

Gymnasium Bäumlihof, 5Bb

MATURAARBEIT

Kann der Computer Werbung erkennen?

Bilderkennung mit einem neuronalen Netzwerk

Georg Schwan

Betreuungsperson

Dr. Christian Boller

Korreferent

Dr. Helmut Locher

Basel, 28. November 2018

Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation	3
1.2 Aufbau der Arbeit	3
1.3 Einschränkung	3
1.4 Problemstellung	4
1.4.1 Logo	4
2 Neuronales Netzwerk	6
2.1 Konzept	6
2.2 Neuron	6
2.3 Architektur	7
2.3.1 Beschriftung	9
2.4 Wie das Netzwerk lernt	9
2.4.1 Kostenfunktion	9
2.4.2 Gradient Descent	10
2.4.3 Backpropagation	11
2.5 Aktivierungsfunktionen	12
2.5.1 Sigmoid	12
2.5.2 Rectified Linear Units	13
2.5.3 Softmax	14
2.6 Convolution	14
2.6.1 Architektur	14
2.6.2 Max Pooling	17
2.6.3 Convolution-Schichten und Fully-connected-Schichten	17
2.7 Regularization	18
2.7.1 Dropout	18
2.7.2 Batch Normalization	19
3 Lösungsansatz	20
3.1 Logo	20
3.2 Werbung erkennen mithilfe des Logos	21
3.3 Werbung erkennen ohne Hilfe des Logos	22

4 Umsetzung	23
4.1 Neuronales Netzwerk	23
4.1.1 Aufbau	23
4.1.2 Benutzung	24
4.1.3 Grafikkartenunterstützung	24
4.1.4 Bildbearbeitung	24
4.2 Webserver	25
5 Ergebnisse und Auswertung	26
5.1 Auswertung	26
5.1.1 Datensatz	26
5.1.2 Methoden	27
5.2 Neuronales Netzwerk mithilfe des Logos	28
5.2.1 Architektur	28
5.2.2 Training	29
5.2.3 Auswertung	29
5.3 Neuronales Netzwerk ohne das Logo	31
5.3.1 Datenbeschaffung	31
5.3.2 Architektur	32
5.3.3 Training	33
5.3.4 Auswertung	33
6 Fazit und Weiterführung	35
6.1 Fazit	35
6.2 Mögliche Weiterführung	35
Literaturverzeichnis	37
Abbildungsverzeichnis	39

Kapitel 1

Einleitung

1.1 Motivation

Vor ein paar Jahren haben wir beim Fernsehen bei Werbepausen immer den Sender gewechselt bis die Werbung vorbei war und das normale Programm weiterlief. Das Problem war nur, dass wir nie wussten, wann die Werbung vorbei war und wir wieder zum vorigen Sender zurückschalten mussten. Meinem Bruder ist damals aufgefallen, dass bei Werbung nie das Logo des Fernsehsenders eingespielt wird. Daraufhin hat er probiert einen Algorithmus zu schreiben, der das Logo vom Sender erkennen kann. Er versuchte das Logo mithilfe von Bedingungen und Schleifen auszudrücken, aber es funktionierte nicht.

Als ich auf der Suche nach einer Idee für eine Maturaarbeit war, erinnerte ich mich wieder an das Problem. Ich wählte einen neuen Lösungsansatz, nämlich die Verwendung eines neuronalen Netzwerks. Die Idee war aber nicht nur ein Logo zu erkennen, sondern auch genau zu verstehen wie ein neuronales Netzwerk funktioniert.

1.2 Aufbau der Arbeit

Im Abschnitt 1.4 wird die genaue Problemstellung erklärt. Im Kapitel 2 wird das neuronale Netzwerke beschrieben. Im Kapitel 3 wird die Lösungsidee präsentiert. Im Kapitel 4 wird grob die Implementierung des neuronalen Netzwerkes beschrieben. Im Kapitel 5 werden die Ergebnisse der Netzwerke präsentiert und ausgewertet. Im Kapitel 6 ziehe ich ein Fazit und zeige mögliche Weiterführungen auf, die die Ergebnisse noch verbessern könnten.

1.3 Einschränkung

Neuronale Netzwerke sind ein sehr umfangreiches Thema und deswegen begrenze ich mich auf Netzwerke, die für die Bilderkennung entscheidend sind. Ich konzentriere mich auf die klassischen "Feedforward" und "Convolution" Netzwerke. Auf "Recurrent" Netzwerke¹ gehe ich nicht

¹ "Recurrent" Netzwerke sind komplexere und erlauben Rückkopplungen, die das Verarbeiten von Sequenzen erlauben, zum Beispiel für die Spracherkennung[22]

näher ein.

1.4 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.

1.4.1 Logo

Wie schon gesagt wird bei Werbung das Senderlogo nicht eingeblendet. Deswegen kann das Problem vereinfacht werden auf die Frage, ob das Senderlogo eingeblendet ist oder nicht (siehe Abbildung 1.1). Man könnte meinen, dass das Erkennen eines Logos relativ simple ist. Zum



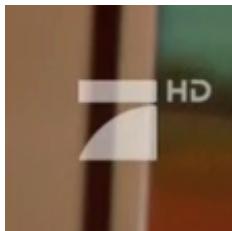
(a) Werbung



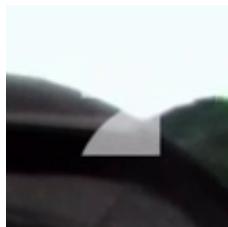
(b) Keine Werbung

Abbildung 1.1: Kein Senderlogo bei Werbung

Beispiel könnte man überprüfen, ob der Bereich, wo das Logo sein sollte, heller ist als ausserhalb.



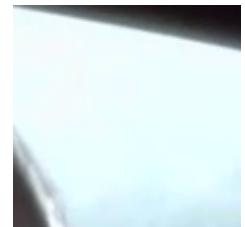
(a)



(b)



(c)

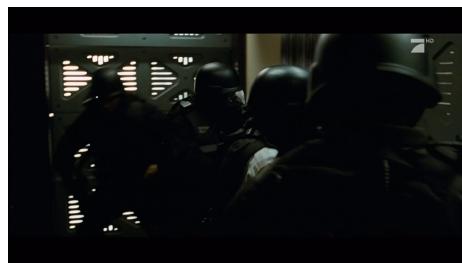


(d)

Abbildung 1.2: Logo mit verschiedenen Hintergründen

Ein Problem des Logos ist, dass es nicht einfach über das normale Bild eingespielt wird, sondern dass man leicht hindurchsehen kann (siehe Abbildung 1.2a). Dadurch kann das Logo nicht an gleichen Pixel erkannt werden. Die grösste Schwierigkeit ist aber, dass bei manchen Hintergründen, vor allem bei Weissen, das Logo abgeschnitten oder kaum bis gar nicht sichtbar ist. Bei Abbildung 1.2 (b) ist das Logo abgeschnitten, bei (c) ist es kaum sichtbar und bei (d) ist es komplett verschwunden.

Eine weitere Schwierigkeit ist, dass das Logo nicht unbedingt immer am gleichen Ort sein muss. Zum Beispiel kann das Logo beim Fernsehkanal Prosieben drei verschiedene Positionen



(a)



(b)



(c)

Abbildung 1.3: Logo an verschiedenen Positionen

haben (siehe Abbildung 1.3). Alle diese Schwierigkeiten machen das Erkennen eines Logos ohne ein neuronales Netzwerk extrem schwierig. Ein neuronales Netzwerk hingegen löst das Problem ziemlich elegant.

Diese Arbeit geht zudem auf die Frage ein, ob ein neuronales Netzwerk auch ohne die Berücksichtigung eines Logos Werbung erkennen kann.

Kapitel 2

Neuronales Netzwerk

Dieses Kapitel bezieht sich auf das Buch von Michael A. Nielsen[13], ausser es wird anders angegeben.

2.1 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 neuronalen Netzwerk wird dem Computer nicht gesagt, wie es das Problem lösen kann, sondern ein neuronales Netzwerk versteht das Problem, indem es Beispieldaten bekommt und an ihnen lernen kann, bis es seine eigene Lösung gefunden hat. Zum Beispiel wollen wir einem Netzwerk beibringen, ob in einem Bild ein Auto vorkommt. Dazu geben wir dem neuronalen 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 neuronalen Netzwerks ist schon lange bekannt. Im Jahre 1957 hat Frank Rosenblatt eine erste Idee eines neuronalen Netzwerks vorgestellt. Die Idee war, aus mathematischen Funktionen unser Gehirn zu modellieren, indem man die biologischen Neuronen und Synapsen als mathematische Funktion ausdrückt.

Die Verwendung neuronaler Netzwerke ist aber erst in den letzten Jahren massiv angestiegen. Dies liegt daran, dass man erst jetzt die nötigen Daten und Rechenleistung zur Verfügung hat.

2.2 Neuron

Unser Gehirn kann Entscheidungen treffen, weil Neuronen, also Nervenzellen, miteinander verbunden sind und sich verstündigen können. Ein einzelnes Neuron ist praktisch nutzlos, aber in grosser Anzahl können sie komplexeste Probleme lösen.

Nach dem gleichen Prinzip funktioniert ein künstliches neuronales Netzwerk. Es besteht aus vielen Neuronen, die als mathematische Funktionen definiert sind und die miteinander

verbunden sind, indem sie Eingangswerte empfangen und ein bestimmtes Ergebnisse jeweils weiterleiten.

Die mathematische Definition eines Neurons in einem neuronale Netzwerk wird in Abbildung 2.1 dargestellt und zeigt wie verschiedene Eingangswerte (in diesem Beispiel x_1, x_2, x_3) zu einem bestimmten Ergebnis y führen.

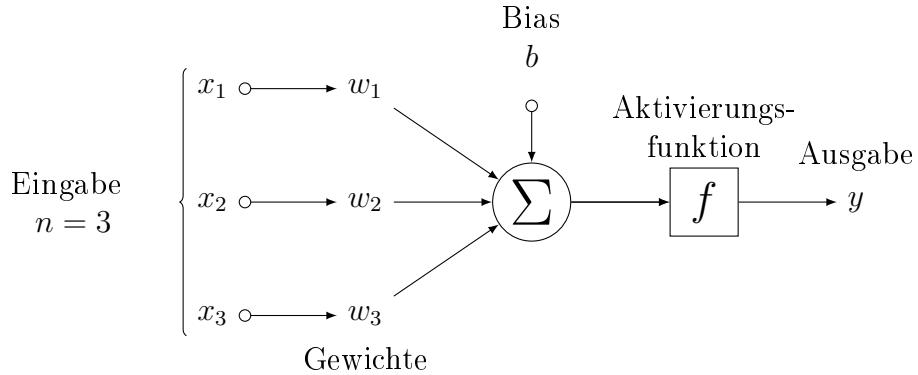


Abbildung 2.1: Einzelner Neuron in einem Neuronalen Netzwerks

Ein Neuron hat n verschiedene Eingaben, die als x_j bezeichnet werden und mit einem spezifischen Gewicht w_j multipliziert werden. Die Ausgabe y erfolgt, indem man alle gewichteten Eingaben, mit einem Bias b , addiert und durch eine so genannte Aktivierungsfunktion f durchlaufen lässt.

