HAW HAMBURG

BACHELORTHESIS
Lukas Reinkemeier

Analyse und Test verschiedener Cycle-GAN Ansätze zur ungepaarten Bild-zu-Bild-Übersetzung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Lukas Reinkemeier

Analyse und Test verschiedener Cycle-GAN Ansätze zur ungepaarten Bild-zu-Bild-Übersetzung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 19. März 2021

Lukas Reinkemeier

Thema der Arbeit

Analyse und Test verschiedener Cycle-GAN Ansätze zur ungepaarten Bild-zu-Bild-Übersetzung

Stichworte

Neuronale Netze, Generierende Modelle, Bildverarbeitung, GAN, CycleGAN

Kurzzusammenfassung

Gegenstand dieser Arbeit ist die Analyse und Verbesserung des CycleGAN-Konzeptes zur Bildtransformation. Ziel ist neben einer eigenen Implementierung eines entsprechenden Modells und Trainingsscriptes die Integration zweier Techniken aus dem Bereich der GAN-Forschung – Wasserstein-GAN und Spectral Normalization- in die Architektur als je eine Trainingsvariante. Die drei Ansätze wurden trainiert und unter Verwendung von FID-Score, FCN-Score und eigener Metriken evaluiert, wobei sich zeigte, dass vor allem Wasserstein-GAN zu einer Optimierung des Trainings beitragen kann.

Lukas Reinkemeier

Title of Thesis

Analysis And Testing Of Different Cycle-GAN Approaches For Unpaired Image-To-Image-Translation

Keywords

Machine Learning, Neural Nets, Generative Models, Image Processing, GAN, CycleGAN

Abstract

The subject of this thesis is the analysis and improvement of the CycleGAN concept for image transformation. In addition to implementing a corresponding model and training script, the aim is to integrate two techniques from the area of GAN research - Wasserstein-GAN and Spectral Normalization - into the architecture as one training variant each. The three approaches were trained and evaluated using FID-, FCN-Score and custom metrics, which showed that Wasserstein-GAN in particular can help optimize training.

Inhaltsverzeichnis

1 Einführung

Wenn man ein Feld der Computerwissenschaften heraussuchen müsste, das zur Zeit die meiste Beachtung erhält, wäre dies wohl der Bereich der KI-Forschung, und konkret der Bereich des *Machine Learning*. Dies zeigt sich zum Beispiel darin, dass Machine Learning in der Wirtschaft immer mehr zum Einsatz kommt: 2019 setzten bereits knapp 60 Prozent aller deutschen Firmen auf Machine-Learning-Technologien [Reder-2019]. Zum anderen kommen KI-Technologien aber auch mehr und mehr in der breiten Gesellschaft an und stehen dem Einzelanwender zur Verfügung. Beispiel FaceApp: Durch Anwendung von neuronalen Netzwerken werden Portrait-Bilder von Nutzern umgewandelt, zum Beispiel in eine gealterte Version oder das Geschlecht wird getauscht [Das-2020].

Dies stellt ein konkretes Beispiel eines Algorithmus zur Bildtransformation dar, also eines Verfahrens, das in der Lage ist, eine Repräsentation bzw. Darstellung einer Szene oder eines Objektes in eine andere Repräsentation der gleichen Szene umzuwandeln. Neben verschiedenen Ansätzen für diese Aufgabe hat sich das CycleGAN-Verfahren mit guten Ergebnissen hervorgetan. Im Vergleich zu anderen Ansätzen [Sangkloy-2016, Karacan-2016] bietet es den Vorteil, dass es für den Trainingsvorgang keinen Datensatz mit gepaarten Bildern benötigt, sondern mit ungepaarten Datensätzen arbeiten kann. Dieser CycleGAN-Ansatz soll in der vorliegenden Arbeit beschreiben und selbst implementiert werden. Ebenso sollen verschiedene Ansätze für eine mögliche Optimierung des Verfahrens erläutert und implementiert werden, um dann in einer Versuchsreihe die Leistung des originalen Ansatzes mit denen der Optimierungsansätze zu vergleichen.

Wie der Name vermuten lässt, setzt CycleGAN die GAN-Architektur ein, welches ein *Deep Learning* Verfahren aus dem Bereich der Bildgenerierung ist. Im Folgenden soll kurz erläutert werden, was D*eep Learning* meint und wie sich GANs und CycleGAN darin einordnen lassen. Im Kapitel 2 wird dann beschrieben wie GANs im Allgemeinen funktionieren, um dann in Kapitel 3 zu erläutern, was den auf GANs basierenden CycleGAN-Algorithmus ausmacht. In Kapitel 4 sollen dann die beiden Optimierungsansätze *Spectral Normalization* und *Wasserstein-GAN* und ihre Implementierung beschrieben werden. Schließlich wird dann in Kapitel 5

eine Versuchsreihe der drei Implementierungen durchgeführt, um dann anhand ausgewählter Evaluationsmetriken Aussagen darüber zu treffen, ob diese Ansätze eine sinnvolle Verbesserung darstellen oder nicht.

1.1 Machine Learning

Grund für den zu Beginn beschriebenen KI-Hype ist der Aufschwung des *Deep Learning* bzw. des Erfolgs neuronaler Netze. Diese stellen ein Teilgebiet des Bereichs des *Representation Learning* dar, welches wiederum ein Teilgebiet des *Machine Learning* ist.

Machine Learning ist einfach gesagt ein Verfahren, dass ein Programm in die Lage versetzt, die Lösung einer Aufgabe zu erlernen, ohne dass ihm diese Lösung explizit einprogrammiert wurde. Etwas technischer formuliert lernt ein solches Programm aus Erfahrungen E in Bezug auf eine Aufgabe T und ein Maß für die Leistung P, sodass seine durch P gemessene Leistung bei T mit der Erfahrung E anwächst [Raschka-2018, S.4]. Erfahrungen macht das Programm dabei in einer Trainingsphase, in der es einen Datensatz als Input erhält und aus diesen Daten Muster, also Wissen extrahiert. Wenn das Programm nach abgeschlossenem Training neue Daten erhält, kann es diese auf Basis des gelernten Wissens einordnen. Beispiel: Ein Programm soll zwischen Spam-Mails und echten Mails unterscheiden können. Dazu wird es mit einem Datensatz bestehend aus einer Menge Spam-Mails und echter Mails “gefüttert” und lernt dann anhand eines für diese Aufgabe angemessenen Trainingsalgorithmus diese zu unterscheiden (E). Wenn es nun nach Abschluss des Trainings eine neue E-Mail als Input erhält (T), kann es mit einer (vom Trainingserfolg abhängigen) Genauigkeit (P) bestimmen, ob die Mail Spam ist oder nicht.

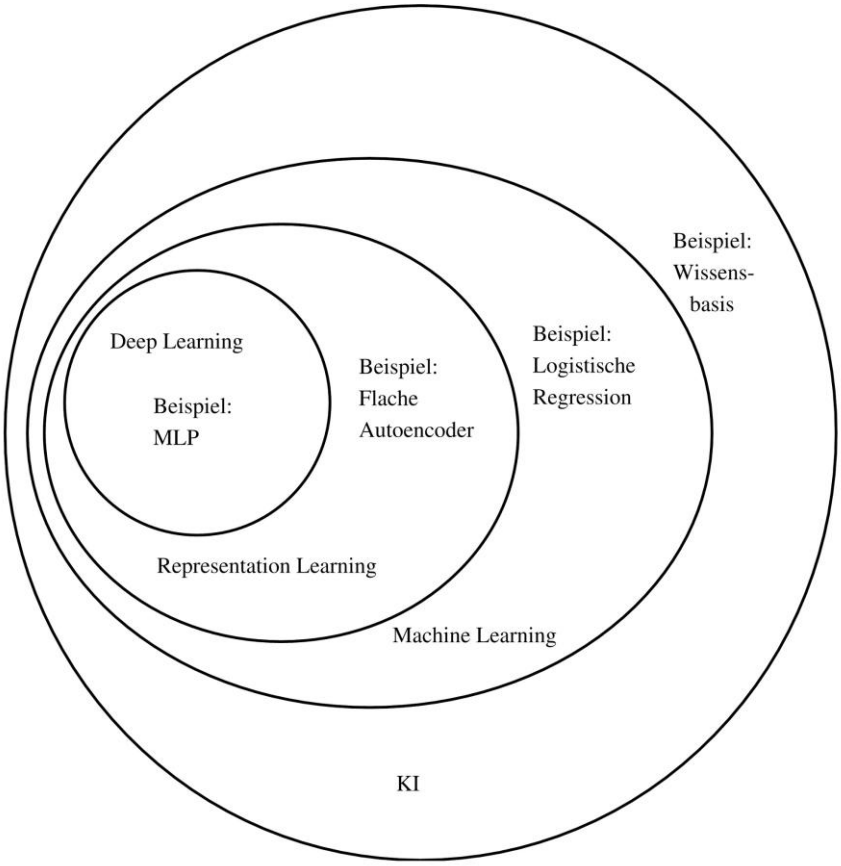

Abb. 1.1: Deep Learning als Teil der KI [Goodfellow-2018, S.32].

1.2 Representation Learning

Der Erfolg des Trainings ist zum einen vom gewählten Trainingsalgorithmus abhängig, zum anderen aber auch von der gewählten Repräsentation der Eingangsdaten: Je komplexer die Eingangsdaten, desto schwieriger das Training. Der Trainingsdatensatz muss also in eine vereinfachte Repräsentation umgewandelt werden, die aber noch informativ genug ist, dass das Programm aus ihr das für die Bewältigung der Aufgabe nötige Wissen ableiten kann. Die Herausforderung besteht also darin, zu wissen welche Merkmale bzw. welche Abstraktionsebene der Daten hierfür relevant ist. Dementsprechend hat sich als Teilgebiet des Machine Learnings das *Representation Learning* gebildet. Dieser Ansatz bedeutet im Wesentlichen, dass das Programm zusätzlich lernt, den Datensatz überhaupt erst in sinnvolle Repräsentationen zu zerlegen, um dann auf Basis der Kombination dieser gelernten Repräsentationen bzw. Merkmale Entscheidungen in Bezug auf die Aufgabe zu treffen [Goodfellow-2018, S.27/28].

1.3 Deep Learning

Die Schwierigkeit dieses Ansatzes ist die Komplexität der Daten und häufig auch die Abhängigkeit der zu erlernenden Merkmale untereinander. Eine Lösung für dieses Problem stellt das *Deep Learning* dar, wobei komplexe Merkmale durch Zusammensetzung einfacherer Merkmale gelernt werden [Goodfellow-2018, S.28]. Dies wird ermöglicht durch die Verwendung von neuronalen Netzen, die in Analogie zum biologischen Gehirn eine Vernetzung mehrerer künstlicher Neuronen darstellen, die in mehreren miteinander verbundenen Schichten angeordnet sind. Jede Schicht stellt dabei eine Abstraktionsebene der Merkmale der Eingabedaten dar. Je mehr Schichten das Netz hat, desto komplexere Repräsentationen kann es also lernen. (Man spricht auch von der Tiefe des Netzes, deshalb *Deep Learning)*.

GANs und damit auch CycleGANs basieren auf einer bestimmten Art dieses Netzes, sogenannten *Convolutional Neural Networks*. Diese sind besonders geeignet für die Verarbeitung von Bilddaten und bestehen aus mehreren kombinierten *Convolutional Layers* sowie *Pooling Layers*. Bei einem *Convolutional Layer* ist ein Neuron einer Schicht nicht mit allen Neuronen der vorherigen Schicht verbunden, sondern nur mit einer Teilmenge (analog zu einem Wahrnehmungsfeld, wobei sich diese Wahrnehmungsfelder von nebeneinander liegenden Neuronen entsprechend überlappen). Die so erhaltenen Eingabewerte werden nun gefiltert bzw. gewichtet durch eine Filtermap der gleichen Größe wie das Wahrnehmungsfeld und das Ergebnis wird an die nächste Schicht geleitet.

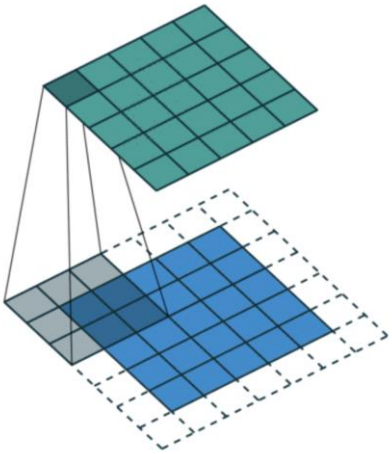

Abb. 1.2: Darstellung des Wahrnehmungsfeldes eines *Convolutional Layers* [Dumoulin-2016, S.23].

Ein *Pooling Layer* reduziert die Menge der weitergeleiteten Daten, indem die Anzahl der Neuronen reduziert wird und jedes Neuron mit einer begrenzten Anzahl Neuronen der vorherigen Schicht verbunden wird. Die Daten dieser Neuronen werden dann aggregiert (z.B. wird bei *max-pooling* der größte der erhaltenen Werte ausgewählt) und weitergeleitet. Dies verringert die Anzahl der Parameter und damit die Rechenlast für das Netzwerk.

