HAW HAMBURG

Bachelor

Julian Tamm

Semantische Latent Space Exploration mit StyleGANs

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

Julian Tamm

Semantische Latent Space Exploration mit StyleGANs

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

Prüfer: Prof. Dr. Michael Neitzke
Zweitprüfer: Prof. Dr. Peer Stelldinger

Eingereicht am: 09.06.2023

Julian Tamm

Thema der Arbeit

StyleGANs und deren Latent Spaces

Stichworte

StyleGAN, unüberwachtes Lernen, Bildgenerierung, Latent-Space-Erkundung, Richtungen im Latent Space

Kurzzusammenfassung

In dieser Bachelorarbeit werden StyleGANs und Verfahren zum Untersuchen von deren Latent Spaces vorgestellt. Hierbei wird ein StyleGAN2-Ada auf einen Datensatz mit 64px großen Bildern trainiert und anschließend mit zwei unüberwachten Verfahren zum Finden von Richtungen im Latent Space untersucht. Zum einen wird das Verfahren nach Unsupervised Discovery of Interpretable Directions in the GAN Latent Space und zum anderen das Verfahren nach CLIP2StyleGAN genauer beleuchtet und angewandt. Die gefundenen Richtungen konnten dann durch manuelle Anpassungen präzisiert werden. Somit wurden erfolgreich semantisch bedeutsame Richtungen im Latent Space gefunden und durch diese wiederum Rückschlüsse auf den Latent Space gezogen.

Julian Tamm

Title of Thesis

StyleGANs and their Latent Spaces

Keywords

StyleGAN, unsupervised learning, image generation, latent space exploration, latent direction

Abstract

In this bachelor thesis, StyleGANs and methods for investigating their latent spaces are introduced. A StyleGAN2-Ada is trained on a dataset of 64px images and then examined using two unsupervised methods for finding directions in the latent space. The method of Unsupervised Discovery of Interpretable Directions in the GAN Latent Space and the CLIP2StyleGAN method are both examined and applied. The discovered directions were further refined through manual adjustments. As a result, semantically meaningful directions in the latent space were successfully identified, allowing for inferences to be drawn about the latent space.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

FID	Fréchet inception distance
SVM	Support Vektor Machine
ICLR	International Conference on Learning Representations
CNN	Convolutional Neural Network

Glossar

Latent Vektor / Latent Code	Ein Latent Vektor beschreibt den Input eines GAN-Generators. Im folgenden Text werden Latent Vektor und Latent Code als Synonym benutzt.
Unit	Als Units bezeichnet man Knoten innerhalb eines neuralen Netzwerks. Es werden auch Einheit und Knoten als Synonym benutzt.
kimg	Ein kimg beinhaltet 1.000 Bilder und stellt einen Schritt im Trainingsprozesses eines StyleGAN2-Ada dar.
arXiv e-paper	arXiv e-papers ist ein kostenloser Distributionsdienst, der auf einem Open-Access-Archiv aufsetzt.
ICLR	**International Conference on Learning Representations** ist eine Konferenz für maschinelles Lernen, die seit 2013 in der Regel Ende April oder Anfang Mai eines jeden Jahres stattfindet.
Verwober/entangled Latent Space	Bei einem verwobenen Latent Space handelt es sich um einen Latent Space, der noch nicht interpretiert wurde oder bei dem eine Interpretation nicht vollständig möglich ist.
Verwobene Richtung	Bei einer verwobenen (entangled und/oder verflechtet) Richtung handelt es sich um eine Richtung im Latent Space, die mehr als eine Veränderung in einem visuellen Konzept darstellt.

1 Einleitung

1.1 Motivation

Das Feld der Bildgenerierungs-KIs hat in den letzten Jahren erheblich an Bedeutung gewonnen, insbesondere durch die Fortschritte bei Generative Adversial Networks (GANs) wie StyleGANs. Doch das gezielte Generieren von Bildern unterlag immer einem Problem. Bei einem GAN entscheidet ein Latent Vektor über die Eigenschaften des generierten Bildes. Dieses Bild entspricht zwar dem trainierten Datensatz, aber im Zweifelsfall nicht der gewünschten spezifischen Ausprägung. Wenn ein StyleGAN mit Hundebildern trainiert wurde, war es zufällig, ob ein Bild eines Hundes mit schwarzem Fell erzeugt wurde oder nicht. Das bedeutet: Im Falle eines zufälligen Latent Vektors ist die Ausgabe auch zufällig. Um diese Bilder aber aktiv im Entstehungsprozess beeinflussen zu können, muss der Latent Space des StyleGANs verstanden werden. Indem der Latent Space analysiert und manipuliert wird, können die gewünschten Eigenschaften der generierten Bilder gezielt gesteuert werden. Dieser Ansatz hat unter anderem das Potenzial, bei der prozeduralen Erstellung von Videospielinhalten von großem Interesse zu sein.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird aufgezeigt, wie zielgerichtet durch manuelle Veränderungen eines Latent Vektors, ein Bild mit speziellen Attributen generiert werden kann. Diese Attribute können beispielsweise ein roter Hintergrund oder blonde Haare sein. Hierbei sollen Richtungen im Latent Space gefunden und sinnvoll genutzt werden. Als Datensatz sollen vergleichsweise kleine (ca. 16x16 Pixel) und wenige (weniger als 10.000) Bilder genutzt wer-

den. Ziel ist es, mit dem zur Verfügung gestellten Datensatz ein StyleGAN zu trainieren und dessen Latent Space nach semantisch interessanten Richtungen zu untersuchen.

Hierzu sollen:

- unterschiedliche StyleGANs auf ihre Kompatibilität mit diesem Datensatz untersucht werden,
- der ausgewählte StyleGAN vollständig mit dem Datensatz trainiert werden,
- der Latent Space des trainierten StyleGANs mit Hilfe von zwei unüberwachten Verfahren nach Richtungen untersucht werden,
- diese Richtungen manipuliert werden, um mögliche Verflechtungen zu lösen.

Diese Bachelorarbeit stellt somit einen geeigneten Ablauf vor, mit dem semantisch interessante Richtungen gefunden und optimiert werden können.

1.3 Inhalt der Arbeit

Inklusive dieser Einleitung ist die Arbeit in insgesamt 6 Kapitel unterteilt.

In **Kapitel 2 Grundlagen** wird ein Überblick über GAN's/StyleGAN's gegeben, um die zugrundeliegenden Verfahren zur Bildgenerierung aufzuzeigen.

In **Kapitel 3 Konzept und Methoden** wird das Konzept des praktischen Teils der Arbeit vorgestellt. Gleichzeitig werden die verwendeten Methoden beschrieben.

In **Kapitel 4 Umsetzung** werden die Methoden und Ansätze aus Kapitel 3 in Bezug auf deren Umsetzung beleuchtet.

In **Kapitel 5 Ergebnisse** werden die Ergebnisse der unterschiedlichen Methoden aus Kapitel 3 und 4 vorgestellt und diskutiert.

In **Kapitel 6 Fazit** wird die Arbeit zusammengefasst, ein Fazit gezogen und ein Ausblick geboten.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der generativen Erstellung von Daten (Bildern) mittels GANs (progressive GAN, StyleGAN) beschrieben. Außerdem wird auf Klassen von allgemeinen Problemen beim Trainieren dieser Netze eingegangen und wichtige Begriffe erklärt.

2.1 Generative Adversarial Networks

GANs sind ein Verbund aus zwei unterschiedlichen Netzwerken. Zum einen ein **Discriminator**, der als Input Daten aus dem gewünschten Ergebnisraum entgegennimmt und dann entscheidet, ob diese real oder generiert sind. Und zum anderen ein **Generator**, der einen zufälligen Eingabevektor auf den gewünschten Ergebnisraum abbildet und somit ein neues Datum generiert. Die Gewichtung innerhalb des Generators wird mit Hilfe des Discrimiators geuptated [1]. Im Laufe dieser Arbeit wird der Ergebnisraum auch Bild-Raum oder vereinfacht Bilder genannt, da das Hauptaugenmerk der Arbeit das Generieren von Bildern ist. Das Gleiche gilt für den Begriff Datum welcher auch als Bild bezeichnet wird.

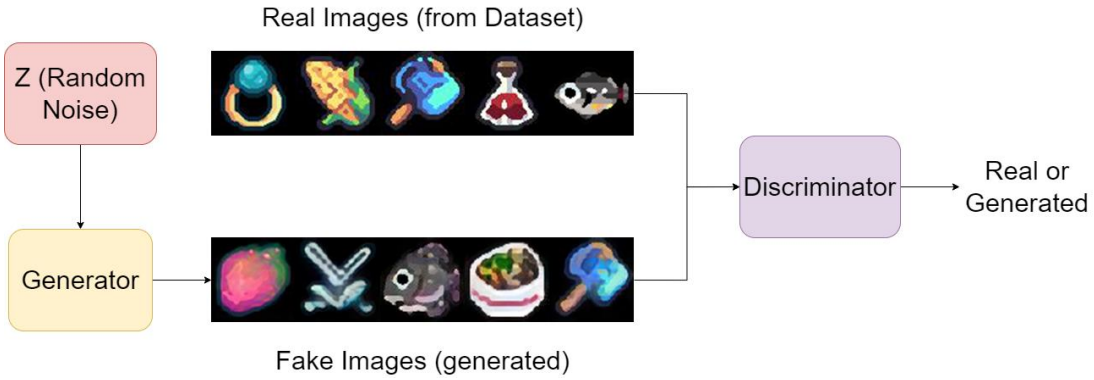

Abbildung 1: Semantische Darstellung eines GANs

In der schematischen Darstellung in Abbildung 1 sieht man exemplarisch das Zusammenspiel zwischen Generator und Discriminator. Der Generator erstellt aus einem zufälligen Eingabevektor (z), auch Latent Vektor oder Latent Code genannt, Bilder. Der Discriminator wird dann anhand echter Bilder und generierter Bilder trainiert. Der Generator wird anhand der Entscheidung des Diskriminators über seine Bilder trainiert.

Wichtig ist, dass beide Netzwerke ungefähr gleich schnell lernen, damit nicht jedes Datum des Generators als real angesehen wird oder umgekehrt: Nicht jedes Datum als generiert erkannt wird.

Adversarial networks

Adversarial Networks oder auch kontradiktorische (gegnerische) Netzwerke arbeiten gegeneinander und werden somit simultan trainiert. Im Falle von GANs versucht der Generator Daten zu erzeugen, die den Discriminator täuschen und somit als echte Daten angesehen werden. Der Discriminator versucht hingegen, die generierten Daten so gut wie möglich von den echten Daten zu unterscheiden. Das heißt, die Netzwerke spielen ein Nullsummenspiel: Ein Zwei-Spieler-Min-Max-Spiel, das wie folgt beschrieben werden kann (mit $\mathbb{E}$ = Erwartungswert):

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Abbildung 2: Min-Max-Spiel beschreibende Funktion, entnommen aus [1]

D, der Discriminator, bekommt als Eingabewert x ein Datum und gibt einen Skalar zurück, der die Wahrscheinlichkeit darstellt, dass x Teil des „echten“ Datensatzes ist. D wird so trainiert, dass der Ausgabewert von D(x) maximiert wird. G, der Generator, bekommt als Eingabewert z und gibt ein passendes Datum zurück. G wird so trainiert, dass log(1 – D(G(z))) minimiert wird. [1] [2](S.11 f)

Der Algorithmus

for Anzahl der Trainings Iterationen **do**

  for k Schritte **do**

  - Sammeln von m zufälligen Eingabevektoren $\{z^{(1)}, \dots, z^{(m)}\}$.
  - Sammeln von m Beispielen $\{x^{(1)}, \dots, x^{(m)}\}$ aus dem echten Datensatz.
  - Updates des Discriminators mit einem Stochastic Gradient Descent (SGD) ähnlichen Algorithmus (z.B. Adam) mit einer passenden Loss Funktion.

  end for

- Erneutes Sammeln von m zufälligen Eingabevektoren $\{z^{(1)}, \dots, z^{(m)}\}$.
- Updaten des Generators mit einem SGD ähnlichen Algorithmus mit einer passenden Loss Funktion

end for

Der Wert k ϵ $\mathbb{N}$ ist hierbei frei wählbar.

Passende Loss Funktionen wären zum Beispiel:

Für den Discriminator:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

Abbildung 3: Discriminator Loss-Funktion, entnommen aus [1]

Für den Generator:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Abbildung 4: Generator Loss-Funktion, entnommen aus [1]

Diese lassen sich direkt von der Min-Max-Spiel-Funktion ableiten. Alternativ kann man auch die Loss Funktion (Verlustfunktion) „Binary Cross-Entropy“ [3] [4] verwenden, da mit genau zwei Status („generiert“ und „echt“) gearbeitet wird. [1]

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Abbildung 5: Binary Cross-Entropy-Loss-Funktion, entnommen aus [4]

2.2 Latent Space

Der Latent Space (verborgener Raum) in einem GAN ist ein Raum von Vektoren mit niedriger Dimensionalität, typischerweise 100 bis 512 Dimensionen (bei GANs), die als Eingabe für den Generator des GANs verwendet werden. Diese Vektoren repräsentieren latente Merkmale oder Eigenschaften, die der Generator verwenden kann, um Bilder oder andere Daten zu generieren. Ein Latent Space ist während des Trainierens nicht einsehbar oder veränderbar. Er beschreibt somit einen Raum, der erst erkundet werden kann, nachdem das Training abgeschlossen ist. [1] [5]

Eine weitere wichtige Eigenschaft des Latent Space ist, dass er kontinuierlich ist [1] [5]. Das bedeutet, dass Änderungen im Latent Space in der Regel zu Änderungen im generierten Output führen. Wenn man beispielsweise eine bestimmte Eigenschaft eines generierten Bildes ändern möchte, kann man den entsprechenden Vektor (Richtung) im Latent Space manipulieren, um diese Änderung zu erzielen.

Das Erkunden eines Latent Spaces, insbesondere der eines StyleGANs (eine Weiterentwicklung des GANs), bildet den Hauptfokus dieser Arbeit.

2.3 GAN-Aufbau

2.3.1 Aufbau von Neuralen Netzwerken

Model: "generator"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2048)	206848
leaky_re_lu_4 (LeakyReLU)	(None, 2048)	0
reshape (Reshape)	(None, 4, 4, 128)	0
conv2d_transpose (Conv2DTran	(None, 8, 8, 128)	262272
leaky_re_lu_5 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 16, 16, 128)	262272
leaky_re_lu_6 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_2 (Conv2DTr	(None, 32, 32, 32)	65568
leaky_re_lu_7 (LeakyReLU)	(None, 32, 32, 32)	0
conv2d_4 (Conv2D)	(None, 32, 32, 3)	1539

Total params: 798,499
Trainable params: 798,499
Non-trainable params: 0

Model: "discriminator"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 64)	1792
leaky_re_lu (LeakyReLU)	(None, 16, 16, 64)	0
conv2d_1 (Conv2D)	(None, 8, 8, 128)	73856
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	147584
leaky_re_lu_2 (LeakyReLU)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 2, 2, 256)	295168
leaky_re_lu_3 (LeakyReLU)	(None, 2, 2, 256)	0
global_max_pooling2d (Global	(None, 256)	0
dense (Dense)	(None, 1)	257

Total params: 518,657
Trainable params: 518,657
Non-trainable params: 0

Abbildung 6 Aufbau eines Generators und eines Discriminators

Abbildung 6: zeigt den Aufbau der neuronalen Netze für den CIFAR10 Datensatz.

Der Discriminator besteht aus einer Reihe von Conv2D Schichten, welche mit LeakyReLU aktiviert werden. Hierdurch verringern sich Höhe und Breite immer weiter (erkennbar an den beiden mittleren Werten in der Output Shape). Am Ende werden die restlichen Neuronen durch die Global Max-Pooling- und die Dense-Schicht auf ein Neuron heruntergebrochen.

Der Generator besteht wiederum aus einer Reihe von Conv2DTranspose Schichten mit LeakyReLU Aktivierung. Somit werden die Höhe und Breite des Bildes immer weiter aufgefächert. Am Ende wird eine Conv2D Schicht genutzt, um die überschüssigen Kanäle auf die richtige Anzahl zu bringen.

2.3.2 Neurale Netzwerk Schichten

Generelle Schichten

Eine normale, dicht vernetzte NN-Schicht wird in Keras durch eine sogenannte **Dense** Schicht dargestellt. Hierbei wird das Skalarprodukt zwischen dem Eingabetensor und dem Kern der Gewichtsmatrix in der dichten Schicht gebildet.

Außerdem gibt es noch eine Schicht, die die Eingaben in die angegebene Form konvertiert. Diese wird als **Reshape** Layer dargestellt und zum Beispiel zur Initialisierung der Bild Dimensionen genutzt.

Convolutional Schichten

Conv2D ist eine 2D-Faltungsschicht (z. B. räumliche Faltung über Bilder). Diese Schicht erzeugt einen Faltungs-Kernel, der mit der Schicht-Eingabe gefaltet wird, um einen Tensor von Ausgaben zu erzeugen. Hiermit wird beim Discriminator das Bild auf eine immer kleinere Dimension gefaltet. [6](S.10)

Ein Convolutional Layer besteht aus einer Gruppe von Filtern oder "Kernels", die auf die Eingabe angewendet werden. Jeder Filter besteht aus einer kleinen Matrix von Gewichten, die während des Trainings angepasst werden, um spezifische Merkmale in den Daten zu erkennen.

Während des Vorwärtsdurchlaufs durch ein CNN wird jeder Filter über das Eingabe-Feature-Map (ein dreidimensionales Array, das die Eingabe darstellt) gefaltet oder "konvolutioniert", um ein neues Feature-Map-Array zu erzeugen. Jeder Wert in der Ausgabe-Feature-Map wird berechnet, indem die Gewichte des Filters mit den Werten des Eingabe-Feature-Maps multipliziert und anschließend summiert werden. Das Ergebnis wird in der entsprechenden Position in der Ausgabe-Feature-Map platziert.

