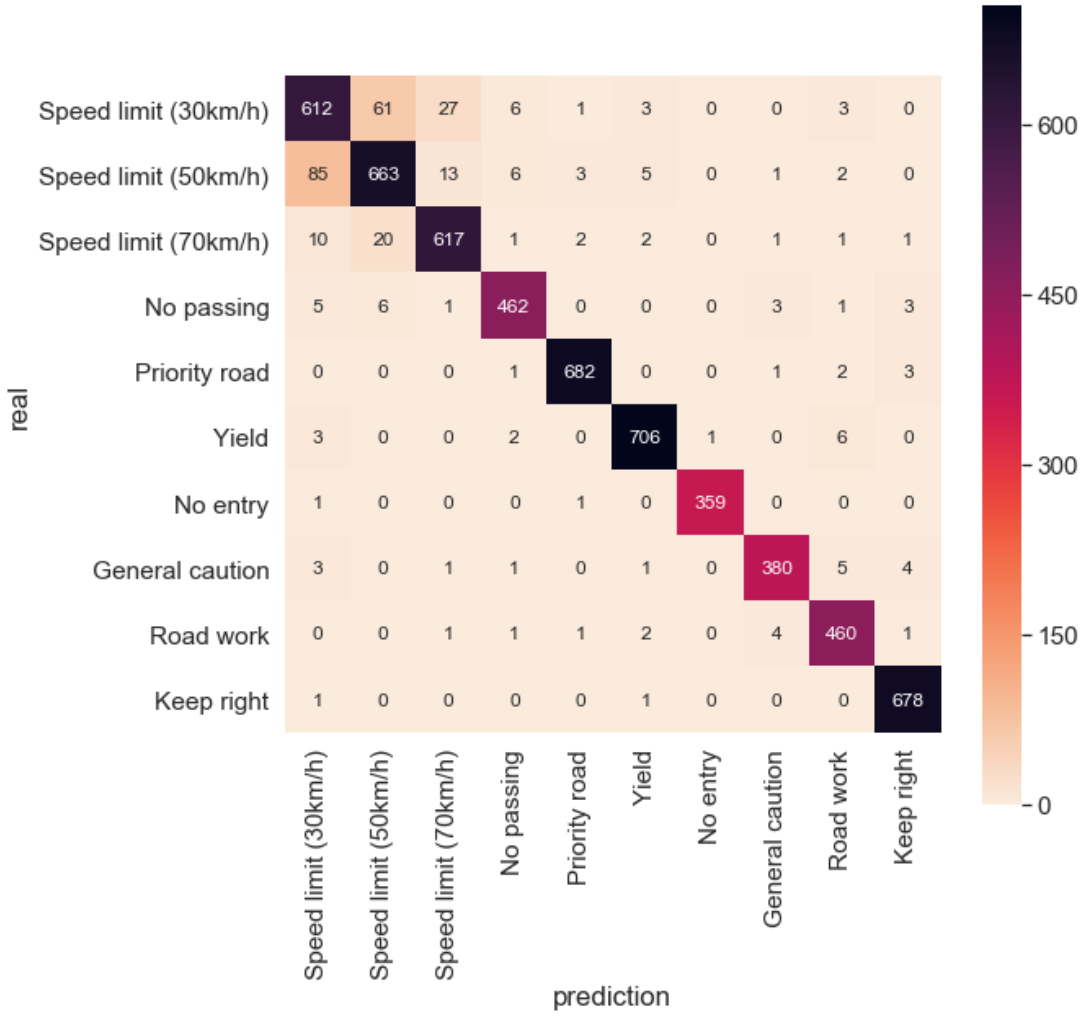


Abbildung 6.8: Die Konfusionsmatrix des SVC Klassifikators mit der besten Erkennungsrate 94.56% für 10K Trainingsproben des Datensatz mit farbigen Bildern. Die beste Score gehört zur „Yield“ Klasse. Die beste klassifizierte Klasse ist „ No entry“ Klasse. (Spyder Konsole)

6.5 Ergebnisse vom Datensatz mit Graustufenbildern

Das Modell wird mit den Graustufenbildern getestet, zu sehen, ob es bessere Ergebnisse liefert. Im Folgenden werden die Ergebnisse des Modells für die gleichen Hyperparameter im Kapitel 6.4 in der Tabelle 6.6 präsentiert.

Tabelle 6.6: Die Trainingszeit und der quadratische Fehler bei jeder CAE Schicht. In den tiefen Schichten erfordert das Training des CAEs mehr Proben zu lernen. Das hat zur Folge, dass die Trainingszeit dieser Schichten steigt.

Trainingsparameter	**CAE01**		**CAE02**		**CAE03**	
Batchgröße	100		1000		2500	
Trainingszeit	0.446 s		7.696 s		17.687 s	
Quadratische Fehler	Min	Max	Min	Max	Min	Max
	0.0004	0.114	-0.0015	0.47	-0.096	6.857

Es ist erkannt, dass der quadratische Fehler bei den Graustufenbildern kleiner als bei den farbigen Bildern ist. Die Trainingszeit der ersten Schicht ist weniger als bei den farbigen Bildern, weil die Graustufenbilder aus einem Kanal besteht und nicht drei wie die farbigen Bilder. Die Trainingszeit der anderen Schichte ist gleich wie bei den farbigen Bildern, was darauf zurückführt, dass die Einstellungen gleich sind.