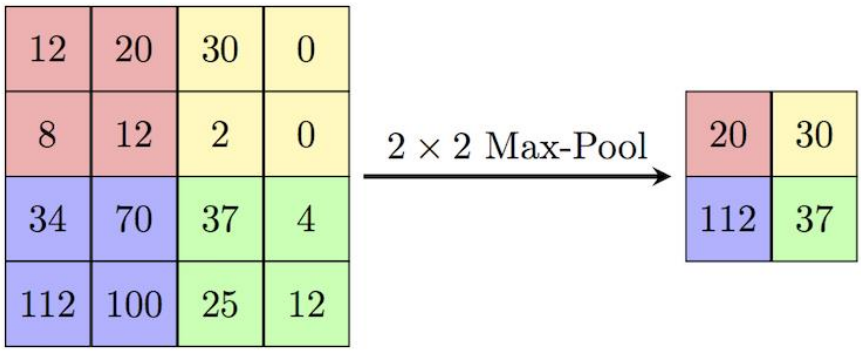

Abb. 1.3: Aggregation durch ein 2x2 *Pooling Layers* mit Max-Pooling.[1]

[1] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

2 GAN

Da die CycleGAN-Architektur im Wesentlichen die Kombination zweier GANs ist, soll zunächst geklärt werden, was GANs sind und wie sie funktionieren.

Generative Adversarial Networks, kurz GANs, wurden zum ersten Mal von Goodfellow u.a. 2014 beschrieben und stellen eine Variante der generativen Modellierung dar, also dem Trainieren von Modellen, die eine gegebene Datenverteilung p_{data} erlernen (etwa aus einem gegebenen Set von Bildern einer gemeinsamen Domäne, z.B. Gesichter), diese in einer bestimmten Weise modellieren (p_{model}) und in der Lage sind, neue Datenpunkte aus dieser Verteilung zu generieren (also z.B. neue Bilder von Gesichtern). Das Ziel dabei ist es zu erreichen, dass p_{data}möglichst genau durch p_{model}abgebildet wird, sodass optimalerweise neu erzeugte Datensätze nicht mehr von "echten" Daten aus p_{data}zu unterscheiden sind [Goodfellow-2017, S.2].

Das GAN-Konzept versucht dieses Ziel zu erreichen, indem zwei Netzwerke gegeneinander antreten: ein Generator G, der einen Datenpunkt aus p_{model}generiert und ein Diskriminator D, der lernt zu unterscheiden, ob ein Datenpunkt aus p_{model} oder p_{data} stammt. Zur Verdeutlichung wird häufig die Analogie von Geldfälscher und Polizei herangezogen: Der Generator als Fälscher versucht möglichst gute Fälschungen zu erzeugen, während der Diskriminator als die Polizei versucht, dieses Falschgeld als solches zu erkennen. Die Konkurrenz, in der beide Seiten zueinander stehen, treibt diese dazu, ihre Methoden immer weiter zu verbessern, bis die Fälschungen so gut sind, dass sie von echtem Geld nicht mehr zu unterscheiden sind. (An diesem Punkt ist ein Gleichgewicht erreicht, an dem die eine Partei nicht mehr besser werden kann, ohne dass die andere Seite ineffektiver wird.) [Goodfellow-2017, S.18]

Formel beschrieben ist der Generator eine differenzierbare Funktion G mit Parametern θ_G mit einem Rauschen z als Input, welches basierend auf p_{model} in einen neuen Datenpunkt als Output umgewandelt wird. Der Diskriminator ist eine differenzierbare Funktion D mit Parametern θ_D und x als Input und einem Skalar als Output, der die Wahrscheinlichkeit angibt, dass x aus p_{data} und nicht aus p_{model} stammt. Generator und Diskriminator besitzen jeweils eine

Kostenfunktion, die abhängig von den Parametern beider Spieler ist und die sie minimieren wollen. Da beide Spieler jedoch nur ihre jeweils eigenen Parameter kontrollieren können, lässt sich diese Situation gut als Spiel beschreiben, dessen Lösung das Finden eines Nash-Gleichgewichts ist (d.h. das Finden eines Punktes an dem es für keinen der beiden Spieler, bei gleichbleibenden Parametern des anderen, sinnvoll ist, die eigenen Parameter zu ändern) [Goodfellow-2017, S.18-19, Goodfellow-2014, S.2-3].

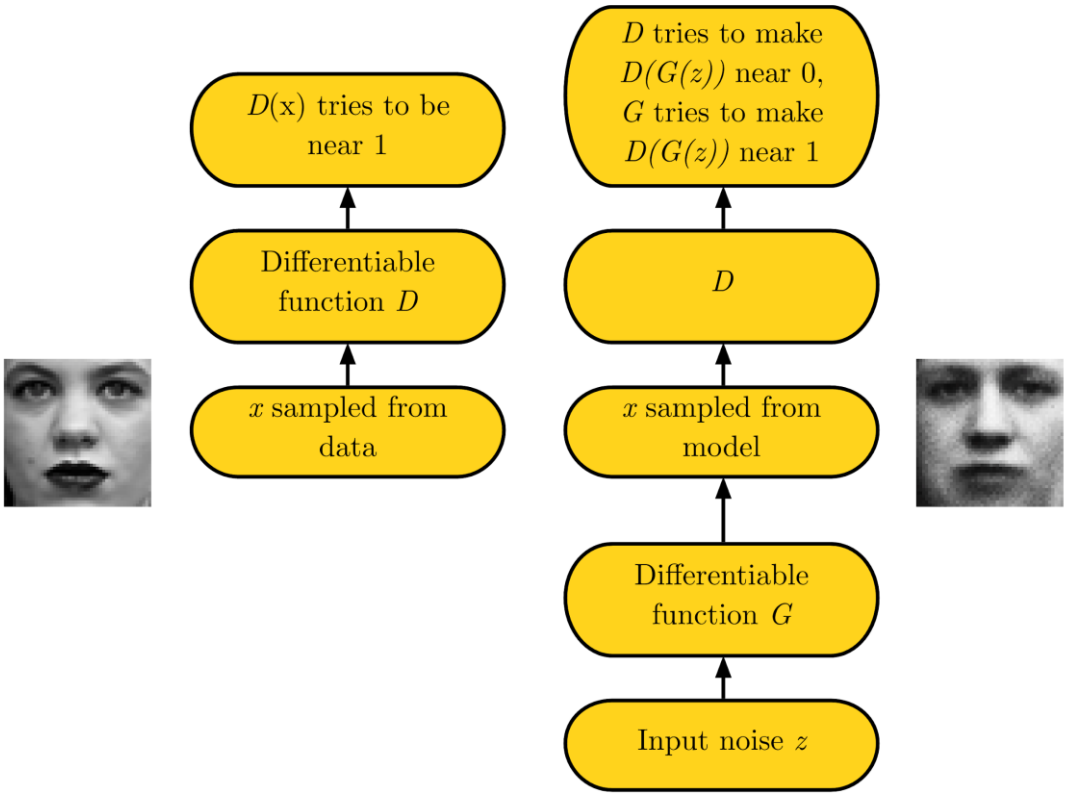

Abb. 2.1: Der GAN-Ablauf, links mit einem Datenpunkt aus p_{data}, rechts mit von G generierten Daten aus p_{model} [Goodfellow-2017, S.19]

D wird dahingehend trainiert, die Wahrscheinlichkeit zu maximieren, dass er richtig bestimmt, ob x aus p_{data} oder p_{model} stammt, also $D(x)$ zu maximieren und $D(G(z))$ zu minimieren. Der Generator hingegen versucht entsprechend $D(G(z))$ zu maximieren, also die Wahrscheinlichkeit, den Diskriminator zu täuschen.

Die Kostenfunktion für den Diskriminator ist die zu minimierende Kreuzentropie, die üblicherweise eingesetzt wird, einen binären Klassifizierer mit Output zwischen 0 und 1 zu trainieren [Goodfellow-2017, S.21]:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} log\, D(x) - \frac{1}{2}\mathbb{E}_{z\sim p_{data}} log(1 - D(G(z)))$$

Wenn man das Szenario als Nullsummenspiel begreift, ergibt sich als Kostenfunktion für den Diskriminator entsprechend:

$$J^{(G)} = -J^{(D)}$$

Während also der Diskriminator die Kreuzentropie minimiert, wird sie vom Generator maximiert, also der Teil $log(1 - D(G(z)))$. Dies ist insofern ungünstig, da dies, wenn der Diskriminator den Output des Generators mit hoher Sicherheit als Fake erkennt (wie vor allem zu Beginn des Trainings), zu sehr schwachen Gradienten für den Generator führt (=*vanishing gradient*). Es würde also entsprechend lange dauern, bis p_{model} in Richtung p_{data} geleitet wird. Eine Lösung dafür ist, dass der Generator nicht die Wahrscheinlichkeit minimiert, dass der Diskriminator korrekt ist, sondern die Wahrscheinlichkeit maximiert, dass der Diskriminator sich irrt. Anstatt also einfach das Vorzeichen von $J^{(D)}$zu drehen, ergibt sich [Goodfellow-2017, S.22-23]:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z log D(G(z))$$

Konkret werden Generator und Diskriminator als mehrschichtige neuronale Netze implementiert, die in Form stochastischer Gradientenminimierung trainiert werden. Dabei wird zwischen k Schritten der Optimierung des Diskriminators und einem Schritt der Optimierung des Generators alterniert. Dies soll bewirken, dass sich der Diskriminator nahe seiner optimalen Lösung bewegt und somit dem Generator sinnvolles Feedback liefert (vorausgesetzt der Generator verändert sich nicht zu schnell) [Goodfellow-2014, S.2-3].

3 CycleGAN

Während das GAN-Konzept gute Ergebnisse bei der Erzeugung neuer Bilder liefert, kann es auch erweitert werden für die Zwecke der Bildtransformation. Die in dieser Arbeit untersuchte Variante ist die der CycleGANs.

Das Ziel des CycleGAN-Ansatzes ist es, Bilder einer gegebenen Szene von einer Repräsentation dieser Szene in eine andere zu überführen.[2] Genauer gesagt wird vorausgesetzt, dass (1) Bilder aus Domäne X eine bestimmte Charakteristik gemeinsam haben, dass (2) Bilder aus Domäne Y eine bestimmte (andere) Charakteristik gemeinsam haben und dass (3) Domäne A und B jeweils in einer Beziehung zueinander stehen. Dabei ist es für den Trainingsprozess nicht erforderlich, dass gepaarte Beispiele vorliegen. Es ist also nicht nötig, dass zu einem Bild x ein Bild y gegeben ist, das genau dieselbe Szene darstellt und sich nur in der gegebenen Charakteristik unterscheidet, die das Bild x der Domäne X und das Bild y der Domäne Y zuordnet.

Zur Verdeutlichung: Der (klassische) Beispielfall wäre ein Set von Bildern, die jeweils Pferde abbilden (1), sowie ein Set aus Bildern, die jeweils Zebras abbilden (2). Die Beziehung (3) zwischen beiden Domänen wäre, dass jeweils ein oder mehrere Tiere zu sehen sind, die sich optisch, abgesehen von Fellfarbe und –muster, kaum voneinander unterscheiden. Nach ausreichendem Training wäre ein CycleGAN in der Lage, ein Bild von einem Pferd so zu transformieren, dass es nachher ein Zebra abbildet, der Rest des Bildes jedoch möglichst unverändert bleibt. Dieser Prozess soll auch in der Gegenrichtung möglich sein.

Für die konkrete Umsetzung wird zum einen das Konzept vom *adversarial loss* von GANs genutzt, zum anderen wird dieses um einen *cycle consistency loss* ergänzt: Es gibt also jeweils zwei Generator-Netzwerke G und F sowie zwei Diskriminator-Netzwerke D_X und D_y, wobei G Bilder aus Domäne X so umwandeln soll, dass D_y nicht in der Lage ist, sie von Bildern aus Domäne Y zu unterscheiden, und das gleiche gilt umgekehrt für F und D_X. Zusätzlich soll

[2] Im Folgenden siehe [Zhu-2017, S.1-5]

gelten, dass wenn ein Bild x von G in Domäne Y überführt und dann von F wieder in X überführt wird, das ursprüngliche Bild herauskommen soll (wobei auch dies wieder in beide Richtungen gelten soll).

