

Hochschule Düsseldorf

Bachelorarbeit

Generative Adversarial Networks zur Erzeugung
von 3D-Avataren

Alexander Pech

736825

Betreuer

Prof. Dr. Christian Geiger

M.Sc. Philipp Ladwig

31. März 2020

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

(Alexander Pech)
Bochum, den 31. März 2020

Mithilfe von *Generative Adversarial Networks* (kurz GANs) ist es möglich Bilder, auf Basis eines Bilddatensatzes, zu erzeugen, welche sich problemlos in diesen eingliedern ließen. Aus einem Datensatz, bestehend aus Gesichtern, lassen sich somit beispielsweise neue Gesichter erstellen.

In der folgenden Arbeit geht es darum, inwiefern sich GANs zur Erstellung von 3D-Avataren nutzen lassen. Dazu wird zunächst der grundlegende Aufbau und die Funktionsweise von GANs beschrieben sowie deren Möglichkeit der Konditionierung, um die Anforderungen an das Erstellen von Avataren zu erfüllen. Zum Schluss wird eine GAN-Architektur entwickelt, welche auf Basis von RGBD-Bildern einen 3D-Face-Avatar erzeugen kann.

With the help of *generative adversarial networks* (GANs for short) it is possible to generate images based on an image data set, which could be integrated into these without any problems. For example, new faces can be created from a data set consisting of faces.

The following work deals with the extent to which GANs can be used to create 3D avatars. For this purpose, the basic structure and functionality of GANs is described, as well as the possibility of customizing them to meet the requirements for creating avatars. Finally, a GAN architecture is developed that can create a 3D face avatar based on RGBD images.

Inhaltsverzeichnis

1. Einleitung	5
2. Generative Adversarial Networks	7
2.1. Deep Convolutional Generative Adversarial Network	9
2.2. Erzeugung dreidimensionaler Objekte	13
2.3. Semantische Bearbeitung im latenten Raum	14
2.3.1. Vektorarithmetik im latenten Raum	15
2.3.2. Analyse des latenten Raumes von trainierten GANs	17
2.4. Konditionierung	19
2.4.1. Conditional-GAN	20
2.4.2. Pix2Pix-GAN	21
3. Anforderungen an das Erzeugen von Avataren in Bezug auf GANs	26
3.1. Erhalt der Identität	27
3.2. Steuerung durch Posen-Informationen	29
3.3. Bisherige Umsetzungen in 2D	31
3.4. Mögliche Umsetzung in 3D durch RGBD-Bilder	31
4. Entwicklung einer GAN-Architektur zur Erzeugung von RGBD-Face-Avataren	33
4.1. Trainingsdaten	33
4.1.1. Aufnahme von RGBD-Bildern	33
4.1.2. Aufbereitung der Tiefendaten	34
4.1.3. Bestimmung von Facial Landmarks	37
4.1.4. Skalierung und Zuschnitt	38
4.2. GAN-Architektur	38
4.3. Trainingsprozess	39
4.4. Evaluierung	40
4.5. Interpretation des RGBD-Bildes zur Erzeugung einer Punktfolge	42
5. Fazit und Ausblick	44
Literatur	45
Abbildungsverzeichnis	48
A. Normalized Mean Error	49

1. Einleitung

Virtuelle Umgebungen bieten einen neuen Raum für Interaktion. Entferntes Zusammenarbeiten in VR/AR kann beispielsweise die Vorteile eines realen Meetings mit denen einer Videokonferenz vereinen. Reale Treffen sind in der Regel intensiver als virtuelle. Eine Videokonferenz bietet hingegen den Vorteil, dass sich die Teilnehmer nicht zeitgleich am selben Ort befinden müssen. Die Vorteile der einen Variante entsprechen dabei meist den Nachteilen der anderen.

Damit zwischenmenschliche Interaktion in einer virtuellen Umgebung möglichst natürlich wahrgenommen wird, sollte sie der Interaktion in der Realität ähneln, da dem Menschen diese Art bereits vertraut ist. Ein möglicher Schritt, um diesem Ziel näher zu kommen, ist die Verwendung von 3D-Avataren. Dabei wird jeder Teilnehmer durch einen Avatar in der virtuellen Umgebung repräsentiert, welchen er steuert. Das Steuern des Avatars sollte bestenfalls unbewusst geschehen, da der Fokus auf der Interaktion untereinander und nicht auf dem Steuern des eigenen Avatars liegen sollte.

Ein denkbare Szenario wäre, dass es sich bei den Avataren, ähnlich wie in Abbildung 1 zu sehen, um 3D-Hologramme handelt und jeder Teilnehmer der Session ein *Head-Mounted Display* (kurz: HMD) trägt. Beim Tragen eines HMDs wird jedoch das Gesicht zu einem Großteil von diesem verdeckt. Da Mimik ein wichtiger Bestandteil von Kommunikation ist, sollte das Gesicht sichtbar sein.



Abbildung 1: virtuelle 3D-Teleportation in Echtzeit von Microsoft Research [1]

Ein möglicher Lösungsansatz für das Verdeckungsproblem wäre eine Rekonstruktion des Gesichtes auf Basis von Facial Landmarks. Bei Facial Landmarks handelt es sich um eine abstrakte Darstellung des Gesichtes. Pose und Mimik des Gesichtes werden dabei durch (meist) 68 Punkte definiert. Die reduzierte Darstellung ermöglicht es Informationen über Pose und Mimik weitestgehend unabhängig vom Aussehen zu übertragen. Merkmale wie Augenabstand oder Gesichtsbreite bleiben erhalten. Die Landmarks können aus Daten von mehreren Quellen zusammengesetzt werden. In Bezug auf den oben beschriebenen Anwendungsfall können die Facial Landmarks auf Basis von Kameras innerhalb und außerhalb des HMDs erstellt werden.

Um das durch das HMD verdeckte Gesicht rekonstruieren zu können, muss dieses auf Basis der Facial Landmarks synthetisch erzeugt werden. In den letzten Jahren haben vor allem *Generative Adversarial Networks* (kurz: GANs) beeindruckende Ergebnisse im Bereich der künstlichen Erzeugung von realistischen Gesichtern gezeigt [11][12]. Auf Basis eines Bilddatensatzes werden dabei neue Bilder erstellt. Eine dreidimensionale und durch Landmarks steuerbare Umsetzung würde es ermöglichen einen 3D-Face-Avatar zu erzeugen, welcher das verdeckte Gesicht im Hologramm ersetzen könnte.

Ziel dieser Arbeit ist es, eine GAN-Architektur zur Erstellung von 3D-Face-Avataren zu entwickeln. Zunächst wird dafür in Abschnitt 2 auf die Funktionsweise von Generative Adversarial Networks sowie einige relevante Weiterentwicklungen eingegangen. In Abschnitt 3 wird beschrieben, wie sich die Anforderungen an das Erstellen von Avataren mit einer GAN-Architektur realisieren lassen. Die konkrete Umsetzung ist in Abschnitt 4 dokumentiert.

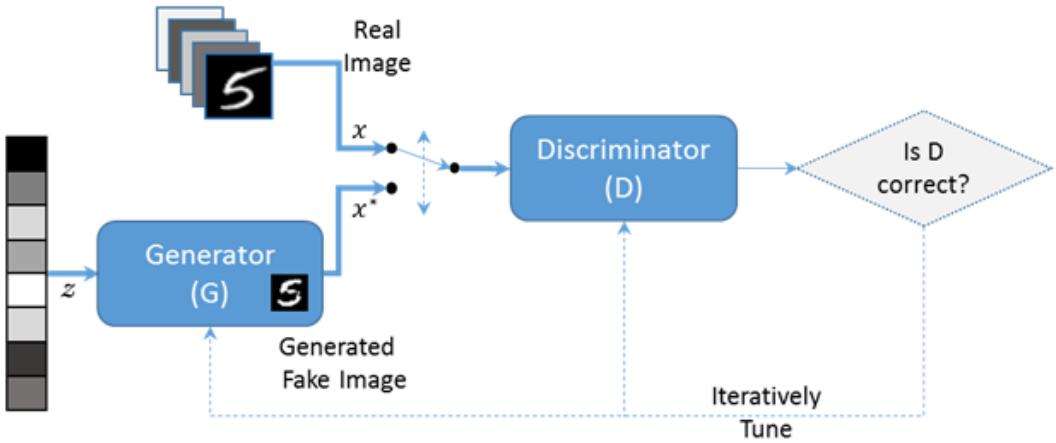


Abbildung 2: Generative Adversarial Network [18]

2. Generative Adversarial Networks

Die ursprüngliche Idee von *Generative Adversarial Networks* (kurz GAN) wurde erstmals von Ian J. Goodfellow et al. 2014 in dem Paper *Generative Adversarial Nets* [6] veröffentlicht. Nach Goodfellow et al. [6] besteht ein GAN aus zwei neuronalen Netzen (fully connected), die gegeneinander arbeiten. Das eine Netz dient dabei der Erzeugung von Bildern, das andere der Bewertung dieser. Im Folgenden werden die Netze *Generator* und *Diskriminatator* genannt (vgl. Abbildung 2).

Der Generator erhält als Eingabe einen latenten Vektor z , der mit Rauschen initialisiert wird. Der Vektor z besteht aus 100 Dimensionen, die je einen zufälligen Wert enthalten. Die einzelnen Zufallswerte sind dabei normalverteilt (Gaußverteilung mit dem Erwartungswert $\mu = 0$ und einer Standardabweichung von $\sigma = 1$).

Die Qualität des durch den Generator G erstellten Bildes ist abhängig von den Netzparametern Θ_g . Bei einem GAN, welches auf Basis eines Datensatzes aus Gesichtern neue Gesichter erzeugen soll, sorgt z dafür, dass immer neue Gesichter entstehen und Θ_g dafür, wie real die Gesichter aussehen.

Der Diskriminatator D soll unterscheiden, ob ein Bild x aus dem Datensatz kommt oder es sich um das Ergebnis des Generators handelt. Die Ausgabe s des Diskriminators entspricht dabei der Wahrscheinlichkeit, mit der ein Bild *echt* oder *falsch* ist (vgl. Gleichung 1 und Gleichung 2).

$$x = G(\Theta_g, z) \quad (1)$$

$$s = D(\Theta_d, x) \quad (2)$$

Beim Trainingsprozess der beiden neuronalen Netze werden die Netzparameter Θ_g und Θ_d optimiert. Der Trainingsprozess der beiden Netze wird von Goodfellow et al. [6] wie folgt beschrieben:

„ D and G play the following two-player minimax-game with value function $V(G, D)$:“ (Goodfellow et al. [6])

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - (D(G(z)))] \quad (3)$$

Das *Minimax-Game* ist wie folgt zu verstehen: der Generator wird darauf trainiert, den Unterschied zwischen den erstellten Bildern und denen des Datensatzes zu minimieren, wohingegen beim Diskriminatator die Wahrscheinlichkeit erhöht werden soll, dass dieser beim Unterscheiden der Bilder richtig liegt.

Im Idealfall sollen die Bilder des Generators nach dem Trainingsprozess so gut sein, dass der Diskriminatator mit einer 50:50-Chance raten muss, ob ein Bild aus dem Datensatz stammt oder vom Generator erstellt worden ist. Für diesen Fall gilt:

$$D(x) = \frac{1}{2} \quad (4)$$

Deutlich wird dies auch, wenn man $V(D, G)$ sowie die beiden Teil-Loss-Funktionen plottet (vgl. Abbildung 3). Der geringste Fehler (Loss) wird erreicht, wenn $D(x) = 1/2$ entspricht. (Der rote Graph in Abbildung 3 entspricht der kompletten Loss-Funktion $V(D, G)$.)

Generative Adversarial Networks liefern unter den generativen Modellen derzeit die qualitativ hochwertigsten Ergebnisse. Oft neigen Generative Modelle dazu, Bilder zu erzeugen die verschwommen sind, was bei GANs nicht der Fall ist, da der Output des Generators vom Diskriminatator überwacht bzw. bewertet wird. Der größte Vorteil von GANs ist dabei aber auch der größte Nachteil, denn die Trainingsstabilität des Gegenspiels von Diskriminatator und Generator ist oft instabil.

Für den Fall, dass eines der beiden Netzwerke zu schnell zu gut lernt und die jeweilige Loss-Funktion (grauer und grüner Graph in Abbildung 3) somit 0 entspricht, geht $V(D, G) \rightarrow \infty$.

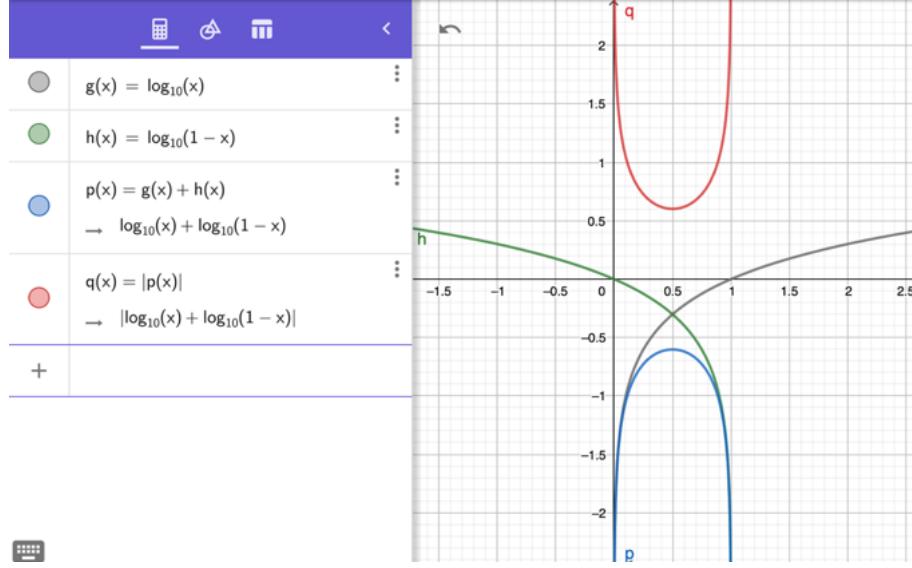


Abbildung 3: MinMax-Game Plot. Die x-Achse entspricht hier $D(x)$ und die y-Achse dem Loss-Wert.

Gemäß dem Fall, dass der Diskriminator sich immer sicher ist, entspricht dies der Problematik des *Overfitting*, bei dem ein Netz die Trainingsdaten auswendig lernt. Egal wie gut die Bilder des Generators wären, sie würden entlarvt werden. In der Regel sind diese zu diesem Zeitpunkt jedoch wesentlich schlechter, da dieser Fall sonst nicht eintreten würde.

Im entgegengesetzten Szenario akzeptiert der Diskriminator alle Bilder des Generators, worauf diese im Trainingsprozess wieder schlechter werden. Der Generator erhält in dem Fall kein Feedback darüber, ob eine Änderung an den Netzparametern für bessere oder schlechtere Ergebnisse sorgt.

Die Verbesserung der Trainingsstabilität von GANs ist, auf Grund der qualitativ hochwertigen Ergebnisse, ein großes Forschungsgebiet mit zahlreichen Ansätzen. Einer der grundlegendsten Ansätze zur Verbesserung der Trainingsstabilität ist die Verwendung von Faltungsschichten (engl.: *Convolutional-Layer*). Erstmals wurden diese, in Bezug auf GAN-Architekturen, beim DCGAN von Radford et al. [23] verwendet.