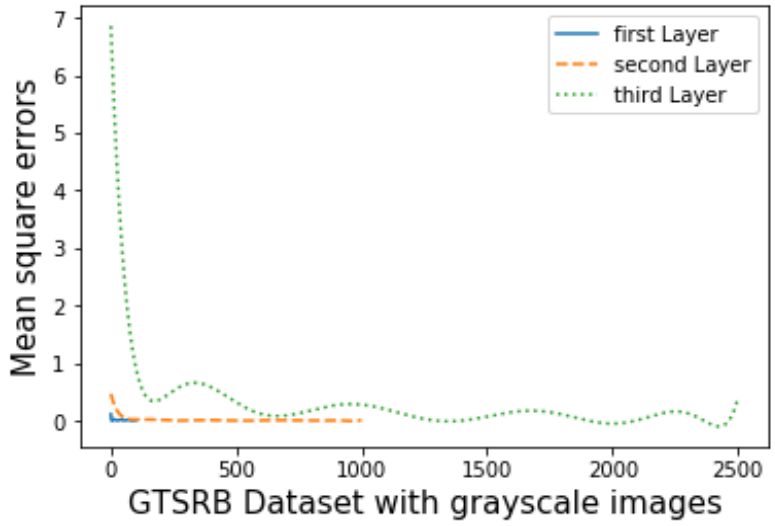
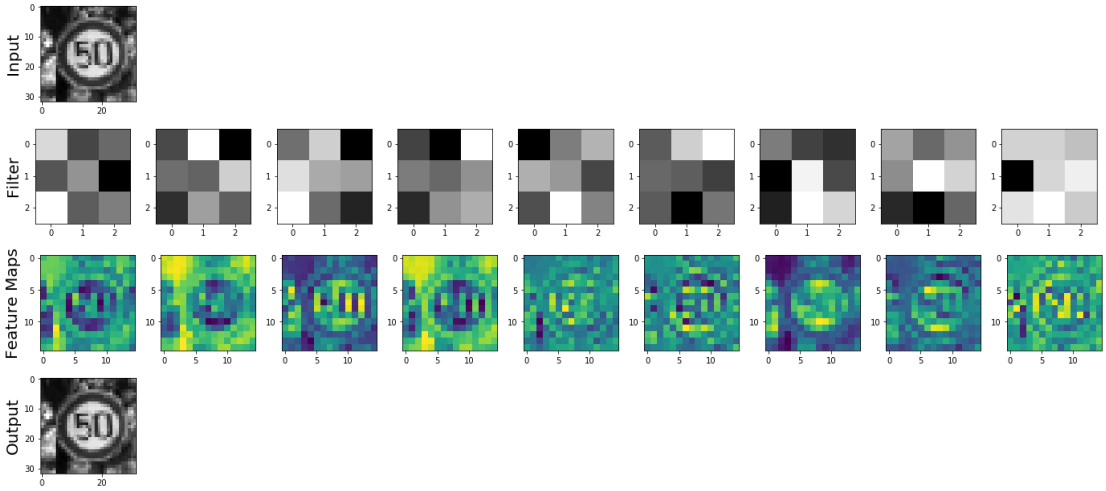
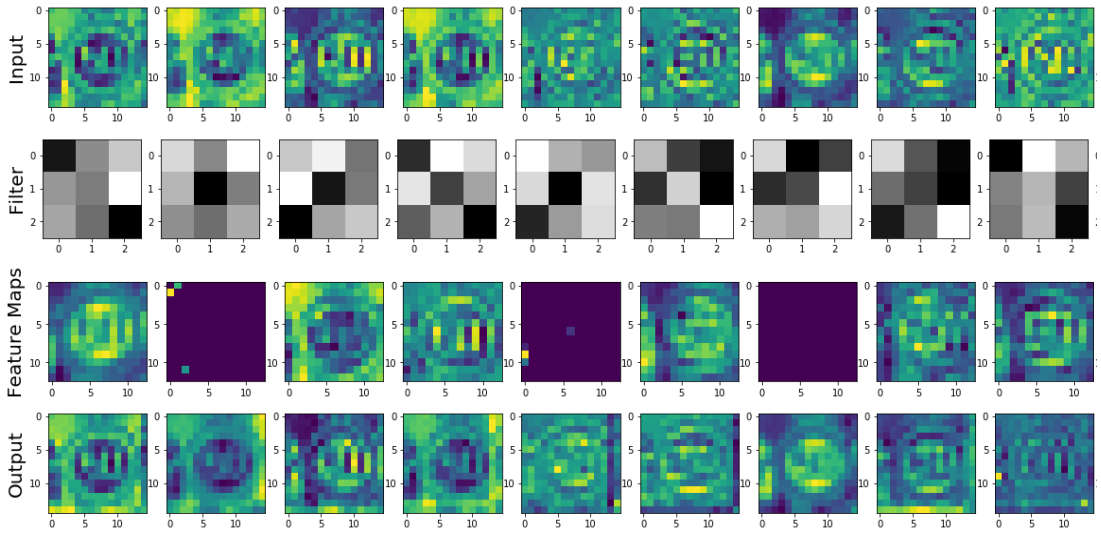


