

Gymnasium Bäumlihof, 5Bb

MATURAARBEIT

Kann der Computer Werbung erkennen?

Bilderkennung mit einem Neuronalen Netzwerk

Georg Schwan

Betreuungsperson

Test1

Korreferent

Test2

Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation	3
1.2 Aufbau der Arbeit	3
2 Problemstellung	4
2.1 Werbung	4
2.2 Neuronales Netzwerk	4
3 Neuronales Netzwerk	5
3.1 Konzept	5
3.2 Neuron	5
3.3 Architektur	6
3.3.1 Beschriftung	7
3.4 Wie das Netzwerk lernt	7
3.4.1 Kostenfunktion	8
3.4.2 Gradient Descent	8
3.4.3 Backpropagation	9
3.5 Aktivierungsfunktionen	10
3.5.1 Sigmoid	10
3.5.2 Rectified linear units	11
3.5.3 Softmax	11
3.6 Convolution	12
3.6.1 Architektur	12
4 Lösungsansatz	15
4.1 Logo	15
4.2 Das Netzwerk trainieren	16
4.2.1 Farbe	16
5 Umsetzung	18
6 Reflexion	19
Literaturverzeichnis	20

Kapitel 1

Einleitung

1.1 Motivation

Vor ein paar Jahren haben wir Fernsehen geschaut und immer wenn Werbung kam haben wir den Sender gewechselt, bis die Werbung vorbei war und das normale Programm weiter lief. Das Problem war nur, dass wir nie wussten wann die Werbung vorbei war. Meinem Bruder ist aufgefallen, dass bei Werbung nie das Logo vom Sender eingespielt wird. Daraufhin hat er probiert ein Algorithmus zu schreiben, der das Logo von einem Sender erkennen kann. Er versuchte das Logo mithilfe von Bedingungen und Schleifen auszudrücken, aber vergebens.

Als ich auf der Suche nach einer Idee für eine Maturaarbeit war erinnerte ich mich wieder an das Problem und an einen neuen Lösungsansatz, nämlich Neuronale Netzwerke, welche Heute überall verwendet werden und extrem Mächtig sind. Die Idee war aber nicht nur ein Logo zu erkennen sondern auch genau verstehen wie ein Neuronales Netzwerk funktioniert und deshalb habe ich mein eigenes Neuronale Netzwerk programmiert.

1.2 Aufbau der Arbeit

Kapitel 2

Problemstellung

Das Ziel dieser Arbeit ist einen Algorithmus zu programmieren der Bilder als Werbung erkennen kann. Dafür wird ein Neuronales Netzwerk benutzt, das sich auf die Bilderkennung beschränkt.

2.1 Werbung

Wie schon gesagt wird bei Werbung das Senderlogo nicht eingeblendet und deswegen kann das Problem vereinfacht werden auf die Frage ob das Senderlogo eingeblendet ist oder nicht (siehe Abbildung 2.1).



(a) Werbung



(b) Keine Werbung

Abbildung 2.1: Kein Senderlogo bei Werbung

2.2 Neuronales Netzwerk

Neuronale Netzwerke sind ein sehr umfangreiches Thema und deswegen begrenze ich mich auf Netzwerke die für die Bilderkennung entscheidend sind. Darunter sind klassische feedforward und convolutional Netzwerke. Auf recurrent neuronale Netzwerke¹ wird nicht näher eingegangen.

¹Ein Neuronales Netzwerk, das geeignet für Sequenzen ist

Kapitel 3

Neuronales Netzwerk

Dieses ganze Kapitel bezieht sich auf das Buch von Michael A. Nielsen[2], ausser es wird anders deklariert.

3.1 Konzept

Wenn man ein normales Programm schreiben will muss man das Problem in viele kleinere aufteilen, bis der Computer fähig ist, es zu lösen. In einem Neuronale Netzwerk wird dem Computer nicht gesagt wie es das Problem lösen kann, sondern ein Neuronales Netzwerk versteht das Problem, indem wir es Beispieldaten geben und es daran lernen kann, bis es seine eigene Lösung gefunden hat. Zum Beispiel, wir wollen einem Netzwerk beibringen ob in einem Bild ein Auto vorkommt, dazu geben wir dem Neuronale Netzwerk viele Bilder, mit und ohne Auto. Mit jedem Bild, dass das Neuronale Netzwerk bekommt, lernt es besser wie ein Auto ausschaut.

Das Konzept eines Neuronales Netzwerk ist nicht etwas neuses. Im Jahre 1957 hat Frank Rosenblatt ein erste Idee eines Neuronales Netzwerk vorgestellt, aber erst in den letzten Jahren ist der grosse Hype ausgebrochen, dies liegt daran, dass man erst jetzt die nötigen Daten und Rechenleistung zu Verfügung hat.

3.2 Neuron

User Gehirn kann Entscheidungen treffe, da wir billionen von Neuronen haben, die miteinander verbunden sind und sich verständigen können. Aber ein Neuron an sich ist praktisch nutzlos, aber in grosser Anzahl können sie komplexeste Probleme lösen.

Nach dem gleichen Prinzip funktioniert ein Neuronales Netzwerk Es besteht aus vielen Neuronen (daher der Name) die miteinander verbunden sind.

