

# **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**  
**Matrikelnummer, Kurs**  
**Betreuer**

24.10.2022 - 08.06.2023  
6526552, TINF20ITA  
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](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](http://www.ib.dhbw-mannheim.de/fileadmin/ms/bwl-ib/Downloads_alt/Leitfaden_31.05.pdf), S. 49

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>V</b>
<b>Abbildungsverzeichnis</b>	<b>VI</b>
<b>Tabellenverzeichnis</b>	<b>VII</b>
<b>Listings</b>	<b>VIII</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung . . . . .	1
1.2 Vorgehensweise . . . . .	1
1.3 Ziel der Arbeit . . . . .	1
<b>2 Stand der Technik</b>	<b>2</b>
2.1 Momentane Lösungen zur Straßenschilderkennung . . . . .	2
2.2 Künstliche Neuronale Netze . . . . .	2
2.2.1 Training . . . . .	4
2.2.2 Convolutional Neural Networks . . . . .	5
2.3 Bildgenerierung ohne Generative Adversarial Networks . . . . .	6
2.3.1 Boltzmann Maschinen . . . . .	6
2.3.2 Deep Belief Netzwerke . . . . .	6
2.4 Bildgenerierung mit Generative Adversarial Networks . . . . .	6
2.4.1 Spieltheorie . . . . .	7
2.4.2 Training . . . . .	7
2.4.3 Architekturen . . . . .	8
2.5 Vorherige Arbeiten . . . . .	9
2.5.1 Generierung deutscher Straßenschilder . . . . .	9
2.5.2 Generierung taiwanischer Straßenschilder . . . . .	10
2.6 Machine Learning Frameworks . . . . .	10
2.6.1 Tensorflow . . . . .	10
2.6.2 Pytorch . . . . .	10
2.6.3 Weitere . . . . .	10
<b>3 Konzeption des Generative Adversarial Networks</b>	<b>11</b>
3.1 Datensatz . . . . .	11
3.2 Framework . . . . .	11
3.3 Architektur . . . . .	11
3.4 Datenaugmentation . . . . .	11
3.5 Training . . . . .	11

<b>4</b>	<b>Implementierung und Training</b>	<b>12</b>
4.1	Hyperparameter . . . . .	12
4.2	Hinzufügen äußerer Einflüsse . . . . .	12
4.2.1	Verwackeln . . . . .	12
4.2.2	Artefakte . . . . .	12
4.2.3	Schnee . . . . .	12
<b>5</b>	<b>Evaluation</b>	<b>13</b>
5.1	Metrik zur Messung der Performanz . . . . .	13
5.2	Vergleich von Unet und Resnet . . . . .	13
<b>6</b>	<b>Zusammenfassung</b>	<b>14</b>
	<b>Anhang</b>	<b>16</b>

# Abkürzungsverzeichnis

<b>CNN</b>	Convolutional Neural Network
<b>CycleGAN</b>	Cycle-Consistent Generative Adversarial Network
<b>GAN</b>	Generative Adversarial Network
<b>GTSRB</b>	German Traffic Sign Recognition Benchmark
<b>KNN</b>	Künstliches Neuronales Netz

# Abbildungsverzeichnis

2.1	Einzelnes Neuron eines Künstliches Neuronales Netzs (KNNs) [4] . . . . .	3
2.2	Vollständiges KNN [4] . . . . .	4
2.3	Trainingsprozess von GANs (angelehnt an [4, S. 654f.]) . . . . .	8
3.1	Beispielbilder aus dem GTSRB Datensatz [9] . . . . .	11

# 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 Momentane Lösungen zur Straßenschilderkennung

### 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. [2, 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 Jahrhunderts, 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. [3, 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 sogenannten *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. [4]

In der nachfolgenden Abbildung ist ein einzelnes künstliches Neuron eines KNNs dargestellt. Das *Plus-Zeichen* steht für die Berechnung des Skalarprodukts der Eingänge und der darauf addierte Bias. Die Aktivierungsfunktion wird durch das *Sigmoid-Zeichen* symbolisiert. Der Ausgang (*rechts*) ist das Ergebnis der Berechnung des Neurons. [4]

