



FOM Hochschule für Oekonomie & Management

Hochschulzentrum Düsseldorf

Master Thesis

im Studiengang Big Data and Business Analytics

zur Erlangung des Grades eines

Master of Science (M.Sc.)

über das Thema

**Evaluierung und Visualisierung des Lernprozesses eines CNN zur
Kategorisierung deutscher Straßenverkehrsschilder**

von

Robin Maasjosthusmann

Erstgutachter : Prof. Dr. Frank Lehrbass

Matrikelnummer : 498595

Abgabedatum : 3. November 2020

Inhaltsverzeichnis

1 Theoretische Grundlagen	1
1.1 Straßenverkehrsschilder	1
1.1.1 Straßenverkehrsschilder in Deutschland	1
1.1.2 German Traffic Sign Recognition Benchmark	2
1.2 Neural Networks	3
1.2.1 Aufbau	4
1.2.2 Gradient Descent	6
1.2.3 Backpropagation	10
1.3 Convolutional Neural Networks	14
1.3.1 Biologische Grundlagen	15
1.3.2 Architektur	16
1.3.2.1 Local Receptive Fields	16
1.3.2.2 Gemeinsame Gewichte und Bias	17
1.3.2.3 Pooling	18
1.4 Visualisierung der Entscheidung	20
1.4.1 Activation Maximization	20
1.4.2 Saliency Map	22
1.4.3 Class Activation Mapping	23
1.5 Bibliotheken	25
Anhang	26
Literaturverzeichnis	27

Abbildungsverzeichnis

Tabellenverzeichnis

1	Übersicht der höchsten erreichten Genauigkeiten auf den GTSRB Datensatz	2
---	---	---

Abkürzungsverzeichnis

AM	Activation Maximization
CAM	Class Activation Mapping
CNN	Convolutional Neural Network
GD	Gradient Descent
GTSRB	German Traffic Sign Recognition Benchmark
KI	Künstliche Intelligenz
LRF	Local receptive Fields
MASTIF	Mapping and Assessing the State of Traffic InFrastructure
NN	Neural Network
RGB	Rot, Grün, Blau
SGD	Stochastic Gradient Descent
SM	Saliency Map
StVO	Straßenverkehrsordnung

1 Theoretische Grundlagen

In diesem Kapitel soll eine Übersicht über die wichtigsten theoretischen Grundlagen der Arbeiten gegeben werden. Die relevanten Fachbegriffe werden definiert und eine Herleitung der mathematischen Grundlagen der angewendeten Methoden wird dargestellt. Im folgenden Abschnitt wird auf Straßenverkehrsschilder und den verwendeten Datensatz eingegangen. Anschließend werden wichtige Aspekte von Neural Network (NN) und Convolutional Neural Network (CNN) erläutert und hergeleitet (Abschnitte 1.2 bzw. 1.3). Abschließend werden die verwendeten Algorithmen zur Visualisierung des Gelernten des CNN (Abschnitt 1.4) und die Python Bibliotheken die verwendet werden (Abschnitt 1.5) vorgestellt.

1.1 Straßenverkehrsschilder

In diesem Abschnitt soll eine kurze Definition von Straßenverkehrsschilder so wie eine Vorstellung der in Deutschland geltenden Rechtslage dieser gegeben werden. Außerdem wird ein Überblick, über verfügbare Datensätze mit Schwerpunkt auf den German Traffic Sign Recognition Benchmark (GTSRB) Datensatz gegeben.

1.1.1 Straßenverkehrsschilder in Deutschland

Straßenverkehrsschilder dienen dazu, den Straßenverkehr zu regeln. Hierbei stellen sie den Verkehrsteilnehmern Warnungen, Informationen und Einzelheiten über Einschränkungen zur Verfügung. Sie fallen dabei unter die Straßenausstattung und werden behördlich angeordnet. Alle gültigen Straßenverkehrsschilder in Deutschland werden in der Straßenverkehrsordnung (StVO) gelistet. Diese wird fortlaufend den aktuellen Gegebenheiten angepasst und durch Novellen aktualisiert. Grundsätzlich basieren die Straßenverkehrsschilder dabei auf dem am 8. November 1968 in Wien beschlossenen *Übereinkommen über den Straßenverkehr* (engl. Originaltitel "Convention on Road Traffic").¹ Dieses wurde bis heute von 36 Nationen unterzeichnet und von 84 Nationen ratifiziert.²

Beleg

In der StVO sind drei Gruppen von Verkehrszeichen definiert:

- Gefahrenzeichen (§ 40 StVO)
- Vorschriftzeichen (§ 41 StVO)

¹ Vgl. UN, 1977, o.S.

² Vgl. UN, 2020, S. 1–14.

Tabelle 1: Übersicht der höchsten erreichten Genauigkeiten auf den GTSRB Datensatz

Team	Methode	Prozentzahl (Genauigkeit)
Arcos-García, Álvarez-García, Soria-Morillo	CNN mit 3 Spatial Transformers	99.71
Ciresan, Meier, Masci, Schmidhuber	Zusammenschluss mehrerer CNNs	99.46
Gecer, Azzopardi, Petkov	Farbenbasierter COSFIRE-Filter zur Objekterkennung	98.97
Stallkamp, Schlipsing, Salmen, Igel	Durchschnittliches menschliches Abschneiden	98.84
Sermanet, LeCun	Mehrstufige CNNs	98.31

Quelle: In Anlehnung an Institut für Neuroinformatik Ruhr-Universität Bochum, Resultate GTSRB, 2019, o.S.

- Richtzeichen (§ 42 StVO)

Neben diesen Verkehrszeichen gibt es noch Zusatzzeichen, welche zusammen mit Zeichen aus den genannten Gruppen verwendet werden. Diese sind in § 39 Abs. 7 StVO aufgelistet.

1.1.2 German Traffic Sign Recognition Benchmark

Für den praktischen Teil dieser Arbeit, wird ein Datensatz von Straßenverkehrsschildern benötigt, mit denen das CNN initial trainiert und bewertet werden kann. Es gibt eine Vielzahl von Datensätzen die hierfür potentiell geeignet wären. Das *Mapping and Assessing the State of Traffic InFrastructure (MASTIF)* Projekt stellt drei verschiedene Datensätze aus den Jahren 2009, 2010 und 2011 zur Verfügung. Der umfangreichste ist hierbei der Datensatz aus 2009 mit 6000 Bildern.³ Shakhuro und Konushin stellen einen Datensatz mit 104.358 verschiedenen russischen Schildern aus dem Jahre 2016 zur Verfügung.⁴ Der bisher umfangreichste Datensatz ist das *Mapillary Traffic Sign Dataset* aus dem Jahre 2020. Es enthält 100.000 Bilder mit insgesamt 325.172 erkannten Schildern. Von diesen Schildern fallen 82.724 Schilder in die im Datensatz gekennzeichneten Schilderklassen. Die Bilder stammen dabei aus der ganzen Welt, eine grobe Verteilung liegt bei 20% Nordamerika, 20% Europa, 20% Asien, 15% Südamerika, 15% Ozeanien und 10% Afrika.⁵

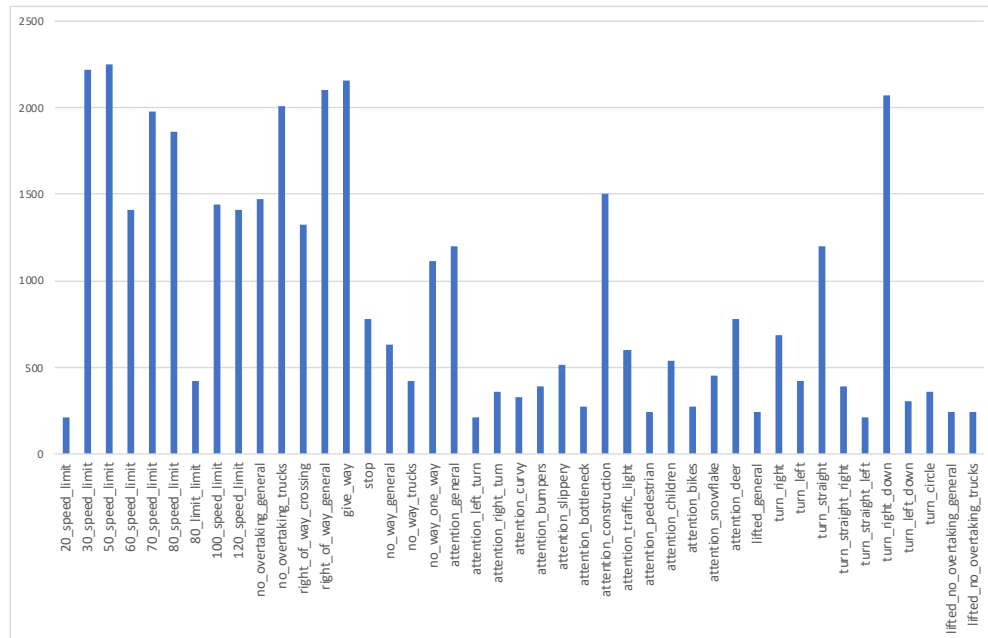
Letztendlich wurde sich für den Datensatz GTSRB entschieden. Dieser umfasst 51.840 Bilder von 1700 verschiedenen deutschen Straßenverkehrsschildern, welche in 43 Klassen unterteilt wurden. Grund für die Auswahl war primär, dass dieser Datensatz die größte Anzahl von deutschen Straßenverkehrsschildern bereitstellt, auf denen der Schwerpunkt dieser Arbeit liegt. Der Datensatz wurde im Jahre 2010 erstellt und die Maße der Bilder

³ Vgl. Šegvić, S. et al., 2010, S. 66-73.

⁴ Vgl. Shakhuro, V., Konushin, A., 2016, S. 294-300.

⁵ Vgl. Ertler, C. et al., 2020, S. 1-17.

Abbildung 1: Anzahl Bilder pro Klasse im GTSRB Trainingsdatensatz



Quelle: Eigene Darstellung, 2020

variieren zwischen 15x15 und 222x193 Pixeln. In Abbildung 1 wird die Anzahl der Bildern pro Klasse im Trainingsdatensatz des GTSRB dargestellt. Die 43 Klassen des Datensatzes fallen alle unter die drei Gruppen der Verkehrszeichen nach § 39 Abs. 2 Satz 2 StVO. Dabei fallen 27 Klassen unter Vorschriftzeichen, 14 Klassen unter Gefahrenzeichen und 2 Klassen unter Richtzeichen. Für den GTSRB liegen eine Vielzahl von Forschungsarbeiten vor.^{6,7,8,9,10} Die fünf Einreichungen mit der höchsten *Accuracy* (Genauigkeit) auf dem Testdatensatz sind in Tabelle 1 aufgelistet. Es ist zu erkennen, dass bereits eine sehr hohe *Accuracy* erreicht wird (99.71%).

