Generierung künstlicher Trainingsdaten für die Straßenschilderkennung in Fahrzeugen mittels Generative Adversarial Networks

Studienarbeit

Studiengang Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Frederik Esau

08.06.2023

Bearbeitungszeitraum	24.10.2022 - 08.06.2023
Matrikelnummer, Kurs	6526552, TINF20ITA
Betreuer	Prof. Dr. Monika Kochanowski

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *Generierung künstlicher Trainingsdaten für die Straßenschilderkennung in Fahrzeugen mittels Generative Adversarial Networks* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 08.06.2023

Frederik Esau

Abstract

Abstract normalerweise auf Englisch. Siehe: http://www.dhbw.de/fileadmin/user/public/Dokumente/Portal/Richtlinien_Praxismodule_Studien_und_Bachelorarbeiten_JG2011ff.pdf (8.3.1 Inhaltsverzeichnis)

Ein „Abstract“ ist eine prägnante Inhaltsangabe, ein Abriss ohne Interpretation und Wertung einer wissenschaftlichen Arbeit. In DIN 1426 wird das (oder auch der) Abstract als Kurzreferat zur Inhaltsangabe beschrieben.

Objektivität soll sich jeder persönlichen Wertung enthalten

Kürze soll so kurz wie möglich sein

Genauigkeit soll genau die Inhalte und die Meinung der Originalarbeit wiedergeben

Üblicherweise müssen wissenschaftliche Artikel einen Abstract enthalten, typischerweise von 100-150 Wörtern, ohne Bilder und Literaturzitate und in einem Absatz.

Quelle: http://de.wikipedia.org/wiki/Abstract Abgerufen 07.07.2011

Diese etwa einseitige Zusammenfassung soll es dem Leser ermöglichen, Inhalt der Arbeit und Vorgehensweise des Autors rasch zu überblicken. Gegenstand des Abstract sind insbesondere

- Problemstellung der Arbeit,
- im Rahmen der Arbeit geprüfte Hypothesen bzw. beantwortete Fragen,
- der Analyse zugrunde liegende Methode,
- wesentliche, im Rahmen der Arbeit gewonnene Erkenntnisse,
- Einschränkungen des Gültigkeitsbereichs (der Erkenntnisse) sowie nicht beantwortete Fragen.

Quelle: http://www.ib.dhbw-mannheim.de/fileadmin/ms/bwl-ib/Downloads_alt/Leitfaden_31.05.pdf, S. 49

Inhaltsverzeichnis

Abkürzungsverzeichnis

CNN	Convolutional Neural Network
CycleGAN	Cycle-Consistent Generative Adversarial Network
GAN	Generative Adversarial Network
GTSRB	German Traffic Sign Recognition Benchmark
KNN	Künstliches Neuronales Netz
PixelRNN	Pixel Recurrent Neural Network
SSIM	Index struktureller Ähnlichkeit (eng.: Structural Similarity Index)
SVM	Support Vector Machine

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Während in großen Teilen des letzten Jahrhunderts Innovationen in der Fahrzeugentwicklung vor allem im Bereich der mechanischen Leistung und Effizienz stattgefunden haben, erwartet man zukünftige Verbesserungen im Automobil besonders softwareseitig [1]. ...

1.1 Problemstellung

1.2 Vorgehensweise

1.3 Ziel der Arbeit

2 Stand der Technik

2.1 Straßenschilderkennung

Eine Straßenschilderkennung zählt mittlerweile zu der Standardausstattung vieler Neuwagen. Im Jahr 2024 tritt zudem eine EU-Verordnung in Kraft, durch die sämtliche neu produzierten Fahrzeuge mit einer solchen Funktion ausgestattet werden müssen [2]. Daran zeigt sich, dass das Thema bereits weitreichend etabliert ist.

Straßenschilder werden zu folgendem Zweck eingesetzt: Es sollen Informationen über die Verkehrssituation und über geltende Vorschriften des Gebiets, in dem sich das Fahrzeug zu einem gegebenen Zeitpunkt befindet, präsentiert werden. Durch Straßenschilder werden unter anderem Geschwindigkeitsvorgaben, Gefahrenhinweise und allgemeine Verkehrsregeln kommuniziert. Dabei sind die Schilder so konzipiert, dass sie sich visuell möglichst von ihrem Hintergrund abheben und leicht voneinander zu unterscheiden sind. Automatische Straßenschilderkennungen können Fahrer*innen in Situationen unterstützen, in denen sie Schilder übersehen oder gezielt missachten. Anstelle dass ein reales Schild beispielsweise nur für einige Sekunden sichtbar ist, bevor es außerhalb der Sichtweite des Fahrzeugführenden ist, ist eine durchgehende Anzeige auf den Displays eines Fahrzeugs möglich. Auch akustische Warnungen oder ein aktives Eingreifen von Sicherheitssystemen sind denkbar, beziehungsweise bereits in Serienfahrzeugen vorhanden.

Eine Straßenschilderkennung erfolgt visuell und wird somit mittels Kameras umgesetzt. Dabei lassen sich viele Schilder durch eine bestimmte Form (Kreis, Dreieck, Achteck, etc.) und ein Piktogramm (Schneeflocke, Person, etc.) oder eine Zahl identifizieren. Somit können die verschiedenen Arten von Straßenschilder in Klassen unterteilt werden, die durch den Erkennungsalgorithmus detektiert werden. Für die praktische Umsetzung solcher Algorithmen werden häufig KNNs verwendet. Diese werden in Kapitel 2.2 thematisch aufgeführt. Besonders relevant für das Thema dieser Studienarbeit ist, wie zuverlässig die momentan ausgelieferten Straßenschilderkennungen sind und welche Situationen die Algorithmen am ehesten zu falschen Aussagen verleiten. Auf dieser Grundlagen kann sich orientiert werden, welche Arten von Bildern vermehrt generiert werden sollen, um die Straßenschilderkennung verbessern zu können.

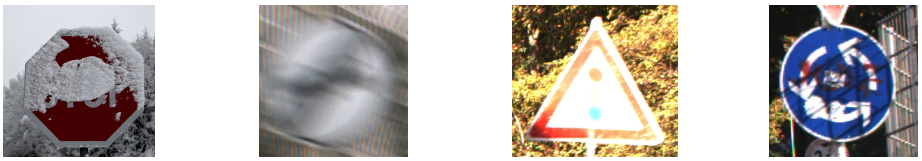
Im Jahr 2019 hat eine Automobilzeitschrift die Straßenschilderkennung von unterschiedlichen Fahrzeugherstellern getestet [3]. Zudem existieren verschiedene Publikationen, die sich mit

der Thematik befassen [4]. Die Ergebnisse des Zeitschriftenartikels weisen darauf hin, dass die Straßenschilderkennung einiger Fahrzeuger bereits weitreichend funktioniert. Geschwindigkeitsvorgaben werden überwiegend erkannt und dem Fahrer auf einem Display angezeigt. Auch existieren bereits akustische Warnungen bei einer Überschreitung der Höchstgeschwindigkeit. Dennoch existieren einige Situationen, die bei mehreren Fahrzeugen zu Problemen bei der Straßenschilderkennung geführt haben. Einen exemplarischen Überblick soll die folgende Grafik bieten: [3]

Aufhebung Ungültigkeit Dunkelheit LED Schilder

Abbildung 2.1: Erschwerende Einflüsse auf die Straßenschilderkennung

Fahrzeuge verschiedener Hersteller haben in den Tests Aufhebungsschilder nicht korrekt interpretiert. Damit sind Schilder gemeint, die entweder Geschwindigkeitsbegrenzungen oder Überholverbote außer Kraft setzen. Des Weiteren sorgten mittels Klebestreifen als ungültig erklärte Schilder, in einigen Fällen Dunkelheit, beispielsweise in Tunneln, und sogenannte LED *Wechselverkehrszeichen* für falsche Aussagen. Auch wird teilweise nicht erkannt, wenn Schilder für eine kreuzende Straße gelten, statt für die Straße, auf der sich das Fahrzeug zu dem gebenen Zeitpunkt befinden. Weitere Aspekte, die in dem Artikel nicht explizit genannt sind, aber laut einer Publikation von 2014 in der Vergangenheit zu Schwierigkeiten geführt haben, sind mitunter die folgenden: [4]



Witterung Unschärfe Belichtung Vandalismus

Abbildung 2.2: Weitere mögliche Einflüsse auf die Straßenschilderkennung

Ausgehend der Erkentnisse der Tests aus dem genannten Artikel, kann in einigen solcher Situationen davon ausgegangen werden, dass sich in vielen Fahrzeugen nicht auf die automatische Erkennung der Schilder verlassen werden kann. Ein Ziel der Hersteller ist das Anbieten von Fahrzeugen, die vollständig autonom, das heißt ohne menschliches Eingreifen, fahren können. Damit dies möglich ist, muss die Software der Fahrzeuge auch solche Grenzfälle korrekt interpretieren. Dies erfordert eine gewisse Menge an Daten, durch die diese Algorithmen trainiert werden.

Ziel dieser Arbeit ist ausgehend davon, gezielt Trainingsbilder erzeugen zu können, die einige dieser Aspekte simulieren. Es soll als alternative Möglichkeit dazu vorgeschlagen werden, sämtliche Trainingsdaten für Grenzfälle eigenständig in realen Fahrsituationen aufzunehmen.

2.2 Künstliche Neuronale Netze

Durch künstliche neuronale Netze (KNNs) können Maschinen lernen, bestimmte Probleme zu lösen, ohne dass ein Mensch vorher explizite Regeln dafür definieren muss. Dies steht im Kontrast zur Methode, der Maschine vorher einen festen, vollständigen Regelsatz bereitzustellen. Letztgenannter Ansatz zeigt in einigen Gebieten nur begrenzten Erfolg, da es für Menschen herausfordernd sein kann, Regelsätze für Vorgänge zu definieren, die unbewusst im Gehirn stattfinden oder viel Kontext erfordern. Zu nennen sind hierbei die visuelle Objekterkennung oder menschliche Sprache. Außerdem können neue, nicht in den Regeln beachtete Situationen dazu führen, dass die Maschine das Problem nicht mehr lösen kann. Die Grundidee hinter KNNs ist deshalb, dass sich die Maschine selber einen Wissensschatz aufbaut, der ihr beim Lösen des Problems hilft. Dies geschieht, indem die Entwickler ihr reale Trainingsbeispiele zeigen. Möchte man einen Algorithmus trainieren, der Schach spielen soll, kann man ihm beispielsweise eine Vielzahl an realen Schachpartien zeigen. Anhand dessen lernt der Algorithmus verschiedene Strategien und baut ein Spielverständnis auf, das womöglich über die menschlichen Fähigkeiten hinausgeht. [5, S. 1ff.]