Eine klassische Aktivierungsfunktion ist die Sigmoid Funktion $f(x) = \frac{1}{1+e^{-x}}$, welche den Wert zwischen 0 und 1 normalisiert. Eine Aktivierungsfunktion ist nötig. Ohne sie wäre ein neuronales Netzwerk eine komplett lineare Funktion, welche nur lineare Probleme lösen könnte[19], da in einem Netzwerk nur multipliziert und addiert wird. Durch die Aktivierungsfunktion kommt eine nicht lineare Funktion hinzu. Diese macht das Netzwerk komplizierter, aber auch mächtiger, da es so Beziehungen von Datenpunkten auch nicht linear miteinander verknüpfen kann. Ohne diese Aktivierungsfunktion wäre das Netzwerk nicht in der Lage, komplizierte Zusammenhänge wie auf Bildern oder in der Sprache zu erkennen[19].

Das Neuron lässt sich als mathematische Funktion wie folgt darstellen:

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

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

Im Abschnitt 2.5 wird noch näher auf die Aktivierungsfunktion eingegangen.

2.3 Architektur

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

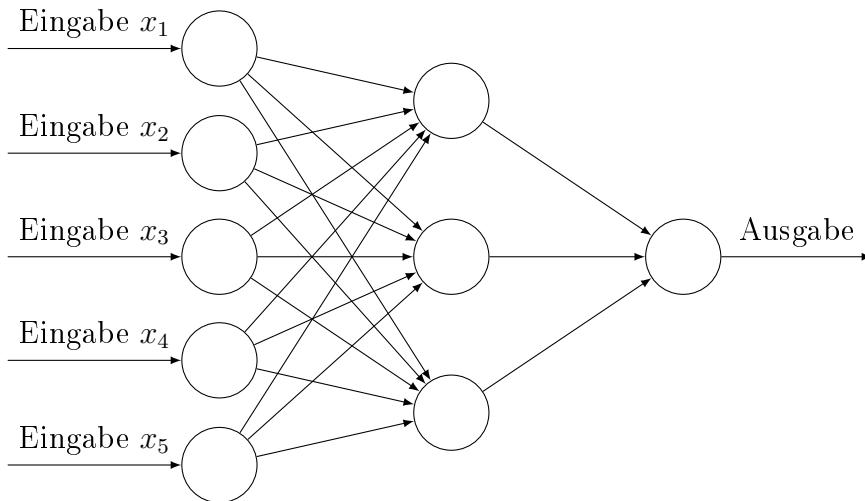


Abbildung 2.2: Mögliche Architektur eines neuronalen Netzwerk

Eine mögliche Architektur kann wie in Abbildung 2.2 ausschauen. Ein Netzwerk wird generell immer in verschiedene Schichten unterteilt. Die linke Schicht wird als Eingabeschicht bezeichnet und die Neuronen in dieser Schicht werden Eingabeneuronen genannt. Das Besondere der Eingabeneuronen besteht darin, dass jeweils nur ein Eingabewert erfolgt, welcher unverändert als Ausgabe an die Neuronen der nächsten Schicht weiter gegeben wird. Die äußerst rechte Schicht wird als Ausgabeschicht bezeichnet, die die Ausgabeneuronen beinhaltet. Die mittleren Schichten, die von der Anzahl her variieren können, werden versteckte Schichten genannt. Die Anzahl der Neuronen in jeder Schicht kann variieren. Abbildung 2.3 zeigt eine andere mögliche Architektur für ein Netzwerk, welche zwei versteckte Schichten hat. Jedes Neu-

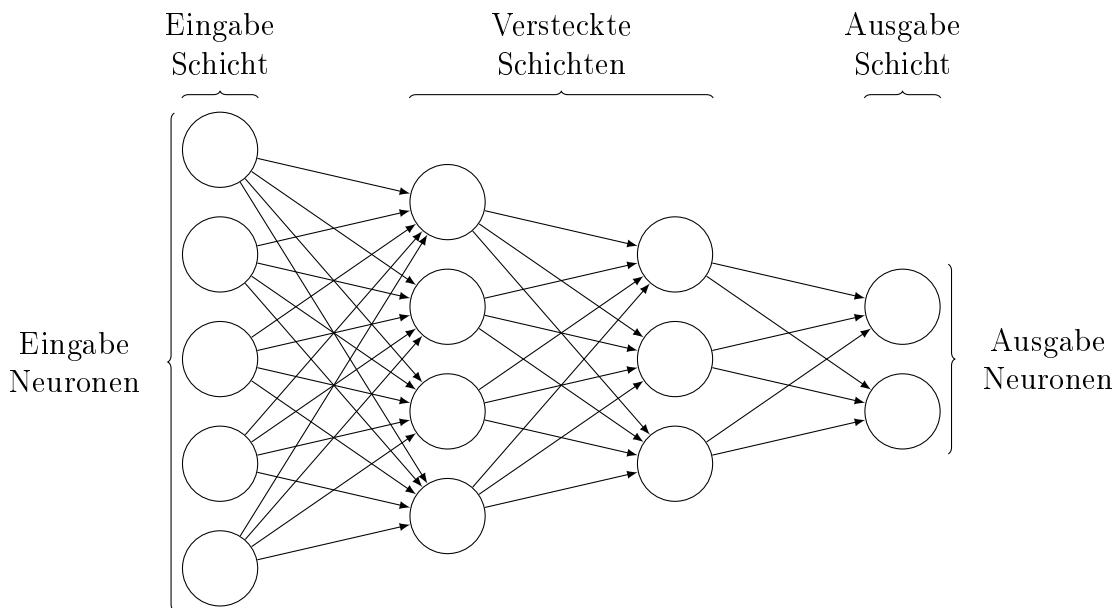


Abbildung 2.3: Neuronales Netzwerk mit 2 versteckten Schichten

ron der vorigen Schicht ist mit jedem Neuron der nachfolgenden Schicht verbunden, welches als *Fully-connected-Schicht* bezeichnet wird. Dies ist ein klassisches Feedforward Netzwerk, welches

nur Verbindungen nach vorne hat. So können keine Schleifen entstehen¹.

2.3.1 Beschriftung

Um eine allgemeine Gleichung zu bestimmen, muss man zuerst die Elemente des Netzwerks benennen. Wir bezeichnen das Gewicht² $w_{k,j}^l$ 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 den Bias des Neurons als b_j^l (siehe Abbildung 2.4).

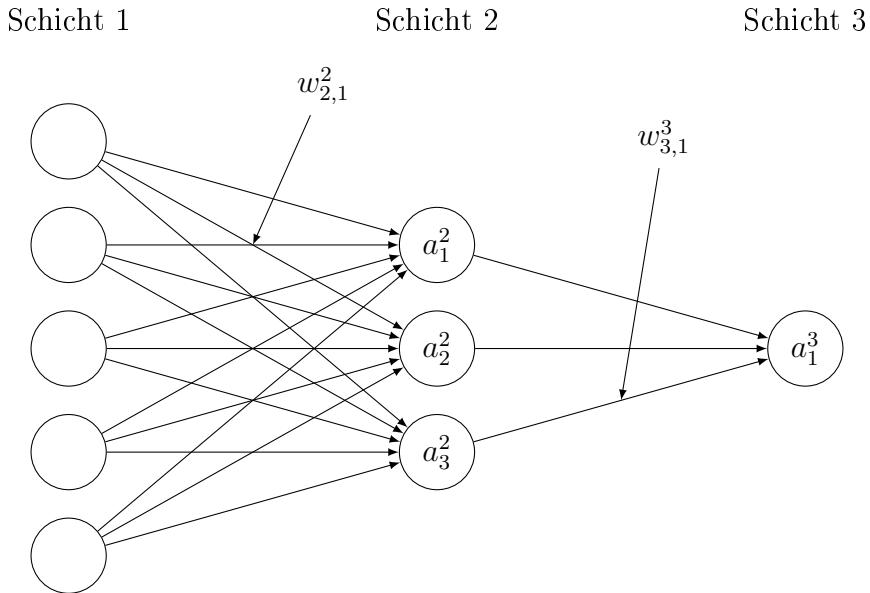


Abbildung 2.4: Bezeichnung der Parameter

Mit dieser Notation kann jedes Neuron des Netzwerks als Funktion wie folgt dargestellt werden.

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

2.4 Wie das Netzwerk lernt

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

2.4.1 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. Der Ausgabewert des kompletten Netzwerks wird als y

¹Es gibt auch Architekturen, in denen Schleifen vorkommen, aber auf diese wird nicht näher eingegangen

²Das l dient nur zur Indexierung und nicht als Potenz

bezeichnet und die entsprechende gewünschte Ausgabe als l . Dabei ist wichtig zu beachten, dass y und l Vektoren sind. Zum Beispiel würde ein Bild, das 10x10 Pixel gross ist, als ein $(10 * 10 =)100$ -dimensionaler Vektor dargestellt werden, wobei jeder Eintrag im Vektor der Grauwert eines Pixels ist. Die Dimension vom Ausgabe Vektor y und des gewünschten Ausgabe Vektor l entspricht der Anzahl Neuronen in der letzten Schicht des Netzwerks, wobei jedes Neuron etwas Bestimmtes aussagt. Zum Beispiel könnte ein Neuron für das Vorkommen eines Autos im Bild stehen, wobei 0 für kein Auto und 1 für ein Auto steht.

$$C(w, b) = \sum_j (y_j - l_j)^2$$

Das Ziel des Netzwerkes ist diese Kostenfunktion zu minimieren, bis so viele Beispieldaten wie möglich $C \approx 0$ entsprechen. Dies geschieht, wenn die Ausgabe des Netzwerks und die gewünschte Ausgabe ähnlich ist.

2.4.2 Gradient Descent

Um diese Kostenfunktion zu minimieren wird ein Algorithmus namens *gradient descent* benutzt. Das Konzept basiert darauf, dass man eine Funktion, in Abhängigkeit einer Variablen ableiten kann und so die Steigung (eng. gradient) an diesem Punkt berechnen kann und die Variable Richtung Minimum anpasst.

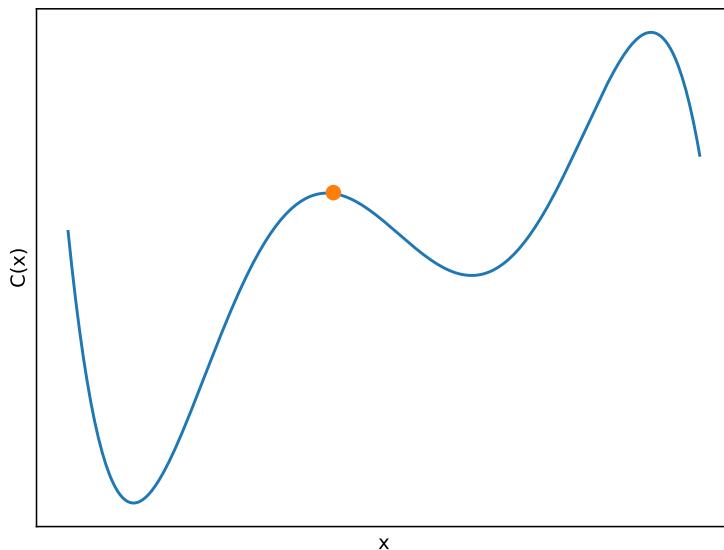


Abbildung 2.5: Kostenfunktion in Abhängigkeit von x

Zum Beispiel hat man eine Kostenfunktion $C(x)$, die von x abhängig ist (siehe Abbildung 2.5).