2.1. Deep Convolutional Generative Adversarial Network

2016 wurde von Radford et al. ein Ansatz geschaffen, mit dem sich das Training von GANs durch eine Veränderung der Struktur der Netze von Generator und Diskriminator stabilisieren lässt. Statt Multi-Layer Perzeprontronen, bei denen die Neuronen eines Layers

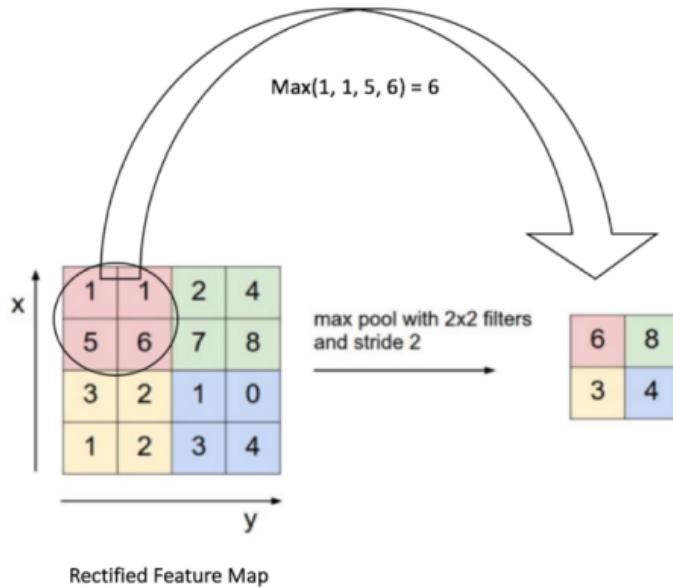


Abbildung 4: Max-Pooling [10]

mit allen Neuronen des darauf folgenden Layers verbunden sind, kommen abgeänderte CNNs zum Einsatz. Standardmäßig bestehen CNNs aus drei verschiedenen Layer-Arten: den Convolutional-Layern, Pooling-Layern und Fully-Connected Layern (vgl. O'Shea et al. [21]). Von Radford et al. [23] werden im Wesentlichen fünf Änderungen vorgenommen:

1. „Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).“ (Radford et al. [23])

Bei klassischen CNNs werden Bildmatrizen in Vektoren umgewandelt, die Informationen über den Bildinhalt darstellen. Aus dem Bild eines Autos wird also beispielsweise die Information extrahiert, dass auf dem Bild ein Auto zu sehen ist. Diese Information wird durch einen Vektor dargestellt, in welchem jede Dimension die Wahrscheinlichkeit für ein mögliches Ausgangsszenario (hier: *abgebildetes Objekt*) enthält (vgl. Abbildung 5). In Bezug auf das genannte Beispiel wäre hier also der Wert in der Dimension, welche das Auto repräsentiert, am größten.

Klassische CNNs sind so aufgebaut, dass auf jede Faltungsschicht ein Pooling-Layer folgt, welcher zur Datenreduktion beiträgt. Beim MaxPooling mit einer Kernelgröße von 2×2 Pixeln wird aus den 4 Pixeln nur das Pixel mit dem größten Wert behalten. Dabei werden 75% der Informationen verworfen (vgl. Abbildung 4).

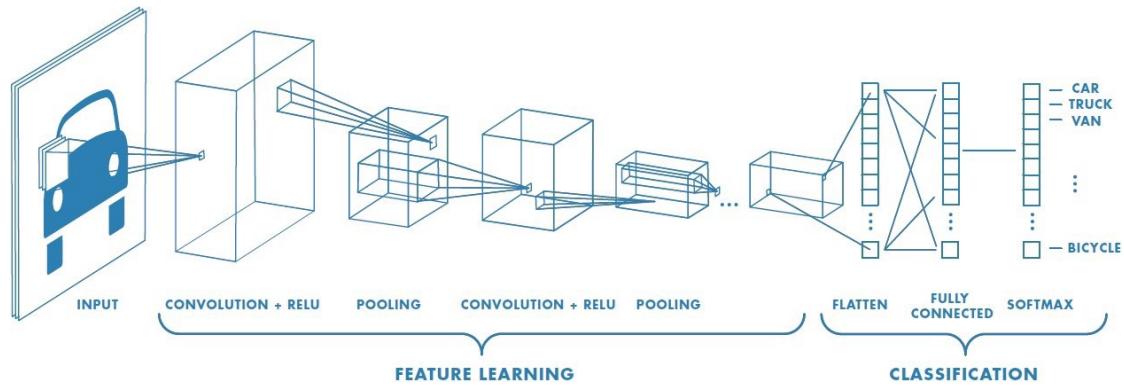


Abbildung 5: Convolutional Neural Network [25]

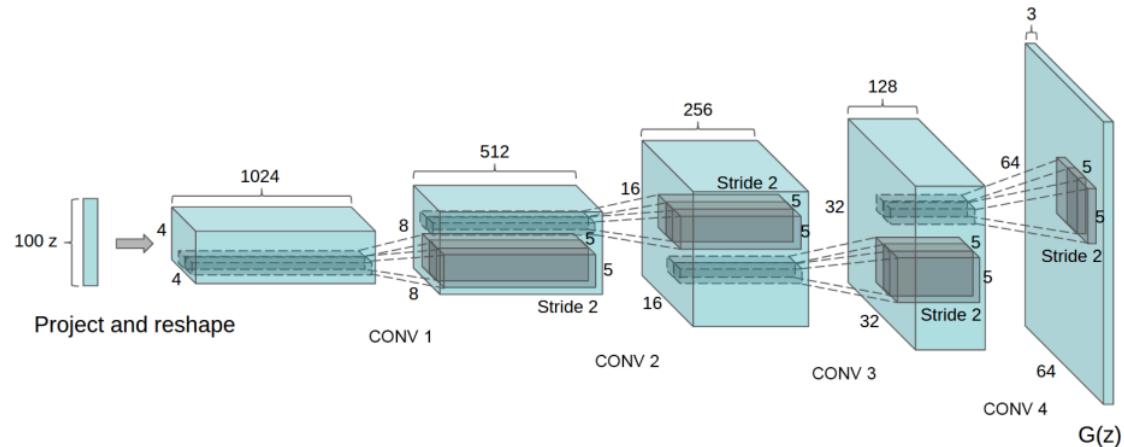


Abbildung 6: DCGAN-Generator von Radford et al. [23]

Bei dem Generator vom DCGAN handelt es sich von der grundlegenden Idee her um ein umgedrehtes CNN (vgl. Abbildung 6), da aus einem latenten Vektor ein Bild entstehen soll. Wenn ein Pooling-Layer umgedreht werden soll, muss der Wert für die dabei neu entstehenden Pixel (*Upscaling*) im Grunde interpoliert oder kopiert werden.

Nach Springberg et al. [28] lassen sich Pooling-Layer auch durch Faltungsschichten mit einer Schrittgröße > 1 darstellen. Man spricht dann vom sogenannten *All Convolutional Network* (kurz: ACN). Durch ACNs können zwar auch keine verworfenen Informationen wiederhergestellt werden, jedoch lassen sich, bei einem umgedrehten ACN, im Trainingsprozess Gewichtungen/Regeln für das Upscaling erlernen.

2. „Use batchnorm in both the generator and the discriminator.“ (Radford et al. [23])

Bei der *Batch Normalization* [7] werden die Batches zwischen den Layern immer wieder normalisiert, was Trainingsprobleme aufgrund einer unvorteilhaften Zusammensetzung der Batches verhindern kann. Die *Batch Normalization* [7] wird zwischen allen Layern des Generators und Diskriminators angewandt mit der Ausnahme, dass vor der Ausgangsschicht des Generators und nach der Eingangsschicht des Diskriminators keine Normalisierung stattfindet, da es sonst leicht zur Oszillation von Beispielen kommen kann.

3. „Remove fully connected hidden layers for deeper architectures.“ (Radford et al. [23])

Bei voll verbundenen Schichten sind alle Neuronen eines Layers mit denen des darüber folgenden Layers verbunden, was die Anzahl der Gewichtungen, die im Trainingsprozess angepasst werden müssen, enorm erhöht und die Trainingszeit somit verlängert.

4. „Use ReLU activation in generator for all layers except for the output, which uses Tanh.“ (Radford et al. [23])

$ReLU(x) = \max(0, x)$	$leakyReLu(x) = \begin{cases} x & , x \geq 0 \\ s * x & , \text{sonst} \end{cases}$
$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$sigmoid(x) = \frac{1}{1+e^{-x}}$

Tabelle 1: Aktivierungsfunktionen

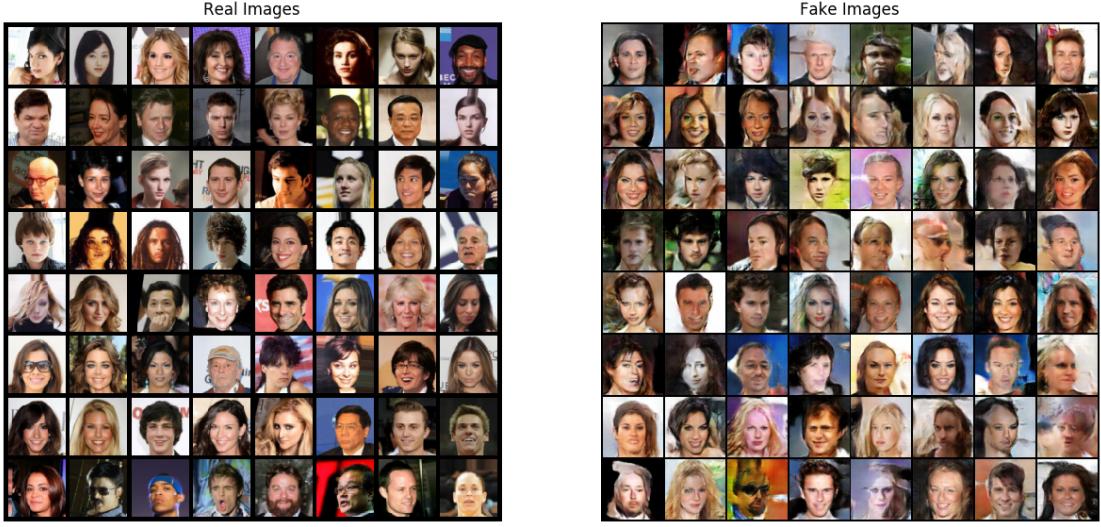


Abbildung 7: links reale Gesichter (Beispielbilder aus Trainingsdatensatz), rechts durch DCGAN erzeugte Bilder

Es konnte von Radford et al. [23] beobachtet werden, dass der Generator schneller lernt, wenn es sich bei der Aktivierungsfunktion im letzten Layer des Netzes um eine Funktion mit eingeschränktem Wertebereich handelt. Für den Tangens hyperbolicus (vgl. Tabelle 1) ist dies gegeben, da für den Wertebereich gilt: $-1 < \tanh(x) < 1$.

5. „Use LeakyReLU activation in the discriminator for all layers.“ (Radford et al. [23])

Die LeakyReLU-Funktion (vgl. Tabelle 1) wird in allen Layern des Diskriminators (mit Ausnahme des Ausgangslayers) verwendet, da sie vor allem bei größeren Auflösungen für bessere Ergebnisse gesorgt hat. Im Ausgangslayer wird die Sigmoid-Funktion verwendet, welche durch $0 < \text{sigmoid}(x) < 1$ ebenfalls einen eingeschränkten Wertebereich aufweist (vgl. Tabelle 1).

Mithilfe eines Bilddatensatz, wie z.B. dem CelebFaces Attributes Dataset (kurz: CelebA) [15], welcher über 200.000 Gesichter enthält, lässt sich ein DCGAN trainieren, welches in der Lage ist neue Gesichter zu generieren (vgl. Abbildung 7).

2.2. Erzeugung dreidimensionaler Objekte

Dreidimensionale Objekte werden in der Regel grundlegend durch eine Punktwolke repräsentiert, auf Basis derer dann ein Mesh erzeugt werden kann. Zweidimensionale Bilder ha-

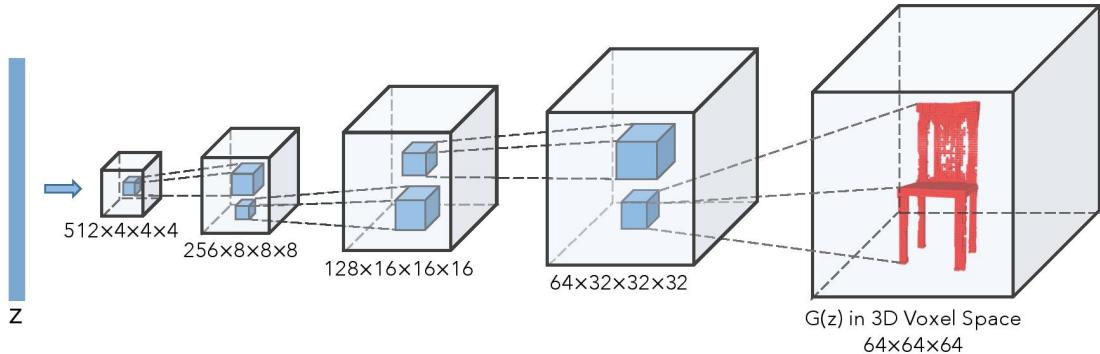


Abbildung 8: Generator des 3D-GAN von Wu et al. [30]

ben gegenüber dreidimensionalen Punktwolken den Vorteil, dass diese in jeder Dimension diskret sind (es gibt diskrete Positions- und Farbwerte) und sich somit Convolutional-Layer anwenden lassen, was bei Punktwolken ohne Weiteres nicht möglich ist. Da die Verwendung von Convolutional-Layer jedoch eine weit verbreitete Methode ist, um den Trainingsprozess von GANs zu beschleunigen und zu stabilisieren, setzten Wu et al. [30] bei der Erstellung ihres 3D-GANs auf Voxel statt Punktwolken. Voxel sind das 3D-Äquivalent zu Pixeln im 2D-Raum und ebenfalls diskret in jeder Dimension. Durch die Verwendung von Voxeln konnten Wu et al. [30] auf dem DCGAN von Radford et al. [23] aufbauen. Der größte grundlegende Unterschied liegt dabei in der Verwendung von 3D-Kernen in den Convolutional-Layern (vgl. Abbildung 8).

Der Diskriminatator spiegelt im Grunde den in Abbildung 8 abgebildeten Generator, was impliziert, dass der Trainingsdatensatz aus Voxelmodellen bestehen muss. Das Erstellen eines Datensatzes aus Voxelmodellen ist jedoch mit weit mehr Aufwand verbunden als das eines Bild-Datensatzes. Da dennoch Tiefeninformationen in den Trainingsdaten benötigt werden, ist möglicherweise die Verwendung von RGBD-Bildern ein akzeptabler Kompromiss.

2.3. Semantische Bearbeitung im latenten Raum

Um mit einem GAN Avatare erzeugen zu können, ist es notwendig, dass sich die Ausgabe des Generators beeinflussen lässt. Der Versuch, den Generator nach dem Trainingsprozess über den Eingangsvektor zu steuern, ist ein semantischer Prozess, da das Ergebnis einer bestimmten Aktion im latenten Raum erst im durch den Generator erzeugten Ausgangsbild sichtbar wird. Im Folgenden wird dieser Prozess genauer beschrieben.

2.3.1. Vektorarithmetik im latenten Raum

Bei einem GAN, welches darauf trainiert wurde neue Gesichter zu erzeugen, ist der Eingangsvektor z (des Generators) verantwortlich für das Aussehen des Gesichtes. Durch unterschiedliche Eingangsvektoren entstehen unterschiedliche Gesichter. Da der Vektor in der Regel ein zufälliges Rauschen enthält bzw. auf Basis dessen erstellt wird, wird er auch als *latent* bezeichnet, was bedeutet, dass aus ihm keine direkten Informationen entnommen werden können.