In den letzten Jahrzehnten erlebte das maschinelle Lernen und damit auch das Gebiet der KNNs einen Aufschwung. Es existiert bereits seit Mitte des vergangenen Jarhunderts, wird allerdings erst durch die zunehmende Rechenleistung und die Verfügbarkeit von großen Datenmengen flächendeckend eingesetzt. Einsatzgebiete für KNNs sind unter anderem die Objekterkennung, das Verstehen von natürlicher Sprache und die Generierung von Text und Bildern. [6, S. 4, 17]

Die Inspiration für KNNs bildet die Informationsverarbeitung des Gehirns in Lebewesen. Die kleinste hier betrachtete Einheit ist das Neuron. Neuronen in KNNs sind konzeptionell inspiriert von realen, biologischen Neuronen, besitzen aber eine deutlich abstrahierte Funktionsweise. In KNNs berechnen sie ein Skalarprodukt ihrer gewichteten Eingangswerte, addieren einen sogennaten *Bias* hinzu und wenden auf das Ergebnis eine nichtlineare Funktion an. Letzere wird auch als Aktivierungsfunktion bezeichnet. Diese Aktivierungsfunktion kann analog dazu gesehen werden, dass biologische Neuronen einen Schwellenwert (*engl.: threshold*) besitzen, der überschritten werden muss, damit das Neuron *feuert*, also einen Impuls an weitere Neuronen weitergibt. Aktivierungsfunktionen sind notwendig, damit neuronale Netze Probleme lösen können, die über die Fähigkeiten einer linearen Regression hinausgehen. Es wird eine nichtlineare Abhängigkeit zwischen dem Eingang X und dem Ausgang Y umgesetzt. [7]

In der Nachfolgenden Abbildung ist ein einzelnes künstliches Neuron eines KNNs darstellt. Das *Plus-Zeichen* steht für die Berechnung des Skalarprodukts der Eingänge und der darauf addierte Bias. Die Aktivierungsfunktion wird durch das *Sigmoid*-Zeichen simbolisiert. Der Ausgang (*rechts*) ist das Ergebnis der Berechnung des Neurons. [7]

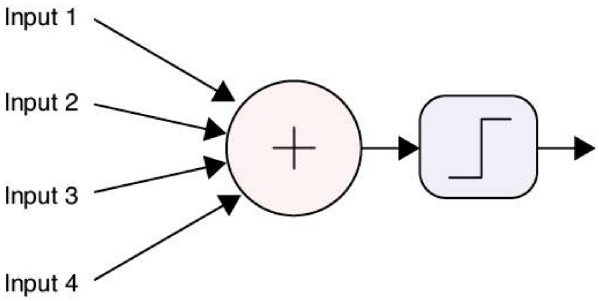


Abbildung 2.3: Einzelnes Neuron eines KNNs [7]

Um komplexe Probleme lösen zu können, werden mehre Neuronen miteinander verbunden und in Schichten angeordnet. Jede Schicht erhält die Ausgangswerte der vorherigen Schicht als Eingang und gibt die daraus berechneten, neuen Werte an die nächste Schicht weiter. In der nachfolgenden Abbildung sind die Neuronen durch Kreise dargestellt und ihre Verbindungen durch Linien. Eine Verbindung stellt dar, dass ein Neuron seinen berechneten Wert an das nachfolgende Neuron weitergibt. Dies geschieht hierbei ausschließlich von *links nach rechts*, womit das KNN als *Feedforward-Netzwerk* bezeichnet wird. Die Eingabeschicht erhält die Eingabewerte des Netzwerks, die Ausgabeschicht liefert die Vorhersage des Modells. Zwischen diesen beiden Schichten befinden sich beliebig viele verarbeitende Schichten, die als *Hidden Layer* bezeichnet werden. [6]

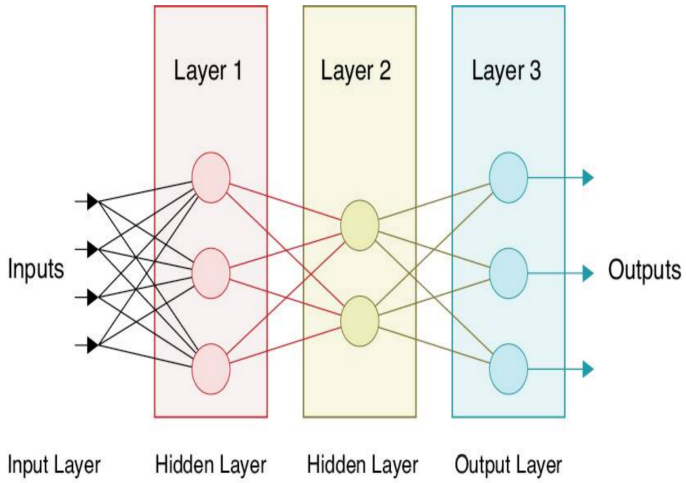


Abbildung 2.4: Vollständiges KNN [7]

Die Vorhersage, gekennzeichnet durch die Werte der Ausgabeschicht, hängt von den jeweiligen Parametern der Neuronen des Netzwerks ab. Dies sind die Gewichte (*engl.: weights*)

der Verbindungen zwischen den Neuronen sowie der Bias der Neuronen. Es existieren auch trainierbare Aktivierungsfunktionen, diese sind jedoch vergleichsweise unüblich. Entwickler sind für den Entwurf der Netzwerkarchitektur zuständig, die Parameter werden jedoch durch das Modell trainiert. Zu Beginn besitzt das Modell zufällige Parameter, wodurch es in der Regel nicht die gewünschte Abbildung zwischen Ein- und Ausgabe implementiert. Ein untrainierter Schachalgorithmus spielt demnach augenscheinlich willkürliche Züge. Ein untrainierter Katzenklassifikator besitzt keinen erkennbaren Wissensschatz darüber, welche Charakteristiken eine Katze optisch auszeichnen. Das Ziel des Trainings ist, dass die Parameter des Modells zunehmend gegen das Optimum konvergieren und so das Modell immer plausibler in seinen Vorhersagen wird.

2.2.1 Training

Damit ein neuronales Netz trainiert werden kann, bedarf es einer Funktion, die die Qualität der Vorhersagen des Modells bestimmt. Erst dadurch kann verglichen werden, ob eine gegebene Änderung der Parameter eine Annäherung an das globale Optimum zurfolge hat. Diese Funktion wird als *Kostenfunktion* bezeichnet. Sie berechnet, gemittelt über alle m Trainingsbeispiele i, die Abweichung des vorhergesagten Wertes $\hat{y}$ von dem tatsächlichen Wert y. Die Kostenfunktion in Abhängigkeit von der Gesamtheit der Gewichte und Biases θ kann beispielsweise folgendermaßen definiert sein:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2 \tag{2.1}$$

Anstelle des Terms $\frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$, der die quadratische Abweichung der vorhergesagten von den erwarteten Werten beschreibt, können auch andere Metriken verwendet werden. Bezeichnet man dies verallgemeinert als Verlustfunktion L *(engl.: loss-function)*, so kann die Kostenfunktion wie folgt definiert werden:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{2.2}$$

Dies wird als überwachtes Lernen bezeichnet, da für die Berechnung der Kostenfunktion *gelabelte* Trainingsdaten erforderlich sind. Das bedeutet, dass jedes Trainingsbild die zu erwartende Ausgabe enthält. Bei Bildern für einen Katzenklassifikator muss beispielsweise für jedes Trainingsbild durch einen Menschen gekennzeichnet werden, ob es eine Katze enthält oder nicht. Erst so kann bewertet werden, inwieweit die jeweiligen Vorhersagen des Modells korrekt sind.

Es ist jedoch nicht nur erforderlich, die Qualität der Vorhersagen des Netzes zu bestimmen. Um die Vorhersagen in zukünftigen Iterationen zu verbessern, ist es notwendig, die Parameter des Modells zu verändern. Darunter fallen, wie bereits erwähnt, die Gewichte und Biases θ. Von ihnen hängt der Wert der Kostenfunktion ab. Die Ausgangssituation ist hierbei folgende: Mit einem gegebenen θ befindet man sich in einem bestimmten Punkt der Kostenfunktion $J(\theta)$. Gesucht ist eine Parameteränderung $\mathrm{d}\theta$, mit der man sich am weitesten an das globale Minimum der Kostenfunktion annähert. Das globale Minimum ist die beste Lösung, die das Modell für die gegebenen Trainingsdaten finden kann und damit das Optimum.

Nähern tut man sich dem Optimum, indem man den Punkt θ in Richtung des negativen Gradienten der Kostenfunktion bewegt. Also die Richtung, in die, aus der momentanen Ausgangsposition, die Kostenfunktion den steilsten Abstieg besitzt. Pro Trainingsiteration nähert man sich dem Optimum nur um einen kleinen Betrag. Diese Annäherung wird als Gradientenabstieg bezeichnet, während der Betrag der Annäherung durch eine sogennante Lernrate α bestimmt wird. Die Lernrate ist ein *Hyperparameter*, und damit klassischerweise ein nicht-trainierbarer Parameter, da sie durch die Entwickler fest bestimmt wird und nicht durch das Modell selbst gelernt wird. Es sind viele Trainingsschritte erforderlich, damit sich das Modell dem Optimum möglichst weit nähert. Der Gradientenabstieg muss demnach häufig durchgeführt werden.

Anhand der nachfolgenden Abbildung, soll der Gradientenabtieg visuell verdeutlicht werden.

2.2.2 Convolutional Neural Networks

KNNs bewähren sich mitunter besonders im Bereich *Computer Vision*. Ein Bereich, der sich mit der Interpretation von Bild- und Videodaten beschäftigt. Hier spielt die Mustererkennung eine tragende Rolle. Es sollen Merkmale erkannt werden, die jedes Objekt eines bestimmten Typs auszeichnen, die jedoch nicht auf jedem Bild die exakt identischen Pixelwerte besitzen. Die typische Form von Katzenohren ist beispielsweise ein Muster, das bei der Katzenerkennung verwendet werden kann. Es ist nicht trivial, allgemeine Regeln zu definieren, welche Pixelmuster als Katzenohr erkannt werden sollen und welche nicht. Deshalb wird hier auf KNNs zurückgegriffen.