Die Variable x wird am Anfang einem zufälligem Wert zugewiesen, welcher dem orangen Punkt auf der Abbildung 2.5 entspricht. Das Ziel besteht darin, x so anzupassen, dass man ein Minimum der Kostenfunktion findet. Um ein Minimum zu finden kann man sich einen Ball vorstellen, der in ein Minimum herunterrollt. Um dies zu berechnen, muss man die Steigung

mithilfe einer Ableitung herausfinden und die Variable in die gegensätzliche Richtung bewegen.

$$a \rightarrow a' = a - \eta \frac{\partial C}{\partial a}$$

wobei η eine kleine positive Zahl (learning rate genannt) ist, die die Geschwindigkeit der Bewegung steuert. Außerdem ist zu beachten, dass der Ball keine Beschleunigung hat. Wenn man diese Gleichung iterativ anwendet, gelangt man früher oder später zum lokalen Minimum der Kostenfunktion (siehe Abbildung 2.6).

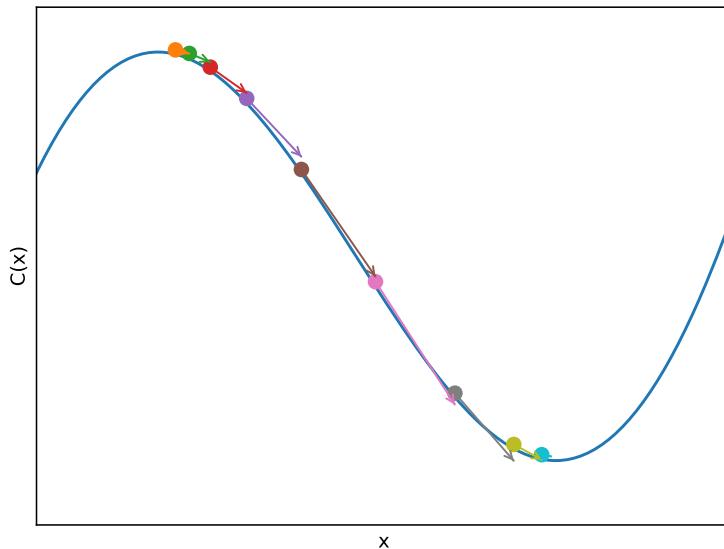


Abbildung 2.6: 2-dimensionaler Verlauf des gradient descent

Der Algorithmus funktioniert auch bei mehr als nur einer Variable und lässt sich für die Gewichte und Biases des Netzwerkes genau gleich berechnen.

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

Durch dieses Verfahren kann relativ einfach ein Minimum auf der Kostenfunktion gefunden werden. Dabei ist aber zu beachten, dass sich auf der Kostenfunktion mehrere Täler befinden können (siehe zum Beispiel Abbildung 2.5). Es besteht bei neuronalen Netzwerken deshalb die Gefahr, dass das gefundene lokale Minimum noch vom globalen Minimum entfernt ist und damit noch nicht das beste Ergebnis erzielt wurde.

2.4.3 Backpropagation

Den 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 mathematisch anspruchsvoll. Die genaue Herleitung ist im Rahmen dieser Arbeit aber nicht

nötig für das Verständnis eines neuronalen Netzwerkes.

Um die Übersicht zu behalten wird eine Zwischenmenge δ_j^l eingeführt, welche 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 der Fehler in der letzten Schicht L bestimmt werden:

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

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:

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

Es ist zu beachten, 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 jeder Fehler von jeder Schicht berechnet werden, indem man von hinten nach vorne durch das Netzwerk läuft.

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 ist:

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

2.5 Aktivierungsfunktionen

Wie schon beschrieben darf eine Aktivierungsfunktion nicht linear sein, da sie sonst nichts Neues zum Netzwerk beiträgt.

2.5.1 Sigmoid

Ein Beispiel für eine Aktivierungsfunktion ist die Sigmoid Funktion $f(x) = \frac{1}{1+e^{-x}}$ (siehe Abbildung 2.7). 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

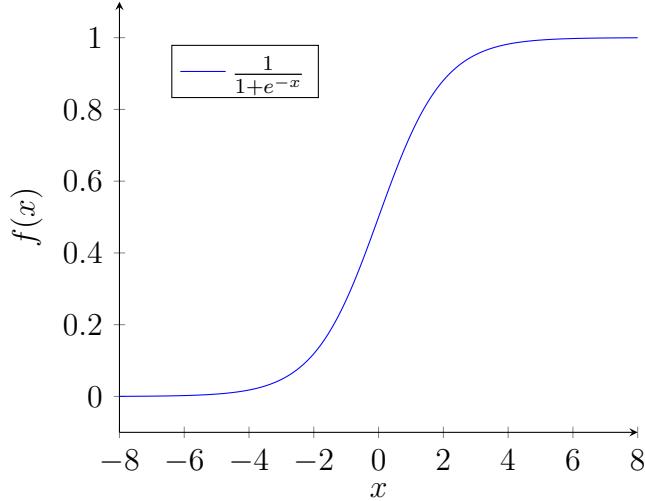


Abbildung 2.7: Sigmoid Aktivierungsfunktionen

der Wert zwischen $-\infty$ und ∞ liegt. Ein Problem der Sigmoid Funktion ist, dass bei einer Ausgabe nahe bei 1 oder 0 die jeweilige Ableitung $f'(x)$ nahe bei 0 ist, 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.

2.5.2 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 2.8) löst das Problem des vanishing

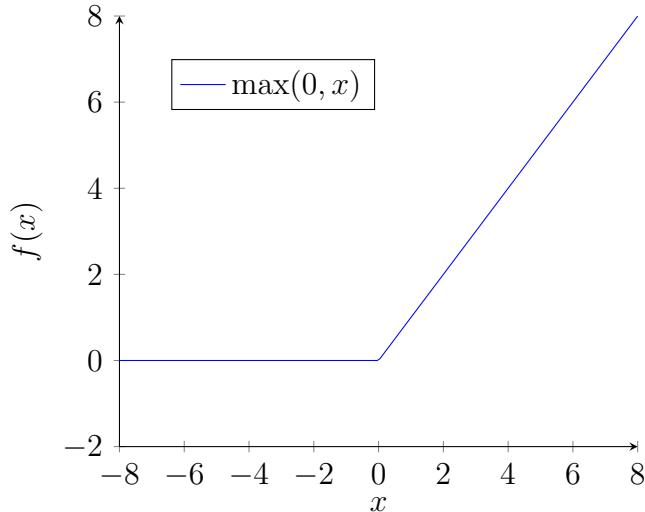


Abbildung 2.8: ReLu Aktivierungsfunktionen

gradient. Praktisch alle Neuronalen Netzwerke benutzen Relu als ihre Aktivierungsfunktion, da es die besten Ergebnisse erbringt[19]. Ein Nachteil ist, dass sie nur in den Versteckten Schichten gut funktioniert, da der Ausgabewert der Funktion unendlich gross sein kann und dadurch die Ausgabe vom Netzwerk nur schwer mit der gewünschten Ausgabe vergleichbar ist.

2.5.3 Softmax

Die softmax Funktion wird verwendet, um eindeutige Klassifikationen zu erreichen und ist definiert als:

$$f(x_j) = \frac{e^{x_j}}{\sum_k e^{x_k}}$$

Das besondere an dieser Aktivierungsfunktion ist, dass sie nicht nur einen Wert braucht, sondern alle Werte der ganzen Schicht, d.h nicht nur ein x_j sondern alle. Ausserdem gibt die Summe aller Resultate $\sum_j f(x_j) = 1$ 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 Ziffern, wo immer nur eine Ziffer pro Bild zu sehen ist. Durch die softmax Funktion sieht man dann eine geschätzte Wahrscheinlichkeit vom Netzwerk für jede Ziffer.

2.6 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. Ausserdem kann das gesuchte Objekt in einem Bild an verschiedenen Orten vorkommen.

2.6.1 Architektur

Die Eingabe für eine Convolution-Schicht ist nicht 1-dimensional, sondern 2-dimensional (siehe Abbildung 2.9). Die Neuronen werden normal verbunden einfach mit dem Unterschied, dass

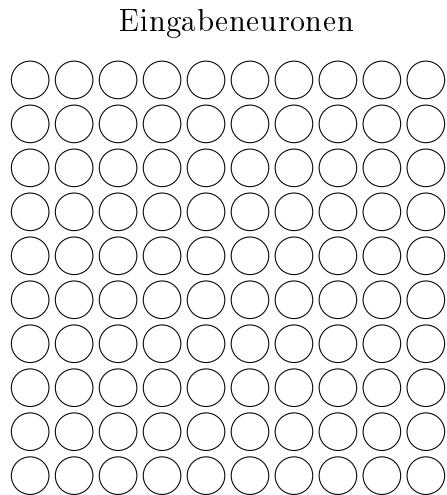


Abbildung 2.9: Eingabeneuronen für eine Convolution-Schicht

nicht jedes Neuron mit jedem Neuron verbunden wird, sondern dass nur ein bestimmter Bereich zum nächsten Neuron verbunden ist. Dieser Bereich wird als *Filter* bezeichnet.

In dem Beispiel auf Abbildung 2.10 wird ein 3x3 Filter benutzt. Der Filter wird dann auf

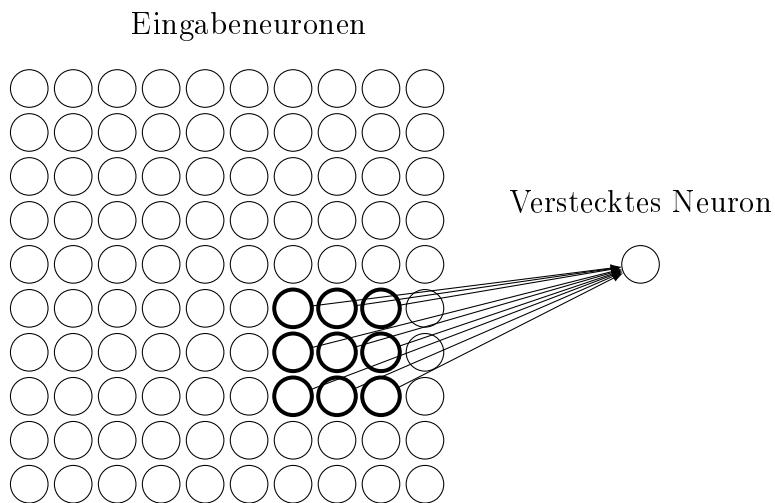


Abbildung 2.10: Verbindung eines versteckten Neurons in einer Convolution-Schicht

den Eingabeneuronen um ein Neuron verschoben, um den nächsten Neuron zu verbinden. Und so geht das weiter, auch nach unten, bis die ganze versteckte Schicht gemacht wurde. Dabei wird die versteckte Schicht auch kleiner (in dem Beispiel wird die 10x10 Schicht zu einer 8x8 Schicht), da der Filter irgendwann am anderen Rand anstößt. Abbildung 2.11 verdeutlicht das Prinzip noch einmal.

Der Filter kann auch um mehr als nur einen Neuronen verschoben werden und man kann in den beiden Richtungen verschiedene Schrittweiten nehmen, zum Beispiel bewegt sich der Neuron nach links um zwei Neuronen und nach unten um drei.

