

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

10. Januar 2022

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Zusammenfassung | 3 |
| 2 | Einleitung | 3 |
| 2.1 | Verwendete Literatur | 3 |
| 2.2 | Theorie | 3 |
| 2.3 | Fragestellung | 4 |
| 3 | Methoden | 4 |
| 3.1 | Convolutional Neural Networks | 4 |
| 3.1.1 | Convolution | 4 |
| 3.1.2 | Pooling-Schicht | 5 |
| 3.1.3 | Max Pooling | 5 |
| 3.1.4 | Global Average Pooling | 6 |
| 3.1.5 | Vollständig Verknüpfte Schicht | 6 |
| 3.2 | Implementierung | 6 |
| 3.2.1 | ConvNet | 6 |
| 3.2.2 | ImageAdapter | 7 |
| 3.2.3 | TrainData | 7 |
| 3.2.4 | Test-Dateien | 8 |
| 3.3 | Tests | 8 |
| 4 | Ergebnisse | 9 |
| 5 | Diskussion | 10 |
| 6 | Abbildungsverzeichnis | 10 |
| 7 | Tabellenverzeichnis | 11 |
| 8 | Anhang | 11 |

1 Zusammenfassung

In diesem Forschungsbericht werden Auswirkungen eines Hintergrundes auf die Erkennung und Identifikation von Objekten in Bildern mithilfe eines sogenannten Convolutional Neural Network vorgestellt und diskutiert. Dies geschieht anhand des Beispiels der Erkennung und Identifikation von Verkehrszeichen auf Straßen. Im ersten Teil des Textes wird die verwendete Literatur, die Theorie und die Fragestellung präsentiert. Anschließend folgen die Methoden, die zur Klärung der Fragestellung beitragen. Unter anderem wird hier erklärt, was ein Convolutional Neural Network ist und wie es hier implementiert wurde. Dort werden im Detail dann die Java-Dateien und deren Methoden sowie Attribute vorgestellt. Der letzte Teil des Abschnitts Methoden präsentiert dann die durchgeführten Tests. Zum Schluss werden die Ergebnisse aufgeführt und zusammengefasst. Daraufhin wird die Fragestellung des Berichts nochmal aufgegriffen und diskutiert. In den letzten beiden Abschnitten findet man sowohl die Verzeichnisse 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 Computer Vision. Computer Vision kann durch verschiedene Arten umgesetzt werden. Eine davon funktioniert mit künstlichen neuronalen Netzen. Dieses wird dann Convolutional Neural Network (kurz CNN oder ConvNet) genannt.

2.1 Verwendete Literatur

2.2 Theorie

Computer Vision kann durch verschiedene Arten umgesetzt werden. Eine davon funktioniert mit künstlichen neuronalen Netzen. Eine fortgeschrittene Version davon wird dann Convolutional Neural Network (kurz CNN oder ConvNet) genannt.

Ein Auto kann dadurch durch ein Video oder ein Bild einer eingebauten Kamera beispielsweise Verkehrszeichen erkennen und auf sie reagieren. Dazu muss in diesem Video oder Bild zunächst ein Schild erkannt werden, es muss identifiziert werden, um welches Schild es sich handelt und es muss reagiert werden. Diese Erkennung und Identifikation ist Aufgabe eines solchen Convolutional Neural Network.

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 **cnn1**. 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.**cnn2**

3.1.1 Convolution

Convolution (im CNN auch Convolutional-Schicht) 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 zur 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.

