Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Dahlem Center for Machine Learning and Robotics

LabelGAN: Generierung realistischer, gelabelter Daten mithilfe eines Generativen Adversativen Netzwerkes

Dorian Wachsmann
Matrikel-Nummer 4756468
Berlin, 23. Juni 2017

Betreuer	Prof. Dr. Tim Landgraf
Erstprüfer	Prof. Dr. Tim Landgraf
Zweitprüfer	Prof. Dr. Raúl Rojas

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Abstract

Generative Adversative Netze(GAN) gelten als eine der spannendsten und vielversprechendsten neuen Ansätze beim Deep Learning. GAN bieten eine Möglichkeit, auf Basis ungelabelter Daten neue Daten zu generieren. Diese Arbeit liefert eine Erweiterung dieses Ansatzes dahingehend, gezielt Bilder auf Basis von festgelegten Labels zu generieren. Die generelle Umsetzbarkeit wird auf dem MNIST-Datensatz präsentiert, während die wahre Herausforderung in der Generierung realistischer, gelabelter Gesichter liegt. Damit verbundene Probleme werden diskutiert, die Unterschiedlichkeit zum MNIST Ansatz erläutert, sowie verschiedene Ansätze zur Lösung aufgezeigt und evaluiert.

2 Einleitung

Durch eine Renaissance der Neuronalen Netze, insbesondere von Tiefen Neuronalen Netzen(Deep Neural Nets), gewinnen die zum Training notwendigen Datensätze eine immer größere Bedeutung. Für ein adäquates Training tiefer Netze sind große Datenmengen gelabelter Eingaben erforderlich, da sonst der Lerneffekt zu gering und damit keine gute Abstraktion gewährleistet ist. Das manuelle Labeln ist sowohl zeit- als auch kostenintensiv, was bei Datensätzen bei Größenordnungen von über 100.000 nicht verwunderlich ist. Anstatt reale Daten zu nehmen und diese in aufwendiger Prozedur von Hand zu labeln, liegt es nahe, stattdessen synthetische Daten zu verwenden, die von einem Computer generiert wurden und daher schon über alle notwendigen Label verfügen. Jedoch hat sich herausgestellt, dass das Training auf synthetischen Daten häufig zu keinen guten Resultaten führt. Der Grund dafür liegt in der Unterschiedlichkeit von realen Daten aus der Problemdomäne und den synthetischen, von Rechnern generierten Daten. Netzwerke lernen dabei häufige unerwünschte Details, die nur in den synthetischen Daten vorliegen und versagen danach in der Anwendung.

In der Computervision entwickelte sich der Trend in den letzten Jahren weg von klassischen algorithmischen Ansätzen, hin zu tiefen Architekturen Neuronaler Netze. In der Bilderkennung und Klassifizierung liefern sich jährlich eine Reihe anerkannter Forschergruppen einen Wettbewerb um die besten Ergebnisse.[1] Doch auch hier stellt sich die Frage nach passenden Datensätzen. Hochwertig gerenderte synthetische Bilder sind teuer und erfüllen trotzdem möglicherweise die Anforderungen nicht komplett. Meine Bachelorarbeit setzt an dieser Problematik an und liefert ein Framework mit dem Ziel, hochwertige, realistische und gelabelte Bilder von Gesichtern zu generieren.

1 `http://image-net.org/challenges/LSVRC/2016/index`

2.1 Motivation

Inspiration und Motivation zu dieser Arbeit liefert die Bachelorarbeit von Leon Sixt aus dem Jahr 2016, die später als Paper veröffentlicht wurde.[1] In dieser Arbeit wird ein Framework vorgestellt, welches in der Lage ist, große Mengen gelabelter Bilder zu erzeugen. Die Idee und Architektur wird später im Abschnitt RenderGAN, siehe Abschnitt 3.2, vorgestellt. RenderGAN wurde für eine spezielle Problemstellung im Kontext der Arbeit des Beesbook-Projekt entworfen. Zum Training der bei Beesbook verwendeten Neuronalen Netze bedarf es gelabelter Bilder, die allerdings von Hand nur in sehr aufwendiger Arbeit gelabelt werden können. Dank RenderGAN kann das Training auf synthetisch generierten Bildern ablaufen und liefert sehr gute Resultate.

Es existieren bereits Arbeiten zu ähnlichen Aufgabenstellungen, wie beispielsweise das Paper "Learning from Simulated and Unsupervised Images through Adversativ Training" von Apple Inc., in dem synthetisch generierte Bilder durch Adversatives Training in der Ausgabe realistischer werden, während die Grundstruktur des Eingabebildes nicht verändert wird.[2] Im Abschnitt 3.6 wird auf dieses Paper genauer eingegangen, da es eine Reihe von Ähnlichkeiten zu der Problemstellung dieser Arbeit aufweist und daher in einigen Punkten gut als Referenz genutzt werden kann. Meines Wissens existiert zum jetzigen Zeitpunkt noch kein Framework, welches in der Lage ist, ein Vorgehen, wie im Apple Inc. Paper beschrieben, für Gesichter anzuwenden.

In den letzten Jahren gab es eine Reihe von Veröffentlichungen, die die Generierung von realistischen Bildern vorangetrieben haben. Abbildung 2.1 zeigt beispielhaft die Qualität von BeGAN [3], das im Mai 2017 präsentiert wurde. BeGAN bietet jedoch keinen Erhalt von Labels bei dem generativen Prozess, sondern generiert die Bilder willkürlich, ohne die Möglichkeit, Parameter, also Labels, festlegen zu können. Sollte das in dieser Arbeit entwickelte Framework dies leisten können und gleichzeitig Bilder von ausreichender Qualität erzeugen, wären zahlreiche Anwendungsfelder denkbar. Es wäre zum Beispiel möglich, Neuronale Netze gezielt auf Basis der synthetischen Bilder zu trainieren, ohne wesentliche Einbußen in Kauf zu nehmen. Dafür soll ein Weg gefunden werden, den generativen Prozess abhängig von den Labels zu machen. Unter Labels versteht man die Information über den Inhalt des Bildes. Im Fall der Gesichtsgenerierung wären Labels denkbar wie, lange Haare, männlich, Brillenträger etc.. Eine ausführlichere Beschreibung der Problemstellung und der Wahl geeigneter Labels erfolgt in dem nächsten

Abbildung 2.1: State of the Art Generierung von Gesichtern. (c)Boundary Equilibrium Generative Adversativ Networks - Google

Kapitel.

2.2 Problemstellung

Es gibt unterschiedliche Arten, Neuronale Netze zu trainieren. Überwachtes Training ist dabei eine einfache, viel verwendete und intuitiv verständliche Art des Trainings. Der Trainingsdatensatz besteht aus Eingaben x, sowie den erwarteten Ausgaben y. Dem Neuronalen Netz wird x gegeben, zu dem es eine Ausgabe $\tilde{y}$ generiert. Nun kann der Abstand über eine geeignete Loss-Funktion bestimmt werden, im einfachsten Fall der L1 Loss: $|y - \tilde{y}|$. Indem das Neuronale Netz lernt, diesen Fehler zu minimieren, lernt es, passende $\tilde{y}$ zu Eingaben x zu generieren.

Diese stark vereinfachte Beschreibung eines Lernvorganges hat den Zweck, auf einen speziellen Aspekt hinzuweisen: Der Datensatz, auf dem trainiert werden soll, benötigt neben den Datensätzen x die erwartete Ausgabe y. Man kann sich leicht vorstellen, dass bei Datensätzen von mehreren hunderttausend Daten das manuelle Eintragen der erwarteten Ausgaben eine zeit- und kostenintensive Arbeit ist. Dies zu automatisieren klingt nach einem paradoxen Problem. Einerseits möchte man ein Neuronales Netz darauf trainieren, beispielsweise Äpfel von Birnen zu unterscheiden, andererseits benötigt man zum Training

dieses Netzes zuerst einen Algorithmus, der genau diese Unterscheidung treffen kann, um den Trainingsdatensatz zu erschaffen. Die Lösung lautet unbewachtes Lernen mit Adversativem Loss zur Generierung von gelabelten Datensätzen.

Für diese Arbeit soll ein Generatives Adversatives Netzwerk[4] gebaut werden. Es soll auf einem ungelabelten Bilddatensatz mittels unbewachtem Lernen trainiert werden, um realistische, gelabelte Gesichter zu erzeugen. Das wirft eine Reihe Fragen und Probleme auf:

P.0	Welche Labels können für ein Gesicht gewählt werden
P.1	Wie muss die Eingabe für den Generator aussehen
P.2	Wie ist der Generator/Diskriminator aufgebaut
P.3	Wie wird abgesichert, dass die Labels während des generativen Vorganges erhalten bleiben
P.4	Wie wird die Qualität der erzeugten Ausgaben getestet

Tabelle 2.1: Problemstellungen

2.3 Lösungsansatz

Für jeden genannten Punkt in Tabelle 2.1 wird im Folgenden eine Lösungsidee beschrieben.

Zweierlei hat der Generator zu leisten. Zum einen muss er das Bild realistischer machen, damit der Diskriminator es nicht mehr von den realen Daten unterscheiden kann. Andererseits muss er die Labels der Eingabe erhalten. Wenn beides gewährleistet werden kann, dann, so die Hoffnung, entstehen große Mengen, realistischer, gelabelte Bilder.

Das führt zu der Frage, welche Labels gewählt werden. Das ist hauptsächlich von der Qualität der vor-gerenderten Gesichter abhängig:
Abbildung 2.2 zeigt exemplarisch ein solches Bild. Den synthetischen Gesichtern fehlen markante Eigenschaften wie Haare oder Accessoires. Daher wurden als Label die Orientierung des Gesichtes im Raum, die Gesichtstextur und die Form gewählt..

Abbildung 2.2: Beispielbild der verfügbaren vor-gerenderten Gesichter

Der Generator bekommt als Input das synthetische Gesicht und einen Zufallsvektor z. Dieser wird auf Bildgröße skaliert und als Channel an das Bild angehängt. Das soll die nötige Varianz des Inputs erhöhen, damit die Ausgabe ebenfalls unterschiedlicher wird.

Die Architektur ist vollständig Convolutional, ohne Fully-Connected Layer. Das orientiert sich an der DCGAN Architektur[6]. Es wird ein WassersteinGAN[7] verwendet, um bekannte Probleme der GAN-Konvergenz auszuschließen. Nähere Details zur Implementierung werden in Kapitel 4 beschrieben.

Ein GAN generiert Bilder auf Basis eines Zufallsvektors. Die Eingabe wird um das synthetische Gesicht $face$ erweitert. Die Blickrichtung, die Gesichtsform und Farbe soll in $G(z, face) = \tilde{x}$ erhalten bleiben. Um das zu gewährleisten, gibt es verschiedene Ansätze. Intuitiv könnte ein L1 Loss $\tilde{x} - face$ sicherstellen, dass der Generator das Ursprungsbild nicht zu stark verändert. Ein anderer Ansatz ist, ein Predictor-Netz P, welches als Input $\tilde{x}$ erhält und als Output $P(\tilde{x}) = \tilde{face}$ liefert. Das Predictor Netzwerk wird darauf trainiert, $face$ möglichst getreu nachzubilden und "bestraft" den Generator, wenn es ihm nicht möglich ist. Dieser Ansatz wird in dieser Arbeit angewandt. Tests mit einem einfachen L1-Loss ergaben keine zufriedenstellenden Resultate.

