

BACHELORTHESIS
Huy-Bao Le

Erzeugung realistischer Trainingsdaten mit generativen Neuronalen Netzen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Huy-Bao Le

Erzeugung realistischer Trainingsdaten mit generativen Neuronalen Netzen

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

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Michael Neitzke

Eingereicht am: 17.06.2021

Huy-Bao Le

Thema der Arbeit

Erzeugung realistischer Trainingsdaten mit generativen Neuronalen Netzen

Stichworte

Maschinelles Lernen, Neuronale Netzwerke, Generative Adversarial Networks, Erzeugung von Trainingsdaten

Kurzzusammenfassung

Machine Learning ist in der heutigen Zeit nicht mehr wegzudenken. Ein Teilbereich ist dabei das Unsupervised Learning, welche in dieser Arbeit in Verbindung mit generativen neuronalen Netzwerken angewendet wird. Jeder Teilbereich des Machine Learnings hat dabei eines gemeinsam: jedes neuronale Netz braucht Trainingsdaten. Gerade in dem Bereich der Bildverarbeitung kommt es umso häufiger vor, dass Datensätze nicht vorhanden oder unausreichend sind für das Training eines robusten Models. In dieser Arbeit geht es darum, mit Hilfe von verschiedenen GAN-Architekturen, synthetische Datensätze zu generieren. Diese generierten Datensätze könnten vom großen Nutzen sein um weitere Machine Learning Modelle zu trainieren, optimieren und robuster zu machen. Immer im Blick soll dabei die Erzeugung von realistisch aussehenden Daten im Vordergrund stehen. Über die Grundlagen, die Hyperparameter beim Training bis zum Effekt von der Beschaffenheit der Datensätze beim Training von GANs werden verschiedene Aspekte, Erkenntnisse und Methoden vorgestellt um realistische Daten zu generieren. Dabei wird auch ein weiterer Ansatz über die Computergrafik mit der Anwendung von BlenderProc in dieser Arbeit angesprochen. Als möglichen Ansatz für die Erzeugung der Daten mit generativen Netzen ist die Image-to-Image Methode gewählt worden, welches ein Teilgebiet des Styletransfers von Bildern ist. Die Idee hierbei ist es herauszufinden ob Bilder aus einer bestimmten Style-Domain in realistische Bilder umgewandelt werden können. Die dabei zu vergleichenden Architekturen sind die von CycleGAN und StyleGAN. Beide Architekturen sind State-of-the-Art Methoden und haben verschiedene Ansätze für die Realisierung des Prozesses. Über die Implementierung beider Modelle in Verbindung mit dem monet2photo Datensatz, wird festgestellt ob das Ziel der synthetischen Erzeugung von Daten möglich ist und welche Architektur dafür besser geeignet ist.

Huy-Bao Le

Title of Thesis

Generating training datasets with generative neural networks

Keywords

Machine Learning, Neural Networks, Generative Adversarial Networks, Generating Datasets

Abstract

Machine Learning has become an integral part of this world. Unsupervised learning in conjunction with generative neural networks is part of this thesis. Every area of machine learning does have something in common: every neural network needs datasets for training. There is a problem of finding fitting datasets in order to train stable models especially in parts of image processing. This thesis is about generating synthetic datasets together with generative adversarial networks, so called GANs. Those generated datasets could be a real benefit for training other machine learning models while optimizing and stabilizing them. Always in mind there should be the fact of generating realistic looking data. Therefore this thesis covers the basics, the hyperparameters when training and the effect of datasets of GANs. Another approach is given through the area of computer graphics via BlenderProc. The used approach in this thesis for generating data with GANs is the image-to-image method, which is a part of image style transfer. The idea is to translate images from a specific style-domain into realistic looking images. Furthermore the architecture of CycleGAN and StyleGAN are compared in this work. Both architectures are state-of-the-art methods and have different approaches of realizing this problem. With the implementation of both models combined with the monet2photo dataset it is proofed whether the goal of generating synthetic realistic looking data is possible and which one is more suitable to do so.

Inhaltsverzeichnis

Abbildungsverzeichnis	vii
Tabellenverzeichnis	ix
1 Einleitung	1
1.1 Zielsetzung	2
1.2 Struktur der Arbeit	3
2 Grundlagen von GANs	5
2.1 Generator	6
2.2 Discriminator	9
2.3 Generator vs. Discriminator	9
3 Training von GANs	11
3.1 Evaluierung von GANs	11
3.2 Probleme beim Training	12
3.3 Hyperparameter und ihre Effekte	14
3.4 Optimierungen und Stabilisierung von GANs	18
3.4.1 Modeldesign	18
3.4.2 Datensatz	19
4 Generierung von Daten mit der Computergrafik	20
4.1 Datensatz Augmentierung	22
5 Architekturen	25
5.1 Datensatz	25
5.2 Cycle GAN	27
5.2.1 Architektur	27
5.2.2 Training	32
5.2.3 Evaluation	33

5.3	Style GAN	36
5.3.1	Architektur	37
5.3.2	Training	41
5.3.3	Evaluation	42
6	Gesamt Evaluation	45
7	Future Work	47
7.1	Fazit	47
	Literaturverzeichnis	49
A	Anhang	53
	Selbstständigkeitserklärung	61

Abbildungsverzeichnis

2.1	Sand clock Architektur eines Autoencoders.	6
2.2	Plot des <i>latent spaces</i> eines Encoders anhand des MNIST Datensatzes. . .	7
2.3	Convolutional Transpose Layer mit einem 2×2 Kernel, padding=0, stride=1. .	8
2.4	Plot des <i>latent spaces</i> mit der Erweiterung durch die Normalverteilung und einer optimierten <i>loss function</i>	8
2.5	AlexNet Architektur. Ein typisches Deep Convolutional Neural Network (CNN).	9
2.6	Generelle Struktur von GANs.	10
3.1	Wolf oder Hund?	12
3.2	Mode Collapse. Der Generator erzeugt stets verrauschte Daten.	13
3.3	Mode Collapse. Der Generator erzeugt stets ähnliche bzw. die gleichen Daten.	14
3.4	Erkennen eines mode collapse Anhand des losses des Models. Grün = D_{real} , Rot = D_{fake} , Schwarz = $\emptyset D_{real+fake}$, Gelb = G	15
3.5	x-Achse: Epoche, Orange: Batchgröße 64, Blau: Batchgröße 256, Lila: Bat- chgröße 1024.	16
3.6	x-Achse: Epoche, Orange: Batchgröße 64 mit lr 0.01, Blau: Batchgröße 1024 mit lr 0.1, Lila: Batchgröße 1024 mit lr 0.01.	17
4.1	Star Wars Szene gerendert mit der Raytracing Methode und Unreal Engine. .	20
4.2	BlenderProc synthetisierte Labels einer gerenderten Szene.	21
4.3	Anwendung einer Bildsegmentierung anhand synthetischen, hyperrealisti- schen Bildern.	22
4.4	Pipeline für die Generierung von Datensätzen.	23
4.5	Generierung von realistischen Daten mit Hilfe von simulierten Daten und Cycle-GAN.	23
4.6	Datenaugmentierung durch Änderung der Farbwerte.	24

5.1	Beispiele aus der Monet-Domäne.	26
5.2	Beispiele aus der Echten Welt-Domäne.	27
5.3	Cycle-GAN Idee.	28
5.4	Cycle-consistency loss.	28
5.5	Cycle-GAN Discriminator Architektur.	29
5.6	Cycle-GAN Generator Architektur.	30
5.7	Erhalt der Bildtemperatur durch die Einführung des Identity loss.	31
5.8	Illustration der losses im Cycle-GAN.	32
5.9	Zwischenergebnisse während des Cycle-GAN Trainings.	33
5.10	Total loss während des Trainings.	36
5.11	Cycle loss in Orange, Summe der Discriminator loss in Blau und ID loss in Rot.	37
5.12	Übersetzung des Style Vektor w und die Verwendung der Parameter y_s und y_b im Generator des Style-GANs.	38
5.13	Adaptive Instance Normalization. Scale y_s und Bias y_b und die modifizierte Instance Normalization.	38
5.14	Beispiel für den Effekt von groben- und feineren Noise Layern. Größere Noise Layer führen zu größere Locken und größeren Hintergrund Featu- res (a)/(b), feine Noise Layer zu feineren Locken und feiner Hintergrund Features (c)/(d).	39
5.15	Weiteres Beispiel für den Effekt von groben- und feineren Noise Layern. Dabei wurde der Style und Noise Layer der vorderen Layern übernommen. High-level Details wie Pose, Gesichtsform und Brille wurden von Source B übernommen.	40
5.16	Illustration der implementierte reduzierten StyleGAN Architektur.	41
5.17	Zwischenergebnisse verschiedenster Epochen während des Trainings.	42
5.18	StyleGAN Loss während des Trainings. In Orange der geplottete Loss des Generators. In Blau der Loss des Discriminators.	44
A.1	Evaluierungsdaten Cycle-GAN in die Realität, positiv	54
A.2	Evaluierungsdaten Cycle-GAN in die Realität, negativ	55
A.3	Evaluierungsdaten Cycle-GAN in die Monet-Domäne, positiv	56
A.4	Evaluierungsdaten Cycle-GAN in die Monet-Domäne, negativ	57
A.5	Evaluierungsdaten StyleGAN, Positiv	58
A.6	Evaluierungsdaten StyleGAN, Negativ 1. Teil	59
A.7	Evaluierungsdaten StyleGAN, Negativ 2. Teil	60

Tabellenverzeichnis

4.1	Ergebnisse der Evaluierung mit echten und synthetisch erzeugten Daten. .	24
5.1	Aufbau des monet2photo Datensatzes.	26
5.2	Erwartete Trainingszeiten für Style-GAN für verschiedene Outputgrößen. .	40

1 Einleitung

Datensätze sind Bestandteil jedes Neuronalen Netzes. Machine Learning (ML) integriert dabei diese Datensätze und verwendet diese um verschiedene Modelle zu trainieren. Dabei gibt es verschiedene Datensätze, unter anderem im Bereich der Bildverarbeitung (image processing), bei dem die Daten aus Bildern bestehen. Diese werden unter anderem für Objekterkennungs- und Klassifizierungs-Netzwerke verwendet. Um robuste und akkurate Ergebnisse der Modelle zu erzielen benötigt man eine hohe Varianz an Schlüssel-Merkmalen und eine große Menge an Bildern im Datensatz. Qualitativ gute, gelabelte Datensätze für überwachte Trainingsalgorithmen (supervised learning) sind schwer zu finden und meistens teuer in der Produktion. Die Erstellung solcher hoch variantenreichen Datensätzen ist zudem mit einem hohen Zeitfaktor geprägt. Da ML in der Verbindung mit der Bildverarbeitung immer weiter voranschreitet, ist der Drang an Datensätzen also stetig am wachsen.

Diese Bachelorarbeit beschäftigt sich mit dem Thema der Erzeugung von realistischen Trainingsdaten und ob es überhaupt möglich ist realistisch wirkende Bilder mit Hilfe von generativen neuronalen Netzwerken zu generieren. Ein Teilbereich von generativen Netzen ist ein Generative Adversarial Network (GAN) [11]. Präsentiert von Ian Goodfellow im Jahr 2014, ist die Framework mittlerweile Grundlage von vielen weiteren generativen Modellen. Auf Basis dieser GANs und der image-to-image Methode [14] wird versucht realistisch aussehende Bilder zu generieren welche in Zukunft genutzt werden können um Datensätze zu Erschaffen oder bereits vorhandene Datensätze zu erweitern. Der image-to-image Ansatz, vorgestellt zum ersten mal im Jahr 2016, versucht dabei eine Überführung zwischen zwei verschiedenen Domänen zu ermöglichen. Konkret wird versucht ein Bild von einer Domäne in eine andere zu übersetzen. Dabei bietet die Computergrafik eine Möglichkeit Bilder für die eine Domäne einfach synthetische zu generieren.

Für die Umsetzung der Arbeit werden zunächst die Grundlagen von GANs geschaffen. Dieses deckt den generellen Grundgedanken und die Funktionsweise der vorgestellten Architekturen ab. Die benutzte Literatur für die Grundlagen basieren auf dem Buch von

Davis Foster, Generative Deep Learning [9]. Auch auf das Training und die Hyperparameter und die Veränderung derer wird in dieser Arbeit mit behandelt.

Wie bei jedem Training eines ML Model können Probleme auftreten, gerade bei GANs. Dadurch, dass zwei neuronale Netze miteinander konkurrieren ist das gesamte Netz sehr filigran und es können mögliche Seiteneffekte auftreten. Diese werden nach möglichen Wissensstand analysiert und es wird eine Lösung für die Vermeidung solcher Probleme dargestellt.

Für die synthetische Erzeugung von Bildern aus einer Domain wird BlenderProc [7] präsentiert. BlenderProc ist ein Ansatz aus der Computergrafik welches bereits echt wirkende Bilder für verschiedenste ML Modelle generieren kann.

Weiterhin werden zwei GAN-Architekturen vorgestellt, CycleGAN [30] und StyleGAN [17]. Die CycleGAN Architektur wurde 2017 veröffentlicht und beinhaltet die Übersetzung von Bildern in beide Richtungen zwischen den Domänen.

Das Konzept von StyleGAN wurde von NVIDIA zum ersten mal im Jahre 2018 veröffentlicht. Im Rahmen dessen wurden in der originalen Version Gesichter von realistisch aussehenden Menschen generiert. Im Dezember 2019 wurden die Qualität der generierten Bilder nochmal verbessert und StyleGAN2 wurde vorgestellt. Beide Architekturen werden in dieser Arbeit implementiert und mit Hilfe des monet2photo Datensatzes ein Konzept entwickelt welches die Übersetzung von einer Domain in die andere verdeutlichen soll.

Das Ergebnis der Arbeit bietet eine vielversprechende Ergebnisse und somit weitere mögliche Ausgangspunkte um in Zukunft größere und komplexere Datensätze zu erschaffen.

1.1 Zielsetzung

In den folgenden Kapiteln wird die Grundlage geschaffen zur Generierung von realistischen Bildern. Die dabei entstandene Frage welche in Zusammenarbeit mit dem betreuenden Prüfer, Prof. Dr.-Ing. Andreas Meisel, entstand lautet: "Ist es möglich Datensätze synthetisch zu erzeugen, welche genutzt werden können um Modelle zu trainieren welche echte Bilder verwenden?". Anlässlich dieser Frage und der stetig voranschreitenden Algorithmen für generative neuronale Netze wurde die Hypothese aufgestellt, dass es möglich sein müsste Datensätze zu erschaffen welche für das menschliche Empfinden wirklichkeitsgetreu wirken. Diese Datensätze müssten dann in der Lage sein, aufgrund der Nähe

zur Echtheit, Modelle zu trainieren und dadurch einen konkreten Nutzen in der realen Anwendung zu ermöglichen.

State-of-the-art Architekturen basierend auf der GAN-Architektur und der image-to-image Übersetzung werden verwendet um Daten zu konstruieren, welche auf das menschliche Auge realistisch wirken. Mit einem Proof-of-Concept soll auf der zuvor erstellten Basis ein Vergleich geschaffen werden welche die Architekturen und Ergebnisse von CycleGAN [30] und StyleGAN [17](2)[20] gegenüberstellt.

1.2 Struktur der Arbeit

Für die Methodik dieser Bachelorarbeit wird ein stetig wachsender Prozess bestrebt.

Somit werden in Kapitel 2 die Grundlagen, Funktionsweise und das Konzept von GANs erklärt. Die Grundlagen zeigen die allgemeine Idee und den typischen Aufbau der Art von generativen Netzen. Es wird beschrieben wie der Gedanke der GANs zwar die gewohnten Konzepte des supervised learning verwendet, jedoch diese erweitert um nicht deterministische Modelle zu entwickeln.

Kapitel 3 befasst sich mit der Thematik des Trainings der Modelle. Wie beim Training aller ML Modelle, müssen die Hyperparameter festgelegt werden. Einige dieser Hyperparameter tragen dazu bei Seiteneffekten während des Trainings und somit auf das Ergebnis zu verursachen. Diese Erkenntnisse sollen genutzt werden um bei der Implementierung der Modelle ein möglichst effektives und gutes Resultat zu erhalten. Aufgrund der komplexen Struktur und dem Zusammenspiel zweier empfindlicher Netze sind Probleme beim Training und somit auch das Versagen des Models keine Seltenheit. Hier geht es darum die möglichen Gefahren beim Training zu vermindern. Dabei reflektiert das Kapitel die Erfahrungen die über die Jahre mit GANs gesammelt wurden und erklärt unter anderem wie die Beschaffenheit von Trainingsdaten eine Auswirkung auf das Training hat.

Um einen anderen Ansatz für die Erzeugung von Daten geht es im Kapitel 4. In Anbetracht der voranschreitenden Entwicklung in der Computergrafik, sehen computergenerierte Bilder immer realistischer aus. Gemäß der Veröffentlichung [7] ist dieses Tool ein direkter Ansatz für die Erzeugung realistisch aussehender Bilder mit dem Open-Source Computergrafiktool Blender [6]. Dabei benutzt BlenderProc eine modulare Pipeline, welche in der Lage ist vielseitige Szenen zu entwerfen und in hochauflösenden Bildern zu rendern.

Das Kapitel 5 stellt die zwei ausgewählten Architekturen, CycleGAN [30] und StyleGAN [17] vor. In diesem Abschnitt der Arbeit geht es darum aktuelle State-of-the-Art Implementierungen vorzustellen. Dabei werden auf die jeweiligen Architekturen, die Besonderheiten, das Training der Modelle und die Evaluation beider Modelle in Verbindung mit dem monet2photo Datensatz eingegangen. Beide Modelle werden zunächst separat evaluiert.

Der endgültige Vergleich beider Netze wird im Kapitel 6 dargestellt. Dabei werden die Ergebnisse beider Modelle gegenübergestellt, die Vor- und Nachteile beider Architekturen kritisch hinterfragt und in Hinblick auf die Zielsetzung bewertet.