Radford et al. [23] stellten sich die Frage nach der Repräsentation von bestimmten Gesichtsmerkmalen innerhalb des Vektors z . Durch Vektorarithmetik konnten sie in einem Experiment feststellen, dass die Merkmale innerhalb des Vektors strukturiert abgebildet sein müssen. Sie versuchten bestimmte Merkmale eines Gesichts, wie beispielsweise ein *Lachen* oder das *Tragen einer Brille* von einem Geschlecht auf ein anderes zu übertragen (vgl. Gleichung 5). Voraussetzung dafür ist, dass der Eingangsvektor zu den im Experiment vorkommenden Bildern bekannt ist, weshalb nur Bilder verwendet werden können, die vom Generator erzeugt worden sind.

$$\begin{aligned} z(\text{lachende Frau}) - z(\text{nicht lachende Frau}) + z(\text{nicht lachender Mann}) \\ = z(\text{lachender Mann}) \end{aligned} \quad (5)$$

Gemäß der Annahme, dass Punkte z_i , die ein bestimmtes Attribut beschreiben, in Z eine Punktwolke bilden, kann es zur Verrechnung hilfreich sein, einen Mittelwert aus mehreren bekannten Punkten mit der selben Attributbeschreibung zu bilden, um so einen Punkt aus der Mitte der Punktwolke zu erhalten (vgl. Gleichung 6).

$$\begin{aligned} \frac{\sum_{i=1}^3 z_i(\text{lachende Frau})}{3} - \frac{\sum_{i=1}^3 z_i(\text{nicht lachende Frau})}{3} \\ + \frac{\sum_{i=1}^3 z_i(\text{nicht lachender Mann})}{3} = z(\text{lachender Mann}) \end{aligned} \quad (6)$$

Abbildung 9 und Abbildung 10 zeigen die Auswirkung von Gleichung 6 auf die Bildausgabe des in Unterabschnitt 2.1 beschriebenen DCGANs (trainiert mit dem CelebA-Datensatz [15]). Es wird dabei deutlich, dass es, über eine gezielte Änderung des Eingangsvektors $z \in Z$, möglich ist, das Bild zu manipulieren. Problemtisch ist dabei allerdings, dass der Raum Z der entsprechenden Verteilung im Datensatz folgt. Für den Fall, dass ein Datensatz z.B. nur aus Gesichtern mit dunklen Haaren besteht, gibt es im Raum Z keine Abbildungsmöglichkeit für blonde Haare. Zudem ist nicht direkt ersichtlich, welche Eigenschaft wie abgebildet wird.

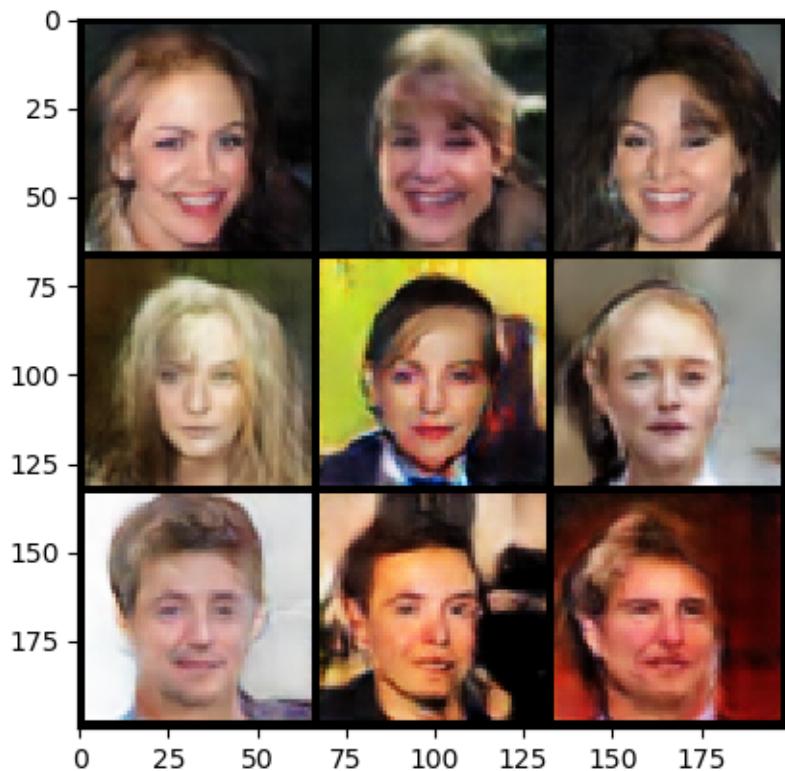


Abbildung 9: Je drei Bilder für die Kategorien *lachende Frau*, *nicht lachende Frau* und *nicht lachender Mann*

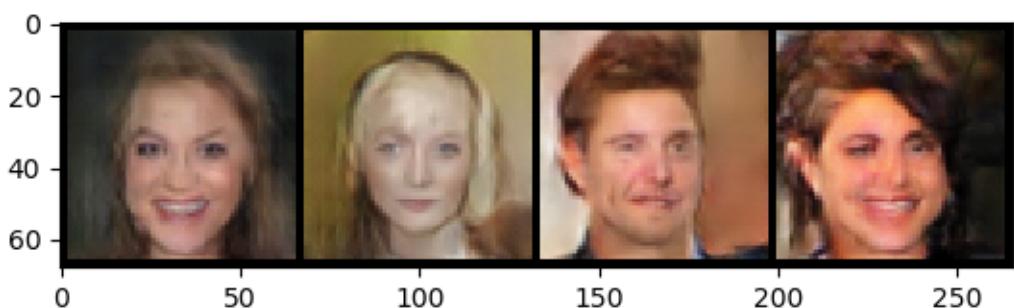


Abbildung 10: Drei Durchschnittsbilder der Kategorien *lachende Frau*, *nicht lachende Frau* sowie *nicht lachender Mann* und das Ergebnis der Verrechnung nach Gleichung 6

Im einem zweiten Experiment konnte gezeigt werden, dass die Pose, in der ein Gesicht auf einem Bild zu sehen ist (also z.B. nach links oder rechts schauend), in Z linear abgebildet ist.

2.3.2. Analyse des latenten Raumes von trainierten GANs

Aufbauend auf der entdeckten *Vektorarithmetik im latenten Raum* (vgl. Unterunterabschnitt 2.3.1) von Radford et al. [23] wurde der latente Raum von Shen et al. [26] weiter untersucht. Das Ziel dabei war es, diesen soweit zu verstehen, dass eine Bearbeitung von Gesichtern im latenten Raum mit „präziser Kontrolle von Gesichtseigenschaften“ (übersetztes Zitat von Shen et al. [26]) möglich sein sollte.

Über die Darstellung bestimmter Eigenschaften im latenten Raum können laut Shen et al. [26] zwei grundlegende Eigenschaften angenommen werden:

1. Da Radford et al. [23] zwischen zwei latenten Vektoren z_1 und z_2 erfolgreich interpolieren konnten, geht aus z_1 und z_2 eine Richtung in Z hervor, entlang der eine bestimmte Eigenschaft beschrieben ist.
2. Da einige Eigenschaften, wie z.B. das *Geschlecht* oder das *Tragen einer Brille* (meist) binär sind, muss es eine Grenze entlang der Richtung aus z_1 und z_2 geben.

Es wird angenommen, dass es eine Hyperebene im latenten Raum gibt, welche die beiden Fälle einer binären Eigenschaft von einander trennt. Die Hyperebene besitzt dabei einen Normaleneinheitsvektor $n \in \mathbb{R}^d$, wobei d die Anzahl der Dimensionen des Raumes Z darstellt ($Z \subseteq \mathbb{R}^d$). Der Abstand zur Ebene steuert dabei die Ausprägung der entsprechenden Eigenschaft, sofern dies möglich ist. Für die Eigenschaft *Tragen einer Brille* könnte auf der einen Seite der Ebene der Status *keine Brille* und auf der anderen der Status *trägt Brille* abgebildet sein. Über den Abstand zur Ebene könnte sich z.B. die Art der Brille ändern lassen. Es könnte durchaus sein, dass eine Sonnenbrille *mehr* Brille ist als eine Lesebrille o.Ä..

Der Abstand zur Hyperebene lässt sich dabei durch die Hessesche Normalform errechnen (vgl. Gleichung 7). Der Abstand ist zudem vorzeichenbehaftet. Ändert sich das Vorzeichen des Abstands, wird auch die entsprechende Eigenschaft invertiert, da z dann auf der anderen Seite der Ebene liegen würde.

$$d(n, z) = n^T z \quad (7)$$

Die Verknüpfung zwischen der Ausprägung einer Eigenschaft lässt sich durch die Gleichung 8 beschreiben. Dabei entspricht $g(z)$ einem vom Generator erzeugten Bild x ($g : Z \rightarrow X$), $f(x)$ einer Funktion, welche die Ausprägung einer bestimmten Eigenschaft im Bild in einen Vergleichswert s (*Scoring*) umwandelt ($f : X \rightarrow S$) und λ einem Skalierungsfaktor, der das Verhältnis zwischen *Scoring* und *Distance* beschreibt.

$$f(g(z)) = \lambda d(n, z) \quad (8)$$

Für die manuelle Anpassung eines Attributs gilt somit:

$$z_{edit} = z + \alpha n \quad (9)$$

bzw.

$$f(g(z_{edit})) = f(g(z)) + \lambda \alpha \quad (10)$$

Bei genauerer Betrachtung von Abbildung 10 fällt auf, dass sich neben dem Attribut *Lachen*, welches bewusst verändert werden soll, sich auch die Frisur ändert. Da Gesichter nicht nur ein, sondern m viele semantische Attribute enthalten und die oben genannten Funktionen für jedes Attribut einzeln Anwendung finden, gilt zusammenfassend Gleichung 11.

$$s \equiv f(g(z)) = \Lambda N^T z \quad (11)$$

Die Vektoren s und N enthalten so viele Einträge wie Attribute in dem Bild vorhanden sind ($s = [s_1, \dots, s_m]^T$ und $N = [n_1, \dots, n_m]$). Λ ist dementsprechend eine diagonale Matrix, welche für jedes Attribut einen Skalierungsfaktor enthält ($\Lambda = diag(\lambda_1, \dots, \lambda_m)$).

Problematisch ist, dass sich die Attribute nur einzeln manipulieren lassen, wenn alle Einträge in N orthogonal zu einander sind. Das diese Bedingung beim Trainingsprozess des GANs erfüllt wird, ist extrem unwahrscheinlich und auch bei dem DCGAN, welches hinter Abbildung 10 steckt, nicht der Fall.

Shen et al. [26] fanden in der Projektion von Vektoren eine Möglichkeit, eine Richtung im latenten Raum ausfindig zu machen, entlang der sich nur eine Eigenschaft ändert. In Abbildung 11 sind (vereinfacht) zwei Hyperebenen mit den entsprechenden Normalenvektoren n_1 und n_2 abgebildet. Projiziert man n_1 auf n_2 (Skalarprodukt aus n_1 und n_2 multipliziert mit n_2) und subtrahiert den projizierten Vektor von n_2 , erhält man einen Vektor p_{n_1} der orthogonal zu n_2 ist (vgl. Abbildung 11 und Gleichung 12).

$$p_{n_1} = n_1 - (n_1^T n_2)n_2 \quad (12)$$

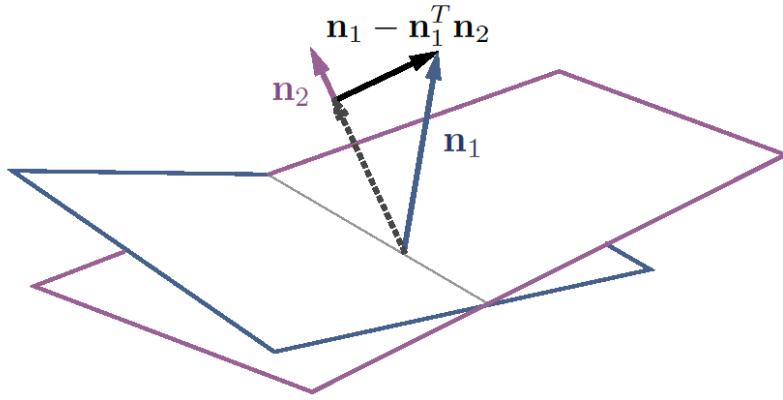


Abbildung 11: Vektorprojektion von Shen et al. [26] (Beachte Gleichung 12)

Entlang der Richtung von p_{n_1} wird das selbe Attribut geändert wie in der Richtung von n_1 , jedoch ohne dabei Einfluss auf das Attribut zu nehmen, welches von n_2 beeinflusst wird.

In vielen Fällen sind jedoch mehr als zwei Attribute miteinander verknüpft. In diesem Fall muss n_1 nicht auf n_2 , sondern die Ebene, die sich aus allen Einfluss nehmenden Normalenvektoren bilden lässt, projiziert werden.

2.4. Konditionierung

Da ein GAN sämtliche Zusammenhänge lernt, auch jene, die nicht offensichtlich sind, ist es schwierig durch semantische Bearbeitung nur ein Attribut zu verändern. Auch wenn ein Attribut erfolgreich verändert werden kann, werden oft weitere Eigenschaften ungewollt mit verändert.

Shen et al. [26] entdeckten zwar eine Methode, die Attribute beim Anwenden von Vektorarithmetik im latenten Raum unabhängig voneinander verändert, jedoch ist diese mit hohem Aufwand verbunden. Da für jedes Attribut zunächst eine Richtung im latenten Raum gefunden werden muss, entlang der, dieses verändert werden kann. Anschließend muss diese Richtung auf eine Hyperebene projiziert werden, innerhalb der alle anderen Attribute konstant sind. Da das ganze relativ kompliziert ist und individuell für jeden Datensatz und Trainingsstand durchgeführt werden muss, ist die Methode zur Beeinflussung der Generatorausgabe eher ungeeignet.

Besser wäre es, vor dem Trainingsprozess eine Steuerungsmöglichkeit festzulegen, welche das GAN im Trainingsprozess lernen soll zu berücksichtigen. Das GAN würde so auf die Steuerung konditioniert werden.

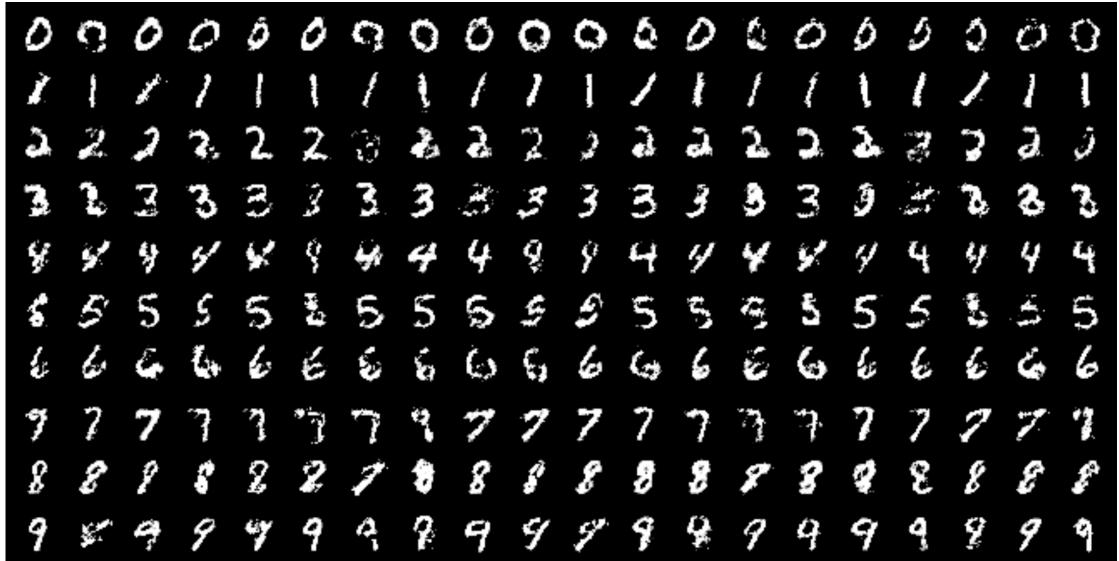


Abbildung 12: Generierte MNIST-Ziffern von Mirza et al. [19] (Jede Zeile wurde mit einem Label konditioniert.)