Ein Neuron in einem Neuronales Netzwerk wird als Mathematische funktion definiert wie Abbildung 3.1 verdeutlicht. Ein Neuron hat n verschiedene Eingaben, die als x_j bezeichnet werden und mit einem spezifischen Gewicht w_j multipliziert werden. Die Ausgabe erfolgt indem man alle gewichteten Eingaben, mit einem Bias b , addiert und durch eine so genannte

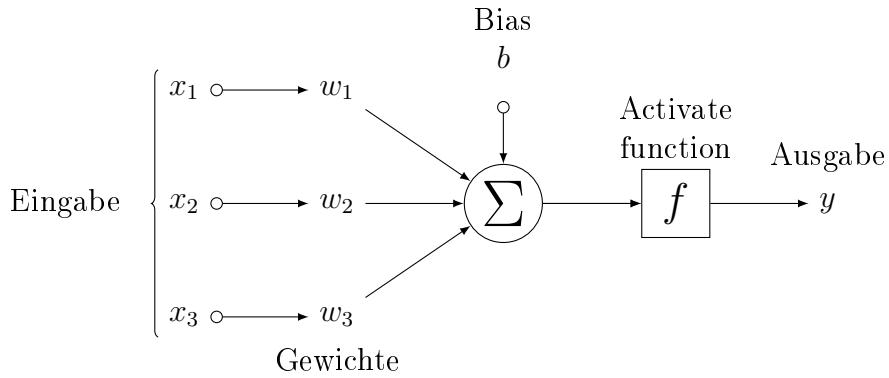


Abbildung 3.1: Einzelner Neuron in einem Neuronalen Netzwerks

Aktivierungsfunktion f durchlaufen lässt. Als Gleichung:

$$y = f \left(\sum_{j=1}^n x_j w_j + b \right)$$

Die Gewichte w_j und der Bias b des Neurons sind die Parameter, die angepasst werden und somit das Neuron lernfähig machen.

Eine Aktivierungsfunktion ist nötig, da ohne eine wäre ein Neuronales Netzwerk eine komplett lineare Funktion, welches nur lineare Probleme lösen könnte.[1] Die meisten Probleme sind viel komplexer als das man sie linear darstellen könnte und deswegen ist eine Aktivierungsfunktion von nötig. Es wird näher auf die Aktivierungsfunktion eingegangen im Abschnitt 3.5

3.3 Architektur

Wie auch im biologischen Gehirn ist ein Neuron allein nutzlos. Erst wenn man die Neuronen miteinander verbindet kann es komplexe Zusammenhänge modellieren.

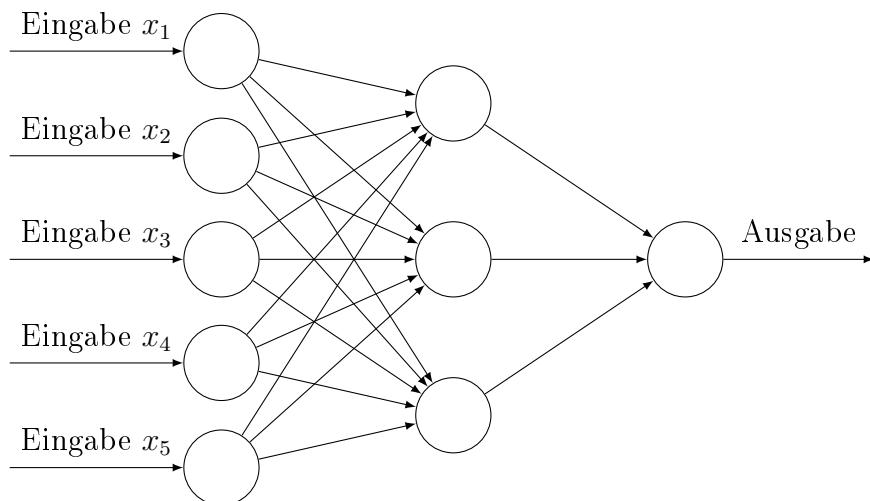


Abbildung 3.2: Mögliche architektur eines Neuronalen Netzwerk

Eine mögliche Architektur kann wie in Abbildung 3.2 ausschauen. Ein Netzwerk wird generell immer in verschiedene Schichten unterteilt. Die linke Schicht wird als eingabe Schicht bezeichnet und die Neuronen in dieser Schicht werden eingabe Neuronen genannt. Analog dazu wird die rechte Schicht ausgabe Schicht genannt, die die ausgabe Neuronen beinhaltet. Die mittleren Schichten, die von der Anzahl variieren können, werden versteckte Schichten genannt. Die Anzahl der Neuronen in jeder Schicht kann auch variieren. Abbildung 3.3 zeigt eine andere mögliche Architektur für ein Netzwerk, welches 2 versteckte Schichten hat. Jeder Neuron von