Im letzten Kapitel 7 geht es um die Frage ob der gewählte Ansatz und Methodik der Zielsetzung und der Hypothese gerecht wird. Basierend auf dieser Arbeit werden im letzten Teil der Arbeit noch mögliche weiterführende Arbeiten angesprochen.

2 Grundlagen von GANs

Generative Adversarial Networks - zu deutsch etwa "Generative gegenstellige Netze" ist ein Teilgebiet des ML und beschreibt das unüberwachte Lernen zweier neuronalen Netze. Vorgestellt wurde die Idee von Ian Goodfellow, einem Mitarbeiter von Google, im Oktober 2016. Die GAN Architektur beschreibt ein Zusammenspiel von zwei Netzen, welche beide die bis dahin weit verbreitete Methode *supervised learning* verwenden. Dabei wird das Problem begrenzt indem zwei Submodelle separat voneinander trainiert werden um sich gegenseitig auszuspielen und voneinander zu lernen. Der *Generator* beschreibt dabei ein Netz, welches dazu dient plausible Ergebnisse von einem bestimmten Typ zu generieren, einer sogenannten Domäne. Eine Domäne kann dabei eine bestimmte Art von Bildern sein, zum Beispiel Bilder von Hunden. Alle Bilder von Hunden wären dann Teil der Domäne. Gegenüber steht das zweite Netz, welches versucht die erzeugten Bilder vom *Generator* zu klassifizieren und zu bewerten. Der sogenannte *Discriminator* prüft dabei ob die erzeugten Bilder Teil der eigentlichen Zieldomäne ist oder ob diese ein Fake sind. Dieser Ansatz dient bis heute als Grundlage für weitere GANs und zeigte bis dato einen neuen Weg für die Generierung von Daten.

Die Schwierigkeit bei einer solchen Architektur steckt in der Idee an sich. Anders wie beim *supervised learning* beruht die Generierung der Daten auf einer Wahrscheinlichkeit statt auf deterministische Algorithmen. Beim *supervised learning* wird stetig das gleiche Ergebnis erwartet bei gleichem Input, bei einem generativen Ansatz kommt die Randomisierung ins Spiel, sodass bei gleichbleibenden Parametern ein komplett anderes Ergebnis erzielt werden kann. Das Model muss also ein stochastisches, zufällig Element besitzen um verschiedene Samples zu generieren.

Die ersten Schritte zur Generierung von Daten wurden vorgestellt im Jahr 2013. Autoencoder (AE) sind fundamentale Strukturen, welche *Deep Learning*-Algorithmen verwenden um generative Netze zu entwerfen. Im folgenden Abschnitt wird erklärt wie der *Generator* auf Basis von AEn funktioniert was diese im Grundgedanken vereinigt.

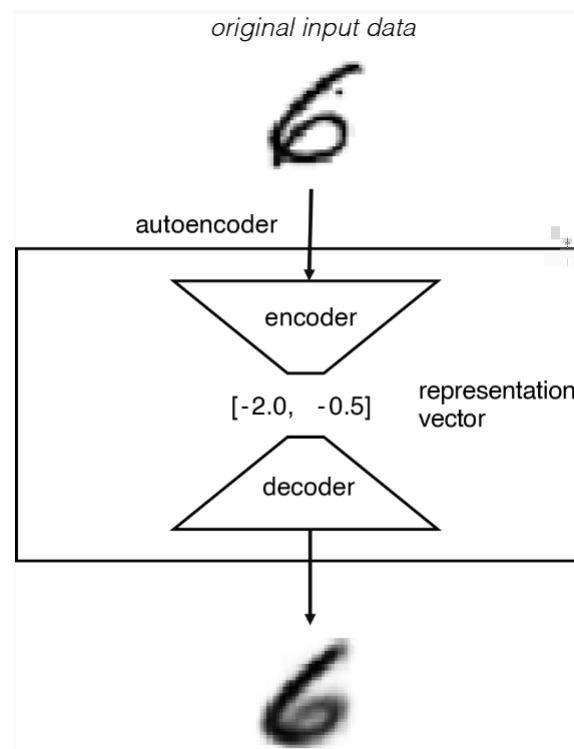


Abbildung 2.1: Sand clock Architektur eines Autoencoders.

Quelle: Generative Deep Learning, Chapter 3, Autoencoders [9]

2.1 Generator

Generative Netze, welches der *Generator* und *AEs* sind, vereinigt den Grundgedanken einer *feedforward* Pipeline. Diese nimmt einen Input nimmt und versucht über zufällige Elemente einen Output zu generieren, welcher den Eigenschaften des Inputs gleichen. Dabei ist die Idee eines Autoencoders die Aufspaltung des Netzes in einen **Encoder** und einen **Decoder**. Mit einer Sanduhr Architektur (siehe Abb. 2.1) wird versucht die Attribute des Input zu identifizieren und diese dann zu rekonstruieren.

Der *Encoder* versucht dabei die Informationen aus den Inputdaten auf einen Raum zu projizieren, welche die Eigenschaften des Inputs in dem selbstgeschaffenen Raum mappt. Bezeichnet wird dieser Raum als Latenter Raum (*latent space*). In diesem *latent space* sollen alle Merkmale der Inputdomäne abbildbar sein. Bei einem *Encoder* wird dabei der Input auf einen Vektor in diesem *latent space* abgebildet. Für die Interpretation dient ein einfaches MNIST Beispiel, bei dem die einzelnen Ziffern des Inputbildes auf einen

2D-Vektor abgebildet werden. Der Plot dieser Projektion ist eingefärbt mit den Ziffern (siehe Abb. 2.2).

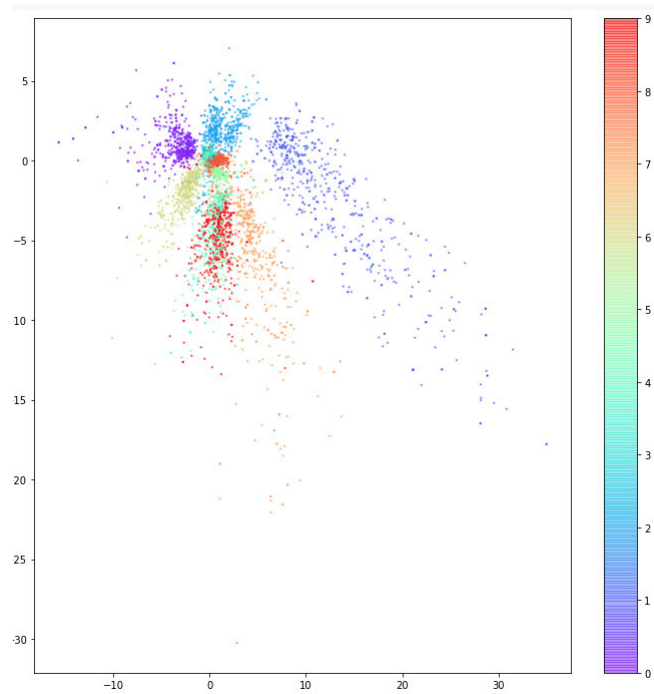


Abbildung 2.2: Plot des *latent spaces* eines Encoders anhand des MNIST Datensatzes.

Quelle: Generative Deep Learning, Chapter 3, Autoencoders [9]

Der *Decoder* versucht aus dem gemappten Vektor den Input wieder herzurichten. In diesem Schritt wird die Randomisierung über Verwendung von ***Convolutional Transpose***-Layern realisiert. Man kann sich diese Layer so vorstellen, dass es das Gegenteil von Faltungen (*convolutions*) ist. Bei der "Auffaltung" werden, anders als beim *Upsampling* Layer, die einzelnen Werte nicht einfach dupliziert sondern es wird versucht über eine Vergrößerung des Input Tensors und interpolieren von Nullwerten mit einem eigenen Faltungskern (*kernel*) ein trainierbaren Kernel zu erschaffen. Für ein einfaches Verständnis der *Convolutional Transpose* Layer ist die Abbildung 2.3. Mathematisch betrachtet ist die ein Convolutional Transpose Layer eine injektive Abbildung, bei der von einem kleineren Raum in einen größeren Raum abgebildet wird.

GANs verwenden die gleiche Semantik von AE, werden jedoch erweitert um eine Normalverteilung. Da das Projektion des Encoders deterministisch ist, wird der gleiche Input immer wieder auf den gleichen Vektor im *latent space* gemappt. Wie man aber in Abbildung 2.2 erkennt, ist die Verteilung der Inputmerkmale nicht gleichmäßig verteilt. Die

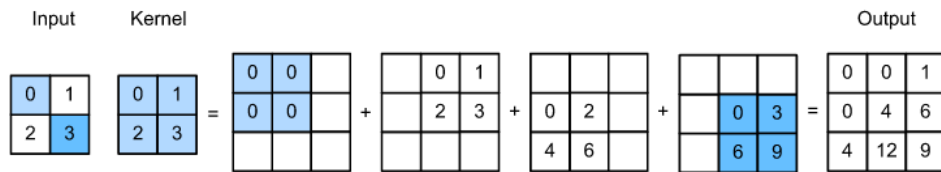


Abbildung 2.3: Convolutional Transpose Layer mit einem 2×2 Kernel, padding=0, stride=1.

Quelle: [http : //d2l.ai/chapter_computer - vision/transposed - conv.html](http://d2l.ai/chapter_computer_vision/transposed_conv.html), Fig. 13.10.1

Normalverteilung soll dem Encoder eine gewisse Wahrscheinlichkeit bei der Weiterreichung an den Decoder geben, dadurch lernt der Generator effektiver, da der *latent space* um die deterministischen Punkte mit entdeckt werden. In der Verbindung mit einer gut gewählten *loss*-Funktion kann dadurch der *latent space* so verändert werden, dass die Distribution von Eigenschaften der Input-Domäne geballter vorliegen. Am Beispiel eines *Variational AE* und der Normalverteilung mit der Kullback-Leibler-loss function und des MNIST Beispiels ist die Veränderung des *latent space* deutlich zu sehen 2.4.

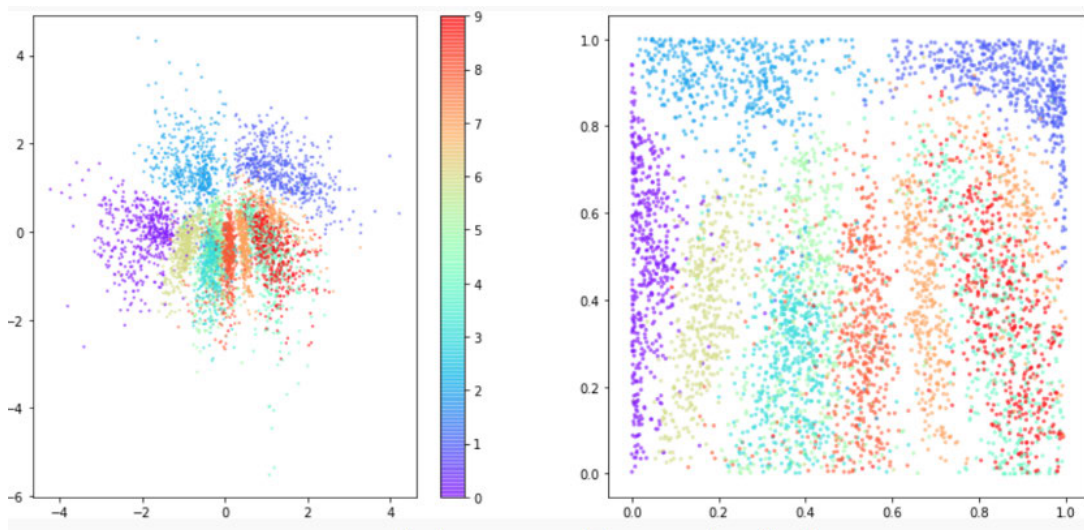


Abbildung 2.4: Plot des *latent spaces* mit der Erweiterung durch die Normalverteilung und einer optimierten *loss function*.

Quelle: Generative Deep Learning, Chapter 3, Analysis of the Variational Autoencoder [9]

2.2 Discriminator

Der *Discriminator* ist lediglich ein Klassifizierungs-Netz. Die Architektur gleicht den bereits bekannten Klassifizierungs-Netzwerken und gleicht dem Aufbau von CNNs. Ein typischer Aufbau wäre zum Beispiel AlexNet (siehe Abb. 2.5), welche *convolution layers*, *maxpool layers* und *fully connected layers* verwendet.

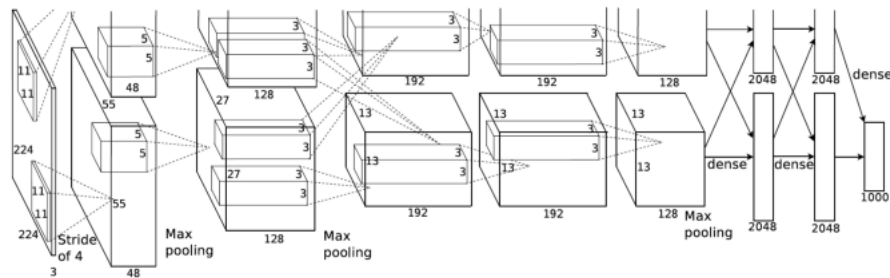


Abbildung 2.5: AlexNet Architektur. Ein typisches Deep CNN.

Quelle: ImageNet Classification with DCNNs [21]

2.3 Generator vs. Discriminator

Wie in den beiden Sektionen vorgestellt werden zwei Netze getrennt voneinander trainiert und versuchen sich dadurch gegenseitig zu verbessern. Der Output des *Generators* ist direkt verbunden mit dem Input des *Discriminators*. Parallel zum Output des *Generators* besitzt der *Discriminator* noch einen weiteren Input, welche die eigentlich echten Bilder der Trainingsdaten enthält. Eine Illustration der Struktur ist gegeben in Abbildung 2.6. Das Ergebnis des *Discriminators* leitet dann den Vorgang der *backpropagation* ein. Dabei wird der Output des *Discriminators* verwendet um die Gewichte des *Generators* zu aktualisieren. Der genaue Vorgang wird im nächsten Kapitel beschrieben.

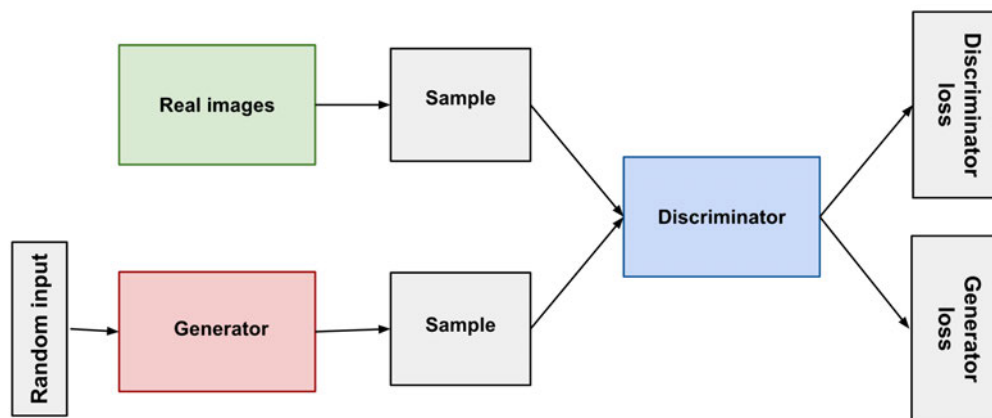


Abbildung 2.6: Generelle Struktur von GANs.

Quelle: https://developers.google.com/machine-learning/gan/gan_structure

3 Training von GANs

Die Schwierigkeit beim Trainieren von GANs ist das Zusammenspiel beider Trainingseinheiten von *Discriminator* und *Generator*. Das Ziel ist es ein stabiles Model zu schaffen, welches bis zu einem bestimmten Ergebnis konvergieren soll. Beide Netze werden getrennt voneinander trainiert. Dabei trainieren beide Netze im Wechsel zueinander, zunächst wird der *Discriminator* trainiert, dann der *Generator*. Wichtig ist, dass beim Training des einen Netzes, die Gewichte des anderen nicht verändert werden dürfen. Grund für die Einfrierung der Gewichte ist das Wechselspiel beider Netze zueinander. Da ein gutes Ergebnis für den *Discriminator* folglich ein schlechtes Ergebnis für den *Generator* ist und anders herum, würde die *Backpropagation* nach jeder Trainingseinheit immer das eine Netz verbessern werden und das andere dementsprechend verschlechtert. Insgesamt funktioniert die *Backpropagation* genauso wie in einem CNN. Nach dem Output des *Discriminators* werden die Gewichte über entsprechend der *loss function* aktualisiert. Das Gleichgewicht beider Netze ist dabei wichtig um den Erfolg des Models zu gewährleisten. Um die möglichen Probleme, wie man diese umgeht und Optimierungen des Models geht es im nächsten Kapitel. In diesem Kapitel geht es um die Trainings-Hyperparameter.

3.1 Evaluierung von GANs

Die Evaluierung von GANs erweist sich als schwierig. Die Bewertung von generativen Netzen ist bis heute ein großes Problem. Grund ist, dass GANs keine *objective loss function* haben. Das liegt an der Beschaffenheit der Netztopologie. Die Bewertung des *Generators* wird über einen sich stetig veränderten *Discriminator* vollzogen. Vorstellen kann man sich dieses Problem, indem die Abbildung 3.1 betrachtet wird. Die Bewertung und Unterscheidung von Wolf und Hund ist auch für uns Menschen eine schwierige Aufgabe. Ab welcher Länge an Zähnen oder Größe des Tieres auf dem Bild ist es ein Wolf? Wann ist es ein Hund, wann ein Wolf? Der *Discriminator* versucht dabei die Merkmale von zusammenhängender Eigenschaften zu bündeln und versucht diese zu unterscheiden.



Abbildung 3.1: Wolf oder Hund?