2.4 Vorgehensweise und Resultate

Für eine strukturierte Herangehensweise wurden Meilensteine definiert. Diese geben der Arbeit einen Rahmen und halfen, während des Entwicklungsprozesses einerseits das Ziel nicht aus den Augen zu verlieren und andererseits schrittweise vorgehen zu können. Folgende Meilensteine wurden definiert:

1. Implementierung Wasserstein-GAN auf MNIST Daten
2. Erweiterung WGAN zu LabelGAN und Tests auf MNIST
3. Implementierung LabelGAN mit synthetischen Gesichtern
    a) LabelGAN auf CelebA mit Mean-Bildern implementieren
    b) LabelGAN auf CelebA mit synthetischen Gesichtern implementieren
    c) Andere Datensätze für LabelGAN austesten/Datensätze kombinieren
4. Tests und Validierung der Implementierung

In der Arbeit wurde ein Improved Wasserstein GAN[5] um ein Predictor Netzwerk P erweitert. Der Name LabelGAN leitet sich aus dem Ziel ab, über den generativen Prozess hinweg vorher definierte Label zu erhalten. Ziel des Predictor Netzes ist es, die Parameter, die dem Generator als Eingabe übergeben wurden, aus dem vom Generator erzeugten Bild wieder zu extrahieren. Dafür wurden einige Anpassungen des WGAN-Frameworks vorgenommen. Als wichtigste Änderung ist die erweiterte Loss Funktion des Generators zu nennen, der zusätzlich bestraft wird, wenn der Predictor nicht in der Lage ist, die Parameter zu extrahieren.

Auf unterschiedlichen Datensätzen liefert LabelGAN unterschiedlich gute Performance. Während die Resultate auf dem MNIST Datensatz sehr vielversprechend sind, gelang es nicht, gleiche Qualität für Bilder von Gesichtern auf CelebA+IMDB zu reproduzieren. Die generelle Qualität der erzeugten Bilder ist gut, jedoch werden bei dem generativen Prozess die Labels weitgehend vernichtet. Es wurde kein Weg gefunden, die hohe Ausfallquote der erzeugten Bilder zu reduzieren. Daher liegt ein großer Aspekt der Arbeit in der Aufdeckung möglicher Ursachen, weshalb der Ansatz mit einem Predictor-Netzwerk nicht funktioniert. In Kapitel 5 werden Fehlerquellen und Probleme diskutiert und anhand von exemplarischen Beispielen die Qualität von LabelGAN präsentiert. Davor

werden die Grundlagen der Arbeit sowie die Architektur und Implementierung vorgestellt.

3 Related Work

3.1 Generative Adversative Netze

Generative Adversative Netze wurden zum ersten Mal von Ian Goodfellow et al. im Jahr 2014 vorgestellt[4]. Die Grundidee besteht in einem Min-Max Spiel zweier konkurrierender Funktionen, typischerweise Neuronale Netze. Die eine Funktion wird im Folgenden Generator G, die andere Diskriminator D genannt. G und D werden parallel trainiert, wobei das Ziel des Generators ist, möglichst realistische Bilder zu generieren, während das Ziel des Diskriminator ist, eine Wahrscheinlichkeit zu berechnen, dass eine Eingabe aus den Trainingsdaten und nicht von dem Generator kommt. Dabei erhält G als Eingabe einen Zufallsvektor z und generiert auf dieser Basis ein Bild $G(z)$. Der Diskriminator D erhält als Eingabe entweder $D(G(z))$ oder $D(x)$, wobei x aus einer realen Datenverteilung stammt, zum Beispiel einer Gesichtsdatenbank. Sobald der Diskriminator lernt, die Bilder besser voneinander zu unterscheiden, ist der Generator gezwungen, realistischere, also Bilder, die der Verteilung eher entsprechen, zu generieren. Goodfellow hat in dem Paper gezeigt, dass ein G existiert, sodass D nicht mehr zwischen realen und generierten Bildern zu unterscheiden weiß.

Formal lässt sich das Min-Max-Spiel folgendermaßen beschreiben:

$$\min_G \max_D \mathbb{E}_x[log(D(x))] + \mathbb{E}_z[log(1 - D(G(z)))] \tag{3.1}$$

Der erste Erwartungswert beschreibt die Performance von D auf den realen Daten x. Je höher dieser Wert, desto besser kann D diese erkennen. Der zweite Summand beschreibt hingegen, wie gut D die generierten Bilder erkennt. Der Generator möchte diesen zweiten Part minimieren, was heißt, dass der Diskriminator die generierten Bilder als reale identifiziert.

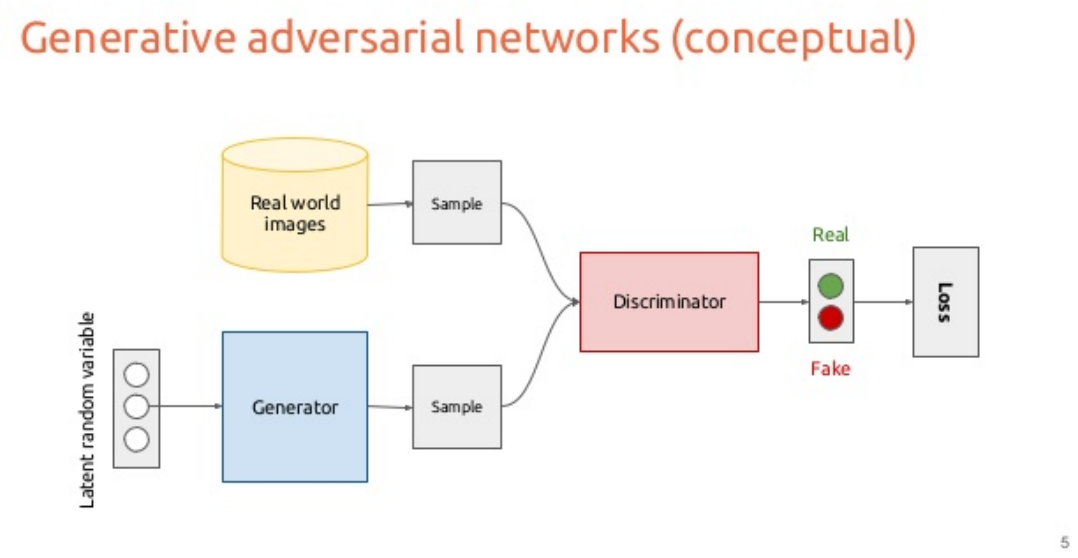

Abbildung 3.1: Quelle: https://wiki.tum.de/pages/viewpage.action?pageId=23562510

3.2 RenderGAN

RenderGAN ist eine Arbeit von Leon Sixt aus dem Jahr 2016 [1] und liefert die Idee für diese Arbeit. RenderGAN kombiniert ein GAN-Framework mit einem 3D-Model für Generierung großer Mengen gelabelter Daten. Das 3D-Model ist Teil des Generators und wird ergänzt durch eine Reihe von Deformierungsfunktionen Φ_i. Diese fügen dem synthetischen Model Rauschen, Licht und Hintergrund hinzu und nähern das Bild so den realen Daten an. Die Parameter für die Funktionen lernt der Generator während des Adversativen Prozesses.

3.3 DCGAN

¨Unsupervised Representation Learning with Deep Convolutional Generative Adversativ Networks¨ wurde 2016 von Alec Radford et al [6] veröffentlicht. In diesem Paper wird eine neue Klasse von Convolutional Neural Nets präsentiert, die Deep Convoultional

1 http://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Bachelor-theses/2016/Sixt/Bachelor-Sixt.pdf

Generative Adversarial Networks(DCGAN) genannt werden. Es werden Architekturparadigmen beschrieben, die in diese Arbeit zum Teil eingeflossen sind. Die Architektur wurde auf verschiedenen Datensätzen getestet und zeigt eine überzeugende Performance. Die relevantesten Ergebnisse, die für diese Arbeit genutzt wurden, sind in der Tabelle 3.1 zusammengefasst.

1.	Nutze keine Pooling Layer, sondern Strided-Convolutional-Layer
2.	Nutze Batchnormalisierung im Generator sowie im Diskriminator
3.	Nutze keine Fully-Connected-Layer
4.	Nutze ReLU Aktivierungsfunktionen im Generator
5.	Nutze LeakyReLU Aktivierungsfunktionen im Diskriminator

Tabelle 3.1: DCGAN-Paradigmen

Eine typische DCGAN Architektur zeigt Abbildung 3.2

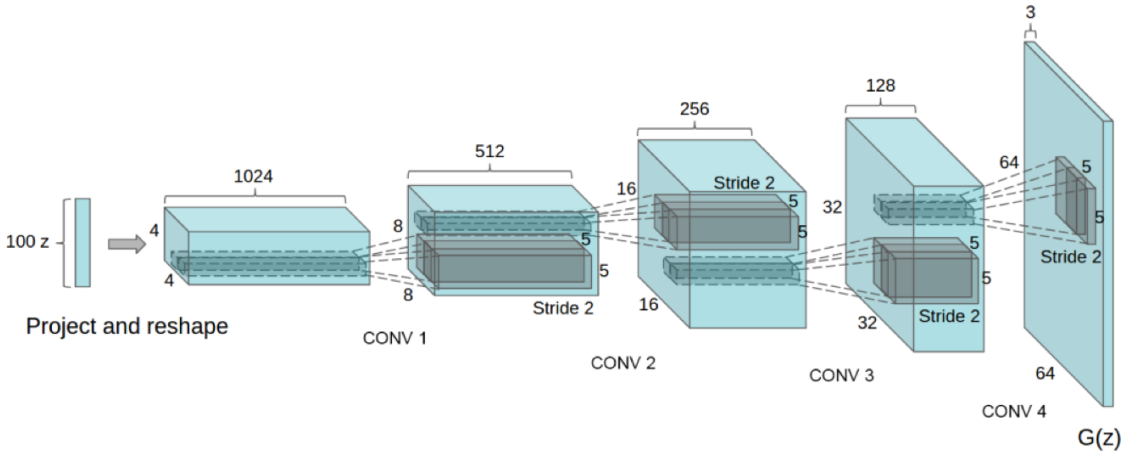

Abbildung 3.2: Typische Architektur eines DCGAN Generators. Quelle: DCGAN Paper

3.4 Wasserstein-GAN

Wasserstein GAN wurde Anfang 2017 von Martin Arjovsky et al[7] vorgestellt. Eines der größten Herausforderungen bei Generativen Adversativen Netzen ist das instabile

Training. Während des Trainingsprozesses kann es zu einer Reihe von unerwünschtem Verhalten kommen, was es nötig macht, den Lernprozess zu überwachen und manuell einzugreifen. Beispiele dafür sind, dass der Generator nur noch sehr ähnliche Bilder generiert, also die Diversität der Bilder nicht mehr gewährleistet oder aber der Diskriminator das Min-Max-Spiel "gewinnt" und damit dem Generator keinen Gradienten zum Lernen mehr liefert.

