Table of Contents

Abb. Abbildung
C-GAN Conditional Adversarial Networks
EMD Earth Mover Distanze
KNN Künstliches Neuronales Netzwerk
GAN Generativ Adverserial Network
DC GAN Deep Convolution Generative Adverserial Network

Rekonstruktion von 3D-Modellen mit Generative Adversarial Networks

Andreas Wiegand
Masterthesis im Bereich künstliche Intelligenz

Gutachterin: Prof. Dr. Ute Schmid

in Zusammenarbeit mit
Fraunhofer-Institut für Integrierte Schaltungen IIS/EZRT
betreut von Oliver Scholz und Franz Uhrmann

Fakultät für Angewandte Informatik, insbesondere Kognitive Systeme
Otto-Friedrich-Universität Bamberg
Bamberg
28.Februar.2019

Rekonstruktion von 3D-Modellen mit Generative Adversarial Networks

von Andreas Wiegand

Masterthesis im Bereich künstliche Intelligenz

Zusammenfassung. Generative Adverserial Network(GAN) ist ein künstliches neuronales Netzwerk(ANN) aus dem Bereich der generativen Modelle. Die Aufgabe des GANs ist es, die Wahrscheinlichkeitsverteilung von Trainingsdaten zu erlernen und anschließend neue Samples aus dieser Wahrscheinlichkeitsverteilung zu generieren. Ziel der hier vorliegenden Arbeit ist es, das Konzept der GANs auf 3D-Punktwolken von Tabakblättern anzuwenden, um die Wahrscheinlichkeitsverteilung von 3D-Daten zu erlernen. Des weiteren soll geprüft werden, ob es mit Hilfe von GANs möglich ist, Punktwolken zu ergänzen, welche beim Erfassen durch ein Scanverfahren unvollständig erfasst worden sind. Diese Unvollständigkeit kann zustande kommen, wenn Artefakte beim Scannen die Sicht auf das zu scannende Objekt verdecken und dieses dann unvollständig aufgenommen wird. Da im verwendeten Trainingsdatensatz nicht genügend valide Daten vorhanden sind, wurde die Datenerhebung durch das Einfügen von Artefakte in die Tabakpunktwolken simuliert. Die Differenzmenge zwischen Tabakpunktwolke und Artefakt simuliert dann das nicht vollständig gescannte Tabakblatt. Die Ergebnisse dieser Arbeit zeigen auf, dass es möglich ist die Punktwolken von Tabakblättern zu erlernen und neue Blätter zu produzieren. Die Qualität der generierten Daten hat jedoch noch Potential zur Steigerung, vergleicht man diese mit den Ergebnissen von Achlioptas, Panos und Diamanti(Achlioptas, Diamanti, Mitliagkas, & Guibas, 2018). Die vollständige Rekonstruktion mit Hilfe von GANS konnte nicht produziert werden. Es zeichnet sich jedoch eine Qualitätssteigerung der Rekonstruktion, durch unterschiedliche Aufbauten von GANs, ab und die Möglichkeit durch eine Veränderung des GAN Aufbaus in zukünftigen Arbeiten, die vollständige Rekonstruktion zu ermöglichen. Durch einen Nebeneffekt, welcher durch die Zielfunktion von Autoencoder gegeben ist, ist es möglich Lücken in der Punktwolke eines Tabakblattes zu schließen und dieses zu rekonstruieren. Dabei handelt es sich jedoch nur um einen Nebeneffekt und es kann in dieser Arbeit nicht gezeigt werden, dass dies als generelle Lösung für die Rekonstruktion von 3D-Punktwolken heran genommen werden kann.

1 Einleitung

Um exaktere Prognosen über Ernteerträge zu treffen und die Früherkennung von Krankheiten an Pflanzen zu verbessern, werden 3D-Scanverfahren eingesetzt. Diese Verfahren scannen Pflanzen und nutzen die gewonnenen Daten um aussagekräftige Faktoren und Probleme zu ermitteln, die mit Hilfe der Daten behoben werden können. Um phentypische Eigenschaften von Pflanzen heraus zu finden, können Machine Learning Ansätze verwendet werden. Diese können dazu beitragen, Korrelationen zwischen Input-Daten und Eigenschaften der Pflanze, wie Ertrag, Krankheitsresistenz oder Nährstoffverbrauch, heraus zu finden.

Jedoch kommt es bei den Scanverfahren zu Verdeckung von Pflanzen, so dass nicht die komplette Pflanze also Datenmodel generiert wird. Dies ist ein Informationsverlust, den man verhindern möchte. Um diesem Verlust von Daten entgegen zu wirken, muss die Möglichkeit geprüft werden, ob die Daten vervollständigt und rekonstruiert werden können. In der hier vorliegenden Arbeit soll ein Ansatz überprüft werden, welcher es ermöglichen soll 3D-Modelle, die als Punktwolken vorhanden sind, zu rekonstruieren. Ein Ansatz der dafür herangezogen werden kann, ist Deep Learning. Deep Lerarning kann dabei helfen den latenten Raum der 3D-Modelle zu erlernen und eine Rekonstruktion von nicht vollständigen Datensätzen zu erhalten.

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text bestimmten Klassen zuordnen. Der Grund für das wachsende Interesse liegt in der niedrigen Fehlerrate bei discriminativen Aufgaben, im Vergleich zu anderen Maschine Learning Ansätzen, wie Desicion Trees oder Markov Chains(Goodfellow, Bengio, Courville, & Bengio, 2016). Die Modelle lernen eine Funktion, welche Input Daten X auf ein Klassen Label Y abbildet. Die Modelle werden dabei von künstlichen Neuralen Netzwerken repräsentiert. Man kann auch sagen, das Model lernt die bedingte Wahrscheinlichkeitsverteilung $P(Y|X)$ (Ng & Jordan, 2002). Generative Modelle haben die Aufgabe die Wahrscheinlichkeitsverteilung von Trainingsdatendaten zu erlernen. Diese generativen Modelle lernen eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann. Somit kann dieses Modell auch für discriminative Aufgaben herangezogen werden. Gleichzeitig können aber neue (x,y) Paare erzeugt werden, was zu dem Ergebnis von neuen Datensätzen führt, welche nicht Teil der Trainingssample sind (Ng & Jordan, 2002). In dieser Arbeit wird speziell auf GAN, aus der Vielzahl von generativen Modellen, eingegangen. Diese wurden von Goodfellow(Goodfellow et al., 2014) entwickelt und ebneten den Weg für Variationen, welche auf der Grundidee von GANs aufbauen. GANs wurden bereits verwendet um 2D-Daten, wie Bilder, zu rekonstruieren, wie beispielsweise im Paper (Dong, Loy, He, & Tang, 2016a) gezeigt wurde, in welchem auf super-hochauflösenden Bildern Artefakte entfernt und das Bild so wieder zu seinem Urzustand zurück geführt wurde.

1.1 Zielsetzung

In dieser Arbeit wird versucht die derzeitig bestehende Forschungslücke anzugehen, welche in der Verarbeitung von Punktwolken durch künstliche Neuronale Netzwerke besteht. Der Bereich, zu dem noch keine Lösung gefunden werden konnte, ist die vollständige Rekonstruktion von Punktwolken. Bei der Rekonstruktion wird eine Punktwolke, welche von ihrem Urzustand abweicht, transformiert und wieder in diesen gebracht. Dieser Zustand kann durch Artefakte entstehen, welche beim Erfassen der Punktwolke im Scanverfahren den Blickwinkel verdecken. Ein Vorverarbeitungschnritt ist es zu prüfen, ob es möglich ist den latenten Raum des Urzustandes dieser Punktwolke zu erlernen. Die Forschungsfragen der hier vorliegenden Arbeit lassen sich definieren als:

Ziel 1 Können durch GANs 3D-Punktwolken der latente Raum von Tabakblättern erlernt werden um neue Datensätze zu generieren?

Ziel2 Können durch GANs 3D-Punktwolken von Tabakblättern welche von Tabakblättern von ihren Urzustand abgebracht wurden rekonstruiert werden?

1.2 Überlick

Diese Arbeit ist folgender Maßen strukturiert. In Kapitel 2 "Grundlagen und ähnliche Arbeiten"werden theoretische Grundlagen welche für diese Arbeit benötigt werden. genauer beleuchtet. Außerdem sollen vorrangegeganngen Arbeiten welche einfluß auf diese Ausüben vorgestellt werden um zu veranschaulichen aus welchen Gründen diese Funktioneren kann.

Kapitel 1 - Grundlage und ähnliche Arbeiten Zunächst wird auf die fundamentalen Grundlagen zu dieser Arbeit eingegangen, wie beispielsweise auf künstliche Neuronale Netzwerken, welche für das Verständnis von Generativ Adverserial Networks obligatorisch sind. Daraufhin werden GANs genauer spezifiziert, auf welchen Theorien sie aufbauen und aus welchen Modulen sie zusammen gesetzt sind. Auch werden Punktwolken genauer spezifiziert und aufgezeigt, auf welche Probleme Maschine Learning Ansätze bei diesen stoßen. Zuletzt wird in diesem Kapitel auf die bisherigen Ergebnisse bei der Rekonstruktion von Daten eingegangen.

Kapitel 2 - Methoden In diesen Kapitel werden die einzelnen Versuchsaufbauten definiert, welche versuchen die Ziele der Datenverteilung von Tabakblättern zu erlernen, sowie die Rekonstruktion von Tabakblätter zu prüfen.

Kapitel 3 - Evaluation und Ergebnisse Es werden die Ergebnisse aus den Versuchsaufbauten 1 und 2 evaluiert und auf ihre Verwendbarkeit in der Praxis hingewiesen.