Quelle: [http : //shibashindig.blogspot.com/2012/01/ancestors – of – shiba – inu.html](http://shibashindig.blogspot.com/2012/01/ancestors-of-shiba-inu.html)

Die eigentliche Evaluierung von generativen Modellen erfolgt zum größten Teil manuell. Beim Training werden dabei regelmäßig Samples generiert, um zu prüfen ob das Training dem gewünschten Zielergebnis näher kommt. Bei der Bildverarbeitung werden dementsprechend Samples während des Trainings evaluiert, bei anderen generativen Modellen wie etwa aus dem *natural language processing* werden Tonsamples evaluiert [3]. Diese Tatsache erschwert auch das Training von GANs. Aufgrund der fehlenden *objective loss function* kann von daher nur schwer beurteilt werden wann generative Modelle konvergieren und nicht mehr besser werden. Zudem ist die händische Evaluierung über die menschliche Wahrnehmung teuer, schwerfällig, objektiv und damit auch schwer zu reproduzieren [2].

3.2 Probleme beim Training

Schlussfolgernd aus dem bereits gewonnen Erkenntnissen sind zwei Probleme schon genannt wurden. Man kann nicht wirklich einschätzen wann Modelle konvergieren und folglich stellt sich auch die Frage warum Modelle so langsam trainieren. Durch diese Unwissenheit könnte das Model während des Trainings auch weitere Probleme erfahren. Ein häufiges Problem ist der ***mode collapse***. Grund dafür ist ein unausgeglichenes GAN-Model, bei dem der *Generator* oder *Discriminator* deutlich stärker ist als der andere.

Die Folge eines *mode collapse* ist einerseits die stetige Erzeugung ähnlicher bzw. gleicher Daten (siehe Abb. 3.2). Grund hierfür ist, dass der *Generator* die Schwächen des *Discriminators* erkannt hat und dieser viel schneller lernt.

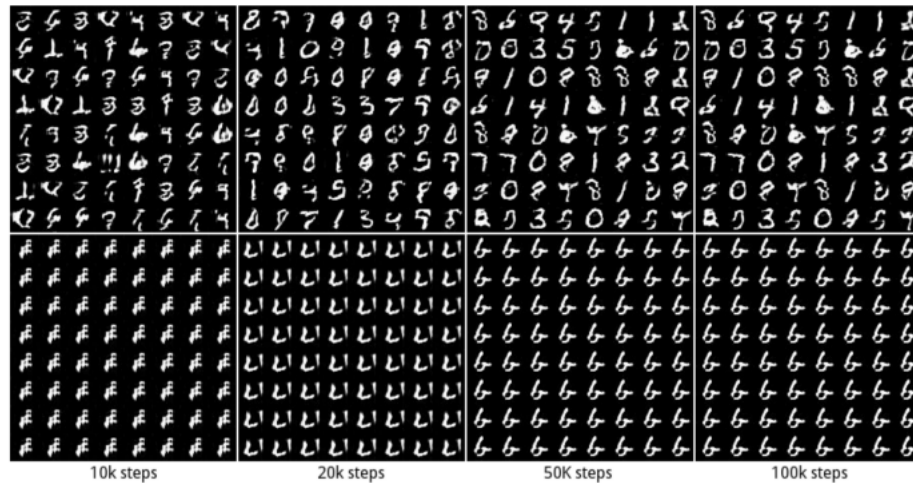


Abbildung 3.2: Mode Collapse. Der Generator erzeugt stets verrauschte Daten.

Quelle: Unrolled Generative Adversarial Networks [23]

Das Gegenteil könnte jedoch auch passieren. Falls der *Discriminator* zu stark ist, ist der Gradient beim Updaten der *weights* zu groß, sodass der *Generator* nur noch unkontrolliert ratet und dadurch sehr undeutliche und meist verrauschte Daten generiert. Ein Beispiel dafür ist in Abbildung 3.3 zu sehen.

Ein weiterer Indiz außer der Überprüfung der generierten Daten mitten im Training ist die Überwachung des *losses*. Ein typisches Merkmal beim Training von GANs ist die starke Oszillation der *losses* des Netze [9]s. Gerade zwischen den Batches kann der *loss* sehr stark schwanken. Der *loss* beider Subnetze sollte aber nichts desto trotz auf lange Sicht stetig zu- oder abnehmen und die starken Schwankungen zwischen den Batches kleiner werden. In Abbildung 3.4 ist der Unterschied zweier Modelle gut zu erkennen.



Abbildungung 3.3: Mode Collapse. Der Generator erzeugt stets ähnliche bzw. die gleichen Daten.

Quelle: 10 Lessons I Learned Training GANs for one Year [25]

3.3 Hyperparameter und ihre Effekte

Wie bei jedem neuronalen Netz gibt es diverse Einstellungsmöglichkeiten für die Hyperparameter. Die Hyperparameter können dabei einen direkten oder aber auch indirekten Effekt auf das Training und das Ergebnis des Models haben. Hier ein Zitat:

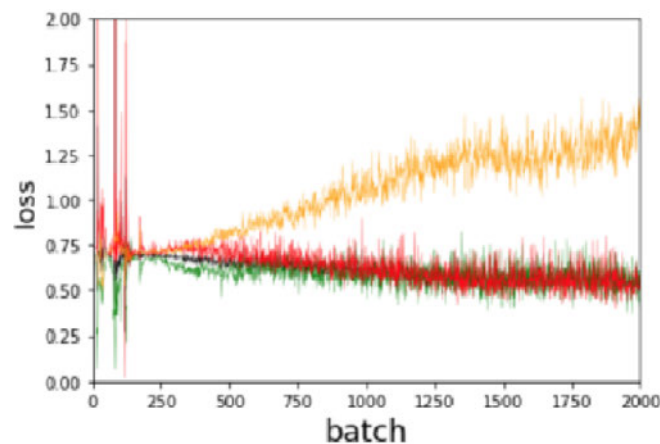
”But with a GAN, every step taken down the hill changes the entire landscape a little. It’s a dynamic system where the optimization process is seeking not a minimum, but an equilibrium between two forces.”, Buch: Deep Learning with Python, S.306 [5]

Typische einstellbare Hyperparameter wären:

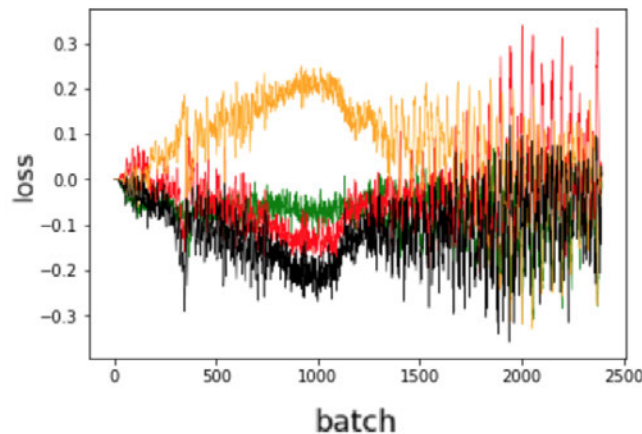
- Batchgröße.
- Lernrate (*learning rate*).
- Aktivierungsfunktion (*activation function*).
- Optimierungsalgorithmus (*optimizer*).
- Loss Funktion (*loss function*).

- Epochen.

Da die Entwicklung von generativen Netzen noch so jung ist, sind die Erforschungen der Effekte dieser Hyperparameter noch im ständigen Wandel. Gerade bei solch komplexen Modellen wie GANs. Unter den GAN-Architekturen gibt es mittlerweile auch so viele Modelle, die je nach Anwendung auf Änderung der Hyperparameter verschieden reagieren. Ein weiteren Einfluss auf die Hyperparameter und deren Effekte ist auch der verwendete Datensatz [27]. Dabei ist es wichtig ein ausgewogenen Datensatz zu verwenden, welche eine hohe Varianz hat an Schlüssel-Merkmalen der Domäne.



(a) Beispiel eines stabilen GANs.

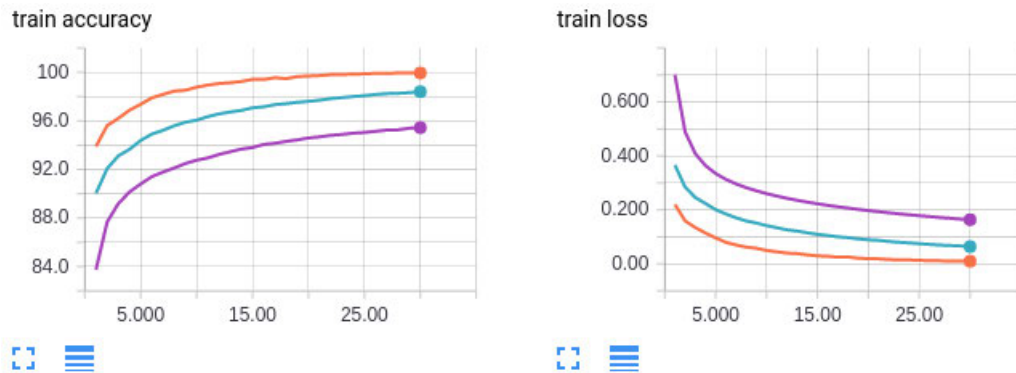


(b) Beispiel eines instabilen GANs. Der loss oszilliert stark.

Abbildung 3.4: Erkennen eines mode collapse Anhand des losses des Models. Grün = D_{real} , Rot = D_{fake} , Schwarz = $\emptyset D_{real+fake}$, Gelb = G

Quelle: Generative Deep Learning[9]

Anders wie in den Anfangszeiten des *deep learnings* hat sich herausgestellt, dass die ausgewählte **Batchgröße** zu Änderungen der Genauigkeit (*accuracy*) und des *losses* führt. Anhand eines einfachen Beispiels mit dem MNIST Datensatz wurde festgestellt, dass die *accuracy* und der *loss* schneller konvergiert sind und auch bessere Ergebnisse erzielt, je kleiner die Batchgröße gewählt wurde (siehe Abb.3.5).



(a) Training accuracy und loss geplottet bei Änderung der Batchgröße.



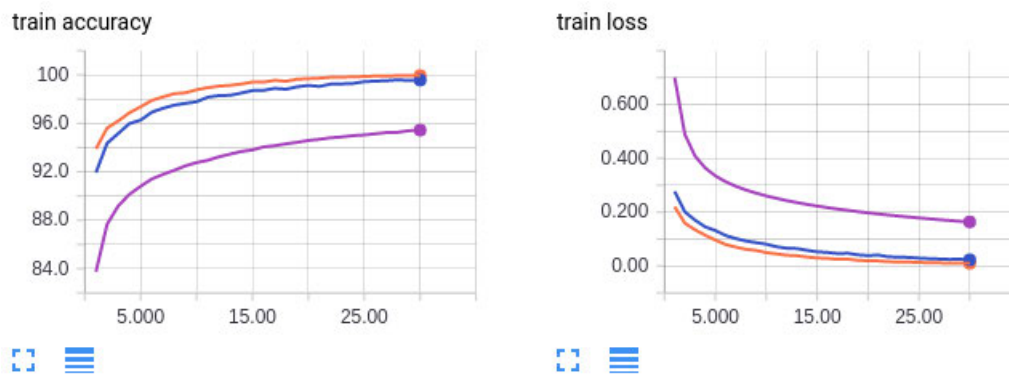
(b) Test accuracy und loss geplottet bei Änderung der Batchgröße.

Abbildung 3.5: x-Achse: Epoche, Orange: Batchgröße 64, Blau: Batchgröße 256, Lila: Batchgröße 1024.

Quelle: Effect on batch size on training [27]

Auch wurde festgestellt, dass durch Erhöhung der *learning rate* den *loss* verbessern kann, bei gleicher Batchgröße. Hierfür wurde im gleichen Model wie im Abschnitt zuvor die *learning rate* variiert und die *accuracy* und *loss* geplottet. 3.6. Man erkennt, dass bei großen Batches der *loss* deutlich geringer ist und schneller konvergiert.

Man erkennt, dass Batchgröße und *learning rate* stark voneinander abhängig sind. Einen wissenschaftlichen Grund oder Beweis ist leider nicht vorhanden, jedoch ist die Hypothese, dass die Daten in den einzelnen Batches von den Gradienten zwischen einander



(a) Training accuracy und loss bei Änderung der learning rate.



(b) Test accuracy und loss bei Änderung der learning rate.

Abbildung 3.6: x-Achse: Epoche, Orange: Batchgröße 64 mit lr 0.01, Blau: Batchgröße 1024 mit lr 0.1, Lila: Batchgröße 1024 mit lr 0.01.

Quelle: Effect on batch size on training [27]

abhängig sind. Die Batches, welche Teile des Datensatzes bündeln, können je nach Zusammensetzung also zu viele Merkmale haben, dass ein effektives Training des Batches erschwert wird.

Für die *activation function* und den *optimizer* sind empirisch die besten Ergebnisse erzielt wurden mit den bereits gängigen Methoden aus von den Deep-CGANs:

- (Leaky) ReLU activation function.
- TanH activation function.
- Adam Optimizer.

Die *ReLU activation function* ist dafür bekannt um das Problem des *vanishing gradients* zu beheben, bei dem die vorderen Layer nicht mehr effektiv trainiert werden. Die *leaky ReLU activation function* erweitert die ReLU Funktion, bei den Werte unter dem Wert 0 mitberücksichtigt werden, in der Hoffnung eventuelle wichtige Informationen nicht komplett zu verlieren. In der Veröffentlichung [26] von Alec Radford, indem ein *Deep Convolutional GAN* verwendet wurde, sind die besten Ergebnisse erzielt worden mit folgender Architektur: Der *Generator* verwendet die *ReLU activation function* mit Ausnahme des Outputlayers, diese verwendet die *tanh activation function*. Die Featuremaps des *Discriminators* werden über die *leaky ReLU* aktiviert. Für das gesamte GAN ist Adam der beste Optimierungsalgorithmus gewesen. Diese Erkenntnis wird bis heute so fortgetragen [4].

3.4 Optimierungen und Stabilisierung von GANs

Die Stabilisierung von GANs ist wie erwähnt immer noch in der aktiven Forschung. Nichts desto trotz haben sich empirisch über die Jahre der Arbeit mit GANs einige Optimierungen herauskristallisiert. Dabei sind die Optimierungen seitens der Architektur und des verwendeten Datensatzes möglich.

3.4.1 Modeldesign

Die *batch normalization* ist wie auch in bei den CNNs eine gute Möglichkeit das Model zu optimieren. Die einzigen Layer bei denen eine *batch normalization* nicht erfolgen sollte ist der Outputlayer des *Generators* sowie der Inputlayer des *Discriminators* [26].

Für die **Vergrößerung der Featuremaps** in der Architektur des *Generators* wurden früher *Upsample*-Layer verwendet. Diese sollte man durch *transpose convolution* Layern ersetzen [8].

Statt der Verwendung von *maxpool* Layern für die **Verkleinerung der Featuremaps** verwendet man bei GANs heutzutage *strided convolution* [26].

Der *Discriminator* sollte eine erhöhte *learning rate* besitzen. Es hat sich herausgestellt, dass es vom Vorteil ist, wenn der *Discriminator* eine doppelt so hohe *learning rate* (two time-scale update rule) besitzt als der *Generator* [13].

Entgegen der im Kapitel 3.2 erwähnten Scheitern der Modelle bei der Konvergierung des *losses* in die Extremen, empfiehlt es sich das Training weiterlaufen zu lassen. Ein plötzliches Konvergieren muss nicht zwangsläufig zur Instabilität des Models führen. Man sollte das Training deshalb nicht zu früh abbrechen [25].

3.4.2 Datensatz

Neben der Größe und der hohen Varianz der Datensatzaufbaus gibt es noch einige andere Möglichkeiten das Model über den Datensatz zu stabilisieren. Auf der Conference on Neural Information Processing Systems (NIPS) wurden 2016 vom Erfinder der GANs, Ian Goodfellow, einige 'tips and tricks' vorgestellt [10]. Dort heißt es: "Die Verwendung von Labels in jeglicher erdenkliche Art und Weise führt nahezu immer zu drastischen Verbesserungen bei der subjektiven Qualität der generierten Dates des Models."

Ein weiterer Ansatz ist das *one-sided label smoothing*. Die Idee dabei ist es bei der Klassifizierung seitens des *Discriminators* für die echten Bilder etwas kritischer zu machen. So soll die Runterskalierung des Output mit einer Zahl <1 multipliziert werden. Das verhindert das extreme Extrapolieren vom Discriminator.

4 Generierung von Daten mit der Computergrafik

Die Stabilität und akkurate Ergebnisse sind im Deep Learning abhängig von dem verwendeten Datensätzen. Die immer voranschreitenden Ideen und Anwendungsmöglichkeiten verlangen aufgrund dessen immer mehr und größere Daten. Neben der rein generativen Methode über neuronale Netze gibt es noch einen weiteren Ansatz über den Bereich der Computergrafik. BlenderProc [7] ist ein open-source Framework welches Blender [6] verwendet. Blender ist ein open-source Projekt für das Bearbeiten und Rendern von computergenerierten Grafiken. BlenderProc nutzt dabei die Technologie des Raytracing [29]. Der Grundgedanke beim Raytracing ist die Berechnung von Lichtstrahlen auf Oberflächen und deren Reflektierung und wird heutzutage unter anderem auch in der Filmindustrie genutzt (Beispiel in Abb.4.1).

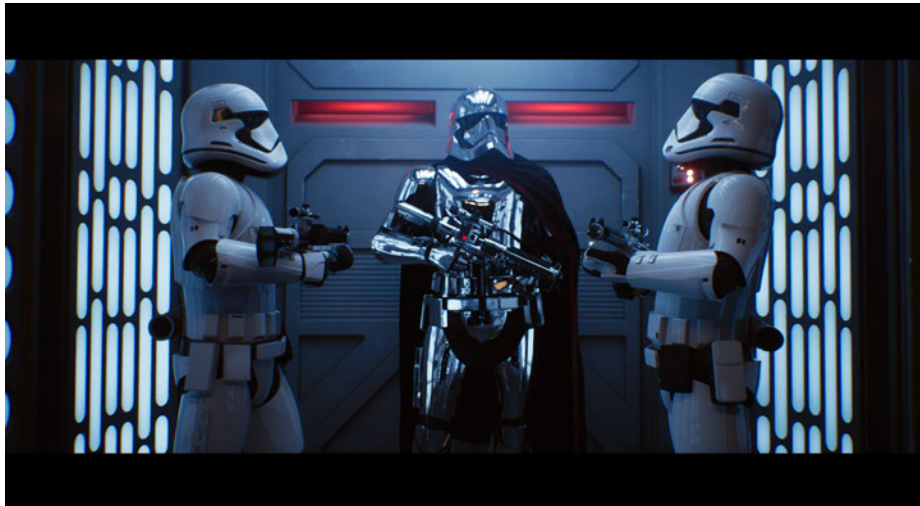


Abbildung 4.1: Star Wars Szene gerendert mit der Raytracing Methode und Unreal Engine.