Die Filter in einem Convolutional Layer haben normalerweise eine kleine räumliche Größe (z.B. 3x3 oder 5x5), können aber eine große Anzahl von Kanälen haben. Die Anzahl der Kanäle entspricht normalerweise der Anzahl der Filter in einem Layer. Wenn zum Beispiel ein

Convolutional Layer mit 32 Filtern definiert wird, hat die Ausgabe-Feature-Map in der Regel auch 32 Kanäle.

Conv2DTranspose ist eine transponierte Faltungsschicht. Der Bedarf an transponierten Faltungen ergibt sich im Allgemeinen aus dem Wunsch, eine Transformation zu verwenden, die in die entgegengesetzte Richtung einer normalen Faltung geht. D.h. ausgehend von einem Tensor, der die Form des Ausgangs einer bestimmten Faltung hat, zu einem Tensor, der die Form des Eingangs hat, wobei ein Konnektivitätsmuster beibehalten wird, das mit der genannten Faltung kompatibel ist. Dadurch wird beim Generator der zufällige Eingangsvektor weiter aufgefächert.

Up-Sampling Schichten

Ein Upsampling-Layer[1] wird genutzt, um die räumliche Auflösung der Eingangsdaten zu erhöhen, indem Node-Inhalte (Datenpunkte) repliziert oder neue hinzugefügt werden. Hierdurch kann das Netz mehr Details in den Daten erfassen, wodurch wiederum die Genauigkeit der Ausgabe erhöht wird. Es gibt viele unterschiedliche Typen, wie zum Beispiel: nearest-neighbor-, linear-, bilinear-, bicubic-, trilinear upsampling. Jeder Typ hat einen anderen Algorithmus zur Skalierung der Eingangsdaten.

Pooling Schichten

GlobalMaxPooling2D: Globale Max-Pooling-Operation für Raumdaten. Der 2D-Global-Max-Pooling-Block führt genau dieselbe Operation wie der 2D-Max-Pooling-Block durch, mit dem Unterschied, dass die Pool-Größe (d. h. horizontaler Pooling-Faktor x vertikaler Pooling-Faktor) die Größe der gesamten Eingabe des Blocks ist, d. h. er berechnet einen einzelnen Max-Wert für jeden der Eingabekanäle. Eingabe: Der 2D-Global-Max-Pooling-Block nimmt einen Tensor der Größe (Eingangsbreite) x (Eingangshöhe) x (Eingangskanäle) und berechnet den Maximalwert aller Werte über die gesamte (Eingangsbreite) x (Eingangshöhe) Matrix für jeden der (Eingangskanäle). Ausgabe: Die Ausgabe ist ein 1-dimensionaler Tensor

[1] https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html

der Größe (Eingabekanäle). Hiermit wird beim Discriminator das schon gefaltete Bild auf ein Neuron heruntergebrochen. [6](S.18)

Aktivierungsschichten

Hierbei handelt es sich um Schichten, die nur eine Aktivierungsfunktion ausführen. Als eigenständige Schicht wurde nur LeakyReLU genutzt. LeakyReLU ist die Leaky-Version einer gleichgerichteten Lineareinheit. Sie ermöglicht eine kleine Steigung, wenn die Unit nicht aktiv ist.

Aktivierungsfunktionen

Eine Aktivierungsfunktion ist eine nicht lineare Funktion, die als Parameter die Summe der gewichteten Outputs der vorherigen Units enthält.

ReLU Funktion

Die ReLU Funktion wendet eine gleichgerichtete lineare Unit-Aktivierungsfunktion an. ReLU bildet das Maximum aus dem Eingabetensor und 0. [6](S.27)

$ReLU(x) = max(x, 0)$

Leaky ReLU

Neben der Standard-ReLU-Aktivierung gibt es noch die Leaky-ReLU-Aktivierung. Hier werden negative Werte mit einem Faktor (α) mit einbezogen. Dieser Faktor ist Prozentwert (zwischen 0 und 1) und meist klein. Dadurch wird dem Vanishing Gradient Problem vorgebeugt. Dazu mehr in Kapitel 4.

LeakyReLU(x) = x, wenn x > 0

LeakyReLU(x) = α * x, sonst

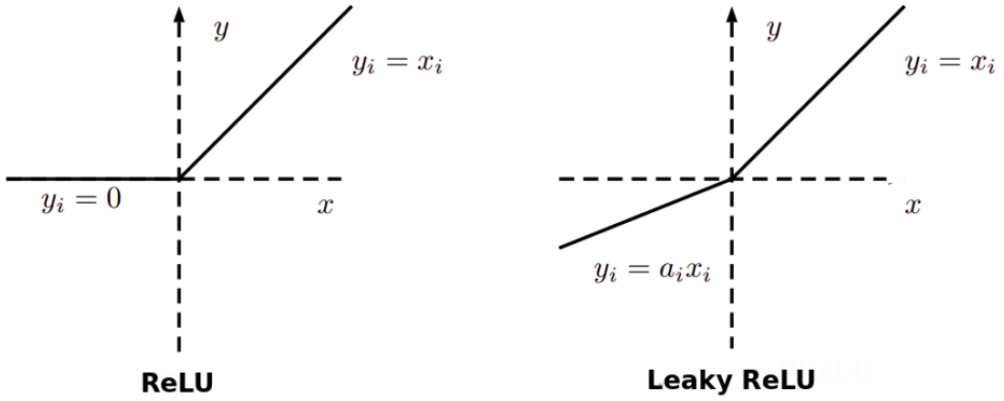

Abbildung 7 Vergleich zwischen ReLU und Leaky ReLU.

Sigmoid Funktion

Wendet die sigmoide Aktivierungsfunktion an. Für kleine Werte (kleiner als -5) liefert Sigmoid einen Wert nahe Null, und für große Werte (größer als 5) liegt das Ergebnis der Funktion nahe 1. Sigmoid entspricht einem 2-Element-Softmax, wobei das zweite Element als Null angenommen wird. Die Sigmoid-Funktion gibt immer einen Wert zwischen 0 und 1 zurück.

sigmoid(x) = 1 / (1 + exp(-x))

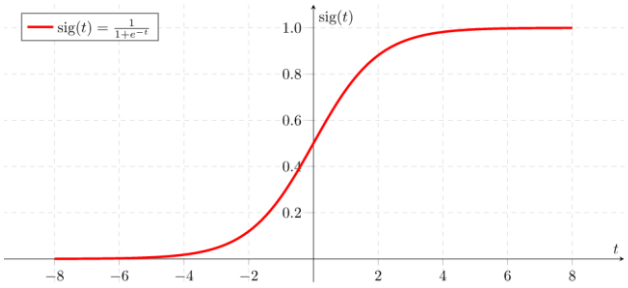

Abbildung 8 Graphische Darstellung der Sigmoid Funktion entnommen aus [2]

Tanh Funktion

Hyperbolische Tangens-Aktivierungsfunktion.

tanh(x) = sinh(x)/cosh(x) = ((exp(x) - exp(-x)) / (exp(x) + exp(-x)))

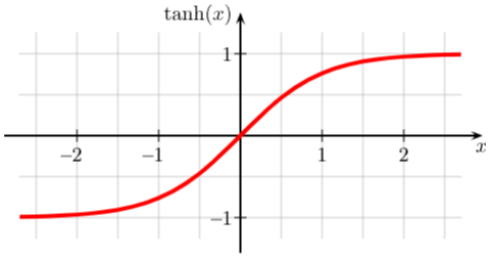

Abbildung 9 Graphische Darstellung der Tanh Funktion entnommen aus [3]

Tanh und Sigmoid werden beim Discriminator für die letzte Schicht genutzt. Außerdem wurden damit die Werte der Farbkanäle von den Bildern an den Wertebereich der jeweiligen Funktion angepasst.

[2] https://de.wikipedia.org/wiki/Sigmoidfunktion

[3] https://de.wikipedia.org/wiki/Tangens_hyperbolicus_und_Kotangens_hyperbolicus

2.4 Bekannte Probleme beim Trainieren eines GANs

2.4.1 Vanishing Gradients

Wie in den Aktivierungsfunktionen beschrieben, wird LeakyReLU als Aktivierungsfunktion zwischen den Schichten genutzt, um das Vanishing-Gradient-Problem zu vermindern. Das Vanishing Gradient Problem tritt bei neuronalen Netzen mit mehreren Schichten auf, die als Aktivierungsfunktionen Sigmoid oder Tanh nutzen. Hierbei können die Gradienten schnell gegen Null gehen, was dazu führt, dass die ersten Schichten nur wesentlich langsamer trainiert/geupdated werden. Zwar wird mit LeakyReLU und ReLU das Vanashing-Gradient-Problem nicht komplett ausgehebelt, aber die Gradienten gehen jetzt nur noch mit einer geringen Wahrscheinlichkeit gegen Null. [6](S.16) [7]

Möglichkeiten dieses Problem zu lösen sind:

1. Temporäre Änderungen an der Loss-Funktion vom Generator: Anstatt log(1 – D(G(z))) zu minimieren, kann der Generator in den anfänglichen Iterationen log(D(G(z))) maximieren.
2. Wasserstein-Loss: Für diese Loss-Funktion muss das GAN Schema verändert werden. Der Discriminator unterscheidet nicht mehr nach generiert und echt, sondern gibt einfach höhere Werte für als echt wahrgenommene Daten zurück. Der Discriminator (hier jetzt Critic) unterscheidet nicht mehr, sondern bewertet. Durch diese Änderungen ist die Loss-Funktion des Generators nur noch D(G(z)), wodurch der Generator immer einen Anhaltspunkt hat, in welche Richtung er verbessert werden sollte. [2](S. 18)

2.4.2 Mode Collapse

Normalerweise sollte der Generator eine große Anzahl an unterschiedlichen Daten ausgeben. Doch wenn der Discriminator ein generiertes Datum als besonders echt ansieht, kann es dazu kommen, dass sich der Generator auf dieses Datum spezialisiert. Dieses Problem wird verstärkt, wenn der Discriminator auf einem lokalen Minimum festhängt. Dadurch kann er das

generierte Datum nicht erkennen und der Generator wird immer besser darin, den Discriminator mit eben diesem Typ von Datum zu täuschen. [2](S. 147ff)

Eine Möglichkeit dieses Problem zu lösen ist der Wasserstein-Loss: Durch den Wasserstein-Loss kann der Discriminator, ohne auf Vanishing Gradients Rücksicht zu nehmen, optimal trainieren. Was dazu führt, dass der Discriminator nicht auf einem lokalen Minimum feststecken bleibt. [7]

2.4.3 Failure to Converge

Ein weiteres Problem bei GANs ist die Instabilität der Angleichung von Discriminator und Generator. Hierbei wird die Entscheidung des Discriminators für den Generator nicht aussagekräftig. Je besser zum Beispiel der Generator wird, desto schwieriger wird es für den Discriminator einen Unterschied zu erkennen. Zu dem Zeitpunkt, in dem der Generator nur noch perfekte Imitationen erzeugt hat, kann der Discriminator nur noch zu 50% richtig liegen, da er nur noch raten kann. Somit wird die Aussage des Discriminators im Laufe des Trainings immer weniger zielgerichtet. Wenn jetzt über den Punkt perfekt generierter Daten hinweg trainiert wird, kann es dazu kommen, dass die Qualität der Daten abnimmt, da der Generator durch zufälliges Feedback trainiert wird. Dieses Problem kann auch auftreten, wenn der Discriminator zu schnell zu gut wird. In diesem Fall wird dem Generator immer gesagt, dass seine Ergebnisse unecht sind und dieser bekommt nur wenige hilfreiche Anhaltspunkte durch die er sich verbessern kann. [7] [8]

2.4.4 Zu große Bilder

Zu große Bilder erzeugen eine hohe Instabilität in GANs. [9] Bilder, die größer als 100x100 Pixel sind, können schon zu Problemen beim Trainieren führen.

2.5 Progressive GANs

Eine Lösung, um stabile Modells für größere Bilder zu trainieren, ist Progressive GAN (auch PGGAN oder der Progressive growing GAN) Ansatz. Hierbei wird stufenweise die Anzahl der Layer während des Trainingsprozesses erhöht [9]. Ein Progressive GAN nutzt einen Generator und Diskriminator Modell mit derselben Struktur eines normalen GANs, hierbei wer-

den aber nicht alle Schichten von Anfang an trainiert. Während des Trainings werden neue Blöcke von Convolutional Layers nach und nach zum Generator und Diskriminator hinzugefügt. [2](S. 19)

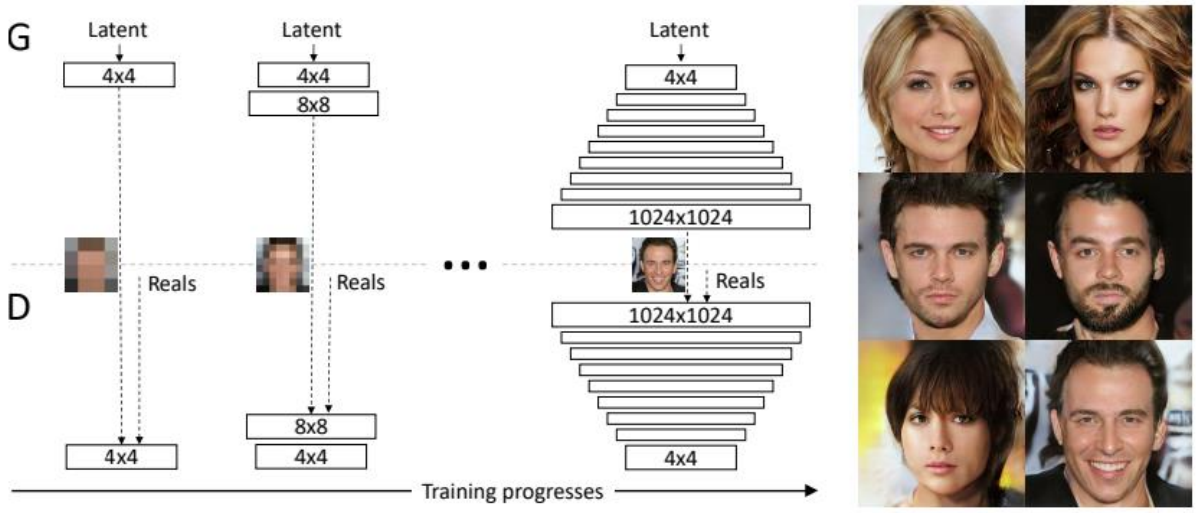

Abbildung 10 Schematische Darstellung des Trainings eines Progressive GANs, entnommen aus [9]

Das Einbinden eines neuen Blocks erfolgt durch die Verwendung einer Skip-Verbindung. Hierfür wird der neue Block mit dem Eingang des Diskriminators/dem Ausgang des Generators verbunden und eine Gewichtung zum vorhandenen Eingangs- oder Ausgangslayer hinzugefügt. Die Gewichtung (Parameter Alpha/a) kontrolliert den Einfluss des neuen Blocks. Diese Gewichtung startet zuerst auf Null oder einer sehr kleinen Zahl und wird linear über Trainingsiterationen zu 1,0 erhöht. In Abbildung 11 wird ein Generator und ein Diskriminator, die ein 16x16 Bild annehmen/ausgeben, mit Hilfe einer Skip-Verbindung auf eine Größe von 32x32 erweitert. [9]

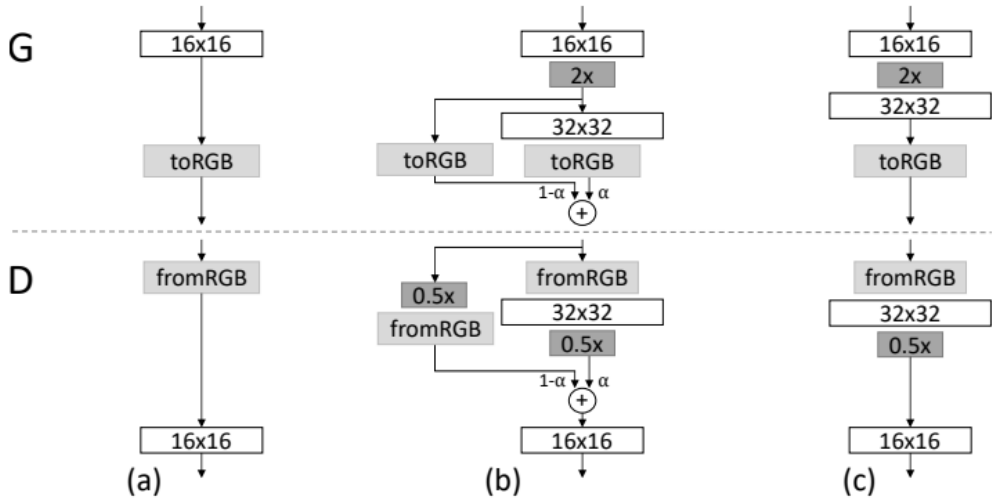

Abbildung 11 Schematische Darstellung der Skip-Verbindung, entnommen aus [9]

2.6 StyleGAN

Die meisten GANs nutzen direkt den Latent Vektor um das Bild zu generieren. Hierdurch ist die Kontrolle über die Bildgeneration nicht hoch. StyleGANs (entwickelt von NVIDIA in 2018) versuchen dieses Problem zu adressieren, indem Aspekte der Bildgeneration in Style und Struktur aufgetrennt werden. Ein Generator eines StyleGANs nutzt als Input den Latent Vektor nur noch implizit, stattdessen werden Style Vektoren (w), welche aus unterschiedlichen Latent Vektoren generiert werden können, genutzt. Der ursprüngliche Vektor wird mit Hilfe eines Mapping Networks zu einem Style Vektor gemappt. Wird nun nur ein Latent Vektor genutzt, wird dieser den generellen Stil des Bildes beschreiben. Style Vektoren hingegen können, je nach dem wann sie genutzt werden, unterschiedliche Teile des Bildes verändern. Nach dem Training geben Style Vektoren eine wesentlich höhere Kontrolle über generierte Bilder. [5] [2](S. 19)

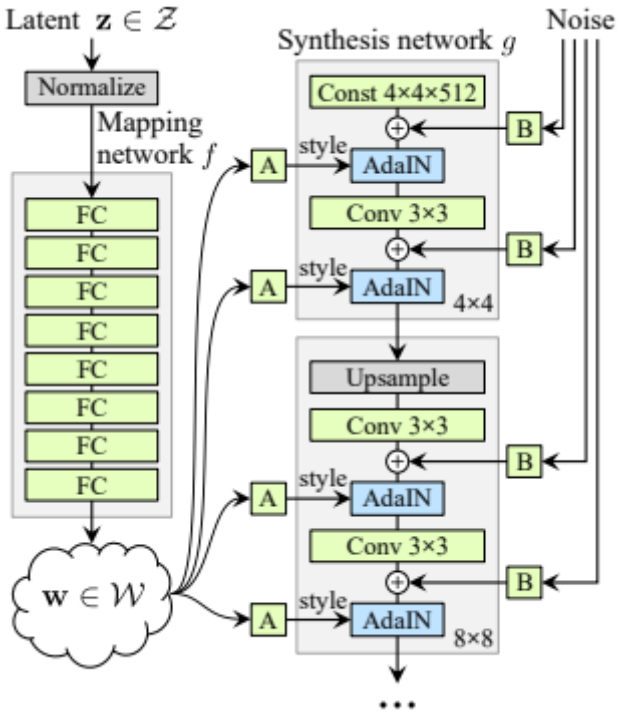

Abbildung 12 Schematische Darstellung eines StyleGAN Generators, entnommen aus [5]

2.6.1 Style-based Generator

Normalerweise wird der Latent Vektor dem Generator direkt in den Input-Layer beigeführt. In dem Style basierten Ansatz wird die Input-Schicht entfernt und durch eine gelernte Konstante ersetzt. Es wird der Latent Code z mit Hilfe eines nicht linearen Mapping-Netzwerks auf einen sogenannten Style Vektor w abgebildet. Dieser Vektor wird dann per adaptive instance normalization (AdaIN) an jedem Convolution-Layer des Generators eingespeist. Nach jeder Faltung wird dann Gaußsches-Rauschen addiert. In Abbildung 12 steht A für erlernte affine Transformation, welche w für AdaIN vorbereitet. B hingegen wendet erlernte Skalierungs-Parameter pro Kanal auf das Rauschen an.