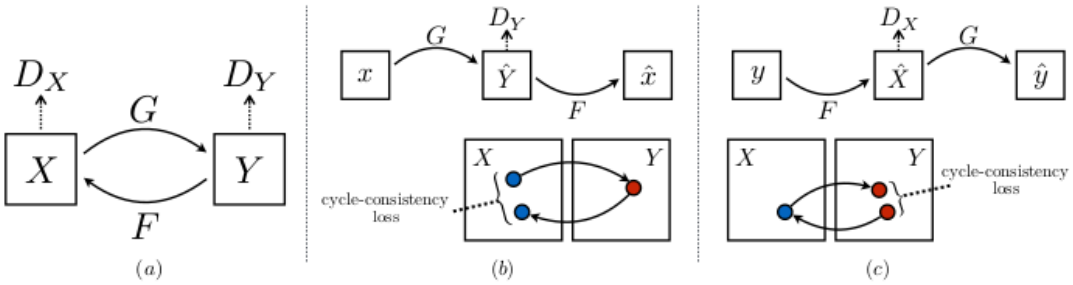

Abb. 3.1: Darstellung von *adversarial loss* (a), *forward* (b) und *backward* (c) *cycle consistency loss* [Zhu-2017, S.3]

3.1 Mathematische Grundlagen

Formal beschrieben ist das Ziel, zwei Abbildungsfunktionen $G : X \rightarrow Y$und $F : X \rightarrow Y$ zu finden unter Verwendung gegebener Stichproben$\{x_i\}_{i=1}^{N}$und $\{y_j\}_{j=1}^{M}$wobei $x_i \in X$ und $y_j \in Y$. Die Datenverteilung ist beschrieben als $x \sim p_{data}(x)$ und $y \sim p_{data}(y)$. Dazu gehören die Diskriminatoren D_X und D_Y, wobei D_X zwischen Bildern $\{x\}$ und transformierten Bildern $\{F(y)\}$ unterscheidet, während D_Y zwischen $\{y\}$ und $\{G(x)\}$ unterscheidet. Die für das Training entscheidenden Terme sind der *adversarial loss*, der die Verteilung der generierten Bilder der Verteilung der Bilder der Zieldomäne anpasst, sowie der *cycle consistency loss*, der verhindert, dass sich die gelernten Abbildungsfunktionen G und F widersprechen.
Die Anwendung des *adversarial loss* auf $G : X \rightarrow Y$ ergibt den Term:

$$\mathcal{L}_{GAN}(G, logD_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[logD_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[log(1 - D_Y(G(x)))]$$

Dies entspricht der Formel aus Kapitel 2, mit dem Unterschied, dass G nicht zufälliges Rauschen als Input erhält, sondern ein Beispiel aus Domäne X.

Im Sinne eines minMax-Spiels beschreibt die erste Hälfte des Terms dabei, mit welcher Wahrscheinlichkeit Bilder, die tatsächlich aus Domäne Y stammen, von D_y als solche erkannt werden (D_y versucht diesen Part also zu maximieren), während die zweite Hälfte beschreibt, mit

welcher Wahrscheinlichkeit die von G generierten Bilder von D_y als Fake erkannt werden ($D_y$versucht entsprechend auch diesen Term zu maximieren, während G versucht ihn zu minimieren, also gute Fakes zu erzeugen):

$$\min_G \max_{D_Y} \mathcal{L}_{GAN}(G, D_Y, X, Y)$$

Ebenfalls wird ein entsprechender Term für $F : X \rightarrow Y$ und den zugehörigen Diskriminator D_X eingeführt, also:

$$\min_F \max_{D_x} \mathcal{L}_{GAN}(F, D_X, Y, X)$$

für

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{data}(x)}[logD_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}[log(1 - D_X(F(y)))]$$

Die so formulierten Terme würden zwar schon ausreichen für eine Architektur, die Bilder aus einer gegebenen Domäne in eine andere umwandelt. Allerdings würde ein solches Netzwerk Bilder erzeugen, die zwar der Zieldomäne entsprechen, aber keinerlei Ähnlichkeit zum Ursprungsbild aufweisen müssten. Um zum Pferd-Zebra-Beispiel zurückzukehren: Ein Bild eines Pferdes auf einer Wiese könnte transformiert werden zu einem Bild mehrerer Pferde an einem Strand. Das erzeugte Bild würde der Zieldomäne entsprechen, aber mit dem Ursprungsbild nicht mehr viel gemein haben. Um diesem Effekt entgegenzuwirken, werden die Abbildungsfunktionen um einen *cycle consistency loss* ergänzt, mit dem Ziel, dass ein Bild, nachdem es von Generator G umgewandelt wurde, von Generator F wieder zurück in das ursprüngliche Bild transformiert werden soll.

Es soll also gelten $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, was als *forward cycle consistency loss* bezeichnet wird. Das gleiche soll auch für die Gegenrichtung gelten, also *backward cycle consistency loss*: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. Dies wird durch den folgenden loss term beschrieben:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||]$$

Dieser Term nimmt umso kleinere Werte an, je weniger sich das Ursprungsbild vom Output unterscheidet, nachdem es durch beide Generatoren gelaufen ist. G und F versuchen also auch diesen Term zu minimieren, es gilt also:

$$\min_{G,F} \mathcal{L}_{cyc}(G, F)$$

Der vollständige Term lautet nun

$$\mathcal{L}(G, F, D_X, D_{Y)} = \mathcal{L}_{GAN}(G, D_Y, X, Y) \; + \; \mathcal{L}_{GAN}(F, D_X, Y, X) \; + \; \lambda \mathcal{L}_{cyc}(G, F)$$

wobei λ die relative Gewichtung von *adversary loss* und *cycle consistency loss* reguliert.

Das Ziel ist zu lösen für:

$$G^*, F^* = arg \min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

Das heißt $D_x D_y$versuchen den Term zu maximieren, also “echte” Bilder korrekt ihrer Domäne zuzuordnen und die von G und F generierten Bilder erfolgreich von “echten” zu unterscheiden, während *G* und *F* versuchen den Term zu minimieren, also sowohl zu erreichen, dass die von ihnen generierten Bilder nicht mehr von “echten” unterschieden werden können, als auch dass diese, wenn sie dem anderen Generator als Input gegeben werden, wieder das Ursprungsbild ergeben.

3.2 Erweiterung

Die Autoren der ursprünglichen Arbeit modifizieren die zuvor aufgestellten Terme, um den Trainingsprozess zu stabilisieren und verbessern. Zum einen wird zur Berechnung des *adversary loss* die negative log-likelihood-Gleichung durch eine mittlere quadratische Abweichung ersetzt, die nicht nur leichter zu berechnen ist, sondern auch den Trainingsprozess stabilisieren und Bilder von höherer Qualität erzeugen soll. Konkret würde ein Generator G versuchen den Term

$$\mathbb{E}_{x \sim p_{data}(x)}[(D(G(x)) - 1)^2]$$

zu minimieren und der entsprechende Diskriminator D versuchen den Term

$$\mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[D(G(x))^2]$$

zu minimieren.

Ebenfalls wird ein weiterer Term, ein *identity loss* eingeführt. In einigen Fällen war eine unnötige Verschiebung des Farbstichs zu beobachten (vor allem bei Style-Transfer-Beispielen, also wenn das gesamte Bild zu einer andren Darstellungsfrom transformiert werden musste). Diesem soll der *identity loss* entgegenwirken. Der zugehörige Term lautet:

$$L_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[||G(y) - y||] + \mathbb{E}_{x \sim p_{data}(x)}[||F(x) - x||]$$

und soll von G und F minimiert werden, was im Effekt bewirkt, dass wenn einer der Generatoren ein Bild aus seiner Zieldomäne als Input erhält (statt wie eigentlich aus seiner Input-Domäne), das Ergebnis sich möglichst nicht vom Ursprungsbild unterscheiden soll.

3.3 Architektur

3.3.1 Diskriminator

Als Diskriminatoren werden sogenannte PatchGAN-Diskriminatoren verwendet. Im Gegensatz zu einem neuronalen Netz, welches für ein gegebenes Bild als Ganzes entscheidet, ob dieses "echt" ist oder nicht (mit einem Skalar als Output), entscheidet der PatchGAN-Diskriminator für NxN große Abschnitte(=Patches) des Input-Bildes, ob sie "echt" sind oder nicht, und gibt dieses Ergebnis als Array aus. Konkret werden 70x70-Patch-GANs verwendet, für Input-Bilder der Dimension 256x256 ist der Output eine 16-stellige Matrix (wobei jeder der 16 Werte die Ausgabe für ein *recpetive field* der Größe 70x70 ist). PatchGAN-Diskriminatoren kommen mit weniger Parametern aus und können mit Bildern von beliebiger Größe arbeiten. Durch die Fokussierung auf (kleinere) Bildausschnitte soll eine bessere Detailschärfe erzielt werden [Isola-2016, S.3-4].

Für den Diskriminator werden Convolution-Layer mit Instanz-Normalisierung (außer nach dem ersten Layer) und *LeakyReLu* (ReLu=Rectified Linear Unit) mit Gefälle=0.2 als Aktivierungsfunktion verwendet. Genauer besteht der Diskriminator aus 4 4x4 Layern mit stride=2, Zero Padding und mit je 64,128,256 und 512 Filtern sowie einem 4x4 Layer mit stride=1, Zero Padding und 512 Filtern. Zum Schluss wird ein 4x4 Layer mit Zero Padding, stride=1 und einem Filter angeschlossen. (Der eine Filter ist letztendlich das Feature "echt", also zur Zieldomäne gehörend, oder nicht.) Diesem folgt keine Aktivierungsfunktion, weshalb der Output wie zuvor erwähnt kein Skalar zwischen 0 und 1, sondern eine Ergebnismatrix ist.

Tab. 3.1 Die Diskriminator-Architektur, aufgeschlüsselt nach Layer, Input-Form und Output-Form (bei einem Eingabebild von 256x256 Pixeln mit 3 Kanälen (R,G,B). None bezeichnet die Batchgröße, die architekturseitig nicht vorgegeben, also frei wählbar ist.

Layer	**Input**	**Output**
InputLayer	(None,256,256,3)	(None,256,256,3)
Conv2D_Layer_C64	(None,256,256,3)	(None,128,128,64)
Leaky_ReLu_1	(None,128,128,64)	(None,128,128,64)
Conv2D_Layer_C128	(None,128,128,64)	(None,64,64,128)
Instance_Norm_1	(None,64,64,128)	(None,64,64,128)
Leaky_ReLu_2	(None,64,64,128)	(None,64,64,128)
Conv2D_Layer_C256	(None,64,64,128)	(None,32,32,256)
Instance_Norm_2	(None,32,32,256)	(None,32,32,256)
Leaky_ReLu_3	(None,32,32,256)	(None,32,32,256)
Conv2D_Layer_C512	(None,32,32,256)	(None,16,16,512)
Instance_Norm_3	(None,16,16,512)	(None,16,16,512)
Leaky_ReLu_4	(None,16,16,512)	(None,16,16,512)
Conv2D_Layer_Patch	(None,16,16,512)	(None,16,16,1)

3.3.2 Generator

Der Generator ist eine Deep Convolutional Encoder-Decoder-Architektur, die ein Input-Bild reduziert/kodiert auf eine Bottleneck-Ebene. Dort wird diese Repräsentation dann von einer Reihe ResNet-Layern interpretiert (hier findet also die eigentliche Umwandlung von Domäne A zu B statt). Anschließend wird die Repräsentation dann wieder von Decoder-Layern auf die originale Bildgröße zurück dekodiert.
Für den Generator werden Convolution-Layer mit Instanz Normalisierung und *ReLu* als Aktivierungsfunktion verwendet. Für das Encoding folgen auf ein 7x7 Layer mit Schrittweite=1, Reflection-Padding und 64 Filtern zwei weitere 3x3 Layer mit je 128 und 256 Filtern, Schrittweite=2 und Zero-Padding. Dem folgen neun ResNet-Blocks, wobei ein ResNet-Block je zwei 3x3 Layer mit 256 Filtern, Schrittweite=1 und Reflection-Padding enthält.

Zum Schluss wird wieder decodiert über zwei 3x3 Layer mit je 128 und 64 Filtern, Schrittweite=2 und Zero-Padding und zum Schluss einem 7x7 Layer mit Schrittweite=1 und 3 Filtern (dies entspricht der Anzahl der Kanäle der Input-Bilder, in diesem Fall 3, da die Bilder im R-G-B Format vorliegen, also für jeden Pixel 3 Werte gegeben sind).