Quelle: Star Wars Reflections Tech Demo

Diese Technologie ist jedoch auch sehr aufwendig und die Erzeugung für solch hyperrealistische Szenen sehr zeitintensiv und Aufwendig. In der Realität sind Objekte und Szenen nicht perfekt modelliert und haben immer vorgefertigte Assets. Für die Verbesserung der Daten kann der generative Ansatz aushelfen. Computergenerierte Daten haben den Vorteil, dass vollständige Informationen der Szene vorhanden sind. Durch diese Informationen ist das labeln oder die Segmentierung von Objekten sehr einfach und kann beim Prozess der Generierung der synthetischen Daten mitgegeben werden (siehe Abb. 4.2).

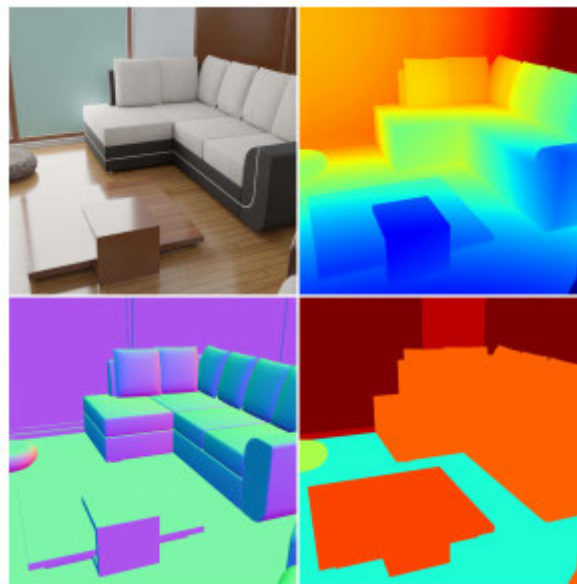
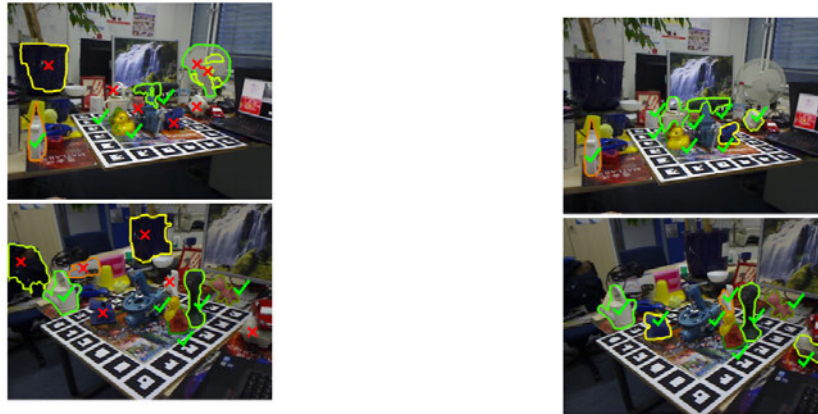


Abbildung 4.2: BlenderProc synthetisierte Labels einer gerenderten Szene.

Quelle: BlenderProc: Reducing the Reality Gap with Photorealistic Rendering[22]

Wie in ersten Abschnitt schon erwähnt wurde, ist die Generierung einer sehr realitätsnahen Szene möglich, jedoch mit einem hohen Zeitaufwand verbunden. Die Erzeugung von "weniger" realistischen Szenen wiederum erscheinen subjektiv zu "glatt und perfekt". Obwohl die Anwendung diese selbst erzeugten Datensätze schon bereits erfolgreiche Ergebnissen im Bereich des ML, in etwa in der Bildsegmentierung fanden (siehe Abb. 4.3 [22]), hat sich jedoch herausgestellt, dass die Kombination mit generativen Netzen und die Augmentierung der Daten bei der Anwendung für weitere Netze das Netz stabiler macht [24].



(a) Bildsegmentierung über eingefügte Objekte mit OpenGL.

(b) Bildsegmentierung über BlenderProc.

Abbildung 4.3: Anwendung einer Bildsegmentierung anhand synthetischen, hyperrealistischen Bildern.

Quelle: BlenderProc: Reducing the Reality Gap with Photorealistic Rendering[22]

4.1 Datensatz Augmentierung

Anhand einer *Object Detection* (Mask R-CNN) wurden verschiedene Datensätze verwendet und evaluiert. Dabei wurde der Datensatz mit echten Bildern mit Hilfe von verschiedenen Methoden erweitert (siehe Abb. 4.4). Drei Methoden werden verglichen:

- Erweiterung des Datensatzes durch fotorealistische Bilder.
- Erweiterung des Datensatzes mit generativen image-to-image Netzen.
- Erweiterung des Datensatzes mit Datenaugmentierung.

Für die Generierung der Daten mit dem generativen Ansatz wurden unter anderem auch die CycleGAN Architektur verwendet, die im nächsten Kapitel näher beschrieben wird. Dabei wurden simulierte Daten über einen EinScan-ProScanner generiert und über den Cycle-GAN neue Daten generiert (Abb. 4.5b).

Die Bildaugmentierung erfolge die Python-Framework imgaug [15]. Mittels des Framework können Bilder generiert werden welche leicht verändert wurden. Die Änderungen können in etwa das Hinzufügen von Rauschen, das Entfernen oder Hinzufügen von Pixeln oder zum Beispiel die Veränderung der Farbwerte sein 4.6.

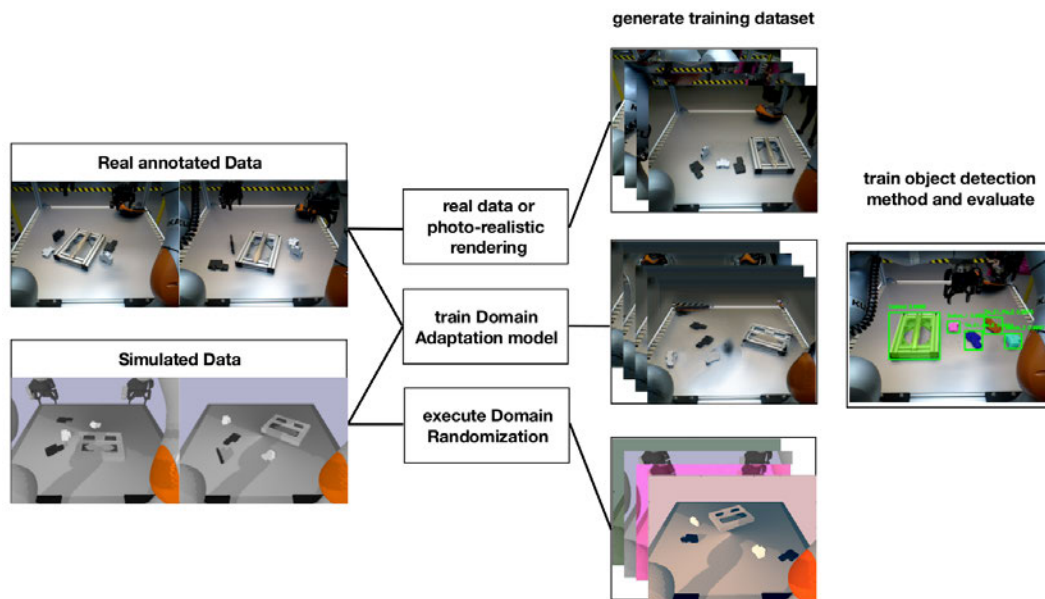
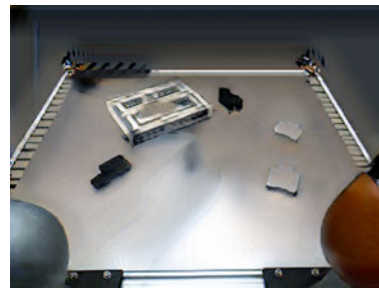


Abbildung 4.4: Pipeline für die Generierung von Datensätzen.

Quelle: Closing the Gap to Real Data [24]



(a) Simuliertes Bild mit Hilfe von EinScan-ProScanner. (b) Generiertes Bild über Cycle-GAN mit Hilfe von a.

Abbildung 4.5: Generierung von realistischen Daten mit Hilfe von simulierten Daten und Cycle-GAN.

Quelle: Closing the Gap to Real Data [24]

Mit den generierten Datensätzen wurden die Modelle trainiert und mit echten Evaluationsbildern ausgewertet. In diesem Experiment wurden dabei folgende Ergebnisse erzielt:

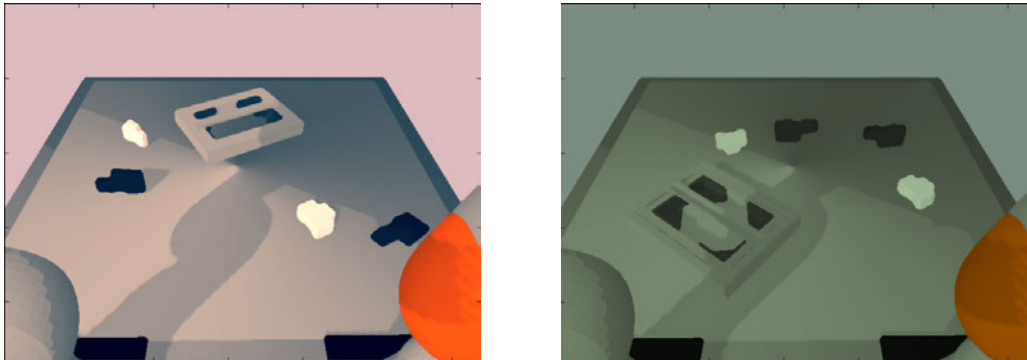


Abbildung 4.6: Datenaugmentierung durch Änderung der Farbwerte.

Quelle: Closing the Gap to Real Data [24]

Tabelle 4.1: Ergebnisse der Evaluierung mit echten und synthetisch erzeugten Daten.

Methode	Datensatzgröße	precision	recall
Echter Datensatz	105	113 / 115	113 / 124
Augmentierung mit imgaug	105 + 200	118 / 118	118 / 124
Cycle-GAN	105 + 200	112 / 112	112 / 124

Man erkennt, dass die *precision* in beiden Fällen besser wurde. Der *recall* ist dabei leicht gesunken. Schlussfolgernd kann man also sagen, dass die Erweiterung des Datensatzes zu Stabilisierung des Models führt.

5 Architekturen

In diesem Kapitel geht es um zwei State-of-the-Art GAN-Architekturen und den generellen Versuch realistische Daten zu generieren. Bei beiden Modellen wird der gleiche Datensatz verwendet um einen direkten Vergleich beider Ergebnisse zu machen. Als Programmiersprache wurde Python [28] verwendet. Die verwendete Python-Version ist 3.8.0. Die Implementierung beider Modelle wurde mit Hilfe der Tensorflow-Framework [1] in der Version 2.2.0 realisiert. Das Training der Modelle erfolgte auf dem lokalen Rechner des Verfassers dieser Bachelor-Arbeit. Als Betriebssystem wurde Ubuntu 20.04 LTS verwendet. Folgende Hardware wurde für das Training der Modelle verwendet:

- Prozessor: IntelCore i5 4690k@4x4,3GHz
- RAM: 32 GB DDR3
- GPUs: NVIDIA GeForce GTX 980Ti + NVIDIA GeForce GTX 970

5.1 Datensatz

Der hier verwendete Datensatz ist der monet2photo Datensatz. Claude Monet war ein französischer Maler, der über seine Jahre als Künstler auf Landschaftsbilder in Frankreich fokussierte. Dabei malte er einige Motive öfters in verschiedensten Tagessituationen und Seasons 5.1. Über die Zeit hat Monet mehrere tausend Bilder gemalt. Sein Zeichenstil hat sich dabei über die Jahre nur leicht verändert, welches eine gute Grundlage ist für die Erstellung eines Datensatzes. Dadurch, dass der Stil sich nur leicht verändert hat, lassen sich die gemalten Bilder als Teil einer Domäne ansehen, welche eine image-to-image Übersetzung mithilfe von generativen Netzen ermöglicht. Der Datensatz besteht dabei Bilder im JPEG-Format in der Größe 256×256 .

Zu den Bildern von Monet wurden weitere echte Fotos als Datensatz hinzugefügt 5.2. Die Bilder sind echte Fotos von verschiedenen Szenen. Der Großteil der Fotos besteht dabei



Abbildung 5.1: Beispiele aus der Monet-Domäne.

Quelle: monet2photo dataset, Berkeley AI Research laboratory

aus Landschaftsbildern, jedoch gibt es auch einige Bilder von Tieren oder modernen Städten, Autos und Menschen. Diese Bilder dienen als Datensatz für die zweite Domäne, der Realität.

Die untere Tabelle 5.1 zeigt die Aufteilung der Daten im Datensatz:

Tabelle 5.1: Aufbau des monet2photo Datensatzes.

Daten	Anzahl der Bilder
Trainingsdaten Monet	1072
Trainingsdaten echte Welt	6289
Evaluierungsdaten Monet	121
Evaluierungsdaten echte Welt	751

Man erkennt, dass die Anzahl an Monet-Daten deutlich geringer sind als die der echten Welt. Der Datensatz ist open-source und wurde zugänglich gemacht von der Berkeley AI Research laboratory (BAIR) der Universität Berkeley.

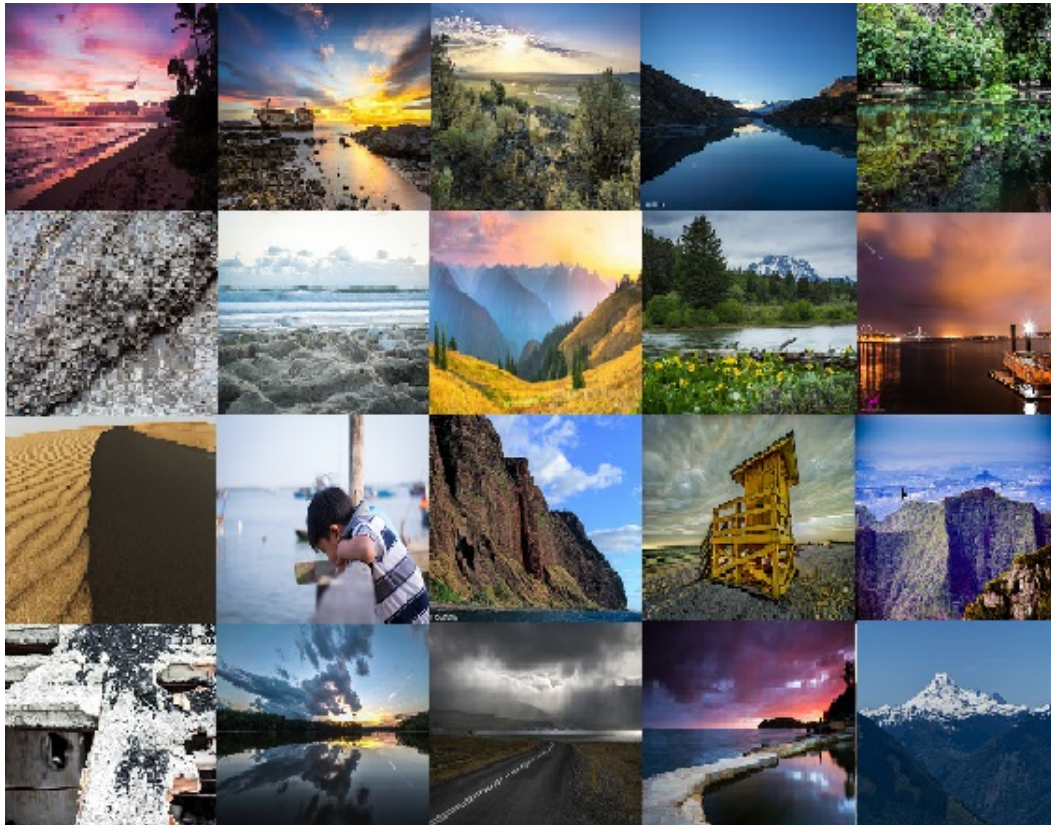


Abbildung 5.2: Beispiele aus der Echten Welt-Domäne.

Quelle: monet2photo dataset, Berkeley AI Research laboratory

5.2 Cycle GAN

Präsentiert wurde das CycleGAN Paper [30] im Jahr 2017 von Mitarbeitern des BAIR. Das Problem der image-to-image Translation bestand bis dato daran, dass für den Style-transfer in eine andere Domäne gepaarte Bilder verwendet werden mussten um ein akkurates Ergebnis zu erhalten. Die Architektur des CycleGANs erlaubt jedoch das Training von ungepaarten Bildern und dadurch eine flexiblere Anwendung bei der image-to-image Übersetzung.

5.2.1 Architektur

Wie der Name CycleGAN bereit impliziert, werden in dieser Architektur gleich zwei GANs verwendet, womit effektiv zwei *Generatoren* und zwei *Discriminatoren* trainiert

werden müssen. Die Idee hierbei ist, dass die beiden GANs sich gegenseitig lehren wie man von der einen in die andere Domäne kommt. Dabei können die einzelnen GANs als eine Art Experte angesehen werden, die dem anderen GAN erklärt, was die Eigenschaften der eigenen Domäne auszusehen haben. Die Illustration 5.3 zeigt den Grundgedanken der Architektur.