Verwendet man hierfür jedoch die bisher beschriebene Netzwerkarchitektur, treten verschiedene Probleme auf. Jedes sogenannte *Feature* des Eingangs wird über die Eingangsschicht in das KNN gespeist. Bei einem Schachalgorithmus kann die Menge aller Features beispielsweise durch die momentane Position aller Figuren auf dem Schachbrett beschrieben werden. Das liegt daran, dass genau diese Werte den Ausgang des Netzwerks beeinflussen. In diesem Fall, welchen Zug der Algorithmus als nächstes spielt. Bei der Bildklassifizierung ist jeder Pixel des

Bildes ein Feature. Ein Netz, das Bilder der Größe 1024x1024 Pixel mit drei Farbkanälen (rot, grün blau) klassifizieren soll, muss demnach folgende Anzahl an Eingängen verarbeiten:

$$1024 \cdot 1024 \cdot 3 = 3.145.728 \tag{2.3}$$

Um eine derartige Anzahl an Eingängen sinnvoll interpretieren zu können, ist ein Netzwerk mit vielen Schichten und Neuronen notwendig. So vielen, dass auch heutige Computer an die grenzen ihrer Rechenleistung gelangen. Mitunter deshalb wird im Bereich Computer Vision auf Convolutional Neural Networks (CNNs) zurückgegriffen.

CNNs basieren auf der Faltung (engl.: Convolution) einer Eingangsmatrix mit einer Faltungsmatrix. Jedes CNN besitzt dabei mitunter mindestens eine Schicht, die eine Faltung durchführt. Diese Schichten werden auch *Convolutional Layer* genannt. Ein Bestandteil eines jeden Convolutional Layers ist die Faltmatrix, welche auch *Kernel* genannt wird. Die Faltung besteht darin, dass diese Matrix über die Werte der Eingangsmatrix geschoben wird, und an jeder Position das Skalarprodukt der jeweiligen Pixelwerte mit den Parametern des Kernels berechnet wird. Dieses Ergebnis ergibt dann den jeweiligen Wert der Ausgangsmatrix.

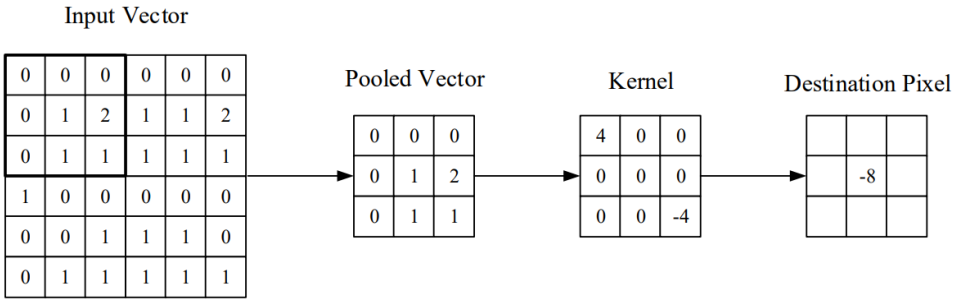


Abbildung 2.5: Beispiel für eine Faltung (engl.: Convolution) [8]

Zunächst wird der Kernel beispielsweise auf die Teilmatrix *links oben* der Eingangsmatrix angewendet. In Abbildung 2.5 ist dies visualisiert. Der betrachtete Teil der Eingangsmatrix wird hier als *Pooled Vector* bezeichnet. Hierauf wird der Kernel, in diesem Fall ein sogenannter *3x3-Kernel*, angewendet. Es wird also das Skalarprodukt der Werte mit dem gleichen Index aus dem *Pooled Vector* und dem Kernel berechnet. In diesem Fall:

2.3 Bildgenerierung mittels Künstlicher Neuronaler Netze

Der Lernfortschritt von Klassifikatoren besteht darin, besser in der Aussage zu werden, ob ein Label y auf einen gegebenen Eingang x zutrifft. Dafür benötigen sie annotierte Trainingsdaten.

Schachalgorithmen sollen hingegen lernen, Züge zu spielen, die die Gewinnchancen des Algorithmus maximieren. Eine weitere Art von KNNs sind *generative Netze*. Sie sollen lernen, neue Daten zu erzeugen, die der Verteilung der Trainingsdaten ähneln. Generative Netze zur Bildgenerierung sollen demnach anhand eines Trainingsdatensatzes lernen, welche Bilder sie erzeugen sollen. Dies lernen sie anhand der statistischen Verteilung der Trainingsdaten. Die Daten sind nicht annotiert, wodurch generative Netze in der Regel in das *Unüberwachte Lernen* einzuordnen sind.

2.3.1 Mathematischer Hintergrund

Generative Netze zur Bilderzeugung sollen beurteilen können, wie wahrscheinlich es ist, dass ein gegebenes Bild aus der Verteilung der Trainingsdaten stammt. Wenn x für jedes mögliche existierende Bild steht, so bilden generative Netze folgende Wahrscheinlichkeitsverteilung ab:

$$\hat{p}(x) \tag{2.4}$$

Für ein gegebenes Bild x gibt $\hat{p}(x)$ einen Schätzwert dafür an, wie wahrscheinlich es ist, dass das Bild aus den Trainingsdaten stammt. Diese Wahrscheinlichkeitsverteilung wird durch das Netz erlernt. Optimiert wird, dass die geschätzte Verteilung der Daten $\hat{p}(x)$ möglichst ähnlich zu der tatsächlichen Verteilung der Trainingsdaten $p(x)$ ist. Ein beispielhafter Vergleich ist in Abbildung 2.6 dargestellt. Es ist erkennbar, dass sich die geschätzte und die tatsächliche Verteilung ähnlich sehen, jedoch nicht identisch sind. Die Abweichung zwischen diesen Verteilungen stellt dabei die Kosten (engl.: den *loss*) dar. Die schwarzen Punkte kennzeichnen Trainingsdaten. Durch sie soll die Verteilung $p(x)$ abgebildet werden. Weniger diversifizierte Trainingsdaten würden sich beispielsweise nur in einem Teilbereich von $p(x)$ befinden. Dadurch könnte das Modell $p(x)$ weniger gut approximieren.

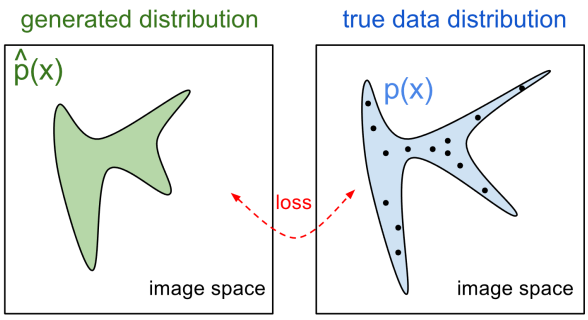


Abbildung 2.6: Beispielhafter Vergleich von $\hat{p}(x)$ und $p(x)$ [9]

Bei der Bildgenerierung versucht das Netz den Wahrscheinlichkeitswert für $\hat{p}(x)$ zu maximieren. Es erlernt durch $\hat{p}(x)$, wie die Verteilung der Trainingsdaten aussieht und versucht anschließend

ausschließlich Bilder zu generieren, die dieser Verteilung folgen. Bezogen auf Abbildung 2.6 befinden sich alle generierten Bilder des trainierten Netzes im grün markierten Bereich.

Es existieren verschiedene Arten generativer Netze. Die Taxonomie, also die Einteilung verschiedener Netze in bestimmte Kategorien, kann Abbildung 2.7 entnommen werden. Einerseits existieren Architekturen, die die Wahrscheinlichkeitsverteilung $\hat{p}(x)$ explizit berechnen. Andere berechnen die Funktion nicht, verwenden sie jedoch implizit. In der Abbildung wird dahingehend zwischen *Explicit Density Models (explizit berechnete Dichte)* und *Implicit Density Models (implizit berechnete Dichte)* unterschieden. Es existieren zudem Unterkategorien, mittels derer eine feinere Kategorisierung durchgeführt wird.

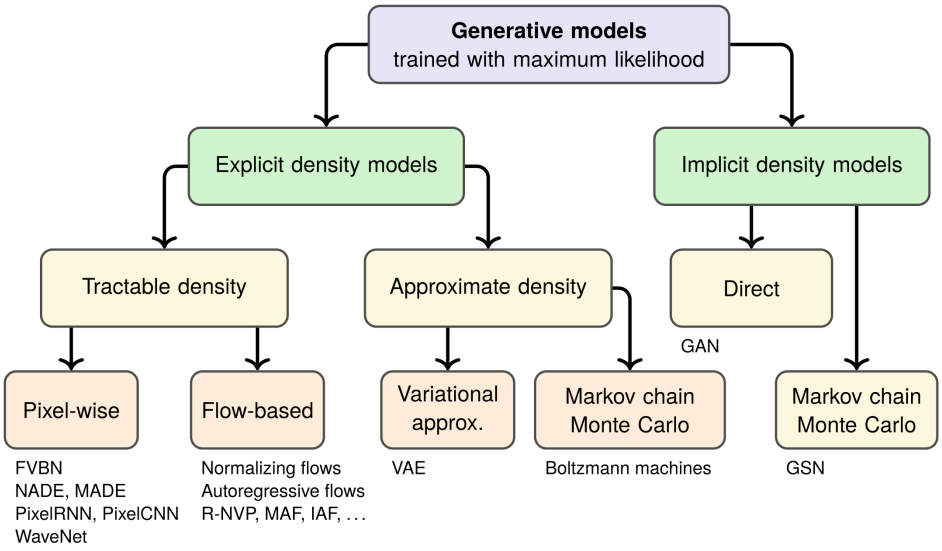


Abbildung 2.7: Taxonomie generativer Modelle [10]

Es soll in diesem Kapitel auf verschiene Architekturen generativer Netze eingegangen werden. Generative Adversarial Networks (GANs) werden gesondert in Kapitel 2.3.3 beschrieben.