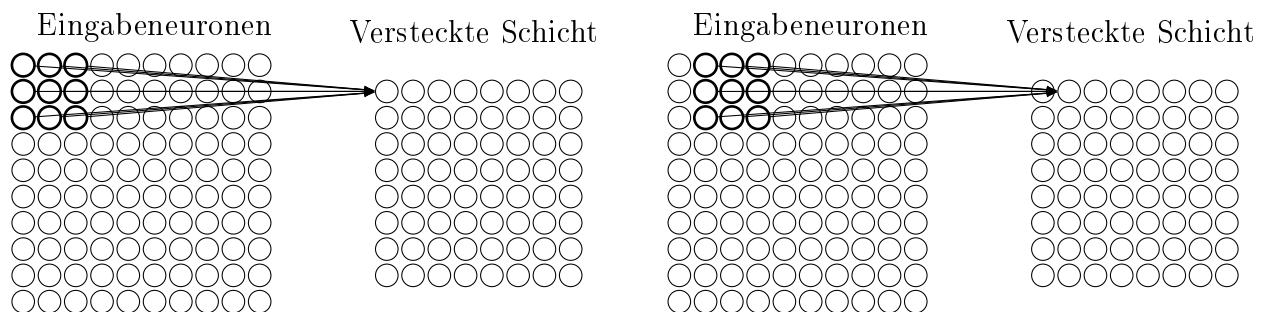


Abbildung 2.11: Bewegung eines Filters über eine Convolution-Schicht

Das Entscheidende am Filter ist, dass er die gleichen Gewichte und Bias verwendet für die Verbindung, d.h bei einem Filter von 5x5 gibt es $(5 * 5 =) 25$ verschiedene Gewichte und einen Bias. Wenn der Filter bewegt wird werden immer die gleichen Gewichte und der gleiche Bias verwendet. Als Gleichung bei einem 3x3 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 und f eine Aktivierungsfunktion.

Dadurch dass immer die gleichen Gewichte und der gleiche Bias für jeden Filter benutzt werden, wird überall das gleiche Merkmal erkannt, auch wenn es sich an einem anderen Ort

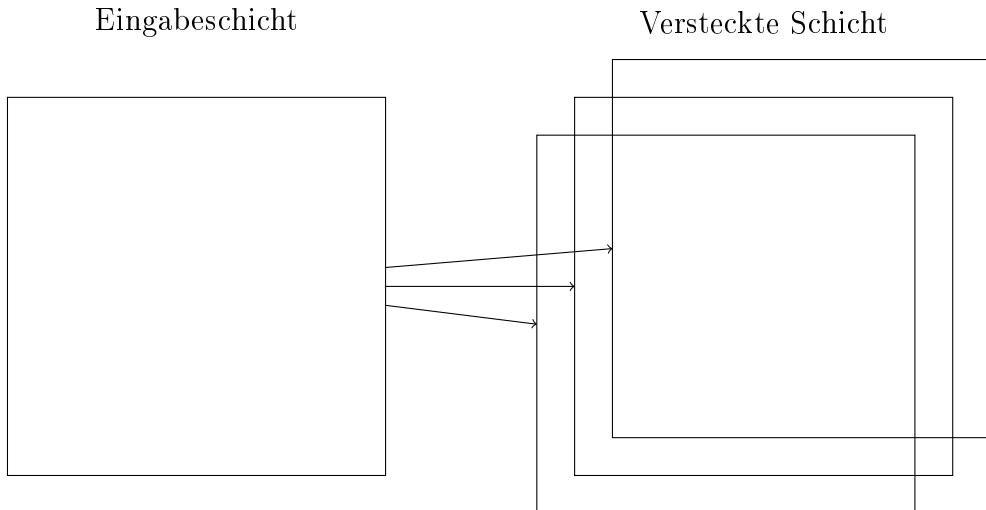


Abbildung 2.12: Convolution-Schicht mit 3 feature maps

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, wodurch mehrere feature maps entstehen (siehe Abbildung 2.12). Der Grund warum die Filter nicht alle das gleiche Merkmal erkennen, liegt an der zufälligen Initialisierung der Gewichte und Biases.

Falls nach einer Convolution-Schicht eine weitere Convolution-Schicht folgt, muss der Bereich des Filters zu allen feature maps erweitert werden[4] (siehe Abbildung 2.13). Man kann

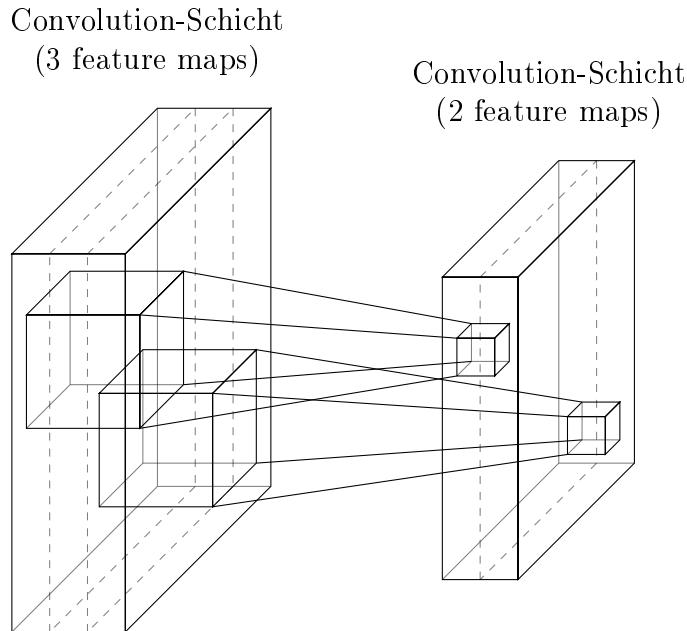


Abbildung 2.13: Zwei Convolution-Schichten hintereinander

es sich so vorstellen, als ob der Filter in der Z-Achse erweitert wird. Mit dieser Erweiterung kann auch die Eingabeschicht aus mehreren feature maps bestehen. Dadurch können auch relativ einfach farbige Bilder als Eingabe verwendet werden. Die Eingabeschicht würde aus drei feature maps bestehen, eine feature map für jeden Farbkanal des Bildes.

2.6.2 Max Pooling

Neben den Convolution-Schichten gibt es auch die Maxpool-Schicht³. Die Idee vom Pooling ist, dass die Informationen in einer Schicht zusammengefasst werden.

Max pooling nimmt eine feature map als Eingabe und lässt auch etwas Ähnliches wie ein Filter darüber laufen. Der Filter bewegt sich genau gleich wie ein normaler Filter bei der Convolution-Schicht, allerdings ist der Unterschied, dass er keine lernbaren Parameter hat und er die Ausgabe nicht wie ein Neuron berechnet, sondern die Ausgabe ist der grösste Wert von den Eingaben (siehe Abbildung 2.14). Ausserdem kann max pooling nur auf eine feature map angewendet werden, d.h der max pool Filter hat keine Z-Achse und muss auf jede feature maps einzeln angewendet werden[4]. Man kann max pooling verstehen als eine Reduzierung

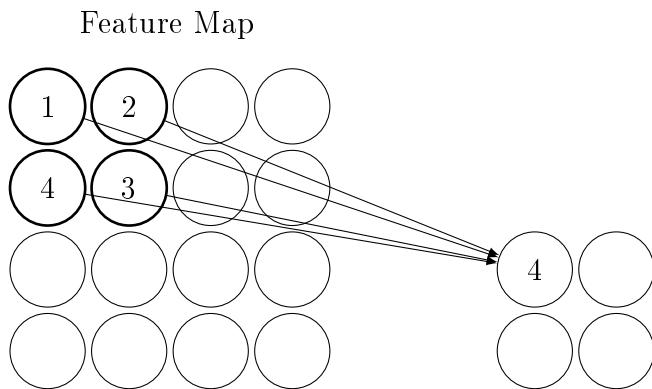


Abbildung 2.14: 2x2 Maxpool-Schicht mit einer Schrittweite von 2

der vorhanden Informationen. Es nimmt das wichtigste Merkmal in einem gewissen Bereich und wirft die weniger wichtigeren Merkmale weg, so dass es weniger Neuronen gibt und die darauffolgenden Neuronen es einfacher haben.

2.6.3 Convolution-Schichten und Fully-connected-Schichten

Um das Netzwerk zu interpretieren muss man eine Fully-connected-Schicht am Schluss haben, da man eine 2-dimensionale Schicht sonst nicht interpretieren kann.⁴ Die Fully-connected-Schicht wird angehängt, indem jedes Neuron von der Convolution-Schicht bzw. Maxpool-Schicht mit jedem Neuron der Fully-connected-Schichten verbunden wird. Es spielt keine Rolle, dass die Neuronen 2-dimensional angeordnet sind.

Ausserdem ist das Trainieren des Netzwerks immer noch genau gleich. Es wird immer noch gradient descent und Backpropagation benutzt, allerdings müssen die Gleichungen der Backpropagation für die Convolution und das max pooling angepasst werden.

³Es gibt noch andere Pooling-Schichten, die aber in dieser Arbeit nicht verwendet wurden

⁴Es gibt Netzwerke, bei denen es eine Convolution-Schicht als Ausgabeschicht hat[16]

2.7 Regularization

Um ein Netzwerk zu trainieren hat man meistens nur eine endliche Anzahl an Daten an denen das Netzwerk lernen kann. Aus dem Grund werden die gleichen Daten mehrmals zum Trainieren verwendet. Dadurch kann ein Problem entstehen: Mit der Zeit kennt das Netzwerk die Trainingsdaten so gut, dass es die Trainingsdaten einfach auswendig lernt und die Daten nicht mehr an ihren gemeinsamen Merkmalen und Zusammenhängen erkennt, sondern an ihren ganz spezifischen Merkmalen die nur für die Trainingsdaten zutreffen. Dadurch werden unbekannte Daten nicht mehr richtig erkannt. Dieses Phänomen ist als *overfitting* bekannt.

2.7.1 Dropout

Beim Trainieren eines Netzwerkes werden Neuronen mit einer bestimmten Wahrscheinlichkeit temporär deaktiviert bzw. ignoriert (siehe Abbildung 2.15). Bei jeder neuen Eingabe zum Trainieren werden neue zufällige Neuronen ausgewählt, dabei sind die Eingabeneuronen und Ausgabeneuronen davon ausgenommen.

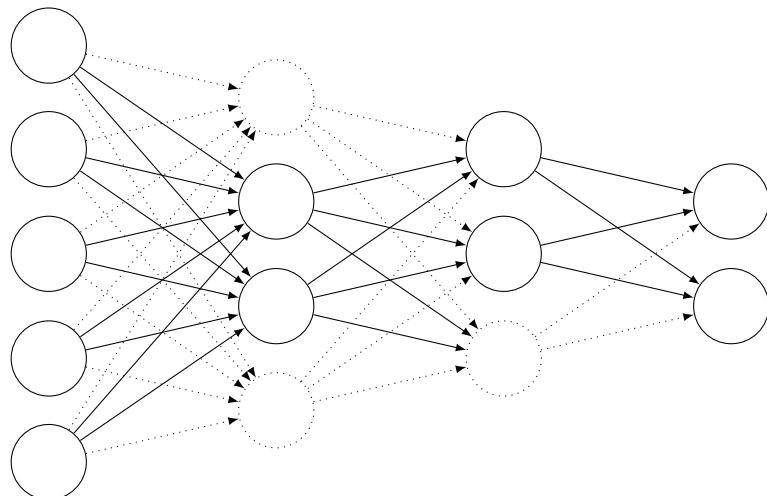


Abbildung 2.15: neuronales Netzwerk mit Dropout

Damit werden bestimmte Gewichte und Biases gelernt, welche davon ausgehen, dass immer ein Teil der Neuronen nicht vorhanden ist. Aber nach dem Training wird der Dropout nicht mehr benutzt und dadurch werden mehr Neuronen gleichzeitig aktiv sein als während dem Trainieren. Um das auszugleichen wird die Ausgabe des Neurons mit der Wahrscheinlichkeit, mit der es deaktiviert wird, multipliziert.