Kapitel 4 - Zusammenfassung und Diskussion Gibt eine Konklusion über die Arbeit und diskutiert die Ergebnisse kritisch. Des weiteren wird ein Ausblick drauf gegeben, inwiefern das Ergebnis für zukünftige Arbeiten von Relevanz ist.

2 Grundlagen und ähnliche Arbeiten

Im folgenden Kapitel wird auf theoretische Grundlagen eingegangen, welche zum Verständnis für Arbeit benötigt werden. Zunächst werden generative Modelle im Allgemeinen vorgestellt, welche den Grundgedanken der Datengeneration für GANs aufzeigen und mit deren Hilfe es möglich gemacht werden soll unkenntliche beziehungsweise beschädigte Modelle von Objekten wieder herzustellen. Diese Theorie baut zunächst auf Datenstruktur von künstlichen Neuronalen Netzwerken auf, welche durch verschiedene Modelle dargestellt werden können. Im Kapitel Conditional-GAN und 3D-GAN werden die theoretischen Grundbausteine der vorherigen Themen vertieft und bilden die Grundlage für das in dieser Arbeit vorgestellte Verfahren zur 3D-Datenrekonstruktion von 3D-Modellen aus Tabakblättern.

2.1 Künstliche Neuronale Netzwerke

Künstliche Neuronale Netzwerke(KNN) sind Datenstrukturen, welche von biologischen neuronalen Netzen wie sie bei Lebewesen vorkommen, inspiriert sind. KNN haben das Ziel ein Funktion f* zu approximieren. Dabei werden Parameter Θ eines Modelsangepasst, um die Abbildung von y = f(x;Θ) zu approximieren. Die Modelle werden auch als feedforward Neuronale Netzwerke betitelt weil der Informationsfluss des Models von Input zu Output fließt und keine Rekursion von Output zu Input statt findet. Diese Approximation wird durch Machine Learning Ansätzen erlernt, man spricht auch von einem Optimierungsproblem. KNN können aus mehreren Schichten, sogenannten Hidden Layer n, bestehen, welche als $f^{(n)}$ dargestellt werden wobei, n >=1 sein muss. Ein 2-Layer KNN ist dann definiert durch f(x)=$f^{(2)}(f^{(1)}(x))$. Man spricht auch von Fully-Connected-Layer. Es gibt einen Input Layer, welcher den Input in das Netzwerk aufnimmt(Goodfellow et al., 2016). Wird ein KNN mit mehr als nur 1 Layer mit Maschine Learning trainiert spricht man auch von Deep Learning. Im vorherigen Beispiel ist dies $f^{(1)}$ und der letzte Layer des Netzwerkes wird Output Layer genannt. Im vorherigen Beispiel ist das $f^{(2)}$. In Abb. 1 sind ein KNN exemplarisch dargestellt. Es soll die Konnektivität der einzelnen Layer veranschaulichen und den Informationsfluss von Input zu Output.

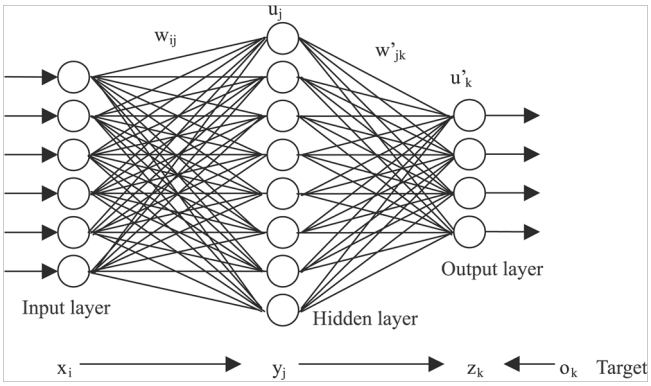

Abb. 1: Künstliches Neuronales Netzwerk(Haque et al., 2018)

Jeder Layer besteht aus künstliche Neuronen. Diese haben ihre Namensgebung von aus der Natur stammende Neuron in Gehirnen von Lebewesen. Neuronen sind die Bausteine, aus denen die Gehirne von Lebewesen wie Fischen, Vögeln und Säugetiere zusammen gesetzt sind. Neuronen, oder auch Nervenzellen haben einen Zellkern der Zentrum der Zelle ist. Um sie herum sind Dendriten welche die Verbindung zu anderen Neuronen her stellt. Neuronen sind untereinander mit dem Axonen verbunden, welche an den enden Synapsen haben die Grenze von Axon zur Nervenzelle einen Spalt bilden. Dieser Spalt kann überwunden werden indem von der Synapse Botenstoffe abgesendet werden, die sich dann an den Rezeptor der gegenüberliegenden Synapse anhaften. Diese Übertragung findet statt wenn an der Synapse ein bestimmter Schwellenwert überschritten wurde von elektrischen Reizen, welche die Zelle abfeuert lässt. Künstliche Neuronen haben diesen Schwellenwert durch sogenannte Gewichte w_{ij} diese sind auf den Verbindungen zwischen den Neuronen in den unterschiedlichen Layern im KNN (vgl. 2). Dabei steht i für die Position im Layer und j in welchen Layer sich das Neuron befindet.

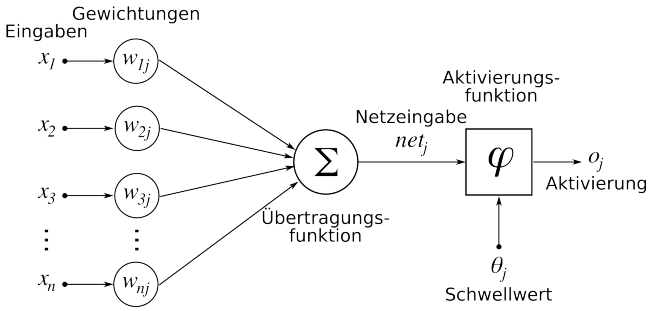

Abb. 2: künstliches Neuron

Jeder Layer eines KNN besteht aus mehreren Neuronen. Ein Neuron θ kann mehrere Inputs X_n erhalten und produziert einen Output ωDie Berechnungen, welche von den Neuron durchgeführt werden sind zunächst jeden Input X_n mit einen Gewicht w_{ij} zu multiplizieren. Anschließend wird die Summe von x*w gebildet. Das Ergebnis wird dann in eine Aktivierungsfunktion λ gegeben. Ein Neuron ist definiert durch:

$$y_k = \lambda(\sum_{j=0}^{m}(w_{kj} + x_j) + b_k)$$

Die Aufgabe der Aktivierungsfunktion λ ist es eine nicht lineare Transformation des Inputs zu erzeugen. Damit kann das KNN nicht lineare Funktionen abbilden und somit komplexere Aufgaben lösen. Es gibt unterschiedliche Aktivierungsfunktionen im folgenden werden einige aufgezählt:

Sigmoid-Funktion $\sigma(x) = \frac{1}{1+exp(-x)}$

Softmax-Funktion $\zeta(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$

Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$

Leacky Rectified Linear Unit(Leaky Relu): $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{sonst} \end{cases}$

Der Funktionsplot von Sigmoid Funktion kann Abb.3 entnommen werden. Dieser veranschaulichen den Wertebereich, welcher von den Aktivierungsfunktion angenommen werden kann und ihren Verlauf.

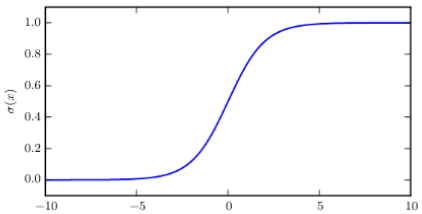

Abb. 3: Sigmoid Funktion

2.1.1 Ziel-Funktion

Die Ziel-Funktion oder auch Loss-Funktion genannt J(θ), muss differenzierbar sein wobei θ das KNN ist und J für die Zielfunktion steht. Das heißt die Funktion f: D $\rightarrow$ R ist differenzierbar an der Stelle x_0. Wenn nun f': x $\rightarrow$ f(x) an jedem Punkt x^n ableitbar ist, ist f differenzierbar. Die Aufgabe der Ziel-Funktion ist es zu messen wie gut unsere Model θ f*(x) approximiert. Es wird der Begriff Kosten verwendet, wie viel Kosten erzeugt das Model beim Lösen der zugewiesenen Aufgabe. Die Wahl, welche Ziel-Funktion gewählt wird ergibt sich aus der Aufgabe des KNN. Diese kann beispielsweise für Klassifikations oder Regressions Aufgaben hergenommen werden. Bei Regression soll eine kontinuierliche Variable von θ als Output generiert werden, wohingegen bei Klassifikationsproblemen der Output Klassen-Labels darstellt(Goodfellow et al., 2016). Es gibt verschiedene Ziel Funktionen im folgenden wird die Cross Entropy Loss Function vorgestellt (Golik, Doetsch, & Ney, 2013). Diese kann verwendet werden um beispielsweiße Klassikationsprobleme zu lösen. Diese ist definiert als:

$$\hat{q}(c \mid x) = \arg\min_{q(c|x)} \{-\sum_{n} \log q(c_{\mathrm{n}} \mid x_{\mathrm{n}})\}$$

Wobei x_n:n=1,...,N die Trainingsdaten sind und c_n:n=1,...,N die möglichen Klassen. Der Output ist die Wahrscheinlichkeit zwischen 0 und 1, ob $x_i \in$ der bestimmten Klasse enthalten ist. Auch kann sie hergenommen um zu messen wie hoch die Differenz zwischen zwei Wahrscheinlichkeitsverteilungen ist.