Ggf ins
Glossar

1.2 Neural Networks

In den letzten Jahren gab es eine Vielzahl von Fortschritten durch Computersysteme, die häufig unter dem Begriff Künstliche Intelligenz (KI) zusammen gefasst werden. So konnte

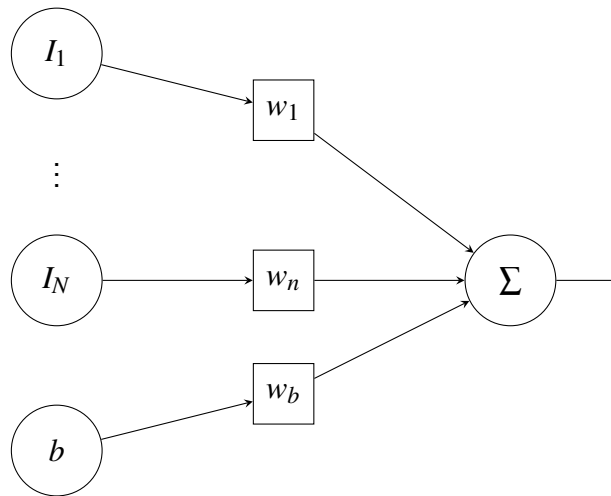
⁶ Vgl. Arcos-García, Á., Álvarez-García, J. A., Soria-Morillo, L. M., 2018, S. 158 - 165.

⁷ Vgl. Gecer, B., Azzopardi, G., Petkov, N., 2017, S. 165–174.

⁸ Vgl. Cireşan, D. et al., 2012, S. 333-33.

⁹ Vgl. Stallkamp, J. et al., 2012, S. 323-332.

¹⁰ Vgl. Sermanet, P., LeCun, Y., 2011, S. 2809-2813.

Abbildung 2: Darstellung eines Perceptron

Quelle: Eigene Darstellung, 2020

zum Beispiel AlphaGo einen der weltweit besten Spieler in dem Brettspiel Go schlagen und das System DeepFace ist in der Lage, menschliche Gesichter nahezu auf menschlichen Niveau erkennen zu können.^{11 12}

Systeme die als KI bezeichnet werden, sind häufig in der Lage, Aufgaben zu bewältigen, die bisher ausschließlich durch das menschliche Gehirn bearbeitet werden konnten. Neben den oben genannten Beispielen zählt hier auch das Erkennen von Bildern oder Sprache dazu. Damit Computer diese Aufgaben erledigen können, haben Wissenschaftler sich vom menschlichen bzw. biologischen Gehirn inspirieren lassen. Systeme, die auf dieser Grundlage basieren und einige der Vorgehensweisen des Gehirns imitieren, werden als NN bezeichnet.

1.2.1 Aufbau

Die grundlegenden Bausteine von NN sind Perceptron und wurden 1958 von Rosenblatt beschrieben.¹³ Als Grundlage verwendete er hierbei unter anderem die Arbeit von McCulloch und Pitts.¹⁴

Ein Perceptron kann zwischen 1 und n Eingangssignalen sowie einen negativen Bias Term¹⁵ b erhalten und hieraus ein Ausgangssignal - auch Aktivierung genannt - erzeugen.

¹¹ Vgl. Spiegel, 2016.

¹² Vgl. Taigman, Y. et al., 2014.

¹³ Vgl. Rosenblatt, F., 1958.

¹⁴ Vgl. McCulloch, W. S., Pitts, W., 1943.

¹⁵ Einige Definitionen schreiben nicht vor, dass der Bias negativ sein muss. Stattdessen wird dieser im weiteren Vorgehen subtrahiert statt summiert zu werden

gen. Für das Ausgangssignal werden alle Eingangssignale jeweils mit einem eigenen - als Gewicht bezeichneten - Faktor multipliziert (siehe Abbildung 2). Die Ergebnisse werden anschließend zusammen mit dem Bias summiert. Ist die Summe kleiner oder gleich 0, ist der Ausgangswert ebenfalls 0, ansonsten beträgt der Ausgangswert 1. Der Bias stellt also den Schwellenwert dar, der von der gewichtete Summe überschritten werden muss, damit das Ausgangssignal 1 lautet.

Es ist möglich, ein Perceptron um eine Aktivierungsfunktion zu ergänzen, sodass das Ausgangssignal reelle Werte annehmen kann. Diese Erweiterung des Perceptron wird als Neuron bezeichnet. Weit verbreitet ist die Verwendung der Sigmoid Funktion (siehe Gleichung 1). Bei der Anwendung einer Aktivierungsfunktion, häufig mit σ dargestellt, wird die gewichtete Summe z in die gewählte Funktion gegeben und das Ergebnis stellt das Ausgangssignal des Neuron dar (siehe Gleichung 2).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (1)$$

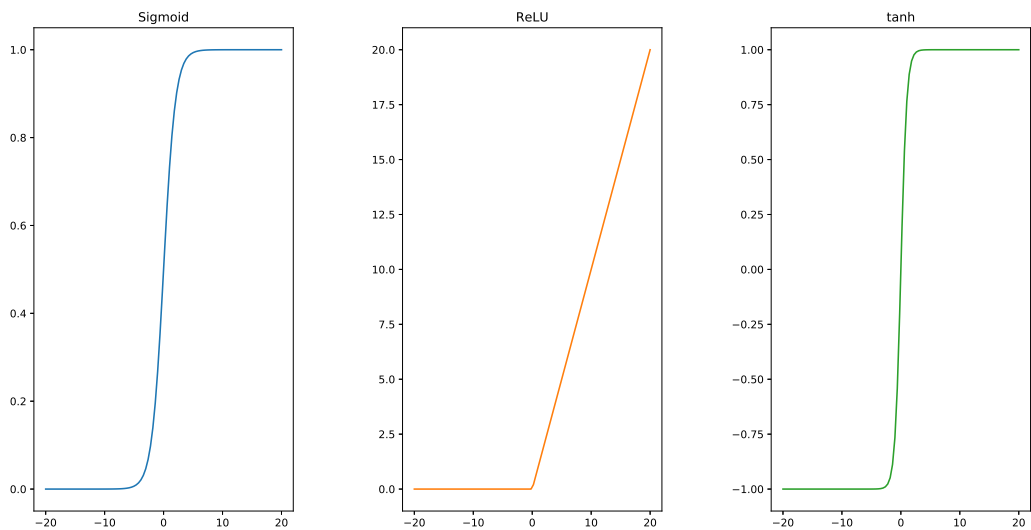
$$a = \sigma\left(\sum_i x_i w_i + b\right) \quad (2)$$

Neben der Sigmoid Funktion werden in modernen NN häufig die ReLu Funktion (Gleichung 3) oder die tanh Funktion (Gleichung 4) verwendet. Alle drei werden in Abbildung 3 gegenüber gestellt. Es ist zu erkennen, dass sowohl bei der Sigmoid Funktion, als auch bei der ReLU Funktion die untere Grenze des Wertebereichs bei 0 liegt. Bei der tanh Funktion, liegt diese bei -1 . Außerdem haben nur die Sigmoid- und tanh Funktion eine obere Grenze - beide 1 - definiert. ReLU kann jeden Wert größer gleich 0 annehmen. Verwendet man eine Aktivierungsfunktion, stellt der Bias kein Schwellwert mehr dar. Er ist als einziger Term in der Berechnung des Wertes der in die Aktivierungsfunktion gegeben wird von den Eingangswerten unabhängig und sorgt somit um eine konstante Verschiebung.

$$ReLU(z) = \max(0, z) \quad (3)$$

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1} \quad (4)$$

Ein NN besteht aus mehreren, verschalteten Neuronen. Diese werden in Layern angeordnet, welche typischerweise mit 1 bis N durchnummeriert werden. Layer 1 wird außerdem

Abbildung 3: Darstellung der Sigmoid-, ReLU- und tanh Funktion

Quelle: Eigene Darstellung, 2020

als Input Layer bezeichnet, da dieser die einzelnen Eingangswerte in das NN darstellt. Layer N wiederum wird als Output Layer bezeichnet, da er das Ergebnis des NN ausgibt. Alle Layer dazwischen ($1 < l < N$) werden als Hidden Layer bezeichnet, da von außen weder direkt auf die Eingangssignale der Neuronen in diesen Layern Einfluss genommen werden kann, noch die Ausgabesignale dieser direkt ersichtlich sind. Jeder Layer kann dabei eine unterschiedliche Anzahl von Neuronen enthalten. Typischerweise sind die Neuronen in einem Layer mit allen Neuronen des vorherigen Layers verbunden, ist dies der Fall, wird der Layer auch *Dense Layer* bezeichnet. Eine allgemeine Darstellung eines NN mit drei Layern ist in Abbildung 4 dargestellt.

1.2.2 Gradient Descent

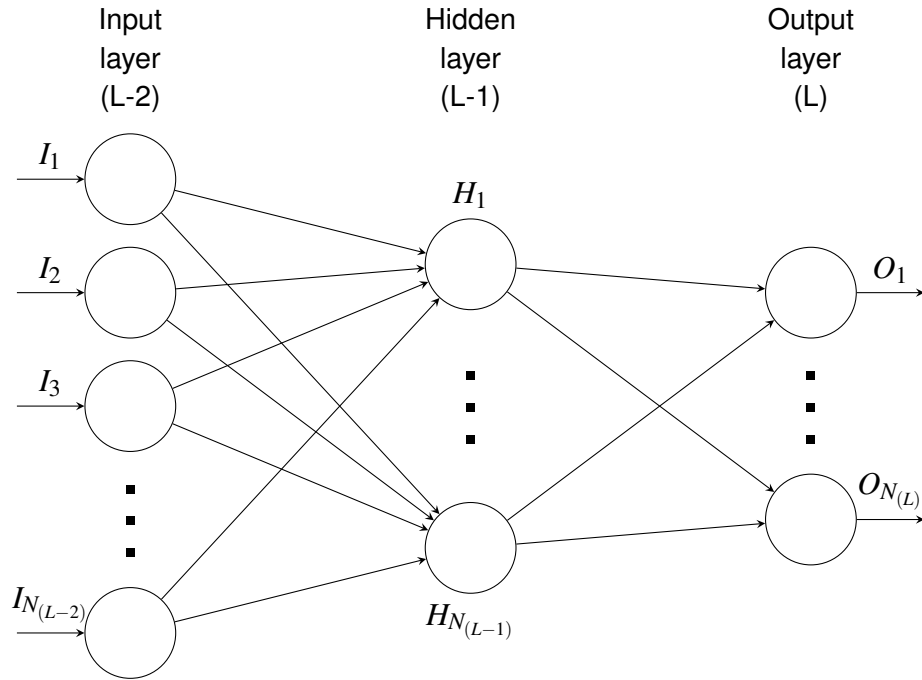
Im Rahmen des Lernprozesses eines NN werden die Gewichte und Bias der einzelnen Perceptron regelmäßig aktualisiert. Dabei liegt das Ziel darin, durch die Optimierung dieser Faktoren den Fehler des NN bei der Vorhersage zu minimieren.