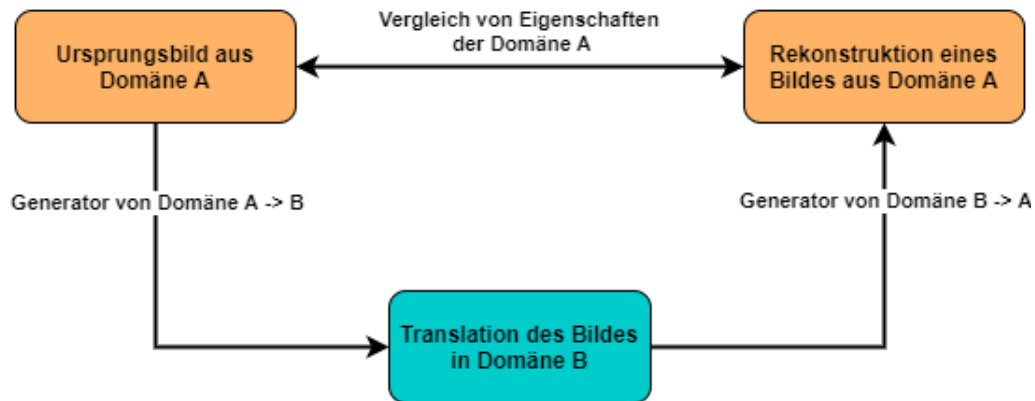


Abbildung 5.3: Cycle-GAN Idee.

Quelle: Huy-Bao Le

Die Idee ist es ein Ursprungsbild aus einer Domäne A zu wählen, dieses über den Generator $G_{A \rightarrow B}$ in die Domäne B zu transferieren und das Ergebnis über den Generator $G_{B \rightarrow A}$ zurück in die Ursprungsdomäne zu transferieren. Das rekonstruierte Bild sollte demnach wieder das Ursprungsbild sein. Die Differenz des rekonstruierten Bild und dem Ursprungsbild wird cycle-consistency loss genannt (siehe Abb. 5.8).

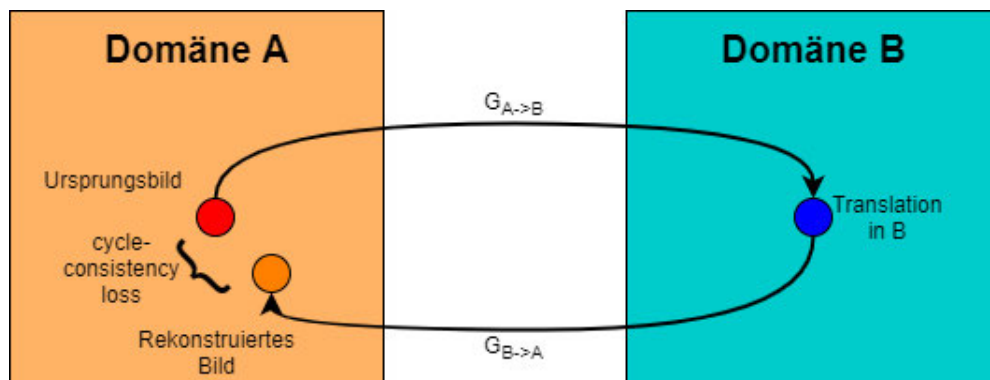


Abbildung 5.4: Cycle-consistency loss.

Quelle: Huy-Bao Le

Über diese Konsistent errechnet sich die Differenz die das Model minimieren soll. Dadurch erhält der Generator aus der Domäne A nicht nur das einzelne Ergebnis seines Discriminators sondern auch die Information darüber von anderen Generator ob die Rücktranslation erfolgreich war. Diese Erkenntnis erzeugt eine höhere Schicht für das CycleGAN, welches mit der Kombination der Feedbacks das Model mit beiden GANs in beide Richtungen optimieren kann.

Die Architektur der Discriminators ist in Abbildung 5.5 zu sehen. Sie besteht aus einfachen Faltungsblöcken mit InstanceNormalization- und LeakyReLU Layern. Anders als bei Classification-Discriminators, bei dem Werte zwischen 0 (fake) und 1 (real) ausgegeben werden, wird die Architektur des PatchGAN-Discriminators [14] verwendet. Der Discriminator teilt das zu prüfende Bild dazu in kleinere Vierecke auf und überprüft dabei für jedes Viereck ob dieser Teil echt oder falsch ist. Der Output des Discriminators ist in dieser Architektur ein 32×32 Tensor welcher in jeder Zelle eine Bewertung der jeweiligen aufgeteilten Vierecke beinhaltet. Der Vorteil dieser Bewertung ist, dass die natürliche gegebene Architektur von CNNs bereits die Bewertung während des Faltungsprozesses durchführt. Die Idee ist es dadurch den Stil des gesamten Bildes anhand Unterteilungen zu bewerten statt des kompletten Bildes.

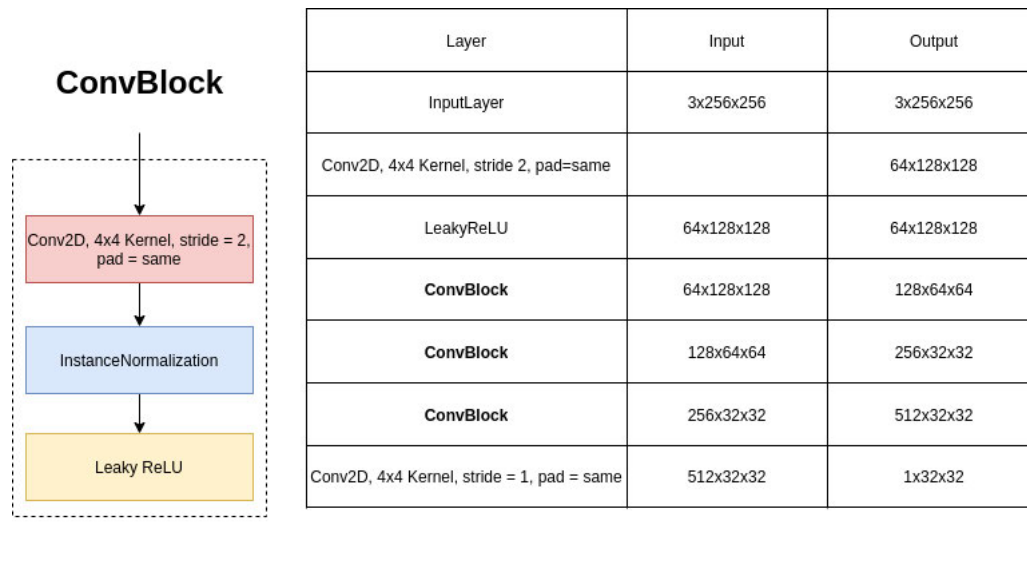


Abbildung 5.5: Cycle-GAN Discriminator Architektur.

Quelle: Huy-Bao Le

Für die Architektur der Generatoren wurde die Residual Neural Network (ResNet)-Architektur [12] gewählt. Die Architektur hat in der Vergangenheit viele Erfolge erzielt in

der Implementierung für Deep-CNNs. Da die *Generators* eine Vielzahl an Eigenschaften erkennen müssen wurde das Netz tief gebaut in der Hoffnung dadurch so viele Merkmale zu erkennen und zu filtern. Die *Generator*-Architektur ist in Abbildung 5.6 zu sehen. Sie besteht im Kern aus der *sand clock* Methode. Die Abbildung in den *latent space* ist der Output vor dem Upsample-Layern und besteht aus einem 3-D Raum begrenzt durch $128 \times 64 \times 64$ Floating-Point Werten. Das implementierte Model besitzt dabei insgesamt 4.047.139 Parameter pro GAN, also insgesamt 8.094.278 Parameter.

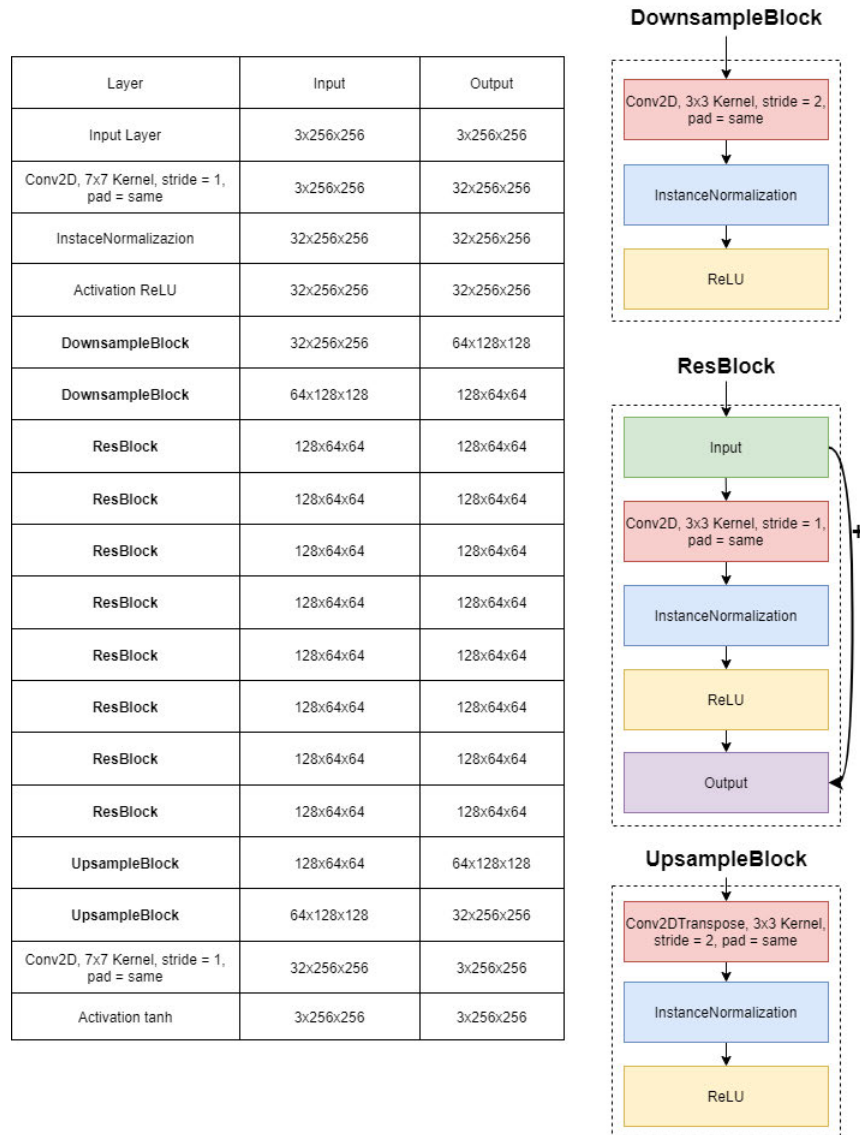


Abbildung 5.6: Cycle-GAN Generator Architektur.

Quelle: Huy-Bao Le

Eine weitere Neuheit in der CycleGAN Architektur ist das *Identity-Mapping*. Sie dient als Unterstützung der Generatoren für die Übersetzung in die andere Domäne. Dabei wird ein Bild aus der Domäne A dem Generator für die Übersetzung in die eigene Domäne mitgegeben, das Ergebnis sollte dann immer noch ein Bild aus der Domäne A sein. Mit Hilfe des Discriminators lässt sich dadurch nochmal ein weiteres Feedback des Ergebnis liefern. Aus der Erfahrung mit CycleGANs erhält die Identity sich dadurch die Bildtemperatur bei der Übersetzung 5.7.

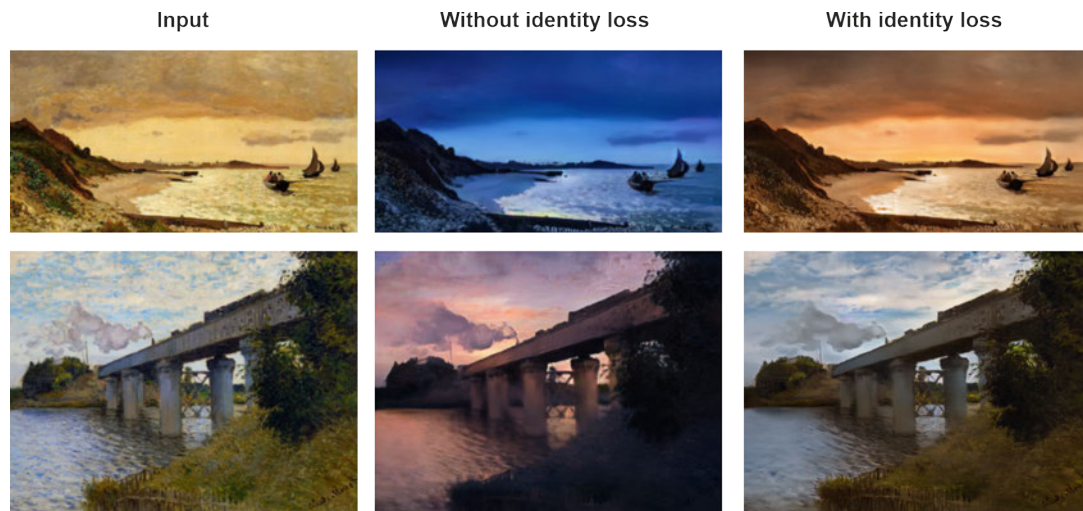


Abbildung 5.7: Erhalt der Bildtemperatur durch die Einführung des Identity loss.

Quelle: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks [30]

Das CycleGAN Model verwendet Batches aus den jeweiligen Domänen und nutzt diese als Input. Die Discriminator werden für die Zeit eingefroren. Für die Berechnung des *losses* werden die Summe von folgenden Kriterien gezogen:

- Bewertung der generierten Bilder der Generatoren (fake A, fake B).
- Bewertung rekonstruierten Bildern (rek. A, rek. B).
- Bewertung des Identity mappings (ID A, ID B).

Die Illustration 5.8 verdeutlicht nochmal die Berechnung der *losses*.

Für die *validation loss* wurde die Mean Square Error (MSE) verwendet, für die *reconstruction loss* und *identity loss* wurde die Mean Absolute Error (MAE) genommen. Der

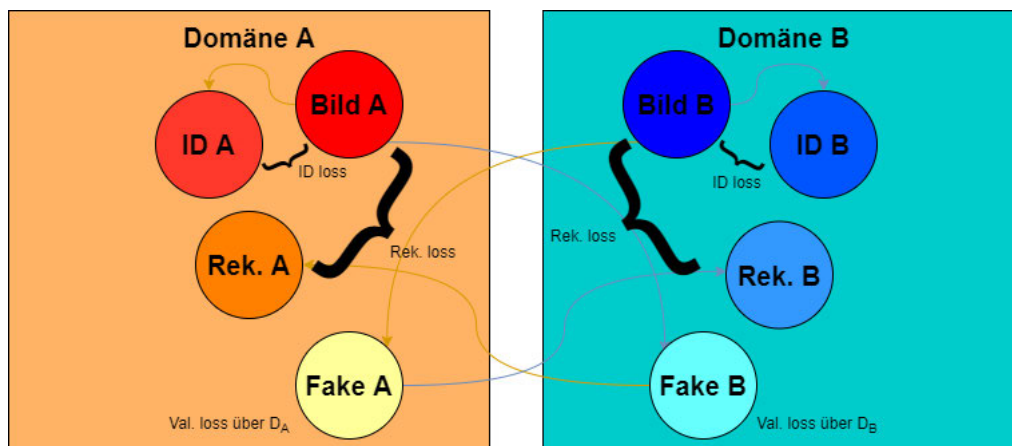


Abbildung 5.8: Illustration der losses im Cycle-GAN.

Quelle: Huy-Bao Le

Grundgedanke ist, dass der Discriminator der eigenen Domäne kritischer auf Fehler achten soll, deswegen die Verwendung des MSE loss-Funktion. Insgesamt berechnen wir also sechs verschiedene losses, welche alle mit in das Training hereinfließen.

5.2.2 Training

Das Training dieser Architektur erfolgt genauso wie jedes andere GAN auch. Die Idee des alternierenden Trainings ist die gleiche, jedoch werden bei jeder Runde beide Discriminators und Generators gleichzeitig trainiert. Das allgemeine Training beginnt mit der Erzeugung über der Generatoren und wird in Batches gespeichert. Das Training der Discriminators erfolgt dann auf den erzeugten Bildern und den echten Bildern der jeweiligen Domäne. Die Generators werden im Anschluss mit Optimierung für die kombinierten Loss Funktionen trainiert. Eine kleine Übersicht des Trainingsverlauf ist in Abbildung 5.9 zu sehen.

Für das Training wurden folgende Hyperparameter festgelegt:

- Batchgröße = 1.
- learning rate = 0,0002.
- Optimizer = Adam.
- 200000 Steps.



Abbildung 5.9: Zwischenergebnisse während des Cycle-GAN Trainings.

Quelle: Huy-Bao Le

Typisch für ein CycleGAN ist die Verwendung der Batchgröße 1, da eine Generalisierung der Trainingsdaten verhindert werden soll. Das Training hat acht Tage gebraucht. Während des Trainings wurden alle zehn Batches jeweils das aktuelle originale Bild aus der Domäne, die Übersetzung, das Rekonstruierte Bild und die Identity des Bildes ausgegeben um eine händische Evaluation des Trainings durchzuführen.

5.2.3 Evaluation

Die Evaluation des Models erfolgt durch die visuelle Bewertung der Ergebnisse. Betrachtet werden die Details der Übersetzungen, zu der subjektiven Bewertung der Bilder wird außerdem über den geplotteten loss überprüft, ob beim Training das Model gescheitert ist.

Die Illustration 5.8 verdeutlicht dabei die Evaluationsmetrik der einzelnen Subnetze des CycleGANs. Der Generator $G_{B \rightarrow A}$ beschreibt alle generierten Bilder die in der Domäne A sind, also die ID A, das rekonstruierte A und die Übersetzung des Ursprungsbildes der anderen Domäne. Der Discriminator ist bei jedem generierten Bild aus der Domäne

betroffen. Das gleiche gilt dementsprechend für die gegenüberliegende Domäne. Die rekonstruierten Bilder ist die Evaluation des gesamten Modells, da hier alle Generators und Discriminators mit einbezogen werden. Ist die Rekonstruktion also exakt das gleiche Bild wie das Original wäre das Modell perfekt und somit die Translation in beide Richtungen gleich gut.