Dropout hilft gegen overfitting, da ein Neuron sich nicht auf andere Neuronen verlassen kann. Dadurch ist es gezwungen mit vielen zufälligen Verbindungen etwas Nützliches anzufangen. Anders gesagt: Das Netzwerk wird robust gegen den Verlust von einzelnen Merkmalen, da es sich nicht auf einzelne Merkmale verlassen kann.

2.7.2 Batch Normalization

Eine weitere Methode um overfitting zu vermeiden ist die *Batch Normalization*. Bei der Batch Normalization werden die Ausgaben der versteckten Schichten normalisiert[5]. so dass in den versteckten Schichten extreme Werte vermieden werden[5]. Ähnlich wie bei Dropout bringt Batch Normalization eine leichte Störung in das Netzwerk, welches gegen overfitting hilft[5].

Die Schichten werden normalisiert, indem beim Trainieren mehrere Eingaben gleichzeitig durch das Netzwerk laufen. Die Ausgaben eines Neurones werden mit dem Mittelwert der Ausgaben subtrahieren und mit der Standardabweichung der Ausgaben dividieren[5].

$$\begin{aligned}\mu &= \frac{1}{m} \sum_{i=1}^m x_i \\ \sigma &= \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2} \\ \hat{x}_i &= \frac{x_i - \mu}{\sigma}\end{aligned}$$

wobei x_i die i^{te} Ausgabe eines Neurons ist.

Ausserdem wird danach noch mit einem bestimmten Wert γ_i multipliziert und einem bestimmten Wert β_i addiert.

$$y_i = \gamma_i \cdot \hat{x}_i + \beta_i$$

wobei γ_i und β_i Parameter sind, die beim Netzwerk trainiert werden wie ein normales Gewicht oder Bias[5]. Die Parameter werden benutzt um dem Netzwerk die Option zu geben die Normalisierung zu verändern oder sogar rückgängig zu machen, wenn es meint, dass es anders besser funktioniert[10].

Kapitel 3

Lösungsansatz

3.1 Logo

Um das Logo zu erkennen muss man zuerst verstehen wie es auf das Bild gelangt. Man könnte meinen, dass das Logo einfach über das andere Bild gelegt wird. Aber das Logo wird eingespielt, indem es mit dem Bild negativ multipliziert wird, d.h das Bild und das Logo werden invertiert, multipliziert und dann wieder invertiert[20].

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

wobei a ein 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 3.1a,b). Eine andere Eigenschaft ist,

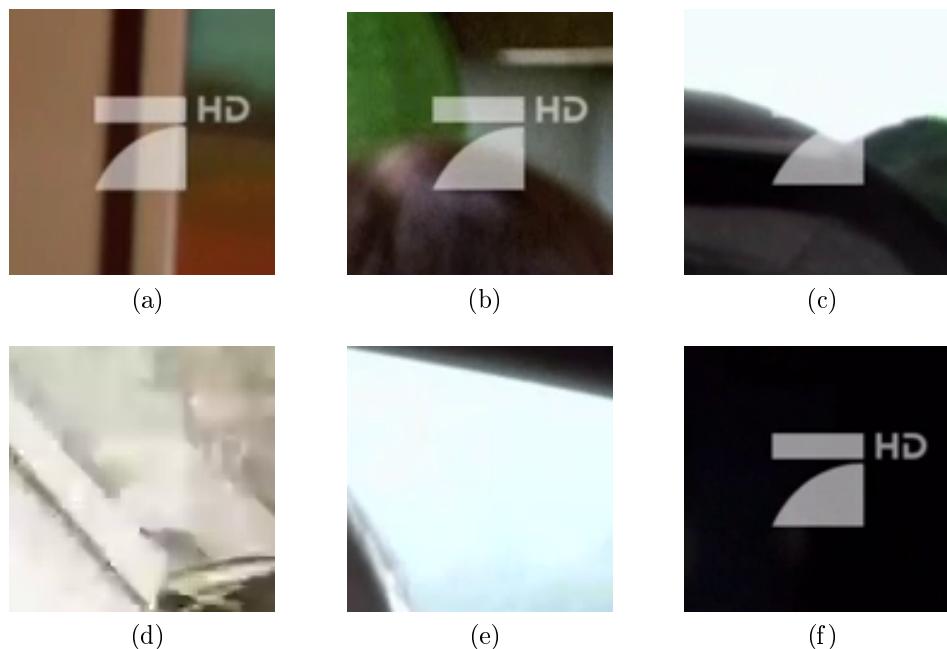


Abbildung 3.1: Logo mit verschiedenen Hintergründen

dass bei manchen Hintergründen, vor allem bei weissen, das Logo abgeschnitten oder kaum bis gar nicht sichtbar ist (siehe Abbildung 3.1c,d,e). 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 jeder Pixel der vorher weiss war, weiss bleibt und so das Logo nicht mehr erkennbar ist. Analog dazu, wenn man für $a = 0$ (schwarz) einsetzt, erhält man: $f(0, b) = b$, was bedeutet, dass sich das Logo bei einem schwarzen Hintergrund im ursprünglichen Zustand befindet. Aus dem Grund kann das Logo komplett herausfiltert und wieder verwendet werden, wenn das Logo auf einem schwarzen Hintergrund ist (siehe Abbildung 3.1f).

3.2 Werbung erkennen mithilfe des Logos

Um ein Netzwerk zu trainieren braucht es sehr viele Daten, die kategorisiert sind. Eine Möglichkeit, diese zu beschaffen, wäre die Bilder von Hand zu kategorisieren. Das Problem dabei ist, dass es eine sehr langwierige und sehr langweilige Arbeit wäre, da es Millionen von Bildern bräuchte. Die andere Möglichkeit ist die Bilder selber zu generieren, indem das Logo, welches auf einem schwarzen Hintergrund ist, auf viele unterschiedliche Bilder darauf multipliziert wird. Die unterschiedlichen Bilder erhält man aus dem Open Images Dataset V3[11], welches um die 9 Millionen Bilder enthält. Da diese Bilder des Open Images Dataset nicht in der richtigen Grösse vorhanden sind, werden sie in mehrere Bilder zerteilt, welche die richtige Grösse haben. Die Bilder könnten auch auf die richtige Grösse skaliert werden. Das Problem dabei ist, dass dadurch viel weniger Bilder pro Sekunden erzeugt werden können und dass es den ganzen Prozess des Lernens deutlich verlangsamen würde.

Das Logo kann nun mit dem Bild negativ multipliziert werden, so dass das Logo an einem zufälligen Ort auf dem Bild erscheint (siehe Abbildung 3.2). Man könnte sich auch überlegen, ob man das Logo nur an Positionen einspielt, wo es auch im Sender vorkommt. Aber für ein Convolution Netzwerk spielt es keine grosse Rolle und ausserdem hält es den Algorithmus allgemeiner. Die Grösse von einem Bild ist 320x180 Pixel, da ein grösseres Bild unnötig viele



Abbildung 3.2: Selbst generiertes Bild mit einem Prosieben Logo

Informationen enthält, was dazu führt, dass das Netzwerk länger lernen müsste. Ein kleineres Bild würde das Logo noch kleiner machen und kaum noch erkennbar (siehe Abbildung 3.3). Ausserdem werden die Bilder als schwarz-weiss Bilder dem Netzwerk gegeben, da man ohne Farben keinen grossen Verlust von wichtigen Informationen hat. Und die Farben würden die Trainingsdauer ziemlich erhöhen, da das Netzwerk viel mehr lernen müsste.



Abbildung 3.3: Prosieben Logo (17x11), extrahiert aus einem 320x180 Bild

3.3 Werbung erkennen ohne Hilfe des Logos

Um Werbung ohne Logo zu erkennen braucht es richtige Bilder vom Sender, da Bilder von Werbung nicht einfach generiert werden können. Da das Kategorisieren von Hand zu lange brauchen würde, wird das neuronale Netzwerk, das mithilfe des Logo die Werbung erkennt, benutzt, um die Bilder zu kategorisieren. Ein Problem dabei ist, dass das Netzwerk nicht perfekt ist. Um das auszugleichen wird die durchschnittliche¹ Vorhersage des Netzwerks genommen, da Werbung bzw. das normale Programm immer am Stück läuft.

Damit sichergestellt ist, dass das neue Netzwerk nicht wieder das Logo erkennt, sondern die Werbung, wird das Bild so zugeschnitten, dass das Logo nicht mehr auf dem Bild vorhanden ist.

Auch hier werden die Bilder als schwarz-weiss Bilder dem Netzwerk gegeben.

¹Mit Durchschnitt ist nicht der Mittelwert gemeint (siehe Abschnitt 5.3.1)

Kapitel 4

Umsetzung

Der ganze Code kann auf Github unter <https://github.com/GeorgOhneH/WerbeSkip> gefunden werden. Die verwendete Programmiersprache ist Python 3[17].

4.1 Neuronales Netzwerk

Der Code für diesen Teil befindet sich im Ordner *deepnet*.

Das neuronale Netzwerk ist objektorientiert programmiert, um leicht neue Schichten und Funktionen hinzuzufügen. Die Benutzerschnittstelle ist angelegt an Keras[3]¹. Als Grundbaustein wird NumPy[18]² benutzt. Da die Geschwindigkeit ein entscheidender Punkt ist, wurde alles als Matrizenmultiplikation implementiert.

4.1.1 Aufbau

Schichten Jeder Type von Schicht ist als eine Klasse implementiert, welche von einer Basisklasse erbt. Somit kann jede Schicht gleich behandelt werden. Jede Schicht muss eine Implementation von der Erstellung der Schicht aus der vorigen Schicht haben und die dazugehörige Backpropagation. Die Schichten sind unabhängig voneinander, bekommen aber immer die Ausgabe der vorigen Schicht bzw. der hintern Schicht bei der Backpropagation. Ausserdem sind die Aktivierungsfunktionen auch als Schicht implementiert.

Kostenfunktionen Jede Kostenfunktion ist auch eine Klasse und erbt auch von einer Basisklasse. Jede Kostenfunktion benötigt die Implementation der Funktion und deren Ableitung.

Optimierer Ein Optimierer enthält die Funktionen des gradient descent bzw. eine Variante davon. Gewisse Varianten des gradient descent können den Prozess des Lernens noch beschleunigen, z.B indem noch ein Impuls in den gradient descent miteinbezogen wird[15]. Auch der Optimierer ist eine Klasse und erbt von einer Basisklasse. Der Optimierer wird an jede Schicht weiter geleitet, um die Gewichte und Biases in der Schicht anzupassen.

¹eine Bibliothek für neuronale Netzwerke

²ein Bibliothek für wissenschaftliche Datenverarbeitung

Netzwerk Dies ist die Hauptklasse, es ist die Benutzerschnittstelle und steuert die Schichten, Kostenfunktionen und Optimierer. Ausserdem implementiert die Klasse noch diverse Funktionen, die zur Auswertung des neuronale Netzwerks hilfreich sind.