Ein typisches Style-Mapping-Netzwerk besteht aus mehreren fully connected/ dense Schichten, welche den Latent Vektor in einem intermedialen Latent Space und somit in einen Style Vektor transformiert. Hierbei ergeben sich mehrere Vorteile: Es ermöglicht zum einen ein genaueres Einstellen der Attribute eines Bildes für Nutzer. Des Weiteren sorgt das Mapping-

Netzwerk für einen wesentlich linearen Raum im Gegensatz zum Latent Space, bei dem es sich um einen sehr verwobenen (entagled) Raum handeln kann. So können zum Beispiel Features, die an den Endpunkten eines Vektors im intermedialen Latent Space nicht vorkommen, nicht in der Mitte des Interpolationweges auftreten. [5]

2.6.2 AdaIN (adaptive instance normalization)

AdaIN ist eine Normalisierungstechnik, welche den Mittelwert und die Varianz des Feature-Inhaltes des Netzes mit dem der Style-Features angleicht. Im Gegensatz zur ursprünglichen Normalisierungstechnik berechnet AdaIN die Normalisierungsparameter dynamisch basierend auf dem Style Input und dem eigentlichen Inhalt des Netzes. Es wird also die Stilinformation auf ein Inhaltsbild angewandt, indem AdaIN das normalisierte Inhaltsbild mit dem Mittelwert und der Varianz des Stilbildes skaliert und verschiebt. Dadurch wird der Style von w an den Feature Inhalt übertragen, während der eigentliche Inhalt erhalten bleibt.

$$\mathrm{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

Abbildung 13 Abbildung der AdaIN Funktion, entnommen aus [10]

Hier repräsentiert x das Inhaltsbild und y das Stilbild. Das Ergebnis der AdaIN-Funktion ist ein neues Bild, das den Inhalt von x mit dem Stil von y kombiniert.

Die Funktion μ berechnet den Mittelwert des Eingabetensors entlang jedem Kanal, und σ berechnet die Standardabweichung. Die AdaIN-Formel wendet Normalisierung auf das Inhaltsbild x an, indem es seinen Mittelwert abzieht und durch seine Standardabweichung teilt. Anschließend skaliert sie das normalisierte Inhaltsbild mit der Standardabweichung des Stilbilds y und addiert den Mittelwert des Stilbilds y. Auf diese Weise hat das resultierende Bild denselben Mittelwert und dieselbe Standardabweichung wie das Stilbild y, aber sein Inhalt ähnelt dem Inhaltsbild x. [5] [10]

2.6.3 Style Vector

Im StyleGAN wird der intermediate Latent Space W verwendet, um den Generator durch adaptive Instanznormalisierung (AdaIN) in jeder Faltungsschicht zu steuern. W wird auch als "Style Raum" bezeichnet und die Vektoren in diesem Raum oft als "Style Vektoren" benannt. Diese Style Vektoren werden während des Trainings erlernt und erfassen die übergeordneten Attribute der generierten Bilder. Durch Modifikation der Style Vektoren kann man das Aussehen der generierten Bilder kontrollieren. Die Style Vektoren können als kontinuierliche Darstellung der übergeordneten Attribute der generierten Bilder betrachtet werden, was eine fein abgestimmte Steuerung der generierten Bilder ermöglicht. [5]

2.6.4 Mixing Regularization

Die Grundidee hinter Mixing Regularization besteht darin, dem Generator während des Trainings mehrere verschiedene Latent Vektoren zu geben und ihn zu zwingen zwischen ihnen zu "mischen", um verschiedene Ausgaben zu generieren. Das bedeutet, dass der Generator nicht nur einen einzigen Latent Vektor verwendet, um jedes Bild zu generieren, sondern eine Mischung aus mehreren Vektoren, um ein breiteres Spektrum an Ausgaben zu erzeugen. [5]

Durch diese Technik wird der Generator gezwungen, ein breiteres Spektrum an Ausgaben zu erzeugen und verhindert so, dass er in eine Mode-Collapse-Situation gerät. Darüber hinaus hilft Mixing Regularization auch dabei, die Diversität der generierten Bilder zu erhöhen, was die Qualität dieser Bilder weiter verbessert. [5]

2.7 Latent space exploration

Latent Space Exploration ist eine Technik, welche bei GAN’s genutzt wird, um das Verständnis des zugrunde liegenden Merkmalsraumes (Latent Space) zu verbessern. Indem man die Werte eines Latent Vektors manipuliert, kann man das Ergebnis eines Generators ändern. Die Hauptaufgabe der Latent Space Exploration ist das Mappen von generierten Features auf eine Position im Latent Space. Dies erlaubt eine Interpolation zwischen latent Vektoren, um somit Übergänge zwischen zwei Vektoren zu erzeugen und syntaktisch bedeutende Latent

Directions (Richtungen im Latent Space) zu erkennen. Dies ermöglicht eine höhere Kontrolle über die Inhalte, die generiert werden.

Um den Latent Space zu erkunden, werden in dieser Arbeit neben den in Kapitel 3 beschrieben Ansätzen noch unterstützende Verfahren genutzt. Hierunter fallen die Support Vector Machine (SVM) und die Hauptkomponentenanalyse (PCA).

2.7.1 Support Vector Machine (SVM)

Eine Support Vector Machine (SVM) ist eine überwachte Lernmethode. Das Hauptziel der SVM besteht darin, eine Hyperebene (hyperplane) im n-dimensionalen Raum zu finden, die die Datenpunkte in verschiedenen Klassen so gut wie möglich trennt. Bei linear separierbaren Daten handelt es sich um Datenpunkte, die in einem Merkmalsraum so angeordnet sind, dass eine Hyperebene die beiden Klassen vollständig voneinander trennen kann.

Wenn ein neuer Datenpunkt eingegeben wird, berechnet die SVM die Entfernung des Punktes zur Hyperebene. Je nachdem, auf welcher Seite des Trennungsraums der Punkt liegt, wird er einer der beiden Klassen zugeordnet.

Bei linear separierbaren Daten kann eine SVM die perfekte Trennung erzielen. In der Praxis sind die Daten jedoch oft nicht perfekt linear separierbar. In solchen Fällen verwendet die SVM eine Technik namens Kernel-Trick, um die Daten in einen höherdimensionalen Raum abzubilden, in dem sie linear separierbar sein können. [11] (Seite 417 ff.)

Support Vector Machines werden genutzt, um eine Hyperebene zwischen zwei Gruppen aus Latent Vektoren zu ziehen. Diese Hyperebene kann nun genutzt werden, um einen Vektor und somit eine Richtung zwischen den zwei Gruppen aus Latent Vektoren zu bestimmen. Auf diese Weise können SVMs verwendet werden, um semantisch interessante Richtungen im Latent Space zu finden, wenn hier zwei Gruppen von Daten mit möglichst gegensätzlichen Attributen gegeben sind.

2.7.2 Principal Component Analysis (PCA)

Die Hauptkomponentenanalyse (Principal Component Analysis, PCA) ist eine statistische Technik, die verwendet wird, um Muster und Beziehungen in einem Datensatz zu identifizieren. Es ist ein Typ des unüberwachtem maschinellen Lernens, das zur Reduzierung der Dimensionalität eines Datensatzes verwendet werden kann.

PCA erlaubt das Finden der Richtungen in den Daten, die die meisten Variationen beinhalten. Diese Richtungen werden als Hauptkomponenten bezeichnet und repräsentieren lineare Kombinationen der ursprünglichen Merkmale im Datensatz. Die erste Hauptkomponente erfasst die meisten Variationen in den Daten, die zweite Hauptkomponente erfasst die zweitmeisten Variationen, usw... Jede Hauptkomponente ist orthogonal zu der Vorherigen.

Die PCA beginnt damit, die Kovarianzmatrix des Datensatzes zu berechnen. Die Kovarianzmatrix misst das Ausmaß wie zwei Variablen linear zusammenhängen. Als Nächstes werden die Eigenvektoren und Eigenwerte der Kovarianzmatrix berechnet. Die Eigenvektoren repräsentieren die Hauptkomponenten und die Eigenwerte repräsentieren die Menge der Variationen, die von jeder Hauptkomponente erfasst werden.

Sobald die Hauptkomponenten identifiziert sind, können sie verwenden werden, um die Dimensionalität des Datensatzes zu reduzieren, indem die Daten auf die Hauptkomponenten projiziert werden. Der resultierende Datensatz hat weniger Dimensionen, erfasst jedoch immer noch einen Großteil der Variationen der ursprünglichen Daten. [12] [11] (Seite 547 ff.)

Diese Hauptkomponenten können wiederum semantisch interessante Richtungen in einem Latent Space darstellen.

3 Konzept und Methoden

Im folgenden Kapitel wird das Konzept der Arbeit und die genutzten Methoden/Technologien vorgestellt. Darunter fällt die Beschreibung des Datensatzes und dessen Eigenschaften und Herausforderungen, die er mit sich bringt und wie versucht wird, diese Herausforderungen zu adressieren. Darüber hinaus werden sowohl zwei unbeaufsichtigte Verfahren als auch zwei manuelle Verfahren zur Suche von Richtungen im Latent Space vorgestellt. Die manuellen Verfahren wurden hierbei von den unbeaufsichtigten Verfahren abgeleitet.

3.1 Anforderungen

Um Bilder zielgerichtet durch manuelle Veränderungen des Latent Vektors erzeugen zu können und den Latent Space eines StyleGANs nach semantisch interessanten Richtungen zu untersuchen, müssen folgende Punkte beachtet werden:

- Datensatz: Es muss ein passender Datensatz gefunden werden, an dem ein StyleGAN trainiert wird. Dieser Datensatz sollte nicht zu groß sein (< 100.000 Bilder) und nicht zu große Bilder enthalten (16px-100px). Grund hierfür ist eine technisch und zeitlich sehr aufwändige Trainingsphase des StyleGANs.
- StyleGAN: Es ist eine Version von StyleGAN zu wählen, welche selbst noch mit kleinen Datensätzen arbeiten kann.
- Latent Space Projektion: Zur zielführenden Nutzung von Richtungen im Latent Space sind die Bilder auf eben diesen Latent Space zu projizieren.
- Suche von Richtungen im Latent Space:

  - Manuell: Es sollte möglich sein, die Richtungsvektoren manuell zu finden, zu prüfen und zu optimieren.
  - Unbeaufsichtigt: Es sollen die Verfahren **Unsupervised Discovery of Interpretable Directions in the GAN Latent Space** (kurz: GANLatentDiscovery) und **CLIP2StyleGAN** zur Suche von Richtungen im Latent Space angewandt werden.
- Des Weiteren sollten Richtungen im Latent Space gespeichert werden, damit sie auf andere Latent Vektoren/Bilder angewandt werden können.

3.2 Datensatz

Der genutzte Datensatz ist eine Reihe von Pixel-Art Icons, die mir von befreundeten Künstlern (https://pixerelia.itch.io/, https://iampixelle.itch.io/) für meine Arbeit zur Verfügung gestellt wurden. Diese Bilder haben unterschiedliche Größen, wobei der Großteil bei 16x16 Pixel liegt. Der Datensatz enthält eine Vielzahl von visuellen Konzepten, wie zum Beispiel Icons zu Fischen, Werkzeugen, Pflanzen und Ringen, aber auch abstrakte Formen wie Icons für Sprechblasen. Insgesamt handelt es sich um einen Datensatz mit 2485 Bildern.

Abbildung 14: Originalbilder des Datensatzes zur Betrachtung skaliert

Der Datensatz besitzt keine Beschriftungen oder Klassifizierungen im Vorfeld. Er wurde deshalb gewählt, weil er mehrere interessante Charakteristiken aufweist: Es handelt sich um einen sehr diversen Datensatz. Es existieren viele Farbkonzepte und Formen, die sich stark voneinander unterscheiden. Trotzdem besteht der Datensatz aus Bildern, die alle einem künstlerischen Stil folgen. Gleichzeitig ist er mit 2485 Bildern klein und dadurch überschaubarer. Eine solche Anzahl an Bildern ist interessant, da der Datensatz auch aus der eigenen Feder stammen könnte.

3.3 Geeignete KI-Modelle im Vergleich

In der Auswahl bieten sich StyleGAN, StyleGAN2, StyleGAN2-ADA bzw. StyleGAN3 an. Deren Anforderungen an diese Arbeit sind in folgender Tabelle aufgelistet:

Anforderungen	StyleGAN	StyleGAN2	StyleGAN2-ADA	StyleGAN3
Trainierbar auf wenige Bilder	nein	nein	ja	ja
Trainierbar auf kleine Bilder	ja	ja	ja	ja
Geeignet für die Vorauswahl der Suchverfahren	nein	ja	ja	nein
Bisherige persönliche Erfahrungen	ja	ja	nein	nein

Tabelle 1: Anforderugnstabelle der KI-Modelle

Als generatives Modell fiel die Wahl auf den StyleGAN2-Ada, da dieser mit kleineren Datensätzen (< 30.000 Bilder) arbeiten kann. Gleichzeitig kann das Modell zu einem einfachen StyleGAN2 konvertiert werden. Dies erleichtert die Handhabung für die unten beschriebenen Verfahren. Außerdem wird es durch die ursprüngliche StyleGAN2 Architektur ermöglicht, direkt mit den originalen Implementierungen von Clip2StyleGAN und GANLatentDiscovery zu arbeiten. StyleGAN2-Ada erlaubt außerdem die Möglichkeit, mit einem Conditional Generator zu arbeiten. Da der Datensatz aber nicht klassifiziert wurde, wird diese Art von Generator nicht genutzt.

3.4 StyleGAN2

Mit dem StyleGAN2 wurde die Qualität der Bilder im Vergleich zum StyleGAN noch einmal erhöht. Nach Überarbeitung der Architektur des Generators von StyleGAN zu StyleGAN2

wurde unter anderem die erste Schicht entfernt und die gelernte Konstante c_1 direkt, als Input für den ersten Style Block, genutzt. [13]

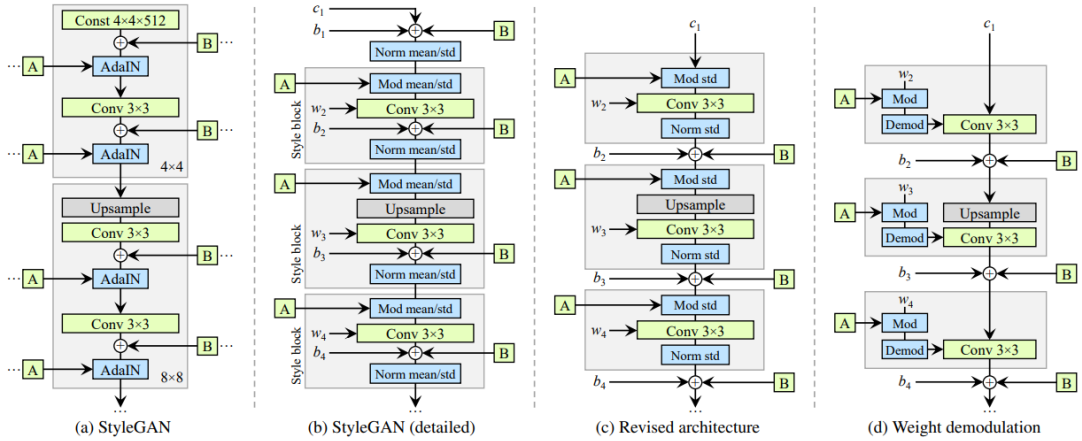

Abbildung 15: Schematische Darstellung der Entwicklung von StyleGAN zu StyleGAN2 [13]

Im StyleGAN2 werden ähnliche Schichten und Aktivierungsfunktionen wie im StyleGAN verwendet. In Abbildung 15 zeigen (a) und (b) die ursprüngliche StyleGAN Architektur und (d) die neue StyleGAN2 Architektur. Hierbei zeigt (c) einen Zwischenschritt in der Entwicklung von StyleGAN2 auf. Der erste sichtbare Unterschied ist das Entfernen der adaptive instance normalization (AdaIN). Stattdessen wird nur die Standardabweichung genutzt. Des Weiteren wird die Addition des zufälligen Rauschens aus dem Style-Block verschoben. Im nächsten Schritt (d) wird nun die Standardabweichung und der Convolutional Layer darunter zu einer Gewichtsmodulation zusammengeführt [13].

$$w'_{ijk} = s_i \cdot w_{ijk},$$

Abbildung 16: Abbildung der Gewichtsmodulation, entnommen aus [13]

In Abbildung 16 werden w und w‘ jeweils als die originalen und die modulierten Gewichte dargestellt. S_i ist die Größe der i-ten Eingabe-Feature-Map. J und k stellen die räumlichen Indizes der Ausgabe-Feature-Map dar.