2.3.2 Pixel Recurrent Neural Networks

Die Architektur der Pixel Recurrent Neural Networks (PixelRNNs) stammt aus dem Jahr 2016. Diese Netze stützen sich explizit auf die Maximierung der Maximum-Likelihood-Schätzung von $\hat{p}(x)$ für jeden Pixel. Sie sind in der genannten Taxonomie den Modellen zuzuordnen, die den tatsächlichen Schätzwert von $p(x)$ berechnen können. [11]

Im folgenden soll geklärt werden, wie der optimale Wert für jeden Pixel eines generierten Bildes bestimmt wird. Ein betrachtetes Bild x der Auflösung $n \times n$ kann in seine einzelnen

Pixel $(x_1, x_2, ..., x_{n^2})$ aufgeteilt werden. In Gleichung 2.5 ist dabei dargestellt, wie die Wahrscheinlichkeit eines jeden Pixels in die gesamte Verteilung $\hat{p}(x)$ einfließt. [11]

$$\hat{p}(x) = \hat{p}(x_1, x_2, ..., x_{n^2}) = \prod_{i=1}^{n^2} \hat{p}(x_i | x_1, ..., x_{i-1}) \tag{2.5}$$

Jeder Pixel x_i besitzt eine eigene Wahrscheinlichkeitsverteilung $\hat{p}(x_i|x_1, ..., x_{i-1})$. Sie ist abhängig von allen anderen Pixeln $x_1, ..., x_{i-1}$ des Bildes. Der absolut optimale Wert eines Pixels kann demnach nur dann berechnet werden, wenn die Werte aller anderen Pixel bekannt sind. Das Produkt aller Wahrscheinlichkeitswerte der einzelnen Pixel ergibt $\hat{p}(x)$. Soll $\hat{p}(x)$ maximiert werden, so müssen die Terme $\hat{p}(x_i|x_1, ..., x_{i-1})$ möglichst hohe Werte liefern. Daraus ergibt sich dann unter gegebenem Kontext für jeden Pixel eine Maximum-Likelihood-Schätzung. Also der Wert, für den $\hat{p}(x)$ möglichst weit gegen *eins* strebt. [11]

Die Idee von PixelRNNs ist, dass bei der Generierung in einer Ecke des Bildes gestartet wird. Das Bild wird zunächst auf einen Pixel reduziert, der im folgenden x_1 genannt wird. Für diesen Pixel wird ein Wert generiert. Anschließend wird x_1 gemeinsam mit einem benachbarten Pixel x_2 betrachtet. Die Wahrscheinlichkeitsverteilung für x_2 ergibt sich dadurch zu $\hat{p}(x_2|x_1)$. Der Wert für x_2 ist somit nur von x_1 abhängig. Da x_1 bekannt ist, kann ein optimaler Wert für x_2 bestimmt werden. Die Wahrscheinlichkeitsverteilung von x_3 ergibt sich zu $\hat{p}(x_3|x_1, x_2)$, die von x_4 zu $\hat{p}(x_4|x_1, x_2, x_3)$. Das Bild wird sukzessive generiert, wobei der momentane Pixelwert für x_i von allen bisher generierten Pixeln abhängig ist. Dieses vorgehen ist in Abbildung 2.8 dargestellt. Der Wert des rot markierten Pixels hängt von allen blau markierten Pixel ab. Ist für diesen ein Wert bestimmt, wird der rechtsseitig benachbarte Pixel als neues x_i gewählt.

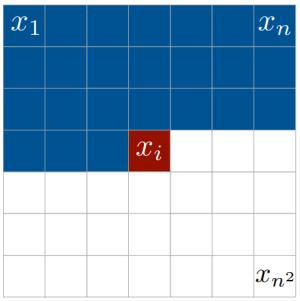


Abbildung 2.8: Bestimmung von $\hat{p}(x)$ mit PixelRNNs [11]

Um die beschriebene Abhängigkeit des momentan generierten Pixels zu allen bisher generierten Pixel umsetzen zu können, besitzen PixelRNNs eine Art *Erinnerung*. Bisher wurden in dieser Arbeit nur sogenannte *Feedforward* Netze behandelt, bei denen der Informationsfluss stets in eine Richtung erfolgt. Nämlich vom Eingang des Netzes zum Ausgang. Es existieren auch *Recurrent Neural Networks*. Sie werden besonders zur Verarbeitung von natürlicher Sprache

eingesetzt (engl.: natural language processing). Abbildung 2.9 bildet hierfür eine beispielhafte Darstellung.

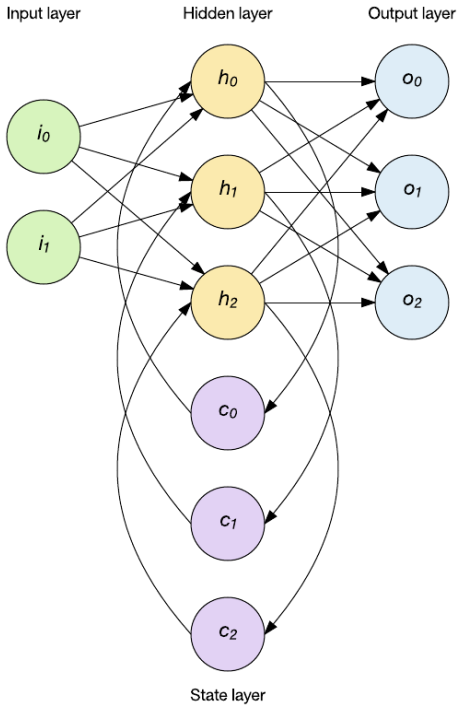


Abbildung 2.9: Darstellung eines Recurrent Neural Networks [12]

In Recurrent Neural Networks spielen die *Zustände* eines Netzes eine besondere Rolle. Ein Zustand wird durch die Eingangs- und Ausgangswerte aller Neuronen zu einem gegebenen Zeitpunkt beschrieben. In PixelRNNs ist der Zustand des Netzes für den Pixel x_2 abhängig von dem Zustand des Netzes für x_1. Um solche Beziehungen darstellen zu können, besitzt das beispielhaft abgebildete Netz die Neuronen c_0 bis c_2, die den vorherigen Wert eines Neurons rekursiv auf seinen Eingang zurückführen. Somit wird der vorherige Zustand des neuronalen Netzes als zusätzlicher Eingang für die Berechnungen genutzt. PixelRNNs nutzen eine besondere Form der Recurrent Neural Networks. Sie arbeiten mit sogenannter *Long Short-term Memory*. Dadurch soll das Problem behoben werden, dass in klassischen Recurrent Neural Networks weit in der Vergangenheit liegende Zustände nur noch einen geringen Einfluss auf den momentanen Zustand haben. [13]

Da die durch ein PixelRNN umgesetzte Verteilung $\hat{p}(x)$ direkt erfassbar ist, wird ihnen nachgesagt, dass die Performanz solcher Netze gut evaluiert werden kann. Es gilt als vergleichsweise leicht, für solche Netze Metriken zur Messung der Performanz umzusetzen. Ein grundlegender Nachteil von PixelRNNs ist, dass die Generierung sequenziell erfolgt. Es ist in dem beschriebenen Verfahren nicht möglich, mehrere Pixel parallel zu generieren, da der Wert eines Pixels von denen aller vorher generierten Pixel abhängig ist. Dies verlangsamt die Generierung, da keine Parallelisierung möglich ist. [13]

Es existieren auch sogenannte *PixelCNNs*, bei denen sich die Berechnung stets nur auf bestimmte Bildbereiche konzentriert. Diese Bildbereiche können parallel zueinander sequenziell berechnet werden. Die Parallelisierung ist jedoch nur während des Trainings des Netzwerks oder während der Evaluation von $\hat{p}(x)$ für gegebene Bilder möglich. Die Bildgenerierung erfolgt auch hier, analog zu PixelRNNs, vollständig sequenziell. [11]

2.3.3 Autoencoder

Das Ziel von Autoencodern ist, den Eingang des Netzes am Ausgang zu rekonstruieren. Dazu setzen sich diese Netze aus drei Bestandteilen zusammen: dem Kodierer, dem latenten Raum und dem Dekodierer. In Abbildung 2.10 ist eine beispielhafte Autoencoderarchitektur dargestellt.

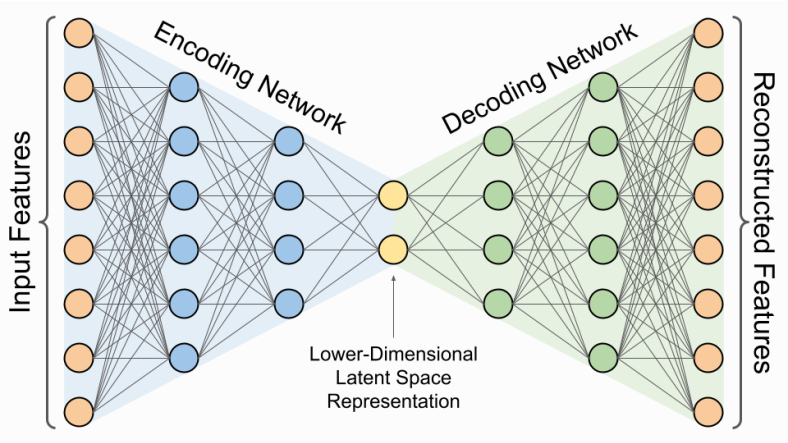


Abbildung 2.10: Architektur eines Autoencoders

Der **Kodierer** besitzt die Aufgabe, Merkmale aus dem Eingang des Netzes zu extrahieren. Diese Merkmale sollen daraufhin mit einer begrenzten Anzahl an Parametern durch den latenten Raum repäsentiert werden. Somit besteht die Aufgabe des Kodierers darin, den Eingang auf seine für das Netz wesentlichen Eigenschaften zu reduzieren. Und zwar erfolgt die Komprimierung dabei so, dass die Informationen gerade so durch den latenten Raum dargestellt werden können.

Bei dem **latenten Raum** handelt es sich um eine einzelne Schicht von Parametern, oder in diesem Kontext: Neuronen. Je mehr Neuronen sich in dem latenten Raum befinden, desto mehr Informationen können von der Kodierung an die Dekodierung übertragen werden. Die Merkmale, die sich in dem latenden Raum befinden, können durch einen Vektor $\vec{z}$ beschrieben werden. Der latente Raum sollte klein genug sein, damit dort nicht alle Merkmale des Eingangs gespeichert werden können. So wird der Kodierer dazu gezwungen, vereinzeilte Merkmale zu extrahieren.