Ein weiteres Problem ist, dass ein Vanilla-GAN keine Information über die Konvergenz liefert. Es wäre praktisch, über die Größe des Loss etwas über die Konvergenz der Modelle zu erfahren.

Wasserstein-GAN liefert eine Lösung für beide Probleme.

Zum Verständnis der Neuerungen ist ein Blick auf die Theorie von Generativen Modellen nötig. Bei dem Training von solchen Modellen sollen Daten aus einer unbekannten Verteilung P_{real} durch eine Verteilung P_θ approximiert werden, wobei θ die Parameter der Verteilung sind. Dazu gibt es zwei verschiedene Ansätze. Zum einen könnte direkt die Dichtefunktion P_θ gelernt und diese über ein Maximum-Likelihood Verfahren optimiert werden. Der zweite Ansatz lernt stattdessen eine differenzierbare Funktion g_θ, die eine Verteilung Z in P_θ transformiert. Z kann dabei eine beliebige Verteilung sein, häufig normal- oder gleichverteilt. Während sich zeigen lässt, dass der erste Ansatz in Probleme läuft, hat sich der zweite Ansatz für generative Modelle als erfolgversprechend erwiesen.[2]

Wir stehen daher vor der Herausforderung, die Funktion g_θ in Hinblick auf die Parameter θ zu lernen. Dazu benötigen wir eine Messung der Distanz d zwischen den Verteilungen. Intuitiv ist verständlich, dass das Minimieren von $d(P_{real}, P_\theta)$ dazu führt, die beiden Verteilungen anzunähern. Das Ziel ist also, die Distanz d zu minimieren.

Es gibt verschiedene Metriken zur Bestimmung der Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen. Das Wasserstein-GAN zeichnet sich dadurch aus, dass sich der Loss über die sogenannte Wasserstein Distanz berechnet. Es gibt verschiedene andere Möglichkeiten, beispielsweise die Kullback-Leibler Divergenz(KL-Divergenz) und die Jenson-Shannon-Divergenz(JS). Anzumerken ist, dass die KL-Divergenz kein Distanzmaß nach Definition ist, da die Anforderung der Symmetrie nicht gegeben ist. Daher existiert ein Unterschied zwischen der KL-Divergenz und der reverse-KL-Divergenz.

2 `http://www.alexirpan.com/2017/02/22/wasserstein-gan.html`

Das Paper beschreibt Sequenzen von Verteilungen, in denen sowohl die KL als auch die JS Divergenz nicht konvergieren, die Wasserstein Distanz hingegen schon. Weiterhin wird bewiesen, dass jede Verteilung, die unter KL, JS, reverse-KL konvergiert, ebenfalls auch unter Wasserstein konvergiert. Daraus folgt, dass alle Probleme, die mit herkömmlichen GAN-Ansätzen gelöst werden konnten, ebenfalls mit Wasserstein lösbar sind und darüber hinaus Verteilungen existieren, die nur unter Wasserstein konvergieren. Dies ist Motivation genug, den Wasserstein-Loss genauer zu betrachten.

Sei $\prod(P_r, P_g)$ das Set aus allen multivariaten Verteilungen γ mit den Randverteilungen P_r und P_g. Es gilt:

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} \mathbb{E}_{x,y\sim\gamma}[||x - y||] \tag{3.2}$$

Wahrscheinlichkeitsverteilungen sind darüber definiert, wieviel "Masse" jedem Punkt zugeordnet ist. Die Wasserstein-Distanz beschreibt die minimale Anstrengung, derer es bedarf, die Masse einer Verteilung in die andere zu transportieren. Jedes γ lässt sich als eine Art Transportplan sehen. Für alle x, y transportiere $\gamma(x, y)$ Masse von x nach y. Das Infinum über alle γ gibt die Wasserstein-Distanz.l[3]

Für die Implementierung wird eine Approximation der Gleichung verwendet:

$$W(P_{real}, P_\theta) = \sup_{||D||_L \leq 1} \mathbb{E}_{x\sim P_{real}}[D(x)] - \mathbb{E}_{\tilde{x}\sim P_\theta}[D(\tilde{x})] \tag{3.3}$$

Erwähnenswert ist an dieser Stelle, dass das Supremum über alle 1-Lipschitz-stetigen Funktionen gewählt wird. Da $D(x)$ die Diskriminator Funktion in dem GAN ist, muss in der Architektur diese Einschränkung beachtet werden. In dem Paper wird dies durch sogenanntes "weight clamping" sichergestellt. Die Gewichte w der Diskriminator-Funktion werden nach jedem Lernschritt auf das Intervall $[-c, c]$ beschnitten. Dadurch bleibt die Lipschitz-Stetigkeit der Funktion erhalten.

Tests mit Wasserstein-GAN haben gezeigt, dass die Performance deutlich besser ist verglichen mit herkömmlichen GAN. [4] Ein weiterer großer Vorteil ist, dass der Diskriminator Loss besser zu interpretieren ist, im Hinblick auf die Konvergenz der Verteilungen. Der Diskriminator gibt keine Wahrscheinlichkeit zurück, sondern einen Wert, sodass $W(P_{real}, P_\theta)$

3 http://www.alexirpan.com/2017/02/22/wasserstein-gan.html
4 https://github.com/martinarjovsky/WassersteinGAN

gegen Null konvergiert, wenn der Diskriminator die Bilder nicht voneinander unterscheiden kann.

Im nächsten Abschnitt werden einige Verbesserungen im Hinblick auf das Beschneiden der Gewichte des Diskriminator zum Erhalt der Lipschitz-Stetigkeit gegeben, die in dieser Arbeit zu einer Verbesserung der Ergebnisse beigetragen haben.

3.5 Improved WGAN

Nur einige Monate nach der Veröffentlichung des WGAN Papers wurde das Paper ¨Improved Training of Wasserstein GANs¨ von Ishaan Gulrajani et al[5] veröffentlicht. In diesem Paper werden Hinweise auf eine stabilere Konvergenz gegeben. Die hauptsächliche Neuerung betrifft dabei das weight clipping, wie im Absatz davor beschrieben. Sie stellen ein anderes Verfahren zur Sicherstellung der Lipschitz-Stetigkeit vor, welches bei deren Tests besser abgeschnitten hat. Anstatt die Gewichte zu beschränken, bestrafen sie die Norm des Gradienten von dem Diskriminator(im folgenden Critic genannt. Diskriminator ist der Begriff, der verwendet wurde, solange eine tatsächliche Diskriminierung zwischen realen und synthetischen Bildern stattfand. Bei WGAN ist das nicht mehr der Fall).

Eine differenzierbare Funktion ist 1-Lipschitz stetig genau dann, wenn sie Gradienten mit Norm kleiner gleich Eins überall hat. Daher ist die Idee, die Gradienten Norm entsprechend des Inputs einzuschränken. Die Loss Funktion des Wasserstein-GAN wird folgendermaßen erweitert:

$$L = \mathbb{E}_{x \sim P_{real}}[D(x)] - \mathbb{E}_{\tilde{x} \sim P_\theta}[D(\tilde{x})] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{3.4}$$

Der erste Teil ist der Original Critic Loss mit Wasserstein-Abstand, wie im Abschnitt davor beschrieben. Der zweite Teil fungiert als Gradienten-Bestrafung. An speziellen Punkten aus der Verteilung $\hat{x} \sim P_{\hat{x}}$ wird der Gradient vom Critic $\nabla_{\hat{x}} D(\hat{x})$ berechnet und die quadratische Distanz von 1 bestraft. Dadurch wird das weight clipping überfällig, weil der Critic von selbst lernt, die Bestrafung möglichst gering zu halten und deshalb die Distanz zu minimieren. λ ist ein Hyperparameter, der bestimmt, wie stark die Bestrafung ausfällt. In dem Paper wird $\lambda = 10$ gesetzt und in Experimenten über verschiedene Architekturen die Stabilität gezeigt.

3.6 SimGAN

¨Learning from Simulated and Unsupervised Images through Adversativ Training¨ wurde 2016 von Apple Inc. veröffentlicht[2]. In dem Paper wird ein Verfahren vorgestellt, synthetisch generierte Bilder realistischer werden zu lassen. Das Paper weist mit diesem Ziel in eine ähnliche Richtung, wie diese Arbeit. Es wird an dieser Stelle nicht weiter auf die Ergebnisse des Papers eingegangen. Es wurde eine Implementierung von SimGAN ausgetestet, doch die Resultate waren für die Problemstellung der Arbeit nicht zufriedenstellend. Ein wesentliches Problem war der L1-Loss, der, kombiniert mit einem Adversativen Loss, die Loss-Funktion für den Generator bildet. Der L1-Loss soll dafür sorgen, den Abstand zu dem Original-Bild nicht zu groß werden zu lassen. Aufgrund der Beschaffenheit der Originalbilder dieser Arbeit mit größeren leeren Flächen, stellte sich das als nicht zielführend hinaus. Die Architektur von SimGAN mit mehreren ResNet Layern wurde trotzdem weitgehend beibehalten.

4 Implementierung

Dieses Kapitel beschäftigt sich mit der konkreten Implementierung von LabelGAN. Dabei wird lediglich auf die Version eingegangen, welche die besten Resultate erzielt hat. Verschiedene andere Implementierungen, die in Tests weniger gut abschnitten, werden hier zugunsten der Einfachheit und Übersichtlichkeit nicht beschrieben. Im ersten Teil wird die Architektur beleuchtet, im zweiten Teil werden die unterschiedlichen Loss-Funktionen für die Netzwerke und wie diese untereinander zusammenhängen beschrieben.

4.1 Model

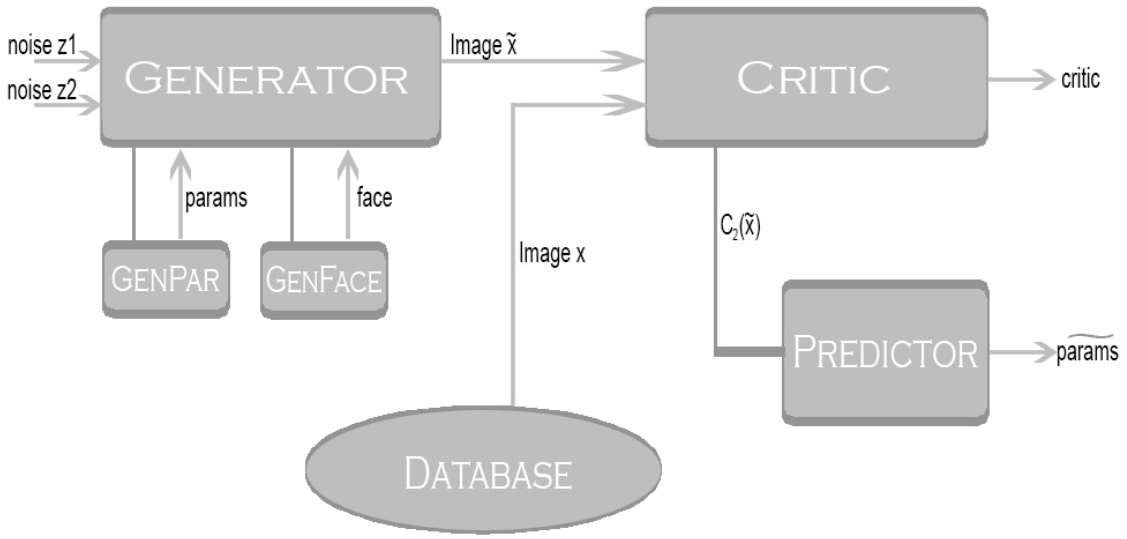

Abbildung 4.1: Vereinfachte Architektur von LabelGAN

Abbildung 4.1 zeigt eine vereinfachte Darstellung der Architektur von LabelGAN. Im folgenden wird erst das generelle Zusammenspiel der einzelnen Komponenten erklärt und danach auf die konkrete Umsetzung vom Generator, Critic und dem Predictor eingegangen.