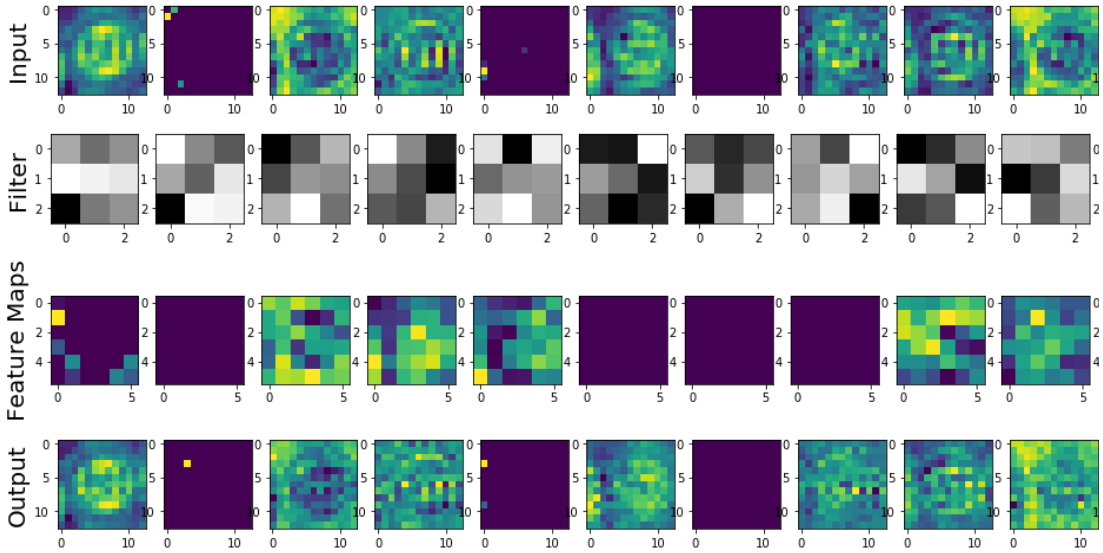
Abbildung 6.9: Die Evolution des quadratischen Fehlers bei der Verwendung drei Convolutional Autoencoder mit Graustufenbildern. Bei der ersten Schicht ist der quadratische Fehler nah von Null. Bei der zweiten Schicht fängt der quadratische Fehler von einem Wert nah von eins an, dann passt sich mit der Zeit an. Bei der dritten Schicht beginnt der quadratische Fehler von einem hohen Wert, dann entwickelt sich bis Wert nah von Null, aber er schwingt immer zwischen 0 und 1 (Spyder Konsole)



a) Die Visualisierung des Eingabebilds (32x32), der Filter (3x3x15), der Feature-Maps (15x15x15) und des rekonstruierten Bildes (32x32) der ersten Schicht. Die Filter werden nach 100 Trainingsproben in 0.446 Sekunden gut gelernt und werden das Eingabebild gut rekonstruiert. Es wird nur 9 Filter und Feature-Maps ausgeschnitten, um die Ergebnisse besser zu zeigen. (Spyder Konsole)



b) Die Visualisierung der Eingabebilder (15x15x15), der Filter (3x3x18), der Feature-Maps (13x13x18) und der rekonstruierten Bilder (15x15x15) der zweiten Schicht. Die Filter werden nach 1000 Trainingsproben in 7.67 Sekunden gut gelernt und werden die Eingaben gut rekonstruiert. Es ist auffällig, dass ein paar Feature-Maps Nullwert haben oder fast Null haben. Es wird nur 9 Eingaben, Filter, Feature-Maps und Rekonstruktion ausgeschnitten, um die Ergebnisse besser zu zeigen. (Spyder Konsole)



c) Die Visualisierung der Eingabebilder (13x13x18), der Filter (3x3x25), der Feature-Maps (6x6x25) und der rekonstruierten Bilder (13x13x18) der dritten Schicht. Die Filter werden nach 2500 Trainingsproben in 18,45 Sekunden gelernt und werden die Eingaben gut rekonstruiert. Es ist auffällig, dass in dieser Schicht mehr Feature-Maps Nullwert haben oder fast Null haben. Es wird nur 10 Eingaben, Filter, Feature-Maps und Rekonstruktion ausgeschnitten, um die Ergebnisse besser zu zeigen (Spyder Konsole)

Abbildung 6.10: Die Visualisierung der Ergebnisse der drei CAE-Schichte mit Graustufenbildern, Hierbei ist die erste Zeile: Die Eingabe, die zweite Zeile: Die Filter, die dritte Zeile: Die Feature-Maps und die vierte Zeile: Die Rekonstruktion der Eingabe.