2.4.1. Conditional-GAN

Beim *Conditional-GAN* (kurz: *cGAN*) von Mirza et al. [19] wird die Eingabe in Generator und Diskriminator je um ein Label y erweitert. Das von Goodfellow et al. [6] in Gleichung 3 beschriebene *Minimax-Game* ändert sich somit wie folgt ab:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x|y))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - (D(G(z|y)))] \quad (13)$$

Mithilfe des Labels lassen sich später Bilder aus einer bestimmten Kategorie y erstellen. Wird das cGAN beispielsweise mit dem MNIST-Datensatz von LeCun et al. [13] trainiert, welcher aus $28 * 28$ Pixel großen Bildern von handgeschriebenen Ziffern besteht, lassen sich durch das Label y neue Bilder von einer bestimmten Ziffer erzeugen (vgl. Abbildung 12).

Das Label y wird von Mirza et al. [19] im MNIST-Experiment (vgl. Abbildung 12) durch einen 10-dimensionalen Vektor dargestellt, welcher immer nur in einer Dimension den Wert 1 enthält. Alle anderen Dimensionen enthalten den Wert 0. Diese Art von Vektor wird auch *One-Hot-Vektor* genannt. Die einzelnen Dimensionen repräsentieren die Labelklassen (hier die Ziffern 0 bis 9). Der One-Hot-Vektor wird mit dem Noisevektor z (vgl. Abschnitt 2) verbunden. Bei einem 100-dimensionalen Rauschvektor wäre der komplette Eingangsvektor für den Generator somit 110-dimensional. Das GAN lernt die Verkettung von Labelvektor und Rauschvektor im Trainingsprozess automatisch zu interpretieren.

2.4.2. Pix2Pix-GAN

Der Ansatz der Konditionierung von GANs von Mirza et al. [19] wird beim *Pix2Pix-GAN* von Isola et al. [8] aufgegriffen und dahingehend erweitert, dass auf einen Rauschvektor z als Eingabe komplett verzichtet wird (s. Tabelle 2).

Architektur	Generator-Mapping
GAN (Goodfellow et al. [6])	$G : z \rightarrow y$
cGAN (Mirza et al. [19])	$G : \{x, z\} \rightarrow y$
Pix2Pix-GAN (Isola et al. [8])	$G : x \rightarrow y$

Rauschvektor: z , Ausgangsbild: y , Label: x

Tabelle 2: Mapping des Generators in GAN, cGAN und Pix2Pix-GAN

Grundsätzlich lernt eine GAN-Architektur einen Raum auf einen anderen Raum zu übertragen (engl. *mappen*). Ausgehend vom GAN nach Goodfellow et al. [6] bedeutet dies, dass am Ende des Trainingsprozesses die Bilder des Datensatzes als Beispiele von gemappten Punkten aus dem latenten Raum Z anzusehen sind. Während des Trainingsprozesses lernt das GAN den kompletten latenten Raum Z zu interpretieren, was es am Ende ermöglicht neue Bilder zu erzeugen, welche eine starke Ähnlichkeit mit den Bildern aus dem Datensatz aufweisen.

Ohne den Rauschvektor z als Eingangsvektor gibt es auch keinen latenten Raum Z (da $z \in Z$). Wird der im Rauschvektor enthaltene stochastische Aspekt nicht kompensiert, lernt das GAN nur die Trainingsbeispiele auswendig. Alle Eingaben, die von den Trainingsdaten abweichen, würden mit extrem hoher Wahrscheinlichkeit zu unzureichenden Ergebnissen führen. Die Ausgabe der gewünschten Ergebnisse kann somit als deterministisch betrachtet werden (Isola et al. [8]).

Durch die Verwendung einer U-Net-Architektur, basierend auf dem *U-Net* von Ronneberger et al. [24], in Kombination mit *Dropouts* kann der stochastische Aspekt sowie fehlende latente Raum anderweitig in den Generator integriert werden.

Das U-Net von Ronneberger et al. [24] ist im Grunde eine Auto-Encoder-Architektur mit *Skip-Connections* (vgl. Abbildung 13). Die Auto-Encoder-Architektur besteht aus zwei Teilen: einem *Encoder* und einem *Decoder*. Durch sie wird der latente Raum in den Generator eingebracht. Der *Encoder* lernt die Eingaben x in einen latenten Raum Z zu mappen ($Encoder : x \rightarrow z \in Z$). Der Decoder lernt aus einem Punkt z im latenten Raum ein Bild y zu generieren ($Decoder : z \rightarrow y$).

Die *Skip-Connections* werden zwischen jedem i -ten und $(n - i)$ -ten Layer eingefügt (vgl. Abbildung 13). Durch die *Skip-Connection* wird dem Generator ermöglicht, für bestimm-

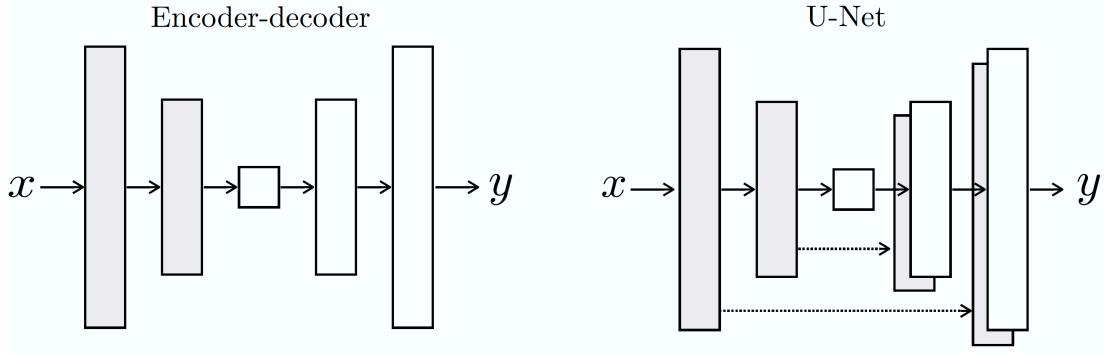


Abbildung 13: Vergleich von Isola et al. [8]: Auto-Encoder (links) vs. U-Net (rechts)

te Strukturinformationen den *Encoding-Decoding*-Prozess teilweise zu umgehen. Welche Strukturinformationen dies genau sind, lernt das Netz von sich aus im Trainingsprozess. Auto-Encoder haben oft das Problem, dass die decodierten Bilder deutlich unschärfer sind als die Eingabebilder in den Encoder. Durch die Verwendung der *Skip-Connections* und einer Loss-Funktion aus L_{cGAN} (vgl. Gleichung 14)¹ und $L1$ (vgl. Gleichung 15) kann dieses Problem behoben werden (vgl. Gleichung 16 und Abbildung 14). Über eine Gewichtungskonstante λ wird das Verhältnis der beiden Teil-Losses bestimmt.

$$L_{cGAN}(D, G) = \mathbb{E}_{x,y}[\log(D(x|y))] + \mathbb{E}_x[\log(1 - (D(x, G(x)))] \quad (14)$$

$$L_1(G) = \mathbb{E}_{x,y}[||y - G(x)||_1] \quad (15)$$

$$L_{pix2pix}(D, G) = L_{cGAN}(D, G) + \lambda L_1(G) \quad (16)$$

Im GAN nach Goodfellow et al. [6] sorgt ein Rauschvektor als Eingabe dafür, dass später beliebige Eingaben zu neuen Ergebnissen führen. Da es in der U-Net-Architektur einen solchen Rauschvektor nicht gibt, wenden Isola et al. [8] auf den Layern des Decoders eine *Dropout-Rate* von 50% an. Die Hälfte aller Verbindungen zwischen den Layern im Decoder-Teil des Generators fällt somit zufällig aus. Der Generator muss so im Trainingsprozess trotz eines extrem hohen und zudem zufälligen Datenverlustes lernen, gute Ergebnisse zu produzieren, was der Verwendung eines Rauschvektors nahekommt.

Der Diskriminator des Pix2Pix-GANs erhält das selbe Eingangsbild x wie der Generator sowie dessen Ausgangsbild $y_{fake} = G(x)$ bzw. das zu x passende Bild y_{real} aus dem Datensatz. Dies entspricht im Grunde der Idee des cGANs (vgl. Unterunterabschnitt 2.4.1),

¹Entspricht nicht dem cGAN aus Unterunterabschnitt 2.4.1 (kein Rauschvektor), sondern drückt nur den konditionellen Anteil des Pix2Pix-GANs aus.



Abbildung 14: Auswirkung der Skip-Connections und Loss-Funktion auf die Qualität der Ausgabe im Pix2Pix-GAN von Isola et al. [8]

bei dem nicht nur das Ergebnis des Generators, sondern auch dessen Unterschied zur Eingabe bewertet wird. Anders als beim cGAN ist die Ausgabe des Diskriminators kein Skalar, das über *echt* oder *falsch* entscheidet, sondern eine Matrix. Durch die Verwendung von Convolutional-Layern (vgl. Radford et al. [23]), repräsentiert jeder Eintrag in der Ausgangsmatrix einen $n * m$ -großen² Bereich (*Patch*) des Eingangsbildes. Isola et al. [8] sprechen deshalb auch von einem *PatchGAN*. Jedes *Patch* wird in der Matrix separat bewertet, wobei sich die Patches im Ausgangsbild überlappen³ und einzelne Pixel des Ausgangsbildes so oft mehrfach in die Bewertung eingehen können. Die in Gleichung 16 beschriebene Loss-Funktion muss für jedes Patch angewandt werden, bevor der endgültige Loss-Wert über alle Werte gemittelt wird (vgl. Gleichung 17)⁴.

$$L_{PatchGAN}(D, G) = \frac{1}{I * J} \sum_{i=0}^I \sum_{j=0}^J L_{pix2pix_{i,y}}(D, G) \quad (17)$$

²bei Verwendung von quadratischen Kerneln in den Convolutional-Layern entspricht $m = n$

³Grund dafür ist *stride < kernelSize* in den Convolutional-Layern

⁴ I und J in Gleichung 17 beschreiben die Größe der Ausgangsmatrix

Abbildung 15 und Abbildung 16 zeigen einen Beispielaufbau von Generator und Diskriminatoren im Pix2Pix-GAN. Die grauen Pfeile in Abbildung 15 symbolisieren einen Downsampling-Convolutional-Layer (*Downsamplingrate* = 2), die blauen Pfeile entsprechen dagegen den Upsampling-Convolutional-Layern (*Upsamplingrate* = 2). Die Convolutional-Layer treten wie im DCGAN von Radford et al. [23] immer in Kombination mit einer ReLu-Funktion (Downsampling/Encoder) bzw. Leaky-ReLu mit einem Slope von 0.2 (Upsampling/Decoder) sowie einer Batch Normalization [7] (außer im ersten Layer) auf (vgl. Unterabschnitt 2.1). Skip-Connections werden durch schwarze (senkrechte) Pfeile dargestellt. Sie verändern die Daten nicht und geben diese 1:1 wieder. Die Featuremaps zwischen den Layern werden durch die grauen, blauen sowie gelben Blöcke repräsentiert. Anhand der gelben Blöcke wird deutlich, wie die Daten, welche durch Skip-Connections gewisse Layer überspringen, wieder in den eigentlichen Durchlaufpfad eingegliedert werden. Die Featuremaps aus zwei Quellen werden gestapelt. Gegenüber dem Bilden einer Summe hat dies den Vorteil, dass im darauf folgenden Convolutional-Layer die Featuremaps aus unterschiedlichen Quellen innerhalb der Kernel unterschiedlich gewichtet werden können (vgl. Abbildung 28).

Die Eingabe der Bilder in den Diskriminatoren erfolgt ähnlich wie das Wiedereingliedern von Daten aus Skip-Connections im Generator (vgl. Abbildung 16).

In der Ausgangsschicht des Generators kommt eine Tanh-Funktion zum Einsatz, um alle Ausgangswerte immer im Bereich $[-1; 1]$ zu halten. Im Diskriminatoren wird die Sigmoid-Funktion verwendet, welche die Ausgabe im Bereich $[0; 1]$ ermöglicht.

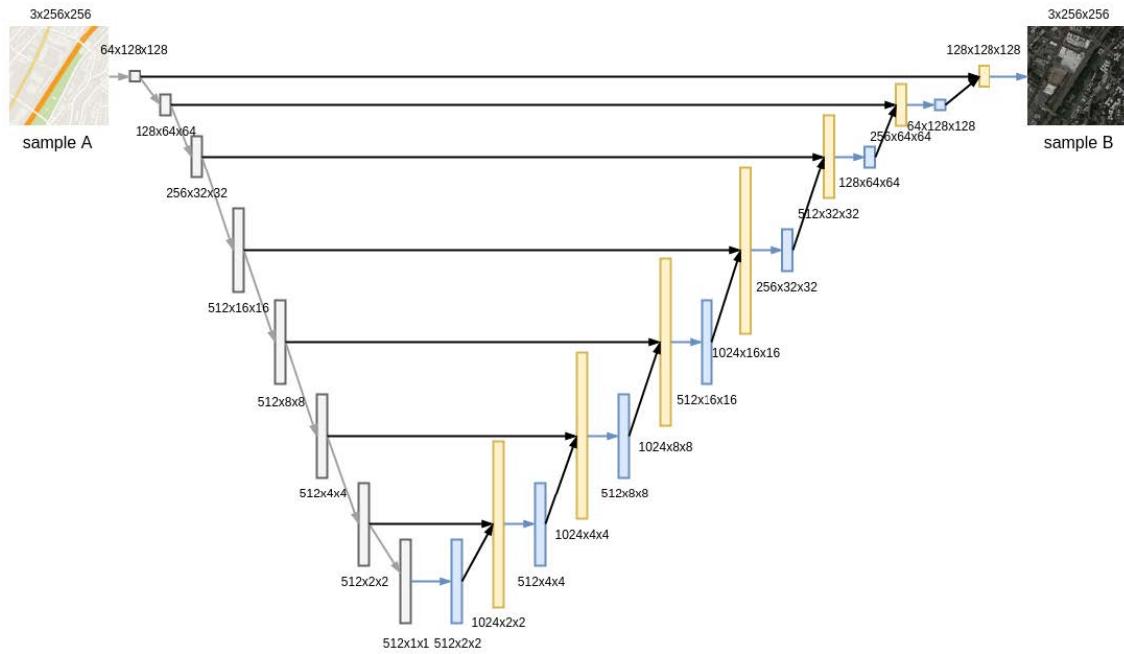


Abbildung 15: Generator im Pix2Pix-GAN für 256*256px große RGB-Bilder [9]

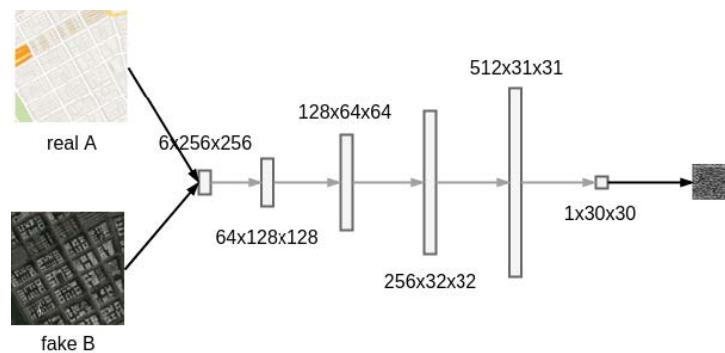


Abbildung 16: Diskriminatator im Pix2Pix-GAN für 256*256px große RGB-Bilder [9]

3. Anforderungen an das Erzeugen von Avataren in Bezug auf GANs

Das Erzeugen eines Avatars lässt sich grundlegend in zwei Aufgaben aufteilen: das Erstellen eines Models und das Kreieren einer Steuerung.