Der Generator bekommt als Input zwei zehn dimensionale, gleich-verteilte Zufallsvektoren $z1$ & $z2$ übergeben. Der Generator funktioniert folgendermaßen:

$$G(GenFace(GenPar(z1)), z2) = \hat{x} \tag{4.1}$$

Der Noisevector $z1$ wird dem Unternetzwerk GenPar(Generate Parameters) übergeben, was ein Neuronales Netz mit zwei Fully-Connected-Layer ist. Es bestimmt die vierhundert Parameter *params*. Das Netzwerk *GenFace*(Generate Face) ist ein vor-trainiertes Netzwerk zur Generierung der synthetischen Gesichter. Es wurde auf dem Datensatz von 20.000 vor-generierten synthetischen Gesichtern im Stil von Abbildung 2.2 trainiert und ist daher in der Lage, eigenständig solche Gesichter zu produzieren. In Kapitel 6 wird eingehender besprochen, weswegen ein solches Netzwerk von Vorteil ist und nicht einfach die existenten synthetischen Bilder als Input für den Generator genommen werden konnten. Zur Generierung eines solchen Gesichtes bedarf es vierhundert Parameter, die von GenPar erstellt werden. Diese definieren Gesichtsform(0-199), Gesichtsfarbe(200-397), Winkel θ(398), Winkel ϕ(399), wobei die letzten beiden Parameter als Winkel die Lage des Gesichtes im Raum beschreiben. Der Generator ergänzt dann das synthetische Gesicht von GenFace um den zehn-dimensionalen Zufallsvektor $z2$, indem der Vektor hoch-skaliert und als Channel an das Bild an gehangen wird. Damit wird die Varianz der Ausgaben stark erhöht. Es folgen fünf Resnet-Layer zum Abschluss.

Das Critic-Netzwerk erhält abwechselnd Bilder x & $\hat{x}$. Wenn das Netzwerk $\hat{x}$ als Input erhält, kommt zusätzlich das Predictor-Netzwerk zum Einsatz. Das Critic-Netzwerk ist vollständig Convolutional. Die Ausgabe der zweiten Conv-Schicht wird dem Predictor übergeben. Die Intuition dahinter ist, dass das Critic-Netzwerk durch zwei Conv-Schichten das Generator-Bild soweit abstrahiert, dass lediglich die relevanten Features erhalten bleiben. Dadurch soll es dem Generator schwerer gemacht werden, die Parameterinformationen anderweitig zu codieren, als in den Hauptkomponenten des Bildes.

Das Predictor-Netzwerk besteht aus vier Convolutional-Layer und einem Fully-Connected

Layer und gibt *parâms* aus. *parâms* ist ein vierhundert-dimensionaler Vektor und soll *params* möglichst gut annähern. Der Generator wird bestraft, wenn das Predictor-Netzwerk dazu nicht in der Lage ist. Die Idee einer solches Architektur ist, den Generator zu zwingen die synthetischen Bilder soweit zu erhalten, dass das Predictor-Netzwerk in der Lage ist, die zugrunde liegenden Parameter aus dem Bild wieder zu extrahieren. Zu diesem Zweck wird die Loss-Funktion des Generators angepasst, wie in Abschnitt 4.2 beschrieben wird. Bevor auf die Loss-Funktionen näher eingegangen wird, wird der Aufbau vom Generator und Critic näher beleuchtet.

4.1.1 Generator

Die Architektur des Generators ist durch das Paper von Apple Inc. Abschnitt 3.6 inspiriert. Idee ist, das synthetische Gesicht bei dem Prozess der Generierung stärker zu gewichten, als bei ausschließlich Conv-Layern möglich. Daher werden fünf Resnet-Blöcke verwendet, deren Aufbau Abbildung 4.2 veranschaulicht. Wie zu sehen ist, wird Batch-

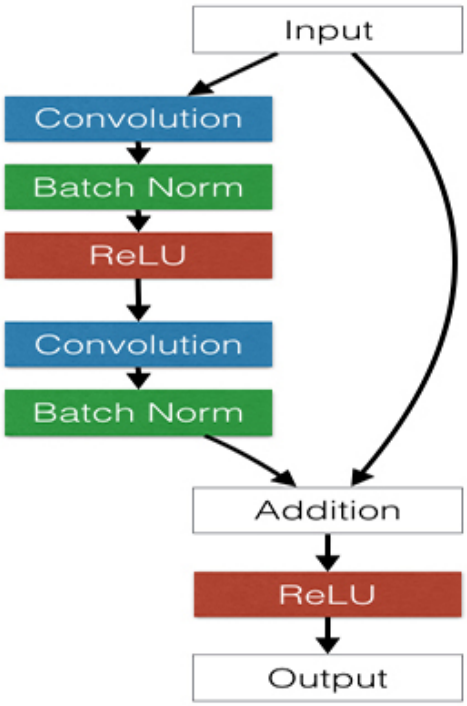

Abbildung 4.2: Resnet Aufbau Quelle: http://torch.ch/blog/2016/02/04/resnets.html

Normalisierung im Generator eingesetzt. Außerdem ReLU-Aktivierungsfunktionen. Nach

den Resnet-Blöcken folgt eine weitere Conv-Schicht und eine Tanh-Aktivierung kombiniert mit einer ReLU-Aktivierung, um die Ausgabe auf das Intervall $[0, 1]$ zu begrenzen.

4.1.2 Critic

Das Critic-Netzwerk orientiert sich beim Aufbau an einer klassischen DCGAN-Architektur. Sieben aufeinanderfolgende Conv-Layer skalieren den Input auf eine 2x2 Map hinunter. Es wird, anders als beim Generator, keine Batch-Normalisierung genutzt. Wie im Paper "Improved WGAN" hingewiesen, kann es sonst zu Konvergenz Problemen kommen. Außerdem werden LeakyReLU Aktiverungsfunktionen eingesetzt.

4.2 Learning

Im Folgenden werden die Loss-Funktionen für Generator, Critic und Predictor präsentiert und besprochen.

Wie in vorigen Kapiteln erklärt, ist LabelGAN eine Erweiterung eines Improved-Wasserstein-GAN. Entsprechend ist die Loss-Funktion vom Critic die gleiche, wie in Abschnitt 3.5 beschrieben:

$$D_{Loss} = \mathbb{E}_{x \sim P_{real}}[D(x)] - \mathbb{E}_{\tilde{x} \sim P_\theta}[D(\tilde{x})] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{4.2}$$

Im Training wird angestrebt, diese Funktion zu minimieren. Dadurch wird der erste Term, der die realen Bilder beschreibt, minimiert, der zweiter Term für die generierten Bilder maximiert und der letzte Term für die Gradienten Bestrafung ebenfalls minimiert.

Spannender ist der Loss des Generators, der um einen Term ergänzt wurde, der den Fehler beschreibt, den der Predictor bei der Extraktion der Labels aus dem Bild macht.

$$G_{Loss} = \mathbb{E}_{\tilde{x} \sim P_\theta}[D(\tilde{x})] + P_{Loss} \tag{4.3}$$

Während das Critic-Netz den ersten Term zu maximieren sucht, möchte der Generator diesen Term minimieren, was zu dem klassischen Min-Max Spiel bei Adversativem Training gehört. Dieser Term entspricht dem WassersteinGAN. Ergänzt wird der Loss jedoch um P_{Loss}, was der Loss des Predictors ist, den der Generator zu minimieren sucht. Minimal ist der Loss, wenn der Predictor in der Lage ist, die Labels korrekt auszulesen. Der Generator ist dadurch forciert, die Bilder so zu generieren, dass der Predictor diese korrekt interpretieren kann.

Der Loss des Predictor-Netzes ist der L2-Abstand der echten und vorhergesagten Parameter des synthetischen Gesichtes.

$$P_{Loss} = ||params - \hat{params}||_2 \tag{4.4}$$

Wobei *params* die wahren Parameter und $\hat{params}$ die vermuteten Parameter sind.

Als Optimierer wird für alle Funktionen der ADAM-Optimizer verwendet.[8]

Während des Lernprozesses wird das Critic-Netz mehr als die anderen Netze trainiert. In jeder Trainingsiteration wurde das Netz dreifach trainiert.

5 Evaluation

In diesem Kapitel wird die Performance von LabelGAN auf unterschiedlichen Datensätzen betrachtet und evaluiert.

5.1 MNIST

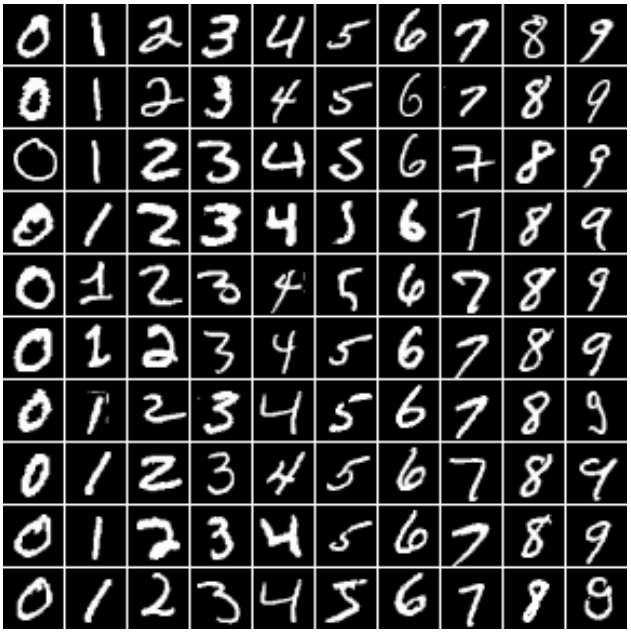

Abbildung 5.1: Exemplarische Ausgabe aus dem MNIST-Datensatz

Der MNIST Datensatz besteht aus 60.000 Trainings - und 10.000 Testbildern von handgeschriebenen Ziffern einer Größe von 28x28 Pixeln.Abbildung 5.1 zeigt beispielhaft, wie diese Ziffern aussehen können. Die geringe Größe der Bilder, sowie der einfache Aufbau mit weißer Schrift auf schwarzem Grund machen diesen Datensatz zu einem idealen Testdatensatz für die generelle Umsetzbarkeit von LabelGAN.