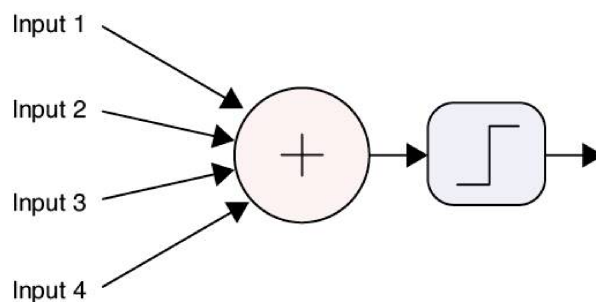


Abbildung 2.1: Einzelnes Neuron eines KNNs [4]

Um komplexe Probleme lösen zu können, werden mehrere 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. [3]

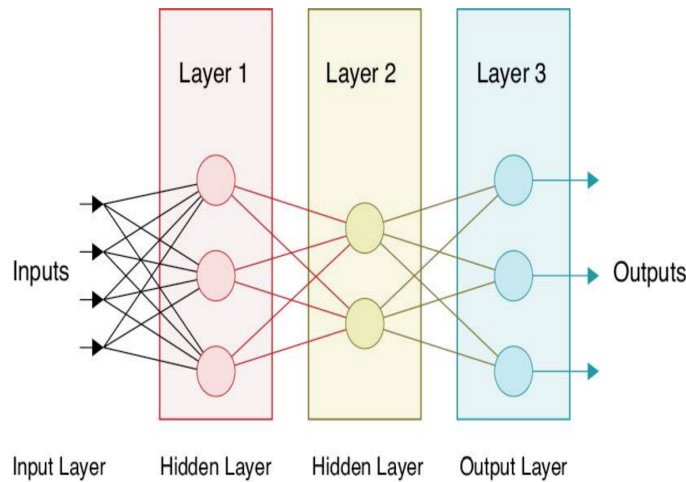


Abbildung 2.2: Vollständiges KNN [4]

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  $\theta$  kann beispielsweise folgendermaßen definiert sein:

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

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

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

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  $\theta$ . Von ihnen hängt der Wert der Kostenfunktion ab. Die Ausgangssituation ist hierbei folgende: Mit einem gegebenen  $\theta$  befindet man sich in einem bestimmten Punkt der Kostenfunktion  $J(\theta)$ . Gesucht ist eine Parameteränderung  $d\theta$ , 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  $\theta$  in Richtung des negativen Gradienten der Kostenfunktion bewegt. Also die Richtung, in die, aus der momentanen Ausgangssposition, 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 sogenannte Lernrate  $\alpha$  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 Gradientenabstieg 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 klassifizieren soll, muss demnach folgende Anzahl an Eingängen verarbeiten:

$$1024 \cdot 1024 = 1.048.576 \quad (2.3)$$

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. Der Begriff *Convolution*, zu deutsch *Faltung*, gibt einen Hinweis auf die Funktionsweise dieser Netze.

## 2.3 Bildgenerierung ohne Generative Adversarial Networks

### 2.3.1 Boltzmann Maschinen

### 2.3.2 Deep Belief Netzwerke

## 2.4 Bildgenerierung mit Generative Adversarial Networks

Generative Adversarial Networks (GANs) sind eine der eingesetzten Methoden, um Algorithmen zu entwickeln, die neue Daten künstlich generieren können. Dazu zählen zum Beispiel Bilder von Menschen, die zwar realistisch erscheinen, die so aber in der Realität nicht existieren.