Beim Erstellen eines Avatars gibt es eine Reihe von Eigenschaften, die vorab definiert werden sollten:

1. Soll es sich um einen zweidimensionalen oder dreidimensionalen Avatar handeln?

In einem zweidimensionalen Raum können Avatare in Form von Bildern repräsentiert werden. In den letzten Jahren hat sich durchaus herausgestellt, dass sich vor allem Generative Adversarial Networks zur Erzeugung von Bildern eignen (s. Abschnitt 2). Zweidimensionale Räume bieten aber nur bedingt einen Vorteil gegenüber einer Videokonferenz, weshalb die Verwendung eines dreidimensionalen Raumes erst den eigentlichen Mehrwert bietet.

2. Wie real sollen die Avatare aussehen?

Ein wichtiger Fokus bei zwischenmenschlicher Interaktion liegt auf dem Erscheinungsbild. Das Erscheinungsbild eines menschlichen Avatars muss sehr detailliert sein, damit seine Erscheinung als *vergleichbar mit der Realität* aufgefasst wird. Da das Erstellen von realistischen Avataren viele komplexe Bereiche mit sich bringt (Haut, Haare, Kleidung), geht der Trend der Zeit hin zu nicht realistischen cartoonartigen Avataren (vgl. Zakharov et al. [31] und Abbildung 17). Abstrakte Avatare, die sich auf einige Kernmerkmale beschränken, werden in der Regel als angenehmer empfunden als photorealistische mit geringen Fehlern (Zakharov et al. [31]).

3. Welchen Umfang sollen die Avatare haben?

Gemäß dem Anwendungsfall gibt es zwei Optionen für Avatare in Bezug auf den Umfang: Gesichtsavatare und Ganzkörperavatare (vgl. Abbildung 17). In Bezug auf eine Umsetzung durch GANs würde das Erstellen eines Gesichtsavatars die Aufgabe zunächst deutlich erleichtern, da das Gesicht nur ein Teil des Körpers ist.

Zudem ist es denkbar, dass ein Ganzkörperavatar, der auf Basis einer Technologie erstellt wird, nicht das bestmögliche Ergebnis darstellt. Da es bei verschiedenen Umsetzungen immer Stärken und Schwächen gibt, ist es wahrscheinlich, dass durch eine Kombination verschiedener Techniken das beste Ergebnis erzielt werden kann (vgl. Abschnitt 1).

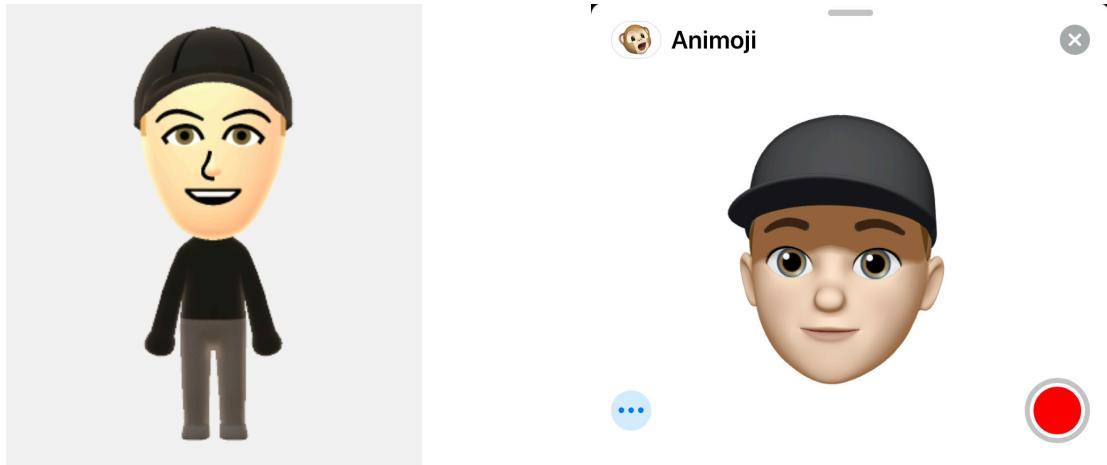


Abbildung 17: Beispiele für cartoon-artige Avatare (links: Mii (Nintendo), rechts: Animoji (Apple))

3.1. Erhalt der Identität

Generative Adversarial Networks können Bilder auf Basis eines Bilddatensatzes erzeugen. Das Ziel des Trainingsprozesses ist es, dass die erzeugten Bilder denen aus dem Datensatz so stark ähneln, dass sie sich nicht mehr unterscheiden lassen. Aus einem Datensatz, bestehend aus Gesichtern, lassen sich somit beispielsweise neue Gesichter erstellen. Die Gesichter sehen dabei jedoch in der Regel so aus, als ob sie von unterschiedlichen Personen stammen würden, was für das Erzeugen von Avataren ungeeignet ist (vgl. Abbildung 7).

Das Aussehen der Gesichter hängt von der Eingabe des Generators und dessen Aufbau ab. Eine Eingabe führt zu einem bestimmten Ergebnis. Im Grunde kann der Generator als eine Art Mapping-Network betrachtet werden, er lernt einer bestimmten Eingabe ein bestimmtes Bild zuzuordnen. Wie dieses Bild am Ende aussieht, hängt jedoch im Grunde von den im Trainingsprozess erlernten Gewichtungen ab und kann somit als zufällig betrachtet werden. Den Trainingsprozess genauer zu verstehen bzw. kontrollieren zu können, ist ein zentraler Teil in der Erforschung von GANs. Durch Änderungen an der Architektur wird oft versucht, diesen Prozess steuern zu können. Beim Conditional-GAN (cGAN) erhält der Generator beispielsweise zusätzlich zum Eingangsvektor ein Label, welches später zum Ausgangsbild passen soll (vgl. Unterunterabschnitt 2.4.1). Der Datensatz muss entsprechend gelabelt vorliegen.

Der konkrete Aufbau von Generator und Diskriminatator hat einen großen Einfluss auf die Qualität der erstellten Bilder. Dies wird anhand einiger wichtiger Entwicklungsstufen, wie DCGAN (Radford et al. [23] 2015), Progressive GAN (Karras et al. [11] 2017) und Style-GAN (Karras et al. [12] 2018) deutlich. Alle vier GANs haben gemeinsam, dass durch

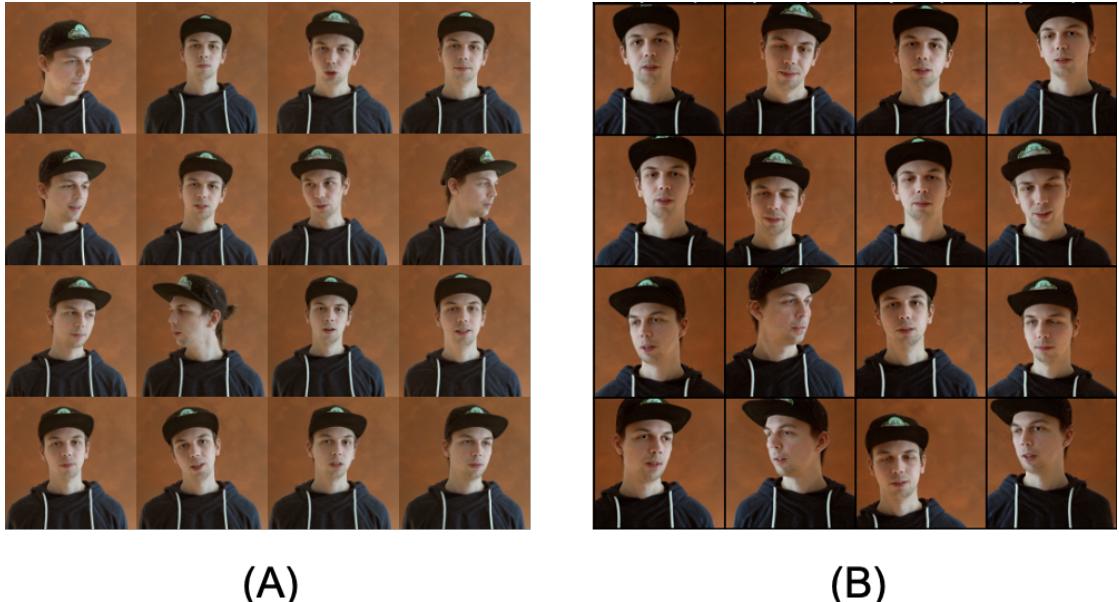


Abbildung 18: (A) Beispielbilder aus Datensatz, (B) durch Progressive-GAN erzeugte Bilder

Veränderung der Struktur von Generator und Diskriminatoren die Qualität der Ergebnisse erhöht werden konnte. Ein Einfluss auf die Identität der abgebildeten bzw. erzeugten Person durch den Aufbau des Generators (mit Eingabe von nur einem Eingangsvektor) wird dabei nicht erkennbar.

Eine einfache Möglichkeit die Identität zu beeinflussen liegt darin, einen Datensatz mit Bildern von nur einer Person zu verwenden. Zwangsläufig werden dann nur Bilder von dieser Person erzeugt (s. Abbildung 18). Für das Erstellen von Avataren ist dies eine denkbare Möglichkeit, wenn das Ziel eine GAN-Anwendung ist, bei welcher der Generator nach dem Trainingsprozess in Echtzeit Bilder erzeugen soll.

Ein Nachteil dieser Möglichkeit ist jedoch, dass für jeden Avatar ein neuer Datensatz erstellt und Trainingsprozess durchlaufen werden muss. Der Datensatz (A) in Abbildung 18 enthält 18447 Bilder. Um die Bilder in (B) zu erzeugen, wurde ein Progressive-GAN [14] mit einer NVIDIA 2080 TI 18h lang trainiert.

Eine alternative Methode zum Erhalt der Identität, welche bei bereits existierenden GAN-Anwendungen für eine ähnliche Problemstellung angewandt wird, ist in Unterabschnitt 3.3 beschrieben, da sie eng verknüpft mit dem Aspekt der Steuerung ist.

3.2. Steuerung durch Posen-Informationen

Um Avatare, in Form eines herkömmlichen 3D-Modells, in VR/AR steuern zu können, muss eine Schnittstelle zur Übertragung von Steuerungsinformationen geschaffen werden. Ähnlich wie bei Videospielen, in denen Charaktere durch Controller-, Tastatur- und/oder Mauseingaben gesteuert werden können. Der Unterschied zwischen Avataren und Videospielcharakteren in Bezug auf die Steuerung liegt in der Komplexität.

Bei Videospielcharakteren werden, durch einfache Eingaben, Animationen auf das 3D-Modell angewandt. Oft ist es z.B. so, dass sich eine Figur nur durch Drücken der vier Pfeiltasten auf der Tastatur im Spiel bewegen lässt - samt Lauf-Animation. Die Bewegungen, die ein Charakter im Spiel ausführen kann, sind dabei auf die vorab hinterlegten Animationen beschränkt. Dies soll bei der Verwendung von Avataren nicht so sein. Der Avatar soll im Grunde ein perfektes Abbild einer realen Person sein und sich auch genau so bewegen. Steuerungen durch herkömmliche Eingabegeräte wie Controller, Tastatur oder Maus sind daher eher ungeeignet. Geeigneter sind in diesem Fall Motion-Capture-Verfahren bei denen Informationen über eine bestimmte Pose in Form von Skelettdaten (oder bei Gesichtern Facial Landmarks) übergeben werden. In Abbildung 19 ist beispielhaft die Erkennung von Skelettinformationen mit OpenPose [5] von Cao et al. [4] inklusive der Gesichts- und Handerkennung von Simon et al. [27] zu sehen. Es können aber auch andere Motion-Capture-Systeme verwendet werden. Wichtig ist lediglich, dass die Daten zur weiteren Verarbeitung am Ende in einem einheitlichen Format vorliegen (einheitliche Anzahl von Joints/Landmarks, 2D oder 3D-Koordinaten ect.). Die aufgenommenen Bewegungen müssen dann auf das Skelett des Modells übertragen werden (vgl. Abbildung 20).

Avatare in Form eines Echzeit-Hologramms, wie in Abbildung 1 zu sehen, bedürfen keiner expliziten Steuermöglichkeit, da hier Identität und Pose zusammen aufgenommen werden.

Für die synthetische Erzeugung eines Teils des Avatars, in Bezug auf das in Abschnitt 1 beschriebene Verdeckungsproblem, sind Steuerinformationen für die relevanten Bereiche notwendig. Um das Gesicht rekonstruieren zu können, muss das GAN auf die Facial Landmarks konditioniert werden, damit es auf Basis einer Eingangsposse Ergebnisse mit der selben Pose liefert.

Im Vergleich zur Animation von herkömmlichen 3D-Modellen, für die, wie oben beschrieben, ein Skelett im Modell benötigt wird, ist dieses bei der Umsetzung durch ein GAN nicht notwendig, da das 3D-Modell nicht von Frame zu Frame verändert, sondern immer neu erstellt wird.