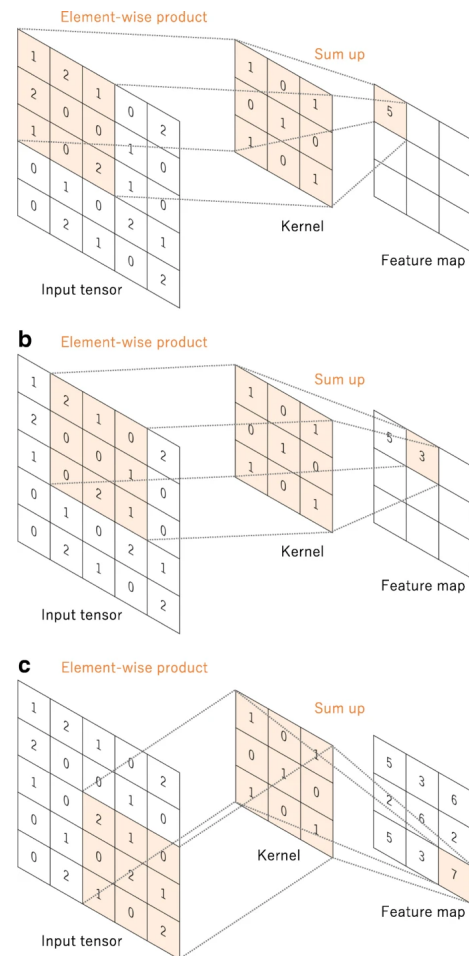


Abbildung 1: Benutzung des Kernels

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.**cnn2**

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.**cnn2**

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.**cnn2**

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.**cnn2**

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 **projektbericht** 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-Methoden `addTrainData(String, int[])` und `addTrainData(String, int[], boolean)` um dieser Trainingsdatensätze hinzuzufügen. Diese nehmen einen Dateinamen von einem Bild und einen ganzzahligen Zielvektor. Zweitere nimmt zusätzlich einen Wahrheitswert, der bestimmt, ob das Bild vor dem B'-nutzen bearbeitet werden soll. Erstere Methode ruft die Zweite mit dem Wahrheitswert `false` auf. 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, int, int)` 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)` lädt das `java.awt.image.BufferedImage` mit dem übergebenen Namen und gibt es zurück. Falls es keine Datei mit diesem Namen gibt oder diese keine Bilddatei ist, wird eine `IllegalArgumentException` eine geworfen. Als nächstes geht die Methode `getRGBs(BufferedImage)` alle Pixel durch und gibt deren hexadezimale RGB-Werte als Matrix für das jeweilige Bild zurück. Die Methode `squareImage(BufferedImage)` ist dafür da, um das Bild quadratisch zuzuschneiden. `scaleImage(BufferedImage)` 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)` benutzt. In der einzigen öffentlichen Methode des `ImageAdapters` `getImageRGBs(String, boolean)` wird mit den genannten Methoden und mithilfe eines übergebenen Dateinamens und eines Wahrheitswerts ein eindimensionales ganzzahliges Array mit allen RGB-Werten per *row-major order*

$$m_{ij} \rightarrow a_{64 \cdot i + j} \quad (1)$$

des aus dem Dateinamen resultierenden Bildes zurückgegeben. Der Wahrheitswert bestimmt, ob das Bild, welches geladen wird, zugeschnitten und skaliert werden soll, oder nicht.

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

Für das Testen wurden 43 Bilder mit Verkehrszeichen gesammelt, einige gut erkennbar, andere schwieriger. Diese Originalbilder sind dann zu 128x128px-Bildern konvertiert worden, um einen Netzeingabevektor konstanter Größe gewährleisten zu können. Die konvertierten Bilder seien im folgenden der Datensatz b . Dieser und ein daraus resultierender Datensatz b' , wobei der Hintergrund durch einen weißen (0xFFFFFFFF) Hintergrund ersetzt wurde, sind nun zum Testen verwendet worden. So ist um ein direkter Vergleich von Datensatz b "mit Hintergründen" und Datensatz b' "ohne Hintergründe" möglich. Die Bilder sind in acht Gruppen aufgeteilt, zeigen also acht unterschiedliche Verkehrszeichentypen: *Andreaskreuz*, *Fußgängerüberweg*, *Gefahrenstelle*, *Vorfahrt gewähren*, *Vorfahrtstraße*, *Verbot der Einfahrt*, *Fußgängerweg* und *Stopp*. Fast alle Bilder sind genau einer Gruppe zugeordnet. Ausnahmen sind zwei Bilder, die jeweils den beiden Gruppen *Fußgängerüberweg* und *Vorfahrt gewähren* zugeordnet sind.

| Gruppen der Bilder | |
|---------------------|--------|
| Gruppe | Anzahl |
| Andreaskreuz | 5 |
| Fußgängerüberweg | 6 |
| Gefahrenstelle | 2 |
| Vorfahrt gewähren | 6 |
| Vorfahrtstraße | 6 |
| Verbot der Einfahrt | 8 |
| Fußgängerweg | 8 |
| Stopp | 4 |

Tabelle 1: Anzahl der Bilder pro Gruppe

Das Convolutional Neural Network, welches zum Testen verwendet wird, ist auf einen 16384-dimensionalen Eingabevektor konfiguriert. Es besitzt vier versteckte Schichten, wobei die Erste und Zweite jeweils 64 Neuronen und die Dritte und Vierte 32 Neuronen haben. Die Ausgabeschicht besteht aus acht Neuronen. Der acht-dimensionale Ausgabevektor klassifiziert das jeweilige/die jeweiligen Verkehrszeichen in dem eingegebenen Bild.

Nun wurde sich angeschaut wie sich die Kosten nach jeder Epoche des Lernens des

CNN abhängig vom verwendeten Datensatz verändern. Dazu wurde durch wiederholtes Anpassen eine Lernrate von 0.1111 gewählt. Das Training ist 1000 mal wiederholt worden, besteht also aus 1000 Epochen. Die Gewichtungen wurden zufällig mit Werten im Intervall $[-0.5, 0.5)$ initialisiert. Das CNN wurde nacheinander mit beiden Datensätzen trainiert. Die Ausgangsgewichtungen waren dabei gleich. Dieses Training wurde mehrfach wiederholt, um Aussagekraft des Tests zu erhöhen.

| Anzahl Neuronen im CNN | | | | | |
|------------------------|----------|----------|----------|----------|--------|
| Input | Hidden 1 | Hidden 2 | Hidden 3 | Hidden 4 | Output |
| 16384 | 64 | 64 | 32 | 32 | 8 |

| Bildgröße | | Training | |
|-----------|--------|----------|----------------|
| Höhe | Breite | Lernrate | Wiederholungen |
| 128 | 128 | 0.1111 | 1000 |

4 Ergebnisse

Das Testen ergab, dass die Kosten bei beiden Datensätzen stetig fallen, also die Steigung der Kosten im gesamten Epochen-Intervall $[1, 1000]$ negativ ist. Bei beiden sind die Kosten vor allem zwischen den ersten beiden Epochen sehr stark gesunken.

