Auswirkungen eines Hintergrundes auf die Identifikation von Objekten in Bildern mithilfe eines Convolutional Neural Network

Alexandra Zarkh, Sui Yin Zhang,
Lennart Leggewie und Alexander Schallenberg

Hochschule Bonn-Rhein-Sieg, Fachbereich Informatik, D-53757

8. Januar 2022

Inhaltsverzeichnis

1 Zusammenfassung

Im folgenden Text werden Sie ein Convolutional Neural Network, oder kurz CNN, welches in Java programmiert wurde, im Detail kennenlernen. Außerdem wird diskutiert, inwieweit sich Hintergründe auf die Identifikation von Objekten in Bildern mithilfe eines Convolutional Neural Networks auswirken. Im ersten Teil des Textes wird die verwendete Literatur, die Theorie und die Fragestellung vorgestellt. Anschließend folgen die Methoden. Unter anderem wird hier erklärt, was ein Convolutional Neural Network ist und wie es hier implementiert wurde. Hier werden im Detail dann die Klassen *ConvNet*, *ImageAdapter*, *TrainData* und die genutzten Testdateien vorgestellt. Der letzte Teil des Abschnitts Methoden präsentiert dann die durchgeführten Test. Zum Schluss werden die Ergebnisse nochmal aufgeführt und zusammengefasst. Genutzte Grafiken und Tabellen findet man ebenfalls dort wieder. Daraufhin wird die Fragestellung des Berichts nochmal aufgegriffen und diskutiert. In den letzten beiden Abschnitten findet man sowohl die Literatur als auch den Anhang.

2 Einleitung

Die Gesichtserkennung am Smartphone ermöglicht das Entsperren eines Gerätes und auf verschiedenen Applikationen die Nutzung von Filtern, die das Gesicht und die Umgebung erkennen und entweder die Umgebung oder bestimmte Eigenschaften und Merkmale in Gesichtern selber verändern. Das Autofahren wird immer einfacher und komfortabler für die Person am Steuer und das hat man unter anderem der Schilder- und Fahrbahnlinienerkennung zu verdanken. Jetzt stellt sich die Frage: wie funktioniert das? Die Antwort ist ein Convolutional Neural Network.

2.1 Verwendete Literatur

2.2 Theorie

Bedauerlicherweise treten einige Probleme bei der Verkehrszeichenerkennung auf, die nicht direkt vom Convolutional Neural Network selber ausgehen, sondern von der Umgebung. Wie zum Beispiel Schilder, die schwer zu erkennen sind, weil sie entweder schon sehr alt und dreckig sind oder zugeklebt wurden. Das erschwert dem CNN die Erkennung des Schildes und kann zu Irrtümern führen, die dann im schlimmsten Fall einen Unfall verursachen können. Ein weiteres Problem könnte der Verkehrsschilderwald in der Stadt sein. Dann wird es für das CNN schwerer, sich das gewollte Verkehrsschild auszusuchen, da mittlerweile sehr viele Schilder an einem Ort angebracht sind. Schwieriger wird es auch für das Convolutional Neural Network ein Verkehrsschild zu erkennen, wenn der Hintergrund

sehr farbig ist und mit den Farben des Verkehrsschildes ähnelt. Dann könnte es schwierig werden, das Bild zurecht zuschneiden und die Ränder des Schildes zu erkennen.

2.3 Fragestellung

Die Leitfrage ist nun: Welche Auswirkungen hat ein Hintergrund eines Bildes auf die Kosten der Kalkulation eines Convolutional Neural Network (CNN), welches ein Objekt im Vordergrund des Bildes erkennen und identifizieren soll?

3 Methoden

Die Fragestellung wird im Folgenden am Beispiel von acht Verkehrszeichen geklärt.