Die Aufgabe eines GANs ist demnach, Daten zu generieren, die denen des definierten Trainingsatzes in einer solchen Weise ähneln, dass nicht mehr unterschieden werden kann, ob ein Bild künstlich generiert ist oder aus dem Trainingssatz stammt. In dem genannten Beispiel besteht der Trainingssatz aus echten Bildern von Menschen, durch die das GAN lernt, neue Bilder zu erzeugen. Im weiteren Verlauf wird das Thema GANs ausschließlich auf die Bilderzeugung bezogen, auch wenn ebenso andere Anwendungsgebiete existieren. [4]

Ein GAN besteht aus zwei Komponenten. Dem sogenannten *Generator* und dem *Discriminator*. Der Generator erzeugt neue Bilder, während der Discriminator für jedes erzeugte Bild rät, ob es aus dem Trainingssatz stammt, oder ob es künstlich generiert ist. Nur der Discriminator hat dabei Zugriff auf den Trainingsdatensatz. Ihm werden klassischerweise abwechselnd Bilder aus dem Trainingssatz und erzeugte Bilder des Generators gezeigt. Die beiden Komponenten des GANs agieren dabei stets gegeneinander. Der Generator versucht den Discriminator in die Irre zu führen, dass seine generierten Bilder in Wahrheit aus dem Trainingssatz stammen würden, während der Discriminator versucht, möglichst gut zu erkennen, ob ein Bild echt ist oder nicht. Bei beiden Komponenten handelt es sich dabei um künstliche neuronale Netze (KNNs). [4]

Während die Bildklassifizierung ein Minimierungsproblem ist, stellt die Bildgenerierung mittels GANs ein Min-Max-Problem dar. Bei ersterem wird versucht, die Loss-Function zu minimieren, sodass die Predictions möglichst gut den Labels der Daten entsprechen. Bei GANs besteht der Unterschied darin, dass einerseits der Generator die Loss-Function zu minimieren versucht, während andererseits der Discriminator sie zu maximieren versucht.

$$\min_G \max_D V(D, G) = E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))] \quad (2.4)$$

### 2.4.1 Spieltheorie

Aus spieltheoretischer Sicht kann das Gegenspiel von Generator und Discriminator auch als *Nullsummenspiel* betrachtet werden.

### 2.4.2 Training

Zu Beginn sind sowohl der Generator als auch der Discriminator nicht gut geeignet für ihre jeweiligen Aufgaben. Der Generator erzeugt Bilder, die denen des Trainingsatzes nicht ähnlich sind, während der Discriminator noch nicht weiß, was die Bilder des Trainingsatzes einzigartig macht. Durch das Training verbessern sich beide Komponenten in ihren Aufgaben. Sie trainieren sich gegenseitig, da sie beide versuchen, das Spiel zu gewinnen.



Das finale Optimum ist, dass der Discriminator so gut in der Unterscheidung zwischen echten und künstlichen Daten wird, wie mit den vorhandenen Daten nur möglich, während der Generator trotzdem in der Lage sein soll, den Discriminator zu überlisten. [4, S. 656]

**Bild ist nicht 100% korrekt! G wird z.B. nicht trainiert, wenn D richtig lag, aber das Bild aus dem Datensatz stammte.**

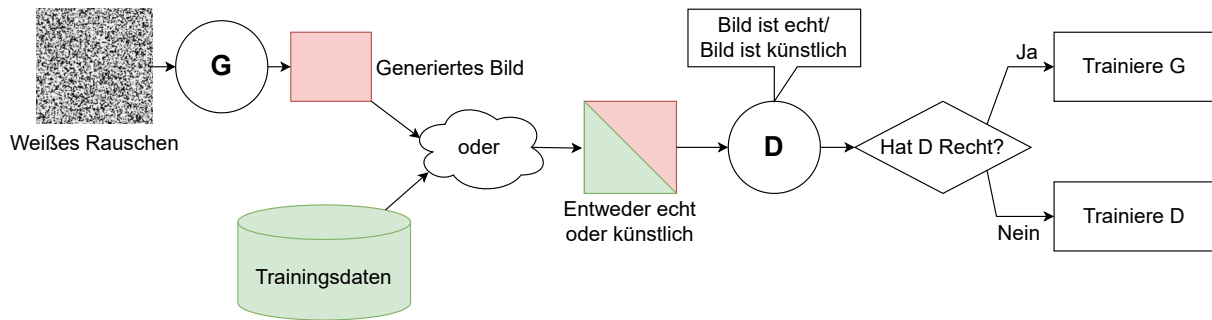


Abbildung 2.3: Trainingsprozess von GANs (angelehnt an [4, S. 654f.])