Außerdem wird die Normalisierung nach der Faltungsschicht zu einer sogenannten „Gewichts-Demodulation“ Operation.

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} {w'_{ijk}}^2 + \epsilon},$$

Abbildung 17: Darstellung der Gewichts-Demodulation, entnommen aus [13]

Die Konstante ε ist ein kleiner Wert, der genutzt wird, um numerische Probleme zu umgehen. Mit der Gewichts-Demodulation werden Artefakte, die sich in den vorherigen Versionen der generierten Bilder bildeten, weitestgehend entfernt.

Das Ersetzen der AdaIN durch die Gewichts-Demodulation hat für eine bessere Qualität der Bilder gesorgt. Dadurch werden die von der AdaIN erzeugten tropfenförmigen Artefakte verhindert. Diese tropfenförmigen Artefakte sind beim StyleGAN ab 64x64 Pixel großen Bildern sichtbar geworden und wurden immer prominenter je größer das Bild wurde. [13]

Die letzte Änderung betrifft das Anwachsen des Generators und Diskriminators nach den Grundlagen des Progressive GAN. Es wurde sich gegen das Wachsen der Architektur entschieden. Stattdessen wurden für den Generator permanente Skip-Verbindungen verwendet und im Discriminator der Residual-Net-Aufbau genutzt. [13]

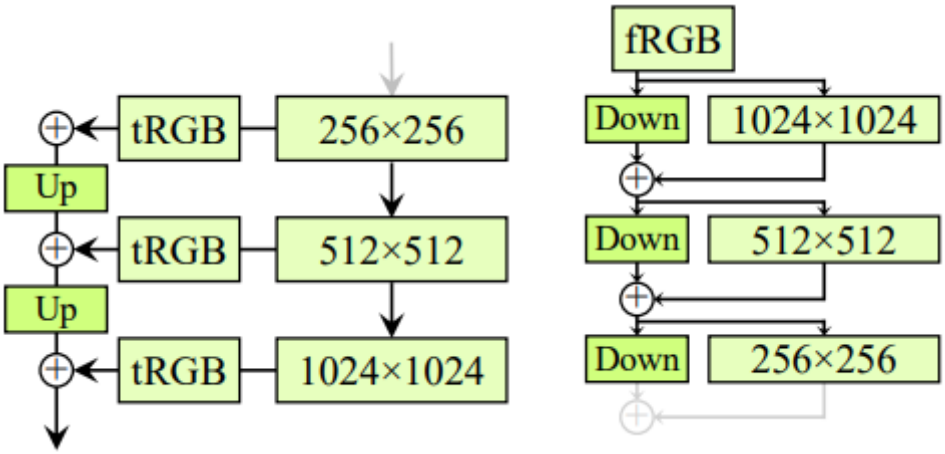

Abbildung 18: Rechts: Generator mit Skip-Verbindungen; Links: Residual Net Architektur für den Diskriminator, entnommen aus [13]

Skip-Verbindungen haben immer noch dieselben Vorteile wie das progressive growing, doch verhindern sie so genannte „Phase-Artefakte“. „Phase-Artefakte“ sind Bildstrukturen, die sich bei einem Latent Shift normalerweise kontinuierlich bewegen sollten, aber stattdessen erst unverändert bleiben und dann von einer zuvor präferierten Position zur nächsten springen.

Abbildung 19: Beispiel eines "Phase" Artefakts, welches in der präferierten Position bleibt, entnommen aus [13]

Die Anwendung der Residual Net Architektur auf die Diskriminator Architektur ähnelt jetzt mehr den typischen Bild-Klassifikator-Modellen und hat laut Karras in [13] zu besseren Ergebnissen geführt.

3.4.1 Preprocessing

Damit die Bilder zum Trainieren eines StyleGANs genutzt werden können, brauchen sie eine einheitliche Größe. Außerdem sollten sie keine Transparenzen enthalten und nur mit RGB-Werten arbeiten. Ein weiteres Problem ist schon durch erste Experimente mit der später genutzten Bild-Captioning KI „CLIP“, welche Grundlage für CLIP2StyleGAN ist, aufgefallen: Die Bilder dürfen nicht zu klein sein. 16x16 Pixel für ein Bild hat bei der Nutzung von CLIP zu schlechten Ergebnissen geführt. Somit sollten die Bilder auf 64x64 Pixel skaliert werden. Dies erhöht die Anzahl der Layer im StyleGAN und somit auch die Auswahl der Layer beim Style-Mixing. Grund, warum nicht direkt zu einem nativen 64x64 Pixel Datensatz gewechselt wurde, ist zum einen die Diversität des Datensatzes und zum anderen die sehr einfache Skalierbarkeit der Bilder. Des Weiteren bringt die Nutzung von Bildern, die für Videospiele ge-

schaffen wurden, einen prozeduralen Aspekt (in 6.2 beschrieben) mit sich. Hier könnte eine mögliche Weiterführung dieser Arbeit liegen.

3.4.2 StyleGAN2-Ada Training

StyleGAN2-Ada oder StyleGAN2 with adaptive discriminator augmentation ermöglicht es mit kleineren Datensätzen bessere Ergebnisse zu erzielen als bisher mit weniger als 30.000 Bildern möglich war. Das größte Problem bei vorherigen Modellen ist, dass der Diskriminator bei kleinen Datensätzen auf die Trainingsbilder overfitted (überangepasst wird) [14]. Das bedeutet, dass der Diskriminator zu schnell gut darin wird, die echten Bilder aus dem Trainingsdatensatz zu erkennen. Dadurch werden die Klassifizierungen, die der Diskriminator dem Generator gibt, immer weniger aussagekräftig. Das sorgt wiederum dafür, dass das Modell beim Trainieren nicht konvergiert, sondern ab einem bestimmten Punkt divergiert.

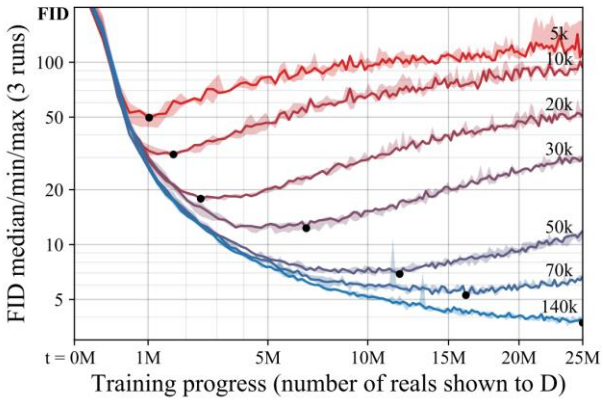

Abbildung 20: Graphische Darstellung des Trainingsprozesses eines StyleGAN2 anhand des Trainingsfortschrittes, der FID-Bildqualität und der Menge an Bildern im Trainingsdatensatz, entnommen aus [14]

Um dieses Problem zu lösen, wird der Datensatz in der Regel durch „dataset augmentation“ vergrößert. Doch diese Augmentationen (z.B. Rotation, Translation oder Verrauschen der Bilder) haben für schlechtere Ergebnisse gesorgt, die eben diese Augmentationen mit beinhalten. Dieser Prozess des Rekonstruierens der Augmentationen im Generator wird auch „leaking“ genannt. [14]

Beim StyleGAN2-Ada sieht der Diskriminator nahezu nur augmentierte Bilder. Der Generator kommt nicht in Kontakt mit den augmentierte Bildern, da diese erst nach dem Erstellen

eines Bildes angewandt werden. Dennoch können diese Augmentationen durch das Feedback des Diskriminators den Generator beeinflussen.

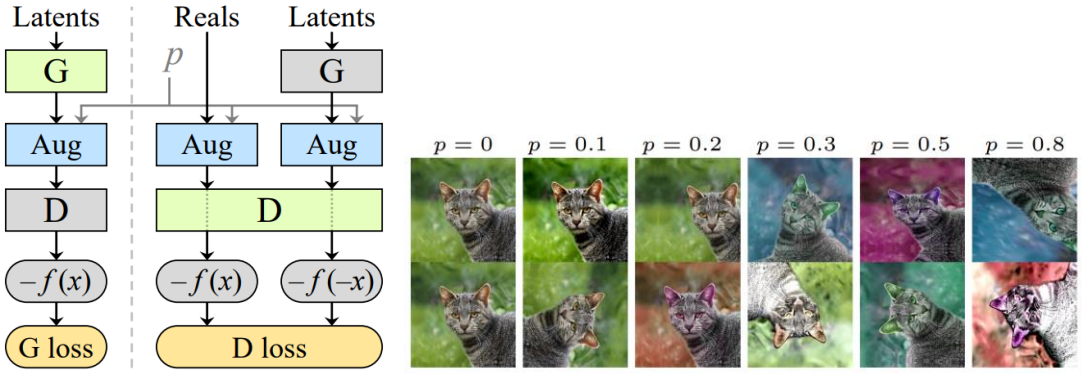

Abbildung 21: Links: Architektur des StyleGAN2-Ada; Rechts: Schematische Darstellung der Auswirkungen von p; Beide Bilder entnommen aus [14]

G ist hierbei der Generator und D der Diskriminator, p kontrolliert die Augmentationswahrscheinlichkeit. f(x) stellt die Funktion log(sigmoid(x)) dar. Um Leaking zu minimieren werden auch nur bestimmte Formen der Augmentation genutzt, wie z.B. 90° Rotationen, Farbänderungen und additiver Noise. Des Weiteren sollte p den Wert 0.8 nicht überschreiten. Außerdem wird die Stärke der Augmentationen dynamisch an die Stärke der Überanpassung gekoppelt. [14]

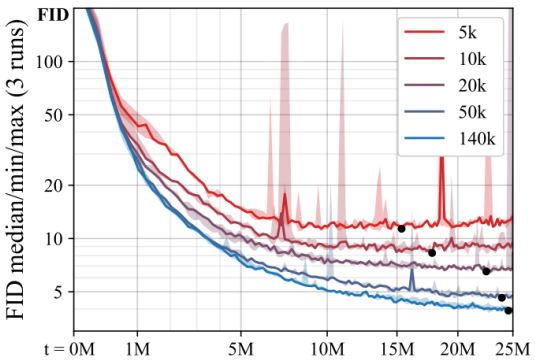

Abbildung 22: Graphische Darstellung des Trainingsprozesses eines StyleGAN2-Ada anhand des Trainingsfortschrittes, der FID-Bildqualität und der Menge an Bildern im Trainingsdatensatz, entnommen aus [14]

3.5 Latent Space Projektion

Ein wichtiger Bestandteil der Latent Space Exploration ist das Projizieren eines Bildes auf den Latent Space. Ziel hierbei ist es, ein Bild bestmöglich mit einem trainierten GAN nachzuempfinden. Wenn das GAN auf einem entsprechenden Datensatz trainiert wurde, sollte ein sehr ähnliches Bild mit einigen abweichenden Details generiert werden können. Diese Projektion ermöglicht dann in den nächsten Kapiteln das Erkennen von semantisch interessanten Richtungen im Latent Space.

Um ein Bild auf den latenten Raum eines vortrainierten StyleGANs zu projizieren, kann man einen optimierungsbasierten Ansatz namens Latent Optimization verwenden. Ziel dieses Ansatzes ist es, einen latenten Code z im latenten Raum des Generators zu finden, der ein Bild erzeugt, das dem Eingabebild ähnelt.

Das Ergebnis dieses Prozesses ist ein Latent Code z, der dem Eingabebild entspricht und verwendet werden kann, um neue Bilder zu erzeugen, die ähnliche visuelle Merkmale wie das Eingabebild teilen. Die Qualität des projizierten Latent Vektors hängt von der Qualität des vortrainierten StyleGANs, dem verwendeten Optimierungsalgorithmus und der Wahl der Verlustfunktion ab.

Um den Latent Vektor von echten Bildern zu erhalten, wird meistens die Latent Code Optimierung (mit der Nutzung des Gradientenabstiegs) unter Verwendung des Perzeptuellen Loss [15] genutzt. Es werden Merkmale (z.B. von einem vortrainierten Modell wie VGG) für das Referenz- und das generierte Bild extrahiert, die Distanz zwischen ihnen berechnet und der Latent Vektor optimiert. Die Initialisierung dieses Zielvektors ist bei dieser Optimierung ein sehr wichtiger Aspekt für Effizienz und Effektivität. In der Regel wird zur Initialisierung ein einzelner zufälliger Vektor gewählt. Man könnte aber auch eine zufällige Menge von N Beispielen generieren und das nächstliegende Bild als Initialisierungsvektor nutzen. [13]

Ablauf des Latent Vektor Optimierungsverfahrens:

Start mit einem vortrainierten StyleGAN-Generator und einem Eingabebild, das auf den latenten Raum projiziert werden soll.

1. Es wird ein zufälliger Rauschvektor z in den Generator geführt, welcher ein entsprechendes Bild generiert.
2. Der Verlust zwischen dem generierten Bild und dem Eingabebild wird nun berechnet. Der Loss kann eine Distanzmetrik wie der Perzeptuelle Verlust [15] sein.
3. Der Rauschvektor z wird, um den Verlust zwischen dem generierten Bild und dem Eingabebild unter Verwendung eines Optimierungsalgorithmus wie Gradientenabstieg zu minimieren, angepasst.
4. Schritt 2 und 3 werden wiederholt, bis der Verlust zwischen dem generierten Bild und dem Eingabebild auf ein zufriedenstellendes Niveau minimiert ist.

[16]

3.6 Manuelle Suche nach Richtungen im Latent Space

Die GANs ermöglichen es, einzelne Attribute des Datensatzes auf orthogonale Weise zu erkunden. Das bedeutet, dass andere Attribute nicht beeinflusst werden. Es kann zum Beispiel von einem Bild eines Hundes ausgegangen werden und mithilfe von gezielten Veränderungen des Latent Vektors in eine bestimmte Richtung ein Bild einer Katze mit gleichen Attributen der ursprünglichen Generation erzeugt werden.

Um Richtungen im Latent Space eines StyleGANs zu entdecken, werden in dieser Arbeit zwei Ansätze untersucht. Diese Ansätze lassen sich auch auf andere Arten von GANs übertragen, solange ein Latent Vektor in irgendeiner Form benötigt wird, um ein Bild zu generieren. Der erste Ansatz beschäftig sich damit, gezielt Bilder in den Latent Space zu projizieren, zwischen diesen zu interpolieren und damit möglicherweise interessante Rückschlüsse auf den Latent Space und spezifische Richtungen darin zu erkennen. Ein weiterer explorativer Ansatz versucht, durch das Ablaufen von mehreren zufälligen Richtungen gleichzeitig semantisch interessante Richtungen zu finden. Hierbei wird immer von demselben Bild/Punkt

im Latent Space ausgegangen. Beide Ansätze bilden eine grobe Grundlage für die später beleuchteten unbeaufsichtigten Verfahren zum Entdecken von Latent Richtungen.

Der erste Ansatz beginnt mit der Suche von zwei Klassen an Bildern, welche möglicherweise durch eine Richtung im Latent Space verbunden sein könnten. Nun werden passende Bilder als Extreme für diese Klassen gesucht, dessen Latent Vektoren helfen könnten, diese Richtung zu bilden. Diese Bilder können entweder aus einem Set zufällig generierter Bilder ausgewählt werden oder passende Bilder aus dem Datensatz sein, welche in Latent Space projiziert wurden. Es können aber auch passende Bilder aus dem Datensatz sein, die im Vorfeld bearbeitet wurden, um eher den Extremen zu entsprechen. Beide Bildersets müssen in den Latent Space übertragen werden (falls noch nicht geschehen) und werden zu jeweils einem Mean Vektor zusammengeführt. Der Vektor der Richtung wird nun durch einfache Arithmetik gebildet. Alternativ kann auch eine Hyperplane mit einer linearen SVM gebildet werden und anhand dessen Beschreibungsvektors die Richtung bestimmt werden. Im nächsten Schritt sollten mehrere Bilder entlang des Vektors gebildet werden, um einen Überblick zu erhalten, ob es sich tatsächlich um eine semantisch bedeutsame Veränderung handelt. Je mehr Bilder für die einzelnen Klassen gefunden werden, desto höher ist die Chance, dass die Richtung keine weiteren unerwünschten Transformationen enthält.

Der zweite Ansatz ist zufälliger, erlaubt aber eine genauere Untersuchung eines lokalen Bereiches im Latent Space. Hierbei wird ein Bild nach eigenem Interesse gewählt und dieses wird in den Latent Space projiziert. Nun werden zufällige, möglichst ungleiche Richtungen ausgesucht (z.B. die orthogonalen Einheitsvektoren des Latent Spaces). Entlang dieser Richtungen werden jetzt vom latenten Vektor des ursprünglichen Bildes ausgehend mehrere Bilder erzeugt.

Für beide Ansätze empfiehlt es sich, eine Animation aus den gesammelten Bildern der entsprechenden Richtung zu erstellen, um schnell einen genauen Überblick über die gefundene Richtung zu erhalten.

3.7 Unüberwachte Suche nach Richtungen im Latent Space

Der GAN Latent Space besitzt meist eine semantisch aussagekräftige Vektor-Arithmetik. Das heißt, es gibt Richtungen im Latent Space, die bei Veränderung entlang dieser Richtung, angewandt auf den Latent Vektor, eine für den Menschen verständliche Veränderung im generierten Bild erzeugt. Zum Beispiel könnte es eine Richtung innerhalb des Latent Spaces geben, welche die erzeugten Bilder abdunkelt oder bestimmte Formen im Bild verändert (Gesichtsausdruck von Personen). Um diese Richtungen zu finden, gibt es zwei Methoden: beaufsichtigte Verfahren (supervised) und unbeaufsichtigte Verfahren (unsupervised). Beaufsichtigte Verfahren benötigen meist eine Form der Klassifizierung der Daten durch Menschen und sind somit je nach Größe des originalen Datensatzes mehr oder weniger arbeitsaufwändig. Im Laufe dieser Arbeit werden zwei unsupervised Verfahren genauer beleuchtet.