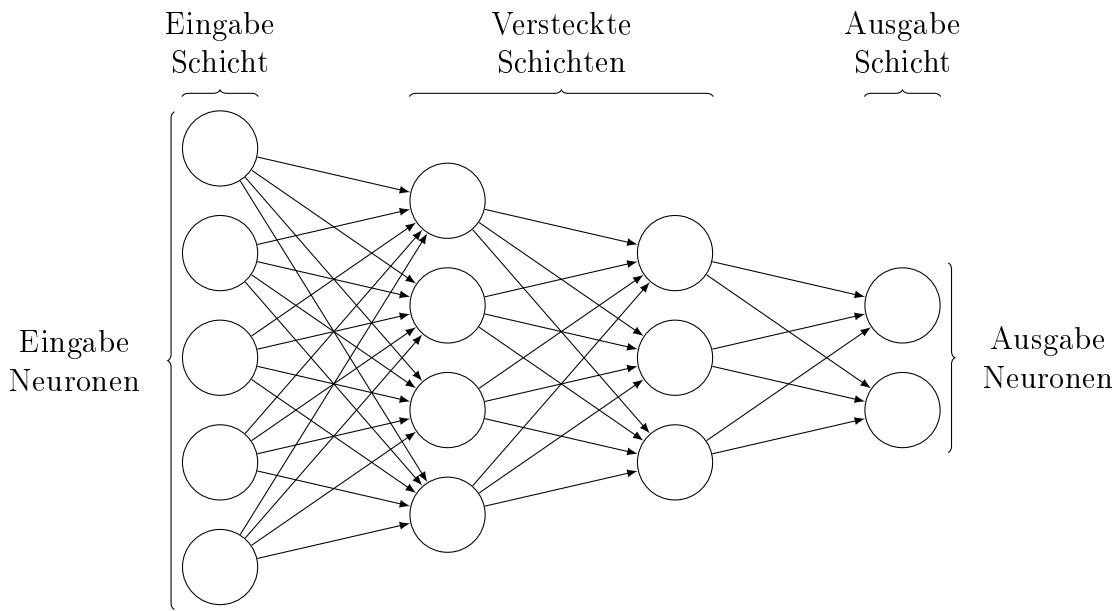


Abbildung 3.3: Neuronales Netzwerk mit 2 versteckten Schichten

der vorigen Schicht ist mit jedem Neuron der nachfolgenden Schicht verbunden. Dies ist ein klassisches feedforawrd Netzwerk. Wichtig zu beachten ist, dass keine Schleifen vorkommen.

3.3.1 Beschriftung

Um eine allgemeine Gleichung zu bestimmen, muss man zuerst die Elemente des Netzwerks benennen. Wir bezeichnen das Gewicht $w_{k,j}^l$ für die Verbindung des k^{ten} Neuron der $(l-1)^{ten}$ Schicht zu dem j^{ten} Neuron der l^{ten} Schicht. Ähnlich dazu bezeichnen wir die Ausgabe des Neurons als a_j^l und der bias des Neurons als b_j^l (siehe Abbildung 3.4).

Mit dieser Notation kann eine Gleichung für das Netzwerk aufgestellt werden, welche der Gleichung einem Neuron ähnelt 3.2.

$$a_j^l = f \left(\sum_k a_k^{l-1} w_{k,j}^l + b_j^l \right)$$

3.4 Wie das Netzwerk lernt

Bis jetzt ging es nur darum wie ein neuronales Netzwerk aufgebaut ist. In dem Abschnitt geht wie ein neuronales Netzwerk, anhand von Daten, lernen kann

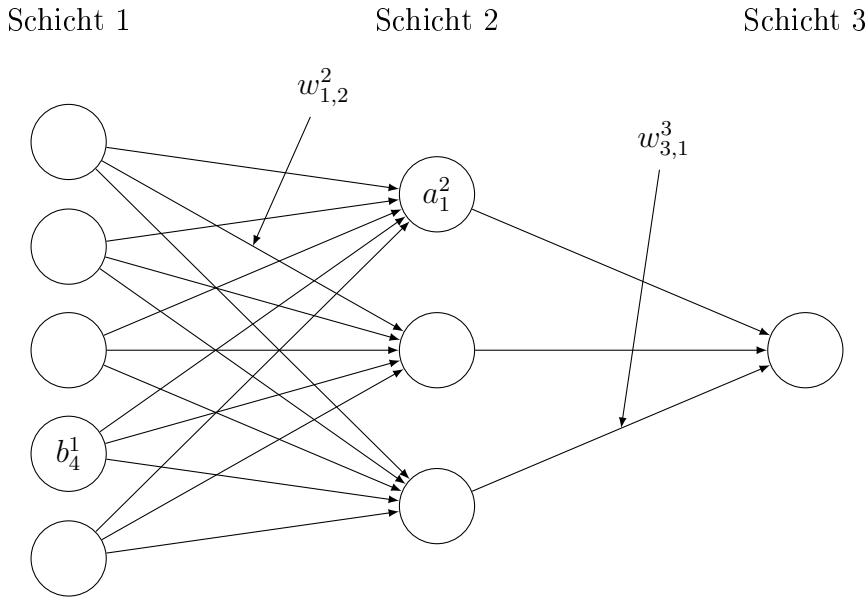


Abbildung 3.4: Bezeichnung der Parameter

3.4.1 Kostenfunktion

Damit ein Netzwerk lernen kann muss man dem Netzwerk zuerst sagen können wie gut oder wie schlecht es gerade ist. Dazu definieren wir eine Kostenfunktion C , die von allen Gewichten w und allen biases b abhängig ist. Das Netzwerk wird als Funktion $y(x)$ bezeichnet. Den Eingabewert wird als x bezeichnet mit dem dazugehörige Label l . Beachte, dass x und l Vektoren sind. Zum Beispiel würde ein Bild, das 10x10 Pixel gross ist, einen $(10 * 10 =) 100$ -dimensionaler Vektor haben und das Label, wenn es 3 ausgabe Neuronen gibt, einem 3-dimensional Vektor.