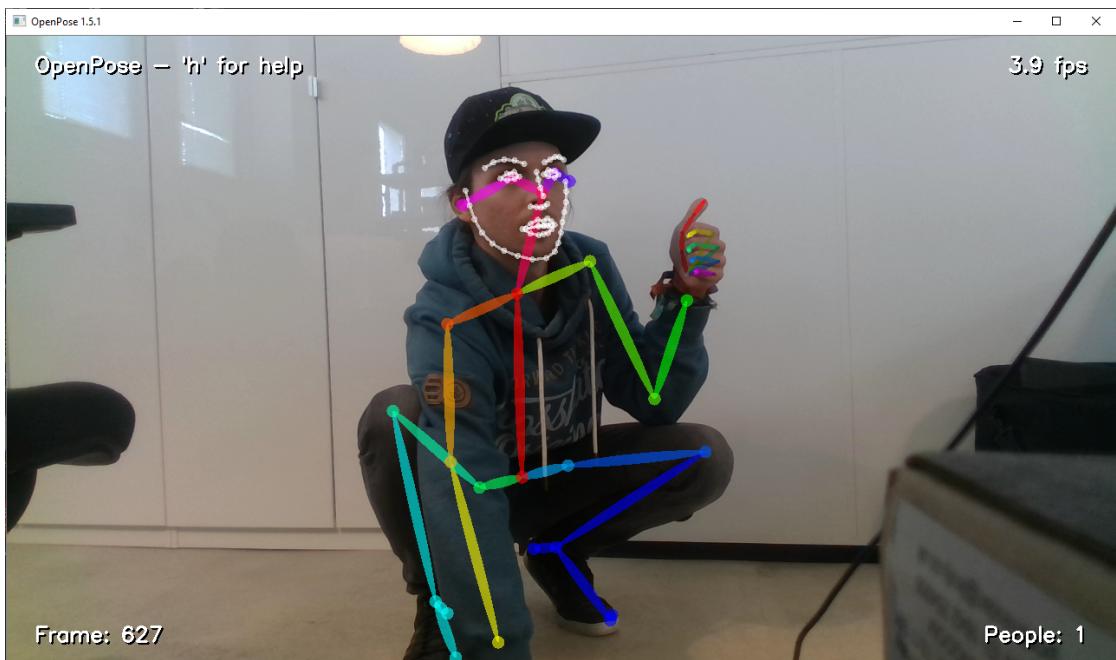


Abbildung 19: Skeletterkennung durch OpenPose [4] inkl. Gesichts- u. Handerkennung [27]

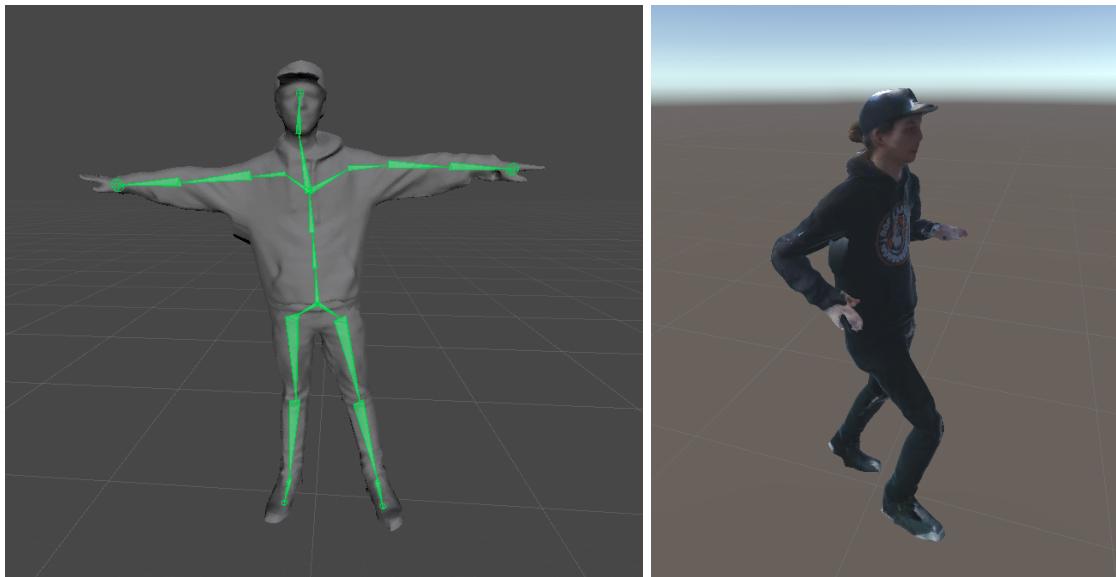


Abbildung 20: Animation durch Skelett (links: Verknüpfung Skelett mit Modell, rechts: Animation durch Positions- und Rotationsänderung der Joints des Skelettes)

3.3. Bisherige Umsetzungen in 2D

Gemäß dem Ausgangsproblem, dass auf eine vorgegebene und konstante Identität eine neue Pose angewandt werden soll, gibt es für eine Umsetzung im zweidimensionalen Raum bereits einige Ansätze in Form von GAN-Architekturen (vgl. Ma et al. [16], Pumarola et al. [22], Zhu et al. [32]).

Alle Ansätze haben dabei gemeinsam, dass im Trainingsprozess auf Basis von einem Bild I_A und Poseninformation P_B (oder P_A und P_B) ein Bild I_B versucht wird zu reproduzieren. I_A und I_B bilden dabei die selbe Identität in unterschiedlicher Pose ab. Der Vorgang wird mit so vielen Identitäten durchlaufen, bis das GAN die Identität scheinbar nur noch auf Basis des Eingangsbildes erzeugt, wodurch sich im Anschluss beliebige Posen auf ein unbekanntes Bild mappen lassen.

Die Eingabe einer Identität über ein Bild stellt einen enormen Vorteil gegenüber dem Vorschlag aus Unterabschnitt 3.1, einen personenspezifischen Datensatz zu verwenden, dar, benötigt dafür aber auch einen wesentlich größeren Datensatz. Gerade in Bezug auf die Erstellung des Datensatzes für eine dreidimensionale Umsetzung ist dies problematisch, da sich Trainingsbeispiele nicht aus bereits existierenden zweidimensionalen Videosequenzen extrahieren lassen.

3.4. Mögliche Umsetzung in 3D durch RGBD-Bilder

In Unterabschnitt 2.2 wurde bereits ein GAN-Architektur zur Erstellung dreidimensionaler Objekte in Form von Voxeln vorgestellt. Da das Erstellen eines Voxeldatensatzes im Vergleich zum Erstellen eines Bilddatensatzes wesentlich aufwendiger ist, könnte die Verwendung von RGBD-Bildern eine Alternative darstellen.

Die Verwendung von RGBD-Bildern liegt quasi zwischen der von Voxeln und 2D-Bildern. Neben Rot-, Grün- und Blauwerten enthält jeder Pixel einen zusätzlichen Tiefenwert. Die 3D-Präsentation ist schlechter als bei Voxeln, da es keine Rückseite gibt. Oft wird dabei deshalb auch von 2,5D geredet. Für den Anwendungsfall, dass, wie Abschnitt 1 beschrieben, ein Face-Avatar erstellt werden soll, welcher dann in ein bestehendes Modell eingefügt werden kann, ist diese Einschränkung akzeptabel.

RGBD-Bilder haben den Vorteil, dass sich die Tiefeninformation als vierter Farbwert interpretieren lässt. GANs, die für das Erstellen von 2D-Bildern geeignet sind, lassen sich so mit wenig Aufwand dahingehend abändern, dass sie RGBD-Bilder produzieren. Grund dafür ist die Verwendung von Convolutional-Layern. Bei der Verwendung von Convolutional-Layern werden Bilder durch Featuremaps repräsentiert. Eine Featuremap kann z.B. Informationen über einen Farbkanal oder im Bild auftretende Strukturen ent-

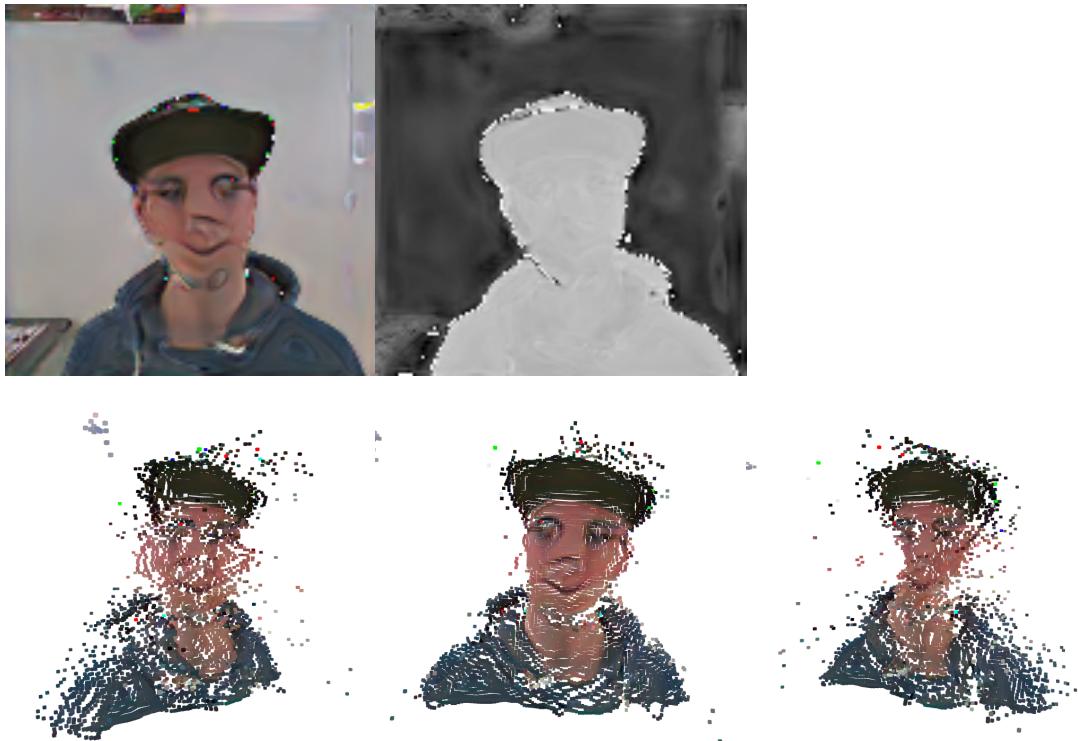


Abbildung 21: Oben: durch Progressive-GAN [14] erzeugtes RGBD-Bild (links: RGB, rechts: Tiefenbild), Unten: 3D-Darstellung des Bildes aus drei Perspektiven

halten. In der Ausgangsschicht des Generators werden bei den meisten GANs drei Featuremaps bestimmt, welche den drei Farbkanälen rot, grün und blau entsprechen. Jede der drei zweidimensionalen Featuremaps besitzt die Größe des Ausgangsbildes. Ändert man die Ausgangsschicht dahingehend ab, dass vier Featuremaps erstellt werden, kann die zusätzliche Featuremap für die Tiefeninformation verwendet werden.

Die in Abbildung 21 zu sehenden Bilder sind das Ergebnis eines Progressive-GANs [14] nach Karras et al. [11], welches lediglich dahingehend abgeändert wurde, dass die Ausgangsschicht des Generators vier Farbkanäle erzeugt und der Diskriminator vier Farbkanäle entgegennimmt. Beim Trainingsprozess muss entsprechend ein RGBD-Datensatz verwendet werden. Die Erstellung eines solchen Datensatzes ist in Unterunterabschnitt 4.1.1 beschrieben. Das GAN wurde mit 892 Bildern und einer NVIDIA 2080 Ti 30h lang trainiert.

4. Entwicklung einer GAN-Architektur zur Erzeugung von RGBD-Face-Avataren

Auf Basis der vorangegangenen Abschnitte soll nun eine GAN-Architektur entwickelt werden, welche realistische RGBD-Face-Avatare erstellen kann.

4.1. Trainingsdaten

Der Datensatz bildet die Grundlage für den Trainingsprozess. In den meisten Fällen besteht er aus mehreren tausend Bildern. Für das Erstellen eines Face-Avatars muss es sich dabei um Bilder von Gesichtern handeln. Um die Identität des Gesichtes auf dem Ausgangsbild des Generators konstant zu halten, kann der Ansatz eines personenspezifischen Datensatzes aus Unterabschnitt 3.1 angewandt werden. Die Verwendung eines Datensatzes mit Bildern von nur einer Person benötigt einen verhältnismäßig kleinen Datensatz. Um die Tiefenschätzung zunächst zu umgehen, müssen die Bilder im RGBD-Format vorliegen und um möglichst viele Ausdrücke zu lernen, sollte das Gesicht in vielen Posen im Datensatz enthalten sein.

Aufgrund der speziellen Anforderungen an den Datensatz muss dieser schon für Testzwecke selbst erstellt werden.

4.1.1. Aufnahme von RGBD-Bildern

Das Erstellen von RGBD-Bildern ist mit speziellen Tiefenbildkameras, wie z.B. der, im Folgenden verwendeten, Azure Kinect von Microsoft möglich (vgl. Abbildung 22). Neben einem 12MP-Sensor für das RGB-Bild besitzt die Kamera einen zweiten 1MP großen Infrarot-Sensor. Die Azure Kinect verwendet zur Tiefenbestimmung ein *Time-of-Flight (ToF)* Verfahren (vgl. docs.microsoft.com [29]). Dabei wird, von der Kinect ausgehend, moduliertes Licht im Infrarotbereich in die Szene geschickt. Mithilfe des IR-Sensors kann dann die Zeit gemessen werden, die das Licht, von der Kamera bis zu einem Objekt in der Szene und zurück braucht. Aus der gemessenen Zeit kann dann die Entfernung errechnet werden. Um einzelne Lichtstrahlen auseinander zu halten, werden die Amplituden der Lichtstrahlen unterschiedlich und eindeutig moduliert (Amplitude Modulated Continuous Wave (AMCW)) (vgl. docs.microsoft.com [29]).

Eine gängige Art RGBD-Bilder zu speichern ist es, Farb- und Tiefeninformationen zu trennen und in zwei separaten Dateien abzuspeichern. Meist wird dafür ein RGB8-PNG (drei Farbkanäle mit je 8Bit) und ein Gray16-PNG (ein Farbkanal mit 16Bit) erstellt. Die 16Bit für Tiefeninformationen sind notwendig, da sich mit 8Bit lediglich 256 Zustände



Abbildung 22: Azure Kinect DK von Mircosoft [17]



Abbildung 23: Beispiel aus RGBD Datensatz

abspeichern lassen. Bei der Azure Kinect werden die Tiefeninformationen in Millimeter angegeben (vgl. docs.mircosoft.com [29]), was bei 8Bit zu einem Dynamikumfang von nur 25,5 cm führen würde.

Beim Arbeiten mit Machine-Learning-Frameworks, wie z.B. Pytorch oder Tensorflow, muss darauf geachtet werden, dass die beiden PNG Bilder zwingend zusammen gehören, wenn Datensatzeinträge zufällig umverteilt werden, um *overfitting* zu verhindern.

4.1.2. Aufbereitung der Tiefendaten

Wie in Unterunterabschnitt 4.1.1 bereits beschrieben, ist es für den Aufnahmeprozess einfacher (und ggf. sogar notwendig für den Fall, dass es nicht nur um Gesichtsaufnahmen geht) die Tiefenwerte in 16Bit aufzunehmen. Im Fall von Gesichtsaufnahmen bewegt sich der aufgenommene Dynamikumfang in einem Bereich, der mit 8Bit in den meisten Fällen darstellbar wäre. Setzt man die Wertebereiche von 8Bit und 16Bit ins Verhältnis

zueinander (vgl. Gleichung 18) wird deutlich, dass ggf. nur rund 0,4% des möglichen Wertebereiches von 16Bit für die Tiefenbeschreibung verwendet werden.

$$\frac{2^8}{2^{16}} = \frac{256}{65536} = 0.00390625 \quad (18)$$

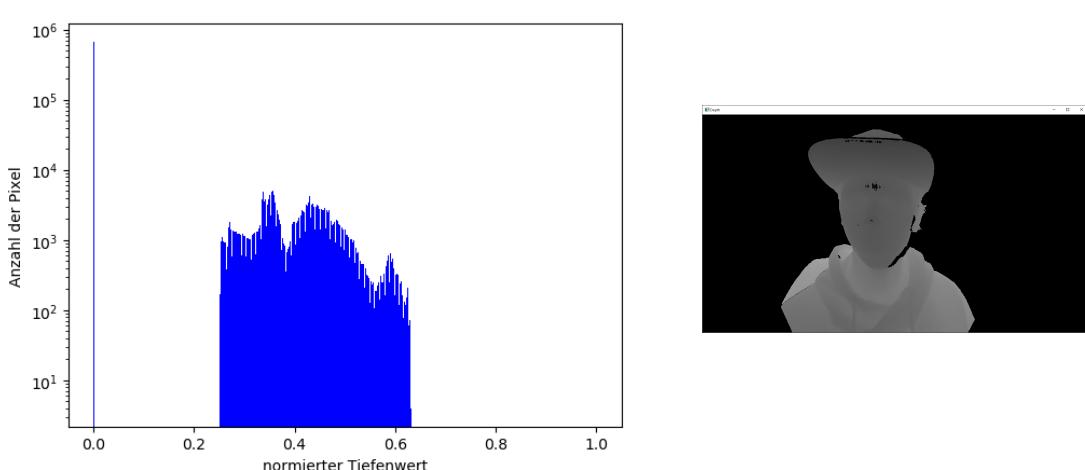
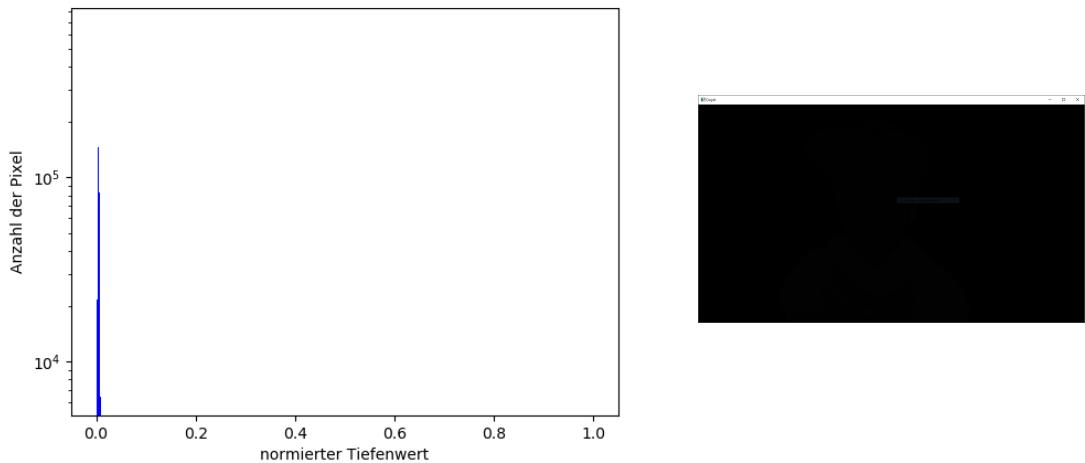
Für die Eingabe in das GAN werden alle Bilder normiert und alle Werte durch Gleitkommazahlen dargestellt. Beim Normieren verringert sich der Dynamikumfang von 0.4% absolut betrachtet weiter. Letztendlich ist der Dynamikumfang zu klein, um Informationen über die Struktur zu vermitteln (vgl. Abbildung 24).