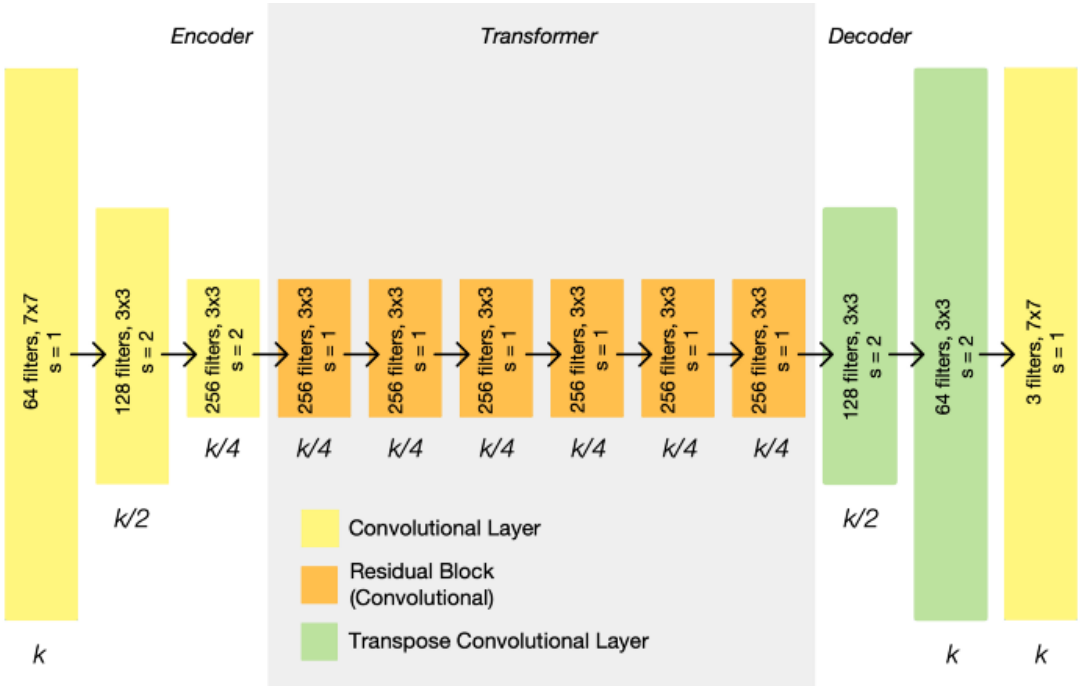

Bild 3.3: Schematische Darstellung eines Generators, wobei k die Größe des ursprünglichen Inputs ist.[3]

[3] https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d (Sarah Wolf)

4 Erweiterungen von CycleGAN

Während die beschriebene Architektur des CycleGAN schon gute Ergebnisse liefert, stellt sich natürlich die Frage, wie sich diese Architektur noch verbessern ließe. Erste Ansätze wären natürlich die üblichen Stellschrauben für CNNs, also die Lernrate, Batch-Größe etc. Es ist aber davon auszugehen, dass die Autoren von CycleGAN selbst schon ausreichend getestet haben, um dann mit den Einstellungen zu arbeiten, die sie für ihre Implementierung angeben (und auch entsprechend in dieser Arbeit verwendet werden).

In der vorliegenden Implementierung und in der Versuchsreihe soll entsprechend stattdessen versucht werden durch Anpassung der Architektur an sich eine Optimierung des Netzwerks zu erreichen.

Für die Optimierung ist dabei nicht nur die Qualität der generierten Bilder von Bedeutung, sondern auch die Stabilität des Trainingsverlaufs. Das wohl häufigste Problem, das beim Training eines GAN-Netzwerkes (und damit auch eines CycleGAN-Netzwerkes) auftreten kann, ist ein Mode-Collapse [Goodfellow-2017, S.34]: Hierbei bildet der Generator verschiedene Inputs auf den gleichen Output ab (im Extremfall alle Inputs auf den gleichen Output; der Generator würde immer das gleiche Bild generieren). Für CycleGANs würde das bedeuten, dass z.B. im Pferde/Zebra Datensatz aus verschiedenen Pferdebildern das gleiche Bild eines Zebras generiert würde (was offensichtlich der Zielsetzung des CycleGANs widersprechen würde, dass nur der Aspekt, in dem sich die beiden Domänen wesentlich unterscheiden, transformiert wird, der Rest des Bildes jedoch unverändert bleibt).

Ein weiteres Problem, das im Trainingsverlauf eines GANs auftreten kann, ist, dass die beiden Netzwerke nicht konvergieren, weil zwischen einigen wenigen Modellen oszilliert wird, die jeweils vom Diskriminator zurückgewiesen werden.

Schließlich sind auch instabile oder verschwindende Gradienten ein mögliches Problem während des Trainings. Im Falle verschwindender Gradienten unterscheiden sich die vom Generator generierten Bilder und die der Zieldomäne derart stark, dass der Diskriminator kein

konstruktives Feedback an den Generator geben kann. Dies liegt daran, dass das Netzwerk immer in kleinen Schritten lernt, die Verteilung der modellierten Domäne bewegt sich also immer nur ein kleines Stück in eine Richtung. Normalerweise sollte der Diskriminator ein entsprechendes Feedback geben, wenn sich das Modell in die richtige Richtung bewegt. Wenn jedoch die generierte und die reale Verteilung sich so stark unterscheiden, dass sie disjunkt sind, kann der Diskriminator kein konstruktives Feedback mehr geben.

Um eine solche Optimierung zu erreichen, die den genannten Problemen begegnet, sollen im Folgenden zwei Ansätze vorgestellt werden, die für die Verbesserung des allgemeinen GAN-Ansatzes entwickelt wurden und für die Versuchsreihe dieser Arbeit in die CycleGAN-Architektur integriert werden sollen.

4.1 Spectral Normalization

Der Anspruch des Spectral-Normalization-Ansatzes ist es, das Training von GANs zu stabilisieren und dabei qualitativ gleichwertige oder sogar besser Bilder zu erzeugen. Dabei ist die wesentliche Idee, die Gradienten in jedem Layer zu regulieren und so zu starke Gradientensprünge zu vermeiden. Dies soll einen stabilen Trainingsverlauf ermöglichen.

Konkret erreicht Spectral Normalization dies durch Kontrolle der Lipschitz-Konstante für die Diskriminatorfunktion, indem sie die Spektralnorm für jedes Layer begrenzt. Was genau das bedeutet, soll hier kurz umrissen werden:

Die Lipschitz-Konstante gibt für eine lipschitz-stetige Funktion an, mit welchem maximalen Wert die Funktion an beliebiger Stelle steigt, oder anders gesagt: der Maximalwert der Ableitung der Funktion [Goodfellow-2018, S.122].

Per Definition ist die Lipschitz-Norm $||g||_{Lip}$ für g gleich $sup_h \sigma(\nabla g(h))$. $\sigma(A)$ist dabei die Spektralnorm der Matrix A [Miyato-2018, S.3].

Die Spektralnorm ist für eine gegebene Matrix der größte Singularwert der Matrix $\sigma(A)$ (äquivalent zu: wie weit kann die Matrix einen gegebenen Vektor strecken). Singuläre Werte können in der Singulärwertzerlegung errechnet werden. Für eine $m \times n$-Matrix A ist diese Zerlegung: $A = UDV^\top$. U ist eine$m \times m$-Matrix, V eine $n \times n$-Matrix und D eine $m \times n$-Matrix. U und V sind orthogonale Matrizen, D ist eine Diagonalmatrix. Die Spalten von U sind die linken, die Spalten von V die rechten Singularvektoren von A, welches die Eigenvektoren von $AA^\top$ bzw. die Eigenvektoren von $A^\top A$ sind. $AA^\top$ und $A^\top A$ haben die gleichen positiven Eigenwerte.

D enthält die Quadratwurzeln dieser Eigenwerte, die Singularwerte. -> Die Spectral Norm für A ist also der größte in D enthaltene Wert [Goodfellow-2018, S.70-71].

Für ein lineares Layer $g(h) = Wh$ eines Netzwerkes (wobei Wdie Gewichte-Matrix ist) ist die Spektralnorm entsprechend gegeben durch $||g||_{Lip} = sup_h \sigma(\nabla g(h)) = sup_h \sigma(W) = \sigma(W)$.

Die Lipschitz-Konstante für eine Funktion ist also gleich ihrem größten singulären Wert, also ihrer Spektralnorm. Spectral Normalization normalisiert nun die Spektralnorm der Gewichtsmatrix W, sodass sie die Lipschitz-Norm $\sigma(W) = 1$aufweist, und zwar durch $\bar{W}_{SN}(W) := W/\sigma(W)$ und $\sigma(\bar{W}_{SN}(W)) = 1$. Wenn dies auf jedes Layer des Diskriminators angewandt wird, erfüllt die Diskriminatorfunktion die Lipschitz-Norm $||f||_{Lip} = 1$ [Miyato-2018, S.3].

Da die Berechnung der Eigenwerte aber sehr aufwendig ist, wird stattdessen der größte Singularwert durch Vektoriteration angenähert, was deutlich weniger rechnerischen Aufwand mit sich bringt [Miyato-2018, S.3].

4.1.1 Implementierung Cycle-SN-GAN

Für die Kombination von CycleGAN und Spectral Normalization zu Cycle-SN-GAN wird konkret in der Architektur die Spectral Normalization für jedes Layer des Diskriminators durchgeführt, im so abgewandelten SN-CycleGAN also in je beiden Diskriminatoren nach jedem Layer. (An entsprechender Stelle ersetzt sie die im CycleGAN verwendete Instanz Normalisierung.) Verwendet wird hierfür die schon vorhandene Implementierung des Paktes *tensorflow-addons*, welches als Eingabe das zu normalisierende Layer hat und als Ausgabe das Layer mit den entsprechend normalisierten Gewichten hat. Die übrige Architektur, also der Generator, der Trainingsablauf sowie die verwendeten Trainingsparameter bleiben gleich.

Tab. 4.1: Die Diskriminator Architektur unter Verwendung von Spectral Normalization.

Layer	**Input**	**Output**
InputLayer	(None,256,256,3)	(None,256,256,3)
Conv2D_Layer_C64	(None,256,256,3)	(None,128,128,64)
Spectral_Norm_1	(None,128,128,64)	(None,128,128,64)
Leaky_ReLu_1	(None,128,128,64)	(None,128,128,64)

Conv2D_Layer_C128	(None,128,128,64)	(None,64,64,128)
Spectral_Norm_1	(None,64,64,128)	(None,64,64,128)
Leaky_ReLu_2	(None,64,64,128)	(None,64,64,128)
Conv2D_Layer_C256	(None,64,64,128)	(None,32,32,256)
Spectral_Norm_2	(None,32,32,256)	(None,32,32,256)
Leaky_ReLu_3	(None,32,32,256)	(None,32,32,256)
Conv2D_Layer_C512	(None,32,32,256)	(None,16,16,512)
Spectral_Norm_3	(None,16,16,512)	(None,16,16,512)
Leaky_ReLu_4	(None,16,16,512)	(None,16,16,512)
Conv2D_Layer_Patch	(None,16,16,512)	(None,16,16,1)
Spectral_Norm_4	(None,16,16,1)	(None,16,16,1)

4.2 Wasserstein-GAN

Im klassischen GAN wird die Divergenz zwischen realer und modellierter Verteilung $p_r$und p_g als Kullback-Leibler-Divergenz oder Jensen-Shannon-Divergenz gemessen (um dann mittels gradient descent die Verteilung deckungsgleich zu bekommen). Problem: Wenn $p_r$und p_g ausreichend stark voneinander abweichen, sind die Unterschiede in der so gemessenen Divergenz minimal (-> vanishing gradient problem), der Generator lernt kaum etwas. Erst ab einer gewissen Nähe beider Verteilungen zueinander ergibt sich ein für den Generator relevanter Gradient. Um diesem Problem zu begegnen, wird im originalen GAN die cost function abgewandelt (statt $log(1 - D(G(z)))$zu minimieren, wird $log(D(G(z)))$maximiert, was zu größeren Gradienten zu Trainingsbeginn führt (siehe Kapitel 2)). Auch mit dieser Variante hat das GAN-Training allerdings mit Instabilität zu kämpfen. Im originalen CycleGAN wird versucht dieses Problem zu lösen durch Verwendung der mittleren quadratischen Abweichung (siehe 3.2).

Ein anderer Ansatz findet sich im Wasserstein-GAN(kurz: WGAN), in dem die Divergenz beider Verteilungen durch die Wasserstein-1-Distanz (auch *Earth Mover Distance*) gemessen wird [Arjovsky-2017, S.3-8]. Diese führt zu aussagekräftigen Gradienten auch bei starker

Divergenz beider Verteilungen und zu einem deutlich gleichmäßigeren Trainingsverlauf. Die ursprüngliche Wasserstein-Formel ist:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

wobei $\prod(\mathbb{P}_r, \mathbb{P}_g)$ die Menge aller Verteilungen der Verbundwahrscheinlichkeit ist, mit den entsprechenden Randverteilungen $\mathbb{P}_r$ und $\mathbb{P}_g$. Anschaulich erklärt gibt $\gamma(x, y)$an, wieviel "Masse" von x nach y bewegt werden muss, um die Verteilung $\mathbb{P}_r$ in die Verteilung $\mathbb{P}_g$ zu transformieren. Die Wasserstein-Distanz gibt dann die Kosten des optimalen Transportplans an, also desjenigen, in dem am wenigsten Masse transportiert werden muss (angezeigt durch das Infimum inf in der obigen Formel).

Da es allerdings keinen effizienten Algorithmus für die Berechnung aller möglichen Verbundwahrscheinlichkeiten in $\prod(\mathbb{P}_r, \mathbb{P}_g)$ gibt, transformieren die Autoren des WGAN-Ansatzes die obige Formel unter Verwendung der Kantorovich-Rubinstein-Dualität zu

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

Kurz gesagt folgt aus dieser Formel, dass der Diskriminator darauf trainiert wird, eine 1-Lipschitz-stetige Funktion zu finden. Daraus resultiert eine Kostenfunktion für den Diskriminator, die letztendlich nicht die Wahrscheinlichkeit angibt, mit der ein Bild echt oder generiert ist, sondern vielmehr im Sinne eines Kritikers die "Echtheit" eines Bildes bewertet, mit niedrigen Werten (die auch kleiner 0 sein könnte) für geringe und hohe Werte für hohe "Echtheit". Die daraus resultierenden Kostenfunktionen für den Generator und Diskriminator sind für den Diskriminator

$$\frac{1}{m}\sum_{i=1}^{m} f(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f(G(y^{(i)}))$$

und für den Generator entsprechend

$$-\frac{1}{m}\sum_{i=1}^{m} f(G(y^{(i)}))$$

, wobei *m* die Batch-Größe angibt (die im originalen CycleGAN allerdings nur 1 beträgt) und *x* und *y* in der Übertragung zu CycleGAN jeweils Bilder aus den Domänen sind.