Die Ergebnisse der **OVA_SVM** Klassifikator mit verschiedenen Batchgrößen werden in der Tabelle 6.7 angezeigt. Es erscheint, dass die Erkennungsrate bei den Graustufenbilder schlechter als bei den farbigen Bildern. Das liegt daran, dass das Model sich nur auf die Form der Bilder konzertiert und nicht mehr auf die Farbe. D.h. weniger Merkmale werden an den Klassifikator geliefert. Die Ergebnisse zeigen eine 65.71% Erkennungsrate bei 1K Batchgröße, Anschließend schwingt die Erkennungsrate bei den 2K, 5K und 10K zwischen 65% und 50%.

Tabelle 6.7: Die Trainingszeit, die Validierungs- und Erkennungsrate von OVA_SVM mit den obengenannten Hyperparametern anhand der Batchgröße mit Graustufenbilder

Batchgröße	**Zeit des Features Extrahieren**	**Trainingszeit**	**Validierungsrate**	**Erkennugsrate**
1K	7.09 s	146.60 s	61.52%	65.71%
2K	17.55 s	299.39 s	50.94%	53.31%
5K	33.71 s	793.12 s	61.57%	65.30%
10K	75.24 s	1501.3 s	48.28%	49.26%

Hiermit wird die Konfusionsmatrix des **OVA_SVM** Klassifikators mit der besten Erkennungsrate 65.71% für 1K Trainingsproben des Datensatz mit Graustufenbildern dargestellt. Das Klassifizierungsergebnis ist allgemein schlecht, sodass viele Daten falsch klassifiziert werden.

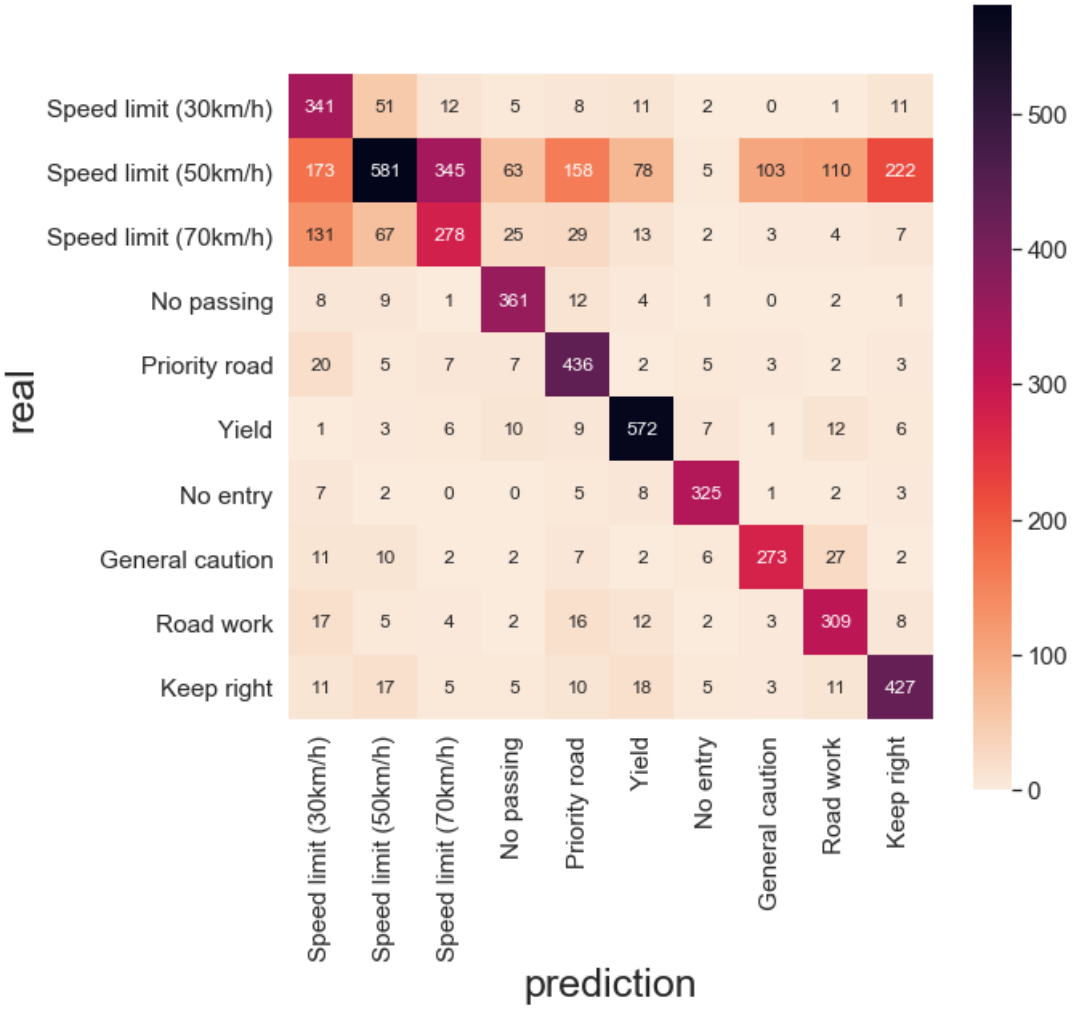


Abbildung 6.11: Die Konfusionsmatrix des OVA_SVM Klassifikators mit der besten Erkennungsrate 65.71% für 1K Trainingsproben des Datensatz mit farbigen Bildern. Die beste klassifizierte Klasse ist „Not entry“. Der beste Score gehört zur „Yield“ Klasse. Die meisten Klassen sind zu den „Speed limit 50km/h“ zugeordnet (Spyder Konsole)

Die Tabelle 6.8 zeigt die Ergebnisse der **SVC** mit Graustufenbildern, sodass die Ergebnisse ähnlich von den Ergebnissen bei der Tabelle 6.6 von den farbigen Bildern. Hier verbessert sich die Erkennungsrate auch, wenn die Batchgröße größerer ist. Darüber hinaus ist die Trainingszeit kürzer als die Trainingszeit bei **OVA_SVM**.