Um diesen Fehler bestimmen zu können, wird eine Verlustfunktion angewendet. Ein einfaches Beispiel einer Verlustfunktion basiert auf der mittleren quadratischen Abweichung.

$$C(w, b) = \frac{1}{2} \sum (y - \hat{y})^2 \quad (5)$$

In Gleichung 5 werden die Gewichte mit w und die Bias mit b bezeichnet. Zusätzlich werden die wahren Klassen durch y und die Ausgabe des NN durch \hat{y} dargestellt.

Abbildung 4: Allgemein Darstellung eines Neural Network mit 3 Layern



Quelle: Eigene Darstellung, 2020

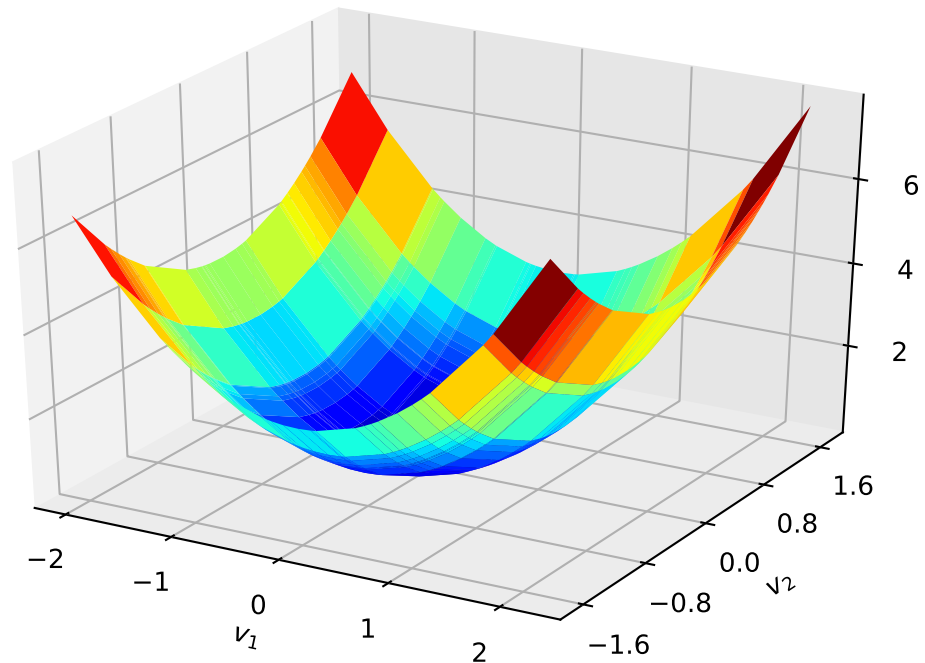
Um den Fehler des NN zu minimieren, muss der globale Tiefpunkt der Verlustfunktion bestimmt werden. Bei Funktionen mit einer Vielzahl von Parametern, ist es zu komplex, diese Aufgabe analytisch zu lösen. *Gradient Descent (GD)* stellt einen iterativen Prozess dar, sich dem nächsten Minimum zu nähern. Bildlich kann dieser Vorgang mit einem Wanderer, der vom Gipfel eines Berges absteigen möchte, verglichen werden. Geht dieser von seinem aktuellen Standpunkt ein paar Schritte in die entgegengesetzte Richtung der steilsten Stelle am Berg und wiederholt diesen Vorgang immer wieder, wandert er immer weiter ins Tal. Betrachtet man Abbildung 5, würde der Wanderer in einem der braunen Quadrate beginnen und sich Schritt für Schritt tiefer bis zum blauen Bereich vorarbeiten.

Mathematisch gesehen kann dies als eine Funktion $L(v)$ gesehen werden, wobei v eine beliebige Anzahl von Parametern darstellt. Für eine bessere Übersichtlichkeit, wird im folgendem mit zwei Parametern, v_1 und v_2 gerechnet.

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \quad (6)$$

In Gleichung 6 wird die Funktion ΔL als Summe der partiellen Ableitungen von v_1 und v_2 definiert. Die hierdurch erhaltenen Gradienten zeigen in die Richtung der größten Änderungen von v_1 beziehungsweise v_2 .

Abbildung 5: Beispiel quadratische Verlustfunktion mit den Parametern v_1 und v_2



Quelle: Eigene Darstellung, 2020

Fasst man die Parameter v_1 und v_2 und die partiellen Ableitungen jeweils in einem Vektor zusammen (Gleichung 7 und 8), kann Gleichung 6 zu Gleichung 9 umgeformt werden. Δv kann hierbei als der Vektor der Veränderung der Parameter gesehen werden.

$$\Delta v \equiv \begin{pmatrix} \Delta v_1 \\ \Delta v_2 \end{pmatrix} \quad (7)$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \quad (8)$$

$$\Delta C \approx \nabla C \cdot \Delta v \quad (9)$$

Um den Gesamtverlust zu reduzieren, muss der Vektor Δv in jedem Durchgang aktualisiert werden. Durch den Gradienten ist die Richtung der größten Änderung vom aktuellen Punkt der Funktion aus bekannt. Da der Gesamtverlust minimiert werden soll, muss in die entgegengesetzte Richtung des Gradienten gegangen werden. Dessen Wert wird also negiert. Außerdem muss eine positive Schrittweite bestimmt werden, die skaliert wie Weit in die Gegenrichtung des Gradienten gegangen wird. Diese wird als *Learning Rate* und häufig

mit α bezeichnet. Gleichung 10 zeigt, wie die Veränderung der Parameter, Δv , mittels α skaliert und das Ergebnis negiert wird.

$$\Delta v = -\alpha \nabla C \quad (10)$$

Wird Gleichung 10 in Gleichung 9 eingesetzt und umgeformt, erhält man Gleichung 11. Da ∇C^2 und α jeweils positiv sind, ist sichergestellt, dass die vorgenommene Veränderung negativ ist. Dies bedeutet, dass der Verlust mit jedem Durchgang verringert wird. Das NN lernt also und seine Ausgaben weichen weniger von den wahren Werten ab. Anzumerken ist hierbei noch, dass das Minimum bei einer zu groß gewählten *Learning Rate* überschritten werden kann. Dies würde dazu führen, dass man den Tiefpunkt überschreitet und wieder ein Stück hoch geht. Außerdem kann das erreichte Minimum ein lokales und nicht globales Minimum sein. Dies ist abhängig vom Startpunkt, da der negierte Gradient immer zum nächstgelegenen Minimum führt.

$$\Delta C \approx -\alpha \nabla C \cdot \nabla C = -\alpha \nabla C^2 \quad (11)$$

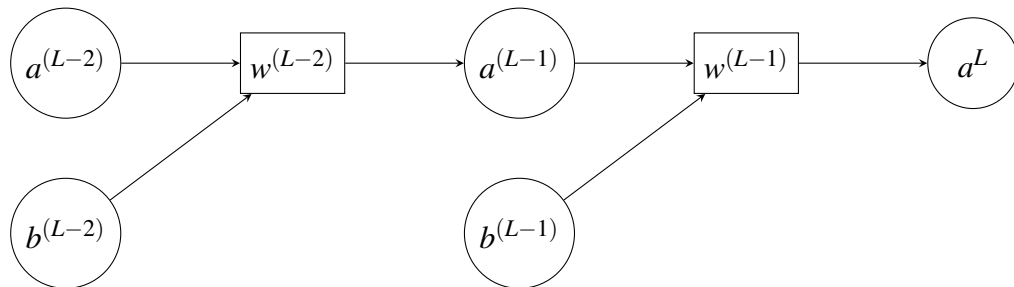
Ersetzt man v_1 und v_2 durch w und b , ist die Berechnung der neuen Werte für die Gewichte w_j und Bias b_j wie in den Gleichungen dargestellt 12 und 13 definiert.

$$w_j \rightarrow w'_j = w_j - \alpha \frac{\partial C}{\partial w_j} \quad (12)$$

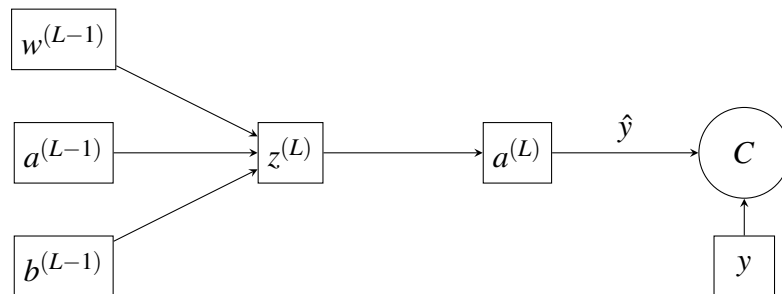
$$b_j \rightarrow b'_j = b_j - \alpha \frac{\partial C}{\partial b_j} \quad (13)$$

GD betrachtet alle Beispiele im Trainingsdatensatz, bevor die Gewichte und Bias das erste Mal angepasst werden. Da dies besonders bei großen Trainingsdatensätzen sehr zeit- und rechenintensiv ist, wird bei diesen Datensätzen häufig Stochastic Gradient Descent (SGD)¹⁶ angewendet. Bei SGD werden die Anpassungen bereits nach einem oder einer Gruppe von Beispielen aus dem Datensatz durchgeführt. Dies hat zwar den Nachteil, dass die Optimierung der Verlustfunktion schlechter sein kann als bei GD, allerdings konvergiert SGD meistens schneller.