Die Performance von LabelGAN ist abhängig von zwei Parametern. Zum einen die Qualität der Bilder. Erwartet wird hier, dass sich die generierten Bilder nicht mehr von den Originalbildern unterscheiden lassen. Zum anderen die Varianz der erzeugten Bilder. Der Generator soll in der Lage sein, wie in Abbildung 5.1 unterschiedliche Handschriften nachzuahmen.

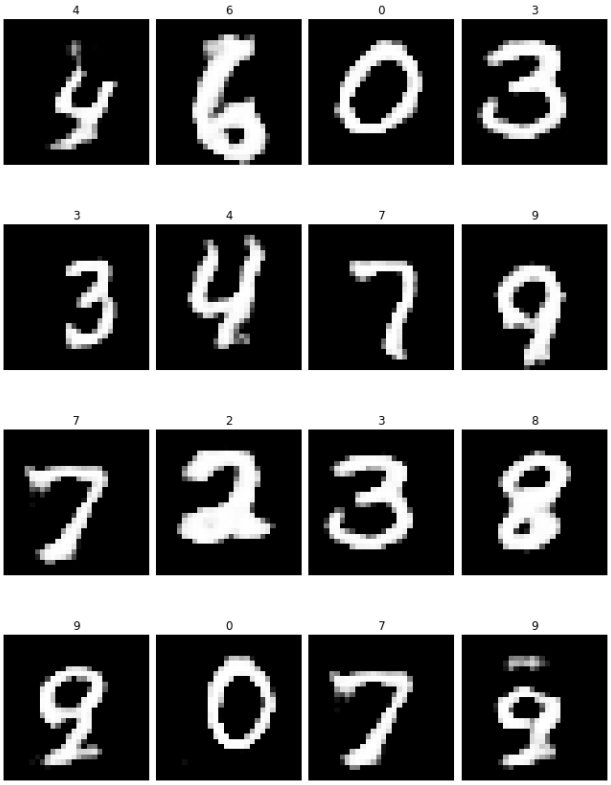

Abbildung 5.2: MNIST Ziffern von LabelGAN generiert

Abbildung 5.2 zeigt generierte Ziffern nach fünf Iterationen. Besondere Beachtung verdient hier die Unterschiedlichkeit zwischen den gleichen Ziffern, die sich in Dicke, Form und Position unterscheiden. In kleinen Zahlen oberhalb der Bilder stehen die Labels, die vorhergesagt wurden. Die Ausfallquote liegt hier bei unter 2%.

Als Ausgangsbilder werden die Mean-Bilder des Datensatzes verwendet. Diese werden erzeugt, indem alle gleichen Ziffernbilder aufeinander addiert und dann durch die Anzahl der Bilder geteilt werden. Diese Mean-Bilder definieren damit gleichzeitig die Klasse des Bildes. Die Aufgabe des Generator ist, die Klasse beizubehalten. Der Rest der Architektur blieb weitgehend gleich, bis auf eine Änderung an dem Generator. Da zu jeder Ziffer nur

ein Mean-Bild existiert, muss der Input für den Generator vergrößert werden. Anstatt dies manuell durch eine Pipeline vor dem Generator zu machen, wird der Generator um eine Funktion zur Verzerrung des Eingabebildes erweitert, einen Diffeomorphismus. Ohne explizit auf diese Funktion einzugehen, bietet sie eine Möglichkeit, ein Bild wie in Abbildung 5.3 zu verzerren. Die Parameter kann der Generator lernen, um die Verzerrung

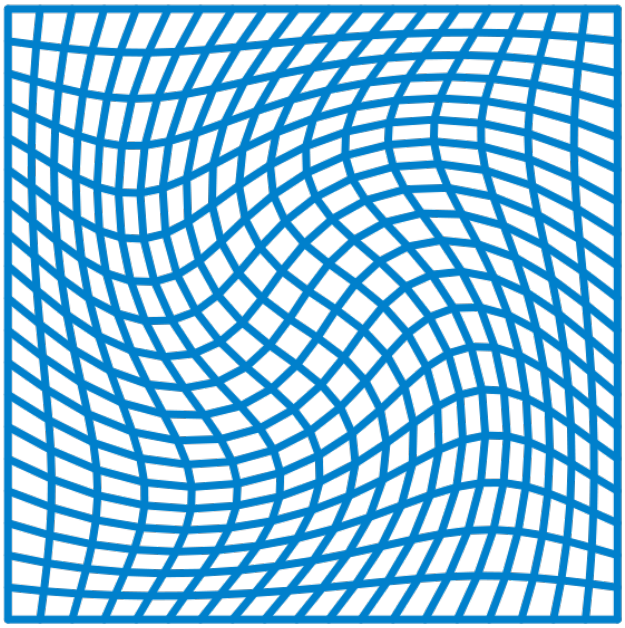

Abbildung 5.3: Quelle: https://de.wikipedia.org/wiki/Diffeomorphismus

in Stärke anpassen zu können.

Schon nach wenigen Iterationen produzierte LabelGAN auf dem relativ einfachen Datensatz MNIST Bilder von sehr guter Qualität, während das Label beibehalten und korrekt vom Predictor bestimmt werden.

5.2 CelebA+IMDB

Eine weitaus interessantere und komplexere Domäne als handgeschriebene Zahlen sind Bilder von Gesichtern. Der CelebA Datensatz[1] besteht aus etwa 200.000 Bildern von Prominenten. Der IMDB Datensatz[2] besteht aus 500.000 Bildern, was zusammen über

1 http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
2 https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/

700.000 Bilder von unterschiedlichsten Gesichtern ergibt. Leider gab es Schwierigkeiten mit beiden Datensätzen, dazu in Kapitel 6 mehr.

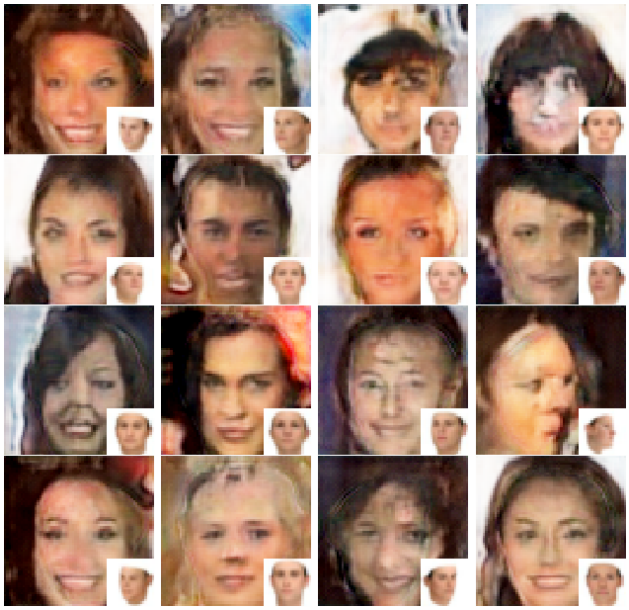

Abbildung 5.4: LabelGAN auf CelebA+IMDB

Abbildung 5.4 fasst die Resultate gut zusammen. Die Qualität der Gesichter ist zumeist gut, wenn auch nicht in der Qualität, wie zum Beispiel BeGAN [3] in der Lage ist zu produzieren. Abbildung 5.5 zeigt den Critic-Loss von LabelGAN. Der Critic-Loss konvergiert

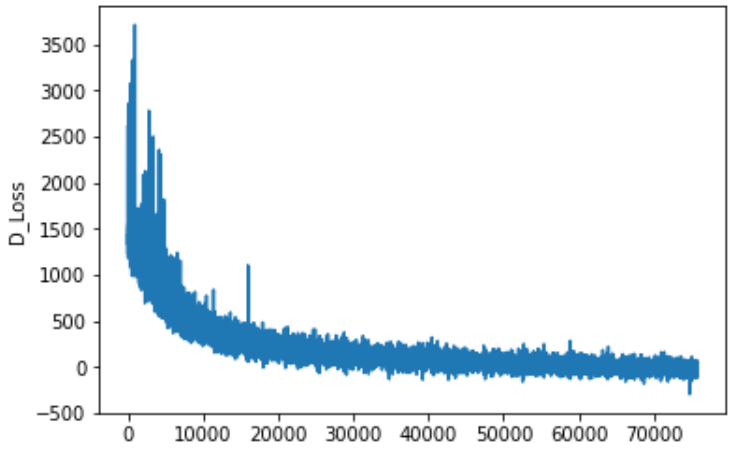

Abbildung 5.5: Critic Loss von LabelGAN

zuverlässig gegen ein Minimum. Dafür bleiben die Labels nur unzureichend erhalten. Der

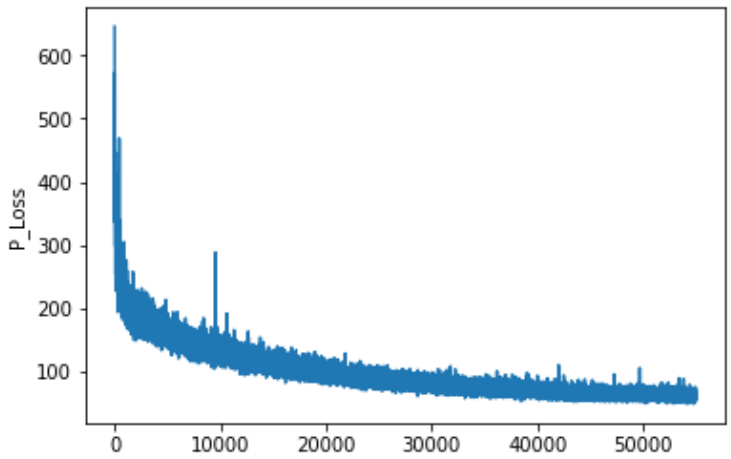

Abbildung 5.6: Predictor Loss von LabelGAN

Predictor-Loss 5.6 konvergiert gleichzeitig ebenfalls gegen ein Minimum, was bedeutet, dass das Predictor-Netzwerk aus den Bildern die Parameter auszulesen weiß. Aus der Kombination der beiden Plots lässt sich schlussfolgern, dass der Generator einerseits lernt, realistischere Bilder zu erzeugen und andererseits lernt, die Parameter der Bilder so zu codieren, dass der Predictor diese lesen kann.

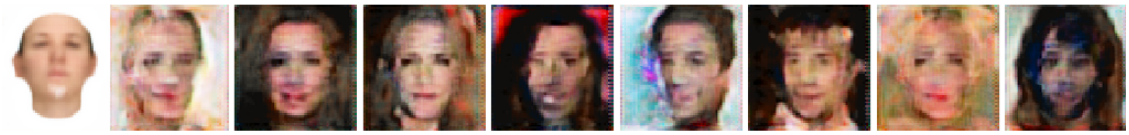

Abbildung 5.7: Ausgabe vom Generator mit gleichem synthetischem Gesicht links

Die Unterschiedlichkeit der generierten Bilder ist bei gleichem Ausgangsbild hoch, was Abbildung 5.7 zeigt. Das zeigt, dass der zehn-dimensionale Noise-Vektor ausreichend Varianz in den Input bringt.