Danach ist die Steigung der Kosten für einige Epochen nahe 0, wird aber von Epoche zu Epoche kleiner. Darauf folgt ein annähernd exponentieller Verlauf. Während bei Datensatz b im Epochen-Intervall $[190, 260]$ kleinere Abweichungen einer Exponentialfunktion zu verzeichnen sind, worauf bis Epoche 420 eine größere Abweichung folgt, ist in Da-

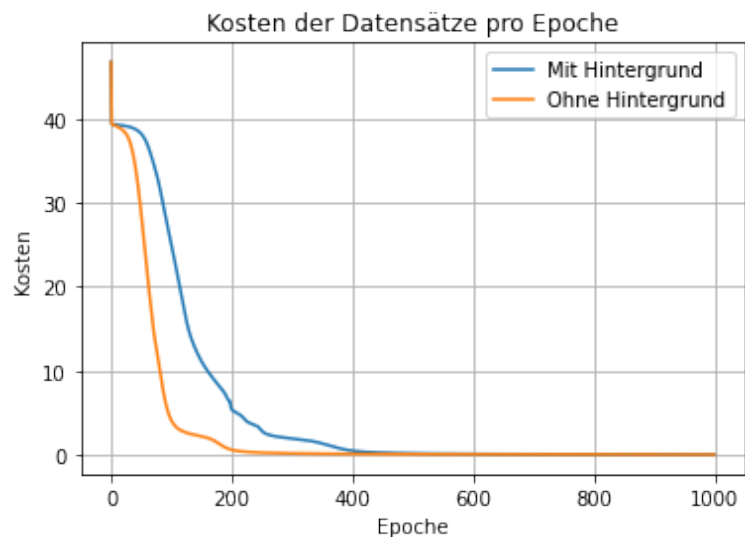


Abbildung 2: Kosten der Datensätze pro Epoche

tensatz b' nur eine größere Abweichung zu finden, wobei diese bereits im Intervall $[100,$

250] liegt. Die Kosten beider Datensätze nähern sich bis Epoche 1000 exponentiell der 0. Vergleicht man beide Verläufe, so sinken anfangs die Kosten von b' ($C_{b'}$), also dem Datensatz “ohne Hintergrund” schneller als die Kosten von b (C_b), also dem Datensatz “mit Hintergrund”. Die Differenz $C_b - C_{b'}$ wird zwar nie 0, jedoch sinkt sie ebenfalls exponentiell, da C_b und $C_{b'}$ beide exponentielle Verläufe aufweisen.

Wiederholt man das Testen mit verschiedenen Gewichtsmatrizen, so ergeben sich auch hier zu den beschriebenen Entwicklungen ähnliche Verläufe.

Welche Auswirkungen hat nun 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? Den Tests ist zu entnehmen, dass der Hintergrund die Erkennung und Identifikation des Objektes, also in dem getesteten Beispiel des/der Verkehrszeichen, erschwert.

5 Diskussion

Die Erschwerung der Erkennung und Identifikation der Verkehrszeichen durch den Hintergrund ist in mehreren Aspekten begründet. Viele dieser Aspekte gehen nicht direkt vom Convolutional Neural Network selber aus, sondern von der Umgebung. Wie zum Beispiel Verkehrszeichen, die schwer zu erkennen sind, weil sie entweder schon sehr alt und dreckig sind oder zugeklebt wurden. Das erschwert dem CNN die Erkennung des Verkehrszeichens und kann zu Irrtümern führen, die dann im schlimmsten Fall einen Unfall verursachen können. Ein weiteres Problem könnte der Verkehrszeichenwald in der Stadt sein. Dann wird es für das CNN schwerer, sich das gewollte Verkehrsschild auszusuchen, da mittlerweile sehr viele Verkehrszeichen an einem Ort angebracht sind. Schwieriger wird es auch für das Convolutional Neural Network ein Verkehrszeichen zu erkennen, wenn der Hintergrund sehr farbig ist und den Farben des Verkehrsschildes ähnelt. Dann könnte es schwierig werden, das Bild zurecht zuschneiden und die Ränder des Verkehrszeichens zu erkennen.**discuss1**

Andererseits kann man den Tests entnehmen, dass durch ausreichend Training mit ggf. komplexeren Netzen und/oder größeren Trainingsdatensätzen die Auswirkungen von Hintergründen minimiert werden können.

6 Abbildungsverzeichnis

| | | |
|---|--|---|
| 1 | Benutzung des Kernels | 5 |
| 2 | Kosten der Datensätze pro Epoche | 9 |

7 Tabellenverzeichnis

| | | |
|---|--|---|
| 1 | Anzahl der Bilder pro Gruppe | 8 |
|---|--|---|

8 Anhang