Das erste Verfahren von Voynov in [17] (kurz: GANLatentDiscovery) beschrieben, basiert auf dem Optimieren von scheinbar zufällig gewählten Richtungen im Latent Space. Durch ein solches Vorgehen können Richtungen gefunden werden, die der Nutzer vielleicht nicht erwartet hätte, da nicht auf vorher klassifizierten Daten gearbeitet wird und somit keine Voreingenommenheit durch den Nutzer entsteht [17].

Das zweite Verfahren namens Clip2StyleGAN, in [18] beschrieben, basiert auf einem anfänglichen Klassifizieren der realen Daten mit Hilfe von CLIP (in 3.7.2 beschrieben) und einer Interpolation zwischen diesen in den Latent Space abgebildeten Daten [18].

Des Weiteren gibt es generative Modelle, bei denen der Latent Space von sich aus disentangled (entwirrt) ist. Eines dieser Modelle wird später im Kapitel Ausblick noch eingehender betrachtet.

3.7.1 GANLatentDiscovery

In diesem Kapitel wird das Verfahren **Unsupervised Discovery of Interpretable Directions in the GAN Latent Space** (kurz: GANLatentDiscovery) aus [17] beschrieben. Dieses Verfahren nutzt neben dem vortrainierten Generator eines StyleGANs noch eine Matrix, welche die Richtungsvektoren enthält und ein neuronales Netz namens Reconstructor.

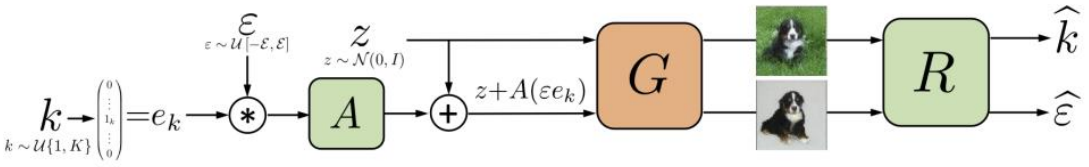

Abbildung 23: Schematische Darstellung des GANLatentDiscovery Systems, entnommen aus [17]

Um interpretierbare Richtungen im Latent Space zu erhalten, werden bei diesem Verfahren sowohl eine **Matrix A** als auch ein neuronales Netz namens **Reconstructor R** optimiert. In Abbildung 23 ist der genaue Ablauf dargestellt. Das Ziel des Algorithmus ist das Finden von Richtungen im Latent Space des vortrainierten Generators G. G bildet Daten vom Latent Space Z in den Bildraum ab. In diesem Verfahren wird G nicht weiter trainiert und dessen Parameter werden nicht verändert. Es gibt zwei Komponenten, die trainiert werden: Erstens die Matrix A und zweitens der Reconstructor R. Die Matrix A hat die Dimension $\mathbb{R}^{d \times K}$, d entspricht hierbei der Dimensionalität des Latent Spaces von G. Der Wert K hingegen entspricht der Anzahl der Richtungsvektoren, die in diesem Verfahren entdeckt werden sollen. Eine Trainingsprobe in diesem Verfahren besteht aus zwei latenten Vektoren, bei dem ein Vektor eine verschobene Version des anderen darstellt. Das heißt, es wird ein Vektor z gewählt, welcher in eine der Richtungen in A verschoben wird. Beide Vektoren – ursprünglich und verschoben – werden dem Generator G als Input gegeben und so im Bildraum abgebildet. Diese beiden Bilder werden nun in den Reconstructor R gegeben. Das Ziel von R ist es, die ursprüngliche Richtung als Index k für die Matrix A und dessen vorzeichenbehafteten Multiplikators ε zu bestimmen. R nimmt somit ein Bildpaar in Form (G(z), G(z + A(εe_k))) an.

Der Vektor A(εe_k) stellt hierbei die Richtung dar, um die z verschoben werden soll. e_k ist ein Einheitsvektor, welcher einer Achse von K folgt. e_k entscheidet, welche Spalte aus A beziehungsweise welche Richtung aus A genutzt werden soll. R bestimmt nun die Parameter $\hat{k}$ und $\hat{\varepsilon}$. $\hat{k}$ stellt die Vorhersage, welcher Index k ∈ {1, …, K} genutzt wird und $\hat{\varepsilon}$ die Vorhersage des Multiplikators dar. [17]

Zum Trainieren dieser zwei Komponenten wird die folgende Loss-Funktion miniert:

$$\min_{A,R} \mathop{\mathbb{E}}_{z,k,\varepsilon} L(A,R) = \min_{A,R} \mathop{\mathbb{E}}_{z,k,\varepsilon} \left[L_{cl}(k,\widehat{k}) + \lambda L_r(\varepsilon,\widehat{\varepsilon}) \right]$$

Abbildung 24: Loss-Funktion für Matrix A und Rekonstruktor R, entnommen aus [17]

Für L_{cl} wird die Cross-Entropy-Loss-Funktion und für L_r die Mean-Absolute-Error-Funktion[4] genutzt. Für die Gewichtung wird λ=0.25 empfohlen. Der Term L_r sorgt dafür, dass die entdeckten Richtungen einen fließenden Übergang haben und somit stetige, also nahtlos fortlaufende Transformationen darstellen. [17]

A und R werden gleichzeitig optimiert, sodass die Spalten in A Bildtransformationen enthalten, die einfach voneinander zu unterscheiden sind, um das Klassifikationsproblem von R zu erleichtern [17]. Wie später in Kapitel Ergebnisse dargestellt wird, entstehen hier für Menschen erkennbare Transformationen.

Durch das explizite Untersuchen des Latent Spaces statt des Style Spaces, ist dieser Ansatz nicht nur auf StyleGANs, sondern auch auf eine Reihe von GANs anwendbar. Zwar sind die gefundenen Richtungen nicht immer für den Menschen semantisch aussagekräftig, können es jedoch noch werden, wenn sie als Style Vektor nur zum Teil durch Style-Mixing in das Bild einfließen.

3.7.2 CLIP2StyleGAN

In diesem Kapitel wird das Verfahren **CLIP2StyleGAN** aus [18] beschrieben. Hierbei werden innerhalb von drei Schritten Richtungen im Latent Space gefunden, optimiert und gleichzeitig beschriftet.

CLIP2StyleGAN versucht Bilder mittels CLIP zu klassifizieren, indem diese in den CLIP-Space übersetzt werden. Anschließend wird zwischen den einzelnen Bildern interpoliert, um so Richtungen im Latent Space zu finden.

[4] https://de.wikipedia.org/wiki/Mittlerer_absoluter_Fehler

Hier wird die **Hauptkomponentenanalyse (PCA – Principal Component Analysis)** genutzt, um Bildergruppen zu bilden und geeignete Richtungen im StyleGAN Latent Space zu entdecken.

Das **Contrastive Language-Image Pre-training (CLIP)** Modell von OpenAI ist ein multimodales Modell, das Wissen über englischsprachige Konzepte mit semantischem Wissen über Bilder kombiniert [19]. In CLIP lassen sich sowohl Bilder als auch Texte mithilfe des CLIP-Text-Encoder und des CLIP-Bild-Encoders in einen gemeinsamen CLIP-Space überführen. Als Nächstes können anhand der räumlichen Nähe im CLIP-Space Bilder mit entsprechendem Text verknüpft werden. Dieses Modell findet vor allem Verwendung in einer neuen Generation von Text-zu-Bild-KIs wie Stable Diffusion (mehr hierzu in Kapitel 6.3 Ausblick).

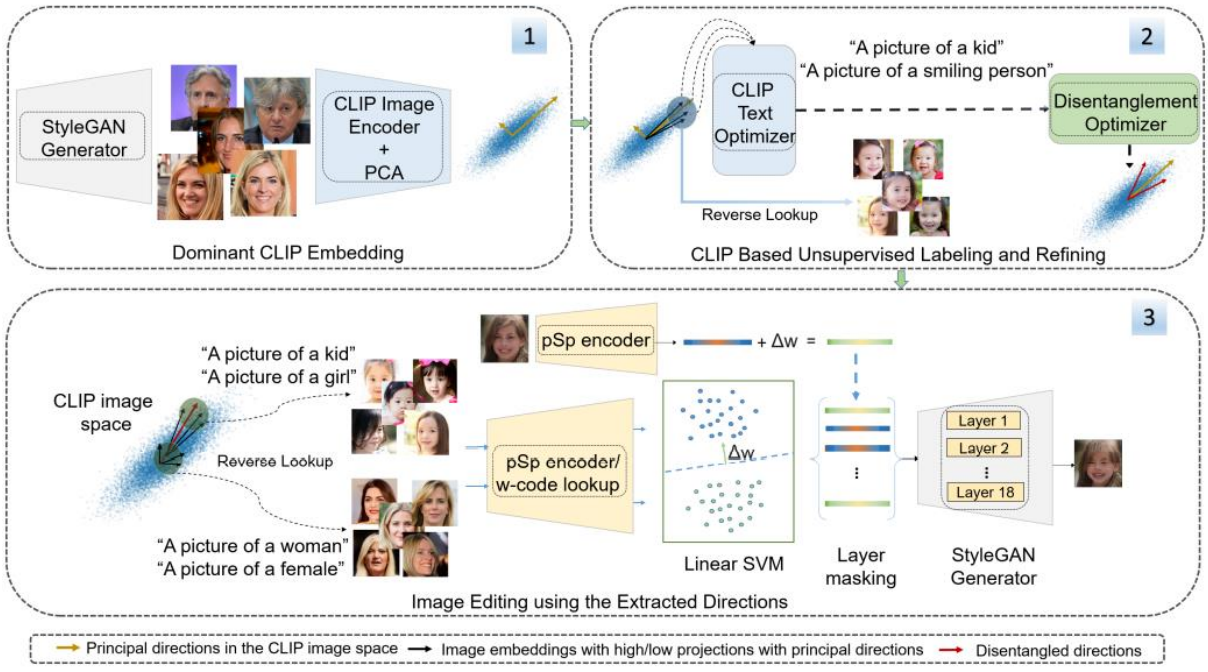

Abbildung 25: Schematische Darstellung des Ablaufes von Clip2StyleGAN, entnommen aus [18]

Das unüberwachte Entdecken von semantisch bedeutsamen Richtungen im Latent/Style Space mit CLIP2StyleGAN besteht aus drei Schritten: Als erstes werden mit Hilfe eines von in den CLIP-Bild-Space überführten Datensatzes semantisch interessante Richtungen berech-

net. Dieser Datensatz kann echte Bilder, generierte Bilder oder den originalen Datensatz, auf dem trainiert wurde, enthalten. Die Anzahl der Bilder kann hierbei beliebig gewählt werden. Im zweiten Schritt werden die gefundenen Richtungen disentangled und mit dem CLIP-Text-Encoder beschriftet. Im letzten Schritt werden die Richtungen in den StyleGAN Latent Space gemappt. [15] [18]

Schritt 1: Ein Satz aus n unterschiedlichen Bildern wird mit Hilfe eines vortrainierten CLIP-Bild-Encoders in den CLIP-Space überführt. Durch den immens großen Datensatz mit dem CLIP-Modelle trainiert werden, besitzen diese eine Vielzahl an visuellen Konzepten. Diese Konzepte sind im Zweifelsfall für einen GAN, welcher nur auf einen sehr spezifischen Datensatz trainiert wurde, nicht zutreffend. Zum Beispiel könnte CLIP den Stil eines Bildsatzes als wichtiger anerkennen als das eigentliche Subjekt des Bildes. So könnte ein Bild von CLIP als „pixel-art“ bezeichnet werden, wo ein Mensch die Klassifizierung „banana“ erkennt. Um einer gemeinsamen Richtung im CLIP-Space vorzubeugen, wird eben dieser mittlere CLIP Latent Space Vektor μ über alle codierten Bilder berechnet. Dieser Vektor μ wird dann von allen codierten Bildern abgezogen. Somit sollten alle CLIP Latent Vektoren, assoziiert zu einer Richtung $\hat{u}$, auf dem Strahl $\mu + \alpha\hat{u}$ mit $\alpha \in \mathbb{R}$ liegen.

Um semantisch interessante Latent Space Richtungen zu entdecken, muss der CLIP-Space auch solche enthalten. Das bedeutet, wenn CLIP für den Datensatz keine Richtungen enthält, wird es in dem Verfahren CLIP2StyleGAN nahezu unmöglich sein auch eine Richtung für den StyleGAN zu finden.

Es wird also erwartet, dass es eine Evolution der Attribute entlang der gefundenen Richtungen gibt: Z.B. sollten sich die Attribute bei einer Richtung, die das Alter einer Person verändert, von „a young person“ zu „an old person“ oder umgekehrt verändern.

Ein Verfahren, um Latent Space Richtungen zu erkennen, ist die PCA. Hierbei wird davon ausgegangen, dass die Daten ungefähr einer Gaußschen Normalverteilung folgen. Die Ausgabe der PCA ist ein Set aus 512 orthogonalen Hauptkomponentenachsen mit dazugehörigen Bildern, welche die Richtungen darstellen. [18]

Schritt 2: Es wurden letztendlich 512 Richtungen gefunden, jede von ihnen besitzt 512 Dimensionen (Dimensionalität des CLIP-Spaces und des Latent Spaces eines StyleGANs). Als

Nächstes muss für eine dieser beliebigen Richtungen ein Set von Worten aus einem gegebenen Lexikon gefunden werden, das die gefundene Richtung beschreibt. Dazu werden die zur Richtung gehörenden Bilder in positive und negative Extrembeispiele aufgeteilt. Im Folgenden wird μ genutzt, um weitverbreitete gemeinsame Attribute der Beispiele zu entfernen und die Unterschiede zu definieren. Nun werden für die Richtung nicht relevante Samples ignoriert. Anschließend werden alle relevanten Samples entlang der Projektion auf die Richtung sortiert ($\hat{u} \cdot (x - \mu)$), wobei die obersten und niedrigsten 50 CLIP-Bild-Embeddings jeweils eine Gruppe bilden. Diese zwei Gruppen werden genutzt, um den Vektor der Richtung zu optimieren. [18]

Algorithm 1: Text prediction using CLIP text encoder

Input: A CLIP vector $\mathbf{x} \in \mathcal{R}^{512}$;
Input: Token embeddings $\mathbf{E} \in \mathcal{R}^{m \times 512}$
Input: The number of GD steps (`maxit`)
Output: the predicted set of labels $\mathcal{L}$

/* Gradient Descent */
Initialize $\mathbf{z} \in \mathcal{R}^m \leftarrow \vec{\mathbf{0}}$;
for $i \in 1 \ldots maxit$ **do**
  $\mathbf{z} \leftarrow \mathbf{z} - \eta \nabla_{\mathbf{z}} L(\mathbf{z}, \mathbf{x}_m)$;
end
$\mathbf{e} \leftarrow \mathbf{E}^T \sigma(\mathbf{z})$;
$s_i = \mathbf{e}_i^T \mathbf{e}, \quad \mathbf{e}_i \in \text{rows}(\mathbf{E})$;
$\{i_1, \ldots, i_k\} \leftarrow$ TopK-Indices(s_i);
$\mathcal{L} \leftarrow \{\mathbf{e}_{i_1}, \ldots \mathbf{e}_{i_k}\}$;

Abbildung 26: Pseudocode für den Klassifizierungsalgorithmus einer Richtung, entnommen aus [18]

Das Lexikon besteht aus m unterschiedlichen Zeichenketten (Token). Diese werden in den CLIP-Text-Encoder mit einem passenden Präfix wie „a picture of a “ als Input übergeben. Das Ergebnis des Encoders wird im Folgenden t genannt. Das Ziel ist ein Token e zu finden, welcher im Clip Latent Space der Richtung x folgt. Da leider der perfekt passende Token e für x höchstwahrscheinlich nicht im Lexikon vorhanden ist, wird ein sogenannter Soft-Selection-Variable-Vektor z eingeführt. Dieser ermöglicht es, mehrere Token als CLIP-

Space-Vektor zu kombinieren. Hierbei gilt e = E^Tσ(z). Hier ist σ die Sigmoid-Funktion mit dem Wertebereich für z zwischen 0 und 1. Der Term E^T($E^T \in \mathbb{R}^{m\times 512}$) bildet die Embedding-Schicht des CLIP-Text-Encoders, genauer eine Matrix aus allen m Token und deren 512-dimensionaler CLIP-Space Darstellung. Das Ziel ist es jetzt ein z zu finden, welches so wenige größer Null Elemente enthält wie nur möglich. Die zu minimierende Verlustfunktion sieht wie folgt aus:

$$\begin{aligned}\mathbf{z} &= \arg\min_{\mathbf{z}_k} L(\mathbf{z}, \mathbf{x_m}) \\ &= \arg\min_{\mathbf{z}_k} \left(d_{\cos}(\mathbf{t}, \mathbf{x_m}) + \lambda H(\sigma(\mathbf{z}))\right)\end{aligned}$$

Abbildung 27: Verlustfunktion zum Finden eindeutiger Text-Klassifikatoren, entnommen aus [18]

d_{cos} ist die Kosinus-Ähnlichkeit (Cosinus Distance)[5] zwischen dem Ziel x_m und dem Token t, encoded mit dem passenden Präfix. H ist die Entropie Funktion[6]. Sobald ein passendes t gefunden wurde, wird ein entsprechendes Set an Wörtern $\mathcal{L}$ (Abbildung 26) errechnet.

Die einzelnen Richtungen haben jetzt Bezeichner, doch die meisten dieser Richtungen können mehrere Bild-Transformationen in einem Vektor enthalten. Zum Beispiel eine Veränderung der Mimik und des Alters der dargestellten Person. Die Liste $\mathcal{L}$ würde alle Token für die einzelnen Transformationen enthalten. Um diese Transformationen aufzutrennen, werden die Token nach ähnlicher Bedeutung (mithilfe des Wu-Palmer word similarity scores) geclustert. Der Wu-Palmer word similarity score berechnet die Verwandtschaft von Wörtern.