4.1.2 Benutzung

Beispiele können unter *deepnet/examples* gefunden werden.

Um das Programm zu benutzen müssen zuerst die Dimensionen der Eingabe bestimmen werden. Dies würde der Eingabeschicht von einem neuronalen Netzwerk entsprechen. Bei einem Netzwerk aus Fully-connected-Schichten ist es die Anzahl der Neuronen. Wenn es hingegen ein Convolution Netzwerk ist, dann bestehen die Eingabedimensionen aus drei Zahlen. Die erste Zahl ist die Anzahl feature map, welche der Anzahl Farbkanäle im einem Bild entsprächen würde. Die zweite ist die Höhe des Bildes und die dritte ist die Breite des Bildes. Danach müssen die verschiedenen Schichten bestimmt werden, darunter sind auch die Aktivierungsfunktionen. Als nächstes muss die Kostenfunktion und der Optimierer definiert werden. Am Schluss muss das neuronale Netzwerk noch trainiert werden, indem die Trainingsdaten dem Netzwerk gegeben werden. Die Trainingsdaten müssen ein NumPy array sein mit der gleichen Form wie die Eingabedimensionen mit dem Unterschied, dass der NumPy array an der ersten Stelle eine weiter Dimension hat, welcher die Trainingsdaten enthält. Die Trainingsdaten können auch als Generator³ übergeben werden, falls nicht genug Arbeitsspeicher vorhanden ist.

4.1.3 Grafikkartenunterstützung

Der Code kann unter *numpywrapper* gefunden werden.

Die Geschwindigkeit des Programmes spielt eine entscheidende Rolle für ein neuronales Netzwerk und die Geschwindigkeit der CPU⁴ ist nicht ausreichend. Um die GPU⁵ zu benutzen, wird Cupy verwendet. Da Cupy genau die gleichen Funktionen wie NumPy hat, kann es einfach mit NumPy ausgetauscht werden. Dazu wird ein selbst geschriebenes Modul verwendet, um einfach zwischen beiden hin und her zuschalten. Die GPU ist schneller als die CPU, da alles mithilfe von Matrizenmultiplikation implementiert ist und die GPU auf Matrizenmultiplikation optimiert ist. Das kann die Berechnungen auf meinem PC⁶ um das 7-fache beschleunigen.

4.1.4 Bildbearbeitung

Der Code kann unter *helperfunctions/image_processing* gefunden werden.

Der Ordner enthält die Funktionen für die Beschaffung und Formatierung der Bilder. Um

³Ein Generator übergibt die Daten in kleinen Portionen

⁴Central processing unit

⁵Grafikkarte

⁶Auf die Hardware des PC wird in Kapitel 5 näher eingegangen

die Bilder zu dekodieren wird OpenCV[2]⁷ und für die Formatierung der Bilder wird NumPy verwendet.

Die grösste Herausforderung war die Beschaffung der live stream Bilder von einem Sender. Um die Bilder des Senders zu erhalten, wird der Teleboy Stream angezapft, nach dem Beispiel von Github[9] und der Stream wird durch ffmpeg⁸ dekodiert.

4.2 Webserver

Der Code kann unter *src*, *app* und *vuedj* gefunden werden.

Um das fertige Programm zu benutzen wird eine Webseite verwendet, die unter der URL *www.werbeskip.com* erreichbar ist. Für das Backend wird Django[6]⁹ und Django Channels[7] verwendet, wodurch ein Websocket benutzt werden kann. So bleibt die Verbindung mit dem Server offen und die Seite ist immer auf dem aktuellen Stand.

Für das Frontend wird VueJs[23] und VuetifyJS[12] verwendet und die Webseite ist eine Single Page Application.

⁷Bibliothek von Funktionen um Bilder zu bearbeiten

⁸Programm für das Aufnehmen, Konvertieren und Streamen von Audio und Video

⁹Ein web Framework

Kapitel 5

Ergebnisse und Auswertung

Alle Berechnungen wurden auf einem PC mit einer Intel i5-4690 CPU (3.50GHz) und einer NVIDIA GeForce RTX 2070 GPU (8 GB) durchgeführt.

5.1 Auswertung

5.1.1 Datensatz

Um die Leistung der Netzwerke richtig auszuwerten wird ein selbst erstellter Datensatz verwendet, welcher aus insgesamt 7830 Prosieben Bildern besteht. Aus den 7830 Bildern sind 4495 Bilder mit Logo und ohne schwarzen Rand (siehe Abbildung 5.1a), 813 mit Logo und einem oberen und unteren schwarzen Rand (siehe Abbildung 5.1b), 813 mit Logo und einem schwarzen Rand auf beiden Seiten (siehe Abbildung 5.1c) und 2095 Bilder ohne Logo und ohne Rand (siehe Abbildung 5.1d). Ausserdem enthält der Datensatz noch 400 Bilder, die ein spezielles Prosieben Logo auf dem Bild haben (siehe Abbildung 5.1e). Diese sind normalerweise exkludiert, ausser sie werden explizit erwähnt.



(a) Normales Logo



(b) Horizontaler Rand



(c) Vertikaler Rand



(d) Ohne Logo



(e) Spezielles Logo

Abbildung 5.1: Verschiedene Arten von Bildern im Datensatz

5.1.2 Methoden

Loss: Die einfachste Methode das Netzwerk auszuwerten ist den Wert der Kostenfunktion anzuschauen, welcher als *Loss* bezeichnet wird. Je kleiner der Loss ist umso besser ist das Netzwerk.

Genauigkeit: Die Genauigkeit gibt Auskunft über den Anteil der richtig erkannten Bilder. Sie wird berechnet, indem die Anzahl richtig erkannter Bilder durch die totale Anzahl Bilder geteilt wird. Das Problem bei der Genauigkeit und beim Loss ist, dass sie nicht sehr aussagekräftig sind, sobald der Datensatz nicht ausgeglichen ist. Zum Beispiel hat man ein Datensatz von 100 Bildern, 90 von den Bildern haben ein Logo und 10 haben keins. Wenn jetzt ein neuronales Netzwerk immer sagt, dass ein Logo auf dem Bild ist, ergebe das eine Genauigkeit von 90%. Das sieht nach einem gutes Ergebnis aus, aber das Netzwerk ist im Grunde nutzlos, da es das Logo nicht erkennt, sondern immer nur die gleiche Ausgabe ausgibt.

Matthews correlation coefficient: Der Matthews correlation coefficient[21] behebt genau dieses Problem. MCC¹ unterscheidet nicht nur zwischen falschen und wahren Vorhersagen des Netzwerk, sondern auch zwischen wahr positiv, wahr negativ, falsch positiv und falsch negativ (Siehe Tabelle 5.1). Dadurch können verschiedene Datensätze miteinander verglichen werden, selbst wenn die Kategorien eine andere Verteilung haben[21].

		Wahrer Zustand	
		Zustand positiv	Zustand negativ
Vorausgesagt Bedingung	Vorhergesagter Zustand positiv	wahr positiv	falsch positiv
	Vorhergesagter Zustand negativ	falsch negativ	wahr negativ

Tabelle 5.1: Verwirrungsmatrix
Tabelle von Wikipedia[21]

MCC gibt einen Wert von -1 bis +1 zurück. Der Wert +1 repräsentiert eine perfekte Vorhersage, 0 nicht besser als eine zufällige und -1 eine komplette Unstimmigkeit zwischen Netzwerk und dem Datensatz. Der MCC wird nach folgender Formel berechnet[21]:

$$MCC = \frac{WP \cdot WN - FP \cdot FN}{\sqrt{(WP + FP)(WP + FN)(WN + FP)(WN + FN)}}$$

wobei WP die Anzahl der wahr positiven ist, WN die Anzahl der wahr negativen, FP die Anzahl der falsch positiven und FN die Anzahl der falsch negativen.

¹Matthews correlation coefficient

5.2 Neuronales Netzwerk mithilfe des Logos

5.2.1 Architektur

Die gewählte Architektur orientiert sich am AlexNet[8]. Es benutzt wie das AlexNet fünf Convolution-Schichten und benutzt nach jeder Convolution- und Fully-connected-Schicht die ReLU Aktivierungsfunktion. Maxpooling wird weniger häufig angewendet als beim AlexNet, da es heute nicht mehr so üblich ist[4] und es werden relativ kleine Werte für Dropout genommen, da genug Daten vorhanden sind und dadurch overfitting kein grosses Problem darstellt. Ausserdem wird in jeder Schicht Batch Normalization verwendet nach dem Beispiel vom ResNet[4].

Die letzte Schicht beinhaltet zwei Neuronen und die softmax Aktivierungsfunktion. Das erste Neuron steht für das Fehlen des Logos, wobei 0 für falsch steht und 1 für wahr. Analog dazu steht das zweite Neuron für das Bestehen des Logos. Durch diese zwei Neuronen und der softmax Aktivierungsfunktion kann die Ausgabe als Wahrscheinlichkeitsverteilung verstanden werden.

Das neuronale Netzwerk wurde mithilfe des *Adam* Optimierers[14] und einer learning rate von 0.001 trainiert. Der Adam Optimierer wird benutzt, da er die learning rate adaptiv anpasst und er als einer der besseren Optimierer gilt[14]. Die learning rate sollte am Anfang vom Training relativ gross sein und am Schluss eher klein, da man am Anfang schnell in die Nähe eines Minimums kommen will und am Schluss will man nicht über das Minimum springen. Theoretisch könnte während dem Training die learning rate manuel angepasst werden, aber der Optimierer macht dies oft einfacher und besser.

Für die Kostenfunktion wird nicht wie im Kapitel 2 die quadratische genommen, sondern die *cross-entropy*[1], welche grundsätzlich gleich funktioniert wie die quadratische, mit dem Unterschied, dass es die Implementierung mathematisch vereinfacht[1].

Genaue Architektur

1. Convolution, Anzahl Filter: 16, Filterbreite: 12, Filterhöhe: 8, Schrittweite: 4
2. BatchNorm
3. ReLU
4. Convolution, Anzahl Filter: 64, Filterbreite: 6, Filterhöhe: 4, Schrittweite: 1
5. BatchNorm
6. ReLU
7. Maxpool, Filterbreite: 3 Filterhöhe: 3, Schrittweite 2
8. Convolution, Anzahl Filter: 128, Filterbreite: 4, Filterhöhe: 4, Schrittweite: 1
9. BatchNorm
10. ReLU
11. Convolution, Anzahl Filter: 256, Filterbreite: 3, Filterhöhe: 3, Schrittweite: 2
12. BatchNorm
13. ReLU
14. Convolution, Anzahl Filter: 512, Filterbreite: 3, Filterhöhe: 3, Schrittweite: 1

15. BatchNorm
16. Dropout 20%
17. Fully-connected-Schicht, Neuronen: 2048
18. BatchNorm
19. ReLU
20. Dropout 20%
21. Fully-connected-Schicht, Neuronen: 2
22. Softmax

5.2.2 Training

Das Training des Netzwerks wurde nach ungefähr 17 Stunden abgebrochen, da es sich kaum noch verbessert hat. Insgesamt wurden 2'540'000 verschiedene Bilder verwendet. Abbildung 5.2 zeigt den durchschnittlichen Loss und MCC des Netzwerks im Verlaufe des Trainings.