Tabelle 6.8: Die Trainingszeit, die Validierungs- und Erkennungsrate von SVC mit den obengenannten Hyperparametern anhand der Batchgröße mit Graustufenbilder

Batchgröße	Zeit des Features Extrahieren	Trainingszeit	Validierungsrate	Erkennugsrate
1K	7.09 s	0.65 s	77.35%	78.67%
2K	17.55 s	1.96 s	82.78%	83.40%
5K	32.71 s	9.60 s	86.88%	90.0%
10K	65.92 s	30.47 s	87.87%	92.13%

In der Konfusionsmatrix des **SVC** Klassifikator mit den Graustufenbildern wird auch wie bei den farbigen Bildern angezeigt. Die meiste Score gehört zur „Yield“ Klasse. Während die „Speed limit 30km/h“ und „Speed limit 50km/h“ Klassen am meisten falsch klassifiziert werden, wird die „No entry“ Klasse am besten klassifiziert.

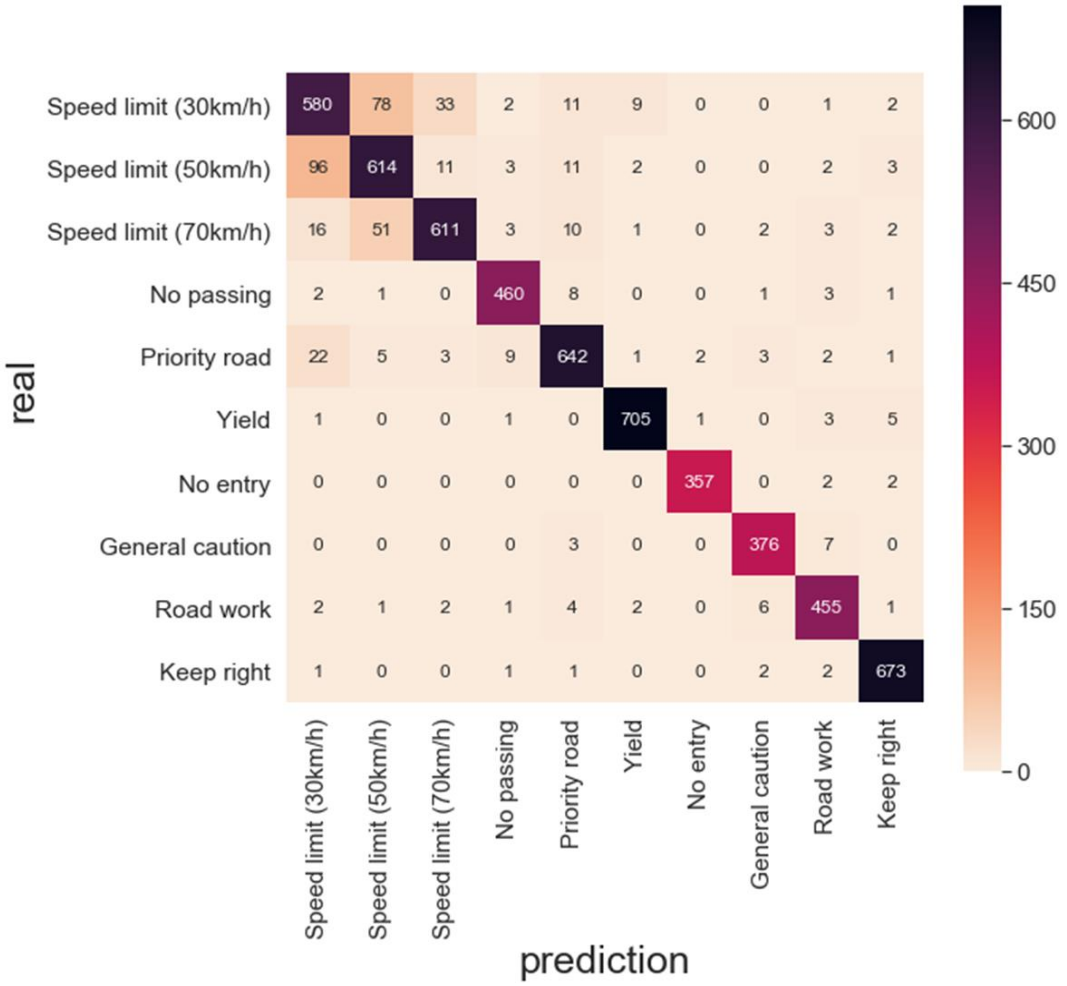


Abbildung 6.12: Die Konfusionsmatrix des SVC Klassifikators mit der besten Erkennungsrate 92.13% für 10K Trainingsproben des Datensatz mit Graustufenbildern. Der beste Score gehört zur „Yield“ Klasse. Die beste klassifizierte Klasse ist „No entry“ Klasse. (Spyder Konsole)