Eine Interpolation über einen Winkel bei gleichen Restparametern zeigt Abbildung 5.8. Die vierhundert Parameter gleichen sich bei allen acht synthetischen Gesichtern, lediglich ϕ wird interpoliert. Zuerst lässt sich feststellen, dass die Qualität in Hinblick auf die Echtheit bei stärkerem Winkel abnimmt. Eine Höhere Qualität entsteht am am ehesten bei frontalen Gesichtern. Die Abbildung zeigt außerdem deutlich, dass zum Beispiel die Hautfarbe bei den generierten Gesichtern stark unterschiedlich ist, obwohl die Parameter für alle Gesichter gleich sind. Da der Predictor dennoch die richtigen Parameter vorhersagen kann,

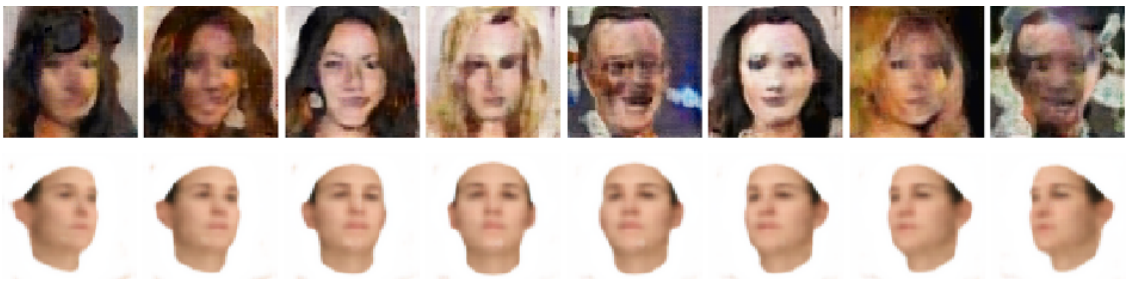

Abbildung 5.8: Interpolation über Winkel bei gleichen Restparametern

müssen sie offensichtlich anderweitig codiert sein. Diesem Problem wird sich im nächsten Kapitel ausführlich gewidmet. Alle Bilder in diesem Abschnitt sind mit der beschriebenen Architektur in Kapitel 4 entstanden. Das heißt im speziellen für den Predictor, es wurde die zweite Schicht vom Critic als Input gewählt. Das ist wichtig, da andere Versuche teilweise dazu führten, dass keine Konvergenz erreicht wurde.

6 Diskussion und Ausblick

6.1 Probleme

Dieses Kapitel beschäftigt sich mit einer Reihe von Problemen, die während der Arbeit aufkamen. Es werden Lösungsvorschläge präsentiert und Optionen für weiterführende Arbeiten besprochen. Besonderes Augenmerk wird auf die Frage gelegt, weswegen LabelGAN bei der Generierung von Gesichtern die Informationen zerstört und der Predictor dennoch in der Lage ist, diese wieder korrekt aus dem Bild zu lesen.

6.1.1 Qualität der Bilder

Ein hoher Detailgrad der generierten Bilder ist notwendig. Daher wurde viel Zeit in die Suche nach einer guten Architektur für das zu Grunde liegende GAN investiert.Abbildung 6.1 zeigt die Qualität der Bilder eine DCGAN-Architektur des Generators auf Basis von Mean-Bildern des CelebA-Datensatzes nach 50.000 Iterationen. Alle weiteren Experimente konnten keine wesentliche Steigerung der Qualität erbringen. Die Qualität der Bilder ist ein Kriterium für die Wahl der Labels. Es macht keinen Sinn, komplexe Emotionen wie Wut, Freude oder Traurigkeit zu wählen, weil dafür die Qualität der Ausgabe nicht gut genug ist. Die Blickrichtung und die Hautfarbe(Textur) des Gesichts sind hingegen in den Bildern gut zu erkennen und eignen sich daher besser. Für komplexere Labels wie Alter o.ä. muss zuerst die generelle Qualität der Bilder gesteigert werden. Es gibt verschiedene Paper, die sich mit der Generierung qualitativ sehr hochwertiger Gesichter beschäftigen und die in der Arbeit berücksichtigt wurden, leider konnte die Qualität dennoch nicht reproduziert werden. Eine mögliche Ursache dafür könnte ein unzureichender Datensatz gewesen sein. Theoretisch existieren mit CelebA und IMBD zwei große Datensätze mit einer Menge an Gesichtern, jedoch weisen beide einige erhebliche Mängel in Bezug auf

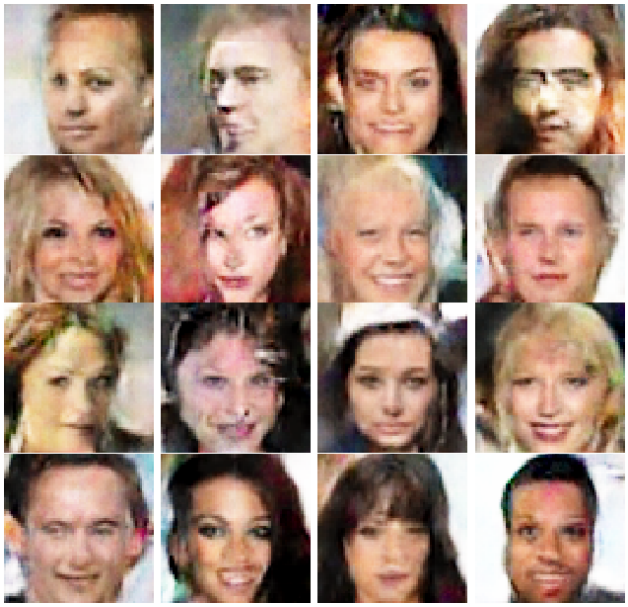

Abbildung 6.1: CelebA Image Synthese generiert nach 58.000 Iterationen

die Problemdomäne dieser Arbeit auf. CelebA, als Datensatz mit über 200.000 Bildern von Prominenten, zeigt diese in nahezu allen Bildern frontal, d.h. bei der Generierung von Bildern auf diesem Datensatz ist der Generator gezwungen, die Blickrichtung ebenfalls frontal zu wählen. Das ist gerade insofern problematisch, da bei den synthetischen Gesichtsbildern zum Teil seitliche Ansichten dabei sind und der Winkel des Gesichts ein Label ist.Abbildung 6.2 visualisiert das Problem.

Um eine höhere Vielfältigkeit zu erhalten, wurde der CelebA Datensatz durch den IMDB Datensatz mit weiteren 500.000 Bildern ergänzt. Leider hatte jedoch auch dieser Datensatz seine Nachteile. Bei genauerer Betrachtung der Bilder stellte sich heraus, dass unter diesen eine hohe Ausfallquote existiert, auf denen also kein Gesicht, sondern etwas anderes zu sehen ist.A.1 Das wirkte sich beim Training auf diesem Datensatz erheblich auf die Bilder aus. Da für die Suche nach weiteren Datensätze die Zeit zu knapp wurde, wurde eine Kombination aus beiden Datensätzen gewählt. Die Hoffnung war, beide Schwächen gegeneinander ausspielen zu können. Außerdem wurde die Architektur des Generators auf dieses Problem angepasst. Wie in Kapitel 4 beschrieben, besitzt der Generator ein vor-trainiertes Unternetzwerk *GenFace*, welches ein synthetisches Gesicht auf Basis der vierhundert Parameter erstellt. Eine berechtigte Frage ist, weswegen nicht die bereits vorhandenen synthetischen Bilder als Input für den Generator genommen wurden, da sich dadurch der Umweg über ein Unternetzwerk erspart bliebe. Der Grund liegt in

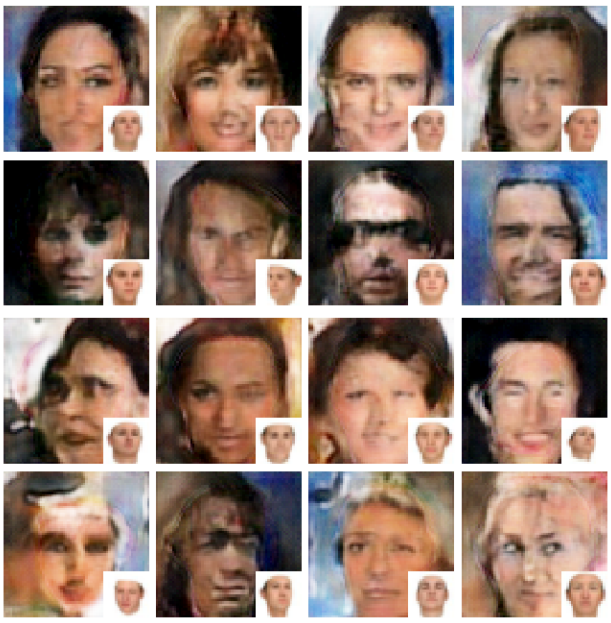

Abbildung 6.2: Winkelproblematik bei reinen CelebA Daten

den beschriebenen Problemen der Datensätze. Abbildung 6.3 zeigt exemplarisch ein

Abbildung 6.3: Synthetisches Gesicht

synthetisches Gesicht, welches für den Generator unbrauchbar ist. Die Winkel des Gesichts sind in diesem Bild erheblich unterschiedlich zu denen aus CelebA. Daher war die Idee, den Generator selber die Parameter *params* trainieren zu lassen, nach denen die Gesichter generiert werden, sodass nur solche synthetischen Gesichter entstehen, die der Generator verarbeiten kann.

Dieser Ansatz führte während des Trainings zu neuen Problemen. Ließ man den Generator das Netzwerk *GenPar* zu Generierung von *params* beliebig lange trainieren, entstanden uniforme, einförmige, kugelähnliche Gesichter. Diese waren für den restlichen Prozess nicht zu gebrauchen. Eine mögliche Lösung war, *GenPar* nur für die ersten Iterationen trainieren zu lassen. Es lernt schnell, extravagante Gesichter, in Bezug auf die Winkel, zu vermeiden. Danach kann das Netzwerk aus dem Lernprozess ausgeschlossen werden.

An dieser Stelle herrscht jedoch auf jeden Fall noch Optimierungsbedarf. Nach meiner Einschätzung kann ein besserer Datensatz die Qualität erheblich steigern.

6.1.2 Codierte Labels

Ein wesentliches Problem bei der Arbeit mit Vorhersage von Labels aus den generierten Bildern ist die unerwünschte Codierung der Labels in Pixeln oder nebensächlichen Bildstrukturen. Zweierlei hat der Generator bei der Generierung des Bildes zu beachten: zum einen die Ähnlichkeit zu den Trainingsdaten, zum anderen die Information über die gegebenen Labels so zu erhalten, dass das Predictor-Netzwerk diese wieder rekonstruieren kann. Im idealen Fall würde der Generator die Winkel ϕ und θ, die in der Hauptkomponentenanalyse(PCA) die Winkel des Gesichtes im Raum angeben, genau über die entsprechende Lage des Gesichtes codieren. Der Predictor könnte die Lage dann zurückrechnen und die Winkel bestimmen. Bei Tests hat sich jedoch herausgestellt, dass der Generator die Bilder anders produziert. Bei der Generierung des Gesichtes ignoriert er die Labels weitgehend und codiert alle nötigen Informationen für den Predictor stattdessen in Metainformationen des Bildes, die das Gesicht nicht wesentlich beeinflussen. Auf diese Weise muss der Predictor lediglich diese Informationen auslesen und kann so die Labels vorhersagen.