Der **Dekodierer** nutzt die Werte aus dem latenten Raum, um den Eingang nachzubilden. Diese Nachbildung stellt den Ausgang des Autoencoders dar. Die Kosten eines Autoencoders können durch die Abweichung zwischen Ein- und Ausgang festgestellt werden.

Da der Zweck von Autoencodern darin besteht, einen Eingang auf seine relevanten Merkmale zu reduzieren, wird der Dekodierer nur während des Trainings verwendet. Beim praktischen Einsatz eines Autoencoders werden lediglich der Kodierer und der latente Raum eingesetzt. Im Gegensatz zu PixelRNNs basieren Autoencoder klassischerweise nicht auf Recurrent Neural Networks, sondern auf Feedforward Netzen.

Diese beschriebene Architektur der Autoencoder ist nicht für die generative Modellierung geeignet, da sie deterministisch ist. Erhält das Modell bestimmte Eingangswerte, so liefert es stets die gleichen Ausgangswerte. Es versucht den Eingang möglichst zu rekonstruieren, wobei der Inhalt $\vec{z}$ des latenten Raums für ein gegebenes Eingangsbild stets gleich ist. Eine zufällige Erzeugung neuer Bilder ist damit nicht möglich. Die sogenannten *Variational Autoencoder* sind eine Architektur, die für die Generierung neuer Daten verwendet werden können. [7]

Variational Autoencoder verfolgen das Ziel, eine Zufallskomponente in die Bilderzeugung einfließen zu lassen. Ein entscheidender Unterschied zu klassischen Autoencodern ist deshalb der folgende: Ein gegebener Eingang x wird auf kein festes $\vec{z}$ kodiert, sondern auf eine Wahrscheinlichkeitsverteilung. Sie wird bezeichnet als:

$$p(z|x) \tag{2.6}$$

Im Gegensatz zu PixelRNNs versucht das Netz somit nicht die Verteilung der Trainingsdaten $p(x)$ zu approximieren, sondern die Verteilung der Merkmale $\vec{z}$ der Trainingsdaten x. Es wird angenommen, dass jedes Merkmal normalverteilt ist. Damit kann jede Komponente z_i des Vektors $\vec{z} = [z_1, z_2, ..., z_n]^\mathsf{T}$ durch eine Gaußsche Normalverteilung $\mathcal{N}(\mu_i, {\sigma_i}^2)$ beschrieben werden. Aufgabe des Kodierers ist damit nicht mehr, aus einem gegebenen x eine Menge von Merkmalen $\vec{z}$ zu bestimmen. Stattdessen soll der Kodierer die Vektoren $\vec{\mu}$ und $\vec{\sigma}$ bestimmen, durch die sich die einzelnen Normalerteilungen von $\vec{z}$ beschreiben lassen.

An den Dekodierer wird ein zufällig aus der Verteilung $p(z|x)$ entnommenes Set an Merkmalen $\vec{z}$ übergeben. Der Dekodierer übersetzt dieses gegebene $\vec{z}$ daraufhin in ein Bild. Im praktischen Einsatz werden bei einem Variational Autoencoder nur der latente Raum und der Dekodierer genutzt. Der Dekodierer erhält zufällige Werte für $\vec{z}$, also zufällige Merkmale, und generiert daraus ein Bild.

Vor- Nachteile?

[14]

2.3.4 Generative Adversarial Networks

Eine weitere Architektur, die zur Bildgenerierung verwendet werden kann, sind die sogenannten *Generative Adversarial Networks* (GANs). Sie wurden unter anderem von Ian Goodfellow im Jahre 2014 entwickelt.

Ein GAN besteht aus zwei Komponenten. Dem *Generator* und dem *Discriminator*. Der Generator erzeugt aus einem zufälligen Eingangsvektor ein Bild. Der Discriminator erhält ein Bild als Eingang und gibt entweder den Wert *0* oder den Wert *1* aus. Er soll erkennen, ob das gegebene Bild aus den Trainingsdaten stammt, oder ob es künstlich generiert wurde. Bei sowohl dem Generator als auch dem Discriminator handelt es sich um KNNs. Diese beiden Komponenten werden so miteinander gekoppelt, dass der Discriminator stets entweder ein erzeugtes Bild des Generators oder ein Bild aus den Trainingsdaten erhält. Dies ist in der nachfolgenden Abbildung beispielhaft dargestellt:

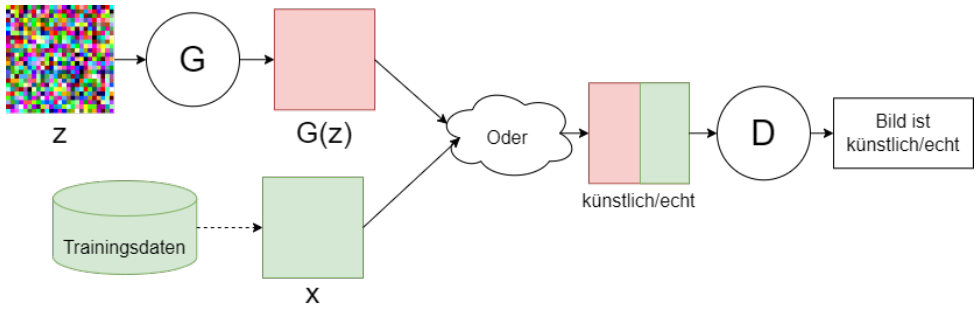


Abbildung 2.11: Zusammenspiel zwischen Generator und Discriminator

Der Generator G erhält einen zufälligen Eingangsvektor z. Letzterer kann als ein weißes Rauschen beschrieben werden. Das generierte Bild ist das Resultat der Funktion $G(z)$ des Generators G. Anschließend wird dem Discriminator D entweder das generierte Bild $G(z)$ oder ein Bild x aus den Trainingsdaten gezeigt. Der Ausganswert des Discriminators ist daraufhin die Einschätzung, ob das gezeigte Bild echt oder künstlich generiert ist. Ziel des Generators ist, dass die Verteilung der künstlichen Daten $p(G(z))$ möglichst ähnlich der Verteilung $p(x)$ ist. Der Discriminator soll die beiden Verteilungen möglichst gut voneinander unterscheiden können. Somit handelt es sich um ein direktes Gegenspiel zwischen Generator und Discriminator. Sie versuchen sich gegenseitig zu überlisten.

Training Dies spiegelt sich auch im Training von GANs wider. $V(D, G)$ stellt die zu optimierende Funktion dar: [15]

$$\min_G \max_D V(D, G) = \mathbb{E}_x[\log(D(x))] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{2.7}$$

Bei dem Training handelt es sich um ein Min-Max-Problem. Der Generator versucht die Funktion $V(G, D)$ zu minimieren, wohingegen der Discriminator sie zu maximieren versucht. Der Term 2.8 stellt die Fehlerrate des Discriminators auf den echten Trainingsdaten dar. Mit anderen Worten: Wie viele echte Bilder er als unecht klassifiziert. Der Term 2.9 beschreibt, wie viele unechte Daten der Discriminator als echt klassifiziert. [7]

$$\mathbb{E}_x[\log(D(x))] \tag{2.8}$$

$$\mathbb{E}_z[\log(1 - D(G(z)))] \tag{2.9}$$

Der Term 2.8 ist nicht vom Generator abhängig, sondern lediglich von $D(x)$. Hierauf hat der Generator keinen Einfluss, wodurch er alleine durch diesen Teil der Kostenfunktion nicht trainiert wird. Der Discriminator besitzt somit zwei Terme, die ihn trainieren, während der Generator nur einen besitzt. Aus diesem Grund werden dem Discriminator in der Regel doppelt so viele unechte Daten wie echte Trainingsdaten gezeigt. Dies soll einem ungleichen Training der beiden Komponenten des GANs entgegenwirken. Der Optimalzustand eines GANs ist, dass der Discriminator so gut wie möglich identifizieren kann, ob ein gegebenes Bild aus $p(x)$ stammt, während der Generator dennoch in der Lage ist, den Discriminator zu überlisten. Das Training von GANs gilt als empfindlich gegenüber den gewählten Hyperparametern und der Netzwerkarchitektur. GANs wird nachgesagt, dass kleine Änderungen in den beiden genannten Aspekten die Qualität der Generierten Bildern signifikant beeinflussen können. [7]

Im praktischen Einsatz wird nur der Generator des GANs verwendet. Der Discriminator wird ausschließlich dazu eingesetzt, mit $p(G(z))$ möglichst gut $p(x)$ zu approximieren, sodass die generierten Bilder im Optimalfall nicht von echten Trainingsdaten zu unterscheiden sind. [7]

Die bisher beschriebene Architektur von GANs wird auch als *Vanilla GAN* bezeichnet. Dies entspricht dem, wie GANs in Goodfellows Publikation definiert werden [15]. Forscher und Anwender haben seit dieser Veröffentlichung verschiedene Limitationen und Probleme bei Vanilla GANs feststellen können. Insbesondere im Hinblick auf spezielle Einsatzgebiete. Ein hier häufig anzutreffender Begriff ist *Modal Collaps*. Damit ist die Situation gemeint, dass der Generator bei beliebigem Input stets dasselbe Bild generiert. Er lernt, dass ein bestimmtes Bild den Discriminator überlisten kann und generiert es deshalb jedes Mal, egal welchen Input man ihm zuführt. Dies ist für diesen Anwendungsfall insofern von Relevanz, alsdass der Generator verschiedene Arten von Straßenschildern generieren soll, wobei jedes Bild zusätzlich einen unterschiedlichen Hintergrund besitzen soll. Lösen lässt sich das Problem des *Modal Collaps* beispielsweise mit sogenannten Cycle-Consistent Generative Adversarial Networks (CycleGANs). [7]

CycleGANs Ein CycleGAN besteht aus zwei miteinander gekoppelten GANs. Diese werden klassischerweise als G und F bezeichnet. Bei G handelt es sich um ein Vanilla GAN, so wie in diesem Kapitel bisher beschrieben. Erweitert wird das Netzwerk jedoch so, dass es den Output von G an F weitergibt. F erhält somit das von G künstlich generierte Bild als Input. Aufgabe von F ist, daraus den ursprünglichen Input, der G zugeführt wurde, zu reproduzieren. Somit soll das gesamte Netzwerk nicht nur neue Bilder generieren können, sondern soll auch von einem generierten Bild zurück auf den zugeführten Input schließen können. Dies lässt sich mathematisch so darstellen, dass G und F folgende Abbildungen implementieren:

$$\mathbf{G} : X \mapsto Y \ \wedge \ \mathbf{F} : Y \mapsto \tilde{X} \tag{2.10}$$

Das Modell G erzeugt somit aus einem gegebenen X ein Y, wohingegen F aus dem Y auf das X schließen soll. Die Behebung des Modal Collaps findet dadurch statt, dass das Netzwerk den Output $\tilde{X}$ von F überprüft und diesen mit dem tatsächlichen Input X vergleicht. Es wird überprüft, wie ähnlich sich $\tilde{X}$ und X sind. Liegt eine zu hohe Diskrepanz vor, kann das Netzwerk darauf schließen, dass G Outputs erzeugt, die nicht in direkter Abhängigkeit zu X stehen.