7 Fazit und Ausblick

In dieser Arbeit wird das Klassifizierungsproblem der Verkehrszeichen des GTSRB Datensatzes zur Navigation selbstfahrender Autos behandelt. Es wird besonders häufig die CNN Architektur mit dem Backpropagation Lernalgorithmus für Bildklassifizierung verwendet. Diese Struktur nimmt jedoch viel Zeit beim Training in Anspruch. Zum Lösen dieses Klassifizierungsproblems wird in dieser Arbeit ein Modell aus Convolutional Autoencoder und Support Vector Machine entwickelt, sodass der Convolutional Autoencoder die Filter zum Extrahieren der Merkmale aus den Bildern vortrainiert und die SVM zur Klassifizierung verwendet wird. Die Filter des Convolutional Autoencoder werden anhand des Contrastive Divergence Lernalgorithmus von RBM anstatt des Backpropagation Lernalgorithmus trainiert. Um ein tiefes Netzwerk von Convolutional Autoencodern zu erstellen, wird das Greedy-Layer-Wise Trainingsverfahren zum Erstellen der Verbindung zwischen den Schichten verwendet. Die SVM Schicht wird durch den SMO Lernalgorithmus zum Finden der optimalen Hyperebenen mit maximalen Margen zwischen den Klassen trainiert. Anschließend werden die Daten klassifiziert. Das Modell wird mit verschiedenen Einstellungen der Hyperparameter getestet, um die passenden Hyperparameter zu finden. Die Convolutional-Autoencoder Schichten in diesem Modell erreichen gute Ergebnisse, sodass die Eingabe in kurzer Zeit gut rekonstruiert wird. Für den Datensatz mit farbigen Bildern brauchen die Filter in den drei Schichten 26,7 Sekunden zum Trainieren. Nach dem Training werden die Merkmale aus einem Batch aus dem Datensatz extrahiert, dann in einem Vektor für die Klassifizierung hinzufügt. Bei der Klassifizierungsphase mittels **OVA_SVM** hängt die Trainingszeit von der Batchgröße ab, sodass bei einer 10K Batchgröße 1912,17 Sekunden dauert. Außerdem wird eine schlechte Erkennungsrate von 66.85% geliefert. Diese lange Trainingszeit führt auf der Suche nach einer Lösung des Optimierungsproblems bei SVM zurück. Im Gegenteil dazu dauert die Klassifizierung bei der Verwendung der **SVC** Funktion nur 26,04 Sekunden und liefert eine bessere Erkennungsrate von 94,56%. Für den Datensatz mit Graustufenbildern werden die Filter in 25,8 Sekunden trainiert. Obwohl die Feature der Bilder in Convolutional Schichten gut extrahiert werden, erreicht der OVA_SVM Klassifikator nach 1501 Sekunden so niedrige Erkennungsrate unter Verwendung der 10K Batchgröße. Anderseits liefert der SVC Klassifikator nach 30 Sekunden eine Erkennungsrate von 92.13% für eine 10K Batchgröße. Basierend auf diesem Ergebnis, wäre es besser anstatt die **OVA_SVM** Funktion, die **SVC** Funktion im Modell einzusetzen, da die **OVA_SVM** ihre Klassifizierungsaufgabe nicht wie erwartet erfüllen kann. Zur Implementierung dieses Modells auf einem selbstfahrenden Auto muss die Erkennungsrate des Modells nah an 100% sein, da ein kleiner Fehler beim selbstfahrenden Auto zu einem Unfall führen kann. Aus diesem Grund muss die Erken-

nungsrate verbessert werden. Diese Verbesserung könnte durch die Entwicklung einer Methode gewährleistet werden, die mehr Features pro Schicht extrahieren und nur die Feature-Maps, die Informationen haben, auswählen und an die nächste Schicht weitergeben kann. Eine andere Möglichkeit wäre die Verwendung eines Dropouts. Diese Technik ist eine Regularisierungsmethode, die die Überanpassung im Netzwerk verringert. Dabei wird beim Training des Netzwerks eine vorher spezifizierte Anzahl von Neuronen ausgeschaltet und für den kommenden Berechnungsschritt nicht berücksichtigt. Diese Methode würde nur zwischen der letzten Faltungsschicht und dem SVM Klassifikator implementiert werden, weil die Faltungsschichten nicht viele Parameter haben und die Überanpassung kein Problem für sie darstellt. Eine Weiterentwicklung des Modells könnte durch die Erkennung der Verkehrszeichnen und im Anschluss an ihre Klassifizierung dargestellt werden. In diesem Fall wäre es möglich, das Modell bei einem Roboter mit einer Kamera einzusetzen und zu testen.