Als Erinnerung: Die Übersetzung erfolgt in beide Richtungen zwischen der Domäne von "Monet" und der Domäne "Realität". Alle visuellen Bewertungen sind dabei subjektiv und erfolgt über die Analyse der ausgegebenen Bilder. Alle Daten sind aufgrund der Größe im Anhang anzufinden. Bei der Evaluierung der Daten ist folgendes aufgefallen:

- Positiv aufgefallen bei der Generierung von Bildern in die Realität [A.1]:
 - Bilder sind kontrastreicher (Abb. A.3a).
 - Tiefeneffekt wurde erkannt (Abb. A.3a).
 - Reflektionen sind akkurat (Abb. A.1c).
 - Abgrenzungen zwischen Objekten sind klarer definiert (Abb. A.1c).
 - Farben sind intensiver (Abb. A.3a).
 - Rekonstruierte Bilder sehen ähnlich aus (Abb. A.3b).
 - Die Komposition des Bildes wurde nicht verändert (Abb. A.1).
- Negativ aufgefallen bei der Generierung von Bildern in die Realität [A.2]:
 - Die Übersetzung von sehr abstrakten Bildern teilweise schwer gefallen (Abb. A.4a).
 - Rekonstruierte Bilder wirken im Vergleich zum Original übersättigt und kontrastreicher (Abb. A.2c).
 - Teilweise entstehen leichte Farbschwankungen (Abb. A.4b).
- Positiv aufgefallen bei Generierung von Bildern in die Monet-Domäne [A.3]:
 - Die Farbtöne des Bildes sind pastellfarbener geworden (Abb. A.3b).
 - Klare Grenzen wurden verwischt und sind nicht mehr so scharf (Abb. A.3a).
- Negativ aufgefallen bei der Generierung von Bildern in die Monet-Domäne [A.4]:

- Seltsame Artefakte an den Rändern (Abb. A.4b).
- Oft sieht die Übersetzung so aus als wäre nur Rauschen hinzugefügt (Abb. A.4a).
- verwischte linien sehen nicht so aus, als wären sie mit einem pinsel gemalt worden (Abb. A.4b).

Es scheint so, als wäre das GAN für Generierung der Daten in der Monet-Domäne schwächer als die in die Realität. Für diese Erkenntnis sprechen die positiven Eigenschaften der Generierung in die Realität. Die übersetzten Bilder sehen realitätsnah aus. Merkmale sind unter anderem die klare Trennung der Objekte und die kontrastreichen und farben starken Bilder. Als Grund für dieses Ungleichgewicht werden zwei Gründe vermutet: zum einen ist der Datensatz sehr unausgeglichen. Der Teil des Datensatzes für die realen Bilder ist um den Faktor 6 größer als die Monet-Domäne 5.1. Das liegt unter anderem daran, dass die Daten für echte Bilder zahlreich vorhanden sind und die Bilder von Monet limitiert. Ein weiterer Grund könnte sein, dass der Zeichenstil von Monet nicht eindeutig genug ist. Im Laufe seiner Karriere hat Monet sich auf zwar auf die Landschaftsmalungen konzentriert, jedoch hat sich sein Malstil über die Jahre verändert. Das erschwert die Filterung der Merkmale für die Domäne.

Neben der visuellen Evaluation sind in Abbildung 5.10 und 5.11 die Losses geplottet während des Trainingsverlauf. Zu sehen sind die geplotteten Kurven der Discriminatoren, des cycle-consistency loss und des identity loss des gesamten Cycle-GANs. Zu sehen ist ein sehr typischer Verlauf eines guten Trainings bis dahin. Obwohl die Ergebnisse schon vielversprechend sind, ist eine Verbesserung durch längeres Training denkbar. Im Vergleich zu weiteren Modellen ist ein stabiles Training bereits ein gutes Indiz für ein erfolgreiches Training. Die Trainingszeiten lassen sich durch verbesserter Hardware deutlich beschleunigen.

Ein großer Vorteil dieser Architektur ist die Konsistenz der Komposition bei der Übersetzung. Die zuvor gelabelten Bilder können bei der Übersetzung erhalten bleiben, sodass ein erneutes Labeln nicht notwendig ist. In Kombination mit einer Computergrafik generierten Datensätzen, bei denen vollständige Informationen der Szene vorliegen, wäre dieser Ansatz somit ein durchaus erfolgreicher Ansatz für die Problemstellung der Arbeit.

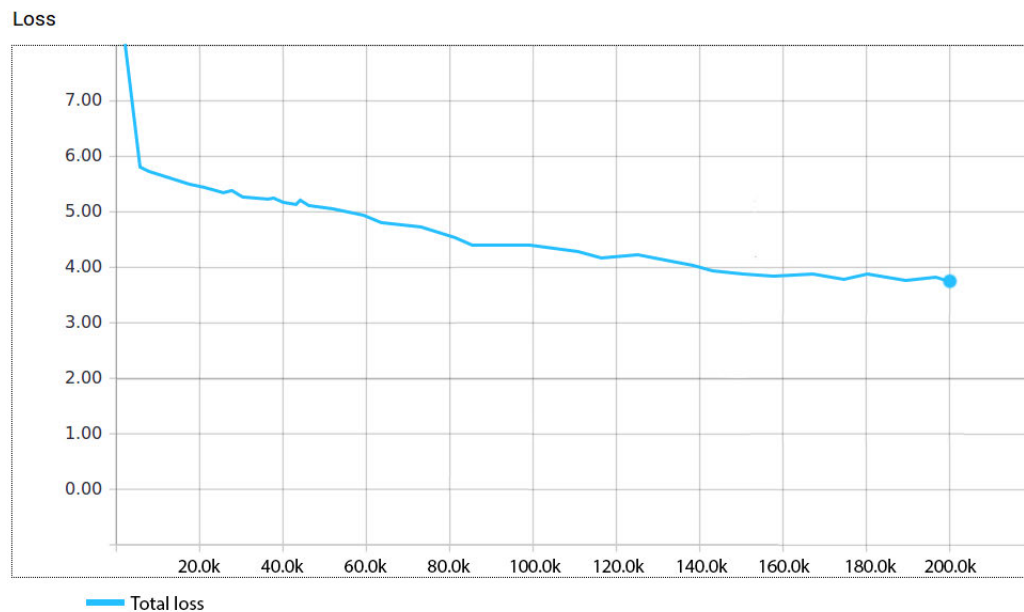


Abbildung 5.10: Total loss während des Trainings.

Quelle: Huy-Bao Le

5.3 Style GAN

Von NVIDIA zum ersten mal im Dezember 2018 vorgestellte StyleGAN [18] Architektur befasst sich mit einer bis dato neuen Methode Bilder zu generieren. Ungleich bei bis dahin konventionellen GAN Architekturen, bei denen der Fokus auf die Verbesserung des Discriminators wert gelegt wurde, liegt hier der Fokus auf die Weiterentwicklung des Generators. Anders als bei den konventionellen Generatoren, bei denen durch randomisierte Verfahren versucht wird ein für den Discriminator befriedigenden Output zu erschaffen, versucht die StyleGAN Architektur einen direkten Einfluss bei der Generierung der Daten zu haben. Der Generator nimmt keinen Vektor aus dem latent space als Input, stattdessen wird mit Hilfe von Style Vektoren aus einem Mapping-Netzwerk sowie durch zufällige Noise Layer versucht stochastische Varianzen bei der Erzeugung zu ermöglichen.

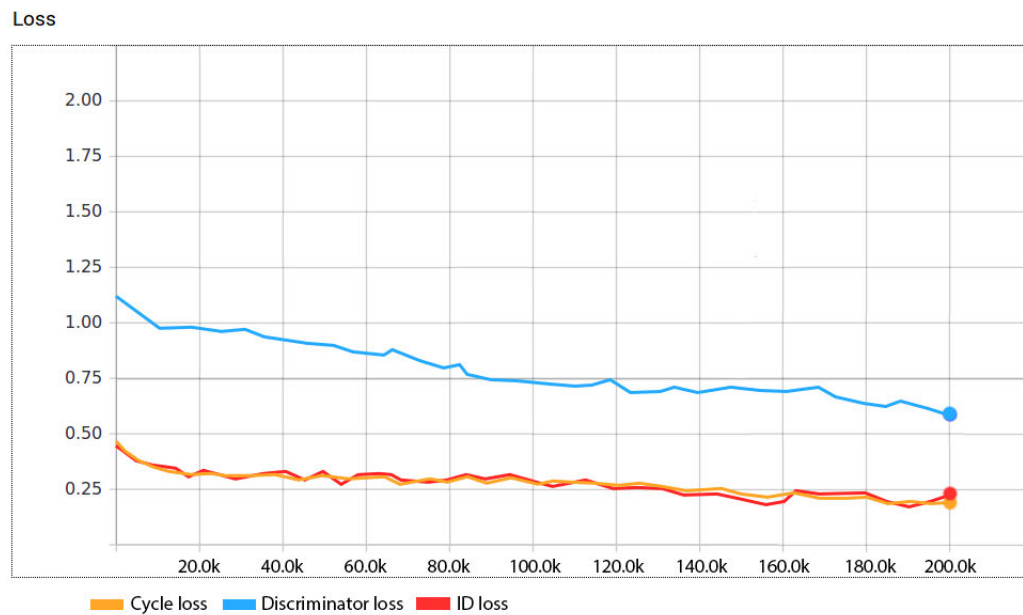


Abbildung 5.11: Cycle loss in Orange, Summe der Discriminator loss in Blau und ID loss in Rot.

Quelle: Huy-Bao Le

5.3.1 Architektur

Basierend auf der Progressive Growing GAN Architektur [16], bei denen der Generator in progressiven Schritten von kleinen- auf immer größer werdenden Tensoren trainiert wird bis das Model einen gewissen Wert erreicht hat, bringt Style GAN einige Änderungen mit sich. Eine Änderung wäre der Wechsel der Interpolation zwischen den Up- und Down-sampling Layern. Statt des nearest-neighbor Verfahrens wird hier bilinear interpoliert.

Der Style Vektor w wird durch ein Style Mapping Netzwerk über einen zufälligen Input im latent space generiert. Dieser wird genutzt um Parameter zu generieren, welche genutzt werden um eine modifizierte Instance Normalization, namens Adaptive Instance Normalization, zu realisieren. Das Mapping-Netzwerk besteht aus Fully connected Layern, welches über einen zufälligen Punkt im latent space als Input einen Style-Vektor erzeugt. Dieser Style-Vektor wird verwendet um dem eigentlichen Generator Informationen über den jeweiligen Style in das zu generierende Bild zu übermitteln. Dafür wird der Vektor über ein eigenes Netzwerk übersetzt in zwei Parametern, y_s und y_b , welche über

die Adaptive Instance Normalization nach jedem Convolution Layer Einfluss auf das Bild hat 5.12.

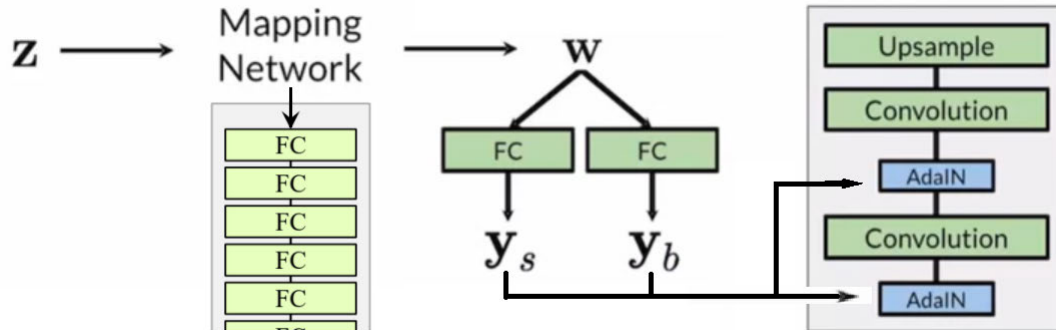


Abbildung 5.12: Übersetzung des Style Vektor w und die Verwendung der Parameter y_s und y_b im Generator des Style-GANs.

Quelle: Huy-Bao Le

Die beiden Parameter sind jeweils der Scale y_s und der Bias y_b . Beide Parameter werden verwendet um die modifizierte Instance Normalization anzupassen 5.13. Ziel ist es, dass der Style-Vektor mit seinen Informationen in den vorderen Convolution Layern die gröbere Komposition des Bildes beeinflusst, während in den hinteren Layern der Style-Vektor die feinen Details verändert.

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Abbildung 5.13: Adaptive Instance Normalization. Scale y_s und Bias y_b und die modifizierte Instance Normalization.

Quelle: A Style-Based Generator Architecture for Generative Adversarial Networks [18]

Alternativ zu den traditionellen Generatoren wird kein Punkt aus dem latent space als Input verwendet, stattdessen erhält das Model einen Input als Tensor mit konstanten Werten. Zudem werden vor jeder Adaptive Instance Normalization ein Noise Layer mit hinzugefügt. Dieser hat den Effekt, dass der Style eine gewisse Varianz erhält und dadurch die Generierung randomisiert. Das Noise Layer wird vor jeder Adaptive Instance Normalization neu generiert, sodass der Style auf den verschiedenen Ebenen mehr Varianz erhält. Die Abbildungen 5.14 und 5.15 zeigen dabei den Effekt von groben- und feineren Noise Layern auf den Output.

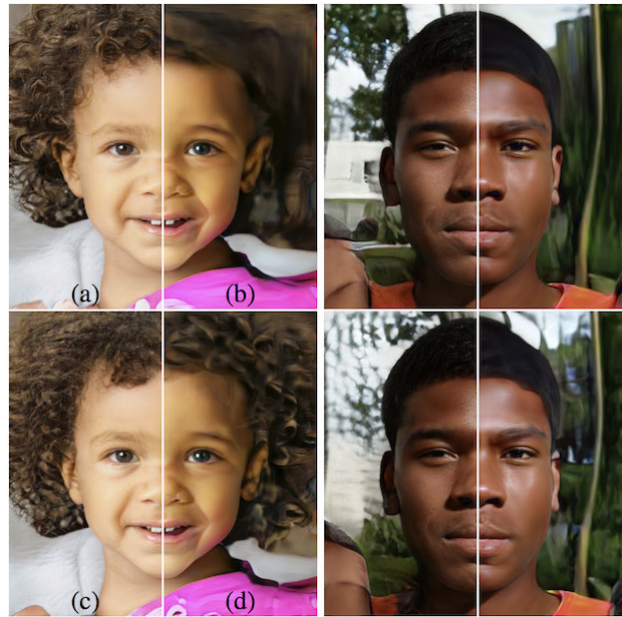


Abbildung 5.14: Beispiel für den Effekt von groben- und feineren Noise Layern. Größere Noise Layer führen zu größere Locken und größeren Hintergrund Features (a)/(b), feine Noise Layer zu feineren Locken und feiner Hintergrund Features (c)/(d).

Quelle: A Style-Based Generator Architecture for Generative Adversarial Networks [18]

Um für noch mehr Varianz bei der Generierung zu sorgen gibt es die Mixing regularization. Dabei werden randomisiert Bilder mit zwei verschiedenen Style-Vektoren generiert. Dabei wird der erste Style-Vektor für die vorderen Ebenen verwendet und für die hinteren der andere.

Für die Implementierung wurde eine kleine Version der StyleGAN Architektur gewählt. Statt der eigentlichen Bildgenerierung von 1024x1024 wurde eine Outputgröße von 256x256 gewählt um Ressourcen und Trainingszeiten deutlich zu verkürzen. Die zu erwartenden Trainingszeiten mit Tesla V100 GPUs sind aus der Tabelle 5.2 [19] zu entnehmen.

Das Style Mapping Netzwerk S beginnt mit einem randomisierten Input aus dem Latent space $z \in Z$ und hat als Output einen gemappten Style Vektor $w \in W$. Über einen Dense Layer werden über eine Affine Transformation der Style Vektor in die Parameter y_s und y_b gefiltert, welche für die Adaptive Instance Normalization in einem Style Block verwendet wird. Nach jedem Convolution Layer des Style Blocks wird ein zufälliges Noise Pattern hinzugefügt bevor die eigentliche Adaptive Instance Normalization ausge-



Abbildung 5.15: Weiteres Beispiel für den Effekt von groben- und feineren Noise Layern. Dabei wurde der Style und Noise Layer der vorderen Layern übernommen. High-level Details wie Pose, Gesichtsförm und Brille wurden von Source B übernommen.

Quelle: A Style-Based Generator Architecture for Generative Adversarial Networks [18]

Tabelle 5.2: Erwartete Trainingszeiten für Style-GAN für verschiedene Outputgrößen.

# GPUs	1024×1024	512×512	256×256
1	41 days 4 hours	24 days 21 hours	14 days 22 hours
2	21 days 22 hours	13 days 7 hours	9 days 5 hours
3	11 days 8 hours	7 days 0 hours	4 days 21 hours
4	6 days 14 hours	4 days 10 hours	3 days 8 hours

führt wird. Der Generator ist ein Netzwerk welches einen konstanten 512x4x4 Tensor als Input hat, welcher über sequenzielle Konkatination von Style Blöcken vergrößert wird bis zu einem 3x256x256 RGB Bild als Output. Die Architektur des Style Mapping- und Generator Netzwerk ist in Abb. 5.16 zu sehen.

Die Discriminator Architektur gleicht einem traditionellen DCNN mit Convolution Layern und LeakyReLU als Aktivierungsfunktion.