¹⁶ Vgl. Robbins, H., Monro, S., 1951, S. 400 ff.

Abbildung 6: Einfaches Neural Network mit nur einem Neuron pro Layer

Quelle: Eigene Darstellung, 2020

Abbildung 7: Explizite Darstellung Einflüsse der Verlustfunktion des Output Layers

Quelle: Eigene Darstellung, 2020

1.2.3 Backpropagation

Wie im letzten Abschnitt beschrieben, müssen für die Anwendung von GD $\frac{\partial C}{\partial w}$ und $\frac{\partial C}{\partial b}$ berechnet werden. Rumelhart et al. haben für eine schnelle und effiziente Berechnung dieser im Jahre 1986 den Backpropagation Algorithmus vorgeschlagen.¹⁷ Dieser stellt eine Erweiterung der Delta-Regel auf NN mit mehr als zwei Schichten dar. Die Grundidee besteht darin, den Fehler des NN auf die einzelnen Gewichte und Bias innerhalb des NN zurückzuführen, so dass diese angepasst werden können. Formuliert man die Grundidee zum Beispiel für die Gewichte um, stellt sich die Frage, inwiefern sich der Fehler des NN ändert, wenn das Gewicht w_{jk} angepasst wird (siehe Gleichung 14).

Quelle
einfü-
gen

$$\frac{\partial C}{\partial w_{jk}} \quad (14)$$

Stellt man sich ein möglichst einfaches NN mit Hidden Layer vor, erhält man das in Abbildung 6 dargestellte NN. Dieses enthält pro Layer nur ein Neuron sowie einen Bias. Alle Gewichte eines Layers wurden für eine bessere Übersichtlichkeit als ein Element repräsentiert. Das heißt, $w^{(L-n)}$ stellt jeweils einen Vektor mit zwei Elementen dar. a^L bzw. $a^{(L-n)}$ bezeichnen die Aktivierung - also die finale Ausgabe - des Neurons.

¹⁷ Vgl. Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986, S. 533 ff.

$$z^{(L)} = w^{(L-1)}a^{(L-1)} + b^{(L-1)} \quad (15)$$

In Abbildung 7 sind die Einflüsse auf die Verlustfunktion für den Output Layer C dargestellt. Es ist zu erkennen, dass die Eingänge in die gewichtete Summe $z^{(L)}$ des Output Layers sich aus den Gewichten $w^{(L-1)}$, der Aktivierung des letzten Hidden Layers $a^{(L-1)}$ und dem Bias $b^{(L-1)}$ zusammensetzen. Die entsprechende Formel ist in Gleichung 15 dargestellt. Das Ergebnis von $z^{(L)}$ wird anschließend in die Aktivierungsfunktion σ gegeben, wodurch die Aktivierung des Output Layers $a^{(L)}$ berechnet wird. Diese stellt auch die Ausgabe des NN dar, welche als \hat{y} bezeichnet wird. Mit der Ausgabe des NN und dem tatsächlich erwarteten Wert y wird anschließend die Verlustfunktion berechnet. Soll also der Fehler des NN verkleinert werden, müssen die drei Eingangswerte $w^{(L-1)}$, $a^{(L-1)}$ und $b^{(L-1)}$ angepasst werden.

Wie in Gleichung 14 dargestellt, muss der Fehler des NN mittels partieller Ableitung auf jeden Einflussfaktor zurückgeführt werden. Hierbei wird die Kettenregel angewendet, die einzelnen Glieder kann man aus Abbildung 7 ablesen. Hierfür beginnt man bei dem Element dessen Anpassung vorgenommen werden soll und wandert den Graph weiter, bis man bei C angekommen ist. Die einzelnen Terme der Kettenregel bestehen dann aus der partiellen Ableitung des nächsten Elements im Pfad über das aktuelle Element. Geht man so - beginnend mit dem Element mit $w^{(L-1)}$ - vor, erhält man Gleichung 16.

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial z^{(L)}}{\partial w^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} \quad (16)$$

Berechnet man die entsprechenden Ableitungen, erhält man Gleichung 17.

$$\frac{\partial C}{\partial w^{(L-1)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y) \quad (17)$$

Der vorderste Teil der rechten Seite der Gleichung ist die partielle Ableitung von der gewichteten Summe $z^{(L)}$ über das Gewicht $w^{(L-1)}$. Der mittlere Teil der Gleichung stellt die Ableitung der gewählten Aktivierungsfunktion σ dar und der hinterste Teil ist die Ableitung der gewählten Verlustfunktion. In Gleichung 17 wurde hierfür Gleichung 5 verwendet.

Für die Berechnung des Bias $b^{(L-1)}$ kann der Pfad von C bis $z^{(L)}$ übernommen werden. Es ändert sich also nur etwas im vordersten Glied der Gleichungen 16 und 17. Außerdem ist der Bias eine Konstante, seine Ableitung (1) wird nicht dargestellt.

$$\frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial z^{(L)}}{\partial b^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = \sigma'(z^{(L)}) 2(a^{(L)} - y) \quad (18)$$

Die Ergebnisse für $\frac{\partial C}{\partial w^{(L-1)}}$ bzw. $\frac{\partial C}{\partial b^{(L-1)}}$ können in Gleichung 12 bzw. 13 eingesetzt werden, um das neue Gewicht und den neuen Bias bestimmen zu können. Die Aktivierung des Neuron im vorherigen Layer - $a^{(L-1)}$ - lässt sich nicht direkt anpassen. Sie hängt vom Gewicht, Bias und Aktivierung der vorherigen Schicht ab. Also muss hier der Fehler mit den selben Gleichungen berechnet werden. Hierfür wird zunächst der Einfluss von $a^{(L-1)}$ auf den Fehler C benötigt, da dieser der Fehler ist, der auf die vorherige Schicht verteilt wird. Auch hier ändert sich nur der vorderste Teil des Pfades, sowie dementsprechend das Ergebnis der Ableitung.

$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = w^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y) \quad (19)$$

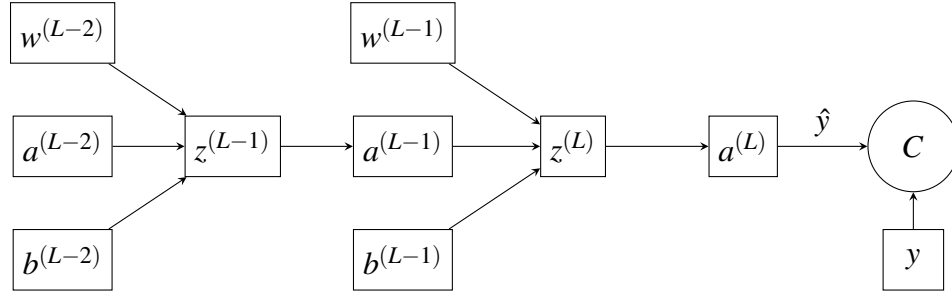
Nachdem nun alle Einflüsse auf den Output Layer berechnet sind, kann ein Schritt weiter zurück gegangen werden. Im NN aus Abbildung 6 ist dies der Input Layer $L - 2$. Die explizite Darstellung aller Einflüsse inklusive diesem Layer ist in Abbildung 8 dargestellt. Wie zu sehen ist, wurden nur die zwei Spalten ganz links zum Diagramm aus Abbildung 7 hinzugefügt. Dank der Kettenregel, spiegelt sich dies auch in den Gleichungen für die neuen Elemente $w^{(L-2)}$, $a^{(L-2)}$, $b^{(L-2)}$ und $z^{(L-1)}$ wieder. In Gleichung 20 wird die Berechnung für den Einfluss des Gewichts $w^{(L-2)}$ dargestellt.

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C}{\partial a^{(L-1)}} = a^{(L-2)} \sigma'(z^{(L-1)}) \frac{\partial C}{\partial a^{(L-1)}} \quad (20)$$

Der letzte Term des mittleren bzw. letzten Teils der Gleichung stellt dabei das Ergebnis aus Gleichung 19 dar. Wie zu erkennen ist, erhält man das Ergebnis des aktuellen Layers also dadurch, dass man die Berechnung für den nachfolgenden Layer um weitere Terme ergänzt. Dieses zurückführen des Fehlers ist der Grund, weshalb diese Methode Backpropagation genannt wird. Hätte das NN noch weitere Schichten, würde dieser Schritt des Erweiterns der Formel immer wiederholt. Da die Terme der Kettenregel separat berechnet werden können, muss also immer nur der neue Teil berechnet werden, wodurch der Algorithmus effizient durchgeführt werden kann.

Typischerweise haben NN mehr als ein Neuron pro Layer, daher müssen die oben genannten Formeln noch leicht angepasst werden. In Gleichungen 21 sind die Formeln für das Vorwärtsthrough des NN dargestellt. Da nun mehrere Neuronen im Output Layer vorhanden sind - N_L - muss über diese summiert werden. Anschließend muss für den quadratischen Mittelwert noch durch die Anzahl der Neuronen geteilt werden (siehe Formel 1 in Abbildung 21).