Abkürzungsverzeichnis

ML Maschinelles Lernen

DL Deep Learning

KI Künstliche Intelligenz

KNN Künstliches neuronales Netzwerk

MLP Multi-Layer Perceptron

CNN Convolutional neurol Network

CAE Convolutional Autoencoder

BM Boltzmann Machine

RBM Restricted Boltzmann Machine

SVM Support Vector Machine

OVA One Versus All

OVO One Versus One

DBN Deep Belief Network

SMO Sequential Minimal Optimization

rbf Radiale Basisfunktion Kernel

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] P. Sermanet und Y. LeCun, „Traffic Sign Recognition with Multi-Scale Convolutional Networks,“ in *IJCNN* , San Jose, CA, United States, 2011.

[2] X. Wang, Y. Zhu, C. Yao und X. Bai, „Traffic Sign Classification Using Tow-Layer Image Representation,“ in *IEEE International Conference on Image Processing*, Melbourne, Australia, 2013.

[3] D. Gross, „Maschinelle Bilderkennung mit Big Data und Deep Learning,“ 03 01 2017. [Online]. Available: https://jaxenter.de/big-data-bildanalyse-50313. [Zugriff am 05 04 2019].

[4] I. B. Y. &. C. A. Goodfellow, Deep Learning, MIT press, 2016.

[5] Y. B. Y. &. H. G. LeCun, „Deep Learning,“ in *nature*, 2015, pp. 521, pages 436–444.

[6] S. Schönbrodt, Maschinelle Lernmethoden für Klassifizierungsprobleme, Springer Spektrum, 2019, ISBN: 978-3-658-25137-6.

[7] F. C. M. &. S. F. Pesapane, „Artificial intelligence in medical imaging: threat or opportunity? Radiologists again at the forefront of innovation in medicine. European radiology experimental,“ Springer, 24 10 2018. [Online]. Available: https://eurradiolexp.springeropen.com/articles/10.1186/s41747-018-0061-6. [Zugriff am 25 08 2019].

[8] J. &. G. A. Patterson, Deep Learning A Partitioner’s Approach, O'Reilly Media, Inc, 2017.

[9] A. S. Walia, „Activation functions and it’s types- Which is better,“ 12 04 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Zugriff am 2019].

[10] D. Cupta, „Fundamentals of Deep Learning- Activation Functions and When to Use Them?,“ 10 06 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/. [Zugriff am 2019].

[11] F.-F. Li, S. Yeung und J. Johanson, „Lecture 7: Training Neural Networks, Part I,“ 2019. [Online]. Available: http://vision.stanford.edu/teaching/cs231n/slides/2019/cs231n_2019_lecture07.pdf. [Zugriff am 02 06 2019].

[12] H. H. Aghdam und . E. J. Heravi, Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification, Switzerland: Springer International Publishing, 2017, ISBN: 978-3-319-57550-6.

[13] H. &. R. E. Süße, Bildverarbeitung und Objekterkennung, Springer Vieweg, 2014, ISBN: 978-3-8348-2606-0.

[14] G. R. A. &. C. S. Rebala, An Introduction to Machine Learning, Springer, 2019, ISBN: 978-3-030-15729-6.

[15] V. &. V. F. Dumoulin, „A guide to convolution arithmetic for deep learning," arXiv preprint arXiv:1603.07285, 2016.

[16] J. Wu, „Convolutional neural networks," Semanticscholar, National Key Lab for Novel Software Technology. Nanjing University, 2019.

[17] „CS231 Convolutional Neural Networks for Visual Recognition.," 2019. [Online]. Available: http://cs231n.github.io/convolutional-networks/#layers. [Zugriff am 13 03 2019].

[18] G. S. R. &. J. H. Tolias, „Particular object retrieval with integral max-pooling of CNN activations," arXiv preprint arXiv:1511.05879., 2016.

[19] A. Vieira, „Predicting online user behaviour using deep learning algorithms," arXiv preprint arXiv:1511.06247, 2016.

[20] V. C. E. &. L. A. Turchenko, „A Deep Convolutional Auto-Encoder with Pooling - Unpooling Layers in Caffe," arXiv preprint arXiv:1701.04949., 2017.

[21] X. J. S. C. &. Y. Y. B. Mao, „Image Restoration Using Convolutional Autoencoders with Symmetric Skip Connections," arXiv preprint arXiv:1606.08921, 2016.

[22] A. B. A. S. E. &. C. P. Barra, „On the equivalence of Hopfield Networks and Boltzmann Machines," pp. 34, 1-9, 10 Jan 2012.

[23] M. A. &. H. G. E. Carreira-Perpinan, „On Contrastive Divergence Learning," *AISTATS 2005 Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics,* pp. Vol. 10, pp. 33-40, 06 08 2005.

[24] L. Ning, R. Pittman und X. Shen, „LCD: A Fast Contrastive Divergence Based Algorithm for Restricted Boltzmann Machine," North Carolina State University, 2017.

[25] G. E. Hinton, „A Practical Guide to Training Restricted Boltzmann Machines," in *In Neural networks: Tricks of the trade,* Springer, 2012, pp. 599-619.