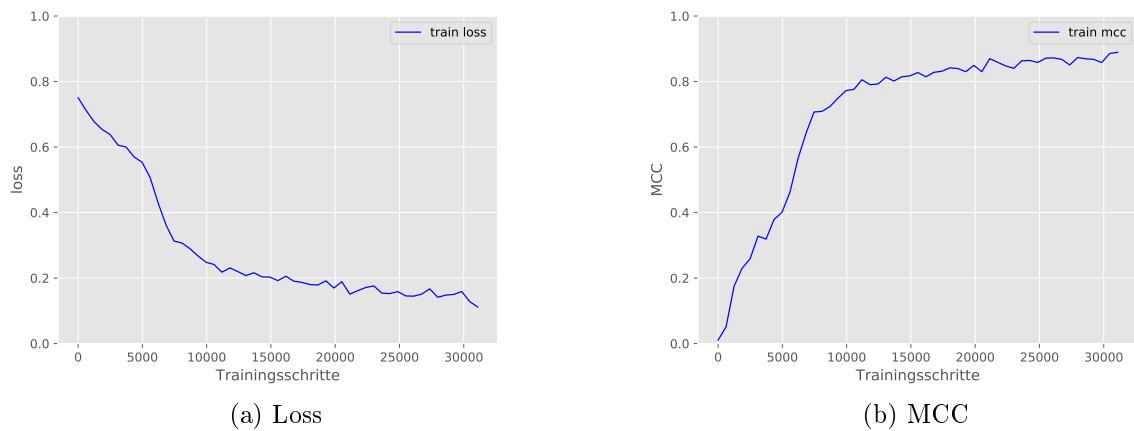


Abbildung 5.2: Loss und MCC im Verlaufe des Trainings

5.2.3 Auswertung

Datensatz

Wenn man 161'400 selbst generierte Bilder mit dem trainierten Netzwerk kategorisiert, dann erhält man durchschnittlich einen Loss von 0.155, eine Genauigkeit von 93.1% und ein MCC von 86.7%. Mit dem selbst erstellten Datensatz, der in Abschnitt 5.1.1 erwähnt wurde, erhält man einen Loss von 0.22, eine Genauigkeit von 91.6% und einen MCC von 80.2%. Der Loss und die Genauigkeit des Datensatzes und der selbst generierten Bilder sind nicht miteinander vergleichbar, da die Verteilung der Bilder anders ist. Die selbst generierten Bilder haben zu 50% kein Logo und zu 50% ein Logo. Der Datensatz hingegen hat zu 27% kein Logo und zu 73% ein Logo.

Der Datensatz schneidet beim MCC um 6.5% schlechter ab als bei den selbst generierten Bildern. Das könnte daran liegen, dass im Fernsehen oft noch neben dem Logo auch andere

Einblendungen sind (siehe Abbildung 5.3) und das Netzwerk diese Art von Bildern noch nie gesehen hat. Aber es ist nicht klar, ob es an den Einblendungen liegt. Es könnte genau so gut



Abbildung 5.3: Einblendungen im Fernsehen

daran liegen, dass im Fernsehen öfters weisse Bilder vorkommen. Das Netzwerk kann die Logos mit weissem Hintergrund schwerer kategorisieren. Genau das ist ein grosses Problem bei einem neuronalen Netzwerk. Eigentlich hat man keine Ahnung wie das Netzwerk genau funktioniert und wie es zu seinen Resultaten kommt.

Spezielles Logo

Wenn man von dem Datensatz nur die Bilder ohne Logo und die Bilder mit dem speziellen Logo (siehe Abbildung 5.4a) auswertet, erhält man einen MCC von 78.2%, welcher nur 2.0% unter dem Wert von den normalen Bildern ist. Das ist ziemlich überraschend, da das Netzwerk diese Art von Logo noch nie gesehen hat und trotzdem so ein gutes Resultat erzielt.

Das spezielle Logo ist fast komplett weiss und ein bisschen kleiner als das normale. Man könnte nun denken, dass das Netzwerk nicht unbedingt das Logo erkennt, sondern nur eine kleine weisse Stelle auf dem Bild (siehe Abbildung 5.4b).



(a) generiertes Bild mit dem speziellen Prosieben Logo



(b) generiertes Bild mit einer weissen Stelle (8x9)

Abbildung 5.4: Vergleich von speziellen Logo und einer weissen Stelle

Wenn man generierte Bildern mit dem speziellen Logo (siehe Abbildung 5.4a) durch das Netzwerk laufen lässt (natürlich auch mit Bildern ohne Logo), erhält man mit insgesamt 207'900 Bildern einen MCC von 87.8%, welcher gleich gut ist, wie wenn das normale Logo benutzt wird. Mit generierten Bildern, auf denen eine kleine weisse Stelle ist (8x9 Pixel, die Grösse des speziellen Logos) erhält man mit insgesamt 214'000 Bildern einen MCC von 68.7%. Dieser Wert

ist 19.1% schlechter als wenn das spezielle Logo auf den generierten Bildern verwendet wird. Das zeigt sehr schön, dass nicht nur die Helligkeit der Stelle entscheidend ist, sondern auch andere Faktoren, wie zum Beispiel die Form.

Wenn man eine weisse Stelle mit der Grösse von 8x8 Pixeln durch das Netzwerk laufen lässt, erhält man auffallenderweise mit insgesamt 160'500 Bildern einen MCC von 25.9%, welcher massiv schlechter ist als bei der Grösse von 8x9 Pixeln. Es zeigt, dass eine weisse Stelle eine Mindestgrösse haben muss, damit es für ein Logo gehalten werden kann.

5.3 Neuronales Netzwerk ohne das Logo

5.3.1 Datenbeschaffung

Um die Bilder von Prosieben zu kategorisieren wird das vorige neuronale Netzwerk, welches das Logo erkennen kann, verwendet. Die Abbildung 5.5 zeigt die Wahrscheinlichkeit der Vorhersage des Netzwerks von live Prosieben Bildern von einem gewissen Zeitraum. Die Vorhersage ist ein Wert zwischen 0 und 1, wobei 1 für das Bestehen des Logos steht und 0 für das Fehlen des Logos. Jeder Punkt repräsentiert eine Vorhersage. Die Zeit ist in Sekunden angegeben und pro Sekunde hat es eine Vorhersage. Auffällig dabei ist, dass das Netzwerk viel sicherer ist, wenn

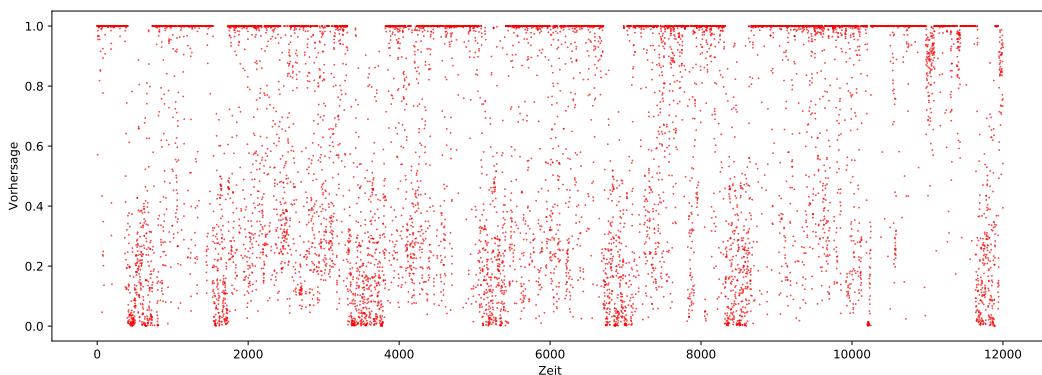


Abbildung 5.5: Vorhersagen-Zeit Diagramm

es ein Logo entdeckt, als wenn es keines findet. Das macht auch Sinn, denn wenn es keines gefunden hat, könnte das daran liegen, dass es keines gibt oder dass das Logo nicht sichtbar ist und es eigentlich eines hat. Dadurch kann man nur selten zu 100% sicher sein, dass kein Logo auf dem Bild vorhanden ist.

Auf dem Diagramm können die Stellen, wo Werbung lief ziemlich einfach erkannt werden. Sie liegen nämlich dort, wo die obere Punktmenge, die fast wie ein Strich ausschaut, für eine längere Zeit verschwindet. Der Algorithmus macht sich dies zunutze, um diese Sequenz von Bildern zu kategorisieren. Es wird geschaut, ob in den letzten 25 Sekunden eine Vorhersage über 0.9 gewesen ist. Wenn dies 5 mal hintereinander zutrifft, wird das Bild als keine Werbung kategorisiert. Analog dazu müssen 5 mal hintereinander die letzten 25 Sekunden die Vorhersage unter 0.9 sein, damit es als Werbung kategorisiert wird. Falls beides nicht zutreffen sollte, wird

das Bild gleich wie das vorige Bild kategorisiert. Abbildung 5.6 zeigt das Resultat des Algorithmus². Die Kategorisierung ist nicht perfekt, aber gut genug, um das Netzwerk zu trainieren.

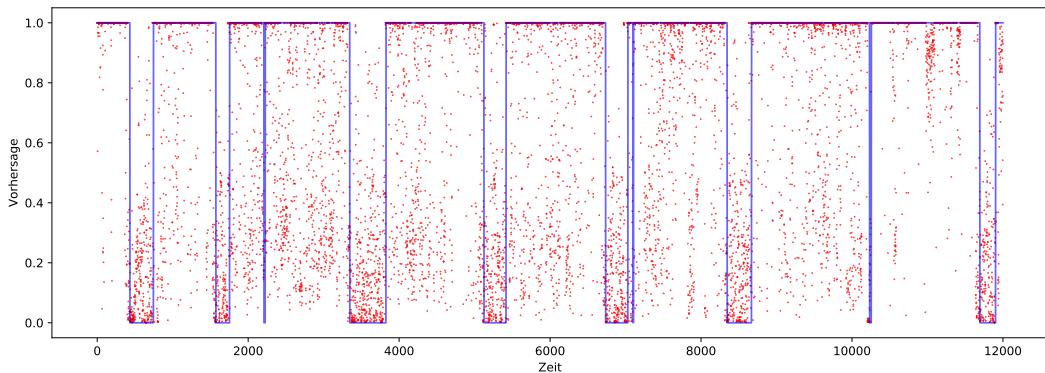


Abbildung 5.6: Vorhersagen-Zeit Diagramm mit Kategorisierung

Der kleine Teil, der falsch kategorisiert ist, hat keinen grossen Einfluss auf den Trainingsprozess.

Ausserdem werden die Bilder noch zugeschnitten, so dass das Logo nicht mehr auf dem Bild vorkommt. Die Bilder haben schliesslich eine Grösse von 269x141 Pixel (siehe Abbildung 5.7).



Abbildung 5.7: Zugeschnittenes Bild (269x141)

5.3.2 Architektur

Es wird die gleiche Architektur benutzt wie für das vorige Netzwerk, welches das Logo erkennt, allerdings mit dem Unterschied, dass mehr Dropout verwendet wurde, da Bilder mehrmals verwendet werden. Ausserdem sind die Parameter des Filters angepasst, da die Bilder eine andere Grösse haben und die gleichen Parameter nicht mehr funktionieren würden.