Kontraste innerhalb eines Bildes lassen sich verschärfen, indem das Histogramm normalisiert wird. Beim Normalisieren der Histogramme aller Tiefenmaps können allerdings Informationen über die Tiefenänderung innerhalb von Bewegungen verloren gehen. Da alle Bilder anschließend den kompletten Wertebereich verwenden, kann es später bei Bewegungen zu Verzerrungseffekten in den Tiefenwerten kommen. Um dies zu vermeiden, kann über den kompletten Datensatz normalisiert werden. Dazu wird aus allen Tiefenmaps der im Datensatz kleinste d_{min} und größte d_{max} Pixelwert gesucht. Die Tiefenmaps können dann mit der folgenden Gleichung 19 korrigiert werden.

$$t_n(x, y) = \frac{t_n(x, y) - d_{min}}{d_{max} - d_{min}} \quad (19)$$

Da es sinnvoll ist, Strukturen des Hintergrundes auszublenden, kann ein Threshold gesetzt werden, ab dem Tiefenwerte, entweder auf den maximalen oder minimalen Wert gesetzt werden. Später lassen sich diese dann leicht herausfiltern, da im Grunde eine Art Maske entsteht. Beim Erstellen eines Gesichtsdatensatzes macht es Sinn den kleinsten Wert zu nehmen, da die Entfernung zwischen Kamera und Gesicht bei der Aufnahme verhältnismäßig klein ist. Somit wird d_{min} gleich 0 gesetzt.

Das Ergebnis der Anwendung von Gleichung 19 mit $d_{min} = 0$ auf die Tiefenmap aus Abbildung 24 ist in Abbildung 25 zu sehen.



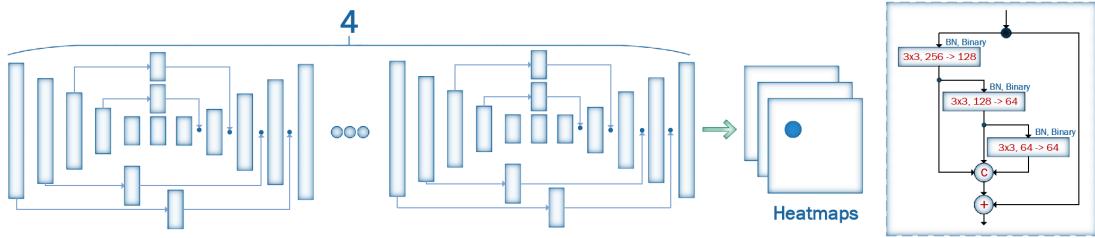


Abbildung 26: Face Alignment Network (FAN) von Bulat et al. [3]

4.1.3. Bestimmung von Facial Landmarks

Da das GAN später keine zufälligen Bilder erzeugen soll, muss der Datensatz gelabelt werden. Als Label dienen hier *Facial Landmarks*, welche vorab für jedes Bild im Datensatz bestimmt werden müssen.

Eine state-of-the-art Methode zur Bestimmung von Facial Landmarks ist das *Face Alignment Network* (kurz: FAN) von Bulat et al. [3]. Das FAN ist ein CNN, welches in der Lage ist 2D sowie 3D-Landmarks auf Basis eines RGB-Bildes zu bestimmen.

Das Netzwerk besteht aus vier hintereinander geschalteten *Hour-Glass*-Architekturen (vgl. Abbildung 26) nach Newell et al. [20], welche von Bulat et al. [3] modifiziert wurden (für weitere Informationen siehe Bulat et al. [3] [2]). Das Netzwerk wurde mit dem 300W-LP Datensatz trainiert, welcher 61.225 Gesichter mit 2D und 3D Facial Landmarks enthält.

Um die vom FAN erzeugten Landmarks mit den Label-Landmarks im Trainingsprozess zu vergleichen, verwenden Bulat et al. [3] den in Gleichung 20 beschriebenen *Nomralized Mean Error* (kurz: NME). N entspricht der Anzahl der Landmarks, x_k der 2D-Position des k-ten Label-Landmarks und y_k der, durch das FAN berechneten, 2D-Position des k-ten Landmarks. Normalisiert wird durch die Wurzel der Fläche der Boundingbox der 2D-Landmarks.

$$NME = \frac{1}{N} \sum_{k=1}^N \frac{\|x_k - y_k\|_2}{\sqrt{w_{bbox} * h_{bbox}}} \quad (20)$$

Das Berechnen von 2D-Landmarks dauert mit dem FAN und einer NVIDIA 2080 TI für 892 Bilder ca. 15 Minuten. Wird die Berechnung auf der CPU (Intel Xeon CPU e5 1620 v3 3,5GHz) durchgeführt, dauert dieser Prozess ca. 45 Minuten.



Abbildung 27: Crop und Resize

Für die Verwendung von RGBD-Bildern genügt die Bestimmung von zweidimensionalen Landmarks. Aus den Landmarks wird dann eine Heatmap erzeugt, welche die Landmarks durch weiße Punkte auf einem sonst schwarzen Bild zeigt. Um dem GAN möglichst viele Informationen zu geben, können die einzelnen Landmarks noch (logisch) miteinander verbunden werden, so dass eine Heatmap wie in Abbildung 27 (rechts) entsteht.

4.1.4. Skalierung und Zuschnitt

Erste Trainingsanläufe zeigten, dass das GAN auch das Verhältnis der Landmarks in Bezug auf die Größe des Bildes lernt. Um später ungewünschte Ergebnisse zu vermeiden, ist es sinnvoll für jedes Bild auf Basis der Landmarks eine Boundingbox zu erstellen und diese zu einem Quadrat (vorausgesetzt, es wird mit quadratischen Bildern gearbeitet) zu erweitern. Anschließend können die Bilder auf dieses Quadrat zugeschnitten und die gewünschte Pixelgröße skaliert werden (vgl. Abbildung 27).

Der komplette Datensatz enthält 5500 RGBD-Bilder mit entsprechender Heatmap.

4.2. GAN-Architektur

Für das grundlegende GAN eignet sich das in Abbildung 15 und Abbildung 16 zu sehende Pix2Pix-GAN (s. Unterunterabschnitt 2.4.2) nach Isola et al. [8].

Der Generator erhält als Eingabe eine 256*256 Pixel große Heatmap der *Facial Landmarks*. Die Ausgabe wird, im Vergleich zu dem in Abbildung 15 zu sehenden Generator, um eine vierte Featuremap erweitert, um RGBD-Bilder erzeugen zu können (vgl. Unterabschnitt 3.4).

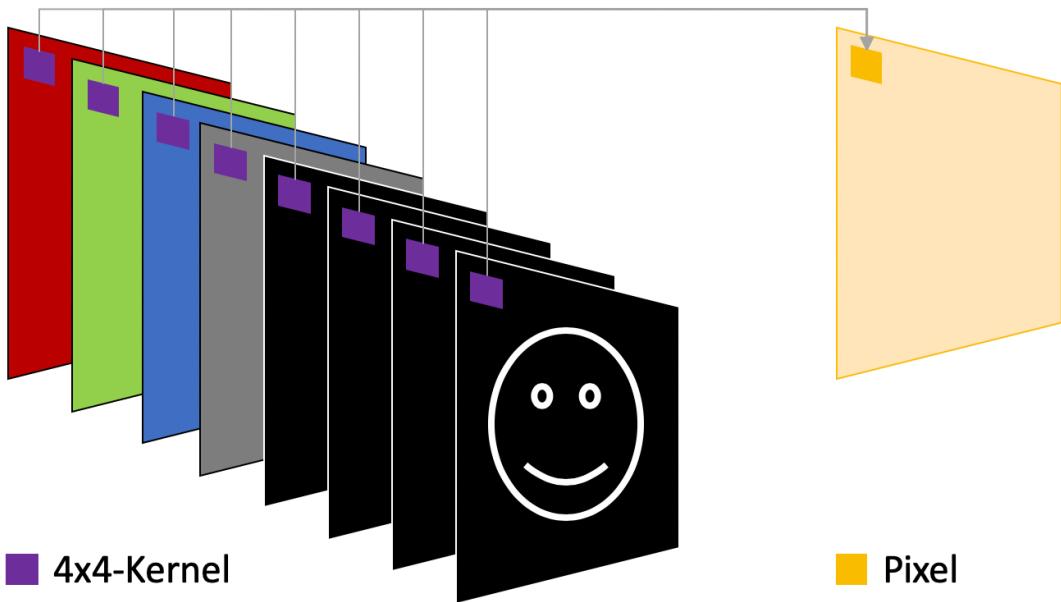


Abbildung 28: Beispiel Faltung für Diskriminator-Input

Der Diskriminator erhält als Eingabe 8 Featuremaps. Die ersten vier entsprechen den vier Kanälen des RGBD-Bildes, die restlichen vier enthalten je die Landmark-Heatmap. Auch wenn die Heatmap durch ein Graustufenbild repräsentiert wird und prinzipiell nur eine Featuremap benötigt, wird sie vierfach in den Diskriminator gegeben, um den Trainingsprozess mit einem ausgeglichenen Verhältnis zwischen RGBD-Bild und Landmarks zu beginnen (vgl. Abbildung 28). Alternativ könnten die initialen Gewichtungen so angepasst werden, dass die Heatmap einen größeren Einfluss erhält (vgl. Unterabschnitt 4.3).

4.3. Trainingsprozess

Vor dem Trainingsprozess werden alle lernbaren Gewichtungen in der GAN-Architektur mit einem zufälligen Wert initialisiert. Die Zufallswerte folgen dabei einer Gaußverteilung mit Erwartungswert 0 und Standardabweichung 0, 2.

Die Batchsize der Eingabedaten in das GAN beträgt 1, was bedeutet, dass die Gewichtungen nach jedem Trainingsbeispiel angepasst werden. Von Isola et al. [8] wird eine Batchsize zwischen 1 und 10 vorgeschlagen.

Generator und Diskriminator werden je mit einer Lernrate von 0,0002 trainiert, wobei beim Diskriminator gilt: $Loss_D = (Loss_{Dreal} + Loss_{Dfake}) * 0.5$, was die Lernrate für den

Diskriminator reduziert und dieser dadurch im Verhältnis zum Generator langsamer lernt. Dies ist notwendig, da der Diskriminator zu Beginn der Trainingsphase seine Aufgabe leicht bewältigen kann. Lernt er dann noch schnell dazu, hat der Generator keine Chance das Erstellen der gewünschten Bilder zu erlernen.

Die Gewichtungskonstante λ regelt in Gleichung 16 das Verhältnis der beiden Teil-Loss-Funktionen (vgl. Gleichung 14 und Gleichung 15) und wird, wie von Isola et al. [8] vorgeschlagen, mit dem Wert 100 initialisiert.

4.4. Evaluierung

Basierend auf den Anforderungen an das Erzeugen von Avataren gibt es drei zu bewertende Kriterien:

1. Korrekte Pose

Um die Korrektheit der Pose zu bewerten, kann der von Bulat et al. [3] bei der Bewertung des *Face Alignment Networks* verwendete *Normalized Mean Error* aus Gleichung 20 in Unterunterabschnitt 4.1.3 verwendet werden. Im Grunde ist die Problemstellung identisch (vgl. Unterunterabschnitt 4.1.3). Um den NME anwenden zu können, müssen aus dem Bild, welches der Generator erzeugt hat, erneut die Landmarks mit dem FAN bestimmt werden⁵. Diese können dann durch den NME mit den Landmarks verglichen werden, welche in den Generator gegeben wurden.

Anzumerken ist bei der Bewertung, dass Fehler, die bei der Bestimmung der Landmarks auftreten, nicht berücksichtigt werden. Auch wenn das FAN von Bulat et al. [3] durchaus gute Ergebnisse liefert, funktioniert es nicht perfekt. Eine Schwachstelle ist beispielsweise das Öffnen und Schließen der Augen.

In Anhang A ist der NME für 10 zufällige Frames berechnet worden. Mit einer durchschnittlichen Abweichung von 3,459 % erreicht das GAN in etwa die Fehlerquote, die das FAN bei der Evaluierung aufweist (vgl. Bulat et al. [3]).

Beim Betrachten mehrerer Ausgabebilder (vgl. Abbildung 29) fällt auf, dass die Augenstellung oft stark von der realen abweicht und sehr zufällig erscheint. Grund dafür ist, dass das GAN keine Informationen über die Augenstellung erhält, da die Pupillen in der Heatmap nicht verzeichnet sind. Das GAN muss die Stellung der Augen somit erraten.

⁵Auch wenn das FAN in der Lage ist dreidimensionale Landmarks zu bestimmen, können hier die zweidimensionalen Landmarks verwendet werden, da die Landmarks aus dem RGB-Bild bestimmt werden.