[26] R. &. A. A. Sathya, „Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification," in *nternational Journal of Advanced Research in Artificial Intelligence,* 2013.

[27] A. Radford, L. Metz und S. Chintala, „Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

[28] A. N. A. &. L. H. Coates, „An Analysis of Single-Layer Networks in Unsupervised

Feature Learning,“ in *In Proceedings of the fourteenth international conference on artificial intelligence and statistics*, Ft. Lauderdale, FL, USA, 2011.

[29] R. S. &. B. A. G. Sutton, Reinforcement Learning: An Introduction Second edition, in progress ****Complete Draft****, Cambridge: MIT press, 2017.

[30] S. Abe, Support Vector Machines for Pattern Classification, London: Springer, 2010, ISBN: 978-1-84996-098-4 .

[31] „CS231n Convolutional Neural Networks for Visual Recognition,“ 25 06 2019. [Online]. Available: http://cs231n.github.io/linear-classify/.

[32] L. V. E. D. C. A. P. M. &. V. A. Rosasco, „Are Loss Functions All the Same?,“ in *Neural Computation*, Neural Computation, MIT Press, 2004, pp. 16(5), 1063-1076.

[33] D. Macedo, „Improving Image Classification Accuracy using Hybrid Systems of Support Vector Machines and Convolutional Neural Networks,“ 15 06 2016. [Online]. Available: https://github.com/dlmacedo/SVM-CNN/blob/master/svm-cnn-article.pdf. [Zugriff am 13 07 2019].

[34] s.-l. developers, „Scikit Learn,“ 2007-2019. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Zugriff am 05 08 2019].

[35] A. Ng, „Support Vector Machines,“ in *CS229 Lecture notes*, Backspaces, 2019.

[36] J. Wu, „Support Vector Machine,“ National Key Lab for Novel Software Technology, Nanjing University, China, LAMDA Group, 2019.

[37] C. J. Burges, „A Tutorial on Support Vector Machines for Pattern,“ in *Data mining and knowledge discovery*, Springer-Data Mining and Knowledge Discovery, 1998, pp. 2(2), 121-167.

[38] Z. Zhang, „Customizing kernels in Support Vector Machines,“ 22 05 2007. [Online]. Available: https://uwspace.uwaterloo.ca/handle/10012/3063. [Zugriff am 16 07 2019].

[39] G. &. G. D. Madzarov, „Multi-class classification using support vector machines in decision tree architecture,“ IEEE EUROCON 2009, 2009.

[40] B. F. A. &. H. J. Wicht, „On CPU performance optimization of Restricted Boltzmann Machine and Convolutional RBM,“ *Artificial Neural Networks in Pattern Recognition* , pp. 163-174, 2016.

[41] G. E. O. S. &. T. Y. W. Hinton, „A Fast Learning Algorithm for Deep Belief Nets,“ in *Neural computation*, MIT Press, 2006, pp. 18(7), 1527-1554.

[42] Y. L. P. P. D. &. L. H. Bengio, „Greedy Layer-Wise Training of Deep Networks,“ in *In Advances in neural information processing systems*, 2007, pp. 153-160.

[43] N. P. D. &. M. M. K. Agarwalla, „Deep Learning using Restricted Boltzmann Machines,“ in *International Journal of Computer Science and Information*

Technologies, 2016.

[44] J. Platt, „Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," 1998.

[45] A. Ng, „CS229 Lecture notes," 21 04 2019. [Online]. Available: http://cs229.stanford.edu/notes/cs229-notes3.pdf. [Zugriff am 13 08 2019].

[46] L. &. L. C. J. Bottou, „Support Vector Machine Solvers," in *Support vector machine solvers. Large scale kernel machines, 3(1), 301-320.*, Researchgate, 2007.

[47] „GTSRB Dataset," 2010. [Online]. Available: http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset. [Zugriff am 25 05 2019].

[48] M. M. R. &. K. M. Elleuch, „A new design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition," in *Procedia Computer Science*, Elsevier B.V., 2016, pp. 1712-1723.

[49] F. J. &. L. Y. Huang, „Large-scale learning with svm and convolutional nets for generic object categorization," in *Computer Vision and Pattern Recognition Conference (CVPR'06)*, 2006.

[50] M. &. K. R. Awad, Efficient learning machines: theories, concepts, and applications for engineers and system designers, Berkeley, CA: Apress, 2015, ISBN:978-1-4302-5990-9.

[51] V. a. V. F. Dumoulin, „A guide to convolution arithmetic for deep learning," arXiv preprint arXiv:1603.07285, 2016.

[52] „CS231n Convolutional Neural Networks for Visual Recognition," 2019. [Online]. Available: http://cs231n.github.io/understanding-cnn/.

[53] K. Simonyan und K. Zisserman, „Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, Published as a conference paper at ICLR 2015, 2014.

[54] M. Kornmeier, Wissenschaftlich schreiben leicht, 4. Hrsg., UTB, 2011.

[55] P. Vixie, „DNS complexity," *Queue*, p. 24, April 2007.

[56] A. Einstein, „ Zur Elektrodynamik bewegter Körper," *Annalen der Physik,* pp. 891-912, September 1905.