Abbildung 8: Explizite Darstellung Einflüsse der Verlustfunktion des gesamten Neural Networks



Quelle: Eigene Darstellung, 2020

Auch die gewichtete Summe in jedem Neuron hängt nun von allen Aktivierungen des vorherigen Layer ($N_{(L-1)}$) ab, so dass hier ebenfalls über diese summiert werden muss. Da es nun mehrere Neuronen pro Layer - und somit mehrere gewichtete Summen pro Layer - gibt, wird mittels dem Index j noch spezifiziert, welches Neuron gerade berechnet wird.

$$\begin{aligned}
 C &= \frac{1}{N_L} \sum_{j=1}^{N_L} \left(a_j^{(L)} - y_j \right)^2 \\
 z_j^{(L)} &= \sum_{k=1}^{N_{L-1}} w_{jk}^{(L-1)} a_k^{(L-1)} + b_j^{(L-1)} \\
 a_j^{(L)} &= \sigma \left(z_j^{(L)} \right)
 \end{aligned} \tag{21}$$

Wendet man die gleiche Vorgehensweise und Überlegung wie für das einfache NN an und nutzt zusätzlich die neue Notation, erhält man für den Einfluss des Gewichtes $w_{jk}^{(L-1)}$ Gleichung 22. Ähnlich wie beim einfachen NN unterscheidet sich die Formel für ein Bias nur durch die geänderten Formelzeichen, daher wird auf eine explizite Darstellung verzichtet.

$$\frac{\partial C}{\partial w_{jk}^{(L-1)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}} \tag{22}$$

Größere Änderungen gibt es für den Einfluss der Aktivierung $a_k^{(L-1)}$. Da jedes Neuron in diesem Hidden Layer mit jedem Neuron im Output Layer verbunden ist, hat es auch Einfluss auf den Fehler jedes Neurons im Output Layer. Möchte man also den gesamten Fehler eines Neurons im Hidden Layer berechnen, muss man die Summe über alle zurückgeführten Fehler aus dem Output Layer bilden (siehe Gleichungen 23).

$$\frac{\partial C}{\partial a_k^{(L-1)}} = \sum_{j=1}^{N_L} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}} \quad (23)$$

Neben dem Hinzufügen der neuen Notation mit Indizes und der Summe (Ich gehe mal davon aus das hier klar ist welche Summe gemeint ist), gibt es keine Änderungen zum Fall im einfachen NN. Dies lässt sich auch beobachten, wenn ein weiterer Schritt zurück gegangen wird. Also Elemente im Layer $L - 2$ betrachtet werden. Für das Gewicht $w_{jk}^{(L-2)}$ wird die Formel in Gleichung 24 dargestellt. Genau wie beim einfachen NN, wurde das Ergebnis des vorherigen Layers um zwei weitere Terme ergänzt. Ähnlich sieht es bei der Aktivierung aus, hier enthält der letzte Term in Summenzeichen aus Gleichung 23 immer den Weg vom aktuellen Layer bis zu C , so dass die zwei anderen Terme in der Summe nur auf den aktuellen Layer angepasst werden müssen.

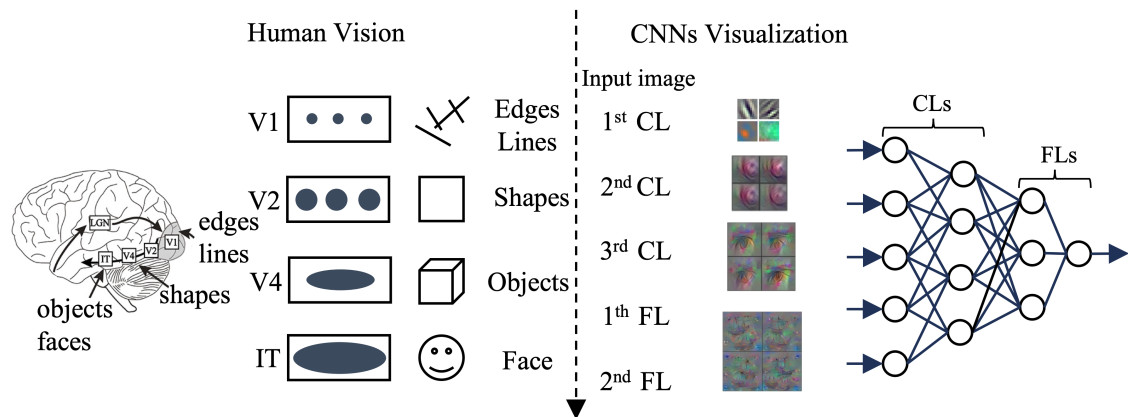
$$\frac{\partial C}{\partial w_{jk}^{(L-2)}} = \frac{\partial z_{jk}^{(L-1)}}{\partial w_{jk}^{(L-2)}} \frac{\partial a_j^{(L-1)}}{\partial z_j^{(L-1)}} \frac{\partial C}{\partial a_j^{(L-1)}} \quad (24)$$

Verwendet man die hergeleiteten Gleichungen und erweitert sie gegebenenfalls für weitere Hidden Layer, ist man in der Lage die Anpassungen aller Gewichte und Bias in einem NN zu berechnen. Der komplette Lernprozess besteht dann aus den folgenden vier Schritten:

1. Durchführen eines Vorwärtsdurchlaufs um die Ausgabe des Output Layers zu erhalten (\hat{y})
2. Ausrechnen der Verlustfunktion ($C(w, b)$), um den Fehler des NN zu erhalten
3. Berechnen der Gradienten von $C(w, b)$ im Bezug auf alle Gewichte und Bias im NN
4. Anpassen der Gewichte und Bias im Netzwerk anhand der berechneten Gradienten und den Ausgangswerten

1.3 Convolutional Neural Networks

Wie bereits dargestellt, sind alle Neuronen in einem NN miteinander verbunden. Dies führt dazu, dass die Zahl der Parameter innerhalb des NN schnell steigen kann. Dies ist zum Beispiel bei Bildern, bei denen jedes einzelne Pixel einen eigenen Eingabewert in das Netzwerk darstellt, der Fall. Da für jeden Parameter Berechnungen durchgeführt werden müssen, skalieren NN dementsprechend schlecht.

Abbildung 9: Vereinfachter Aufbau visueller Cortex und CNN

Quelle: Qin et al., 2018

Zusätzlich entstehen Formen und Objekte in Bildern erst durch den räumlichen Zusammenhang einiger Pixel. Daher erscheint das Vorgehen, alle Neuronen im Input Layer (Pixelwerte) mit allen Neuronen im ersten Hidden Layer zu verbinden, als wenig sinnvoll.

Aus diesen und weiteren Gründen, werden heutzutage CNN für die Klassifizierung von Bildern verwendet. Durch besondere Bausteine sorgen diese dafür, dass sowohl der räumliche Zusammenhang der Pixel betrachtet wird, als auch die Anzahl von Parametern besser skalierbar ist. Basierend auf den biologischen Vorbildern - siehe nächster Abschnitt - schlugen Fukushima und Miyake im Jahre 1972 *Neocognitron* vor, welches als ein Vorgänger von CNN gesehen werden kann.¹⁸ LeCun et. al verwendeten in ihrem Artikel *Handwritten digit recognition with a back-propagation network* von 1990 einem dem *Convolutional Layer* (siehe Abschnitt 1.3.2.1) ähnlichen Aufbau.¹⁹ Der Artikel *Gradient-based learning applied to document recognition* von LeCun et. al aus dem Jahre 1998 beschreibt erstmals den Aufbau eines heute üblichen CNN.²⁰

1.3.1 Biologische Grundlagen

CNN basieren auf Beobachtungen der Funktionsweise des visuellen Cortex bei Menschen und Tieren. So haben Hubel und Wiesel 1959 bzw. 1962 gezeigt, dass einzelne Sehzellen nur auf Veränderungen in einzelnen Bereichen der Netzhaut reagieren. Diese Bereiche wurden von Hubel und Wiesel als *receptive field* bezeichnet.^{21 22}

¹⁸ Vgl. Fukushima, K., Miyake, S., 1982, S. 267–285.

¹⁹ Vgl. LeCun, Y., Boser, B. E. et al., 1990, S. 396–404.

²⁰ Vgl. LeCun, Y., Bottou, L. et al., 1998, S. 2278–2324.

²¹ Vgl. Hubel, D. H., Wiesel, T. N., 1959, S. 574–591.

²² Vgl. Hubel, D. H., Wiesel, T. N., 1962, S. 106–154.

In Abbildung 9 wird der vereinfachte visuell Cortex eines Menschen dem Aufbau eines CNN gegenübergestellt. Auf der linken Seite ist zu erkennen, dass beim visuellen Cortex verschiedene Schichten (V1, V2, V4 und IT) vorliegen, welche nacheinander geschaltet sind. Jede dieser Schichten enthält *receptive fields*, wobei diese mit jeder Schicht größer werden. Die früheren Schichten reagieren hierbei auf einfache Formen wie Linien oder Kanten und die späteren auf komplexere Formen wie ganze Objekte. Bei allen Schichten können die gesamten Elemente an jeder Stelle des *receptive field* auftreten und trotzdem für eine Aktivierung sorgen. Die rechte Seite zeigt den typischen Aufbau eines CNN, bei dem die früheren Schichten (CL) einfache und spätere Schichten (FL) komplexe Formen erkennen.²³

1.3.2 Architektur

CNN haben drei grundlegende Ideen bzw. Bausteine, die sie von anderen NN unterscheiden. Diese sind *Local receptive Fields (LRF)*, *gemeinsame Gewichte und Bias* sowie *Pooling* und werden in diesem Abschnitt genauer beleuchtet.

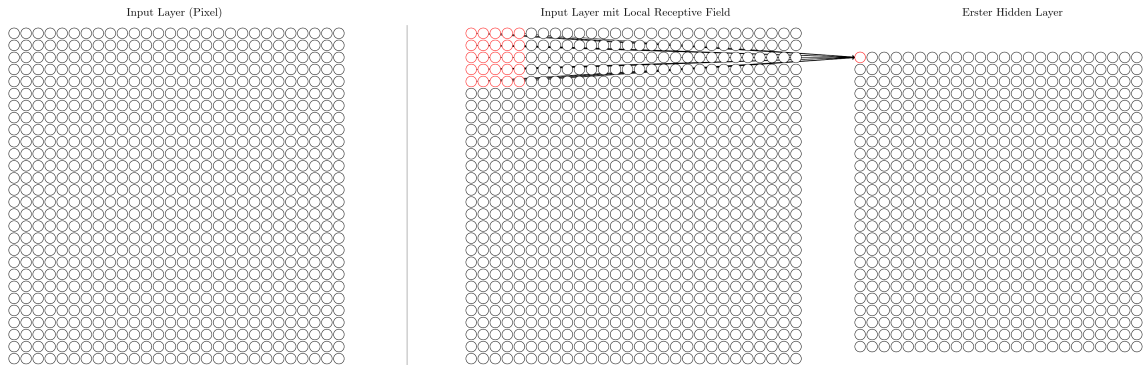
1.3.2.1 Local Receptive Fields

Die Grundidee hinter LRF ist - ähnlich wie beim visuellen Cortex - Neuronen zu erzeugen, die in bestimmte Bereiche auf Formen reagieren. Anders gesagt, sollen die einzelnen Neuronen einen Teil des Bildes betrachten und hier Formen erkennen können. Stellt man sich das Eingangssignal als Matrix statt einem Vektor vor, lässt sich das Ganze einfacher visualisieren. Ein Bild mit 28x28 Pixel entspricht also einer 28x28 Matrix, siehe Abbildung 10 links. Um nun die LRF nachzubilden, wird eine kleinere Matrix - typischerweise als *Filter* oder *Kernel* bezeichnet - über das Bild geschoben. Typische Maße für den Kernel sind hierbei 3x3, 4x4 oder 5x5. Alle Pixel die vom Kernel überdeckt werden, werden in der Berechnung für den Pixelwert in der Ausgabe zusammengefasst (siehe Abbildung 10 rechts). Bei einer Matrix mit den Maßen 5x5 bedeutet dies, dass ein Pixel in der Ausgabe 25 Werte zusammenfasst. Die Werte innerhalb der Kernel-Matrix stellen dabei die Gewichte für jeden einzelnen Pixel dar. Zusätzlich hat jeder Pixel in der Ausgabe auch noch einen eigenen Bias.(Erklärst du irgendwo was ein "Bias" ist?)