Auf diese Weise wird ein repräsentatives Wort behalten und zu ähnliche Kennzeichen entfernt. Wenn am Ende mehr als ein Token übrig ist, handelt es sich um eine Richtung mit mehreren Bild-Transformationen. In diesem Fall wird versucht, einzelne passende und atomare Vektoren für die übrigen Token zu finden.

[5] https://de.wikipedia.org/wiki/Kosinus-%C3%84hnlichkeit

[6] https://de.wikipedia.org/wiki/Entropie_(Informationstheorie)

Schritt 3: Sobald die Richtungen disentangled und bezeichnet sind, können sie durch das Überführen der Extrembeispiele in den GAN Latent Space projiziert werden. Anschließend kann mittels SVM eine Richtung berechnet werden. [18]

4 Umsetzung

Im folgendem Kapitel werden die Umsetzung und Nutzung der verwendeten Konzepte und Methoden vorgestellt. Der Focus liegt hierbei auf der Art und Weise, wie in den entsprechenden Verfahren trainiert wird. Die Umsetzung beinhaltet das Trainieren des StyleGAN2s und das Nutzen von CLIP2StyleGAN, GANLatentDiscovery und der zwei manuellen Ansätze. Des Weiteren wird beschrieben, wie mit dem Code interagiert werden kann. Die in Kapitel 3 beschriebenen Verfahren wurden im Rahmen dieser Arbeit in Python implementiert. Hierzu wurden Pytorch und Torchvision als Basis-Frameworks eingeführt. Außerdem wurden die offiziellen Libraries von StyleGAN3, CLIP2StyleGAN und GANLatentDiscovery genutzt. Als Implementation für StyleGAN2 wurde auf die Pytorch Version von rosinality zurückgegriffen.

4.1 Setup

Alle Experimente wurden in Google Colab [20] ausgeführt. Aus diesem Grund besteht der gesamte Code aus Jupyter Notebooks, mit denen die gewünschten Libraries importiert und genutzt werden. Es empfiehlt sich, die Ergebnisse automatisch in Google Drive zu hinterlegen. Von dort werden auch meine Ergebnis-Modells bereitstellt. Die Notebooks sind in einzelne Kapitel aufgeteilt. Jedes Kapitel beginnt mit einem Setup und jede Zelle ist mit einem begleitenden Text über dessen Funktion versehen. Zu Beginn der Experimente standen die Libraries in ihren damaligen Versionen zur Verfügung (Pytorch Version 1.8.1 Torchvision Version 0.9.1, in Python 3.9). Zwischenzeitlich erfolgte ein Wechsel von Google Colab auf Python 3.10 [21], deshalb musste hier auf die offizielle StyleGAN2-Ada Implementation aus

StyleGAN3 zurückgegriffen werden. Das Package Ninja wird als Build-System genutzt. Außerdem werden Packages wie tqdm, numpy und PIL eingesetzt. Zur Ausführung des Codes wird eine cuda-fähige Grafikkarte vorausgesetzt. Hierbei reicht die in Google Colab kostenlos bereitgestellte NVIDIA T4 Grafikkarte (GPU) aus.

Alle Zellen der Notebooks folgen demselben Prinzip: Die Zellen teilen sich eine Import- und Methoden-Zelle. Diese müssen vor dem Start der nachfolgenden Zellen ausgeführt werden. Jede Zelle besitzt ihr eigenes Set von Parametern und Variablen. Aufbauend auf generellen Anfangswerten für alle Zellen sind meist zu jeder Zelle noch spezifische Parameterwerte vorhanden. Die generellen Parameter können genutzt werden, um den Code für andere Modelle mit der StyleGAN2-Architektur nutzbar zu machen, wie z.B. latent, n_mlp, size und channel_multiplier. Diese Parameter werden letztendlich genutzt, um den Generator an die gegebene Aufgabe anzupassen.

4.2 Datensatz Preprocessing

Auf den meisten Bildern ist ein transparenter Hintergrund vorhanden, der durch einen schwarzen Hintergrund ersetzt wurde. Die quadratischen Bilder wurden mit Hilfe eines xBR Shaders [22] auf mindestens 64x64 Pixel hoch skaliert, da CLIP erfahrungsgemäß Probleme mit zu keinen Bildern hat. Große Bilder wurden reduziert bzw. Bilder mit anderen Seitenverhältnissen wurden gestaucht, jeweils auf 64x64 Pixel. Darüber hinaus wurden keine weiteren Bild-Transformationen durchgeführt, da StyleGAN-ADA eine built-in Bild-Transformations-Pipeline hat. Um die Bilder in einem Datensatz zusammenzufassen und die Transparenzen zu entfernen, wird das Skript dataset_tool.py aus dem StyleGAN3 Repository genutzt. Dieses bereitet die ungeordneten Bilder aus einem Ordner zu einem Datensatz auf, der als Grundlage für das anschließende Training dient. Hierbei wird jedes einzelne Bild untersucht und transformiert. Abschließend wird das gesamte Set im angegebenen Ausgabeorder als Datensatz gespeichert.

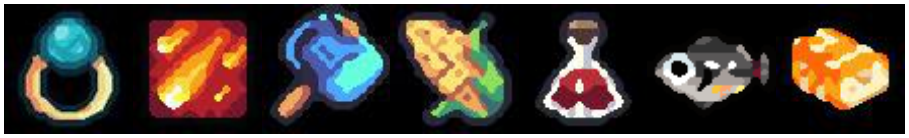

Abbildung 28: Bilder des Datensatzes nach dem Preprocessing

4.3 StyleGAN2-Ada Training

Es wurde ein unconditional StyleGAN2-Ada Modell mit Hilfe der offiziellen Implementation trainiert. Dieses Modell wurde auf einer T4-GPU in Google Colab trainiert.

Das Training benötigt für die Aufgabe eine lange Rechenzeit. Die Nutzung des kostenlosen Modells von Google Colab ist jedoch zeitlich begrenzt und wird in der Regel vor Abschluss des Trainings vorzeitig beendet. Für das erfolgreiche Trainieren des StyleGANs ist deshalb auf das Bezahl-Modell zu wechseln. Sobald die zufällig generierten Bilder angemessen aussahen, wurde das Training am letzten Checkpoint beendet. Um die Güte der Bildqualität zu bestimmen, wurden die generierten Bilder mit dem Originaldatensatz abgeglichen. Es wurden ca. 10.000 kimg trainiert, wobei ein kimg 1.000 Bilder beinhaltet und einen Schritt im Trainingsprozess darstellt. Ein (kimg) Trainingsschritt brauchte hierbei ca. 25 Sekunden mit einer T4-GPU. Das gesamte Training mit 10.000 Trainingsschritten dauerte somit durchgängig ca. 3 Tage. Die Begrenzung auf 10.000 kimg kann zwar bedeuten, dass der Konvergenzpunkt nicht erreicht wird und die Qualität des StyleGANs mit höheren Werten noch verbessert werden könnte. Jedoch wurde aufgrund der langen Rechenzeit das Training begrenzt, da selbst so schon eine gute Bildqualität erreicht wurde.

Train.py wird hierbei mit folgenden Parametern gestartet:

- Als Achitektur wird der StyleGAN2 gewählt. Diese Art des Trainierens nutzt automatisch die Ada Pipeline.
- Für Mapping network depth wird 2 gewählt.
- Die Lernrate des Generators und des Diskriminators wird auf 0,0025 gesetzt.

- Der Capacity multiplier (Gesamtmultiplikator für die Anzahl der Kanäle) wird auf 16.384 gesetzt. (= 128^2, halber Default-Wert wird empfohlen für kleine Bildgrößen)
- Gamma (= R1 regularization weight) wird mit 0,1024 gewählt.

Diese Einstellungen wurden von den Autoren von [14] hier[7] empfohlen.

Um den Trainingsablauf zu starten, wird als erstes der Datensatz geladen. Falls schon ein vortrainiertes Modell (Generator und Diskriminator) vorhanden ist, kann auch dieses für ein weiteres Training geladen werden. Als Nächstes wird die Ada Pipeline initialisiert und der Trainingsprozess wird gestartet. Anhand der geladenen Trainingsdaten werden sowohl der Loss des Generators als auch der Loss des Discriminators berechnet. Im Anschluss werden die Gewichte der beiden Netze aktualisiert. Als Loss-Funktion wird für beide, Generator und Discriminator, die Softplus-Funktion[8] genutzt. Diese werden aber je nach Trainingsphase beim Generator durch die path length regularization aus [13] und beim Diskriminator durch die R1 regularization aus [23] ersetzt. Als Optimizer wird Adam[9] verwendet.

4.4 StyleGAN2-Ada Konvertierung zu StyleGAN2

Um die Gewichte von StyleGAN2-Ada zu rosinality's StyleGAN2 zu konvertieren, wurde auf die Konvertierungs-Funktion der dvschultz's StyleGAN2-Ada Implementation zurückgegriffen. Diese Konvertierungsfunktion überträgt die Namen aus der offiziellen StyleGAN2-Ada Architektur in die rosinality's StyleGAN2 Architektur. Bei beiden Architekturen handelt es sich um ein und dieselbe, da sich StyleGAN-Ada von der rosinality's StyleGAN2 nur in der Art und Weise wie trainiert wird unterscheidet. Dennoch müssen die Namenskonventionen der einzelnen Schichten angepasst werden, damit das Modell für die nächsten Schritte überhaupt zu nutzen ist.

[7] https://github.com/NVlabs/stylegan3/blob/main/docs/configs.md

[8] https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html

[9] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[32, 512]	float32
mapping.fc1	262656	-	[32, 512]	float32
mapping	-	512	[32, 10, 512]	float32
synthesis.b4.conv1	2622465	32	[32, 512, 4, 4]	float32
synthesis.b4.torgb	264195	-	[32, 3, 4, 4]	float32
synthesis.b4:0	8192	16	[32, 512, 4, 4]	float32
synthesis.b4:1	-	-	[32, 512, 4, 4]	float32
synthesis.b8.conv0	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.conv1	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.torgb	264195	-	[32, 3, 8, 8]	float16
synthesis.b8:0	-	16	[32, 512, 8, 8]	float16
synthesis.b8:1	-	-	[32, 512, 8, 8]	float32
synthesis.b16.conv0	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.conv1	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.torgb	264195	-	[32, 3, 16, 16]	float16
synthesis.b16:0	-	16	[32, 512, 16, 16]	float16
synthesis.b16:1	-	-	[32, 512, 16, 16]	float32
synthesis.b32.conv0	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.conv1	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.torgb	264195	-	[32, 3, 32, 32]	float16
synthesis.b32:0	-	16	[32, 512, 32, 32]	float16
synthesis.b32:1	-	-	[32, 512, 32, 32]	float32
synthesis.b64.conv0	1442561	4112	[32, 256, 64, 64]	float16
synthesis.b64.conv1	721409	4112	[32, 256, 64, 64]	float16
synthesis.b64.torgb	132099	-	[32, 3, 64, 64]	float16
synthesis.b64:0	-	16	[32, 256, 64, 64]	float16
synthesis.b64:1	-	-	[32, 256, 64, 64]	float32
---	---	---	---	---
Total	22243608	11632	-	-

Abbildung 29: Tabellarische Darstellung des trainierten StyleGAN2 Generators mit Mapping-Netzwerk

Abbildung 29 zeigt die Architektur des StyleGAN2 Generators. Insgesamt gibt es 5 Convolutional Blöcke, in die der Style-Vektor w eingespeist wird.

4.5 Latent Space Projektion

Für die Projektion von Bildern in den StyleGAN2 Latent Space wird rosinality's StyleGAN2 Implementierung und das dazugehörige projector.py Skript genutzt. Diese Implementation nutzt ein von Pytorch bereitgestelltes VGG-Modell als Vergleichsmetrik für den perzeptuellen Loss. Alle Ergebnisse wurden mit einer Start-Learning-Rate von 0.01 und 1000 Steps generiert. Als Optimizer wird hierfür Adam verwendet. Hierbei wird dem in Kapitel 3.5 Latent Space Projektion beschriebenen Algorithmus gefolgt.

4.6 Manuelle Suche nach Richtungen im Latent Space

Die zwei manuellen Ansätze aus Kapitel 3.6 wurden mit Hilfe rosinality/StyleGAN2 implementiert. Die Ansätze 1 und 2 wurden nach den in Kapitel 3 beschriebenen Verfahren implementiert. Sie teilen sich einen Großteil der Funktionalität und arbeiten somit auf einem gemeinsamen Satz an Methoden.

Der manuelle Ansatz 1 für die Berechnung der Richtungen im Latent Space nutzt zwei Klassen an Bildern. Als Eingabeparameter dienen zwei Pfade zu Ordnern, die jeweils Latent Vektoren der zwei Klassen enthalten. Des Weiteren gibt es einen Parameter mit dem entschieden wird, ob mit einem SVM die Richtung errechnet oder ob über zwei Mean Vektoren die Richtung gefunden werden soll.

Der manuelle Ansatz 2, welcher explorativ arbeitet, nutzt als Eingabeparameter einen Startpunkt und einen Parameter mit dem entschieden wird, ob zufällige Richtungen oder orthonormale Richtungen genutzt werden sollen. Da die Latent Vektoren in unterschiedlichen Speicherformaten vorliegen, je nachdem wie sie erstellt wurden, wird zwischen pt und npy Dateien unterschieden.

Außerdem wurden einzelne Zellen implementiert, mit denen Richtungen getestet werden können. Darunter fällt auch eine Zelle, die Style Mixing ermöglicht. Hiermit lassen sich nicht perfekt ausgerichtete Richtungen im Latent Space besser nutzen. Das Style Mixing wurde durch das Einspeisen unterschiedlicher Style Vektoren, je nach Schicht des Generators implementiert. Dies ermöglicht Formen und Farbe voneinander zu trennen und ist eine der großen Stärken des StyleGANs, vor allem für die Latent Space Exploration. So können Style-Vektoren, die die Form verändern sollen, in den oberen/ersteren Schichten und Style Vektoren, welche Farbänderungen beinhalten, in den unteren/letzteren Schichten eingesetzt werden. Hierbei werden ein latenter Startvektor und eine Richtung genutzt. Der Parameter code1_goes_first beschreibt, ob der Startvektor im oberen oder im unteren Teil des Generators eingesetzt werden soll. Der Parameter inject_index definiert dann den Wechselpunkt zwischen den Styles.

4.7 Manuelle Richtungs-Optimierung

Neben den unüberwachten und überwachten Ansätzen wurde mit einer Notebook-Zelle noch die Möglichkeit gegeben, zwei Richtungen miteinander zu vermischen. Diese Zelle nutzt als Parameter zwei Richtungsvektoren, zwei Gewichtungen für die Vektoren und einen Latent Startpunkt. Der Startpunkt dient nur zur visuellen Darstellung, welche eine Überprüfung der Richtung erlaubt. Eine komplexere Vorgehensweise zum Optimieren einer Richtung sieht zum Beispiel folgendermaßen aus:

1. Gegeben ist eine Näherungsrichtung.
2. Diese Näherungsrichtung wird auf einen Latent Vektor (gewählt oder zufällig) angewandt.
3. Der Ergebnisvektor wird als Startvektor in den manuellen Ansatz 2 gegeben.
4. Nun wird eine der Ergebnisrichtungen, welche das Ergebnis verbessert hat, mit dem ursprünglichen Näherungsrichtung vermischt, um so die Richtung zu optimieren.

4.8 Unüberwachte Suche nach Richtungen im Latent Space

Bei den unüberwachten Verfahren zur Suche von Richtungen im Latent Space aus Kapitel 3.7 wurden jeweils die offiziellen Implementationen genutzt. Außerdem mussten beide Implementationen leicht angepasst werden, damit sie auf dem trainierten StyleGAN und der Entwicklungsumgebung funktionierten.

4.8.1 GANLatentDiscovery

Da GANLatentDiscovery direkt mit dem StyleGAN arbeitet, musste nur dieser und die Bildgröße angegeben werden. Dieses Verfahren lief durch das Trainieren des Reconstructors und der Matrix A mehrere Tage. Insgesamt trainierte GANLatentDiscovery auf einer NVIDIA T4-GPU ca. 48 Stunden. Es wurden orthogonale Startrichtungen gewählt. Der Optimizer ist wieder Adam, sowohl für den shift_predictor (Reconstructor R) als auch für den deformator

(Matrix A). Als Loss-Funktionen wurden, wie oben beschrieben, die Cross-Entropy-Loss-Funktion in Kombination mit der Mean-Absolute-Error-Funktion genutzt.

4.8.2 CLIP2StyleGAN

Um Clip2StyleGAN verwenden zu können, müssen die Bilder erst in den CLIP-Space übersetzt werden. Hierzu wurde die Offizielle CLIP-Implementation von OpenAI und ein vortrainiertes CLIP-Modell (ViT-B/32) genutzt, um 100.000 vorher generierte Bilder in den CLIP-Space zu projizieren. ViT-B/32 ist eines von mehreren vortrainierten Clip Modellen, die zum Encoden von Bildern und von Texten eingesetzt werden können. ViT-B/32, im Speziellen, codiert die Bilder in einen 512-dimensionalen Latent Space. Sobald dies geschehen ist, wurde dem CLIP2StyleGAN Ablauf gefolgt. Als erstes wurde ein PCA auf die CILP latent Vektoren durchgeführt, dann die einzelnen Bezeichner der Richtungen ermittelt und optimiert. Schließlich wurden die dazu gehörigen Bilder in den StyleGAN2 Latent Space projiziert. Als letztes wurde aus den Extremen ein Vektor, welcher die endgültige Richtung darstellt, gebildet. Durch die Geschwindigkeit eines PCAs waren sehr schnell (in der Regel nach ca. 1 bis 2 Stunden) erste Ergebnisse vorhanden. Doch das Übersetzen der Bilder in den Latent Space war sehr zeitaufwendig. Erste Ergebnisse waren also schnell zu sehen, aber eine Projizierung aller gefundener Richtungen dauerte sehr lange.

5 Ergebnisse

In diesem Kapitel wird auf die einzelnen Verfahren und deren erzielten Resultate eingegangen. Außerdem werden gesammelte Erkenntnisse über den spezifischen Latent Space des trainierten StyleGAN2s beleuchtet und die Effektivität der beiden unüberwachten Verfahren zur Bestimmung von Richtungen im Latent Space genauer analysiert.