3.1 Convolutional Neural Networks

Ein Convolution Neural Network, auch ConvNet oder CNN genannt, ist ein Deep-Learning-Algorithmus, der ein Bild aufnimmt, verschiedenen Objekten im Bild eine Bedeutung zuweist und diese Objekte voneinander unterscheiden kann [1]. Das CNN ist ein mathematisches Konstrukt das normalerweise aus Schichten oder Bausteinen besteht. Die ersten beiden Schichten, Convolution und Pooling-Schicht, extrahieren bestimmte Merkmale aus einem Bild, während die dritte Schicht, vollständig verknüpfte Schicht, die extrahierten Merkmale in die endgültige Ausgabe umsetzt. Eine typische Architektur besteht aus Wiederholungen eines Stapels von mehreren Convolution-Schichten und einer Pooling-Schicht gefolgt von mehreren vollständig verknüpften Schichten. Der Vorgang, bei dem die Eingabedaten durch diese Schichten in Ausgabedaten umgewandelt werden, wird als Vorwärtspropagation bezeichnet. Die in diesem Abschnitt erläuterten Convolution und Pooling-Operationen beziehen sich jedoch auf ein 2D-CNN, ähnliche Operationen können aber auch für 3D-CNN durchgeführt werden.[2]

3.1.1 Convolution

Convolution ist eine lineare Operation, die für die Merkmalsextraktion verwendet wird. Um bestimmte Merkmale zu extrahieren wird ein Array, der sogenannte Kernel, auf das Eingabe-Bild, das auch zu einem Array umgewandelt wurde, namens Kernel, angewendet. Ein elementweises Produkt zwischen jedem Element des Kernels und dem Eingabetensor wird an jeder Stelle des Tensors kalkuliert und summiert, um den Ausgabewert an der zugehörigen Stelle des Ausgabetensors, einer so genannten Merkmalskarte, zu erhalten. Dieses Verfahren wird unter Anwendung mehrerer Kernel wiederholt, um eine beliebige Anzahl von Merkmalskarten zu erzeugen, die jeweils verschiedene Merkmale des Eingabetensors darstellen. Die hier beschriebene Convolution-Operation lässt nicht zu, dass die

Mitte eines Kernels das äußerste Element des Eingabetensors überdeckt, und verringert somit die Höhe und Breite der Merkmalskarte. Das Auffüllen mit Nullen ist eine Technik zu Lösung dieses Problems, bei der Zeilen und Spalten mit Nullen auf jeder Seite des Eingabetensors angefügt werden, so dass das Zentrum eines Kernels auf das äußerste Element passt und die gleiche Dimension in der Ebene durch die Convolution-Operation beibehalten wird. Der Abstand zwischen zwei aufeinanderfolgenden Kernelpositionen wird als Stride bezeichnet, der auch die Convolution-Operation definiert. Üblicherweise wird ein Stride der Größe 1 gewählt, jedoch wenn man ein Downsampling der Merkmalskarten erreichen möchte, wird ein Stride größer als 1 verwendet.

3.1.2 Pooling-Schicht

Eine Pooling-Schicht bietet eine klassische Downsampling-Operation, die die Dimensionalität der Merkmalskarten in der Ebene reduziert, um eine Übersetzungsinvarianz gegenüber kleinen Verschiebungen und Verzerrungen einzuführen und die Anzahl der nachfolgenden lernbaren Parameter zu verringern. Es ist hervorzuheben, dass es in keiner der Pooling-Schichten einen lernbaren Parameter gibt, während Filtergröße, Stride und Padding Hyperparameter bei Pooling-Operationen sind, ähnlich wie bei Convolution-Operationen.[2]

3.1.3 Max Pooling

Die gängigste Form der Pooling-Operation ist das Max-Pooling, bei dem aus den eingegebenen Merkmalskarten Felder extrahiert werden, der maximale Wert in jedem Feld ausgegeben und alle anderen Werte verworfen werden. Üblicherweise wird ein Filter der Größe 2 x 2 mit einem Stride von 2 verwendet. Dadurch wird die In-Plane-Dimension der Merkmalskarte um den Faktor 2 verkleinert.[2]

3.1.4 Global Average Pooling

Eine weitere erwähnenswerte Pooling-Operation ist das Global Average Pooling, dass eine extreme Art des Downsamplings durchführt. Hierbei wird eine Merkmalskarte mit einer Größe von Höhe x Breite in ein 1 x 1 Array verkleinert, indem der Durchschnitt aller Elemente in jeder Merkmalskarte gebildet wird, während die Tiefe der Merkmalskarte erhalten bleibt. In der Regel wird dieser Vorgang nur einmal vor den vollständig verknüpften Schichten durchgeführt. Der erste Vorteil des Global Average Pooling's ist, dass die Anzahl der lernbaren Parameter reduziert wird und der zweite Vorteil ist, dass das CNN Eingaben variabler Größe akzeptieren kann.[2]