$$C(w, b) = \sum_j (y(x)_j - l_j)^2$$

Das Ziel des Netzwerkes ist diese Kostenfunktion zu minimieren, bis idealerweise $C \approx 0$, dies geschieht wenn die Ausgabe des Netzwerks und des Label ähnlich sind.

3.4.2 Gradient Descent

Um diese Kostenfunktion zu minimieren wird ein Algorithmus names *Gradient Descent* benutzt. Das Konzept basiert darauf, dass man eine Funktion, in abhängigkeit einer Variablen, ableiten kann und so die Steigung and diesem Punkt berechnen kann und die Variable richtung Minimum anpasst.

Bild Einfügen

Zum Beispiel hat man eine Kostenfunktion $C(a, b)$ die von a und b abhängig ist. Wenn man diese Graphisch abbildet (siehe Abbildung REF).

Die Variablen a und b werden am Anfang zufällige Werte zugeteilt und das Ziel ist sie so an-

zupassen, dass man das Minimum der Kostenfunktion findet. Um das Minimum zu finden kann man sich einen Ball vorstellen, der in das Minimum herunterrollt. Um dies zu berechnen muss man die Steigung, mithilfe einer Ableitung, herausfinden und die Variable in die gegensätzliche Richtung bewegen (siehe Abbildung REF).

Bild Einfügen

Für Bewegung ergibt sich:

$$\begin{aligned} a \rightarrow a' &= a - \mu \frac{\partial C}{\partial a} \\ b \rightarrow b' &= b - \mu \frac{\partial C}{\partial b} \end{aligned}$$

wobei μ eine kleine positive Zahl (learning rate genannt) ist, die die Geschwindigkeit der Bewegung steuert. Ausserdem beachte, dass der Ball keine Beschleunigung hat. Wenn man diese Gleichung nun immer wieder anwendend gelangt man zum Minimum der Kostenfunktion.

Der Algorithmus funktioniert auch bei mehr als nur 2 Variablen und sieht fürs Neuronale Netzwerk identisch aus.

$$\begin{aligned} w_{k,j}^l \rightarrow w_{k,j}^{l'} &= w_{k,j}^l - \mu \frac{\partial C}{\partial w_{k,j}^l} \\ b_j^l \rightarrow b_j^{l'} &= b_j^l - \mu \frac{\partial C}{\partial b_j^l} \end{aligned}$$

Durch dieses Verfahren kann zwar relativ einfach das Minimum gefunden werden, dabei ist aber zu beachten, dass es nur ein lokales Minimum ist und kein globales.

3.4.3 Backpropagation

Der Algorithmus um $\frac{\partial C}{\partial w_{k,j}^l}$ und $\frac{\partial C}{\partial b_j^l}$ zu berechnen wird als Backpropagation bezeichnet und ist der mathematisch schwerste Teil dieser Arbeit. Es ist aber nicht unbedingt nötig für das Verständnis eines Neuronale Netzwerkes. Es wird auch nicht näher auf die Beweise der Gleichungen eingegangen (oder soll ich???), da es sonst komplizierter wird und im Grunde ist es nur die Anwendung der Kettenregel.

Um die Übersicht zu behalten wird eine Zwischenmenge δ_j^l eingeführt, welches als *Fehler* bezeichnet wird. Der Fehler sagt aus wie gut oder schlecht ein Neuron ist und ist definiert als:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

wobei z_j^l die Ausgabe von einem Neuron ohne die Aktivierungsfunktion ist, also $a_j^l = f(z_j^l)$.

Mit dieser Definition kann man den Fehler in der letzten Schicht L bestimmen:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$$

In unserem Fall benutzen wir eine Quadratische Kostenfunktion $C = \sum_j (a_j^L - y_j)^2$ bei der die Ableitung $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_j)$ ist und können δ_j^L einfacher definieren als:

$$\delta_j^L = 2(a_j^L - y_j) f'(z_j^L)$$

Bei der berechnung des Fehlers δ_j^l Abhängig von δ_j^{l+1} bekommt man:

$$\delta_j^l = \sum_k w_{j,k}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

Beachte, dass bei $w_{j,k}^{l+1}$ das j und k vertauscht sind, so dass man durch alle Neuronen der $(l+1)^{ten}$ Schicht durch iteriert. Mit dieser Gleichung kann man jeden Fehler von jeder Schicht berechnen, indem man von hinten durch das Netzwerk durchläuft. Ähnlich wie wenn man sich beim Netzwerk nach vorne bewegt.