2.1.2 Backpropagation Algorithmus

Um nun KNNs zu trainieren und den gewünschten Output y zu generieren wird der Backpropagation Algorithmus benutzt. Dieser zählt zu den Optimierungs Algorithmen für KNN und arbeitet schneller und effizienter auf Neuronalen Netzwerken als andere Optimierungsalgorithmen vor ihm. Das mathematische zugrundeliegende Konzept ist ein Optimierungsproblem der die partitielle Ableitung von $\frac{\partial J}{\partial w}$, wobei J die Zielfunktion und w die Gewichte im zu optimierenden Neuronalen Netzwerk, sind. Für eine Funktion f(x) = y ist die Ableitung definiert als f'(x) oder $\frac{dy}{dx}$ und gibt die Steigung der Funktion an Punkt x an. Durch die Steigung am Punkt x ist man nun in der Lage eine Änderung von der Ableitung x dahin gehend zu optimieren (Goodfellow et al., 2016). Dieses Verfahren hilft dabei das KNN dahingehen zu optimieren den gewünschten Output y zu erlangen. Dieses Ziel wird durch die Ziel-Funktion einen KNN beschrieben(vgl. 2.1.1). Das Verfahren wird in Abb. 4 veranschaulicht.

Wenn nun f'(x) = 0 gibt es keinerlei Information über die Steigung. Dies ist aber kein Indiz dafür,dass f ein Optimum erreicht hat. Es könnte, wie in Abbildung 5 unten dargestellt ein lokales Minimum sein, was bedeutet, dass an diesen Punkt ein Minimum erreicht ist aber im Funktionsverlauf ein noch niedrigeres Minimum vorhanden ist. Oder einen Sattelpunkt welche einen Übergang zu einen

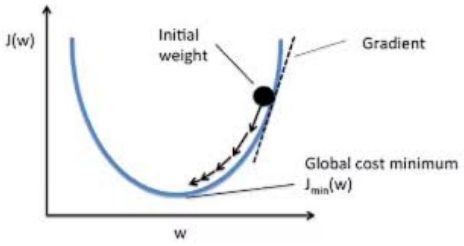

Abb. 4: Optimierung einer Zielfunktion

anstieg der Funktion bildet. Ist nun die die Funktion f definiert als f: $\mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{R}$ hat sie als Input mehrere Variablen. Die partielle Ableitung $\frac{\partial f(x)}{\partial x_{\mathrm{i}}}$ zeigt an wie sehr sich $f(x)$ ändert wenn $\mathrm{x_i}$ geändert wird. Der Gradient $\nabla_{\mathrm{x}} f(x)$ ist ein Vektor, welcher alle partiellen Ableitungen von f enthält. Nun kann f optimieren werden, in dem, in die Richtung die Gewichte $\mathrm{w_{ij}}$ von θ dahin gehen verändert werden in dem das Gradientenverfahren absteigend zu $f'(\mathrm{x}) = 0$ optimiert wird. Dieses Verfahren wird Gradient Descent genannt (Goodfellow et al., 2016).

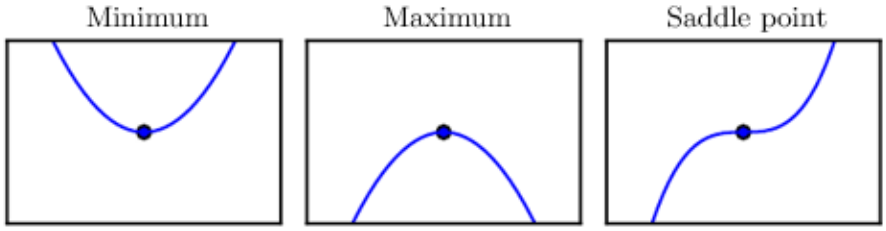

Abb. 5: Minimum,Maximum und Sattelpunkt einer Zielfunktion (Morabbi et al., 2018)

2.1.3 Momentum

Momentum hat das Ziel das Gradientenverfahren des Backpropagation Algorithmus 2.1.2 zu beschleunigen, um ein effizienter Ergebnis herbei zu führen und ein schnelleres Lernen erreichen. Definiert ist dies durch:

$$v_{\mathrm{t}+1} = \mu v_{\mathrm{t}} - \epsilon \nabla \mathrm{f}(\theta_{\mathrm{t}}) \theta_{\mathrm{t}+1} = \theta_{\mathrm{t}} + v_{\mathrm{t}+1}$$

wobei $\epsilon > 0$ die Lernrate ist, $\mu \in [0,1]$ das Momentum und $\nabla \mathrm{f}(\theta_{\mathrm{t}})$ der Gradient

von θ_t ist. Je größter das Momentum, desto schneller bewegt sich der Gradient abwärts. Da dieser am Anfang einer Lernphase der üblicherweise hoch ist empfiehlt sich zunächst mit einen niedrigen Momentum zu Arbeiten da sonnst die Gefahr besteht über das globale Optimum hinaus zuschießen. Wenn nun das Training stagniert, was aus Gründe der Aufbau der Zielfunktion zurückzuführen ist das zur nähe des globalen Optimums flache Täler entstehen welche das Trainings verlangsamen und es zu keiner Verbesserung kommt, kann man durch Momentum erzwingen größere Gradienten Sprünge einzugehen. Und sich somit schneller zu einen globalen Optimum zu bewegen oder aber aus einen lokalen Optimum hinaus Richtung eines globalen Optimum zu bewegen kann Momentum benutzt werden(Sutskever, Martens, Dahl, & Hinton, 2013).

2.1.4 Regularization

Ein Problem welches alle Machine Learning Anwendungen teilen ist es wenn der Trainingsalgorithmus zunächst eine gutes Trainingsergebnis erzielt, aber dann auf dem Testdatensatz ein schlechtes Ergebnis erlangt, was Overfitting genannt wird. Es gibt einige Möglichkeiten die eingesetzt werden können, welche dann aber ebenfalls auch das Trainingsergebnis verschlechtern. Man spricht davon das Model zu Regularizieren.

Data Argumentation

Je mehr Daten beim Training verwendet werden desto mehr kann generalisiert werden. Data Argumentation heißt, dass man mehr Datensätze erstellt. Dies kann durch eine Veränderung der Trainingsdatensätze im Allgemein geschehen (Goodfellow et al., 2016), wenn man beispielsweise 3D-Punktwolken rotiert, um diese mehrfach zu nutzen. Eine weitere Möglichkeit ist es, bestimmte Datenpunkte in einen Datensatz ändern indem man ein Objekt, welches auf einem Bild dargestellt wird, verkleinert oder vergrößert(Goodfellow et al., 2016).

Dropout

Bei Dropout wird während des Lernprozess ein gewisser Prozentsatz der Neuronen in jedem Layer nicht verwendet. Dies zwingt das Netzwerk, welches sonnst die Abhängigkeiten zwischen den Neuronen lernt, eine generalisiertere Lösung zu finden(Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014).

L2-Regularization

Dabei wird das Gewicht des Neurons während des Trainings näher zu seinem Ursprung, dass heißt dem Wert seiner Initialisierung, gedrängt. Dies passiert indem ein Regularisierungswert $\Omega = \frac{1}{2}||w||\frac{1}{2}$ an die Zielfunktion gehängt wird. Dieses sorgt dafür, dass während dem Training der Gewichts

Vektor welche durch das Gradienten Verfahren ermittelt wird, schrumpft, was für eine höhere Varianz während des Trainings sorgt, welches wiederum das KNN dazu zwingt mehr zu generalisieren(Goodfellow et al., 2016).

2.1.5 Batch Normalisation

Wie in Kapitel 2.1.2 gezeigt hat das stochastischen Gradienten Verfahren Vorteile gegenüber dem normalen Gradienten Ermittlung Verfahren beim Training von KNN. Dadurch das der Input in jeden Layer abhängig von den vorherigen Layern ist können Änderungen von Werten in frühen Layern des KNN große Auswirkungen in tieferen Layern im Netzwerk haben. Dadurch resultiert dass in Trainingsabläufen die Verteilung der Gewichte in den jeweiligen Layern verlangsamt wird. Batchnormalization soll die Werteänderung von Gewichten verringen. Dieses Problem wird auch Covariance Shift genannt. Um dies zu verhindert zeigten Sergey Ioffe und Christian Szegedy (Ioffe & Szegedy, 2015) eine Methode, welche Batch Normalisation genannt wird. Je mehr Layer das Netzwerk hat desto stärker ist der Covariance Shift. Batch Normalisation besteht aus zwei Algorithmen. Algortithmus 1 verändert den eigentlichen Input von Layer n zu einen normalisierten Input y und Algorithmus 2 verändert das eigentliche Training eines Batch normalisierten Netzwerkes(Ioffe & Szegedy, 2015).

Algorithm 1: Batch Normalisierung angewand auf x über Input bei Mini-Batch

1 Input: Werte von x über einen Mini-Batch $B = \{x_1, ..., x_n\}$
2 $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x^i$
3 $\sigma_B \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu_B)^2$
4 $\hat{x_i}_B \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}}$
5 $y_i \leftarrow \gamma\hat{x_i} + \beta \equiv BN_{\gamma;\beta}(x_i)$
6 Output: $\{y_i = BN_{\gamma;\beta}(x_i)\}$

In Schritt 2 des Algorithmus 1 wird der Erwartungswert für alle Inputs von Mini-Batch B berechnet und in Schritt 3 die Varianz. In Schritt 3 wird nun der der normalisierte x_i berechnet welche dann mit β und γ multipliziert werden. Diese Werte sind neue Gewichte im Neurnonalen Netzwerk welche während des Trainingsprozesses angepasst werden. ϵ in der Gleichung in Zeile 4 ist nur dafür da damit nicht durch 0 geteilt werden kann. In Zeile 8 - 11 werden die Inferenz Schritte beschrieben, bei welchen der Minibatch des Trainings ersetzt wird (Ioffe & Szegedy, 2015). In Abb. 6 ist veranschauchlicht wie die Batch-Normalization-Layer in das KNN eingebaut werden.