Im ursprünglichen WGAN-Ansatz wird die Lipschitz-Stetigkeit des Diskriminators garantiert durch weight clipping, das hieße bei jedem Gradientenupdate des Diskriminators die Reichweite für w auf einen Bereich $[-c, c]$ zu beschränken. Da dies jedoch keine optimale Weise darstellt, die Lipschitz-Beschränkung zu gewähren, wird die Weiterentwicklung des WGAN-Ansatzes WGAN-GP [Gulrajani-2017] verwendet, die statt weight clipping eine Gradient Penalty einführt, das heißt bei jedem Update der Gradienten des Diskriminators wird das Modell abgestraft, wenn sich die Norm der Gewichtematrix von der Zielnorm 1 entfernt.

4.2.1 Implementierung Cycle-WGAN

Um die ursprüngliche GycleGAN-Architektur mit WGAN zu verknüpfen, müssen an drei Stellen Veränderungen vorgenommen werden: Zum einen wird die bislang verwendete Verlustfunktion durch die Wasserstein-Verlustfunktion ersetzt. Des Weiteren wird bei jedem Trainingsdurchlauf der Diskriminator dreimal für jedes Update des Generators geupdated. (Das originale WGAN-Konzept sieht fünf Updates vor; um die Trainingszeit in einem angemessenen Rahmen zu halten, wurden jedoch drei Updates gewählt). Schließlich muss auch bei jedem Update des Diskriminators die Gradient Penalty errechnet und dem Verlust des Diskriminators hinzugefügt werden.

5 Testreihe

5.1 Trainingsparameter

Bei der Wahl der Trainingsparameter wurden die in der originalen CycleGAN-Implementierung verwendeten Werte übernommen und auch in den beiden Varianten soweit möglich nicht verändert.

Allgemein werden die Netzwerke mit einer Lernrate von 0.0002 unter Verwendung eines Adam-Optimizers trainiert, die Gewichte werden zu Beginn entsprechend einer Gaußschen Normalverteilung initialisiert (mit Mittelpunkt 0 und Standardabweichung von 0.02). Für die relative Gewichtung von *adversarial* und *cycle-consistency loss* wird $\lambda = 10$ gewählt, während der hinzugefügte *identity loss* mit $0.5 * \lambda$gewählt wird.

Des Weiteren wird zum Trainieren der Diskriminatoren nicht jeweils das letzte vom jeweiligen Generator generierte Bild verwendet, sondern jeweils ein Pool der letzten 50 generierten Bilder gespeichert. Mit einer Wahrscheinlichkeit von 50% wird entweder das aktuell generierte Bild oder eines aus dem Pool an den Diskriminator gegeben. Dies soll ein Oszillieren des Models reduzieren (also, dass während des Trainings die generierten Bilder zwischen zwei möglichen Repräsentationen hin und her wechseln, statt zu konvergieren) [Zhu-2017, S.5].

Die verwendeten Bilder wurden vor Trainingsbeginn auf die Dimension 256x256 Pixel im RGB-Format umgewandelt.

Insgesamt wurde jede Implementierung für jeweils 200 Epochen trainiert. In jeder Epoche wurde jeweils der komplette Datensatz mit einer Batch-Größe von 1 durchiteriert. Dabei wird das jeweils trainierte Modell alle 10 Epochen gespeichert, um später in der Evaluation Aussagen über die Qualität des Modells über den Trainingsverlauf hinweg treffen zu können.

5.2 Trainingsdatensatz

Es wurden jeweils zwei Anwendungsfälle trainiert, die auch in der originalen CycleGAN Ausarbeitung Anwendung fanden:

- Der horse2zebra Datensatz. Dieser enthält 1067 Pferde und 1334 Zebra Bilder.[4]
- Der cityscape Datensatz. Für diesen wurden jeweils 2975 Bilder von Dashcam Bildern und Segmentierungskarten ausgewählt.[5]

[4] horse2zebra.zip von https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets

[5] leftImg8bit_trainvaltest.zip und gtFine_trainvaltest.zip von https://www.cityscapes-dataset.com/downloads (erfordert Registrierung)

6 Evaluation

6.1 Über die Schwierigkeit der Evaluation von GANs

Nach Abschluss des Trainingsvorgangs sollen natürlich die drei so trainierten CycleGAN Varianten auf ihre Güte hin evaluiert werden. Dies stellt jedoch keine triviale Aufgabe dar: Da GANs (und damit auch CycleGANs) nicht versuchen eine einzige objektive Funktion zu optimieren, sondern ein Gleichgewicht zwischen zwei Funktionen herzustellen, ist der Verlust pro Epoche kein aussagekräftiger Wert in Bezug auf die Qualität des Modells [Goodfellow-2017, S.42]. Mithin gibt es auch keinen allgemein anerkannten Standard zur Evaluierung von GANs.

Da die Verlustfunktion keine direkten Rückschlüsse über die Qualität des Modells zulässt, ist die intuitiv naheliegendste Alternative, die vom Modell generierten Bilder direkt auf ihre Qualität hin zu überprüfen.

Dabei ergeben sich für die Evaluierung speziell des CycleGAN-Anwendungsfalls, also der Transformation von Bildern zwischen zwei Domänen, die folgenden Kriterien, denen das Evaluierungsverfahren genügen muss:

- Da die generierten Bilder der jeweiligen Zieldomäne entsprechen sollten, müssen die gewählten Evaluierungsmetriken in der Lage sein zu ermitteln, ob eine Ähnlichkeit zwischen den generierten Bildern und denen der Zieldomäne besteht
- Die Metriken sollten aufdecken, ob bei verschiedenen Ursprungsbildern die generierten Bilder sich ebenfalls voneinander unterscheiden. (Es sollte also erkennbar sein, ob ein Mode-Collapse stattgefunden hat, also bei verschiedenen Eingaben immer dasselbe Bild generiert wird.)
- Es sollte erkennbar sein, ob ein aus einem Ursprungsbild generiertes Bild immer noch die Szene des Ursprungsbildes darstellt, aber nur in dem Aspekt verändert, der die beiden Domänen wesentlich unterscheidet (also z.B. im Fall “Pferd zu Zebra” sollte das generierte Bild an der Stelle des Pferdes ein Zebra darstellen, den Rest aber unverändert lassen).

Eine häufig verwendete Art der qualitativen Analyse wären Human Perception Studies, also die Bewertung der Bilder durch menschliche Probanden. Dies wäre jedoch im Rahmen dieser Arbeit nicht praktikabel. Neben qualitativer Analyse sind jedoch im Laufe der Forschung an GANs einige quantitative Evaluierungsverfahren vorgeschlagen worden. Für diese Arbeit wurden drei Metriken ausgewählt, die am geeignetsten erscheinen, die zuvor genannten Kriterien zu erfüllen.

6.2 Evaluationsmetriken

6.2.1 Fréchet Inception Distance

Als mögliche Evaluationsmetrik schlagen Goodman e.a. den Inception score vor [Salimans-2016]. Dieser wird auf eine Menge Bilder angewandt und misst die Vielfalt der generierten Bilder und ob sie erkennbar ein bestimmtes Objekt abbilden (Hund, Katze etc). Diese Metrik basiert auf dem Inception Classifier, ein Bild-Klassifizierungsnetzwerk von Google, das darauf trainiert ist, viele verschiedene Labels in Bildern zu erkennen. Dies ist jedoch problematisch, da der IS nur Sinn ergibt für generierte Bilder, die den Labels entsprechen, auf die der Inception Classifier trainiert wurde (ist also nicht auf beliebige Anwendungsfälle generalisierbar). Auch werden hier nur die generierten Bilder für sich betrachtet, unabhängig von der eigentlichen Zieldomäne.
Mittlerweile wird deshalb als neuer Evaluierungsstandard für GANs die Fréchet Inception Distance (FID) als Metrik verwendet [Heusel-2017]. Die FID soll die Ähnlichkeit zwischen generierten und echten Bildern besser erkennen können. Wie beim IS wird das vortrainierte Inception-Modell verwendet, allerdings ohne Verwendung des letzten Layers, welches die Wahrscheinlichkeiten für mögliche Kategorien ausgibt. Stattdessen wird das Pooling-Layer davor verwendet, um die Feature-Vektoren für Input-Bilder zu ermitteln. Dadurch muss der eigene Anwendungsfall nicht mehr im Rahmen der Daten liegen, auf denen das Inception-Modell trainiert wurde, kann aber die schon trainierten tieferen Layer dieses Modells zur Erkennung von Strukturen in den Bildern verwenden. Die so ermittelten Feature-Vektoren werden dann als mehrdimensionale Normalverteilung zusammengefasst durch Berechnung von Durchschnitt und Kovarianz der Bilder. Diese Verteilung wird berechnet für die Menge der generierten und der realen Bilder. Die Frechet-Distanz zwischen diesen beiden Verteilungen ergibt den FID-Score. Dabei signalisiert ein niedriger Wert, dass die generierten Bilder den der Zieldomäne ähneln und dass sie ähnlich verteilt sind (dass also nicht immer das gleiche Bild generiert

wurde). Ein hoher Wert signalisiert entsprechend, dass die generierten Bilder denen der Zieldomäne nicht ähneln oder dass sie sich untereinander zu sehr ähneln.[6]

6.2.2 Klassifizierungsscore

Um vergleichen zu können, wie gut die untersuchten Netzwerke jeweils Pferde zu Zebras und andersrum transformieren können, wurde ein separates Klassifizierungsnetzwerk trainiert. Dieses Netzwerk wurde anhand von Bildern aus vier Tierklassen – Pferd, Zebra, Giraffe und Schaf – trainiert. Es gibt für ein gegebenes Bild mit einer Wahrscheinlichkeit zwischen {0,1} an, zu welcher der vier Klassen das Bild gehört. Erste Implementierungen dieses Klassifizierers litten unter einer zu hohen Sicherheit, das heißt, es wurden nur Werte nah an 0 und 1 ausgegeben, selbst für Fälle, in denen das Bild falsch zugeordnet wurde oder keiner der Klassen klar zuzuordnen war. Um dies zu verhindern, wurde das Netzwerk zusätzlich mit Ausreißern trainiert [Hendrycks-2018]. Dies hat den Effekt, dass das Netzwerk für Bilder, die es nicht zuordnen kann, einen Wert von $\frac{1}{Anzahl\ Klassen}$ ausgibt. Je sicherer es das Bild zuordnen kann, desto näher liegt der Wert dann an 1 für die jeweilige Klasse (und an 0 für alle anderen Klassen).[7]

Zum Vergleich der CycleGAN-Netzwerke soll so jeweils für die Richtung Pferd->Zebra und Zebra->Pferd die durchschnittliche Wahrscheinlichkeit ermittelt werden, mit der der Klassifizierer die generierten Bilder den Klassen Pferd und Zebra zuordnet. Wenn für die Zielklasse ein Wert nahe 1 ermittelt wird, scheint die Transformation gut gelungen zu sein. Wenn der Wert nahe $\frac{1}{Anzahl\ Klassen}$ liegt, aber für die Ursprungsklasse nahe 1 liegt, würde das bedeuten, dass die Bilder zum Großteil unverändert geblieben sind. Wenn die Werte für beide Klassen niedrig sind, ist die Transformation fehlgeschlagen.

Im Vergleich der drei CycleGAN-Implementierungen sollte also das Netzwerk mit dem höheren Klassifizierungsscore besser in der Lage gewesen sein, mit seinen generierten Bildern die

[6] Da das letzte verwendete Layer des Inception-Netzwerkes 2048 Features enthält, ist zur Berechnung des FID-Scores ein Datensatz von ebenfalls mindestens 2048 Bildern empfohlen. Da der PferdZebra-Datensatz nicht die nötige Anzahl Bilder pro Domäne aufweist, werden die fehlenden Bilder generiert durch zufällige Auswahl von Bildern aus dem Datensatz mit anschließendem Vergrößern und zufälligem Zuschnitt auf die Ausgangsgröße.

[7] Für das Training des Klassifizierers wurden je 1000 Bilder pro Klasse aus dem COCO-Image-Dataset verwendet, siehe: https://cocodataset.org/

Zieldomäne zu “treffen”, also die wesentlichen Features, die ein Pferd bzw. ein Zebra ausmachen, in seiner modellierten Verteilung abzubilden und entsprechende Bilder zu erzeugen.