Diesem Verhalten muss entgegengewirkt werden, damit der Generator gezwungen wird, Labels in dem Hauptelement(dem Gesicht) zu codieren. Zwei Ansätze wurden ausgetestet:

- Addition von Rauschen
- Schicht vom Critic als Eingabe für den Predictor

Für die erste Variante wurde auf jeden Pixel des vom Generator erzeugten Bildes $\hat{x}$ gleichverteiltes Rauschen aus dem Intervall $[0, 0.5]$ addiert. Die Idee dazu ist, dass dem Generator die Möglichkeit genommen wird, die Parameter Informationen in einzelnen Pixeln zu codieren, weil diese durch das Rauschen leicht verschoben werden. Erstaunlicherweise konvergiert der Predictor dennoch, ohne dass die Qualität der Bilder zugenommen hätte. In dieser Arbeit wurde das Verhalten bei Addition von Rauschen nicht weiter beleuchtet, sondern mehr Wert auf die Zweite Variante gelegt. Für mögliche folgende Arbeiten wäre es interessant, dieses Verhalten nachzuvollziehen. Eventuell muss das Rauschen schlicht intensiviert werden oder aber der Generator findet andere Wege, die Parameter zu codieren.

Der zweite Ansatz wurde implementiert und ausführlicher getestet. Alle bisher vorgestellten Ergebnisse wurden wie in Kapitel 4 beschrieben, mit der zweiten Conv-Schicht vom Critic Netz als Input für den Predictor gewonnen. Wie die Ergebnisse allerdings gezeigt haben, scheint die Abstraktion des Bildes bei dieser Architektur nicht groß genug, sodass der Generator lernen kann, die Parameter für den Predictor in dem Bild zu codieren. Eine intuitive Idee ist es daher, tiefere Schichten vom Critic zu wählen. Je tiefer die Schicht, desto weniger Informationen, die dafür aber präziser und spezieller sind, hält das Critic-Netz bereit. Folgende Resultate wurden mit der dritten Schicht erzielt. Abbildung 6.4 zeigt eine Interpolation über einen Winkel bei ansonsten gleichen Parametern. Das GAN durchlief 100.000 Trainings-Iterationen. Anders als bei den vorigen Resultaten lässt sich

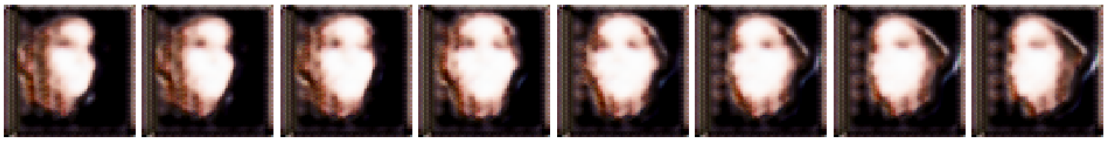

Abbildung 6.4: Interpolation eines Gesichts mit 3.Schicht als Predictor-Input nach 100.000 Iter.

hier eindeutig die Codierung des Winkels im Bild nachvollziehen. Dafür ist der Generator nicht in der Lage, die Bilder an Originaldaten anzunähern, sondern erhält weitgehend die Struktur des Eingabebildes. Verdeutlichen tun dies die Plots des Losses vom Critic und Predictor. Abbildung 6.5 und Abbildung 6.6 zeigen den Loss. Die unterschiedliche Skalierung der Achse hängt mit dem häufigerem Training von Critic gegenüber Predictor zusammen. Beide Plots entstanden im gleichen Trainingszyklus, nur dass das Critic-Netz dreimal häufiger pro Iteration trainiert wurde. Offensichtlich konvergiert keine der beiden

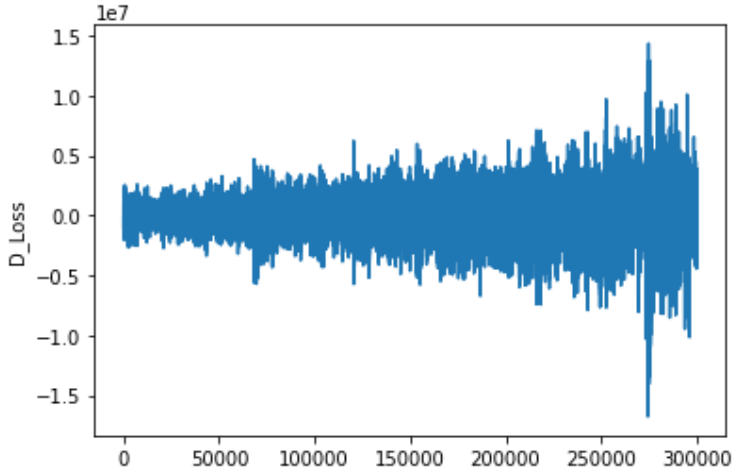

Abbildung 6.5: Critic-Loss mit 3.Schicht als Predictor-Input

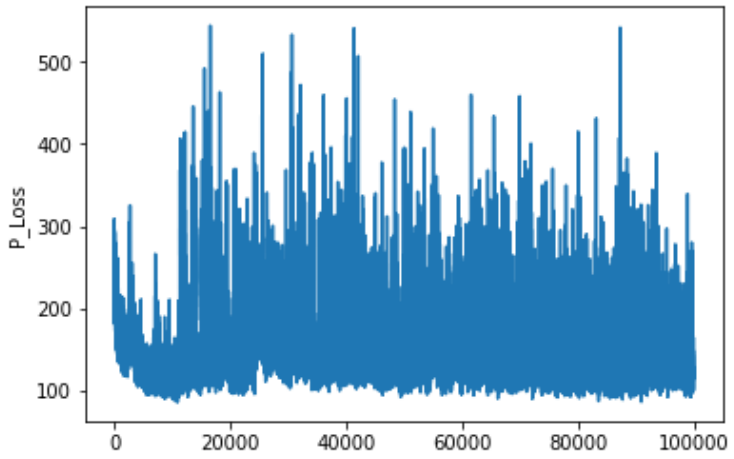

Abbildung 6.6: Predictor-Loss mit 3.Schicht als Predictor-Input

Funktionen. Offensichtlich ist der Predictor nicht mehr in der Lage, die Parameter korrekt zu bestimmen. Der Loss bleibt konstant hoch, was den Generator zwingt, die Bilder sehr nah an dem Ursprungsbild *face* zu orientieren. Dadurch ist das Critic-Netz in der Lage, mit fortschreitendem Training echte und synthetische Bilder perfekt zu unterscheiden. Das zeigt der expandierende Critic-Loss.

Hier gibt es viel Spielraum für Experimente. Möglicherweise muss das Predictor-Netz tiefer gebaut werden, um die Parameter dennoch zu extrahieren. Doch ist es zu stark, dann geschieht das Gleiche wie bei der jetzigen Architektur und dem Generator wird die Möglichkeit gegeben, die Parameter anders zu codieren. Durch die Komplexität und den

aufwendigen, zeitintensiven Lernprozess, ist es auch eine Zeitfrage, wie viel Ressourcen in, möglicherweise nicht zielführende, Experimente mit unterschiedlichen Architekturen gelegt wird. Der Lernprozess von GAN ist schwer einsehbar und daher schwierig zu kontrollieren. An welcher Stelle welches Netzwerk letzten Endes zu stark oder zu schwach ist oder ob die Idee selbst so nicht umsetzbar ist, kann in dieser Arbeit nicht abschließend beantwortet werden.

Es bleiben Fragen und Optionen für anschließende Arbeiten.

6.2 Perspektive

Machine Learning und insbesondere dem Bereich Deep Learning wird zur Zeit in der Forschung sehr viel Aufmerksamkeit gewidmet. Entsprechend werden monatlich Verbesserungen und neue Ansätze präsentiert. Generative Adversative Netze machen in diesem Punkt keine Ausnahme. Während dieser Arbeit wurde zum Beispiel das Paper ¨Improved Training of Wasserstein GANs¨ veröffentlicht. Dies geschah zu einem Zeitpunkt, wo es noch berücksichtigt werden konnte und in diese Arbeit eingeflossen ist. Inzwischen gibt es eine Reihe neue Paper, die verschiedentliche Verbesserungen vorschlagen. Ein interessanter neuer Ansatz ist, die Wasserstein-Distanz durch die Cramer-Distanz zu ersetzten und damit bessere Konvergenz zu erzielen.[9] Ein möglicher nächster Schritt könnte sein, die Cramer-Distanz zu implementieren und zu schauen, ob diese besser als die Wasserstein-Distanz konvergiert, wenn tiefe Schichten vom Critic als Eingabe für den Predictor genommen werden.

Soll die Idee mit den synthetischen Gesichtern weiter verfolgt werden, dann ist es eine Überlegung wert, Ausschau nach passenderen Datensätzen zu halten. Einen guten Überblick bietet zum Beispiel die Website www.face-rec.org/databases [1]. Bei den meisten der auf der Website vorgestellten Datenbanken muss ein Antrag für die Verwendung gestellt werden. So zum Beispiel für den MegaFace Datensatz [2], für den ein Antrag gestellt wurde. Erst zum Ende dieser Arbeit wurde der Antrag genehmigt, sodass er nicht mehr in die Arbeit einfließen konnte. Die Bilder sind verfügbar und es wäre interessant, die Performance von LabelGAN auf diesen Daten zu testen.

1 http://www.face-rec.org/databases/
2 http://megaface.cs.washington.edu/

Großes Potential liegt nach meiner Einschätzung in der Architektur des Critic-Netzwerkes. Bisher wird eine DCGAN-Architektur verwendet, welche aus sieben Schichten besteht und mit jeder Schicht die Größe halbiert. So werden die Informationen, die das Eingabebild liefert, ebenen-weise komprimiert und reduziert. Es ist fraglich, welche Informationen das Critic Netz als wichtig für die Unterscheidung zwischen realen und generierten Bildern einstuft. Das heißt es bleibt unklar, welche Informationen verworfen werden und welche auch in tieferen Schichten erhalten bleiben. Betrachtet man die Ergebnisse aus den Tests mit der dritten Schicht, so lässt sich erkennen, dass in dieser offensichtlich Informationen über die Parameter nicht mehr erhalten sind, weswegen der Predictor versagt bei dem Versuch, diese zu erkennen. Das ist ein Unterschied zu der zweiten Schicht, in der alle nötigen Parameter Informationen noch erhalten sind.