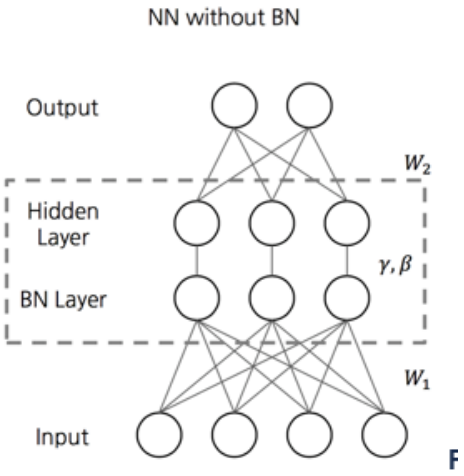

Abb. 6: Neuronales Netzwerk mit Batch-Normalization-Layer(*batchnorm Chun*, n.d.)

2.2 Datenformat Punktwolken

Punktwolken sind eine Menge von N Punkten, welche im Vektorraum dargestellt werden können. Jeder Punkt n ∈ N wird durch seine (x,y,z) Koordinaten im Euklidischen Raum dargestellt. Punkte können zusätzliche Features gegeben werden, wie Farbe oder Material. Es gibt unterschiedliche Dateiformate, welche für die Abspeicherung von Punktwolken herangezogen werden können bespiele dafür sind PLY, STL oder OBJ. Das Polygon File Format(PLY) speichert die einzelnen Koordinaten in einer Liste, welche Vertex List genannt in. In Abb. 7 kann eine Beispieldatei entnommen werden in der dieser Aufbau dargestellt ist.

```
ply
format binary_little_endian 1.0
element vertex 2800
property float x
property float y
property float z
end_header
0 0 0
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
1 0 0
1 0 1
1 1 1
1 1 0
1 0 0
```

Abb. 7: Polygon File Format

Punktwolken können als Menge betrachtet werden. Die jeweiligen Punkte sind geordnet in dieser Listen gespeichert jedoch spielt es keine Rolle für den Punktwolkencompiler bei der Visualisierung der Liste an welcher Listenposition ein jeweiliger Punkt geführt wird, die Liste wird jedes mal gleich angezeigt, egal welche Permutation der einzelnen Punkte in der Liste durchgeführt wird. In Abb. 8 kann eine visualisierte Punktwolke eines Tabakblattes entnommen werden.

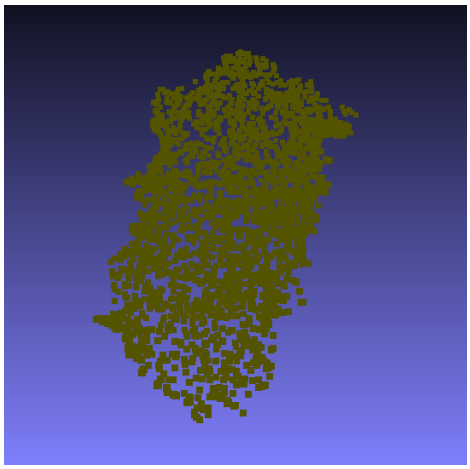

Abb. 8: Visualisierte Punktwolke eines Tabakblattes

2.2.1 Datenaufnahme von Tabakpflanzen

Die Punktwolken können Beispielsweise von den TERRA-REF Feld Scanner von der University von Arizona Maricopa Agricultural Center and USD Arid Land Researh Station in Maricopa aufgenommen werden. In Abb. 9 ist Recht eine Skizze dargestellt. In Abb. 9 Links ist der Scankopf des TERRA-REF dar gestellt.

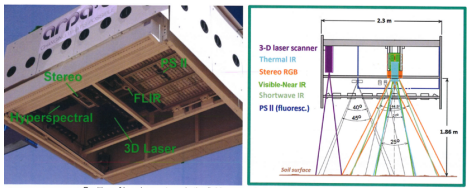

Abb. 9: Scankopf und Scanaufbau für die 3D-Punktwolken Gewinnung

Der 3D Laser Scanner erzeugt 3D Punktwolken. Dabei werden die Objekte durch den Scanner erfasst und eine 3D Repräsentation, welche durch Punkte in einen 3 Dimensionalen Koordinaten System erfasst werden können, dargestellt. Dabei wird ein Laser über das zu scannende Objekt gefahren durch reflektion des Laserstrahls auf der Oberfläche des Objektes können x,y,z Koordinaten des jeweiligen Punktes auf den Objekt bestimmt werden. Da nun beim Scannen eines Objektes durch andere Objekte die Oberfläche verdeckt sein können. Wie beispielsweise beim Scannen von Tabakpflanzen, Blätter den Scankopf es nicht Möglich macht Objekte unterhalb dieses zu erreichen. Können Scans unvollständig sein. Dieses Problem für zu den Ziel dieser Arbeit, eine Möglichkeit zu Prüfen diese unvollständigen Punktwolken zu vervollständigen. Genaueres zum diesen Thema in Kapitel 2.8.

2.2.2 Die Schwierigkeit bei mit 3D-Data bei Machine Learning Ansätzen

Vergleicht man 3D-Data auf ihre Dimensionalität mit anderen Datenformaten wie Bild,- Audio-, und Textdateien steigt der Informationsgehalt und dadurch auch die Komplexität für das Anwenden von Maschine Learning Algorithmen enorm. Besonders bei Nicht-Euklidischen 3D-Daten wie Punktwolken, welchen keine Struktur zu Grunde liegt ist dieses gegeben (Ahmed et al., 2018).

Wie im Kapitel 2.2 gezeigt, sind Punktwolken als Menge gespeichert, in der keine Relation untereinander besteht. Was bedeutet, dass es für den Punktwolkencompiler nicht von Relevanz ist auf welchem Platz die einzelnen Punkte abgespeichert werden. Die Punktwolke wird immer gleich angezeigt, egal in welcher Permutation der einzelnen Punkte abgespeichert abgespeichert werden. Vergleicht man nun ein Bild mit 512 Pixeln und 3 RGB-Farbkanälen ist einer Dimension von 391680 erreicht. Vergleicht man dies mit einer Punktwolke in einen 125 cm ^{3}großen Bereich. Da die einzelen Koordianten eines Punktes als Rationale Zahlen dargestellt werden und rationale Zahlen abzählbar unendlich sind ist der Suchraum unendlich groß. Dies führt zu einen erheblichen mehr Aufwand für Machine Learning Ansätzen wie, Deep Learning für Punktwolken.

Ein weitere Schwierigkeit, die für Deep Learning mit Punktwolken besteht ist, dass die aus Kapitel Convolutional Neural Network beschrieben Convolutionel-Layer einen großen Beitrag bei dem Fortschritt von Deep Learning gebracht hat. Da sie helfen die Strukturen von strukturierten Daten zu lernen und den latenten Raum zu entdecken. Da jedoch Pointclouds unstrukturiert sind hilft es nicht unbedingt diese Tools auch bei Punktwolken einzusetzen(Achlioptas et al., 2018).

2.3 Convolution Neural Networks

Convolution Neural Netzworks(CNN) sind eine besondere Art von künstlichen neuronalen Netzwerken, sie sind dafür konzipiert auf Datensätzen zu arbeiten welche in eine Matrix Form gebracht worden sind. Der Input eines CNN können beispielsweise Bilder sein welche durch die Matrix $A = \begin{bmatrix} a_1 & a_1 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ \vdots & n_n & \ddots & \vdots \\ x_1 & x_2 & x_3 & x_n \end{bmatrix}$ dargestellt werden. Jedes Element x_{ij} stellt einen Pixel eines Bildes da, wobei $x_n \in [0{,}255]$. Die Matrix $A^{w \cdot b \cdot c}$ stellt $w \cdot b \cdot c = N$ dimensionale Matrix da. Wobei w die Länge und b Breite des Bildes entspricht. c sind die Farbspektren eines Bildes und sind in einen RGB-Farbraum 3 beziehungsweiße in einen schwarzweiß Bild 1. Nachdem der Input eines CNN definiert ist kommt nun der Aufbau. CNN setzen sich aus mehrere Schichten von Convolution Layern zusammen. Ein Netzwerk kann mehrere N-Layer haben. Wobei jeder Layer aus mehreren Con-

volution oder auch Kernels genannt, zusammengesetzt ist. Ein Aufbau kann aus Abbildung 10 entnommen werden(Goodfellow et al., 2016).

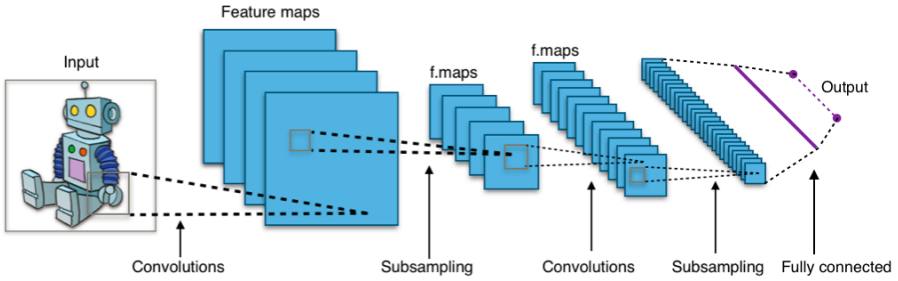

Abb. 10: Convolutional Neural Network

Die Kernels, also die einzelnen Filter, von denen jeder N-Layer k besitzt sind $K^{n \cdot n}$ Matrizen jedes k_{ij} in einem Filter einspricht einen, aus den üblichen Neuronalen Netzwerk Architektur bekannten Gewichte. Diese Gewichte werden dann durch den Backprobagation-Algorithmus in der Trainingsphase des Netzwerkes angepasst um den Verlust der Ziel-Funktion durch bestimmten des Gradianten zu minimieren. Das in Abb. 10 dargestellte Subsampling ist der Output aus den Convolutional Layern(Goodfellow et al., 2016).