5.1 Die Vorgehensweise

Um das Ziel der Arbeit zu erreichen, musste eine Vorgehensweise zur gezielten Generierung von einem Bild erarbeitet werden:

1. Ein StyleGAN Modell wird trainiert oder ein vortrainiertes beschafft. Hierzu wurde in 3.3 beschrieben, wie eine StyleGAN Architektur zu einem Datensatz passend gewählt werden kann und in 4.3, wie das Training abläuft.
2. Zur Ausführung kommt ein Verfahren zur unüberwachten Ermittlung von Richtungen im Latent Space. Es sollten mindestens eins, bestenfalls mehrere Verfahren genutzt werden, um die Chance, schwach bis gar nicht verwobene Richtungen zu finden, zu erhöhen. Dafür wurden GANLatentDiscovery und CLIP2StyleGAN vorgestellt und deren Verwendung beschrieben.
3. Resultierende Richtungen müssen ggf. manuell klassifiziert und qualitativ beurteilt werden.
4. Schwach verwobene Richtungen können noch durch das in 4.7 beschriebene Verfahren manuell optimiert werden.
5. Die resultierenden Richtungen können nun auf z.B. projizierte Bilder angewandt werden, um Bilder mit gewünschten Attributen zu erzeugen.

5.2 Trainierter StyleGAN2

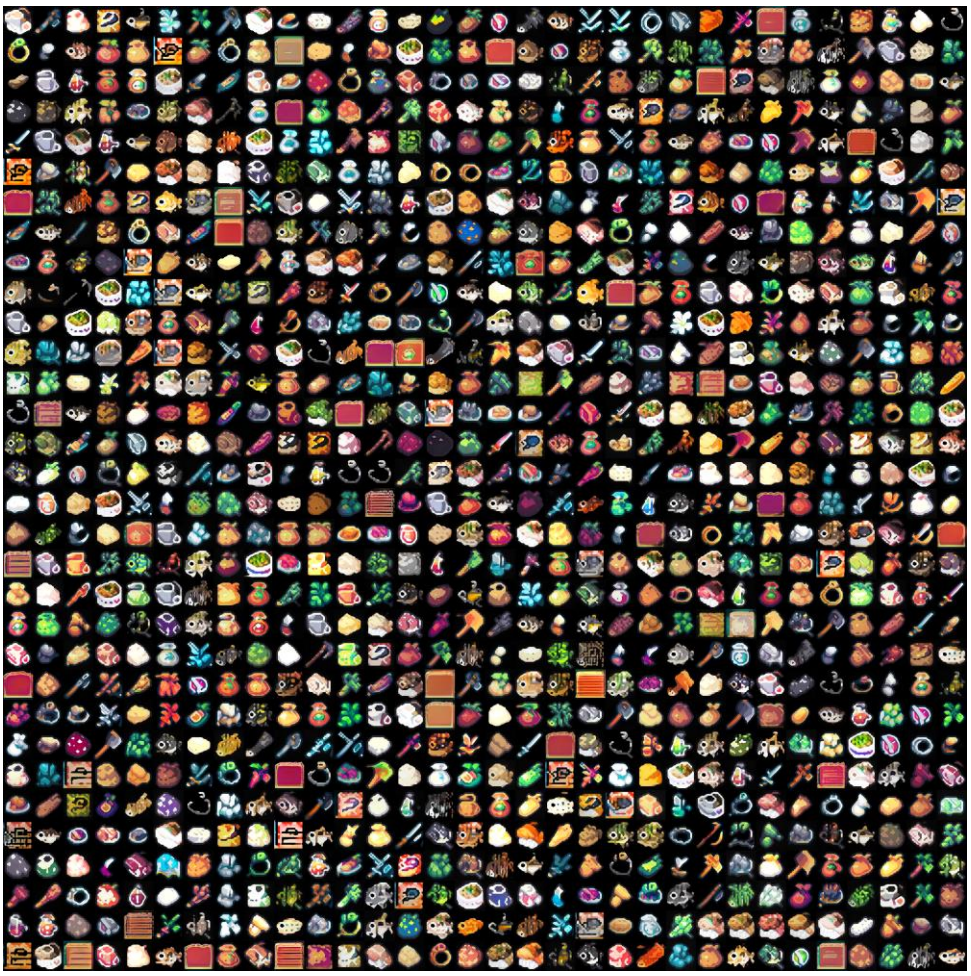

Abbildung 30: Trainings Stichprobe

Abbildung 30 zeigt eine Ausgabe zufälliger Bilder des StyleGANs, welche nach Anlegen eines Checkpoints immer mit erstellt werden. Mit Hilfe dieser Collagen kann auf einen Blick die derzeitige Qualität, wenn auch nur grob, ermittelt werden.

Die Ergebnisse des StyleGANs haben, im Vergleich zu anderen generativen KIs, selbst mit der niedrigen Anzahl an Trainingsbildern eine gute Qualität erreichen können. Einige spezielle Formen scheinen leider nicht gut dargestellt zu werden. Darunter fallen z.B. gekreuzte

Schwerter und ein Großteil der abstrakteren Icons. Ein Grund für diesen Unterschied in der Qualität könnte die geringe Anzahl an Referenzbildern gegenüber der Menge an visuellen Konzepten sein. Außerdem könnte der StyleGAN2 noch durch ein längeres Training verbessert werden, um damit weitere gute Formen zu erlernen.

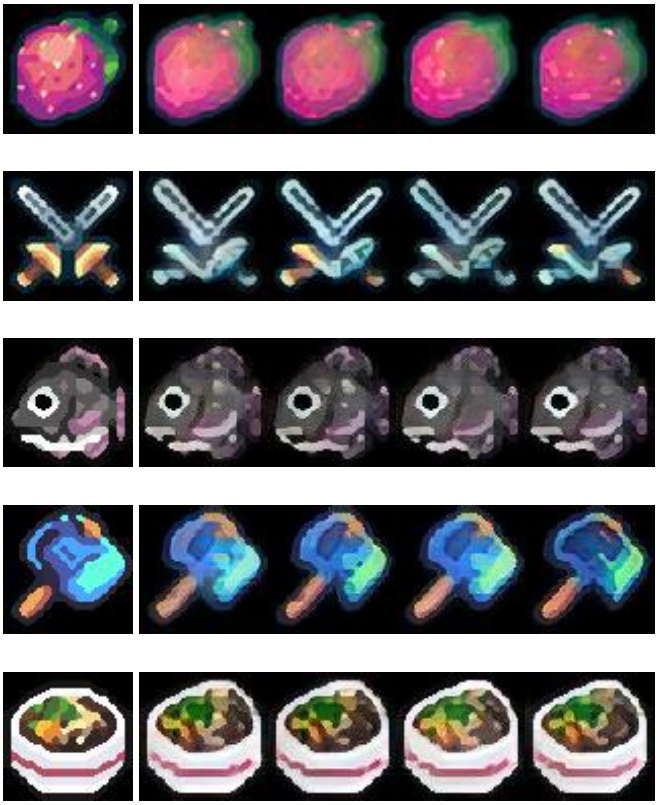

Abbildung 31: Vergleich der Originalbilder zu den Projizierten Bildern

Die rechten Bilder in Abbildung 31 sind Originalbilder und alle folgenden Bilder sind Projektionen. Hier zeigt sich die Qualität der projizierten Bilder je nach Art des Originalbildes. Da der größte Sub-Datensatz mit ca. 300 Bildern der von Fischen war, war auch die Generierung damit am erfolgreichsten. Von den gekreuzten Schwertern hingegen befanden sich im Trainingsdatensatz nur ca. 20 Bilder.

Interessant ist die Form der untersten Bildreihe in Abbildung 31. Hier scheint die Form der gefüllten Schüssel eindeutig von einer anderen Form beeinflusst worden zu sein. Als Grund

hierfür könnte die Ähnlichkeit zwischen hellen Fischen und Schüsseln anzunehmen sein. In den untersten Bildreihen von Abbildung 33 und Abbildung 36 wird deutlich, dass die angewinkelte Form der Schüssel direkt in die Stirn und die Kopfflosse des Fisches übergeht. Hieraus kann geschlossen werden, dass sich Fische und Schüsseln nicht nur nebeneinander im Latent Space befinden, sondern sich auch gegenseitig in der Form beim Training beeinflusst haben. Es kann aber auch Fälle geben, bei denen der Projektor in einem lokalen Minimum feststeckt und kein geeignetes Bild generiert wird, da immer mit einem zufälligen Start-Latent-Vektor gearbeitet wird.

5.3 Evaluation der Ansätze

In diesem Unterkapitel werden die beiden unbeaufsichtigten Ansätze auf ihre Ergebnisse und deren Qualität untersucht. Mit diesen Ergebnissen wird versucht, Schlüsse auf den Latent Space zu ziehen.

5.3.1 GANLatentDiscovery

GANLatentDiscovery hat nach dem Training 512 unterschiedliche Richtungen gefunden. Für die folgenden Beispiele wurde aus den 512 Richtung eine kleine Auswahl getroffen. In Tabelle 2 sind diese zur Übersicht aufgelistet.

Verfahren	Richtung	Attribut
GANLatentDiscovery	18	Gekreutzte Schwerter
GANLatentDiscovery	21	Verdunkeln/Aufhellen
GANLatentDiscovery	68	Grün
GANLatentDiscovery	85	Aufhellen/Verdunkeln
GANLatentDiscovery	170	Fisch
GANLatentDiscovery	316	Nicht sinnvoll
GANLatentDiscovery	256	Orange/Gelb
GANLatentDiscovery	485	Gegenstände auf Tellern

Tabelle 2: Verwendete Richtungen in den Beispielen zu GANLatentDiscovery

In der späteren Anwendung der Richtungen 21 und 85 hat sich gezeigt, dass jeweils die positive Änderung zum besseren Ergebnis führte. So ist Richtung 21 geeigneter zum Abdunkeln, wohingegen Richtung 85 besser beim Aufhellen zu verwenden ist.

An dieser Stelle hat sich die Auswertung als komplex herausgestellt, da jede Richtung manuell untersucht und klassifiziert werden musste. Am schwierigsten zu erkennen sind Richtungen, die nur im „lokalen“ Kontext sinnvoll sind. So könnte zum Beispiel eine Richtung den Wechsel in Formen beschreiben, der abhängig vom Startpunkt ist und ggf. in einem anderen Kontext keinen Sinn ergibt. Ein gutes Beispiel hierfür ist Richtung 18, welche in den meisten Fällen auf einen Bereich im Latent Space zeigt, der Schwerter und Werkzeuge (gekreuzte und nicht gekreuzte) darstellt. Diese Richtung funktioniert aber nicht für alle Startpunkte.

Abbildung 32: Beispielhafte Darstellung von GANLatentDiscovery Richtung 18: Gekreuzte Schwerter

In Abbildung 32 und allen folgenden Darstellungen von Richtungen bezieht sich das rot umrahmte Bild auf den Startvektor bzw. Startpunkt. Alle Bilder rechts vom Startvektor ausgehend unterliegen entlang der Richtung einem positiven Shift und alle Bilder links davon haben einen negativen Shift.

Andere Richtungen haben keinen Kontext und können unter „global“ semantisch sinnvoll eingeordnet werden. Darunter fallen Richtungen, deren Konzepte auf alle Bilder gleichermaßen funktionieren, wie zum Beispiel Verdunkeln/Aufhellen von Bildern (Richtung 21, Abbildung 35). Die visuell eindrucksvollsten Ergebnisse beruhten hierbei auf Richtungen, die für Farbverläufe gesorgt haben.

Insgesamt lassen sich die Richtungen grob in zwei Gruppen unterteilen. Die eine Gruppe beschreibt vornehmlich eine Veränderung in der Form, wie zum Beispiel Richtung 170 (Abbildung 33). Die andere Gruppe beschreibt hingegen eine Veränderung in der Farbdarstellung, wie zum Beispiel Richtung 68 (Abbildung 34). Weitere Gruppen könnten sich auf Unterschiede im Hintergrund beziehen. Da im Trainingsdatensatz nur schwarz als Hintergrund-

farbe vorhanden ist, wurden hier erwartungsgemäß keine abweichenden Bilder und somit auch keine Richtungen diesbezüglich gefunden

Abbildung 33: Beispielhafte Darstellung von GANLatentDiscovery Richtung 170: Fisch

Abbildung 34: Beispielhafte Darstellung von GANLatentDiscovery Richtung 68: Grün

Abbildung 35: Beispielhafte Darstellung von GANLatentDiscovery
Richtung 21: Verdunkeln/Aufhellen

Abbildung 36: Beispielhafte Darstellung von GANLatentDiscovery Richtung 316

In Abbildung 36 wird zum Beispiel keine sinnvolle Richtung ersichtlich. In diesem Fall ist die Richtung zu verwoben, da zu viele Konzepte gleichzeitig behandelt wurden.

Abbildung 37: Beispielhafte Darstellung von GANLatentDiscovery Richtung 485: Gegenstände auf Tellern

In Abbildung 37 scheinen die Startvektoren der oberen 3 Beispielreihen dasselbe visuelle Konzept zu übernehmen, die unterste Bildreihe hingegen nicht. Der Cluster “Gegenstände auf Tellern” wird in Kombination mit den Startpunkten der oberen 3 Bilderreihen und der gewählten Richtung gefunden. Der Startpunkt der untersten Reihe könnte weit hinter dem Cluster im Latent Space liegen. Es könnte aber auch sein, dass der Startpunkt in Kombination mit der Richtung am Cluster vorbeiführt und ihn somit verfehlt.

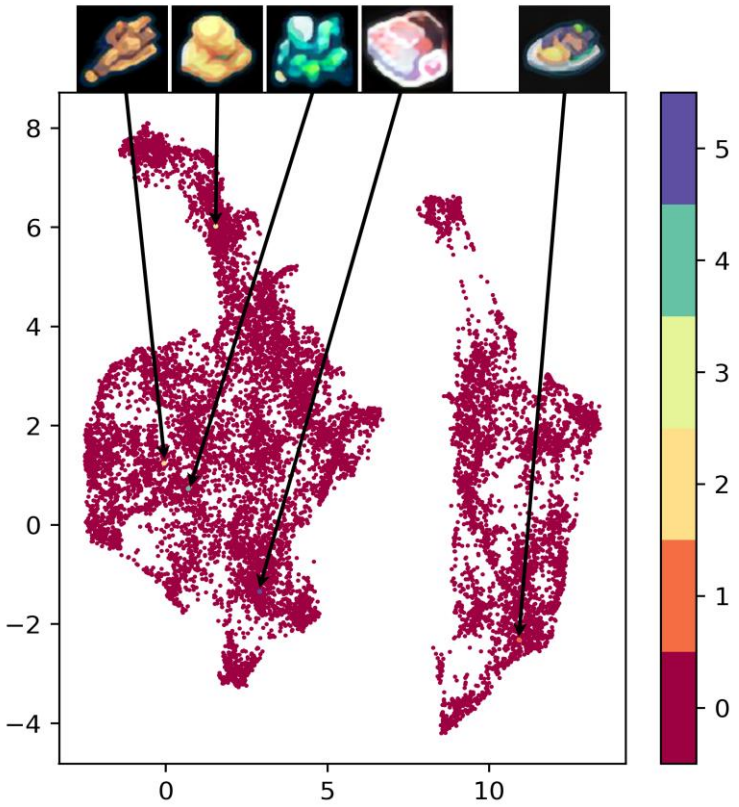

Abbildung 38: Dimensionsverkleinerung via UMAP[10] von Latent Vektoren

Eine Analyse des Latent Spaces mit UMAP zeigt hier zwei übergreifende Cluster, in denen die Positionen von fünf Bildern dargestellt werden. Einmal vier Bilder welche die Startpunkte von Abbildung 37 zeigen und einmal ein Bild von einem Gegenstand auf einem Teller (das Ziel der Richtung 485). Die Start-Latent-Vektoren befinden sich jeweils im selben übergrei-

[10] https://umap-learn.readthedocs.io/en/latest/

fenden Cluster, während der Ziel-Vektor sich in dem anderen übergreifenden Cluster befindet. Es gehen bei der Reduzierung von 512 Dimensionen auf 2D sehr viele Informationen verloren, wodurch keine eindeutige Evaluierung der oben beschriebenen Hypothesen erfolgen kann. Trotzdem ist durch die Positionierung der einzelnen Startvektoren zu erkennen, dass der 4. Startvektor von den restlichen abweicht und in Kombination mit Richtung 485 an dem Ziel-Cluster vorbeiführt.

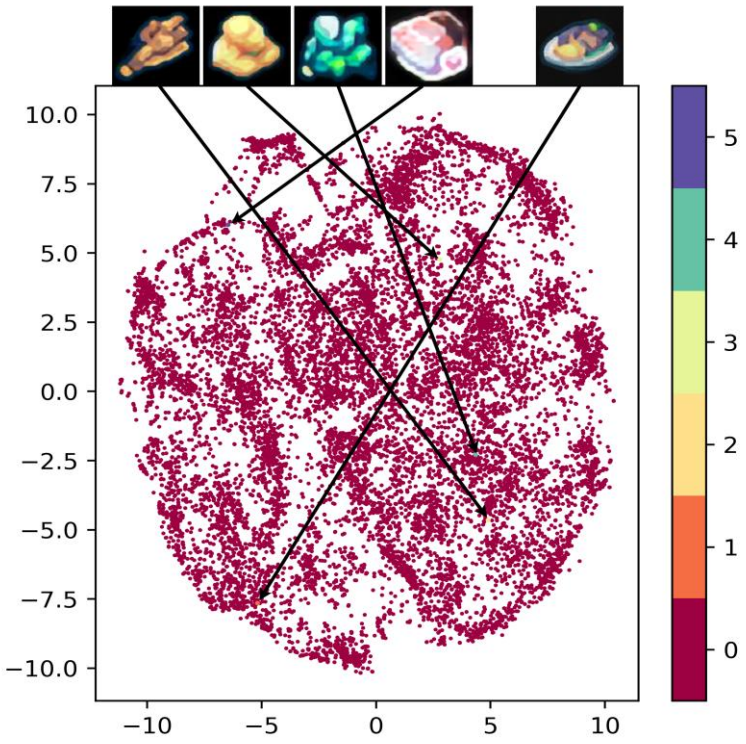

Abbildung 39: Dimensionsverkleinerung via TSNE[11] von Latent Vektoren

Dies wird durch eine Analyse des Latent Spaces mit TSNE (Abbildung 39) noch deutlicher. Alle Start-Vektoren außer dem 4. befinden sich im positiven Bereich entlang Dimension 1 (Abszisse), während der 4. im negativen Bereich liegt. Somit wird die Hypothese, dass der 4. Startpunkt in Kombination mit der Richtung am Cluster vorbeiführt und ihn somit verfehlt, sehr wahrscheinlich, wie die Evaluierung zeigt.