Wandert der Kernel über das Bild, kann er um n Schritte nach rechts bzw. um m Schritte nach unten verschoben werden. Hierdurch wird bestimmt, welche Pixel in der Ausgabe zusammengefasst werden. Diese Schrittweite wird als *Stride* bezeichnet. Betrachtet man

²³ Vgl. Qin, Z. et al., 2018, S. 6.

Abbildung 10: Input Layer als Matrix (links) und Beispiel LRF zwischen Input und 1. Hidden Layer



Quelle: In Anlehnung an Nielsen, Introduction CNN, 2018, S. 170 f.

ein Bild mit 28x28 Pixeln, einen 5x5 Kernel und einen Stride von 1, besteht die Ausgabe aus 24x24 Pixeln. Dies liegt daran, dass man den Kernel von der Ausgangsposition nur 23 Mal um eins nach rechts bzw. unten schieben kann. Um zu verhindern, dass die Matrix im nachfolgenden Layer kleiner wird, kann *Padding* angewendet werden. Hierbei werden um das Bild herum weitere Pixel ergänzt. Typischerweise geschieht dies symmetrisch, allerdings ist es auch möglich unterschiedlich viele Reihen auf jeder Seite hinzuzufügen. Der Wert der hinzugefügten Pixel wird typischerweise auf 0 gesetzt.

Die Anzahl der Spalten in der Ausgabe kann mit Gleichung 25 berechnet werden.²⁴ Hierbei steht i für die Anzahl der Spalten in der Matrix die das ursprüngliche Bild darstellt, p_{links} für die Anzahl von Spalten die durch das Padding links hinzugefügt wurden (p_{rechts} dementsprechend für die rechts), k für die Maße des Kernels und s für den Stride. Sind das Bild und der Kernel quadratisch und durch Padding wurde an allen Seiten gleich viele Spalten hinzugefügt, entspricht das Ergebnis auch der Anzahl an Zeilen in der Ausgabe.²⁵

$$o = \left\lfloor \frac{i + p_{links} + p_{rechts} - k}{s} \right\rfloor + 1 \quad (25)$$

1.3.2.2 Gemeinsame Gewichte und Bias

Wie bereits erwähnt, stellen die Werte der Kernel-Matrix die Gewichte für jeden einzelnen Pixel im Bild dar. Allerdings werden außerdem die gleichen Werte innerhalb des Kernels für alle Verschiebungen verwendet. Bei einem 28x28 Layer und einer 5x5 Matrix, werden

²⁴ Vgl. Dumoulin, V., Visin, F., 2018, S. 15.

²⁵ Vgl. Nielsen, M., 2015, S. 169-171.

also alle 24x24 Pixel im Ausgang mit den gleichen Gewichten und Bias berechnet (siehe Gleichung 26).

$$pixel_{j,k} = \sigma \left(b + \sum_{l=0}^z \sum_{m=0}^s w_{l,m} a_{j+l,k+m} \right) \quad (26)$$

Hierbei steht (z) für die Anzahl der Zeilen und s für die Anzahl von Spalten der Matrix des Bildes. b ist der gemeinsame Bias, w das Gewicht an Position l, m der Matrix und a der Wert innerhalb des Kernels. Typischerweise wird die berechnete Summe noch in eine Aktivierungsfunktion gegeben, welche durch σ dargestellt wird.

Das bedeutet, dass durch den Kernel überall auf dem Bild die gleichen Formen beziehungsweise Objekte erkannt werden. Man stelle sich zum Beispiel vor, dass der Kernel durch das Anpassen seiner Gewichte gelernt hat vertikale Linien zu erkennen. Dann ist es sinnvoll, dass diese vertikale Linie überall auf dem Bild erkannt werden kann. Genau dies wird durch die gemeinsamen Gewichte und Bias erreicht.

Das Element, auf welches ein Kernel reagiert, wird als *Feature* bezeichnet. Daher werden die Matrizen in der Ausgabe auch *Feature Maps* genannt. Insgesamt wird ein Layer in einem CNN, welcher Feature Maps mit gemeinsamen Gewichten anwendet, als *Convolutional Layer* bezeichnet. Ein CNN soll mehr als ein Feature pro Convolutional Layer erkennen können. Daher können pro Convolutional Layer n verschiedene Kernels verwendet werden. Mit jedem Kernel entsteht eine neue Matrix in der Ausgabe. Die Ausgabe stellt somit eine dreidimensionale Matrix dar, bildlich gesehen kann man sich einen Stapel von zweidimensionalen Matrizen vorstellen (siehe Convolution Layer in 12). Grundsätzlich reagieren Kernels in Convolutional Layern die früh im CNN vorkommen eher auf einfache Formen wie Kanten oder Ecken. Später reagieren Convolutional Layer dann auf komplexere Formen bzw. Objekte. Auf der rechten Seite in Abbildung 9 ist dies dargestellt. Während im ersten CL Layer einfache Linien erkannt werden, können spätere CL Layer Objekte wie Augen erkennen.²⁶

1.3.2.3 Pooling

Neben dem beschriebenen Convolutional Layer, haben CNN einen weiteren besonderen Layer. Dieser wird als *Pooling Layer* bezeichnet und typischerweise direkt hinter einem Convolutional Layer angewendet.

²⁶ Vgl. Nielsen, M., 2015, S. 169-171.

Ein Pooling Layer vereinfacht hierbei die vorliegenden Informationen, also die Matrizen die der vorherige Layer ausgegeben hat. Folgt der Pooling Layer auf einen Convolutional, sind dies die Ausgaben der Feature Maps, also die Werte, die durch die Berechnung der Aktivierungsfunktion erhalten werden.

Beim Pooling werden mehrere Werte zusammengefasst, es wird also erneut eine Art Filter (Quadrat) über die Matrix gelegt und alle überdeckten Werte werden zusammen betrachtet. Am verbreitetsten sind hierbei das *Max Pooling* und das *Average Pooling*. Wie durch die Namen zu erkennen, wird beim Max Pooling der maximale Wert innerhalb der betrachteten Werte verwendet, während beim Average Pooling der Durchschnitt berechnet wird. Beides ist in Abbildung 11 beispielhaft dargestellt. Die Maße der Matrix verändern sich entsprechend der gewählten Filter Matrix, bei einer 2x2 Matrix wird die Anzahl der Spalten und Zeilen halbiert. Das Pooling wird hierbei auf jede Feature Map separat angewendet, die Anzahl der Matrizen ändert sich also nicht.

Durch Pooling wird geprüft, ob eine Form oder ein Objekt durch den vorherigen Convolutional Layer erkannt wurde. Die Grundannahme geht hierbei davon aus, dass die Information das dieses Element vorhanden ist, aber nicht die genaue Position relevant ist. Durch das Pooling bleibt diese Information erhalten, nur die Genauigkeit der Positionsangabe sinkt. Hierfür wird wiederum die Anzahl der Parameter stark reduziert.

In Abbildung 12 ist der Beginn eines CNN dargestellt. Hier wird ein Convolutional Layer als erster Hidden Layer verwendet und dessen Ausgabe (Feature Maps) werden als Input für den Pooling Layer (zweiter Hidden Layer) verwendet. Dessen Ausgabe wird in einen Dense Layer gegeben, also einem Layer der mit allen Neuronen des vorherigen Layers verbunden ist.²⁷

²⁷ Vgl. Nielsen, M., 2015, S. 169-171.

Abbildung 11: Beispiel Max Pooling und Average Pooling

163	131	0	42
248	247	161	89
13	120	62	8
40	19	23	168

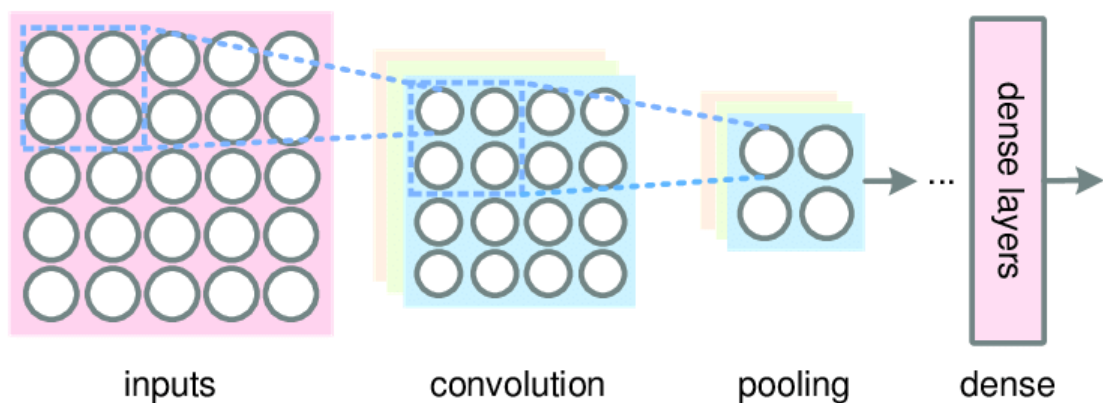
Max Pooling:

$$\max(163, 131, 248, 247) = 248$$

Average Pooling:

$$\frac{163+131+248+247}{4} = 197$$

Quelle: Eigene Darstellung, 2020

Abbildung 12: Convolutional und Pooling Layer im Zusammenspiel

Quelle: Wang et. al, Wireless Layer, 2017, S. 4

1.4 Visualisierung der Entscheidung

1.4.1 Activation Maximization

Activation Maximization (AM) stellt eine Methode dar, um ein Bild x^* als Eingabe für das CNN zu erzeugen, welches die Ausgabe eines Neurons maximiert (siehe Gleichung 27, wobei θ für die Parameter - Gewichte und Bias - des CNN steht). Dabei kann AM auf Neuronen aus jedem Layer eines CNN angewendet werden. Weiter verbreitet ist die Anwendung auf den letzten Convolutional Layer, in welcher jedes Neuron für eine Klasse steht. In diesem Fall erzeugt AM also ein Bild, welches die Ausgabe des Neurons für die Klasse k maximiert. Anhand dieses Bildes können Rückschlüsse gezogen werden, was das CNN gelernt hat um die Klasse k zu erkennen.