Da Input und Kernel unterschiedliche Größen haben und man den gesamten Input mit den Kernel abdecken möchte, bewegt sich der Filter um s Position auf den Input und führt erneut einen Berechnungschritt durch. Dieser Vorgang wird Stride genannt. An jeder Position wird das Produkt von jeden x_{ij} des Input und k_{ij} des Kernel durchgeführt. Anschließend werden alle Produkte aufsummiert. In Abbildung 11 ist dieser Vorgang verdeutlicht. Zusätzlich gibt es die Möglichkeit für das sogenannte Zero Padding P. Dabei werden mehrere 0 um die Input Feature Map, am Anfang und Ende der Axen anfügt. Dies ist notwendig wenn Kernel und Input Größe nicht kompatible zueinander sind. Die Anzahl der möglichen Positionen ergeben sich aus der Kernel Größe und den Input des jeweiligen Kernel sowie des Strides. Die Output Größ W kann berechnet werden durch $W = (W\text{-}F+2P)/s+1$ berechnet werden. Wobei F für die Größe des Kernel steht (Dumoulin & Visin, 2016).

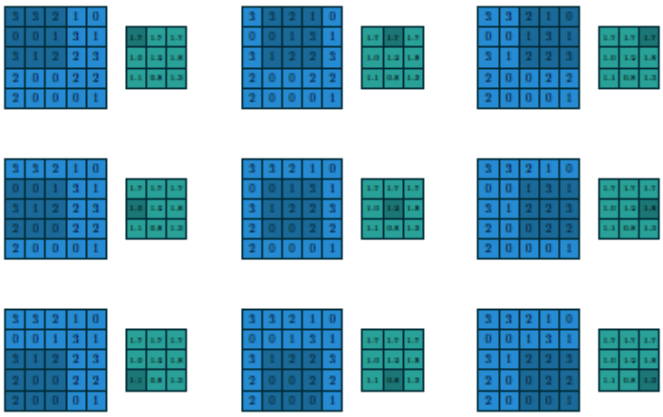

Abb. 11: Convolution Beispiel(Dumoulin & Visin, 2016)

Um besser zu verstehen welche Auswirkungen die Anzahl der Kernels in Layer n auf die Größe des Outputs von n und die Anzahl der Kernels in Layer n+1 für den nächsten Layer haben, wird ein Beispiel aufgezeigt. Der erste Layer hat 20 Kernels mit der Größe 7x7 und Stride 1. Der Input A für einen Kernel K ist ein 28x28 Matrix. Der Output aus diesen Filter sind 20 22x22 Feature Maps. Wäre der Input ein 28x28x3 Bild mit 3 RGB Channels sein, der Output 60 22x22 Feature Maps. Allgemein kann Convolution Layer als Supsampling gesehen werden und Stride gibt an wieviele Dimensionen bei diesen Prozess pro Convolution Layer entfernt werden soll. Der letzte Layer ist ein Fully-Connected Layer welcher den typischen Anforderungen von ANN entspricht (Dumoulin & Visin, 2016).

Transposed Convolution, auch genannt Fractionally Strided Convolution oder Deconvolution, ist eine Umkehrfunktion von der üblichen Convolution. Es verwendet die gleichen Variablen wie Convolution. Dabei wird ein Kernel K mit der Größe N x N definiert der Input I mit der Größe N x N und Stride s = 1. Deconvolution kann wie Convolution angesehen werden mit Zero Padding auf dem Input. Das in Abbildung 12 gezeigte Beispiel zeigt einen deconvolution Vorgang mit eine 3x3 Kernel über einen 4x4 Input. Dies ist gleich mit einen Convolution Schritt mit einen 3x3 kernel auf einen 2x2 Input und einer 2x2 Zero Padding Grenze. Convolution ist Supsampling und mit Deconvolution wird Upsampling betrieben. Durch diesen Schritt kommt es zu einer Dimensionserhöhung des Inputs. Die Gewichte der Kernels bestimmen wie der Input transfomiert wird. Durch mehrere Schichten von Deconvolution Layer kann von einer Input Größe NxN auf eine Output Größe KxK, wobei K > N mit Abhängigkeit von Kernel und Stride abgebildet werden(Dumoulin & Visin, 2016).

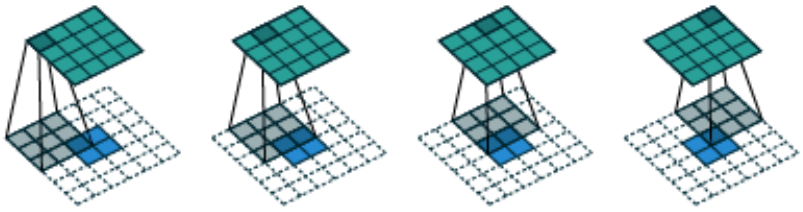

Abb. 12: Deconvolution Beispiel (Dumoulin & Visin, 2016)

2.4 Autoencoder

Autoencoder gehören zu den generativen Modellen im Bereich des Machine Learnings. Generative Modelle haben das Ziel eine Wahrscheinlichkeitsverteilungen zu erlernen. Anschließend kann diese als ein Modell genutzt werden um Samples aus dieser zu erzeugen. Die Modelle können dabei beispielsweise auf ANN oder Markov Chains trainiert werden(Goodfellow et al., 2016). Im folgenden liegt der Fokus auf ANNs. Allgemein gehalten können jegliche Typen von Daten wie Text, Bild oder Audiodateien für generative Modelle herangezogen werden. Es gibt unterschiedliche Typen von generativen Modellen, welche sich vom Aufbau des Neuronalen Netzwerk und der Zielfunktion unterscheiden. Beispiele dafür sind Boltzmann Maschine, Autoencoder oder Deep Belief Networks(Goodfellow et al., 2016).

Autoencoder sind eine andere Art von Model aus den Bereich der generativen Modelle. Ihre Aufgabe besteht darin einen Input zu komprimieren und aus der komprimierten Information den Input wieder herzustellen. Die Technik auf welche Autoencoder zurückgreifen, nennt sich Dimensionreduktion. Dabei wird die Dimension der Daten so reduziert um Informationen bei zu behalten, welche als relevant gelten. Diese Technik findet auch in anderen Machine Learning Anwendung, wie beispielsweise der Principale component anaysis(PCA) Anwendung(Hinton & Salakhutdinov, 2006). Ein Autoencoder besteht aus 2 Bestandteilen. Erstens einen Encoder e parametrisiert mit ϕ welcher einen Input x $\in$ von x^{i} wo x ein Vektor der Länge i ist und damit die Input Dimension bestimmt. Dieser wird durch den Encoder auf einen Vektor z^{k} abgebildet wobei k¡i ist. Zweitens der Decoder d parametrisiert durch θ, bekommt als Input z^{k} und bildet z auf x^{l} ab wobei l = i. Und somit die gleiche Dimension wie der Input. Aufgabe ist es nun, dass der Encoder den Input z so gut komprimiert, dass der Encoder es schafft das x $\approx$ z. Eine grafische Darstellung kann aus Abb. 13 entnommen werden(Goodfellow et al., 2016).

Die Parameter ϕ und θ werden durch den in Kapitel 2.1.2 vorgestellten Algorithmus trainiert und erlernt so können beispielsweise durch Fully-Connected-

Layer, Convolutional-Layer oder Deconvolutional-Layer modelliert werden. Eine Metrik um zu messen wie das Model seine Aufgabe erfüllt, könnte beispielsweiße die Cross-Entropy-Funktionen sein welche schon in Kapitel 2.1.1 vorgestellt worden ist. Weitere spezifische Zielfunktionen für Autoencoder welche mit 3D Punktwolken arbeiten und für die folgende Arbeit von belangen sind, werden nun vorgestellt.

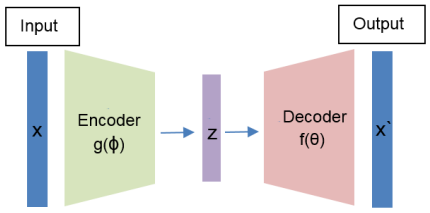

Abb. 13: Autoencoder

Bei Punktwolken als Datentyp erzielt die Cross-Entropy-Funktion keine guten Ergebnisse da diese invariant zu ihrer Permutation sind. Da dieser Arbeit der Input 3d Pointcloud sind und diese Sets invariant zu ihren Permutationen sind. Ändert sich Anordnung meiner einzelnen Punkte in mein Set bleibt das dargestellte Ergebnis unverändert. Deshalb kann nicht auf übliche Zielfunktionen welche für strukturierte Daten wie Bilder verwendet werden, zurück gegriffen. Die Herausforderung besteht darin zwischen zwei unterschiedliche Sets von Punkten heraus zu finden, wie hoch die Diskrepanz zwischen den beiden Sets ist(Ravanbakhsh, Schneider, & Poczos, 2016).

Eine Möglichkeit, speziell für Autoencoder, welche mit Punktwolken arbeiten, ist die Earth Mover Distance(EMD). Bei dieser sind X_1 und X_2 zwei Punktwolken mit jeweils x_n definierten Punkten(Fan, Su, & Guibas, 2017). Definiert ist sie durch die Funktion:

$$d_{\text{EMD}} = \min_{\theta: X_1 \to X_2} \sum_{x \in X_1} ||x - \theta(x)||_2; wobei\ \theta : X_1 \to X_2\ bijektiv\ ist$$

Grafisch kann man sich die Berechnung wie in Abb. 14 darstellen. Ein Punkt von $x_n \in X_1$ wird den nähsten Punkt $y_n \in X_2$ zugewiesen. Wobei die Distanz durch die Euklidische Distanz der jeweiligen Punkte ermittelt wird.