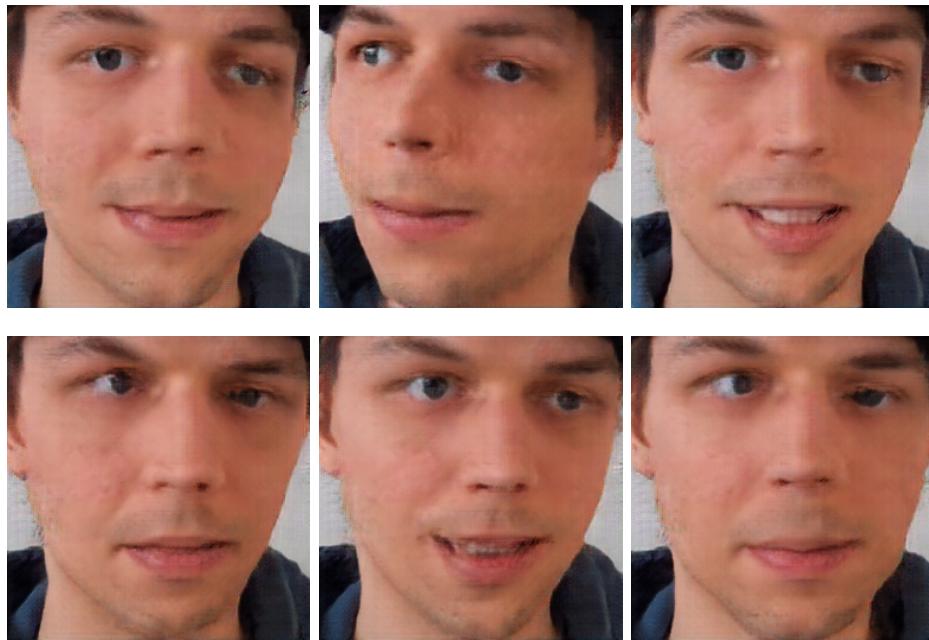


Abbildung 29: Beispiel Ausgabebilder (ohne Tiefe)

2. Korrekte Identität

Aufgrund dessen, dass dem GAN nur Bilder von einer Person als Trainingsdaten vorliegen, kann dieses keine andere Identität lernen. Unterschiede können lediglich in der Qualität der Bilder auftreten. Identität und Qualität der Bilder werden bereits während des Trainingsprozesses vom Diskriminator bewertet. Unter der Voraussetzung, dass der Trainingsprozess stabil verläuft, kann demnach ab dem Punkt, an dem von der Konvergierung des Trainingsprozesses auszugehen ist, der Losswert des Diskriminators zur Bewertung herangezogen werden. Zu Beginn des Trainingsprozesses unterscheidet der Diskriminator zwischen Rauschbildern und Bildern mit einem Gesicht. Der Prozess verschiebt sich jedoch dahin gehend, dass zwischen einem fotografierten und synthetisch erzeugtem Gesicht unterschieden werden muss. Ausschlaggebend ist dann die Qualität der generierten Bilder.

3. Erzeugungsdauer

VR-Anwendung laufen oft mit 90 FPS. Unter Vernachlässigung sämtlicher Weiterverarbeitungen sollte das Erzeugen eines Bildes durch das GAN also idealerweise nicht länger als $11ms$ dauern ($1s/90fps = 0.011s$). In Tabelle 3 ist die Dauer von 10 Generierungsprozessen aufgelistet. Im Schnitt liegt die Generierung mit ca. $17,5ms$ rund $6,4ms$ über den angestrebten $11ms$.

Generierungsprozess	Dauer / Sekunden
Bild 1	0,019988775
Bild 2	0,016990185
Bild 3	0,015991449
Bild 4	0,014994383
Bild 5	0,014994144
Bild 6	0,017989159
Bild 7	0,016986132
Bild 8	0,018988848
Bild 9	0,020988226
Bild 10	0,016992569
Durchschnitt	0,017490387
Soll für 90 Hz	0,011111111
Verhältnis Durchschnitt/Soll in %	157,4134827

Tabelle 3: Erzeugungsdauer

4.5. Interpretation des RGBD-Bildes zur Erzeugung einer Punktwolke

Um die Ausgabe des GANs weiterzuverarbeiten, ist es in den meisten Fällen notwendig aus dem RGBD-Bild ein Mesh zu erzeugen. Zur Erzeugung des Meshes muss das RGBD-Bild zunächst als Punktwolke interpretiert werden. In Tabelle 4 sind beispielhaft vier Ergebnisse als Punktwolke aus drei Blickrichtungen sichtbar.

Input Heatmap	Output als Point Cloud
	
	
	
	

Tabelle 4: Ergebnisse des RGBD-Face-Avatar-GANs als Punktwolken

5. Fazit und Ausblick

Wie in Tabelle 4 zu sehen, lassen sich mit dem Pix2Pix-GAN von Isola et al. [8] unter bestimmten Rahmenbedingungen (vgl. Abschnitt 4) 3D-Face-Avatare auf Basis von RGBD-Bildern erzeugen.

Bei der Verwendung von Facial Landmarks zur Steuerung sollten diese durch Eye Tracking um zwei Landmarks für die Pupillen ergänzt werden, um eine kontrollierte Augenstellung zu ermöglichen.

Mit einer Berechnungszeit von ca. 17,5ms ist der Generierungsprozess nicht weit davon entfernt für 90-Hz-VR-Anwendungen verwendbar zu sein. Der Trainingsprozess könnte durch die Verwendung mehrerer Grafikkarten (und der Erhöhung der Batchsize auf min. zwei) ggf. noch beschleunigt werden.

In Bezug auf entferntes Zusammenarbeiten in VR könnte die Anwendung so verwendet werden, dass zwischen zwei Endgeräten lediglich die Landmarks/Joints ausgetauscht werden. In Bezug auf die Netzwerkauslastung ist dies extrem performant. Auf den Endgeräten läuft dann je ein GAN mit den erlernten Gewichtungen des anderen. Die Gewichtungen müssen vorab einmalig untereinander ausgetauscht werden. Die Ausgabe des GANs kann dann live in die virtuelle Umgebung integriert werden.

Der größte Nachteil des in Abschnitt 4 erstellten RGBD-Avatar-GANs ist die Notwendigkeit eines personenspezifischen Datensatzes, der zudem auch alle später sichtbaren Posen (mit Fokus auf die Extrema) ungefähr gleich stark enthalten sollte. Um den Datensatz anzufertigen, wäre es denkbar eine Anwendung zu erstellen, die eine Person videogeführt dazu bringt, alle Posen vor einer RGBD-Kamera einzunehmen. Wird anstelle von mehreren Fotos ein Video aufgenommen, kann aus den einzelnen Frames der Datensatz erstellt werden. Das Einnehmen der Posen sollte möglichst spielerisch vermittelt werden, um die Person von der eigentlichen Aufgabe abzulenken. Die Absicht dahinter ist möglichst natürliche Posen aufnehmen zu können. Oft lachen Personen beispielsweise auf angekündigten Fotos anders als auf unbemerkt aufgenommenen Fotos.

Es gibt durchaus bereits GAN-Architekturen, welche auf Basis von nur wenigen oder gar einem Bild neue Posen, auf die vorgegebene Identität anwenden können (vgl. Unterabschnitt 3.3). Alle Anwendungen haben jedoch gemeinsam, dass sie vorab mit einem Datensatz aus mehreren tausenden Personen trainiert wurden. Da das Erstellen von Datensätzen, die über zwei Dimensionen hinausgehen, extrem aufwendig ist, existiert derzeit kein geeigneter Datensatz. Durch das Sammeln und Vereinen der, nach der oben beschriebenen Methode, erstellten Datensätze, ließe sich ein solcher Datensatz nach und nach aufbauen, was eine spätere Umsetzung ohne personenspezifischen Datensatz ermöglichen würde.

Literatur

- [1] Wayne Chang Ben Cutler, Spencer Fowers. <https://www.microsoft.com/en-us/research/project/hoportoation-3/>, jul 2015.
- [2] Adrian Bulat and Georgios Tzimiropoulos. Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [3] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, 2017.
- [4] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.
- [5] CMU-Perceptual-Computing-Lab. <https://github.com/cmu-perceptual-computing-lab/openpose> (zugriff: 11.02.2020 15:17), 2017.
- [6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.
- [8] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.
- [9] Greg Walters John Hany. *Hands-On Generative Adversarial Networks with PyTorch 1.x*. Packt Publishing Ltd., December 2019.
- [10] Ujjwal Karn. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>, aug 2016.
- [11] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017.
- [12] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2018.

- [13] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- [14] Bingchen Liu. <https://github.com/odegeasslbc/Progressive-GAN-pytorch>, jun 2019.
- [15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [16] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation, 2017.
- [17] Microsoft. <https://azure.microsoft.com/de-de/services/kinect-dk/> Stand: 25.03.20 11:46.
- [18] Microsoft. https://www.cntk.ai/pythondocs/CNTK_206A_Basic_GAN.html Stand: 06.01.2020 13:55, 2017.
- [19] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.
- [20] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.
- [21] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.
- [22] Albert Pumarola, Antonio Agudo, Alberto Sanfeliu, and Francesc Moreno-Noguer. Unsupervised person image synthesis in arbitrary poses, 2018.
- [23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.
- [24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [25] Sumit Saha. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (Stand: 06.01.2020 14:49), dec 2018.
- [26] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing, 2019.
- [27] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

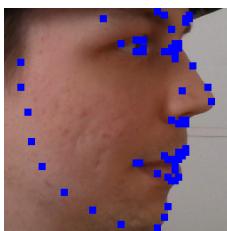
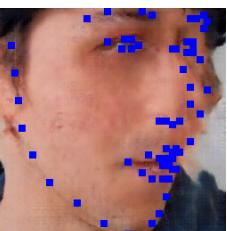
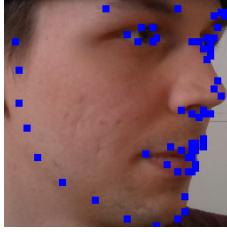
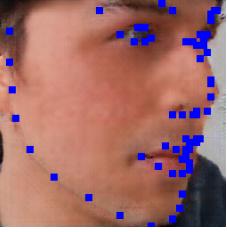
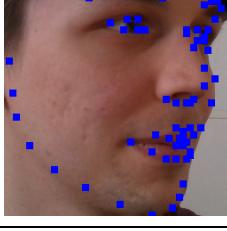
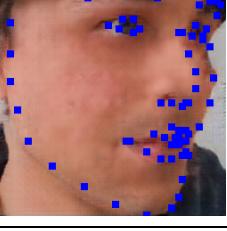
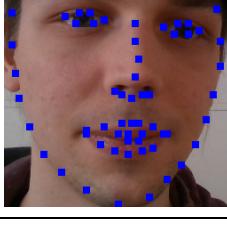
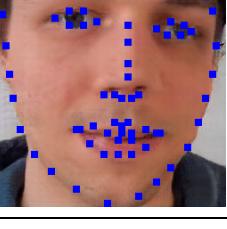
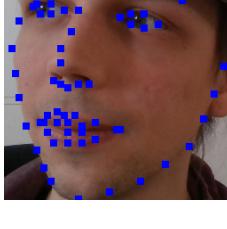
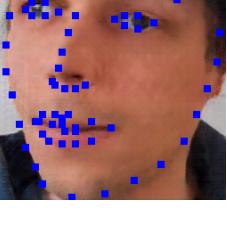
- [28] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2014.
- [29] Brent Allen Tetyana Sych, Phil Meadows. <https://docs.microsoft.com/de-de/azure/kinect-dk/depth-camera>, 2019.
- [30] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.
- [31] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models, 2019.
- [32] Zhen Zhu, Tengteng Huang, Baoguang Shi, Miao Yu, Bofei Wang, and Xiang Bai. Progressive pose attention transfer for person image generation, 2019.

Abbildungsverzeichnis

1.	virtuelle 3D-Teleportation in Echtzeit von Microsoft Research [1]	5
2.	Generative Adversarial Network [18]	7
3.	MinMax-Game Plot. Die x-Achse entspricht hier $D(x)$ und die y-Achse dem Loss-Wert.	9
4.	Max-Pooling [10]	10
5.	Convolutional Neural Network [25]	11
6.	DCGAN-Generator von Radford et al. [23]	11
7.	links reale Gesichter (Beispielbilder aus Trainingsdatensatz), rechts durch DCGAN erzeugte Bilder	13
8.	Generator des 3D-GAN von Wu et al. [30]	14
9.	Je drei Bilder für die Kategorien <i>lachende Frau</i> , <i>nicht lachende Frau</i> und <i>nicht lachender Mann</i>	16
10.	Drei Durchschnittsbilder der Kategorien <i>lachende Frau</i> , <i>nicht lachende Frau</i> sowie <i>nicht lachender Mann</i> und das Ergebnis der Verrechnung nach Gleichung 6	16
11.	Vektorprojektion von Shen et al. [26] (Beachte Gleichung 12)	19
12.	Generierte MNIST-Ziffern Quelle: Mirza et al. [19]	20
13.	Vergleich von Isola et al. [8]: Auto-Encoder (links) vs. U-Net (rechts)	22
14.	Auswirkung der Skip-Connections und Loss-Funktion auf die Qualität der Ausgabe im Pix2Pix-GAN von Isola et al. [8]	23
15.	Generator im Pix2Pix-GAN für 256*256px große RGB-Bilder [9]	25
16.	Diskriminator im Pix2Pix-GAN für 256*256px große RGB-Bilder [9]	25
17.	Beispiele für cartoon-artige Avatare	27
18.	(A) Beispielbilder aus Datensatz, (B) durch Progressive-GAN erzeugte Bilder	28
19.	Skeletterkennung durch OpenPose [4] inkl. Gesichts- u. Handerkennung [27]	30
20.	Animation durch Skelett	30
21.	Oben: durch Progressive-GAN [14] erzeugtes RGBD-Bild (links: RGB, rechts: Tiefenbild), Unten: 3D-Darstellung des Bildes aus drei Perspektiven	32
22.	Azure Kinect DK von Microsoft [17]	34
23.	Beispiel aus RGBD Datensatz	34
24.	Histogramm (links) der aufgenommenen Tiefenmap (rechts)	36
25.	Histogramm (links) der über den Datensatz normalisierten Tiefenmap (rechts)	36
26.	Face Alignment Network (FAN) von Bulat et al. [3]	37
27.	Crop und Resize	38
28.	Beispiel Faltung für Diskriminator-Input	39
29.	Beispiel Ausgabebilder (ohne Tiefe)	41

A. Normalized Mean Error

In Tabelle 5 wird die Pose auf den realen Inputbilder mit der Pose auf den synthetisch erzeugten Outputbildern des GANs aus Abschnitt 4 anhand des NME aus Gleichung 20 verglichen.

Input	Output	NME in %	Bemerkung
		12,703	Extremfall: Der Datensatz enthält keine oder nicht viele Bilder in dieser Pose.
		3,910	
		3,895	
		4,209	
		2,986	

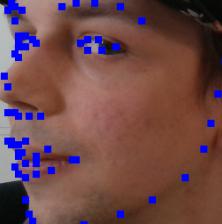
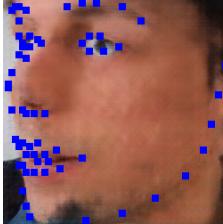
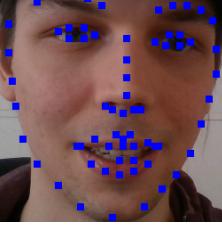
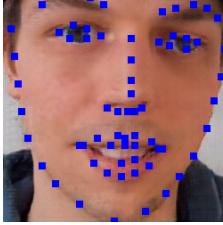
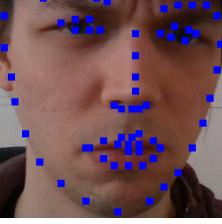
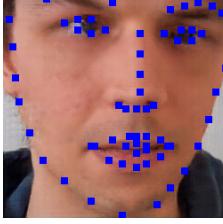
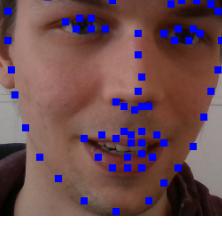
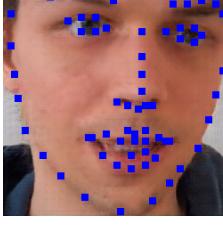
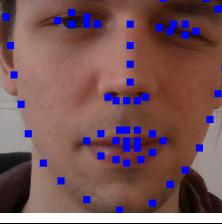
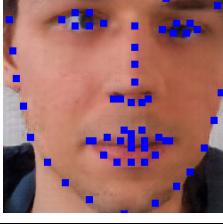
		4,713	
		3,229	
		2,844	
		2,685	
		2,659	
Durchschnitt:		3,459	Beispiel 1 wurde bei der Berechnung des Durchschnitts ausgelassen.

Tabelle 5: Normalized Mean Error zur Bewertung der Pose des RGBD-Avatar-GANs