(Auf Empfindlichkeit von GANs für Hyperparameter eingehen)

### 2.4.3 Architekturen

Die bisher beschriebene Architektur von GANs wird auch als *Vanilla GAN* bezeichnet. Dies entspricht dem, wie GANs in Goodfellow's Publikation definiert werden [5]. 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. Lösen lässt sich dieses Problem beispielsweise mit sogenannten Cycle-Consistent Generative Adversarial Networks (CycleGANs).

**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 aus einem Input einen gewissen Output generieren können,

sondern soll auch von einem gegebenen Output zurück auf den Input schließen können. Dies lässt sich mathematisch so darstellen, dass  $G$  und  $F$  folgende Abbildungen implementieren:

$$\mathbf{G} : X \mapsto Y \wedge \mathbf{F} : Y \mapsto X \quad (2.5)$$

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.

Was deutlich wird, ist dass hierbei die Idee verworfen werden muss,  $G$  ein weißes Rauschen als Input zuzuführen. CycleGANs sind für spezielle Anwendungsgebiete gedacht, in denen ausgewählte, und somit nicht-zufällige Eingabebilder verwendet werden. Dazu zählen Gebiete wie *Style Transfer* oder die Transformation von Bildern, respektive Bildelementen. Es wird sich im Verlauf der Arbeit zeigen, dass die Problemstellung dieser Studienarbeit ein solches Anwendungsgebiet darstellt.

$$L_{GAN}(G, D_Y, X, Y) = E_y[\log D_Y(y)] + E_x[\log(1 - D_Y(G(x)))] \quad (2.6)$$

$$L_{cyc}(G, F) = E_x[||F(G(x)) - x||_1] + E_y[||G(F(y)) - y||_1] \quad (2.7)$$

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F) \quad (2.8)$$

[6]

**Wasserstein GANs** [2] [4]

## 2.5 Vorherige Arbeiten

### 2.5.1 Generierung deutscher Straßenschilder

[7]

## **2.5.2 Generierung taiwanischer Straßenschilder**

[8]

## **2.6 Machine Learning Frameworks**

### **2.6.1 Tensorflow**

### **2.6.2 Pytorch**

### **2.6.3 Weitere**

# 3 Konzeption des Generative Adversarial Networks

German Traffic Sign Recognition Benchmark (GTSRB)



(a)



(b)



(c)



(d)

Abbildung 3.1: Beispielbilder aus dem GTSRB Datensatz [9]

[9]

## 3.1 Datensatz

## 3.2 Framework

## 3.3 Architektur

## 3.4 Datenaugmentation

## 3.5 Training

# **4 Implementierung und Training**

## **4.1 Hyperparameter**

## **4.2 Hinzufügen äußerer Einflüsse**

### **4.2.1 Verwackeln**

### **4.2.2 Artefakte**

### **4.2.3 Schnee**

# **5 Evaluation**

## **5.1 Metrik zur Messung der Performanz**

## **5.2 Vergleich von Unet und Resnet**

## **6 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] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [3] D. Sonnet, *Neuronale Netze Kompakt, Vom Perceptron zum Deep Learning*. Wiesbaden: Springer Vieweg Wiesbaden, 2020. DOI: 10.1007/978-3-658-29081-8.
- [4] A. S. Glassner, „Deep Learning: A Visual Approach,“ in San Francisco: No Starch Press, 2021, S. 649–673, ISBN: 978-1-7185-0072-3.
- [5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza u. a., *Generative Adversarial Networks*, 2014. DOI: 10.48550/ARXIV.1406.2661.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] J. Stallkamp et al., „The German Traffic Sign Recognition Benchmark: A multi-class classification competition,“ in *IEEE International Joint Conference on Neural Networks*, 2011, S. 1453–1460.



# Anhang

A. test

B. test

C. test