Genau Architektur

1. Convolution, Anzahl Filter: 16, Filterbreite: 9, Filterhöhe: 9, Schrittweite: 4
2. BatchNorm
3. ReLU
4. Convolution, Anzahl Filter: 64, Filterbreite: 6, Filterhöhe: 6, Schrittweite: 1
5. BatchNorm

²Dieser Algorithmus wird auch von dem Webserver verwendet

6. ReLU
7. Maxpool, Filterbreite: 3 Filterhöhe: 3, Schrittweite 2
8. Convolution, Anzahl Filter: 128, Filterbreite: 4, Filterhöhe: 4, Schrittweite: 1
9. BatchNorm
10. ReLU
11. Convolution, Anzahl Filter: 256, Filterbreite: 3, Filterhöhe: 3, Schrittweite: 2
12. BatchNorm
13. ReLU
14. Convolution, Anzahl Filter: 512, Filterbreite: 3, Filterhöhe: 3, Schrittweite: 1
15. BatchNorm
16. Dropout 25%
17. Fully-connected-Schicht, Neuronen: 2048
18. BatchNorm
19. ReLU
20. Dropout 50%
21. Fully-connected-Schicht, Neuronen: 2
22. Softmax

5.3.3 Training

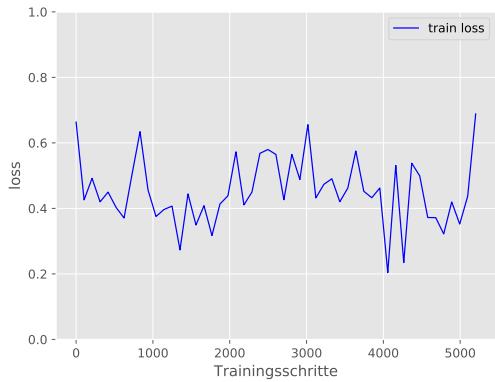
Das Training des Netzwerks wurde nach ungefähr 43 Stunden abgebrochen, da es sich nicht verbessert hat. Insgesamt wurden 339'200 Bilder durch das Netzwerk geschleust, wobei auch gleiche Bilder mehrmals verwendet worden sind. Dieser Trainingsprozess dauerte länger und hat weniger Bilder verarbeitet, als das Netzwerk, welches das Logo erkennen kann. Das liegt daran, dass nicht die GPU die Geschwindigkeit den Trainingsprozess begrenzt, sondern die Beschaffung der Daten. Aus einem Livestream können nur eine begrenzte Anzahl Bilder pro Sekunde extrahiert werden und dadurch verlangsamt sich den Prozess erheblich.

Abbildung 5.8a,b zeigt den durchschnittlichen Loss und MCC, des Netzwerk im Verlaufe des Trainings.

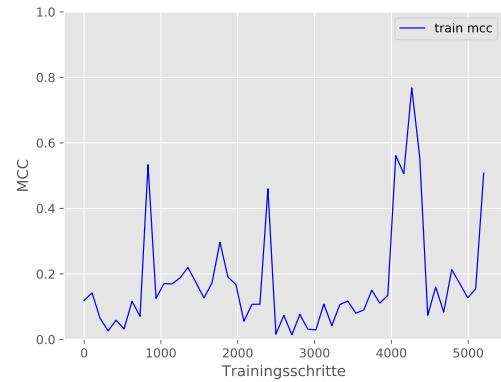
5.3.4 Auswertung

An Abbildung 5.8a,b erkennt man, dass sich der Loss und der MCC während dem Training nicht positiv verändert haben und das Netzwerk nichts viel gelernt hat. Auch die Auswertung des Datensatzes (mit den speziellen Logos) bestätigt, dass das Netzwerk nicht viel gelernt hat. Man erhält einen Loss von 1.354, eine Genauigkeit von 37.2% und einen MCC von 16.4%, welche alle sehr schlechte Werte sind. Ein zufälliges initialisiertes Netzwerk erzielt einen Loss von 1.287, eine Genauigkeit von 41.3% und einen MCC von 15.8%. Das zeigt, dass das trainierte Netzwerk genauso schlecht ist wie ein zufällige initialisiertes Netzwerk. Man kommt zum Schluss, dass das Netzwerk in keiner Weise fähig ist, die Werbung zu erkennen.

Es könnte natürlich sein, dass das Netzwerk eine kompliziertere Architektur haben muss



(a) Loss



(b) MCC

Abbildung 5.8: Loss und MCC im Verlaufe des Trainings

und mehr Daten braucht um ein bessere Ergebnisse zu erbringen. Da aber für diese Arbeit nur sehr begrenzt Ressourcen zur Verfügung stehen, kann dies nicht weiter getestet werden.

Kapitel 6

Fazit und Weiterführung

6.1 Fazit

Das Ziel in dieser Arbeit war zu überprüfen, ob die Erkennung von Werbung mithilfe eines neuronalen Netzwerks möglich ist. Das Ziel wurden nur indirekt erreicht.

Es war ohne die Berücksichtigung des Logos nicht möglich einem Convolution Netzwerk beizubringen wie Werbung ausschaut und an welchen Kriterien man die Werbung erkennen könnte. Das Misslingen könnte daran liegen, dass Werbung zu ähnlich zum normalen Programm ist oder dass das Netzwerk viel komplizierter sein muss und viel mehr Daten braucht.

Hingegen war die Erkennung von Werbung des Senders Prosieben mithilfe des Logos ein Erfolg. Es war möglich mit einem Convolution Netzwerk das Prosieben Logo in jeder beliebigen Position auf einem schwarz-weissen Bild zu erkennen. Da das Prosieben Logo nur eingespielt wird, wenn keine Werbung läuft, kann ziemlich exakt bestimmt werden, ob Werbung läuft oder nicht.

6.2 Mögliche Weiterführung

In dieser Arbeit wurde nur ein winziger Bruchteil von Methoden für ein neuronales Netzwerk präsentiert. Die Resultate, die in der Arbeit gezeigt worden sind, können sicher noch verbessert werden. Die Netzwerke könnten mit neueren Methoden und mehr Rechenleistung verbessert werden.

Eine denkbare Weiterführung für das Netzwerk wäre, dass es nicht nur eine Art von Logo erkennen kann, sondern auch noch andere Logos. Zum Beispiel könnte dann ein Netzwerk neben dem Prosieben Logo auch das SRF Logo erkennen.

Ob Werbung in diesem Moment läuft wird bestimmt, indem das Netzwerk jede Sekunde das aktuellen live Prosieben Bild auswertet und dann von diesen Daten der Durchschnitt genommen wird. Man könnte nun anstatt selber den Durchschnitt zu berechnen, die Daten einem Recurrent Netzwerk geben, das dann den Durchschnitt berechnet. Ein Recurrent Netzwerk hat einen feedback loop, welches ihm erlaubt einen Speicher zu haben[22]. Dadurch können auch wenn die Daten immer nur einzeln gegeben werden, die vorigen Daten miteinbezogen werden. Wenn

man ein Recurrent Netzwerk benutzen würde, müsste man nicht mehr selber den Durchschnitt berechnen, sondern die Aufgabe würde eine Recurrent Netzwerk übernehmen, welches dann auch eine bessere Lösung finden würde.

Literaturverzeichnis

- [1] Eli Bendersky. The softmax function and its derivative. <https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative>, 2016. [Online; accessed 28. November 2018].
- [2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [3] François Chollet et al. Keras. <https://keras.io>, 2015.
- [4] cs231n. Convolutional neural networks (cnns / convnets). <http://cs231n.github.io/convolutional-networks>, 2018. [Online; accessed 28. November 2018].
- [5] Firdaouss Doukkali. Batch normalization in neural networks. <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c>, 2017. [Online; accessed 28. November 2018].
- [6] Django Software Foundation. Django [computer software]. [https://django-project.com](https://.djangoproject.com), 2013. [Online; accessed 28. November 2018].
- [7] Django Software Foundation. Django [computer software]. <https://channels.readthedocs.io>, 2016. [Online; accessed 28. November 2018].
- [8] Hao Gao. A walk-through of alexnet. <https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637>, 2017. [Online; accessed 28. November 2018].
- [9] Roman Haefeli. watchteleboy. <https://github.com/reduzent>, 2018. [Online; accessed 28. November 2018].
- [10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [11] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2017.

- [12] John Leider et al. Material design component framework. <https://vuetifyjs.com>, 2016. [Online; accessed 28. November 2018].
- [13] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. [Online; accessed 28. November 2018].
- [14] S. Ruder. An overview of gradient descent optimization algorithms. *ArXiv e-prints*, September 2016.
- [15] Sebastian Ruder. An overview of gradient descent optimization algorithms. <http://ruder.io/optimizing-gradient-descent>, 2016. [Online; accessed 28. November 2018].
- [16] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. *ArXiv e-prints*, December 2014.
- [17] Python Core Team. Python: A dynamic, open source programming language. <https://www.python.org>, 2015. [Online; accessed 28. November 2018].
- [18] Oliphant Travis E. A guide to numpy. <http://www.numpy.org>, 2006. [Online; accessed 28. November 2018].
- [19] 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 28. November 2018].
- [20] Wikipedia contributors. Blend modes — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Blend_modes, 2018. [Online; accessed 28. November 2018].
- [21] Wikipedia contributors. Matthews correlation coefficient — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Matthews_correlation_coefficient, 2018. [Online; accessed 28. November 2018].
- [22] Wikipedia contributors. Recurrent neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Recurrent_neural_network, 2018. [Online; accessed 28. November 2018].
- [23] Evan You et al. The progressive javascript framework. <https://vuejs.org>, 2014. [Online; accessed 28. November 2018].

Abbildungsverzeichnis

1.1	Kein Senderlogo bei Werbung	4
1.2	Logo mit verschiedenen Hintergründen	4
1.3	Logo an verschiedenen Positionen	5
2.1	Einzelner Neuron in einem Neuronalen Netzwerks	7
2.2	Mögliche Architektur eines neuronalen Netzwerk	8
2.3	Neuronales Netzwerk mit 2 versteckten Schichten	8
2.4	Bezeichnung der Parameter	9
2.5	Kostenfunktion in Abhängigkeit von x	10
2.6	2-dimensionaler Verlauf des gradient descent	11
2.7	Sigmoid Aktivierungsfunktionen	13
2.8	ReLU Aktivierungsfunktionen	13
2.9	Eingabeneuronen für eine Convolution-Schicht	14
2.10	Verbindung eines versteckten Neurons in einer Convolution-Schicht	15
2.11	Bewegung eines Filters über eine Convolution-Schicht	15
2.12	Convolution-Schicht mit 3 feature maps	16
2.13	Zwei Convolution-Schichten hintereinander	16
2.14	2x2 Maxpool-Schicht mit einer Schrittweite von 2	17
2.15	neuronales Netzwerk mit Dropout	18
3.1	Logo mit verschiedenen Hintergründen	20
3.2	Selbst generiertes Bild mit einem Prosieben Logo	21
3.3	Prosieben Logo (17x11), extrahiert aus einem 320x180 Bild	22
5.1	Verschiedene Arten von Bildern im Datensatz	26
5.2	Loss und MCC im Verlaufe des Trainings	29
5.3	Einblendungen im Fernsehen	30
5.4	Vergleich von speziellen Logo und einer weissen Stelle	30
5.5	Vorhersagen-Zeit Diagramm	31
5.6	Vorhersagen-Zeit Diagramm mit Kategorisierung	32
5.7	Zugeschnittenes Bild (269x141)	32
5.8	Loss und MCC im Verlaufe des Trainings	34

Abbildungen aus dem Kapitel 2 sind von Michael A. Nielsen's Buch[13] inspiriert.

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.

Basel, 28. November 2018