CycleGANs sind für spezielle Anwendungsgebiete gedacht, in denen ausgewählte, und somit nicht-zufällige Eingabewerte verwendet werden. Dazu zählen Gebiete wie *Style Transfer* oder die Transformation von Bildern, respektive Bildelementen. Diese Studienarbeit stellt eine solche Problemstellung dar, da das auf dem generierten Bild gezeigte Straßenschild durch den Eingang des Netzwerks definiert wird.

Das Training von CycleGANs basiert auf mehreren Kostenfunktionen. Die GANs G und F besitzen jeweils einen eigenen *Adversarial Loss*. Damit ist die in Gleichung 2.7 beschriebene Kostenfunktion eines Vanilla GANs gemeint. Im Kontext von CycleGANs ist sie, für das GAN G wie folgt definiert:

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_y[\log D_Y(y)] + \mathbb{E}_x[\log(1 - D_Y(G(x)))] \tag{2.11}$$

Die Funktion für GAN F ist identisch, mit dem Unterschied, dass sie von F und D_X statt von G und D_Y abhängt.

Als weitere Kostenfunktion besitzen CycleGANs einen *Cyclic Loss* (Gleichung 2.12). Die beiden Summanten der Gleichung setzen sich daraus zusammen, wie weit die Pixelwerte von den generierten Bildern und den echten Bildern auseinander liegen. Mit $G(x)$ wird ein Bild $\tilde{y}$ generiert, F generiert anschließend aus diesem $\tilde{y}$ wieder ein $\tilde{x}$. Wenn $\tilde{x}$ und x möglichst ähnlich sind, dann kann davon ausgegangen werden, dass die generierten Bilder des Netzwerks

G in direkter Abhängigkeit von dem Input X stehen. Was hierbei berechnet wird, ist die durschnittliche, absolute Abweichung der Pixelwerte.

$$L_{cyc}(G, F) = \mathbb{E}_x[||F(G(x)) - x||_1] + \mathbb{E}_y[||G(F(y)) - y||_1] \tag{2.12}$$

Um die gesamten Kosten des CycleGANs zu erhalten, werden die bisher beschriebenen Kostenfunktionen addiert. Der *Cyclic Loss* $L_{cyc}(G, F)$ wird dabei mit einem absoluten Wert λ multipliziert, um die Kosten dieser Funktion im Vergleich zu den *Adversarial Losses* gewichten zu können. In der Veröffentlichung der CycleGANs wird ein λ von 10 verwendet. Somit wird mit einem vergleichweise hohen Gewicht versehen, dass die generierten Bilder in direkter Abhängigkeit zu den Eingangswerten des CycleGANs stehen. Der Wert λ stellt einen Hyperparamer dar. [16]

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda \cdot L_{cyc}(G, F) \tag{2.13}$$

[16]

Wasserstein GANs **Auf Taxonomie eingehen**

2.4 Vorherige Arbeiten

Durch eine Recherche haben sich zwei Arbeiten gezeigt, die sich, analog zu dieser Studienarbeit, mit der künstlichen generierung von Straßenschildern mittels KNNs beschäftigen. Beide Arbeiten konzentrieren sich darauf, Bildausschnitte zu erzeugen, die ein Straßenschild zeigen und eine geringfügige Menge an Hintergrund um das Schild.

2.4.1 Generierung Taiwanischer Straßenschilder mittels DCGAN

Eine der beiden Arbeiten wurde im Jahr 2021 veröffentlicht. Sie konzentriert sich auf die Generierung taiwanischer Straßenschilder. Dafür wird ein *DCGAN* verwendet, ein GAN, das im Generator und im Discriminator eine tiefe CNN Architektur besitzt. Getestet wird, inwiefern künstlich generierte Trainingsbilder die Erkennung von Straßenschildern verbessern können. Es werden in der Arbeit vier Arten von Verkehrsschildern generiert. [17]

Für jede der vier Klassen wird das GAN mit 350 Bildern trainiert. Die Bildgrößen variieren dabei, wobei die Maximalgröße bei 200x200 Pixel liegt. Die generierten Bildgrößen korrespondieren zu denen der Trainingsbilder. Es sollen in der Arbeit bewusst keine größeren Bilder als 200x200 Pixel erzeugt werden, da Straßenschilder häufig nur einen kleinen Teil des Sichtfelds auf der

Straße ausmachen. Das Training erstreckt sich auf bis zu 2000 Epochen, was bedeutet, dass das GAN während des Trainings 2000 mal alle Trainingsbilder als Eingabe erhält. [17]

Da die Anzahl an Trainingsbildern beschränkt ist, generiert das Modell keine völlig neuartigen Bilder, sondern für jede Klasse jeweils vergleichsweise ähnlich aussehende:

Abbildung 2.12: Beispielergebnisse der Generierung taiwanischer SChilder [17]

Zur Beurteilung der Generierung wird mitunter der sogenannte Index struktureller Ähnlichkeit (eng.: Structural Similarity Index) (SSIM) verwendet. Statt dass beispielsweise die Differenz aller entsprechenden Pixelwerte berechnet wird, werden hier die Aspekte *Kontrast*, *Leuchtdichte* und *Struktur* der generierten und der echten Bilder verglichen. Dafür werden keine Berechnungen mit einzelnen Pixelwerten durchgeführt, sondern es wird mit den Mittelwerten und der Standardabweichung der Pixelwerte gerechnet. Auf die zugehörigen Formeln wird in Kapitel 6 eingegangen. [17]

Der Nutzen der generierten Bildern wird anhand eines Modells zur Objektdetektion getestet. In dem Fall, ein sogenanntes *YOLO* Modell. Die Detektion erfolgt auf größeren Bildern, auf denen mehrere Straßenschilder zu sehen sind. Für das Training des Modells werden mit etwa gleicher Gewichtung die für das GAN verwendete Trainingsdaten und generierte Daten des GANs verwendet. Zur Evaluation wurden hierbei Bilder verwendet, die insgesamt 40 Straßenschilder beinhalten. Die Ergebnisse können folgender Tabelle entnommen werden: [17]

Modell	Reale Trainingsbilder?	Künstliche Trainingsbilder?	Genauigkeit
Densenet	Ja	Ja	92%
Resnet	Ja	Ja	91%
Densenet	Ja	Nein	88%
Resnet	Ja	Nein	63%

Tabelle 2.1: Vergleich der Objekterkennung mit und ohne künstliche Trainingsdaten

Das Resultat ist, dass die Erkennung durch die generierten Trainingsdaten verbessert wird. Sowohl das Densenet als auch das Resnet liefern durch sie genauere Ergebnisse. [17]

2.4.2 Generierung Deutscher Straßenschilder mittels CycleGAN

Eine weitere Publikation, die sich mit der künstlichen Generierung von Bildern mit Straßenschildern konzentriert, verwendet einen Datensatz, der deutsche Straßenschilder enthält. [18] [19] Auf den Datensatz wird näher in Kapitel 3 eingeganen, da er auch die Basis für diese Arbeit bildet. Die genannte Publikation ist aus einer Masterarbeit an der Ruhr Universität Bochum entstanden. Dort wurde ebenfalls der genutzte Datensatz, der German Traffic Sign Recognition Benchmark (GTSRB), veröffentlicht. Die Netzwerkarchitektur ist hier eine andere als bei der bisher beschriebenen Generierung taiwanischer Schilder. In dieser Arbeit wird ein CycleGAN statt eines *Vanilla GANs* verwendet. Die beiden Generatoren basieren auf einem Resnet. [18]

Für die Generierung erhält das Netzwerk das Piktogramm eines Straßenschilds als Eingang. Das CycleGAN soll einen möglichst realistisch wirkenden Hintergrund um das Straßenschild erzeugen. Vor der Generierung werden die Piktogramme der Straßenschilder zufällig rotiert und ein zufälliger einfarbiger Hintergrund erzeugt. Letzteres soll eine weitere stochastische Komponente für die Generierung bilden, damit der generierte Hintergrund nicht zu sehr von der Art des Straßenschilds abhängt. Für das Training des Netzes wird eine präparierte Version des GTSRB verwendet. Sie beinhaltet folgende Besonderheiten:

- Nur Bilder mit einer Mindestauflösung von 64x64 Pixeln werden verwendet
- Die Bilder werden so zugeschnitten, dass sie quadratisch sind
- Die Klassen werden ausbalanciert, um der asymmetrischen Verteilung an Trainingsbildern pro Klasse entgegenzuwirken
- Insgesamt besteht der präparierte Datensatz aus 12.212 Bildern

[18]

generierte Beispiele einfügen

Es erfolgt eine Evaluation, inwiefern die künstlich generierten Trainingsbilder zwei verschiedene Klassifikatoren verbessern können. Dabei handelt es sich um eine sogenannte Support Vector Machine (SVM) und ein CNN. SVMs sind eine Art von trainierbaren Klassifikatoren, die nicht auf KNNs basieren [7]. Einerseits werden die Algorithmen mit realen Trainingsdaten trainiert und andererseits vollständig mit generierten Daten. Die Ergebnisse bezüglich der Genauigkeit der Klassifikation je nach der Art der Trainingsbilder ist in Tabelle **??** dargestellt. Es lässt sich erkennen, dass die Klassifikation um jeweils etwa 7-9% ungenauer ist, als mit realen

Trainingsdaten. Es ist jedoch auch erwartbar, dass die Genauigkeit etwas geringer ausfällt, da die generierten Bilder der Verteilung des echten Trainingsdatensatzes folgen sollen, dies jedoch nicht zu 100% möglich ist. [18]