$$x^* = \underset{x}{\operatorname{argmax}} a_{i,l}(\theta, x) \quad (27)$$

Um das Bild zu generieren, wird ein iterativer Prozess verwendet. In diesem werden die Werte jedes Pixels im Bild in jedem Durchgang so angepasst, dass die Ausgabe des Zielneurons weiter maximiert wird. Der Prozess kann dabei in drei Schritte unterteilt werden:

1. Es wird ein Bild $x = x_0$ mit zufälligen Werten für alle Pixel generiert (Rauschen). Dieses Bild wird dann in das CNN gegeben, um einen vorwärts Durchlauf durchzuführen.
2. Mithilfe von Backpropagation wird die partielle Ableitung des Zielneurons $a_{i,l}$ im Bezug auf x durchgeführt ($\frac{\partial a_{i,l}}{\partial x}$). Hierbei bleiben die Gewichte und Bias im CNN unverändert.
3. Mithilfe des berechneten Gradienten werden alle Pixeln in x angepasst, so dass die Aktivierung des Zielneurons weiter erhöht wird. Dabei wird mittels einer Schrittweite (auch Learning Rate, η) gesteuert, wie stark die Anpassungen sind (Gleichung 28).

$$x \leftarrow x + \eta \cdot \frac{\partial a_{i,l}(\theta, x)}{\partial x} \quad (28)$$

Der Prozess wird beendet, wenn im erzeugten Bild kein Rauschen mehr vorhanden ist. Da dies ein Zustand ist, der bei CNN mit mehreren Layern selten erreicht wird, kann außerdem ein Schwellwert an Rauschen definiert werden, ab welchem der Prozess abbricht.

Ein weiteres Problem von komplexen CNN mit vielen Layern ist, dass die Muster in den späteren Layern für den Menschen immer weniger interpretierbar werden. Um diese Interpretierbarkeit zu erhöhen, können verschiedene Methoden zur regularisierung angewendet werden. Hierbei wird der Gleichung ein Term $-\lambda(x)$ hinzugefügt, der als Bias beim Erstellen des Bildes x^* wirkt (siehe Gleichung 29).

$$x^* = \underset{x}{\operatorname{argmax}} (a_{i,l}(\theta, x) - \lambda(x)) \quad (29)$$

Für $\lambda(x)$ können verschiedene Methoden verwendet werden, bekannte Implementationen sind zum Beispiel *l_2 decay*, *Gaussian blur*, oder *mean image initialization*.^{28,29,30} Jede dieser Methoden hat seine eigenen Effekte, *l_2 decay* verhindert zum Beispiel, dass einige

²⁸ Vgl. Simonyan, K., Vedaldi, A., Zisserman, A., 2014, S. 1–8.

²⁹ Vgl. Yosinski, J. et al., 2015, S. 1–12.

³⁰ Vgl. Wei, D. et al., 2015, S. 1–7.

extreme Pixelwerte das erzeugte Bild dominieren. Die verschiedenen Methoden können auch kombiniert werden, um die verschiedenen Effekte nutzen zu können.³¹

1.4.2 Saliency Map

Während AM den Fokus auf die Visualisierung der Aktivierung einzelner Neuronen legt, betrachtet Saliency Map (SM) das gesamte CNN. Die Grundidee hinter SM ist es, die Pixel zu ermitteln, welche den größten Einfluss auf die Zuordnung in eine bestimmte Klasse haben.

Nimmt man ein Bild I_0 , eine Klasse c und ein trainiertes CNN mit der Bewertungsfunktion $S_c(I)$, ist das Ziel alle Pixel nach ihrem Einfluss auf den Wert $S_c(I_0)$ zu sortieren. Im einfachen Fall einer linearen Bewertungsfunktion (siehe Gleichung 30, mit I als eindimensionaler Vektor des Bildes und w_c bzw. b_c als Parametervektoren des CNN) gibt bereits die Größe der Gewichte w an, wie wichtig die dazugehörigen Pixel für die Zuordnung in Klasse c sind.

$$S_c(I) = w_c^T I + b_c \quad (30)$$

Die Bewertungsfunktion innerhalb eines CNN ist allerdings eine nicht lineare Funktion, so dass dieses Prinzip nicht direkt übernommen werden kann. Geht man allerdings von dem Bild I_0 aus, kann sich dem Wert von $S_c(I)$ mit einer linearen Funktion genähert werden. Hierfür wird eine Taylorreihe erster Ordnung berechnet (siehe Gleichung 31, mit w gleich $\left. \frac{\partial S_c}{\partial I} \right|_{I_0}$).

$$S_c(I) \approx w^T I + b \quad (31)$$

Eine mögliche Interpretation von $\left. \frac{\partial S_c}{\partial I} \right|_{I_0}$ ist, dass die Größe der Ableitung anzeigt, welche Pixel am wenigsten verändert werden müssen, um das Ergebnis der Bewertungsfunktion am stärksten zu verändern. Es wird davon ausgegangen, dass diese Pixel mit der Position des Objekt der bewerteten Klasse im Bild korrespondieren.

Möchte man SM auf ein Farbbild (z.B. im Rot, Grün, Blau (RGB) Farbraum, wobei jede Farbe ein eigener Layer im Eingabebild ist) anwenden, müssen zunächst die Ableitungen $w \left(\left. \frac{\partial S_c}{\partial I} \right|_{I_0} \right)$ mittels Backpropagation berechnet werden. Anschließend wird der maximale Wert der drei Layer als Wert für den Pixel in der SM Matrix M verwendet (siehe Gleichung 32, mit $w_{h(i,j,c)}$ als Index eines Pixel an der Position i, j im Layer c).³²

³¹ Vgl. Erhan, D. et al., 2009, S. 1-14.

³² Vgl. Simonyan, K., Vedaldi, A., Zisserman, A., 2014, S. 3f.

$$M_{ij} = \max_c |w_{h(i,j,c)}| \quad (32)$$

1.4.3 Class Activation Mapping

Class Activation Mapping (CAM) wurde 2016 von Zhou et al. vorgestellt und wird auf bereits trainierte CNN angewendet. Das Ziel von CAM ist es, innerhalb eines Bildes die relevanten Bereiche für die Zuordnung zu einer Klasse zu markieren. Damit CAM verwendet werden kann, muss das CNN einen bestimmten Aufbau haben. Nach dem letzten Convolutional Layer muss es einen *Global Average Pooling Layer* vor dem *fully-connected Layer* geben.

Der letzte Convolutional Layer erzeugt dabei k Feature Maps, für jeden Kernel eine. Damit hat die Ausgabe des Convolutional Layers drei Dimensionen. Die Höhe v , die Breite u und die Anzahl von Feature Maps k . Pro Feature Map wird Global Average Pooling angewendet, das heißt es wird der Mittelwert über alle Pixel in einer Feature Maps A^k gebildet (siehe Gleichung 33).

$$avgpool_k = \frac{1}{z} \sum_{i=1}^u \sum_{j=1}^v A_{ij}^k \quad (33)$$

Nachdem Global Average Pooling auf alle Feature Maps angewendet wurde, bleibt ein Vektor mit k Nummer über. Diese werden in einen fully-connected Layer gegeben. Die Gewichte w zwischen dem Ergebnis des Global Average Pooling und dem fully-connected Layer, werden dabei während des Trainingsprozesses des Netzwerkes gelernt.

Zur Erzeugung der Ergebnis Matrix - bei visueller Darstellung auch Heatmap genannt - von CAM, werden die gelernten Gewichte w mit den Feature Maps aus der Aktivierung des letzten Convolutional Layers A_k multipliziert. Zu beachten ist hierbei, dass CAM die Aktivierung für das aktuelle Bild und auf eine angegebene Klasse darstellt. Gleichung 34 zeigt die Formel zur Berechnung von CAM für die Klasse c .

$$cam_c = \sum_k w_k^c A^k \quad (34)$$

Das Ergebnis von CAM ist somit eine Matrix mit den Maßen der Feature Maps des letzten Convolutional Layer. Häufig wird dieses hochskaliert, so dass die Heatmap über das Originalbild gelegt werden kann.³³

³³ Vgl. Zhou, B. et al., 2016, S. 2-4.

Um die Beschränkung der Architektur des CNN durch CAM aufzuheben, haben Selvaraju et al. 2016 mit Grad-CAM die Verallgemeinerung von CAM vorgeschlagen.

Bei dieser Verallgemeinerung benötigt das CNN keinen Global Average Pooling Layer nach dem letzten Convolutional Layer mehr. Es kann eine beliebige Anzahl von Layern nach dem Convolutional Layer vorhanden sein, die einzige Bedingung für diese ist, dass sie differenzierbar sind. Laut den Autoren von Grad-CAM wird dabei der letzte Convolutional Layer als Grundlage der Visualisierung verwendet, da zu erwarten ist, dass dieser das beste Gleichgewicht zwischen Interpretierbarkeit und räumlicher Auflösung bietet.

Wie bei CAM, werden alle k Feature Maps gewichtet addiert um die Ergebnismatrix zu erhalten. Der Unterschied liegt in der Berechnung der Gewichte. Während CAM die Gewichte w des CNN verwendet, berechnet Grad-CAM eigene Gewichte α , welche auf den Gradienten basieren.