Eine weitere Möglichkeit ist die Chamfer Distance(CD). Wie auch zuvor sind X_1 und X_2 zwei Punktwolken mit jeweils x_n definierten Punkten(Fan et al., 2017). Definiert ist sie durch die Funktion:

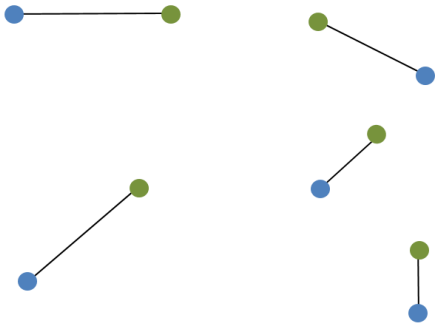

Abb. 14: Earth Mover Distance

$$d_{\mathrm{CD}}(X_1, X_2) = \sum_{x \in X_1} \min_{y \in X_2} ||x - y|| + \sum_{y \in X_2} \min_{x \in X_2} ||x - y||$$

Der Unterschied ergibt sich zwischen den beiden Metriken, dass bei der EMD von der Ausgangswolke die Punkte zu der anderen jeweils optimiert werden. Wohingegen bei der CD die Distanzen von und zu der Ausgangspunktwolke berechnet werden. Dies geht aus den Abb. 15 hinaus in welche die Distanzberechnung der einzelnen Punkte dargestellt wird(Fan et al., 2017).

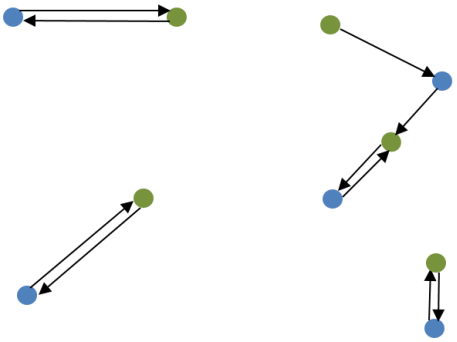

Abb. 15: Chamfer Distance

2.5 Generative Adversarial Network

In den letzten Jahren konnte sich das GAN als best practice Ansatz bei den generativen Modellen herausarbeiten was Performancegründe bei der Trainierbarkeit und Qualität der generierbaren Daten zu Grunde liegt(Goodfellow et al., 2016). Die Modelle arbeiten nach dem Maximum Likelihood Schätzverfahren(ML-Schätzer) in dem die Parameter θ dahingegen angepasst werden, dass die unsere beobachtetet Daten am ehesten passen. Man kann ML-Schätzer als Kulback-Leibler(KL) Divergenz darstellen und das generative Modelle das Ziel haben die KL Divergenz zwischen den Trainingsdaten P_r und den generierten Daten P_g zu minimieren.Diese ist definiert durch:

$$KL(P_r||P_g) = \int_x P_r log \frac{P_r}{P_g} dx$$

Ein GAN besteht besteht aus zwei KNN, dem Discriminator D und dem Generator G. Das Ziel des G ist es, Daten x zu erzeugen, welche nicht von Trainingsdaten y unterschieden werden können. Dabei wird eine vorangegangene Input Noise Variable $p_z(z)$ verwendet, welche eine Abbildung zum Datenraum $G(z;\Phi_g)$ herstellt. Dabei sind Φ_g die Gewichte des neuronalen Netzwerkes von G. Der Discriminator hat die Aufgabe zu unterscheiden, ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von Trainingsdaten y stammt(Goodfellow et al., 2014). Die Zusammensetzung zwischen den beiden Netzwerken kann aus Abbildung 16 entnommen werden.

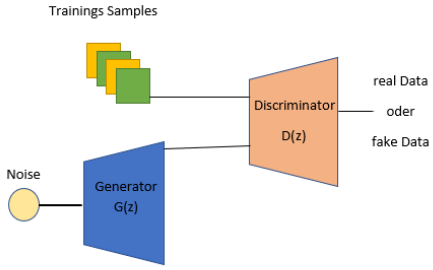

Abb. 16: Generativ Adverserial Network

Der Discriminator ist definiert durch $D(x;\Phi_d)$. Wobei Φ_d die Gewichte des Descriminators sind und D(x) die Wahrscheinlichkeit ist, dass x von den Trainingsdaten stammt und nicht von p_g. Die Wahrscheinlichkeitsverteilung für unsere Trainingsdaten ist p_r. Im Training werden dann Φ_d so angepasst, dass die Wahrscheinlichkeit Trainingsbeispiele richtig zu klassifizieren maximiert wird. Und Φ_g wird dahingehen trainiert die Wahrscheinlichkeit zu minimieren, so dass D

erkennt dass Trainingsdatensatz x von G erzeugt wurde. Mathematisch ausgedrückt durch log(1 - D(G(z))). Die gesamte Loss-Funktion des vanilla GAN ist definiert als

$$min_{\mathrm{G}} max_{\mathrm{D}} V(D,G) = E_{\mathrm{x}\sim \mathrm{p_{data}(x)}}[logD(x)] + E_{\mathrm{z}\sim \mathrm{p_z(z)}}[log(1-D(G(z))]$$

diese beschreibt ein Minmax Spiel zwischen G und D. Welches das globale Optimum erreicht hat wenn $\mathrm{p_g} = \mathrm{p_r}$. Das heißt, wenn die Datenverteilung, welche von G erzeugt wird, gleich der unserer Trainingsdaten ist(Goodfellow et al., 2014). Das Training erfolgt durch den folgenden Algorithmus:

Algorithm 2: Minibatch stochastic gradient descent Training für Generative Adversarial Networks. Die Anzahl der Schritte welche auf den Discriminator angewendet wird ist k

for *Anzahl von Training Iterationen* **do**
  for *k Schritte* **do**
    •Sample minibatch von m noise Samples $\mathrm{z}^{(1)}$,...,$\mathrm{z}^{(\mathrm{m})}$von noise $\mathrm{p_g(z)}$
    • Sample minibatch von m Beispielen $\mathrm{x}^{(1)}$,...,$\mathrm{x}^{(\mathrm{m})}$von Daten Generationsverteilung $\mathrm{p_{data}(x)}$
    • Update den Discriminator zum aufsteigenden stochastischen Gradianten:
    $\nabla_{\Phi_{\mathrm{d}}} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(\mathrm{i})}) + log(1 - D(G(z^{(\mathrm{i})})))]$
  end
  • Sample minibatch von m noise Samples $\mathrm{z}^{(1)}$,...,$\mathrm{z}^{(\mathrm{m})}$ von noise $\mathrm{p_g(z)}$
  • Update den Generator mit den absteigenden stochastischen Gradianten:
  $\nabla_{\Phi_{\mathrm{g}}} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(\mathrm{i})})))$
end

Beim Training wird ein stochastischer Minibatch von mehren Trainingsdaten gleichzeitig erstellt. Dies soll dabei helfen, dass der Generator sich nicht auf bestimmte Gewichte fest fährt und auf Trainingssätze kollabiert. So weisen die erzeugten Daten mehr Variationen auf (Salimans et al., 2016). D wird zunächst in einer inneren Schleife auf n Trainingsätzen trainiert, womit man Overfitting von D vermeiden will, was zur Folge hätte, dass D nur den Trainingsdatensatz kopieren würde. Deshalb wird k mal D optimiert und ein mal G in der äußeren Schleife.

Ein möglicher Aufbau von GAN wird in Abbildung 18 dargestellt. Dies ist das sogenannte Deep Convolution GAN(DC GAN), welches dafür konzipiert wurde auf Bilddaten zu arbeiten. Dabei besteht der Generator aus mehren Schichten von Deconvolution Layern. Welche den Input Noise Variable $\mathrm{p_z(z)}$ auf y abbildet. D besteht aus mehren Schichten von Convolution Layern und bekommt als Input die Trainingsdaten, oder die von G erzeugten Y, und entscheidet über die Klassifikation(Radford, Metz, & Chintala, 2015).

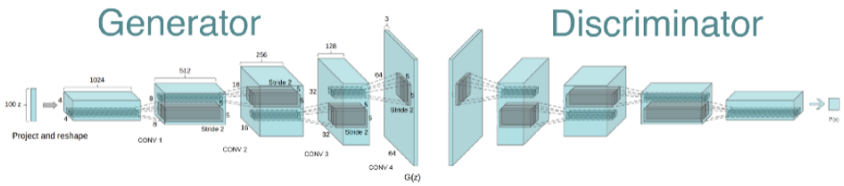

Abb. 17: Deep Convolutional GAN(*dc-gan online book*, n.d.)

Das asymmetrische Verhalten der KV Divergenz zu schlechten Trainingsergebnissen führen. Goodfellow (Goodfellow et al., 2014) zeigte, dass sich die MinMax Loss-Funktion des GAN auch als Jensen-Shannon Divergenz(JS Divergenz) darstellen lässt. Diese ist definiert als

$$D_{\mathrm{JS}}(P_{\mathrm{r}}||P_{\mathrm{g}}]) = \tfrac{1}{2}D_{\mathrm{KL}}(P_{\mathrm{r}}||\tfrac{P_{\mathrm{g}}+P_{\mathrm{r}}}{2}]) + D_{\mathrm{KL}}(P_{\mathrm{q}}||\tfrac{P_{\mathrm{g}}+P_{\mathrm{r}}}{2}])$$

wobei P_r die Wahrscheinlichkeitsverteilung der Trainingsdaten ist und P_g die des Generators. Huzár (Huszár, 2015) zeigte, dass durch das symmetrische Verhalten der JS Divergenz ein potentiell besseres Trainingsergebnis entstehen kann, im Vergleich zu der KL Divergenz. Damit zeigte er weshalb GANs im Vorteil gegenüber anderen generativen Modellen sind. Abbildung 18 veranschaulicht dieses Konzept. Der linke Graph zeigt 2 Normal Verteilungen. In der Mitte wird die KV Divergenz der beiden Normal Verteilungen dargestellt. Rechts ist die JS Divergenz der Beiden dargestellt. Man sieht sehr gut das asymmetrische Verhalten der KV und das symmetrische der JS. Dadurch lassen sich aussagekräftigere Gradianten bestimmen, welche zum Optimieren von D und G benötigt werden(Huszár, 2015).