6.2.3 FCN-Score

So wie es schon die Autoren des originalen CycleGAN-Ansatzes getan haben, soll auch hier ein FCN-Score ermittelt werden. Dafür wird als ein Anwendungsfall mit einem Datensatz mit gepaarten Bildern trainiert (aber ohne, dass das Training davon beeinflusst wird; die Prämisse eines ungepaarten Datensatzes bleibt für das Training also erhalten). Dadurch können bei der Evaluierung die generierten Bilder mit den schon vorhandenen Bildern verglichen werden. Hierfür wird der Cityscape-Datensatz verwendet. Der Cityscape-Datensatz enthält eine Menge von Dashcam-Bildern von verschiedenen Straßenszenen aus diversen deutschen Innenstädten. Zu jedem Bild gibt es unter anderem eine semantische Segmentierungskarte, also eine reduzierte Darstellung des Bildes, in der erkannte Instanzen von Objekten als farbiger Umriss dargestellt sind, mit je einer Farbe pro Label (also alle Autos als blaue Umrisse, Fußgänger rot, Bürgersteig pink etc.).

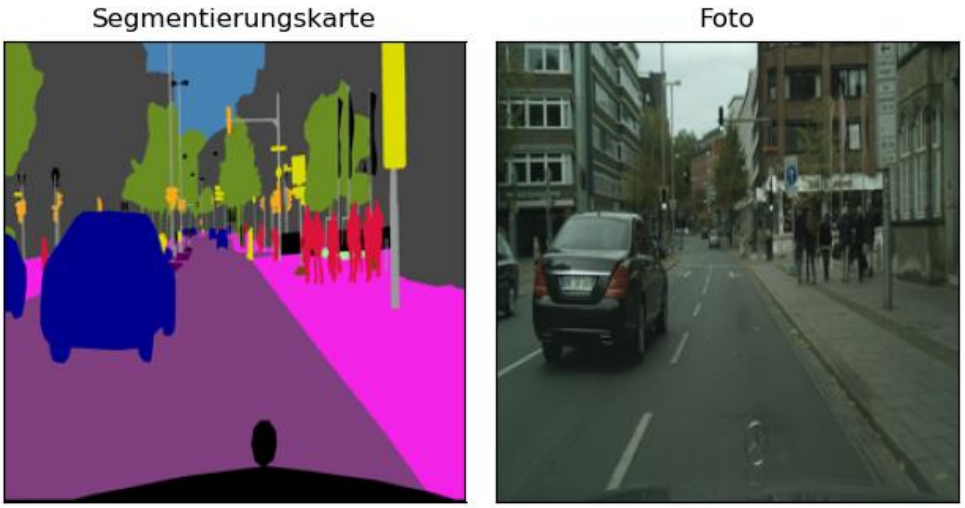

Abb. 6.1: Gepaarte Beispielbilder aus dem Cityscapes-Datensatz

Für die Transformation Segmentierungskarte->Foto wird ein FCN-Score berechnet[8], der ermittelt, wie gut ein bereits trainiertes Netzwerk für semantische Segmentation die generierten Bilder interpretieren kann. Der Score gibt an, mit welcher Sicherheit das Segmentierungsnetzwerk in einem Bild die für die Cityscape-Szene typischen Elemente erkennt, also Straße, Bürgersteig, Auto, Fußgänger etc. Dabei wird aus dem vom CycleGAN-Netzwerk generierten Foto durch das Segmentierungsnetzwerk eine Segmentierungskarte erzeugt. Diese wird dann mit der schon vorhandenen zugehörigen Segmentierungskarte anhand von Standardmetriken für semantische Segmentierung verglichen. Für die Transformation Foto->Segmentierungskarte werden diese Metriken entsprechend direkt berechnet, also die vom CycleGAN-Netzwerk generierte Segmentierungskarte mit der schon vorhandenen verglichen. Die für den FCN-Score verwendeten Metriken sind durchschnittliche Pixelgenauigkeit, durchschnittliche Pixelgenauigkeit pro Klasse und die durchschnittliche Intersection-Over-Union pro Klasse (also wieviel Schnittmenge generiertes und tatsächliches Segmentierungslabel haben).

6.3 Ergebnisse

6.3.1 Manuelle Analyse

Da wie in 6.1 erwähnt die Evaluation der trainierten Modelle über die Analyse der von den Modellen generierten Bilder erfolgen soll, wurden nach Abschluss des Trainings pro Implementierung pro Anwendungsfall und für jedes alle 10 Epochen gespeicherte Modell jeweils die Bilder des Datensatzes von Domäne A nach B und von B nach A transformiert.

Bevor für die jeweiligen Implementierungen die zuvor genannten Metriken ermittelt werden, soll zunächst eine stichprobenartige manuelle Analyse der generierten Bilder erfolgen, um offensichtliche Trends oder Anomalien festzuhalten.

Bei erster Betrachtung der erzeugten Bilder alle Epochen, zeigt sich für das Pferd/Zebra-Beispiel, dass bei allen Ansätzen die Transformation Zebra->Pferd deutlich schwerer zu fallen scheint als umgekehrt. Selbst nach 200 Epochen erfolgt hier nur eine mehr oder weniger starke Braunfärbung, die für Zebras charakteristischen Streifen sind manchmal noch vorhanden. Im Gegenzug wird von allen Ansätzen erkannt, dass die Transformation Pferd->Zebra schwarzweiße Streifen beinhalten muss. Der originale Ansatz zeigt dieses Verhalten schon nach den

[8] Der FCN-Score wird in [Zhu-2017] verwendet, wurde aber zur Evaluierung von GANs erstmal in [Isola-2016] verwendet und basiert auf [Long-2014]

ersten zehn Trainingsepochen, beim Cycle-SN-GAN-Ansatz ist es nach 30 Epochen erkennbar und beim Cycle-WGAN-Ansatz sogar erst nach etwa 70 Epochen.

Im Vergleich der drei Ansätze zeigt sich, dass Cycle-SN-GAN und Cycle-WGAN weniger Trainingsfortschritt erzielen konnten: In den erzeugten Zebrabildern sind die ursprünglichen Pferde manchmal nur zum Teil gestreift oder die in manchen Fällen ursprünglich braune Fellfärbung ist noch zu erkennen, während bei den erzeugten Pferdebildern häufig nur eine mehr oder weniger leichte Braunfärbung des Zebras erfolgt ist, die Streifen sind häufig noch gut erkennbar. Es fällt allerdings auch auf, dass gerade in den späteren Epochen der originale Ansatz "übereifrig" ist und gerade in den späteren Epochen häufig mehr als nur das dargestellte Pferd bzw. Zebra verändert. Gerade die Transformation hin zu Zebras erzeugt manchmal Bilder, die komplett mit schwarz-weißen Streifen versehen sind. In den beiden alternativen Ansätzen ist dieses Verhalten deutlich schwächer ausgeprägt.

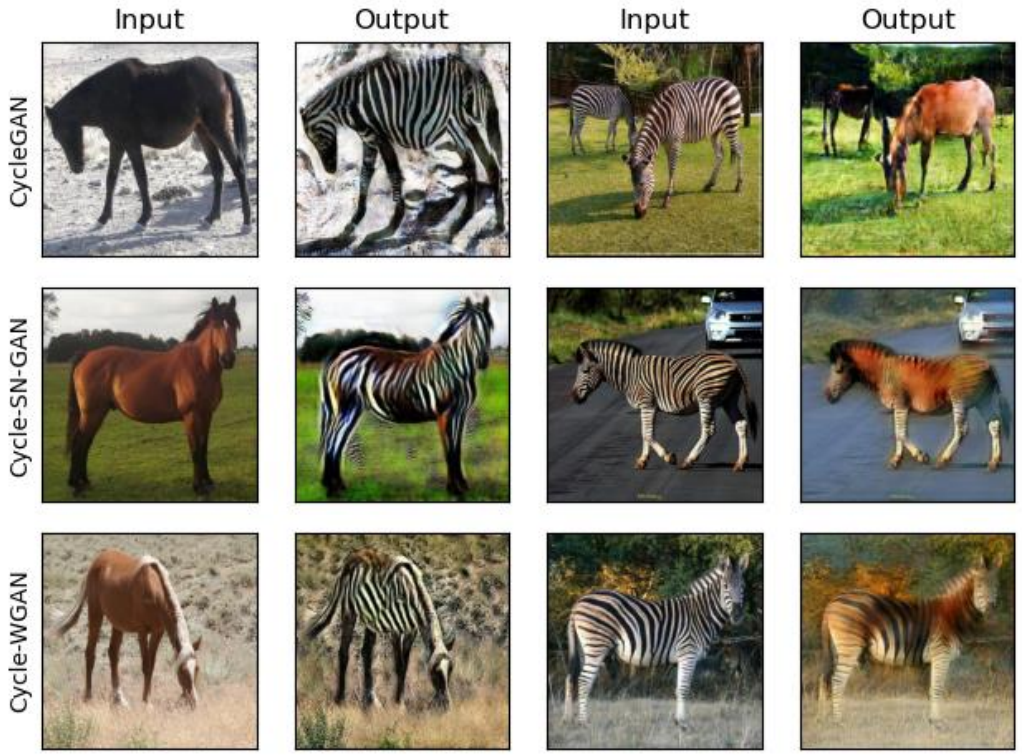

Abb. 6.2: Beispiele für Transformationen im Pferd/Zebra-Beispiel

Im Cityscape-Beispiel lässt sich der Lernfortschritt am besten anhand der generierten Segmentierungskarten erkennen: Während zu Beginn des Trainings zwar die Farbpalette schon deutlich auf den Bereich der originalen Segmentierungskategorien eingeschränkt wird, sind trotzdem noch viele Details innerhalb der Segmente erkennbar. Mit zunehmendem Trainingsverlauf gleichen die generierten Bilder mehr und mehr den originalen Segmentierungskarten und auch die Kategorien (mit blau für Autos, rot für Passanten etc.) scheinen auf den ersten Blick gut getroffen worden zu sein. Nur die Abgrenzungen der einzelnen Bildbereiche entspricht häufig nicht ganz der entsprechenden Segmentierungskarte. Die nach 200 Epochen vom originalen Ansatz generierten Fotos sind von weitem betrachtet zunächst von echten Dashcam-Bildern tatsächlich kaum zu unterscheiden. Erst bei näherem Hinsehen fällt auf, dass zwar einzelne Bildbereiche wie Straßen, Autos, Häuser etc. gut voneinander zu unterscheiden sind, aber in sich häufig keine sinnvolle Struktur aufweisen (Autos haben mehrere Stoßstangen oder sind nicht in sinnvoller Richtung orientiert, Bäume sind meistens nur ein einheitlicher grüner Pixelbrei, Passanten sind nur verschwommene Schemen).

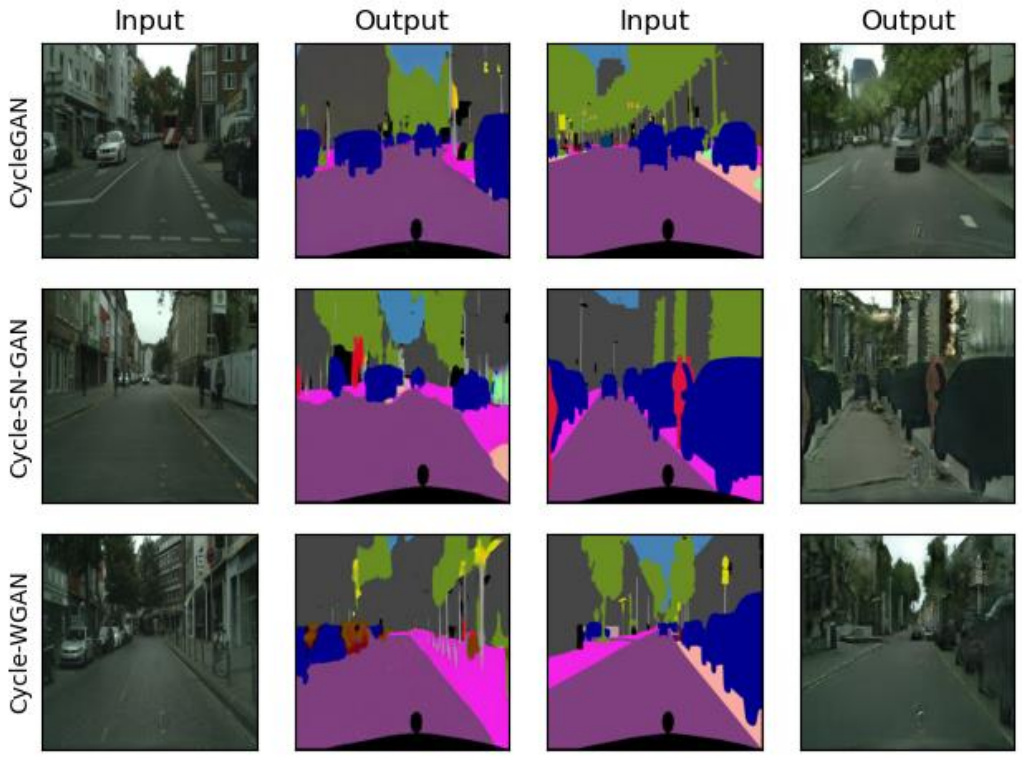

Abb. 6.3: Beispiele für Transformationen im Cityscape-Beispiel.