3.1.5 Vollständig Verknüpfte Schicht

Die Ausgangsmerkmalskarten der letzten Convolution oder Pooling-Schicht werden in der Regel abgeflacht, das heißt in ein eindimensionales Zahlenfeld umgewandelt und mit einer oder mehreren vollständig verknüpften Schichten, auch als dichte Schichten bekannt, verbunden, in der jeder Eingang mit jedem Ausgang durch ein lernbares Gewicht verbunden ist. Sobald die von den Convolution-Schichten extrahierten und von den Pooling-Schichten heruntergestuften Merkmale erstellt sind, werden sie von einer Untergruppe vollständig verknüpften Schichten auf die endgültigen Ausgaben des Netzes abgebildet, zum Beispiel die Wahrscheinlichkeiten für jede Klasse bei Klassifizierungsaufgaben. Die letzte vollständig verknüpfte Schicht hat in der Regel die gleiche Anzahl von Ausgangsknoten wie die Anzahl der Klassen.[2]

3.2 Implementierung

Die Implementierung in Java 17 besteht aus den vier neuen Klassen *ImageAdapter*, *ConvNet*, *RoadSignLabel*, *RoadSignTest* und einem Record *TrainData*. Zum Testen wird eine weitere Klasse und ein Enum verwendet. Diese vier neuen Klassen, das Record sowie die Testdateien werden im Folgenden Absätzen erklärt.

3.2.1 ConvNet

Die Klasse *ConvNet* soll ein Convolutional Neural Network darstellen und erbt daher von der im Projektbericht [3] unter Abschnitt 2 genannten Klasse *Network*, was jener die gleichen Eigenschaften verleiht. Des Weiteren besitzt sie eine *java.util.ArrayList* vom Generic-Typ *TrainData* und einen *ImageAdapter* als private konstante Attribute. Erstere dient dem Speichern von Trainingsdatensätzen, die dann mit dem Aufrufen der überladenen Methode *train(double, int)* benutzt werden. Diese ruft die originale Instanz-Methode *train(double[][], double[][], double, int)* der Klasse *Network* mit den Daten aus der *ArrayList* auf. Außerdem bietet die Klasse *ConvNet* die Instanz-Methode *addTrainData(String, int[])* um dieser Trainingsdatensätze hinzuzufügen. Diese nimmt einen Dateinamen von einem Bild und einen ganzzahligen Zielvektor. Zwei private nicht-statische Hilfsmethoden *getImages()* und *getTargets()* dienen der besseren Handhabung der *ArrayList*.
Die Klasse erstellt durch den Konstruktor *ConvNet(String, int, int, int, int[])* (Pfad zum Ordner mit den Bildern, gewünschte Bildweite, -höhe, Anzahl der Output-Neuronen, Anzahl der Hiddenlayers und deren Neuronen) von ihr abgeleitete Objekte.

3.2.2 ImageAdapter

Der *ImageAdapter* ist dafür da, um das Bild zu laden und skalieren. Die Klasse besteht aus sechs Methoden und einem Konstruktor. Im Konstruktor *ImageAdapter(String path,*

int imgWidth, int imgHeight) werden Pfad zu den Bildern, gewünschte Bildhöhe sowie -breite als Parameter übergeben, mit denen dann die jeweiligen Skalierungswerte für das jeweilige Bild initialisiert werden. Die Methode *loadImage(String filename)* lädt das *java.awt.image.BufferedImage* mit dem übergebenen Namen und gibt es zurück. Falls es keine Datei mit diesem Namen gibt bzw. diese keine Bilddatei ist, wird eine *FileNotFoundException* bzw. *IllegalArgumentException* eine geworfen. Als nächstes geht die Methode *getRGBs(BufferedImage img)* alle Pixel durch und gibt deren hexadezimale RGB-Werte als Matrix für das jeweilige Bild zurück. Die Methode *squareImage(BufferedImage img)* ist dafür da, um das Bild quadratisch zuzuschneiden. *scaleImage(BufferedImage img)* skaliert ein Bild mit einer beliebigen Höhe oder Breite. Um ein java.awt.image.Image in ein java.awt.image.BufferedImage konvertieren, wird die Methode *convertToBufferedImage(Image image)* benutzt. In der einzigen öffentlichen Methode des *ImageAdapter*s *getImageRGBs(String filename)* wird mit den genannten Methoden und mithilfe eines übergebenen Dateinamens ein eindimensionales ganzzahliges Array mit allen RGB-Werten des aus dem Dateinamen resultierenden Bildes zurückgegeben.