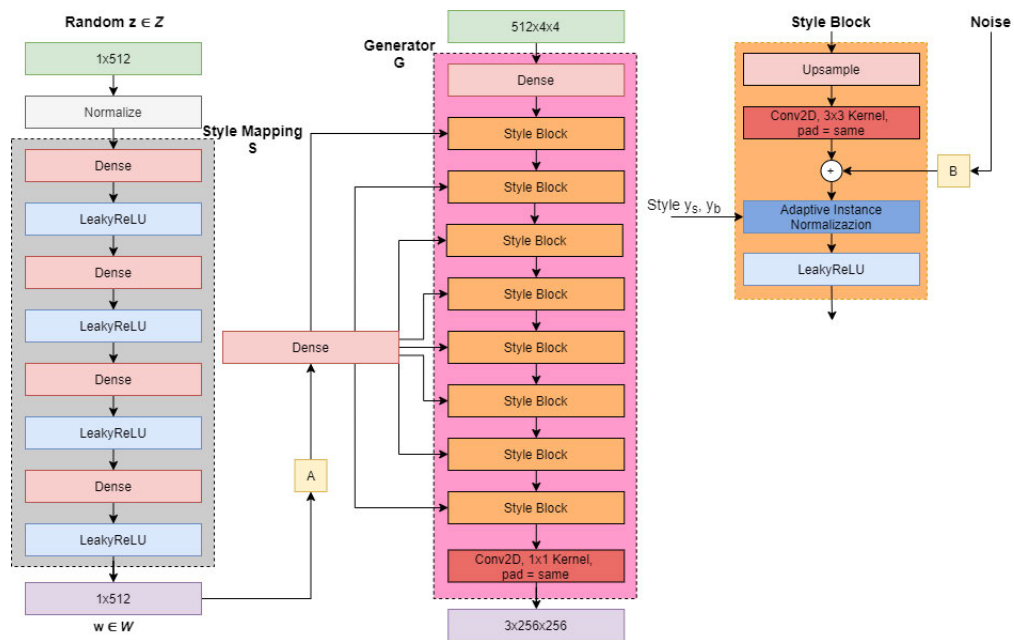


Abbildung 5.16: Illustration der implementierten reduzierten StyleGAN Architektur.

Quelle: Huy-Bao Le

5.3.2 Training

Als Trainingsdaten für dieses GAN wurde lediglich der fotorealistische Datensatz gewählt. Die Monet Bilddaten hätten für ein proof-of-concept sicherlich gereicht, jedoch ist dieser im Vergleich zum fotorealistischen Datensatz deutlich limitierter. Das Training wurde insgesamt für 10 durchgehende Tage ausgeführt. Dabei wurden alle 64 Steps Samples generiert um die Entwicklung des Trainings regelmäßig zu evaluieren. Für das Training wurden folgende Hyperparameter festgelegt:

- Batchgröße = 12.
- learning rate = 0,0001.
- Optimizer = Adam.
- 500000 Epochen.
- Loss function = Mean squared error.

Um eine Generalisierung der Trainingsdaten zu vermeiden wurde dabei eine kleine Batchgröße ausgewählt. Eine Kompilation der Zwischenergebnisse beim Training ist in Abbildung 5.17 zu sehen.

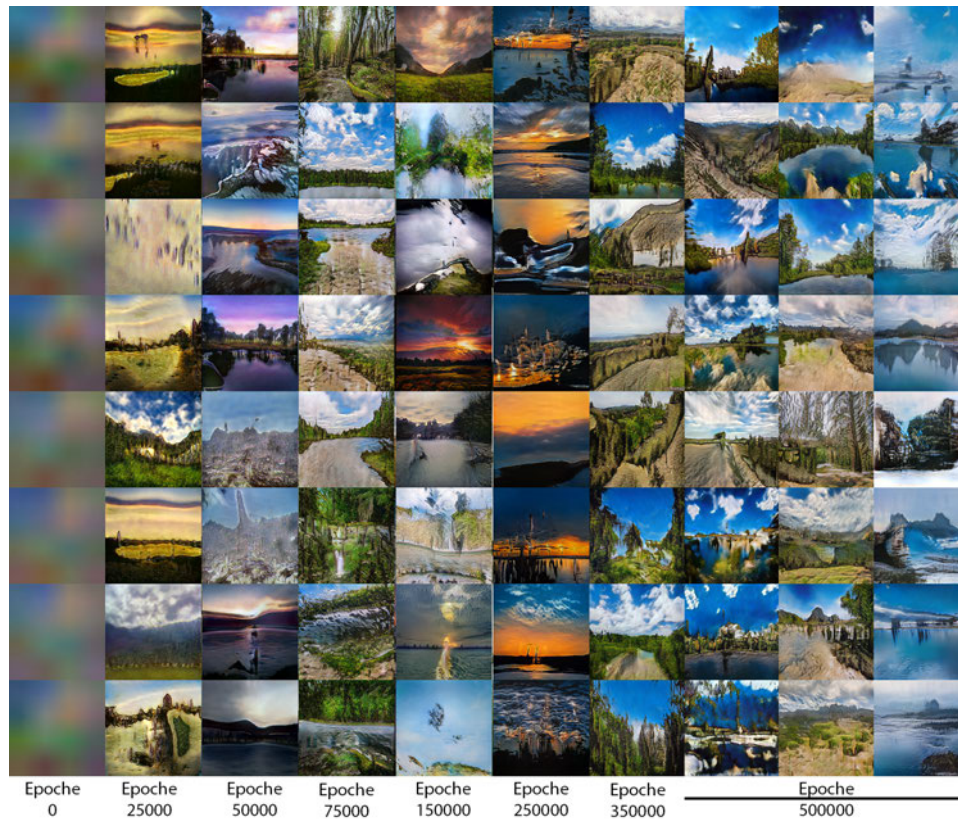


Abbildung 5.17: Zwischenergebnisse verschiedenster Epochen während des Trainings.

Quelle: Huy-Bao Le

5.3.3 Evaluation

Auch in diesem Fall wird die Evaluation des Models anhand einer visuellen Bewertung vollzogen. Zudem wird der Verlauf des Losses analysiert welche weitere Erkenntnisse geben soll für das Training des Style-GANs.

- Positive Ergebnisse [A.5]:
 - Das Model hat eine Vielfalt an Bildern geschaffen mit verschiedensten Kompositionen.

- Die erzeugten Bilder wirken relativ realitätsnah in Bezug auf Farben, Kontraste und klarer Linien.
- Details wie die Abgrenzung von Himmel, Horizont und Land wurden erkannt.
- Falls Wasser zu sehen ist, wurden Reflektionen gut erkannt (Abb. A.5a, Abb. A.5b).
- Negative Ergebnisse [A.6, A.7]:
 - Den Bildern fehlt es an genaueren Details, bei näheren Betrachten sind Details teilweise "wolkig" (Abb. A.6d).
 - Einige Objekte im Bild sind teilweise deformiert und wirken realitätsfern (Abb. A.6a).
 - Es wurden vorwiegend Landschaftsbilder generiert in verschiedenen Kompositionen.
 - Einige Kompositionen machen keinen Sinn, so gibt es teilweise Schneehaufen in einer Frühlingsumgebung (Abb. A.7d).

Die Ergebnisse zeigen, dass ein gewisses Level an Detail zu Realität vorhanden ist. Beim Näheren betrachten stellt sich jedoch heraus, dass feinere Details etwas "wolkig" dargestellt werden, das spricht dafür, dass die feinen Details noch nicht wirklich vom Model verstanden wurden. Als mögliche Gründe für dieses Verhalten werden folgende Vermutungen aufgestellt:

- Im verwendeten Datensatz gibt es eine viel zu große Varianz an Bildern, vorwiegend Landschaftsbilder, weniger Personen, kaum Städte, welches das Erlernen von Details schwierig macht
- Die verwendeten Ausgangsbildgrößen mit 256×256 sind zu klein relativ zur Inputgröße des Generators. Dadurch ist das Noise Layer in Relation zu groß damit das GAN feine Details erkennen kann.
- Im Vergleich zu den erwarteten Trainingszeiten 5.2 wurde das Training verkürzt und zudem mit schwächeren privaten GPUs trainiert. Dadurch könnte es sein, dass das Model zu wenig Zeit hatte um weitere Details zu Erlernen.

In der Abbildung 5.18 ist zu erkennen, dass das Model sich stabil gehalten und hat keine Anzeichen zeigt zu kollabieren während der Trainingsphase.

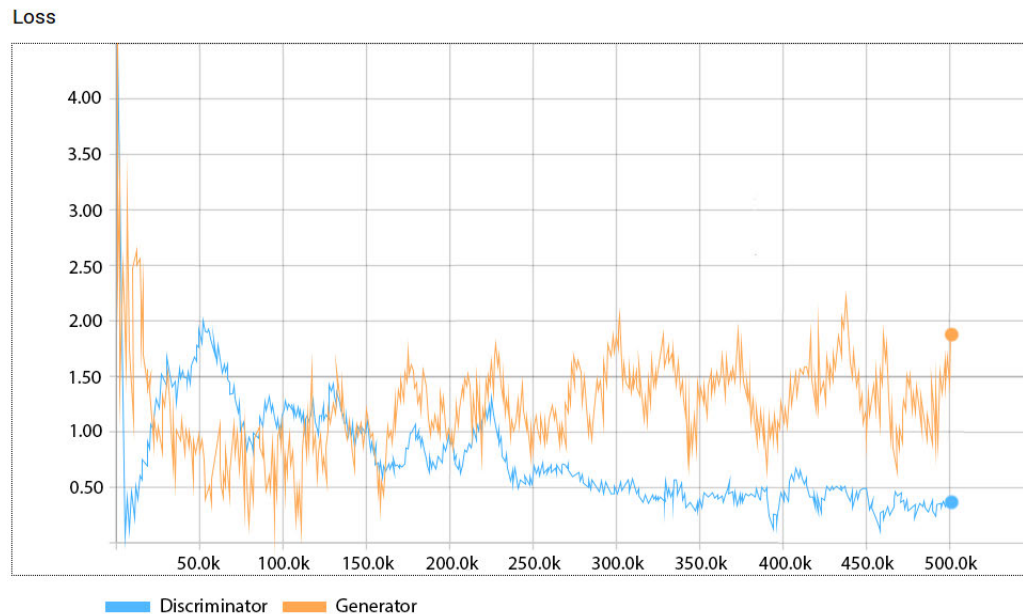


Abbildung 5.18: StyleGAN Loss während des Trainings. In Orange der geplottete Loss des Generators. In Blau der Loss des Discriminators.

Quelle: Huy-Bao Le

Ein Vorteil dieser Art für die Generierung von Bildern ist, dass nur ein Datensatz vorhanden sein muss. Durch diese Methode kann man weitere artgleiche Bilder generieren. Ein negativer Punkt wäre, dass diese neu generierten Bilder komplett randomisiert erstellt werden und bei großer Varianz in den Trainingsdaten das zu erwartete Bild schwer vorherzusagen ist. Ein spezifischer Datensatz, welcher weniger verschiedene Bilder hätte, würde das Training vermutlich verkürzen und würde bei der Erzeugung der Bilder genauere und detaillierte Bilder erzeugen. Aufgrund dieser Varianz ist ein automatisiertes Labeling der generierten Datensätze nicht möglich. Das würde dementsprechend zusätzlich Zeit benötigen und somit auch ein negativer Aspekt dieses Modells sein. Vermutlich hätte ein effektiveres Training mit besseren Grafikkarten sowie längere Trainingszeiten das Ergebnis verbessert. Nichts desto trotz zeigt dieser Versuch, dass ein großes Potenzial bei dieser Art der Bildgenerierung vorhanden ist.

6 Gesamt Evaluation

Beide Implementierungen haben einen Erfolg bei der Erzeugung von realistischen Bildern gezeigt. Durch die Auswahl eines sehr umfangreichen Datensatzes mit vielen verschiedenen Szenarien und Motiven, konnten zwei unterschiedliche Architekturen gewählt werden, welche das Ziel haben synthetische Bilder zu erzeugen, welche auf das menschliche Auge realistisch wirken.

Der Cycle GAN Ansatz bietet dabei einen guten Ansatz für eine vollständig synthetische Generierung von Daten. In Kombination mit computergenerierten Szenen von bereits existierenden Frameworks im Bereich der Computergrafik lassen sich gewünschte Kompositionen von Daten schnell erzeugen und in die Realität übersetzen. Dabei sollte eine Balance geschaffen werden zwischen den Aufwand bei der Erzeugung über die Computergrafik und bei der Erstellung des Datensatzes für die Realität. Obwohl Modelle, die rein mit der Computergrafik generierte Daten trainiert wurden, schon gute Erfolge bei der Anwendung mit realistischen Bildern zeigten, erzielte man bessere Erfolge wenn man die Daten mit Hilfe von generativen Netzen augmentiert. Dadurch erspart man sich nicht nur Zeit bei der aufwendigen Modellierung der Grafikszenen sondern kann auch ohne tiefes Verständnis in der Computergrafik Szenen unaufwendig designen und durch maschinelles Lernen in die Realität transferieren. Ein weiterer Vorteil besteht durch die vollständige Information bei der Synthese von Bildern in der Computergrafik: durch das Cycle-GAN Model können bereits generierte Label vollständig übernommen werden, da die Komposition des Bildes nicht verändert wird. Ein Nachteil dieses Ansatzes wäre die limitierte Anzahl an Bildern die man generiert. Bei der Übersetzung werden lediglich die vorhandenen computergenerierten Bilder in die Realität augmentiert. Komplett neue Bilder mit neuen Kompositionen kann man dadurch nicht erhalten.

Der StyleGAN Ansatz bietet dem Nutzer eine Möglichkeit viele, neue randomisierte Bilder zu erzeugen. Dadurch lassen sich Datensätze generieren in einer hohen Varianz an Kompositionen im zukünftigen Einsatz von zu trainieren Modellen das Model stabiler machen können. Das Training eines solchen Modells benötigt aber viel Zeit, selbst mit

den aktuellen Stand der Hardware. Zudem ist das Model fehleranfällig und kann leichter kollabieren im Vergleich zu dem CycleGAN Model. Das in dieser Arbeit trainierte Modell zeigte sich als möglichen Weg realistische Bilder zu erzeugen, jedoch haben die Ergebnisse noch Raum für Verbesserungen. Ein Nachteil dieser Anwendung sind die nicht gelabelten generierten Bilder. Diese müsste man bei Bedarf manuell hinzufügen, welche jedoch mit weiterem Aufwand verbunden ist. Die aufgelisteten Gründe in dem Kapitel 5.3.3 bieten eine gute Aussicht das Model in Zukunft zu optimieren.

Beide umgesetzten Modelle bieten einen Einstiegspunkt in die Generierung von Datensätzen für zukünftige Modelle. Dabei müsste man in Betracht ziehen welchen Anwendungsfall man mit den Datensätzen verfolgt. Als Option lassen sich beide Modelle auch als Erweiterung für bereits vorhandene Datensätze sehen. Diese Erweiterung ist mit der Augmentierung von Datensätzen gleichzusetzen. Durch das hinzufügen der synthetischen Bilder erhält der Datensatz mehr Varianz, welches zu stabileren Modellen führen kann.

7 Future Work

Im vorangegangenen Kapitel wurden diverse Probleme bei den Modellen identifiziert. Einige Probleme des Cycle-GAN sind auf den Datensatz zurückzuführen. Eine tatsächliche Generierung von synthetischen Daten mit BlenderProc würde bei der Übersetzung mit einer Cycle-GAN Architektur möglicherweise das Model verbessern. Die Daten wären durch die Erzeugung von Skripten konsistent und könnten den Datensatz ausgleichen.

Die Ergebnisse des StyleGAN Models zeigen noch Raum für Optimierungen. Die möglichen Gründe 5.3.3 können hierbei ein guter Einstieg sein für weitere Arbeiten.

Beide Implementierungen können mit Hilfe von stärkerer Hardware und Zeit schnellere und bessere Ergebnisse erzielen.

Das Training eines Models mit synthetischen Daten wurde in dieser Arbeit nicht behandelt. Ein Vergleich zu konventionellen trainierten Models anhand einer realen Anwendung, z.B. mittels einer Object-Detection, wäre ein möglicher Ansatz für weitere Arbeiten.

7.1 Fazit

Diese Arbeit beschäftigte sich mit der Erzeugung realistischer Trainingsdaten mit generativen neuronalen Netzen. Seit der Festlegung des Themas ging es mit der Recherche und Umsetzung los. Dabei wurden durch die Aneignung an Wissen über generelle GAN Architekturen, das Training von GANs und die Anwendung und Optimierung solcher Modelle, Ansätze gewählt welche verschiedene Möglichkeiten bieten solche Datensätze zu generieren.

Dabei wurden zwei State-of-the-Art Architekturen ausgewählt. Der CycleGAN Ansatz brauchte für das Training einen weiteren Datensatz um die Übersetzung in ein realistisches Bild zu ermöglichen. Die Computergrafik mit dem BlenderProc Framework bietet

dabei einen guten Ansatz solche Datensätze unaufwendig synthetisches zu generieren und dabei vollständige Informationen über die Daten zu haben. Dieses ermöglicht die Annotation von Daten bereits vor der Erzeugung. StyleGAN bietet die Möglichkeit neue Bilder zu generieren welche eine hohe Varianz an Schlüssel-Merkmalen einer Domäne besitzen.

Beide Modelle wurden auf privaten GPUs trainiert. Dabei wurden im Trainingsvorgang regelmäßig Zwischenergebnisse generiert um die Entwicklung zu beobachten um ggf. bei möglichen Model collapse zu reagieren. Aufgrund der Größe und der Komplexität der state-of-the-Art Modelle hat das Training einen großen Teil der Bearbeitungszeit eingenommen. Beide Modelle haben sich als denkbaren Weg zur Zielführung erwiesen. Trotz nicht optimalen Ergebnissen ist die Evaluation beider Ansätze positiv ausgefallen und bieten Ausgangspunkte für die Erzeugung solcher Datensätze. Raum für Verbesserungen sind möglich durch Anpassungen der Hyperparameter beim Training sowie die bereits erkannte Probleme könnten genutzt werden um weitere Perspektiven für Arbeiten zu entwickeln. Trotz der brauchbaren Ergebnisse wurden potenzielle zukünftige Ansätze identifiziert, welche auf dieser Arbeit aufbauen könnten.