Die Gleichung für die Änderungsrate der Kosten in Bezug auf ein Bias im Netzwerk ist genau der Fehler:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Die Gleichung für die Änderungsrate der Kosten in Bezug auf ein Gewicht im Netzwerk:

$$\frac{\partial C}{\partial w_{k,j}^l} = a_k^{l-1} \delta_j^l$$

3.5 Aktivierungsfunktionen

Das einzige was noch fehlt ist wie eine Aktivierungsfunktionen genau ausschaut. Wie schon gesagt darf eine Aktivierungsfunktionen nicht linear sein, da sie sonst nichts neues dem Netzwerk beiträgt.

3.5.1 Sigmoid

Ein Beispiel für eine Aktivierungsfunktion ist die Sigmoid Funktion $f(x) = \frac{1}{1+e^{-x}}$ (siehe Abbildung 3.5). Besonders an dieser Funktion ist, dass sie den Ausgabewert zwischen 0 und 1 eingrenzt, was uns erlaubt den Ausgabewert des ganzen Netzwerkes besser zu deuten, als wenn der Wert zwischen $-\infty$ und ∞ liegt. Ein Problem der Sigmoid Funktion ist, dass wenn die Ausgabe nah bei 1 oder 0 ist, dann ist die Ableitung $f'(x)$ davon auch nah bei 0, was den Fehler δ_j^l sehr klein hält und so das Netzwerk nur noch langsam lernen lässt. Dieses Problem ist als *Vanishing gradient problem* bekannt.

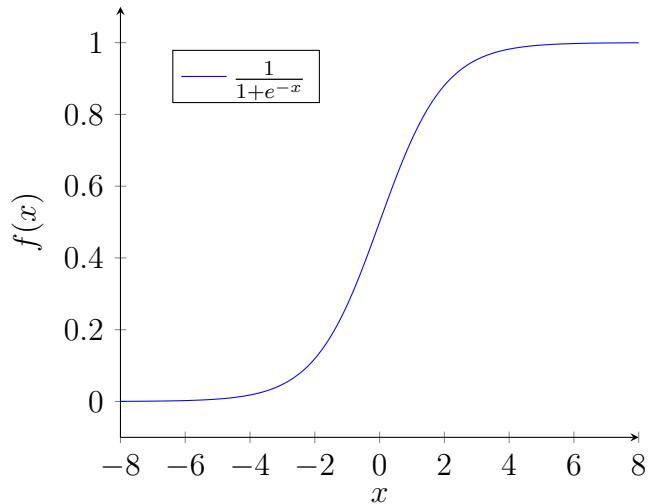


Abbildung 3.5: Sigmoid Aktivierungsfunktionen

3.5.2 Rectified linear units

Eine andere populäre Aktivierungsfunktion ist die Rectified linear units Funktion oder kurz ReLu. Die Funktion $f(x) = \max(0, x)$ (siehe Abbildung 3.6). löst das Problem des Vanishing

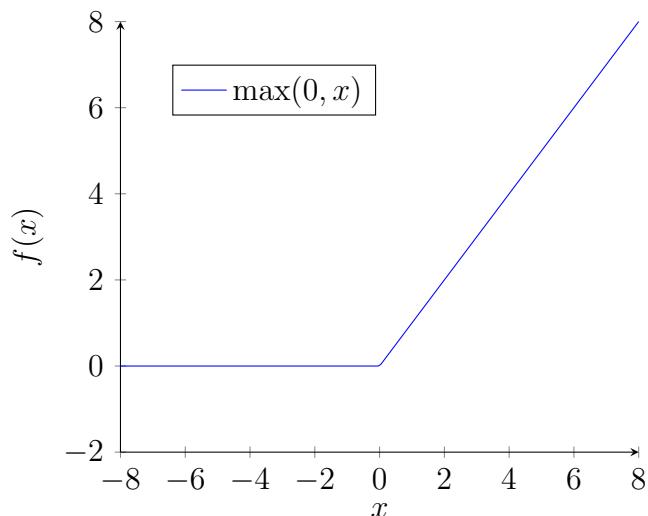


Abbildung 3.6: ReLu Aktivierungsfunktionen

gradient und praktisch alle Neuronalen Netzwerke benutzen Relu als ihre Aktivierungsfunktion, da es die besten Ergebnisse erbringt. Ein Nachteil ist, dass sie nur in den Versteckten Schichten gut funktioniert, da der Ausgabewert des Netzwerks in einem unbestimmten Bereich ist.

3.5.3 Softmax

Die Softmax funktion wird verwendet um eindeutige Klassifikationen zu machen und ist definiert als:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

Das besondere an dieser Aktivierungsfunktion ist, dass sie nicht nur einen Wert braucht, sondern alle Werte der ganzen Schicht. Außerdem gibt die Summe aller Resultate von a^L gleich 1 und kann deswegen als eine Wahrscheinlichkeitsverteilung verstanden werden. Dies ist oft sehr hilfreich, da viele Probleme nur ein richtiges Resultat haben, zum Beispiel hat man Bilder von Zahlen, wo immer nur eine Zahl pro Bild zu sehen ist. Und durch die softmax Funktion sieht man dann eine geschätzte Wahrscheinlichkeit vom Netzwerk für jede Zahl.