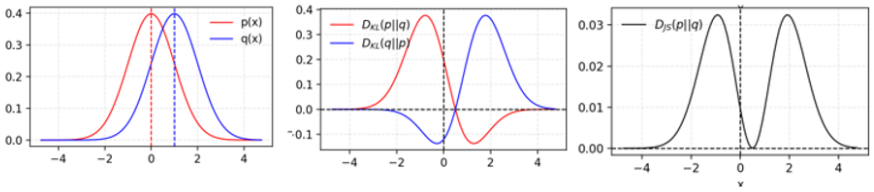

Abb. 18: KL Divergenz und JS Divergenz (*gan lilian weng*, n.d.)

2.5.1 Probleme mit Generative Adverserial Networks

Wie auch anderen generativen Modelle haben auch GANs noch Schwächen bezüglich

der Trainingsabläufe und der Qualität der generierten Daten. Im Folgenden wird auf einige Probleme eingegangen welche im darauffolgenden Kapitel Lösungsansätze aufgezeigt werden.

Equilibrium
D und G betreiben ein MinMax Spiel. Beide versuchen das Nash Equilibrium zu finden. Dies ist der bestmögliche Endpunkt in einen nicht kooperativen Spiel. Wie in dem Fall von GAN wäre das wenn $p_g = p_r$. Es wurde gezeigt, dass das Erreichen dieses Punktes sehr schwierig ist, da durch die Updates der Gewichte mit den Gradianten der Loss-Funktion starke Schwingungen der Funktion enstehen können. Dies kann zur Instabilität für das laufende Training führen (Salimans et al., 2016).

Vanishing gradient
Dies beschreibt das Problem, wenn D perfekt trainiert ist mit D(x) = 1, $\forall \in p_r$ und D(x)=0 $\forall x \in p_g$. Die Loss-Funktion würde in diesem Fall auf 0 fallen und es gäbe keinen Gradianten, für den die Gewichte von G angepasst werden können. Dies verlangsamt den Trainingsprozess bis hin zu einem kompletten Stopp des Trainings. Würde D zu schlechten trainiert mit D(x) = 0, $\forall \in p_r$ und D(x)=1 $\forall x \in p_g$. Bekommt G kein Feedback über seine Leistung bei der Datengeneration hat er keine Möglichkeit p_r zu erlernen (Salimans et al., 2016).

Mode Collapse
Während des Trainings von GAN kann es dazu kommen, dass der Generator möglicherweise auf eine Einstellung seiner Gewichte fixiert wird und es zu einem sogenannten Mode Collapse führt. Was zur Folge hat, dass der Generator sehr ähnliche Samples produziert (Arjovsky, Chintala, & Bottou, 2017).

Keine aussagegräftigen Evaluations Metriken
Die Loss Funktion der GANs liefert keine aussagekräftigen Evaluationsmöglichkeit über den Fortschritt des Trainings. Bei discriminativen Modellen im üblichen Maschine Learning besteht die Möglichkeit Validierungsdatensätze zu verwenden und an diesen die Genauigkeit des Modells zu testen. Diese Möglichkeit besteht bei GANs nicht(Huang et al., 2018).

2.5.2 Lösungsansätze für Generative Adverserial Networks Probleme

Nun werden einige Techniken aufgezeigt, welche die unter Abschnitt Probleme mit GAN genannten Schwierigkeiten angehen und zu einem effizienteren Training führen, damit eine schnellere Konvergenz während des Trainings erreicht wird.

Feature matching

Dies soll die Instabilität von GANS verbessern und gegen das Problem des Vanishing Gradient angehen. G bekommt eine neue Loss-Funktion und ersetzt die des üblichen Vanilla GAN. Diese soll G davon abhalten, sich an D über zu trainieren und sich zu sehr darauf zu fokussieren, D zu täuschen und gleichzeitig auch versuchen die Datenverteilung der Trainingsdaten abzudecken(Salimans et al., 2016).

Minibatch discrimination

Um das Problem des Mode Collapse zu umgehen, so dass es nicht zu einem Festfahren der Gewichten von G kommt, wird beim Trainieren die Nähe von den Trainingsdatenpunkten gemessen. Anschließend wird die Summe über der Differenz aller Trainingspunkte genommen und dem Discriminator als zusätzlicher Input beim Training hinzugegeben (Salimans et al., 2016).

Historical Averaging

Beim Training werden die Gewichte von G und D aufgezeichnet und je Trainingsschritt i verglichen. Anschließend wird an die Lossfunktion je Trainingschritt die Veränderung zu i-1 an die Loss-Funktion addiert. Damit wird eine zu starke Veränderung bei den jeweiligen Trainingsschritten bestraft und soll gegen ein Model Collapse helfen (Salimans et al., 2016).

One-sided Label Smoothing

Die üblichen Label für den Trainingsdurchlauf von 1 und 0 werden durch die Werte 0.9 und 0.1 ersetzt. Dies führt zu besseren Trainingsergebnissen. Es gibt derzeit nur empirische Belege für den Erfolg, jedoch nicht weshalb diese Technik besser funktioniert(Salimans et al., 2016).

Adding Noises

Noise an den Input von D zu hängen kann gegen das Problem des Vanishing gradienten helfen und das Training verbessern(Salimans et al., 2016).

Wasserstein-GAN

Die Wasserstein Metric oder auch Earth Mover Distance(EMD) genannt misst die Minimum Kosten welche entstehen wenn man Daten von der Datenverteilungp p_r zur Datenverteilung p_g überträgt. Es wird oft auch von Masse oder Fläche gesprochen, welche von p_r zu p_g getragen wird. Definiert ist sie durch:

$$W(p_r,p_g)=\inf_{\gamma\sim\prod(P_r,P_g)}E_{(x,y)\sim\gamma}[\|x - y\|]$$

p_r steht für die reale Datenverteilung zu welche uns Daten im Form von Trainingsdaten zur Verfügung stehen und p_g steht für die generierte Datenverteilung welche von einen Model erzeugt wird. Dabei wird nun das Infinum von allen möglichen Transportplänen γ welche in $\prod$ enthalten sind ausgewählt, welche der kostenkünstige Plan ist die Daten von p_g zu p_r zu

übertragen. Vorteile sind unter anderem, dass der Gradient gleichmäßiger ist und das WGAN lernt besser auch wenn der Generator schlechtere Daten erzeugt im Vergleich zum üblichen GAN(Arjovsky et al., 2017). Durch die Kantorovich-Rubinstein Methode kann die Wasserstein-Distanz umgeformt werden in zu:

$$W(p_r,p_\theta)=\sup_{\|f\|_L\leq 1} E_{x\sim P_r}[f(x)] - E_{x\sim P_\theta}[f(x)]$$

Durch diese Umformung ist es nun Möglich das GAN die EMD nutzt um die von G generierten Daten mit den realen Daten zu vergleichen. Dabei übernimmt der Discriminator nun die Aufgabe eines Critic welcher nun nicht mehr 0 oder 1 für Fake oder Real ausgibt sondern einen Score welcher angibt wieviel Masse von der P_θ umverteilt werden muss damit der Generator bessere Ergebnisse liefert. Das Wasserstein GAN liefert bis dahin die erfolgreichste Verbesserung zum üblichen Vanilla-GAN von Goodfellow und erlaubt es gegen die in Kapitel 2.5.1 aufgezeigten Probleme Vanishing gradient und Model Collapse anzugehen(Arjovsky et al., 2017).

2.6 Conditional-GAN

Conditional-GAN(C-GAN) ist eine Modifikation des ursprünglichen GAN von Goodfellow, welches erlaubt bedingte Wahrscheinlichkeiten in Datensätze zu erlernen. Das heißt zusätzliche Informationen in den Lernprozess einzuspeisen um den Output zu modifizieren. Im ursprünglichen GAN gibt es keine Möglichkeit auf den Output des Generators Einfluss zu nehmen. Dabei wird das Modell so verändert das eine zusätzliche Information y als Input in den Discriminator und Generator zugefügt wird. Dabei kann y jegliche Information sein wie Label, Bilddaten oder 3D-Daten. Dementsprechend muss die Zielfunktion dahin gehend angepasst werden bedingte Wahrscheinlichkeiten zu lernen

$$min_{\mathrm{G}} max_{\mathrm{D}} V(D,G) = E_{\mathrm{x}\sim \mathrm{p_{data}(x)}}[logD(x|y)] + E_{\mathrm{z}\sim \mathrm{p_z(z)}}[log(1 - D(G(z|y))]$$

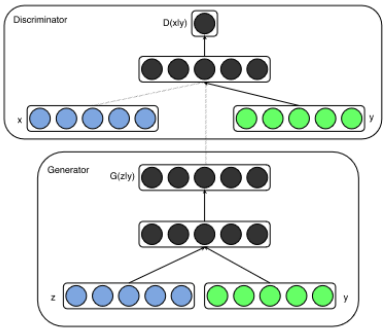

Abb. 19: Conditional Adverserial Network(Mirza & Osindero, 2014)

Im Abb. 19 kann der Informationsfluss und die Konnektivät der einzelnen Module entnommen werden. Die Module Generator und Discriminator bleiben gleich und können von ihrem Aufbau für die jeweiligen Datentyp verändert werden und Beispielsweiße durch Convolutional-Layer, Deconvolutional-Layer oder Fully-Connected-Layer bestehen.(Mirza & Osindero, 2014).