[11] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

5.3.2 CLIP2StyleGAN

Mit CLIP2StyleGAN wird eine beispielhafte Richtung dargestellt, diese zeigt Tabelle 3 zur Übersicht.

Verfahren	Richtung	Attribut
CLIP2StyleGAN	0	Nicht sinnvoll

Tabelle 3: Verwendete Richtungen im Beispiel zu CLIP2StyleGAN

Es wurde festgestellt, dass mit CLIP2StyleGAN innerhalb des Latent Spaces keine global brauchbaren Richtungen erzeugt werden konnten. Grund hierfür könnte Clip sein. CLIP konnte nämlich keine guten Vorhersagen auf einen Großteil der Richtungen geben. Somit wurden die Richtungen nicht mit sinnvollen Begriffen definiert und der optimize_direction Schritt musste mit zufällig gewählten Begriffen fortgesetzt werden. Ein weiter Grund könnte auch die PCA sein. Bei dieser wird versucht, möglichst viele Veränderungen mit einer Richtung darzustellen, weshalb mit diesem Latent Space nur verwobene Richtungen gefunden werden konnten. In anderen Datensätzen könnten mit diesem Verfahren wahrscheinlich gute Ergebnisse erhalten werden, doch der genutzte Datensatz vereint augenscheinlich zu viele Formen. Um diese These zu untersuchen, wurden zwei Bildersets im Latent Space interpoliert. Einmal wurde eine Veränderung ausschließlich im Farbspektrum (zu sehen in Abbildung 40) und einmal eine Veränderung nahezu ausschließlich in der Form (zu sehen in Abbildung 41) interpoliert.

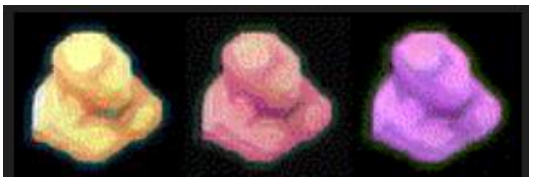

Abbildung 40: Manuell erstellter Farb-Shift - von Gelb über Rot zu Violett

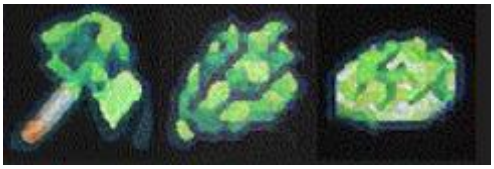

Abbildung 41: Manuell erstellter Form-Shift - von grünem Hammer zu Teller mit Grünzeug

Während der Farb-Shift ausschließlich die Farbe verändert (Gelb -> Rot -> Violett), wird in dem Form-Shift nicht nur die Form verändert. Im Zwischenschritt ist zum Beispiel kein Grau zu erkennen, während im Start-Bild und im End-Bild Grau vorhanden ist. Dies kann auf eine höhere Verflechtung im Latent Space zurückzuführen sein.

Diese Hypothese weiter zu stützen, bedürfte es einer Visualisierung mindestens mehrerer tausend klassifizierter Latent Vektoren, um die Verflechtung des Latent Spaces besser zu erkennen. Da aber der Datensatz nicht klassifiziert wurde, konnten hier keine besseren Erkenntnisse gewonnen werden.

Ein weiterer Nachteil von CLIP2StyleGAN war, dass zum Optimieren der Richtungen User-Input gebraucht wurde, um die Bezeichner der Richtung zu wählen. Des Weiteren mussten nach dem Optimieren die Bilder wieder in den Latent Space überführt werden. Hierdurch können Teile der Richtung verfälscht werden. Insgesamt wurden einzelne Richtungen schneller nutzbar gemacht als beim GANLatentDiscovery Ansatz. Die gesamte Prozedur hat jedoch durch das Projizieren von ca. 100 Bildern pro Richtung und den User-Input CLIP2StyleGAN insgesamt länger gedauert.

Abbildung 42: Beispielhafte Darstellung von CLIP2StyleGAN Richtung 0

5.3.3 Style Mixing und Manuelle Optimierung der schon gefundenen Richtungen

Im Ergebnis haben nahezu alle Richtungen kleine Nebeneffekte, die nicht gewünscht sind. So verändert die Richtung 256 von GANLatentDiscovery nicht nur die Farbe zu Orange/Gelb, sondern verändert auch stark die Form. Um die zum Teil immer noch sehr verwobenen Richtungen besser aufzutrennen kann Style-Mixing genutzt werden.

Abbildung 43: Beispielhafte Darstellung von GANLatentDiscovery Richtung 256: Orange/Gelb

Abbildung 44: Stylemixing: Farblicher Shift entlang Richtung 256

Abbildung 44 zeigt ganz links das Ausgangsbild, daneben das verschobene Bild und als drittes das mittels Style Mixing verschobene Bild. Generiert wurde das dritte Bild, indem der w code des zweiten Bildes ab dem 5. Layer genutzt wurde (inject_index = 5). Vergleicht man nun das erste und das dritte Bild, so fällt auf, dass der Farbwechsel trotzdem die Form und die zugrundeliegenden Schattierungen nicht beeinflusst hat. Das Bild ist allerdings insgesamt etwas dunkler geworden. Um diesen Effekt zu kontern, wurde zusätzlich Richtung 85 von GANLatentDiscovery genutzt (Abbildung 45).

Abbildung 45: Stylemixing: Farblicher Shift entlang Richtung 256 und Richtung 85

6 Fazit

6.1 Zusammenfassung

Ziel der Arbeit war das zielgerichtete Generieren eines Bildes, durch manuelles Verändern eines Latent Vektors. Dieses Ziel sollte durch das Aufschlüsseln des Latent Spaces eines trainierten StyleGANs erreicht werden. Das heißt, es sollten Richtungen im Latent Space gefunden und sinnvoll genutzt werden. Hierfür wurde ein Ablauf zum Finden, Untersuchen, Optimieren und Anwenden dieser Richtungen im Latent Space entwickelt. Um diese Richtungen zu finden, wurden zwei unbeaufsichtigte Verfahren untersucht.

Im ersten Ansatz, nach **Unsupervised Discovery of Interpretable Directions in the GAN Latent Space**, wurden Richtungen durch das Optimieren von zufällig initialisierten Vektoren gefunden. Dabei wurden Richtungen entdeckt, welche interessant und semantisch bedeutungsvoll sind.

Der zweite Ansatz, nach **CLIP2StyleGAN**, hat mit Hilfe eines PCAs auf Clip-Bild-Embeddings Richtungen gefunden. Dies führte jedoch zu keinen nutzbaren Richtungen. Grund hierfür war sicherlich, dass der Datensatz embedded in den CLIP-Space nicht kontinuierlich abgebildet wurde. Das kann sowohl an der Größe der Bilder gelegen haben, als auch an der Natur der Bilder (z.B. Pixel-Art-Bilder).

Insgesamt wurden semantisch interessante Richtungen im Latent Space gefunden und angewandt. Außerdem wurden diese miteinander kombiniert und verändert, um ein Bild gezielt zu erstellen. Hierbei haben insbesondere die manuellen Ansätze dazu beigetragen, die schon ermittelten Richtungen für das menschliche Auge zu optimieren.

Durch die Untersuchung des umliegenden Latent Spaces konnte die gewünschte Form eines Bildes präzisiert werden und durch die Interpolation zwischen zwei Klassen von Bildern konnten auch weitere Farbübergänge als Richtungen mit aufgenommen werden.

6.2 Erweiterung des Codes und noch existierende Probleme

Das Training des StyleGAN2 wurde zeitlich beschnitten. Ein längeres Trainieren wäre zwar durchaus möglich gewesen, doch hätte dies unter Umständen nicht zu einer bedeutsamen Verbesserung geführt, falls der Konvergenzpunkt überschritten worden wäre. Als Abbruchkriterium kann u.a. die Perzeptuelle-Pfadlänge dienen. Sobald diese nicht mehr fällt und stattdessen ansteigt, ist der Konvergenzpunkt überschritten.

Es könnten weitere Verfahren zur Richtungsfindung genutzt werden. Zum Beispiel könnte ein PCA direkt auf dem Latent Space laufen. Dies könnte die aufgetretenen Unstimmigkeiten mit CLIP zwar lösen, allerdings würden auch die Beschriftungen der Richtungen ohne CLIP wegfallen. Als ein weiteres Verfahren zur Richtungsfindung wäre z.B. GANspace noch zu nennen.

Letztendlich könnte die Qualität der Bilder und der gefundenen Richtungen durch längeres Trainieren und Auswechseln der Verfahren erhöht werden.

Eine mögliche Weiterführung des Projektes könnte auch das prozedurale Generieren von Videospielinhalten sein. So könnten Texturen oder Icons je nach Spielphase neu generiert werden, denn damit lässt sich eine dynamische Anpassung an das Spielgeschehen verwirklichen. Hierfür müsste jedoch die Qualität der Bilder erhöht und der Latent Space noch weiter aufgeschlüsselt werden.

Des Weiteren könnten Experimente auf weniger diversen Datensätzen (wie Flickr-Faces-HQ) durchgeführt werden. Diese StyleGANs könnten dann auf einem weniger verwobenen Latent Space aufsetzen und damit das Finden von Richtungen im Latent Space durchaus vereinfachen.

6.3 Ausblick

Im letzten Jahr wurden generative Modelle bekannt, die für den Menschen aufgeschlüsselte Latent Spaces beinhalten. Darunter fallen vor allem die Latent Diffusion Modelle wie Stable Diffusion, aber auch StyleGAN-T.

6.3.1 Stable diffusion

Stable diffusion beruht auf der Latent Diffusion, bei dem ein Bild schrittweise von einem rauschigen Anfangszustand zu einem hochauflösenden Ergebnis entwickelt wird. Die Methode basiert auf einem generativen Modell, das aus zwei Komponenten besteht: einem Diffusionsprozess und einem invertierten Modell. Der Diffusionsprozess simuliert den schrittweisen Übergang des Bildes von einem Rauschzustand zu einem rauschfreien Zielzustand. Das invertierte Modell ermöglicht es, von dem Zielzustand auf den Rauschzustand zurückzuschließen. Verwendet werden die Fähigkeiten von CLIP, um den Bildvergleich und die Bewertung der visuellen Qualität der generierten Bilder zu unterstützen. Indem die semantische Ähnlichkeit zwischen den erzeugten Bildern und einer gegebenen Textbeschreibung berücksichtigt wird, kann das Training des Modells gelenkt werden und die erzeugten Bilder somit besser kontrollieret werden. Durch die Integration von CLIP in den Prozess der hochauflösenden Bildsynthese ermöglicht das Modell eine gezieltere und kontrolliertere Generierung von Bildern, die den semantischen Inhalten der gegebenen Textbeschreibung entsprechen. [24]

6.3.2 StyleGAN-T

Basierend auf StyleGAN-XL (StyleGAN3) versucht StyleGAN-T mit den state-of-the-art Diffusions-Modellen zu konkurrieren. Dieses Modell nutzt wie Stable Diffusion CLIP als Führung. Genauer gesagt, wird die CLIP-Führung in die Loss-Funktion mit einbezogen. Gleichzeitig wird ein Text-Encoder als weiterer, trainierbarer Bestandteil mit aufgenommen. Die große Stärke gegenüber den Diffusions-Modellen ist hierbei die Geschwindigkeit des GANs. So können in einem Zeitraum unter einer Sekunde Bilder mit 512x512 Pixeln erstellt werden. Diese generierten Bilder sind aber in höheren Auflösungen von schlechterer Qualität als die durch vergleichbare Diffusions-Modelle erstellten Bilder. [25]

Literaturverzeichnis

[1] J. P.-A. M. M. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, „Generative Adversarial Nets“, Université de Montréal, Canada: arXiv e-paper, 10.06.2014.

[2] R. Razavi-Far, A. Ruiz-Garcia, A. Ruiz-Garcia und J. Schmidhuber, „Generative Adversarial Learning: Architectures and Applications“, Switzerland: Springer-Verlag, eBook, 07.02.2022.

[3] Google for Developers, „Loss Functions“, 18.07.2022, [Online]. Available: https://developers.google.com/machine-learning/gan/loss. Mountain View, CA, USA: Google LLC, [Zugriff am 14.05.2023].

[4] D. Godoy, „Understanding binary cross-entropy / log loss: a visual explanation“, 21.11.2018, [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. Toronto, Canda: Towards Data Science, [Zugriff am 14.05.2023].

[5] S. L. T. A. Tero Karras, „A Style-Based Generator Architecture for Generative Adversarial Networks“, NVIDIA: arXiv e-paper, 29.03.2019.

[6] J. Schmidhuber, „Deep Learning in Neural Networks: An Overview“, University of

Lugano & SUPSI, Switzerland: arXiv e-paper, 08.10.2014.

[7] Google for Developers, „GAN Common Problems“, 18.07.2022, [Online]. Available: https://developers.google.com/machine-learning/gan/problems. Mountain View, CA, USA: Google LLC, [Zugriff am 14.05.2023].

[8] Google for Developers, „GAN Training“, 18.07.2022. [Online]. Available: https://developers.google.com/machine-learning/gan/training. Mountain View, CA, USA: Google LLC, [Zugriff am 14.05.2023].

[9] T. A. S. L. J. L. Tero Karras, „Progressive Growing of GANs for Improved Quality, Stability, and Variation“, ICLR 2018, NVIDIA and Aalto University, Finland: arXiv e-paper, 26.02.2018.

[10] S. B. Xun Huang, „Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization“, Cornell University, New York, USA: arXiv e-paper, 30.07.2017.

[11] T. Hastie, R. Tibshirani und J. Friedman, „The Elements of Statistical Learning Data Mining, Inference, and Prediction, Second Edition“, New York, USA: Springer New York, eBook, 26.08.2009.

[12] J. S. J. A. K. S. David Winant, „Latent Space Exploration Using Generative Kernel PCA“, KU Leuven, Belgium: arXiv e-paper, 28.05.2021.

[13] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen und T. Aila, „Analyzing and Improving the Image Quality of StyleGAN“, NVIDIA and Aalto University: arXiv e-paper, 23.03.2020.

[14] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen und T. Aila, „Training Generative Adversarial Networks with Limited Data“, NVIDIA and Aalto University,

Finland: arXiv e-paper, 07.10.2020.

[15] R. Zhang, P. Isola, A. A. Efros, E. Shechtman und O. Wang, „The Unreasonable Effectiveness of Deep Features as a Perceptual Metric“, UC Berkeley, OpenAI and Adobe Research: arXiv e-paper, 10.04.2018.

[16] R. Abdal, Y. Qin und P. Wonka, „Image2StyleGAN++: How to Edit the Embedded Images?“, KAUST, Saudi Arabia, Cardiff University, UK: arXiv e-paper, 07.08.2020.

[17] A. B. Andrey Voynov, „Unsupervised Discovery of Interpretable Directions in the GAN Latent Space“, Yandex, Russia and National Research University Higher School, Moscow, Russia: arXiv e-paper, 24.06.2020.

[18] P. Z. J. F. N. J. M. P. W. Rameen Abdal, „CLIP2StyleGAN: Unsupervised Extraction of StyleGAN Edit Directions“, KAUST, Saudi Arabia, Miami University, UCL and Adobe Reserch: arXiv e-paper, 09.12.2021.

[19] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark und G. Krueger, „Learning Transferable Visual Models From Natural Language Supervision“, OpenAI, San Francisco, USA: arXiv e-paper, 26.02.2021.

[20] Google Colab, „Google Colab“, [Online]. Available: https://colab.research.google.com/notebooks/basic_features_overview.ipynb. Mountain View, CA, USA: Google LLC, [Zugriff am 14.05.2023].

[21] E. Johnson, „Colab Updated to Python 3.10“, 27.04.2023, [Online]. Available: https://medium.com/google-colab/colab-updated-to-python-3-10-27eb02daa162. San Francisco, USA: A Medium Corporation, [Zugriff am 14.05.2023].

[22] M. Laumeister, „Online XBR Pixel Art Upscaler“, [Online]. Available: https://www.maxlaumeister.com/pixel-art-upscaler/. California, USA: Maximillian Laumeister, [Zugriff am 14.05.2023].

[23] L. Mescheder, A. Geiger und S. Nowozin, „Which Training Methods for GANs do actually Converge?“, MPI Tübingen, Germany, ETH Zürich, Switzerland, Microsoft Research, Cambridge, UK: arXiv e-paper, 31.07.2018.

[24] R. Rombach, A. Blattmann, D. Lorenz, P. Esser und B. Ommer, „High-Resolution Image Synthesis with Latent Diffusion Models“, Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany and Runway ML: arXiv e-paper, 13.04.2022.

[25] A. Sauer, T. Karras, S. Laine, A. Geiger und T. Aila, „StyleGAN-T: Unlocking the Power of GANs for Fast Large-Scale Text-to-Image Synthesis“, University of Tübingen, Tübingen AI Center, Germany and NVIDIA: arXiv e-paper, 23.01.2023.

[26] A. H. J. L. S. P. Erik Härkönen, „GANSpace: Discovering Interpretable GAN Control“, Aalto University, Finland, Adobe Research and NVIDIA: arXiv e-paper, 14.12.2020.

[27] A. Jahanian, L. Chai und P. Isola, „On the "steerability" of Generative Adversial Networks“, ICLR 2020, Massachusetts Institute of Technology, Cambridge, USA: arXiv e-paper, 17.02.2020.

Erklärung zur selbstständigen Bearbeitung

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