Als Grundlage für die α Werte verwendet Grad-CAM die Ausgabewerte des CNN, bevor diese in den Softmax Layer gegeben werden (y^c). Der eigentliche Prozess von Grad-CAM kann in drei Schritte unterteilt werden. Zunächst wird der Gradient von y^c mit Bezug auf die Feature Maps A^k berechnet ($\frac{\partial y^c}{\partial A^k}$). Dieser ist abhängig von dem aktuellen Eingabebild, da dieses die Grundlage für die Feature Maps und y^c liefert. Wie bei CAM ergibt ein zweidimensionales Eingabebild ein dreidimensionalen Gradienten (v, u, k). Im zweiten Schritt werden die α Werte berechnet. Jede Feature Map k erhält dabei einen Wert für die gewählte Klasse c (α_k^c). Für diesen Wert wird über die Höhe und Breite der Feature Map gelaufen um Global Average Pooling durchzuführen (siehe Gleichung 35). Dies bedeutet, dass am Ende dieses Schrittes eine Matrix mit der Form $[1,1,k]$ oder vereinfacht der Vektor $[k]$ vorliegt. Jede Feature Map hat somit einen α Wert, der als Gewicht verwendet werden kann.

$$\alpha_k^c = \frac{1}{Z} \sum_{i=0}^u \sum_{j=0}^v \frac{\partial y^c}{\partial A_{ij}^k} \quad (35)$$

Im dritten Schritt wird nun die Ergebnismatrix von Grad-CAM berechnet. Diese ist die lineare Kombination der gewichteten Feature Maps, wobei das Ergebnis in die ReLU Funktion gegeben wird (siehe Gleichung 36). Die ReLU Funktion sorgt dabei dafür, dass nur positive Werte einbezogen werden, da alle negativen Werte auf 0 gesetzt werden.

$$L_{\text{Grad-CAM}}^c = \text{ReLU} \left(\sum_k \alpha_k^c A^k \right) \quad (36)$$

Wie bei CAM hat die Ergebnismatrix/Heatmap von Grad-CAM die Form $u \times x$, sie muss also hoch skaliert werden, damit sie über das Eingangsbild gelegt werden kann.³⁴

1.5 Bibliotheken

³⁴ Vgl. *Selvaraju, R. R.* et al., 2020, S. 4-6.

Anhang

Anhang 1: Beispielanhang

Dieser Abschnitt dient nur dazu zu demonstrieren, wie ein Anhang aufgebaut sein kann.







Anhang 1.1: Weitere Gliederungsebene

Auch eine zweite Gliederungsebene ist möglich.

Anhang 2: Bilder

Auch mit Bildern. Diese tauchen nicht im Abbildungsverzeichnis auf.

Abbildung 13: Beispielbild

Name	Änderungsdatum	Typ	Größe
 abbildungen	29.08.2013 01:25	Dateiordner	
 kapitel	29.08.2013 00:55	Dateiordner	
 literatur	31.08.2013 18:17	Dateiordner	
 skripte	01.09.2013 00:10	Dateiordner	
 compile.bat	31.08.2013 20:11	Windows-Batchda...	1 KB
 thesis_main.tex	01.09.2013 00:25	LaTeX Document	5 KB

Literaturverzeichnis

- Arcos-García, Álvaro, Álvarez-García, Juan A., Soria-Morillo, Luis M.* (2018): Deep Neural Network for Traffic Sign Recognition Systems: An Analysis of Spatial Transformers and Stochastic Optimisation Methods, in: *Neural Networks*, 99 (2018), S. 158–165
- Cireşan, Dan, Meier, Ueli, Masci, Jonathan, Schmidhuber, Jürgen* (2012): Multi-Column Deep Neural Network for Traffic Sign Classification, in: *Neural Networks*, 32 (2012), S. 333–338, [Zugriff: 2020-06-04]
- Erhan, Dumitru, Bengio, Y., Courville, Aaron, Vincent, Pascal* (2009): Visualizing Higher-Layer Features of a Deep Network, in: *Technical Report*, Univeristé de Montréal (2009), Nr. 1341, S. 4–6, [Zugriff: 2020-10-22]
- Fukushima, Kunihiko, Miyake, Sei* (1982): Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition, in: *Competition and Cooperation in Neural Nets*, o. O.: Springer, 1982, S. 267–285
- Gecer, Baris, Azzopardi, George, Petkov, Nicolai* (2017): Color-Blob-Based COSFIRE Filters for Object Recognition, in: *Image and Vision Computing*, 57 (2017), S. 165–174, [Zugriff: 2020-10-19]
- Hubel, D. H., Wiesel, T. N.* (1959): Receptive Fields of Single Neurones in the Cat's Striate Cortex, in: *The Journal of Physiology*, 148 (1959), Nr. 3, S. 574–591, [Zugriff: 2020-10-17]
- Hubel, David H, Wiesel, Torsten N* (1962): Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex, in: *The Journal of physiology*, 160 (1962), Nr. 1, S. 106–154
- LeCun, Yann, Boser, Bernhard E, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne E, Jackel, Lawrence D* (1990): Handwritten Digit Recognition with a Back-Propagation Network, in: *Advances in Neural Information Processing Systems*, o. O., 1990, S. 396–404
- LeCun, Yann, Bottou, Léon, Bengio, Yoshua, Haffner, Patrick* (1998): Gradient-Based Learning Applied to Document Recognition, in: *Proceedings of the IEEE*, 86 (1998), Nr. 11, S. 2278–2324
- McCulloch, Warren S., Pitts, Walter* (1943): A Logical Calculus of the Ideas Immanent in Nervous Activity, in: *Bulletin of Mathematical Biophysics*, 5 (1943), Nr. 4, S. 115–133, [Zugriff: 2020-10-11]

- Nielsen, Michael* (2015): *Neural networks and deep learning*, San Francisco, CA: Determination press San Francisco, CA, 2015, 169–178, [Zugriff: 2020-09-26]
- Qin, Zhuwei, Yu, Fuxun, Liu, Chenchen, Chen, Xiang, ,George Mason University, 4400 University Dr, Fairfax, VA 22030, USA, ,Clarkson University, 8 Clarkson Ave, Potsdam, NY 13699, USA* (2018): How Convolutional Neural Networks See the World — A Survey of Convolutional Neural Network Visualization Methods, in: *Mathematical Foundations of Computing*, 1 (2018), Nr. 2, S. 149–180, [Zugriff: 2020-10-17]
- Robbins, Herbert, Monro, Sutton* (1951): A Stochastic Approximation Method, in: *The annals of mathematical statistics* (1951), S. 400–407
- Rosenblatt, F.* (1958): The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. In: *Psychological Review*, 65 (1958), Nr. 6, S. 386–408, [Zugriff: 2020-10-11]
- Rumelhart, David E, Hinton, Geoffrey E, Williams, Ronald J* (1986): Learning Representations by Back-Propagating Errors, in: *nature*, 323 (1986), Nr. 6088, S. 533–536
- Šegvić, Siniša, Brkić, Karla, Kalafatić, Zoran, Stanisavljević, Vladimir, Ševrović, Marko, Budimir, Damir, Dadić, Ivan* (2010): A Computer Vision Assisted Geoinformation Inventory for Traffic Infrastructure, in: *13th International IEEE Conference on Intelligent Transportation Systems*, IEEE, o. O., 2010, S. 66–73
- Selvaraju, Ramprasaath R., Cogswell, Michael, Das, Abhishek, Vedantam, Ramakrishna, Parikh, Devi, Batra, Dhruv* (2020): Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization, in: *Int J Comput Vis*, 128 (2020), Nr. 2, S. 336–359, [Zugriff: 2020-06-11]
- Sermanet, Pierre, LeCun, Yann* (2011): Traffic Sign Recognition with Multi-Scale Convolutional Networks, in: *The 2011 International Joint Conference on Neural Networks, 2011 International Joint Conference on Neural Networks (IJCNN 2011 - San Jose)*, San Jose, CA, USA: IEEE, 2011-07, S. 2809–2813, [Zugriff: 2020-10-19]
- Shakhuro, Vladislav, Konushin, Anton* (2016): Russian Traffic Sign Images Dataset, in: *Computer Optics*, 40 (2016), Nr. 2, S. 294–300
- Simonyan, Karen, Vedaldi, Andrea, Zisserman, Andrew* (2014): Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, in: *Bengio, Yoshua, LeCun, Yann* (Hrsg.), *2nd International Conference on Learning Representations, ICLR 2014, Conference Track Proceedings, International Conference on Learning Representations*, Banff, AB, Canada, 2014, S. 1–8
- Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.* (2012): Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition, in: *Neural Networks* (2012), Nr. 0, S. 323–332
- Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, Wolf, Lior* (2014): Deepface: Closing the Gap to Human-Level Performance in Face Verification, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, o. O., 2014, S. 1701–1708

- UN* (1977): Convention on Road Traffic, Wien, Österreich, 1977-05-21, [Zugriff: 2020-10-21]
- Wei, Donglai, Zhou, Bolei, Torrabra, Antonio, Freeman, William T.* (2015): Understanding Intra-Class Knowledge inside CNN, in: CoRR, abs/1507.02379 (2015), [Zugriff: 2020-10-26]
- Yosinski, Jason, Clune, Jeff, Nguyen, Anh Mai, Fuchs, Thomas J., Lipson, Hod* (2015): Understanding Neural Networks through Deep Visualization, in: CoRR, abs/1506.06579 (2015), [Zugriff: 2020-10-26]
- Zhou, Bolei, Khosla, Aditya, Lapedriza, Agata, Oliva, Aude, Torralba, Antonio* (2016): Learning Deep Features for Discriminative Localization, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA: IEEE, 2016-06, S. 2921–2929, [Zugriff: 2020-06-16]

Internetquellen

Dumoulin, Vincent, Visin, Francesco (2018): A Guide to Convolution Arithmetic for Deep Learning, arXiv: 1603.07285 [cs, stat], <<http://arxiv.org/abs/1603.07285>> (2018-01-11) [Zugriff: 2020-10-17]

Ertler, Christian, Mislej, Jerneja, Ollmann, Tobias, Porzi, Lorenzo, Neuhold, Gerhard, Kuang, Yubin (2020): The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale, arXiv: 1909.04422 [cs], <<http://arxiv.org/abs/1909.04422>> (2020-05-07) [Zugriff: 2020-10-19]

Spiegel (2016): Google-Computer Alpha Go besiegt Lee Sedol 4:1 - DER SPIEGEL - Netzwelt, <<https://www.spiegel.de/netzwelt/gadgets/alphago-besiegt-lee-sedol-mit-4-zu-1-a-1082388.html>> (2016-03-15) [Zugriff: 2020-10-11]

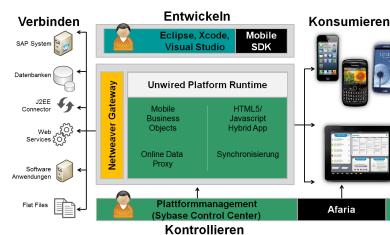
UN (2020): United Nations Treaty Collection, <https://treaties.un.org/Pages/ViewDetailsIII.aspx?src=TREATY&mtdsg_no=XI-B-19&chapter=11&Temp=mtdsg3&lang=en> (2020-10-21) [Zugriff: 2020-10-21]

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit **einverstanden/nicht einverstanden**, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Düsseldorf, 3.11.2020

(Ort, Datum)



(Eigenhändige Unterschrift)