Im Vergleich der drei Implementierungen fällt auch in diesem Beispiel sofort auf, dass Cycle-SN-GAN und Cycle-WGAN deutlich weniger Fortschritt im Training gemacht haben: Während die generierten Segmentierungskarten zwar auch auf den Farbbereich der Zieldomäne eingeschränkt sind und der Detailgrad angemessen reduziert ist, scheinen doch häufiger Kategorien falsch zugeordnet worden zu sein und die einzelnen Bildbereich weniger deutlich voneinander abgetrennt zu sein. Die generierten Fotos weisen einen deutlich geringeren Detailgrad auf und bestehen zwar gemäß der Zieldomäne hauptsächlich aus Grautönen, einzelne Objekte/Bereiche der Bilder sind jedoch nur verschieden schattierte Umrisse ohne innere Details. Dabei scheint Cycle-SN-GAN am schlechtesten abzuschneiden.

Im Cityscape-Beispiel fällt allerdings auch eine Besonderheit auf: Der originale CycleGAN-Ansatz generiert in Richtung Segmentierungskarten->Foto einige Bilder, die sich optisch kaum voneinander unterscheiden. Nach Sichtung des verwendeten Datensatzes wird klar, dass bei etwa einem Drittel der verwendeten Bilder die sich am unteren Bildrand befindliche Motorhaube des Autos, aus dem heraus die Fotos gemacht wurden, leicht verschoben zum Rest des Bildes ist. Das originale Netzwerk scheint diese Bilder als eine Untervariante des Datensatzes klassifiziert zu haben und behandelt sie in der Transformation alle gleich, generiert also jeweils ein fast identisches Bild. Während dies zu Trainingsbeginn noch kaum zu erkennen ist, zeigt sich nach 20 Epochen deutlich, dass einige der vom originalen Ansatz erzeugten Foto-Bilder kaum voneinander zu unterscheiden sind. Es hat also ein partieller Mode-Collapse stattgefunden. Cycle-SN-GAN und Cycle-WGAN weisen diese Anomalie nicht auf.

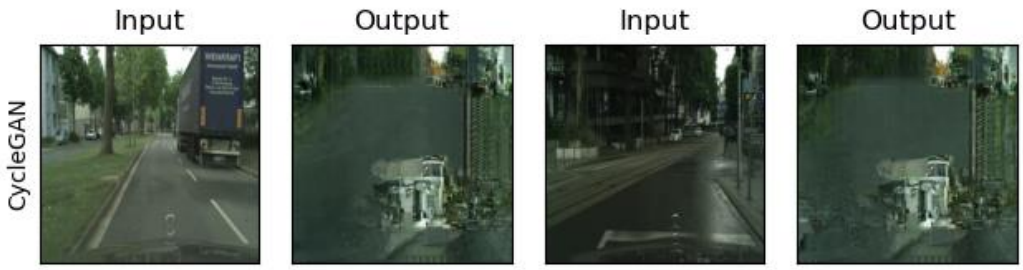

Abb. 6.4: Beispiele für partiellen Mode-Collapse bei CycleGAN im Cityscape-Beispiel

6.3.2 Analyse und Interpretation der Metriken

Die Analyse der erhobenen Metriken bestätigt die Beobachtungen aus 6.3.1[9]:

FID-Score für Pferd/Zebra

Der originale CycleGAN-Ansatz weist nach 200 Epochen in Pferd->Zebra-Richtung einen FID-Score von 48,97 auf, in Richtung Zebra->Pferd einen Wert von 61,35. Der Eindruck, dass Transformation zu Zebras besser erlernt wurde, bestätigt sich also.

Im Vergleich zeigt sich auch bei Cycle-SN-GAN und Cycle-WGAN ein besserer FID-Score in Pferd-Zebra-Richtung als in Zebra-Pferd-Richtung. Mit jeweils 82,65 und 124,04 für Cycle-SN-GAN sowie 130,95 und 138,68 für Cycle-WGAN sind die ermittelten FID-Scores deutlich schlechter als die des originalen CycleGANs.

FID-Score für Cityscapes

Nach 200 Epochen kann der CycleGAN-Ansatz einen FID-Score von 98,22 in Richtung Foto->Segmentierungskarte und 160,49 in Richtung Segmentierungskarte>Foto aufweisen. Cycle-SN-GAN kommt hier auf jeweils 95,80 und 218,09; Cycle-WGAN erreicht Werte von je 97,96 und 159,18. Während also die erzeugten Segmentierungskarten von etwa gleicher Qualität sind, erreicht Cycle-WGAN bei den erzeugten Fotos den besten Wert.

Dabei sollte natürlich berücksichtigt werden, dass der partielle Mode-Collapse beim originalen CycleGAN-Ansatz in Richtung Segmentierungskarten->Fotos den Wert stark nach oben treibt. Dass die von CycleGAN erzeugten Fotos im Erfolgsfall besser sind als die von Cycle-SN-GAN und Cycle-WGAN, zeigt sich, wenn man für die Ermittlung des FID-Scores die Bilder außen vorlässt, die der CycleGAN-Ansatz fälschlicherweise gleich transformiert: In diesem Fall erreicht CycleGAN Werte von 101,31 für Foto->Segmentierungskarte und 73,77 für Segmentierungskarte -> Foto. Die erzeugten Fotos hätten hier im Vergleich also den besten Wert. (Cycle-SN-GAN und Cycle-WGAN erreichen fast denselben Score wie mit dem unbeschnittenen Datensatz, sie haben die so aussortierten Bilder also nicht anders interpretiert.)

[9] Im Folgenden werden Teil der erhobenen Metriken präsentiert. In Gänze finden sich die ermittelten Werte in Anhang A, der auf der beigefügten DVD enthalten ist.

Tab. 6.1: FID-Scores nach 200 Trainingsepochen für das Pferd/Zebra- und das Cityscape-Beispiel. Der Wert in Klammern ist der FID-Score für CycleGAN ohne Berücksichtigung der Bilder, die von CycleGAN gleich interpretiert werden.

FID-Score	CycleGAN	Cycle-SN-GAN	Cycle-WGAN
Pferd -> Zebra	48,97	82,65	130,95
Zebra -> Pferd	61,35	124,04	138,68
Foto->Segmentierungs-karte	98,22(101,31)	95,80	97,96
Segmentierungskarte-> Foto	160,49(73,77)	218,09	159,18

Klassifizierer-Score Pferd/Zebra

Dass alle drei Netzwerke die Transformation Pferd->Zebra leichter erlernt haben als Zebra->Pferd, zeigt sich auch am Klassifizierer-Score - besonders für Cycle-SN-GAN und Cycle-WGAN:

Tab. 6.2: Klassifizierer-Scores nach 200 Trainingsepochen für das Pferd/Zebra-Beispiel

Klassifizierer-Score	Cycle-GAN	Cycle-SN-GAN	Cycle-WGAN
Pferd -> Zebra	88,9%	87,4%	57,9%
Zebra -> Pferd	85,8%	54%	25,8%

Der selbst trainierte Klassifizierer ordnet nach beendetem Training für den CycleGAN-Ansatz die erzeugten Zebrabilder im Schnitt mit 88,9% Wahrscheinlichkeit der Klasse Zebra zu und

die erzeugten Pferdebilder mit 85,8% Wahrscheinlichkeit der Klasse Pferd zu. Cycle-SN-GAN kommt hier auf je 87,4% und 54%, während Cycle-WGAN sogar nur 57,9% und 25,8% erreicht.

Es fällt auf, dass der Cycle-SN-GAN-Ansatz hier fast gleich gut wie der CycleGAN-Ansatz abschneidet, obwohl durch den FID-Score eine deutlich schlechtere Qualität attestiert wurde. Dies könnte dadurch erklärt werden, dass der Klassifizierer nur Auskunft darüber gibt, ob er ein Pferd bzw. Zebra erkennt, während der FID-Score ja, indem er die Feature-Vektoren für echte und generierte Bilder vergleicht, in gewisser Weise auch darüber Auskunft gibt, wie realistisch die erzeugten Bilder sind. Zusätzlich berücksichtigt der FID-Score auch, wie divers die erzeugten Bilder im Vergleich zur Zieldomäne sind. Ein schlechterer FID-Score bei ähnlich gutem Klassifizierer-Score könnte also darauf hinweisen, dass der Cycle-SN-GAN die Pferd->Zebra-Transformation zwar gut genug erlernt hat, um den Klassifizierer zu täuschen, die so erzeugten Bilder allerdings nicht so realistisch oder vielfältig sind wie die von CycleGAN erzeugten. Eine erneute Betrachtung der generierten Bilder zeigt auch tatsächlich, dass die von CycleGAN erzeugten Zebra-Bilder deutlich realistischer erscheinen, bzw. die erzeugten Zebra-Muster auch deutlich vielfältiger und differenzierter erscheinen.

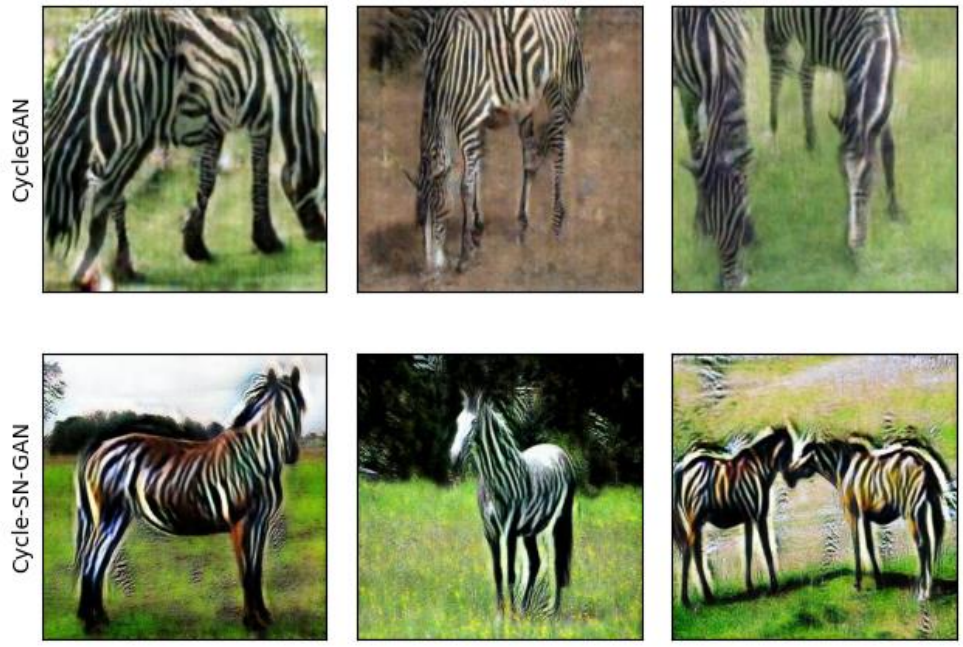

Abb. 6.5: Generierte Zebras von CycleGAN und Cycle-SN-GAN im Vergleich

FCN-Scores Cityscapes

Der Vergleich der erzielten FCN-Scores nach 200 Epochen Training bestätigt genau wie der FID-Score den Eindruck, dass die Transformation Foto->Segmentierungskarte leichter zu erlernen war als andersherum und von allen drei Ansätzen etwa gleich gut erlernt wurde, und darüber hinaus, dass der originale CycleGAN-Ansatz die Transformation Segmentierungskarte-Foto besser erlernt hat als Cycle-SN-GAN und Cycle-WGAN. So erlangt der CycleGAN-Ansatz trotz des partiellen Mode-Collapses Werte von 35,07%, 10,17% und 0,0583 für durchschnittliche Pixelgenauigkeit, durchschnittliche Pixelgenauigkeit pro Klasse und durchschnittliche Intersection-Over-Union pro Klasse im Vergleich zu 27,63%, 8,59% und 0,0451 für Cycle-SN-GAN und 28,93%, 9,33% und 0,0490 für Cycle-WGAN.

Tab. 6.3: FCN-Scores nach 200 Trainingsepochen für die Transformation Segmentierungskarte -> Foto im Cityscapes-Beispiel

Segmentierungskarte-> Foto	Cycle-GAN	Cycle-SN-GAN	Cycle-WGAN
Durchschnittliche Pixelgenauigkeit	35,07% (38,67%)	27,63%	28,93%
Durchschnittliche Pixelgenauigkeit pro Klasse	10,17% (11,06%)	8,59%	9,33%
Durchschnittliche IoU pro Klasse	0,0583 (0,0651)	0,0451	0,0490

Tab. 6.4: FCN-Scores nach 200 Trainingsepochen für die Transformation Foto -> Segmentierungskarte im Cityscapes-Beispiel

Foto -> Segmentierungs-karte	Cycle-GAN	Cycle-SN-GAN	Cycle-WGAN
Durchschnittliche Pixelgenauigkeit	46,91% (48,16%)	49,16%	48,97%
Durchschnittliche Pixelgenauigkeit pro Klasse	12,41% (13,07%)	12,41%	13,04%
Durchschnittliche IoU pro Klasse	0,0820 (0,0860)	0,0843	0,0866

Nach dieser ersten Sichtung der Zahlen scheint es eindeutig zu sein, dass die beiden implementierten Varianten (zumindest bei ansonsten gleich gewählten Parametern wie im originalen Ansatz) schlechtere Ergebnisse liefern als der originale Cycle-GAN-Ansatz. Es lohnt sich allerdings auch ein Blick auf den Verlauf der erhobenen Metriken über den Trainingsverlauf hinweg:

So erkennt man beim Pferd/Zebra-Beispiel, dass der originale CycleGAN-Ansatz einen starken Qualitätsgewinn über die ersten 40 Epochen in Pferd->Zebra-Richtung und in den ersten 90 Epochen in umgekehrter Richtung aufweisen kann, danach aber auf dem gleichen Niveau stagniert. Auch der Cycle-SN-GAN-Ansatz erreicht, wenn auch erst nach 130 Epochen, in Richtung Pferd->Zebra und 90 Epochen in umgekehrter Richtung ein Plateau. Der Cycle-WGAN-Ansatz hingegen hat zwar einen erkennbar weniger steilen Qualitätszuwachs in den ersten Epochen aufzuweisen, lernt dafür aber mehr oder weniger gleichmäßig über den gesamten Trainingsverlauf hinweg und ist auch nach 200 Epochen noch nicht am Stagnieren.