Die DCGAN Architektur sollte daher möglicherweise ersetzt werden. Eine Alternative ist ein Auto-Encoder Netz. Dieses könnte auf einem Datensatz vortrainiert werden, sodass es die Bildes codiert. Im Anschluss wird es in den Adversativen Prozess eingebunden.Mit BeGAN[3] wird vorgemacht, wie ein Auto-Encoder als Diskriminator verwendet werden kann, doch es bleibt offen, wie der Predictor am besten von einer solchen Architektur profitiert. Spätere Arbeiten können an dieser Stelle ansetzen.

Literaturverzeichnis

[1] T. L. Leon Sixt, Benjamin Wild, *RenderGAN: Generating Realistic Labeled Data*. https://arxiv.org/pdf/1611.01331.pdf, 2017.

[2] O. T. e. a. Ashish Shrivastava, Tomas Pfister, *Learning from Simulated and Unsupervised Images through Adversarial Training*. https://arxiv.org/pdf/1612.07828.pdf, 2016.

[3] L. M. David Berthelot, Thomas Schumm, *BEGAN: Boundary Equilibrium Generative Adversarial Networks*. https://arxiv.org/pdf/1703.10717.pdf, 2017.

[4] M. M. e. a. Ian J. Goodfellow, Jean Pouget-Abadie, *Generative Adversarial Nets*. https://arxiv.org/pdf/1406.2661.pdf, 2014.

[5] M. A. e. a. Ishaan Gulrajani, Faruk Ahmed, *Improved Training of Wasserstein GANs*. https://arxiv.org/pdf/1704.00028.pdf, 2017.

[6] L. M. Alec Radford, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. https://arxiv.org/pdf/1511.06434.pdf, 2016.

[7] L. B. Martin Arjovsky, Soumith Chintala, *Wasserstein GAN*. https://arxiv.org/pdf/1701.07875.pdf, 2017.

[8] J. L. B. Diederik P. Kingma, *ADAM: A Method for Stochastic Optimization*. https://arxiv.org/pdf/1412.6980.pdf, 2017.

[9] W. D. Marc G. Bellemare, Ivo Danihelka, *The Cramer Distance as a Solution to Biased Wasserstein Gradients*. https://arxiv.org/pdf/1705.10743v1.pdf, 2017.

A Anhang

A.1 Code

Der gesamte Code, der für diese Arbeit geschrieben wurde, befindet sich zum einen auf dem Server "thekla.imp.fu-berlin.de". Das Hauptprogramm "LabelGAN-CelebA-Faces.ipynb" ist in einem Jupyter Notebook geschrieben. Zum anderen liegt der Code zusätzlich in einem Repository auf github.com [1]. Dieser Code lässt sich nach dem herunterladen jedoch nicht korrekt ausführen, da die nötigen Datenbanken CelebA+IMDB fehlen. Diese müssen manuell geladen und im Code auf den Speicherort verwiesen werden.

Auf dem Thekla-Server finden sich weitere Jupyter Notebooks. Diese entstanden während des Entwicklungsprozesses und markieren abgeschlossene Stadien.

A.2 IMDB-Bilder

1 `https://github.com/lodrice/LabelGAN`

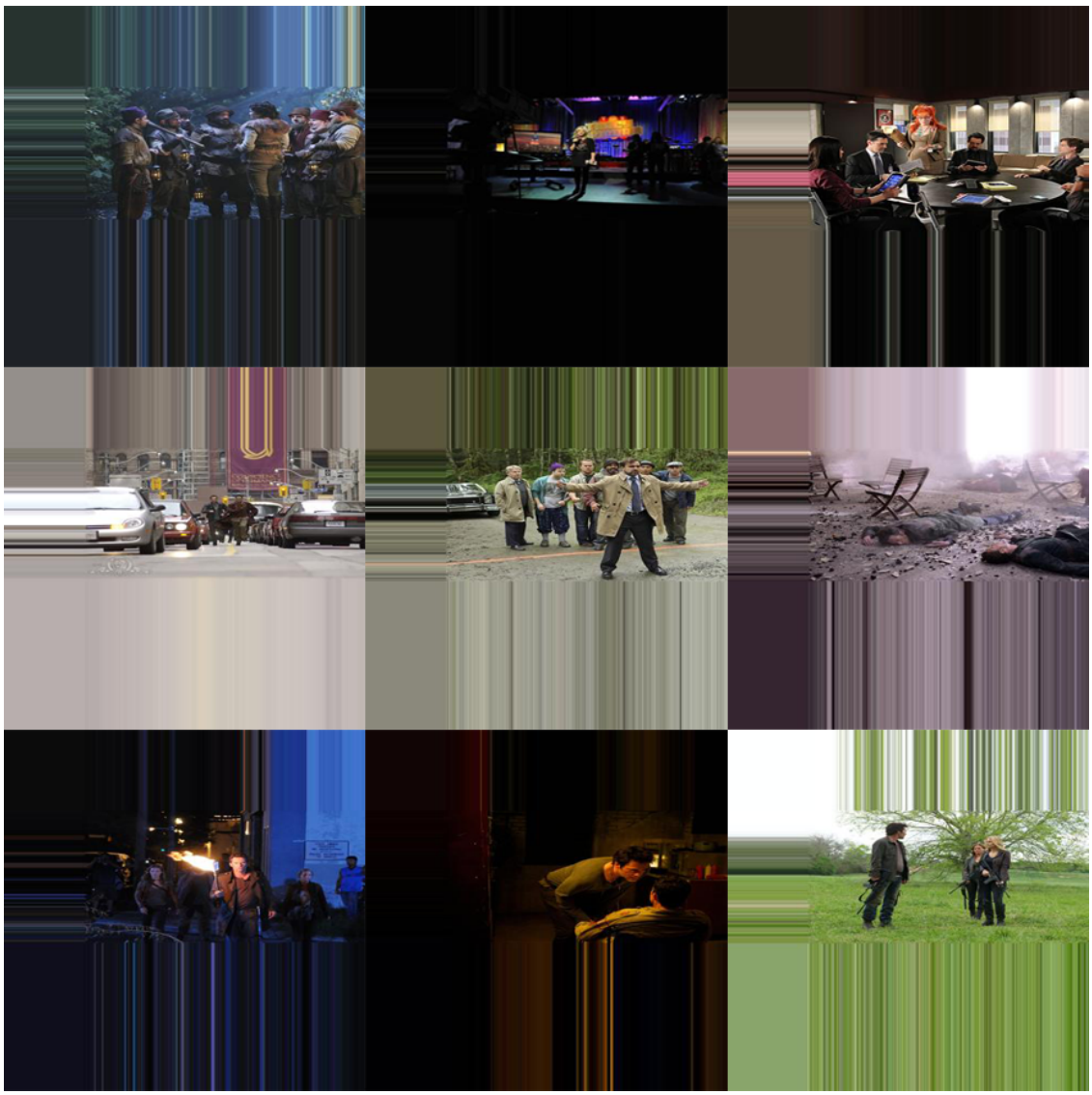

Abbildung A.1: Exemplarische Darstellung von Bildern aus dem IMDB Datensatz, die keine eindeutigen Gesichter zeigen. Während des Trainings führte dies zur Generierung von unerwünschten Artifakten

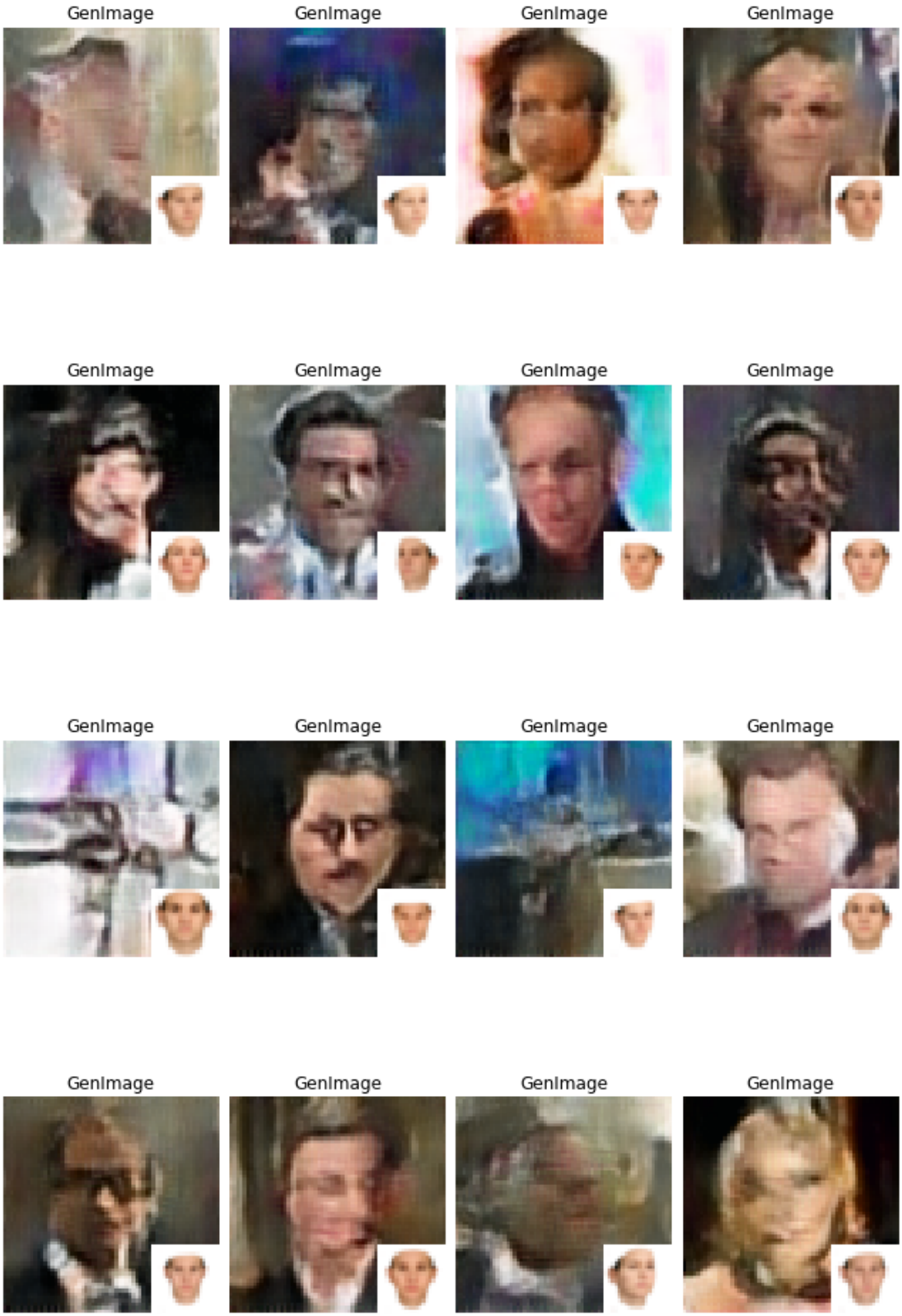

Abbildung A.2: Exemplarische Darstellung von generierten Bildern auf Basis des IMDB Datensatzes. Durch falsche Bilder in dem Datensatz entstanden teilweise unerwünschte Artifakte, die keine Ähnlichkeit zu Gesichtern besitzen