Modell	Reale Trainingsbilder?	Künstliche Trainingsbilder?	Genauigkeit
CNN	Ja	Nein	95,42%
SVM	Ja	Nein	87,97%
CNN	Nein	Ja	87,57%
SVM	Nein	Ja	79,27%

Tabelle 2.2: Vergleich der Klassifikation mit echten und künstlichen Trainingsdaten [18]

2.5 Machine Learning Frameworks

2.5.1 Tensorflow

2.5.2 Pytorch

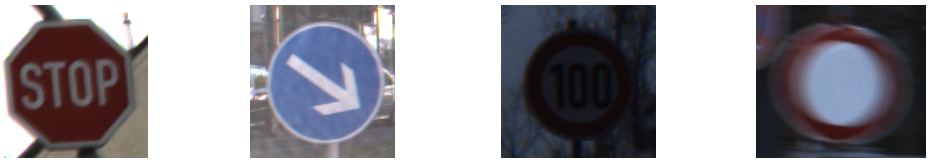
2.5.3 Weitere

3 Konzeption des Modells

3.1 Datensatz

Analog zu der in Kapitel 2.4.2 beschriebenen Arbeit wird der GTSRB als Datensatz für diese Studienarbeit verwendet. Dies hängt mit der Größe des Datensatzes zusammen, mit der vergleichsweise kleinen Auflösung der einzelnen Bilder und damit, dass sich diese Arbeit auf die Genereriung deutscher Straßenschilder beschränkt. Es soll jedoch in Kapitel 6 ebenfalls geprüft werden, inwiefern das Resultat der Arbeit genutzt werden kann, um Bilder zu generieren, die Straßenschilder aus anderen Ländern zeigen. Länder, die das Wiener Übereinkommen über Straßenschilder unterzeichnet haben, besitzen vergleichsweise ähnlich aussehende Straßenschilder. Durch das Übereinkommen wird mitunter die grundlegende Form und Farbe verschiedener Schilder bestimmt. [19] [20]

Der GTSRB besteht aus mehr als 50.000 Bildern verteilt auf 43 Klassen respektive 43 verschiedenen Arten von Straßenschildern. Beispiele aus dem Datensatz sind in folgender Abbildung dargestellt. [19]



(a) (b) (c) (d)

Abbildung 3.1: Beispielbilder aus dem GTSRB Datensatz [19]

Die Bilder des Datensatzes besitzen unterschiedliche Seitenverhältnisse und verschiedene Auflösungen. Ein Großteil der Bilder ist jedoch kleiner als 100x100 Pixel. Die Bilder basieren auf Videos, die durch die Autoren tagsüber im Straßenverkehr aufgenommen wurden. Die Trainingsbilder sind ungleich auf die Anzahl an Klassen verteilt. Dies hängt mitunter damit zusammen, dass die jeweiligen Schilder nicht gleich häufig im Straßenverkehr verwendet werden. Unterteilt wird der Datensatz in 39.209 Trainingsbilder und 12.630 Testbilder. Da das Ziel dieser Arbeit jedoch nicht ist, einen Klassifikator mit den generierten Bildern zu trainieren, können auch die Testbilder für das Training des Netzes verwendet werden. Es wird nicht zwischen Trainings- und Testbildern unterschieden. [19]

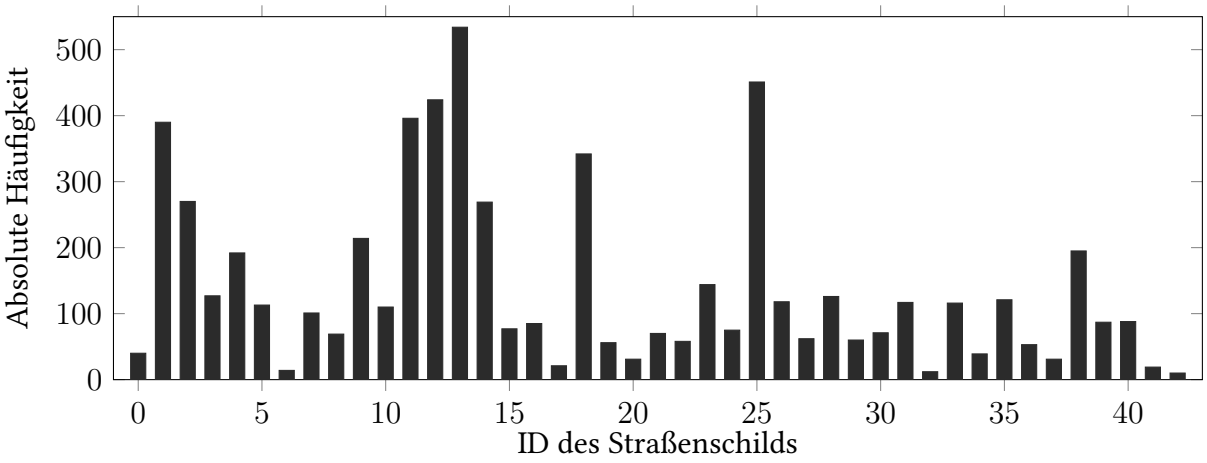


Abbildung 3.2: Häufigkeitsverteilung der Arten von Straßenschildern im präparierten Datensatz

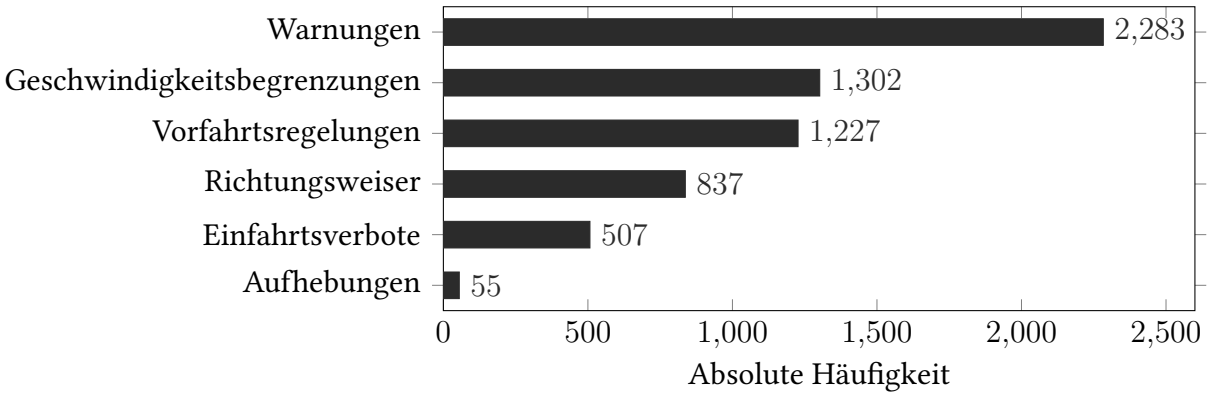


Abbildung 3.3: Häufigkeitsverteilung der Kategorien von Straßenschildern im präparierten Datensatz

Der Datensatz wird in dieser Arbeit zunächst so präpariert, dass nur Bilder für das Training verwendet werden, die mindestens 50 Pixel breit oder hoch sind. Dies wird im Verlauf geändert, sodass die Mindestgröße 75 Pixel beträgt.

Die Verteilung der Daten ist auch im präparierten Datensatz nicht homogen. Die nachfolgenden Diagramme zeigen die Verteilung der Daten innerhalb des präparierten Datensatzes. Für die Erhebung mussten die Testdaten manuell ihren Klassen zugeordnet werden.

3.2 Framework

Tensorflow Addons

Tensorflow Graphics Erklären: Wieso benutze ich, wo möglich, Tensorflow Graphics statt OpenCV? Wieso gibt es Tensorflow Graphics überhaupt?

OpenCV Erwähnen: OpenCV wird nur da benutzt, wo keine Tensorflow funktionen verwendet werden können. Beeinträchtigt die Performance.

3.3 Architektur

In Kapitel 2.3 werden verschiedene generative Netzwerkarchitekturen vorgestellt. Zunächst soll darauf eingegangen werden, welche dieser Herangehensweisen gewählt wird. Jede der Architekturen besitzt ihre Vor- und Nachteile. Der größte Nachteil von PixelRNNs ist, dass die Pixel eines Bildes nicht parallel zueinander generiert werden können. Dies verlangsamt die Generierung und damit das Training des Modells.

Entscheidungsmatrix?

3.4 Datenaugmentation

Bevor die Piktogramme an den Generator übergeben werden, werden sie zufällig rotiert. Dadurch muss der Generator die Rotation nicht eigenständig lernen und dieser Aspekt der Generierung lässt sich deterministisch bestimmen. Dabei soll die Rotation nicht nur in x-y-Richtung erfolgen, sondern auch eine dreidimensionale Rotation simuliert werden. Und zwar so, als sei das Schild aus einer beliebigen Frontalperspektive aufgenommen worden.

Um bestimmte Transformationen eines Bilds mittels einer Matrixpultiplikation darstellen zu können, wird häufig ein sogenanntes *homogenes Koordinatensystem* verwendet. Dabei wird das Koordinatensystem um eine weitere Dimension erweitert. Ein Punkt $p = [x, y]^\mathsf{T}$ kann somit um einen beliebigen Wert in z-Richtung verschoben werden. Dadurch wird ein Punkt $\tilde{p}$ im homogenen Koordinatensystem durch drei Koordinaten $\tilde{x}$, $\tilde{y}$ und $\tilde{z}$ beschrieben. Transormationen werden in der homogenen Darstellung durchgeführt und anschließend werden daraus die kartesischen Koordinaten x und y bestimmt. Somit erhält man aus der Transformation erneut ein zweidimensionales Bild. [21] [22]

Dies wird für eine dreidimensionale Rotation der Piktogramme benötigt. Die Rotation soll durch drei *eulersche Winkel* beschrieben werden. Das bedeutet, dass sie sich aus einer Rotation um die z-Achse, einer um die y-Achse und einer um die x-Achse zusammensetzt. Dies ist in der nachfolgenden Grafik abgebildet. Die bläulichen Balken zeigen dabei die Achse an, um die gedreht wird. Die erste Rotation ist um die z-Achse, wodurch der Balken in die dritte Bildebene geht. [22]

Abbildung 3.4: Rotation der Straßenschilder mittels eulerscher Winkel

Jede Rotation ist durch einen einzelnen Winkel um die jeweilige Achse bestimmt. Kombiniert man die Rotationen, kann die resultierende Tranformation somit durch drei Winkel $(\alpha_z, \alpha_y, \alpha_x)$ eindeutig beschrieben werden. Für die Erzeugung einer zufälligen Rotation müssen randomisierte Werte für diese Winkel bestimmt werden. [22]

Zusätzlich zu der Rotation, soll das Modell die Piktogramme zufällig in ihrer Größe skalieren. Die genannten Augmentationen dienen dazu, die Verteilung der real aufgenommenen Schilder abbilden zu können. Im Datensatz besitzen die Schilder eine unterschiedliche Größe und sind aus verschiedenen Perspektiven aufgenommen. Dadurch dass die Augmentation deterministisch ist, kann sie dazu genutzt werden, um gezielt nur Bilder durch das Modell zu generieren, die aus bestimmten Perspektiven und mit festgelegten Größen generiert wurden. Alternativ kann auch die randomisierte Augmentation beibehalten werden, um eine möglichst große Bandbreite an unterschiedlichen Bildern zu erzeugen.