3.6 Convolution

Bis jetzt ging es nur um Schichten die völlig miteinander verbunden sind. Für die Bilderkennung kann das suboptimal sein, da bestimmte Eigenschaften eines Bildes nicht miteinbezogen werden, wie zum Beispiel die Beziehung von nebeneinander liegenden Pixeln und das gesuchte Objekt in einem Bild an verschiedenen Orten vorkommen kann.

3.6.1 Architektur

Die Eingabe für einen convolutional Schicht ist nicht 1-Dimensional, sondern 2-Dimensional (siehe Abbildung 3.7). Die Neuronen werden normal verbunden, einfach mit dem Unterschied,

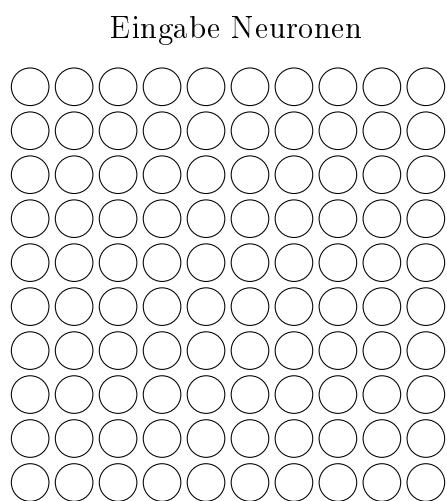


Abbildung 3.7: Eingabe Neuronen für eine convolution Schicht

dass nicht jeder Neuron mit jedem Neuron verbunden wird, sondern dass nur ein bestimmter Bereich zum nächsten Neuron verbunden ist. Dieser Bereich wird als *Filter* bezeichnet und in dem Beispiel auf Abbildung 3.8 wird ein 3x3 Filter benutzt. Der Filter wird dann auf den Eingabe Neuronen um ein Neuron verschoben, um den nächsten Neuron zu verbinden. Und so geht das weiter, auch nach unten, bis die ganze verdeckte Schicht gemacht wurde. Dabei wird die verdeckte Schicht auch kleiner, in dem Beispiel wird die 10x10 Schicht zu einer 8x8 Schicht, da der Filter irgendwann am anderen Rand anstößt. Abbildung 3.9 verdeutlicht das Prinzip noch einmal.

Der Filter kann auch um mehr als nur einen Neuronen verschoben werden, und man kann

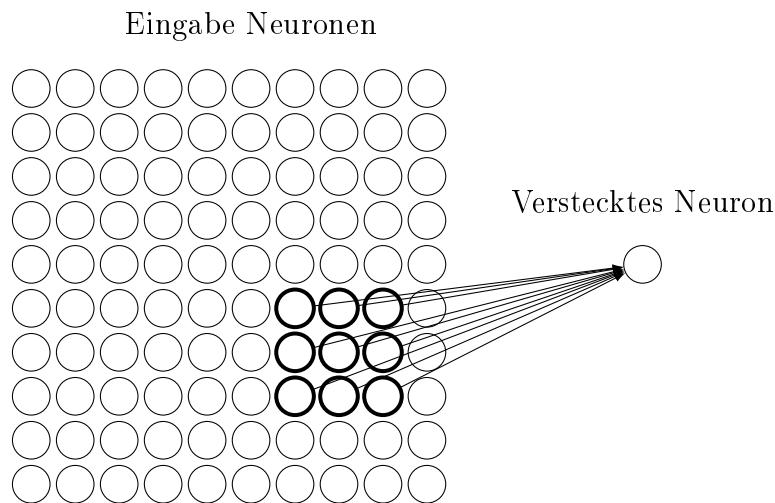


Abbildung 3.8: Verbindung eines versteckten Neurons in einem convolution Schicht

in den beiden Richtungen verschiedene Werte nehmen, zum Beispiel bewegt sich der Neuron nach links um zwei Neuronen und nach unten um drei.

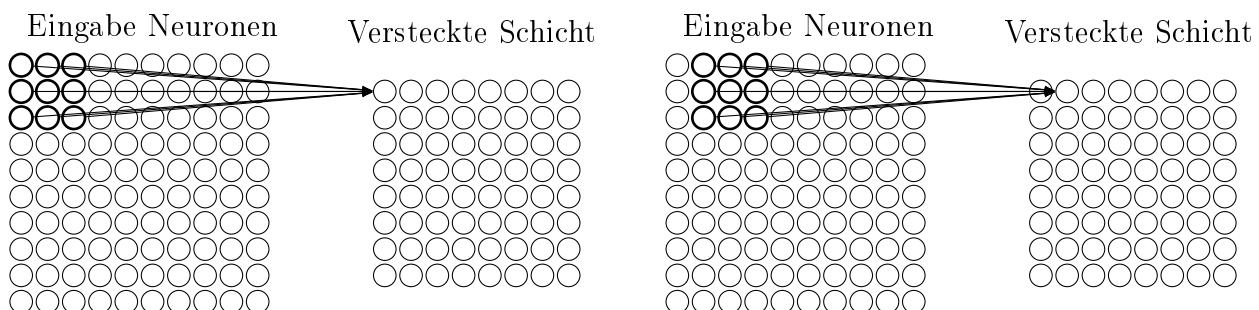


Abbildung 3.9: Bewegung eines Filters über eine convolution Schicht

Das entscheidende am Filter ist, dass er die gleichen Gewichte und Bias verwendet für die Verbindung, d.h bei einem Filter von 5×5 gibt es $(5 * 5 =) 25$ verschiedene Gewichte und einen Bias. Wenn nun der Filter bewegt wird werden die gleichen Gewichte und der gleiche Bias verwendet. Als Gleichung bei einem 3×3 Filter:

$$a_{j,k}^{l+1} = f \left(\sum_{p=0}^2 \sum_{m=0}^2 w_{p,m}^l a_{j+p,k+m}^l + b^l \right)$$

wobei $a_{x,y}$ der Neuron an der Position x, y ist.

Dadurch das immer die gleichen Gewichte und der gleiche Bias für jeden Filter benutzt werden, wird überall das gleiche Merkmal erkannt selbst wenn es sich an verschiedenen Orten befindet, deswegen wird die Ausgabe von dem Filter als *Feature Map* bezeichnet. Normalerweise will man mehr als nur ein Merkmal erkennen und deswegen werden mehrere Filter verwendet (siehe Abbildung ??).

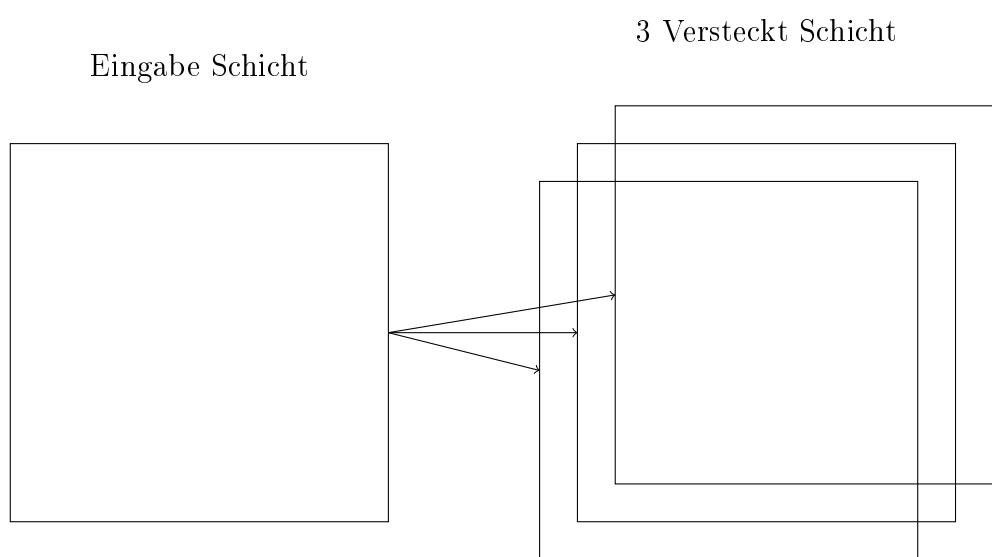


Abbildung 3.10: Convolution Netzwerk mit 3 Feature Maps

Kapitel 4

Lösungsansatz

experimental

4.1 Logo

Um das Logo zu erkennen muss man zuerst verstehen wie es auf das Bild gelangt. Man könnte meinen das Logo wird einfach über das ander Bild gelegt wird, aber das Logo wird eingespielt indem es mit dem Bild negativ multipliziert wird, d.h die das Bild und das Logo werden invertiert, multipliziert und dann wieder invertiert.[3]

$$f(a, b) = 1 - (1 - a)(1 - b)$$

wobei a das Bild ist und b das Logo und die Werte von jedem Pixel von 0 (Schwarz) bis 1 (Weiss) gehen.

Das bewirkt, dass man leicht durch das Logo hindurchsehen kann, wodurch der Hintergrund hinter dem Logo auch eine Rolle spielt (siehe Abbildung 4.1a,b). Eine grosse Schwierigkeit ist, dass bei manchen Hintergründen, vor allem bei weissen, das Logo abgeschnitten oder kaum bis gar nicht sichtbar ist. Bei Abbildung 4.1 (c) ist das Logo abgeschnitten, bei (d) ist es kaum sichtbar und bei (e) ist komplett verschwunden. Das ist auch einfach zu erklären. Wenn man für $a = 1$ (Weiss) in die Formel oben einsetzt erhält man: $f(1, b) = 1$, was bedeutet, dass das ganze Bild weiss ist und das Logo verschwunden ist. Analog dazu, wenn man für $a = 0$ (Schwarz) einsetzt erhält man: $f(0, b) = b$, was bedeutet, dass das Logo, bei einem schwarzem Hintergrund, im ursprünglichen Zustand befindet. Wegen dem kann man das Logo komplett herausfiltern und wieder verwenden wenn man das Logo mit einem schwarzen Hintergrund findet (siehe Abbildung 4.1f).