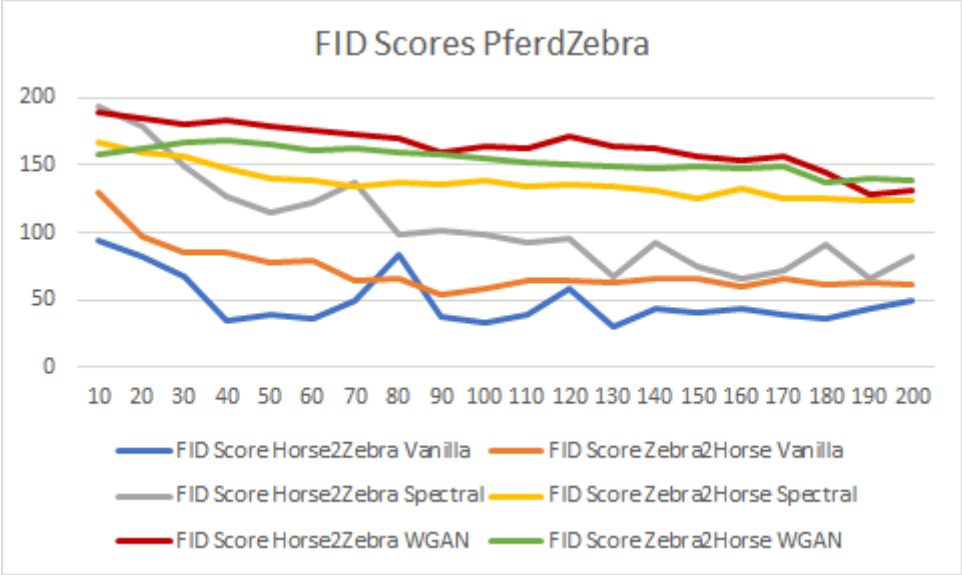

Abb. 6.6: FID-Scores für das Pferd/Zebra-Beispiel über den Trainingsverlauf

Im Cityscape-Beispiel ist dieser Trend etwas weniger stark ausgeprägt, dafür erkennt man hier noch besser als im Pferd/Zebra-Beispiel, dass Cycle-WGAN einen deutlich gleichmäßigeren, weniger instabilen Trainingsverlauf aufzuweisen hat als Cycle-SN-GAN und vor allem Cycle-GAN. So produzierte letzterer in den Epochen 120 und 140 in Richtung Foto->Segmentierungskarte nur bunten Pixelbrei, um sich danach wieder auf die vorherige Qualität einzupendeln.

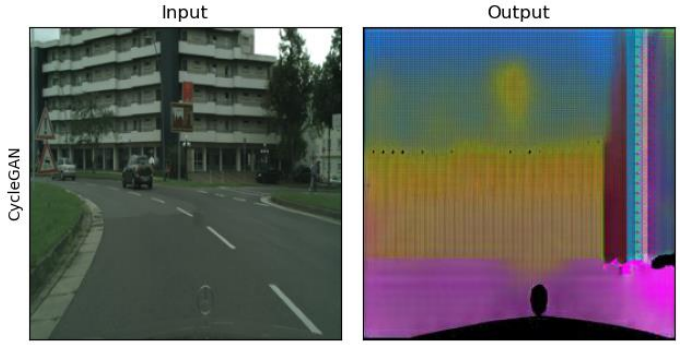

Abb. 6.7: Beispiel für Transformation von Foto -> Segmentierungskarte von CycleGAN nach Epoche 140 im Cityscapes-Beispiel

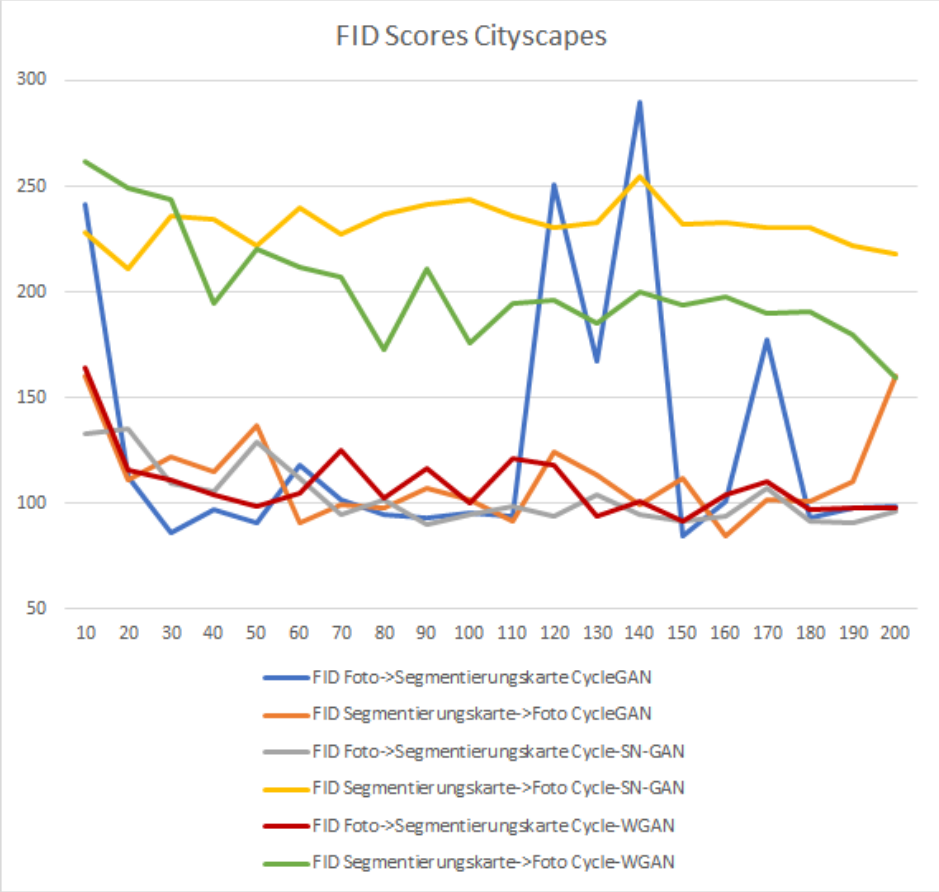

Abb. 6.8: FID-Scores für das Cityscapes-Beispiel über den Trainingsverlauf. Gut zu erkennen ist der instabilere Trainingsverlauf von CycleGAN, mit Score-Sprüngen in den Epochen 120, 140 und 170

7 Fazit und Ausblick

Die manuelle Auswertung der generierten Bilder sowie die Analyse der erhobenen Metriken über den Trainingsverlauf haben gezeigt, dass bei einer “naiven” Ergänzung des CycleGAN-Ansatzes um die Techniken der Spectral Normalization und des Wasserstein GANs (also ohne Anpassung der im originalen Ansatz gewählten Trainingsparameter), eher eine Verschlechterung als eine Verbesserung der Bildqualität zu beobachten ist. Allerdings zeigt sich auch, dass für Cycle-SN-GAN und Cycle-WGAN durch die Normalisierung der Diskriminatoren auf Lipschitz-Kontinuität wie erhofft ein stabilerer Trainingsverlauf erreicht werden konnte: Während der originale CycleGAN-Ansatz im Cityscape-Beispiel einen partiellen Mode-Collapse erlitten hat, ist dieser bei den beiden Alternativansätzen nicht zu beobachten gewesen.

In weiterführenden Versuchsreihen wäre entsprechend der nächste Schritt mit verschiedenen Lernraten für die beiden Varianten zu experimentieren. Da bei GANs eine zu hoch gewählte Lernrate häufig zu instabilem Training beiträgt, würde das stabilisierte Training von Cycle-SN-GAN und Cycle-WGAN womöglich eine höhere Lernrate (und damit ein schnelleres Konvergieren) zulassen, ohne dass die Qualität darunter leidet.

Zudem zeigte sich für den Cycle-WGAN-Ansatz, dass auch nach 200 Trainingsepochen immer noch Lernfortschritte zu beobachten waren. Eine mögliche Fortführung der Untersuchung wäre hier also, das Training einfach so lange weiterzuführen, bis keine Verbesserung der Qualität mehr zu beobachten ist. Wenn die dann erzielte Qualität höher als beim originalen CycleGAN-Ansatz ist, wäre also zumindest ein Qualitätsgewinn, wenn auch auf Kosten der Trainingszeit, festzustellen.

Es lässt sich also schließen, dass Spectral Normalization und Wasserstein-GAN als Modifikation der CycleGAN-Architektur nur auf den ersten Blick erfolglos waren, aber auf den zweiten Blick eine weitere, größer angelegte Untersuchung durchaus rechtfertigen.

Literaturverzeichnis

[Arjovsky-2017]: Arjovsky, M.; Chintala, S. & Bottou, L.: Wasserstein GAN, In: *CoRR* abs/1701.07875, 2017

[Das-2020]: Das, S.: The AI Behind FaceApp, - https://analyticsindiamag.com/the-ai-behind-faceapp/ - Zugriffsdatum: 17.03.2021, 2020

[Dumoulin-2016]: Dumoulin, V. & Visin, F.: A guide to convolution arithmetic for deep learning, In: *CoRR* abs/1603.07285, 2016

[Goodfellow-2014]: Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A. C. & Bengio, Y.: Generative Adversarial Networks, In: *CoRR* abs/1406.2661, 2014

[Goodfellow-2017]: Goodfellow, I. J.: NIPS 2016 Tutorial: Generative Adversarial Networks, In: *CoRR* abs/1701.00160, 2017

[Goodfellow-2018]: Goodfellow, I.: Deep Learning : Das umfassende Handbuch : Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze, Frechen: *Verlags GmbH & Co. KG.*, 2018

[Gulrajani-2017]: Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V. & Courville, A. C.: Improved Training of Wasserstein GANs, In: *CoRR* abs/1704.00028, 2017

[Hendrycks-2018]: Hendrycks, D.; Mazeika, M. & Dietterich, T. G.: Deep Anomaly Detection with Outlier Exposure, In: *CoRR* abs/1812.04606, 2018

[Heusel-2017]: Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B. & Hochreiter, S.: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, In: Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.

& Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA.*, 2017, S. 6626--6637

[Isola-2016]: Isola, P.; Zhu, J.-Y.; Zhou, T. & Efros, A. A.: Image-to-Image Translation with Conditional Adversarial Networks, In: *CoRR* abs/1611.07004, 2016

[Karacan-2016]: Karacan, L.; Akata, Z.; Erdem, A. & Erdem, E.: Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts, In: *CoRR* abs/1612.00215, 2016

[Long-2014]: Long, J.; Shelhamer, E. & Darrell, T.: Fully Convolutional Networks for Semantic Segmentation, In: *CoRR* abs/1411.4038, 2014

[Miyato-2018]: Miyato, T.; Kataoka, T.; Koyama, M. & Yoshida, Y.: Spectral Normalization for Generative Adversarial Networks, In: *CoRR* abs/1802.05957, 2018

[Raschka-2018]: Raschka, S.: Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow : Konzepte, Tools und Techniken für intelligente Systeme, Heidelberg: *O'Reilly dpunkt.verlag*, 2018

[Reder-2019]: Reder, B.: Noch viel zu tun bei KI und ML. - https://www.computerwoche.de/a/noch-viel-zu-tun-bei-ki-und-ml,3547342 – Zugriffsdatum: 17.03.2021, 2019

[Salimans-2016]: Salimans, T.; Goodfellow, I. J.; Zaremba, W.; Cheung, V.; Radford, A. & Chen, X.: Improved Techniques for Training GANs, In: *CoRR* abs/1606.03498, 2016

[Sangkloy-2016]: Sangkloy, P.; Lu, J.; Fang, C.; Yu, F. & Hays, J.: Scribbler: Controlling Deep Image Synthesis with Sketch and Color, In: *CoRR* abs/1612.00835, 2016

[Zhu-2017]: Zhu, J.-Y.; Park, T.; Isola, P. & Efros, A. A.: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, In: *CoRR* abs/1703.10593, 2017

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original