3.5 Training

Überlicherweise soll beim Training von neuronalen Netzen die Verlustfunktion gegen den Wert *null* streben. Bei diesem Modell soll die Verlustfunktion jedoch gegen einen Wert konvergieren, der größer ist als *null*. In diesem Fall schaffen es weder die Generatoren, die Funktionswerte zu minimieren, noch können die Diskriminatoren die Funktionswerte weiter maximieren. In diesem Fall ist das sogenannte *Nash-Gleichgewicht* erreicht. Das Training ist somit beendet, da sich das Modell mit den gegebenen Daten nicht weiter verbessern kann.

4 Implementierung und Training

Viele Trainingsbilder haben nicht das passende Seitenverhältnis. Sie einfach so zu laden würde dazu führen, dass die verzerrt werden. load dataset from directory bietet Parameter an, der dafür sorgt, dass die Bilder gecropppt werden. Central crop, genau das, was ich möchte.

Trainingsbilder die kleiner als 256x256 Pixel sind (was die meisten sind) werden bei der Vergrößerung durch tensorflow vergälttet. Dadurch sind sie nicht verpixelt, sondern nur etwas verschwommen. Gut so.

4.1 Hyperparameter

5 Augmentation der generierten Bilder

Die Augmentation der generieten Bilder wird durch das Python Modul `utils.image_augmentation` implementiert.

5.1 Ungültige Straßenschilder

In Kapitel 2.1 ist bereits beschrieben, dass als ungültig markierte Schilder offenbar eine Herausforderung für heutige Straßenschilderkennungen darstellen können. Das Ziel dieser Studienarbeit ist, solche Fälle simulieren zu können. Aus diesem Grund ist dieser Anwendungsfall implementiert.

Beispiele für durch dieses Projekt erzeugte, ungültige Straßenschilder sind in der nachfolgenden Abbildung dargestellt:

Bei der Umsetzung bieten sich zwei Möglichkeiten. Zum einen kann das CycleGAN darauf trainiert werden, solche Bilder eigenständig zu generieren. Dafür würden Trainingsdaten benötigt, die solche Schilder zeigen. Der Datensatz müsste somit um weitere Bilder ergänzt werden. Weitere reale Bilder hinzuzufügen ist nicht ohne weiteres möglich. Es wäre jedoch auch denkbar, bereits vorhandene TRainingsbilder mit einer Bildbearbeitungssoftware so anzupassen, dass die ungültige Schilder zeigen.

Eine weitere Möglichkeit ist, das Kreuz, das die Ungültigkeit eines Schildes markiert, nachträglich in die generierten Bilder einzufügen. Die Schwierigkeit ist hierbei, dass die Straßenschilder zufällig rotiert und skaliert sind. Das Kreuz muss so transformiert werden, dass es sich stest zentral und mit einer angepassten Rotation auf dem Schild befindet.

5.2 Bewegungsunschärfe

Verwackelte Bilder können insbesondere dann entstehen, wenn sich das Fahrzeug mit einer hohen Geschwindigkeit bewegt. Hierbei kann eine sogenannte Bewegungsunschärfe auftreten. Dies tritt bei einer Kamera auf, wenn sich das Bild während der Belichtungszeit deutlich verändert. Wenn also die Geschwindigkeit des Fahrzeugs groß ist im Vergleich zur Belichtungszeit.

Darauf eingehen, dass man das auch mit einer Fourier Transformation umgesetzen kann

Eine Bewegungsunschärfe kann mittels einer Faltung des Bildes mit einer Faltmatrix realisiert werden.

5.3 Schnee

Die Generierung von Schnee erfolgt in mehreren Schritten:

1. Erstelle ein Bild, das aus zufälligen schwarzen und weißen Pixeln besteht. Die Anzahl an weißen Pixel soll kleiner sein als die der schwarzen Pixel.
2. Führe auf dem Bild ein Gaußsches Weichzeichen aus. Hierdurch verschmieren die einzelnen weißen Pixel zu größeren Punkten.
3. Führe auf dem Bild eine Bewegungsunschärfe aus. Dadurch wird die Bewegung der Schneeflocken entlang einer Windrichtung simuliert.
4. Mache den schwarzen Hintergrund transparent und füge das erstellte Bild auf ein generiertes Straßenschild-Bild ein.

TODO: Datei einbinden und so linenumbers fixen

```
1 def add_snow(img_tensor, snow_intensity, motion_blur_intensity,
  ↪  motion_blur_direction, p_snowflake_min=0.02, p_snowflake_max=0.5):
```

Listing 5.1: Hizufügen von Schnee: Funktionsdeklaration

Test Test

6 Evaluation

Die Aufgabe der Evaluation ist prinzipiell folgende: Es soll gemessen werden, wie ähnlich die Wahrscheinlichkeitsverteilung der generierten Bilder zu der Verteilung der Trainingsbilder ist. Außerdem soll bestimmt werden, wie realistisch die generierten Bilder sind.

6.1 Evaluation der Generierung

6.1.1 Farbhistogramme

6.1.2 Wasserstein-Distanz

6.1.3 Klassifizierung

6.2 Evaluation der Augmentierung

6.2.1 Mean Opinion Score

7 Zusammenfassung

Literatur

[1] M. Staron, „AUTOSAR (AUTomotive Open System ARchitecture),“ in *Automotive Software Architectures: An Introduction*. Cham: Springer International Publishing, 2021, 97ff. ISBN: 978-3-030-65939-4. DOI: 10.1007/978-3-030-65939-4_5.

[2] EU-Kommission, *Regulation (EU) 2019/2144 of the European Parliament and of the Council*, Art. 6 Abs. 2c, 5. Sep. 2021.

[3] H. Ippen und M. Bach, „Verkehrsschild-Erkennung Test,“ *Autozeitung*, 2. Apr. 2019.

[4] A. Gudigar, S. Chokkadi und R. U, „A review on automatic detection and recognition of traffic sign,“ *Multimedia Tools and Applications*, Jg. 75, S. 333–364, 2016. DOI: 10.1007/s11042-014-2293-7.

[5] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] D. Sonnet, *Neuronale Netze Kompakt, Vom Perceptron zum Deep Learning*. Wiesbaden: Springer Vieweg Wiesbaden, 2020. DOI: 10.1007/978-3-658-29081-8.

[7] A. S. Glassner, „Deep Learning: A Visual Approach,“ in San Francisco: No Starch Press, 2021, ISBN: 978-1-7185-0072-3.

[8] K. O'Shea und R. Nash, *An Introduction to Convolutional Neural Networks*, 2015. DOI: 10.48550/ARXIV.1511.08458.

[9] A. Karpathy, P. Abbeel, G. Brockman u. a., *Generative Models*, https://openai.com/blog/generative-models/, Letzter Zugriff: 13.01.2023, 16. Juni 2016.

[10] S. I. Nikolenko, „Generative Models in Deep Learning,“ in *Synthetic Data for Deep Learning*. Cham: Springer International Publishing, 2021, S. 97–137. DOI: 10.1007/978-3-030-75178-4_4.

[11] A. van den Oord, N. Kalchbrenner und K. Kavukcuoglu, „Pixel Recurrent Neural Networks,“ *CoRR*, 2016. DOI: 10.48550/arXiv.1601.06759.

[12] M. T. Jones, *Recurrent neural networks deep dive*, https://developer.ibm.com/articles/cc-cognitive-recurrent-neural-networks, Letzter Zugriff: 15.01.2023, 16. Aug. 2017.

[13] A. Oussidi und A. Elhassouny, „Deep generative models: Survey,“ in *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, 2018, S. 1–8. DOI: 10.1109/ISACV.2018.8354080.

[14] D. Bank, N. Koenigstein und R. Giryes, „Autoencoders," *CoRR*, 2020. DOI: 10.48550/arXiv.2003.05991.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza u. a., *Generative Adversarial Networks*, 2014. DOI: 10.48550/ARXIV.1406.2661.

[16] J.-Y. Zhu, T. Park, P. Isola und A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2017. DOI: 10.48550/ARXIV.1703.10593.

[17] D. Christine et al., „Synthetic Data generation using DCGAN for improved traffic sign recognition," *Neural Computing and Applications*, S. 1–16, Apr. 2021. DOI: 10.1007/s00521-021-05982-z.

[18] D. Spata, D. Horn und S. Houben, „Generation of Natural Traffic Sign Images Using Domain Translation with Cycle-Consistent Generative Adversarial Networks," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, S. 702–708. DOI: 10.1109/IVS.2019.8814090.

[19] J. Stallkamp et al., „The German Traffic Sign Recognition Benchmark: A multi-class classification competition," in *EEE International Joint Conference on Neural Networks*, 2011, S. 1453–1460.

[20] United Nations Economic Commission for Europe, *Convention on Road Traffic*, 28. März 2006.

[21] W. Burger und M. J. Burge, „Digital Image Processing: An Algorithmic introduction," in Springer Cham, 2022, Kap. Geometric Operations, S. 601–637. DOI: 10.1007/978-3-031-05744-1.

[22] F. Dunn und I. Parberry, „3D Math Primer for Graphics and Game Development," in A K Peters/CRC Press, 2011, Kap. Rotation in Three Dimensions. Adresse: https://gamemath.com/book/orient.html.

Anhang

A. test

B. test

C. test