Literaturverzeichnis

- [1] ABADI, Martín ; AGARWAL, Ashish ; BARHAM, Paul ; BREVDO, Eugene ; CHEN, Zhifeng ; CITRO, Craig ; CORRADO, Greg S. ; DAVIS, Andy ; DEAN, Jeffrey ; DEVIN, Matthieu ; GHEMAWAT, Sanjay ; GOODFELLOW, Ian ; HARP, Andrew ; IRVING, Geoffrey ; ISARD, Michael ; JIA, Yangqing ; JOZEFOWICZ, Rafal ; KAISER, Lukasz ; KUDLUR, Manjunath ; LEVENBERG, Josh ; MANÉ, Dan ; MONGA, Rajat ; MOORE, Sherry ; MURRAY, Derek ; OLAH, Chris ; SCHUSTER, Mike ; SHLENS, Jonathon ; STEINER, Benoit ; SUTSKEVER, Ilya ; TALWAR, Kunal ; TUCKER, Paul ; VANHOUCHE, Vincent ; VASUDEVAN, Vijay ; VIÉGAS, Fernanda ; VINYALS, Oriol ; WARDEN, Pete ; WATTENBERG, Martin ; WICKE, Martin ; YU, Yuan ; ZHENG, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. – URL <http://tensorflow.org/>. – Software available from tensorflow.org
- [2] BORJI, Ali: Pros and Cons of GAN Evaluation Measures. In: *CoRR* abs/1802.03446 (2018). – URL <http://arxiv.org/abs/1802.03446>
- [3] BROWNLEE, Jason: *How to evaluate generative adversarial networks*. <https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/>. 2019
- [4] BROWNLEE, Jason: *How to train stable generative adversarial networks*. <https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/>. 2019
- [5] CHOLLET, François: *Deep Learning with Python*. Manning, November 2017. – ISBN 9781617294433
- [6] COMMUNITY, Blender O.: *Blender - a 3D modelling and rendering package*. Stichting Blender Foundation, Amsterdam: Blender Foundation (Veranst.), 2018. – URL <http://www.blender.org>

- [7] DENNINGER, Maximilian ; SUNDERMEYER, Martin ; WINKELBAUER, Dominik ; ZIDAN, Youssef ; OLEFIR, Dmitry ; ELBADRAWY, Mohamad ; LODHI, Ahsan ; KATAM, Harinandan: *BlenderProc*. 2019
- [8] DUMOULIN, Vincent ; VISIN, Francesco: *A guide to convolution arithmetic for deep learning*. 2016
- [9] FOSTER, David ; SAFARI, an O'Reilly Media C.: *Generative Deep Learning*. 1st edition. O'Reilly Media, Inc, 2019. – URL https://safarivjv.auth0.com/authorize?client_id=Utnilml1RXgzYFIwZrhSxell9EDRaL2v&response_type=code&connection=queensland-university-of-technology&redirect_uri=https://www.safaribooksonline.com/complete/auth0-oauth2/&state=/library/view/-/9781492041931/?ar. – Electronic reproduction. Boston, MA : Safari. Available via World Wide Web
- [10] GOODFELLOW, Ian: *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2016
- [11] GOODFELLOW, Ian J. ; POUGET-ABADIE, Jean ; MIRZA, Mehdi ; XU, Bing ; WARDE-FARLEY, David ; OZAIR, Sherjil ; COURVILLE, Aaron ; BENGIO, Yoshua: *Generative Adversarial Networks*. 2014
- [12] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: *Deep Residual Learning for Image Recognition*. 2015
- [13] HEUSEL, Martin ; RAMSAUER, Hubert ; UNTERTHINER, Thomas ; NESSLER, Bernhard ; HOCHREITER, Sepp: *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2017
- [14] ISOLA, Phillip ; ZHU, Jun-Yan ; ZHOU, Tinghui ; EFROS, Alexei A.: *Image-to-Image Translation with Conditional Adversarial Networks*. 2016
- [15] JUNG, Alexander B. ; WADA, Kentaro ; CRALL, Jon ; TANAKA, Satoshi ; GRAVING, Jake ; REINDERS, Christoph ; YADAV, Sarthak ; BANERJEE, Joy ; VECSEI, Gábor ; KRAFT, Adam ; RUI, Zheng ; BOROVEC, Jirka ; VALLENTIN, Christian ; ZHYDENKO, Semen ; PFEIFFER, Kilian ; COOK, Ben ; FERNÁNDEZ, Ismael ; DE RAINVILLE, François-Michel ; WENG, Chi-Hung ; AYALA-ACEVEDO, Abner ; MEUDEEC, Raphael ; LAPORTE, Matias u. a.: *imgaug*. <https://github.com/aleju/imgaug>. 2020. – Online; accessed 01-Feb-2020

- [16] KARRAS, Tero ; AILA, Timo ; LAINE, Samuli ; LEHTINEN, Jaakko: *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018
- [17] KARRAS, Tero ; LAINE, Samuli ; AILA, Timo: *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018
- [18] KARRAS, Tero ; LAINE, Samuli ; AILA, Timo: A Style-Based Generator Architecture for Generative Adversarial Networks. In: *CoRR* abs/1812.04948 (2018). – URL <http://arxiv.org/abs/1812.04948>
- [19] KARRAS, Tero ; LAINE, Samuli ; AILA, Timo: *StyleGAN — Official TensorFlow Implementation*. <https://github.com/NVlabs/stylegan>. 2019
- [20] KARRAS, Tero ; LAINE, Samuli ; AITTALA, Miika ; HELLSTEN, Janne ; LEHTINEN, Jaakko ; AILA, Timo: *Analyzing and Improving the Image Quality of StyleGAN*. 2019
- [21] KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; HINTON, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F. (Hrsg.) ; BURGESS, C. J. C. (Hrsg.) ; BOTTOU, L. (Hrsg.) ; WEINBERGER, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, S. 1097–1105. – URL <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [22] MAXIMILIAN DENNINGER, MARTIN SUNDERMEYER, DOMINIK WINKELBAUER, DMITRY OLEFIR, TOMAS HODAN, YOUSSEF ZIDAN, MOHAMAD ELBADRAWY, MARKUS KNAUER, HARINANDAN KATAM, AHSAN LODHI: *BlenderProc: Reducing the Reality Gap with Photorealistic Rendering*. <https://sim2real.github.io/assets/papers/2020/denninger.pdf>. 2019. – [Online; accessed 22-August-2020]
- [23] METZ, Luke ; POOLE, Ben ; PFAU, David ; SOHL-DICKSTEIN, Jascha: Unrolled Generative Adversarial Networks. In: *CoRR* abs/1611.02163 (2016). – URL <http://arxiv.org/abs/1611.02163>
- [24] NAWAL HAFEZ: *Closing the Gap to Real Data*. <https://wiki.tum.de/display/ldv/Reality+Gap>. 2019. – [Online; accessed 22-August-2020]
- [25] PASINI, Marco: *10 Lessons I Learned Training GANs for one Year*. <https://towardsdatascience.com/10-lessons-i-learned-training->

- [generative-adversarial-networks-gans-for-a-year-c9071159628](#). 2019
- [26] RADFORD, Alec ; METZ, Luke ; CHINTALA, Soumith: *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015
- [27] SHEN, Kevin: *Effect of batch size on training dynamics*. <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#:~:text=Interestingly%20we%20can%20recover%20the,1024%20only%20achieves%20about%2096%25>. 2018
- [28] VAN ROSSUM, Guido ; DRAKE, Fred L.: *Python 3 Reference Manual*. Scotts Valley, CA : CreateSpace, 2009. – ISBN 1441412697
- [29] WIKIPEDIA CONTRIBUTORS: *Ray tracing (graphics)* — *Wikipedia, The Free Encyclopedia*. [https://en.wikipedia.org/w/index.php?title=Ray_tracing_\(graphics\)&oldid=972772193](https://en.wikipedia.org/w/index.php?title=Ray_tracing_(graphics)&oldid=972772193). 2020. – [Online; accessed 22-August-2020]
- [30] ZHU, Jun-Yan ; PARK, Taesung ; ISOLA, Phillip ; EFROS, Alexei A.: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017

A Anhang



(a) Positiv Realität 1.

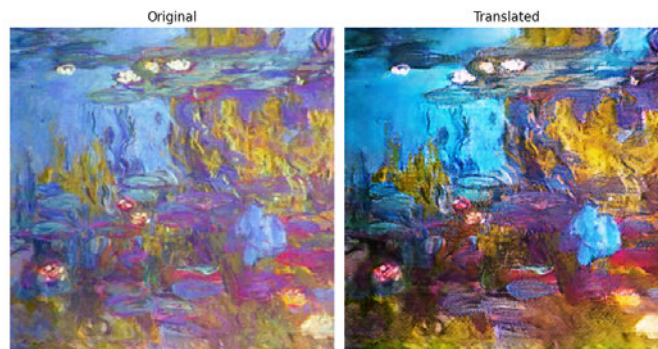


(b) Positiv Realität 2.



(c) Positiv Realität 3.

Abbildung A.1: Evaluierungsdaten Cycle-GAN in die Realität, positiv
Quelle: Huy-Bao Le



(a) Negativ Realität 1.



(b) Negativ Realität 2.



(c) Negativ Realität 3.

Abbildung A.2: Evaluierungsdaten Cycle-GAN in die Realität, negativ
Quelle: Huy-Bao Le

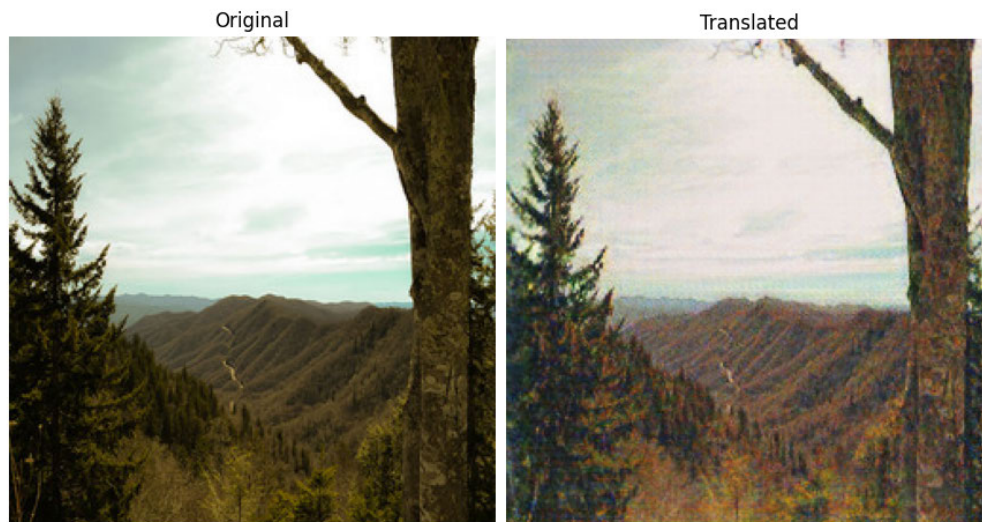


(a) Positiv Monet 1.

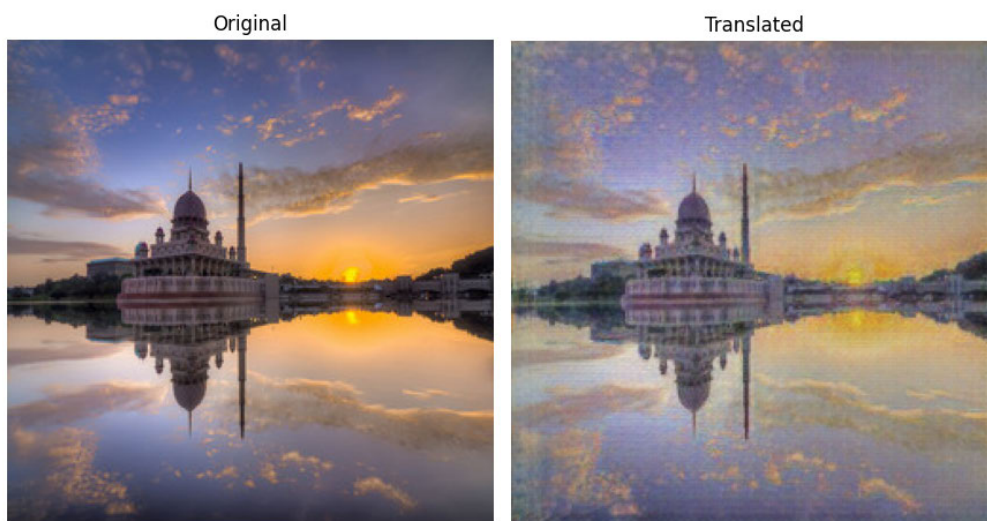


(b) Positiv Monet 2.

Abbildung A.3: Evaluierungsdaten Cycle-GAN in die Monet-Domäne, positiv
Quelle: Huy-Bao Le

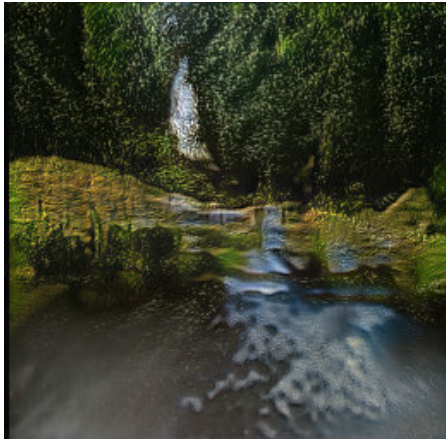


(a) Negativ Monet 1.



(b) Negativ Monet 2.

Abbildung A.4: Evaluierungsdaten Cycle-GAN in die Monet-Domäne, negativ
Quelle: Huy-Bao Le



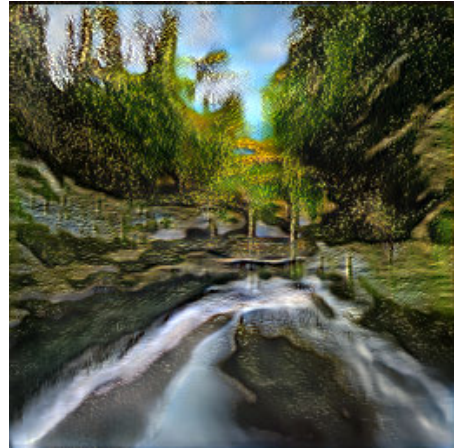
(a) Positiv StyleGAN 1.



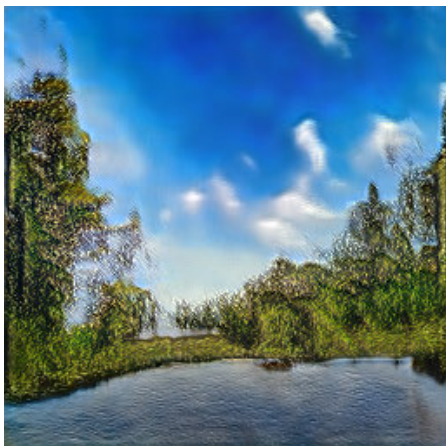
(b) Positiv StyleGAN 2.



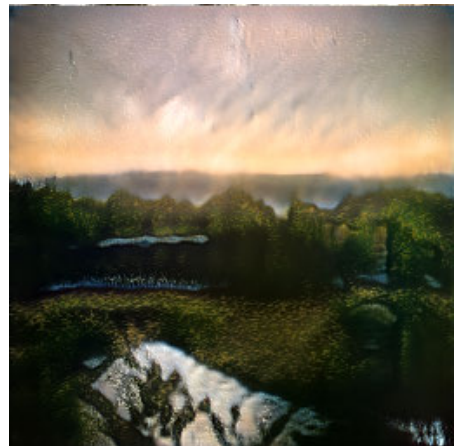
(c) Positiv StyleGAN 3.



(d) Positiv StyleGAN 4.



(e) Positiv StyleGAN 5.



(f) Positiv StyleGAN 6.

Abbildung A.5: Evaluierungsdaten StyleGAN, Positiv
Quelle: Huy-Bao Le



(a) Negativ StyleGAN 1.



(b) Negativ StyleGAN 2.



(c) Negativ StyleGAN 3.



(d) Negativ StyleGAN 4.

Abbildung A.6: Evaluierungsdaten StyleGAN, Negativ 1. Teil

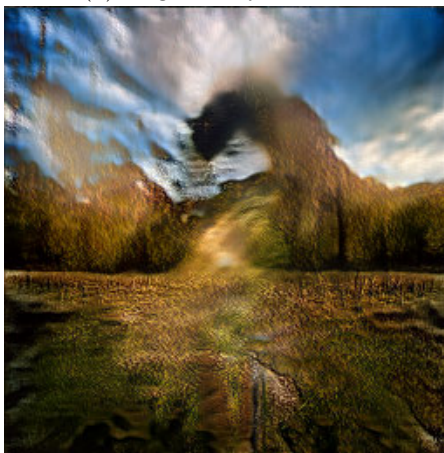
Quelle: Huy-Bao Le



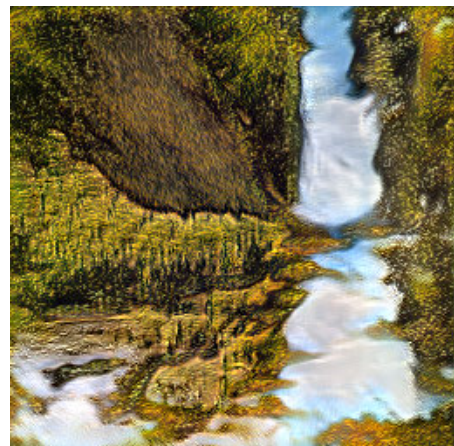
(a) Negativ StyleGAN 1.



(b) Negativ StyleGAN 2.



(c) Negativ StyleGAN 3.



(d) Negativ StyleGAN 4.

Abbildung A.7: Evaluierungsdaten StyleGAN, Negativ 2. Teil

Quelle: Huy-Bao Le

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Erzeugung realistischer Trainingsdaten mit generativen Neuronalen Netzen

ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und 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