$$\vec{V_{wh}} = B_{w \times h} \cdot \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

3.2.3 TrainData

Das Record *TrainData(int[], int[])* besitzt einen Vektor mit den RGB-Werten eines Bildes und einen Zielvektor. Außerdem hat es zwei Instanz-Methoden *getImageAsDoubleArray()* und *getTargetAsDoubleArray()*, welche die jeweiligen ganzzahligen Vektoren als Gleitkommazahl-Vektoren zurückgibt, und eine private nicht-statische Hilfsmethode *getIntAsDoubleArray(int[])*.

3.2.4 Test-Dateien

Für das Testen wurden ein Enum *RoadSignLabel* und eine Klasse *RoadSignTest* implementiert. Das Enum zählt die möglichen Ergebnisse des Netzes auf, indiziert sie und kann den jeweils richtigen Zielvektor durch die Instanz-Methode *getTarget()* zurückgeben. In der Klasse befindet sich die für die Forschungsfrage relevante Main-Methode des Programms. Diese ruft eine private statische Methode *test()* in der selben Klasse auf. In dieser wird ein *ConvNet* deklariert und initialisiert, mit Datensätzen bestückt und trainiert. Dabei werden einige hilfreiche Ausgaben gemacht.

3.3 Tests

4 Ergebnisse

5 Grafiken & Tabellen

Schildbilder im Trainingsdatensatz	
Andreaskreuz	5
Fußgängerüberweg	6
Gefahrenstelle	2
Vorfahrt gewähren	5
Vorfahrtsstraße	6
Verbot der Einfahrt	8
Fußgängerweg	8
Stopp	4

(1)

Anzahl Neuronen im CNN					
Input	**Hidden 1**	**Hidden 2**	**Hidden 3**	**Hidden 4**	**Output**
4096	64	64	64	64	8

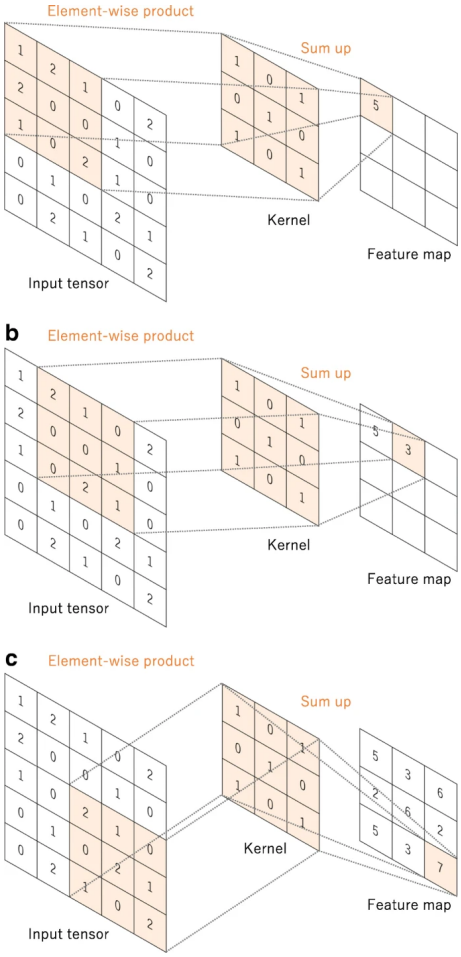
(2)

Bildgröße	
Höhe	**Breite**
64	64

(3)

Training	
Lernrate	**Wiederholungen**
0.12	100000

(4)



(5)

6 Diskussion

7 Literatur

[1] S. Saha. (2018). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, Adresse: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53` (besucht am 08. 01. 2022).

[2] R. Yamashita, M. Nishio, R. K. G. Do und K. Togashi. (2018). Convolutional neural networks: an overview and application in radiology, Adresse: `https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9` (besucht am 08. 01. 2022).

[3] L. Leggewie, A. Schallenberg, A. Zarkh und S. Y. Zhang, "Neuronale Netze Projektbericht," 2021, unpublished.

8 Anhang