Ein andere Schwierigkeit ist, dass das Logo nicht unbedingt immer am gleichen Ort sein muss. Zum Beispiel hat Prosieben 3 verschiedene Positionen (siehe Abbildung 4.2).

Alle diese Schwierigkeiten machen das erkennen eines Logo's ohne ein Neuronales Netzwerk extrem schwer. Ein Neuronales Netzwerk hingegen löst das Probleme ziemlich elegant.

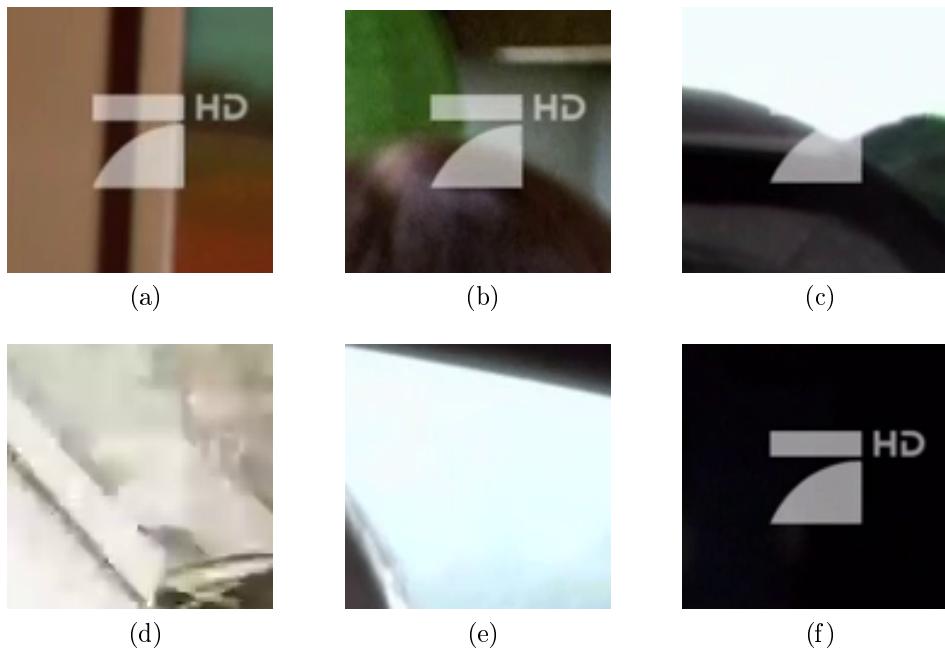
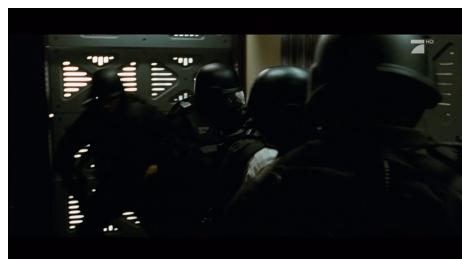


Abbildung 4.1: Logo mit verschiedenen Hintergründen

4.2 Das Netzwerk trainieren

4.2.1 Farbe

Die erste Frage, die sich stellt ist, ob man das Netzwerk mit farbigen oder mit schwarz-weiss Bildern trainieren will. Dazu wird ein eigen gesammeltes Datensatz benutzt, das 7830 Proseben Bilder besteht. Aus dem wurde die Logos ausgeschnitten mit einem Rand von 10 Pixeln.



(a)



(b)



(c)

Abbildung 4.2: Logo an verschiedenen Positionen

Kapitel 5

Umsetzung

Kapitel 6

Reflexion

Literaturverzeichnis

- [1] Anish Singh Walia. Activation functions and it's types-which is better? <https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f>, 2015. [Online; accessed 14. Oktober 2018].
- [2] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. [Online; accessed 14. Oktober 2018].
- [3] Wikipedia contributors. Blend modes — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Blend_modes, 2018. [Online; accessed 14. Oktober 2018].

Abbildungsverzeichnis

2.1	Kein Senderlogo bei Werbung	4
3.1	Einzelner Neuron in einem Neuronalen Netzwerks	6
3.2	Mögliche architektur eines Neuronalen Netzwerk	6
3.3	Neuronales Netzwerk mit 2 versteckten Schichten	7
3.4	Bezeichnung der Parameter	8
3.5	Sigmoid Aktivierungsfunktionen	11
3.6	ReLU Aktivierungsfunktionen	11
3.7	Eingabe Neuronen für eine convolution Schicht	12
3.8	Verbindung eines versteckten Neurons in einem convolution Schicht	13
3.9	Bewegung eines Filters über eine convolution Schicht	13
3.10	Convolution Netzwerk mit 3 Feature Maps	14
4.1	Logo mit verschiedenen Hintergründen	16
4.2	Logo an verschiedenen Positionen	17

Ehrlichkeitserklärung

Die eingereichte Arbeit ist das Resultat meiner persönlichen, selbstständigen Beschäftigung mit dem Thema. Ich habe für sie keine anderen Quellen benutzt als die in den Verzeichnissen aufgeführten. Sämtliche wörtlich übernommenen Texte (Sätze) sind als Zitate gekennzeichnet.

Insert Datum