2.7 3D-GAN

Das besondere am 3D Raum im Vergleich zu normalen 2D Bildern ist die Steigerung der Dimension und zu gleich der hohe Informationsgehalt welcher in 3D Objekten steckt. Das Ziel von 3D-GAN ist es die Datenverteilung der zugrunde liegenden 3D-Modellen zu erlernen. Dabei wird der latente Objektraum erfasst und soll dadurch die Wahrscheinlichkeiten für einzelne Objektklassen enthalten.

Es wurden schon mehrere Versuche von generativen Modellen auf 3D Daten durchgeführt wie von Wu, Jiajun und Zhang (Wu, Zhang, Xue, Freeman, & Tenenbaum, 2016) welche in ihren Modell mit mit 3D-Voxel Daten arbeiten und damit ein GAN trainiert haben.Auch Achlioptas, Panos und Diamanti(Achlioptas et al., 2018) welche ein GAN auf Punktwolken trainiert. Da in folgender Arbeit die 3D-Daten durch Punktwolken dargestellt werden wird die Arbeit von Achlioptas, Panos und Diamanti näher beleuchtet.

Die Architektur des typischen 3D-GAN oder auch RAW-GAN betitelt ist dem Vanilla GAN von Goodfellow ähnlich. Der Input Layer ist ein Fully-Connected Layer welcher der Anzahl der Punkte je Punktwolke * 3 entspricht. Dieser bekommt als Input einen Noise-Vektor welcher aus einer Normal Verteilung entnommen wird und durch mehrere Layern gereicht bis hin zum Output Layer welche die Anzahl der gewünschten Punkte besteht. Also Zielfunktion kann mit der KL-Zielfunktion gearbeitet werden oder mit der Wasserstein Metrik welche im Versuchsaufbau von Achlioptas, Panos und Diamanti besser Ergebnissee geliefert hat. Der Aufbau unterscheidet sich nicht von Vanilla-GAN(Achlioptas et al., 2018).

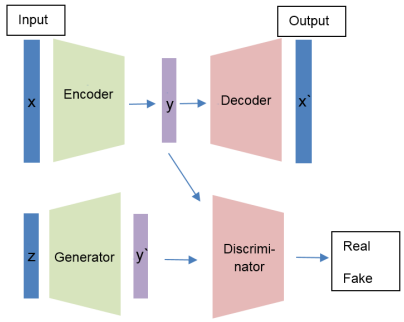

Abb. 20: Latent-GAN

Eine weitere Möglichkeit ist das Latent-GAN, dieses benutzt eine andere Aufbau gegenüber dem RAW-GAN, welches dabei helfen soll den latenten Raum

der Objekte zu erlernen. Zunächst wird ein Autoencoder mit den vorhanden Trainingsdaten(vgl. 2.4) trainiert. Der Aufbau ist der selbe einen üblichen 3D-Autoencoder beschrieben in Kapitel 2.4. Ziel dabei ist den latenten Raum der Trainingsdaten zu erlernen und eine Kompression der Daten um den Suchraum zu welcher beim RAW-GAN durch die Inputdimension gegeben ist, zu verringern und dadurch das Training des GANs zu erleichtern(Achlioptas et al., 2018).

Bevor das Training des Latent-GAN beginnen kann werden nun die Trainingsdaten durch den vorher trainierten Encoder des 3D-Autoencoder auf die festgelegte Output-Dimension komprimiert. Anschließend werden diese komprimierten Daten verwendet um das GAN zu trainierten. Dabei erlernt das GAN komprimierten Code zu produzieren, welcher anschließend durch den Decoder des 3D-Autoencoder wieder auf die ursprüngliche Größe gebracht werden kann. Durch dieses Verfahren war es möglich Daten in einer guten Qualität zu produzieren und eine Datenverteilung des zugrundeliegenden Models zu erlernen. Der gesamte Ablauf kann in Abb. 20 entnommen werden(Achlioptas et al., 2018).

2.8 Rekonstruktion von Daten

Derzeit setzt Deep Learning neue Maßstäbe bei der Rekonstruktion von Daten, wie Bildern oder Texten. Bei der Rekonstruktion geht es darum Daten, welche von ihren Urzustand verändert wurden, sei es durch Artefakte oder manuelle Bearbeitung, wieder dahin zurück geführt werden. In Abb. 21 ist eine Rekonstruktion eines Bildes dargestellt welches eine Wiederherstellung eines Hundes zeigt. Deep Learning hat in diesem Bereich besonders bei Bilder große Erfolge erzielt. Da diese als Matrizen dargestellt werden können, liefern sie eine Datenstruktur, auf welche KNN arbeiten können. Auch Bearbeitung mit Convolutionen-Layer auf Bilddaten zählt als Erfolg für die Weiterverarbeitung(Dong, Loy, He, & Tang, 2016b). Durch die genannten Methoden werden bessere Strukturen für das jeweilige Ziel, in diesem Fall die Rekonstruktion, gelernt und ermöglicht es wie in verschieden Papern wie (Dong et al., 2016a) welche auf super-hochauflösenden Bildern Artefakte entfernt und das Bild wieder zum Urzustand zurück führt.

Auch wie die Arbeit von Liu, Gulin und Reda (Liu et al., 2018) konnte ähnliche Ergebnisse liefern. Rekonstruktion auf 3D-Daten wurde von Yi, Li und Shao (Yi et al., 2017) durchgeführt. Diese Arbeiteten beinhaltet jedoch die Rekonstruktion von 2D Bildern auf 3D Modellen. Dabei wurde mit Hilfe von Autoencoder gearbeitet was auch für die folgenden Arbeit von Bedeutung sind(Yi et al., 2017). Da diese helfen durch die Dimensionreduktion eine einfachere Daten Weiterverarbeitung zu liefern.

Abb. 21: Bild Rekonstruktion eines Hundes welches durch ein Artefakt zerstört wurde

3 Methoden

In diesen Kapitel werden die Methoden für die Versuchsaufbauten 1 und 2 aufgezeigt welche die Ziele 1 und 2 überprüfen sollen, diese wurden in **??** festgehalten.

Ziel 1 Können durch GANs 3D-Punktwolken von Tabakblättern erlernt werden um neue Datensätze zu generieren?

Ziel2 Können durch GANs 3D-Punktwolken von Tabakblättern welche von Tabakblättern von ihren Urzustand abgebracht wurden rekonstruiert werden?

Dabei geht es darum ob mit Hilfe von GANs der latente Objektraum eines Tabakblattes gelernt werden kann und anschließend durch den Generator des GAN Tabakblätter erzeugt werden können. Dieses Versuchsaufbau wird in Kapitel 3.3 behandelt. Die dazugehörigen Trainingsdaten werden in Kapitel 3.1 vorgestellt. Beim Ziel 2 soll überprüft werden ob mit Hilfe von GAN es möglich ist Artefakte von Tabakblätter zu entfernen und den Urzustand wieder her zustellen ist. Dieses Verfahren wurde in Kapitel 2.8 beschrieben. Die Trainingsdaten für diesen Versuchsaufbau können aus Kapitel 3.2 entnommen werden.

Versuchsaufbau 1.2 und 2.1 wurden auf einen Computer mit Ubuntu 14.05 Betriebssystem mit einen IntelR CoreTM i7-7700k mit 4.50GHz einer Geforce GTX 1080 mit 8G Grafikspeicher. Training und Testen wurden mit CUDA 9.0 und cudNN 7.1.1. Also Programmiersprache wurde Python 2.7 bzw. 3.5 für Datengenerierung. Als ANN Libary wurde Tensorflow 1.5 genutzt und TFlearn 0.3.2. Für die EMD und Chamfer loss für den Autoencoder wurde die Implementierung von fanqgme (https://github.com/fanhqme/PointSetGeneration) benutzt.

Versuchsaufbau 1.1 und 2.2 wurden auf einen Computer mit Windows Betriebssystem mit einen IntelR CoreTM i7-7700k mit 4.50GHz einer Geforce GTX 1080 mit 8G Grafikspeicher. Training und Testen wurden mit CUDA 9.0 und cudNN 7.1.1. Also Programmiersprache wurde Python 3.5 verwendet. Als ANN Libary wurde Tensorflow 1.10 genutzt.

3.1 Datensatz 1. Versuchsaufbau

Der erste Datensatz SStühle"besteht aus 6778 Punktwolken mit je 2048 Punkten. Dieser wurde aus den Shapenet Datensatz(www.shapenet.org) entnommen. Die Daten sind PLY Datenformat abgespeichert. Ein Beispielpunktwolke kann aus Abbildung entnommen werden. Der Datensatz wurde in der Arbeit von Achlioptas, Diamanti, Mitliagkas und Guibas(Achlioptas et al., 2018) verwendet welches auch in Kapitel 2.7 vorgestellt wurde ist. In dieser Arbeit dient er als Validierungsdatensatz um die Qualität der generierten Daten des GAN welches mit Tabakblättern trainiert wurde zu vergleichen.

Die Daten für den zweiten Datensatz Blätter stammen vom Fraunhofer Institut deren Gewinnung wurde in Kapitel 2.2 beschrieben. Der Grunddatensatz bestand aus mehreren 3D-Scans von Tabakpflanzen bei den die Blätter der Pflanze zu einen Datensatz zusammengefügt wurden sind. Der Blätter Datensatz besteht aus 422 Punktwolken welche dann durch ein Punktreduktionsverfahren auf jeweils 2048 Punkten je Punktwolke reduziert wurden ist. Aus Komplexitätsgründen wurden der Farbchannel außen vor gehalten um den Informationsgehalt der Daten zu reduzieren und ein Training zu vereinfachen. Außerdem spielen Farben bei den derzeitigen Ziel der Arbeit, Rekonstruktion von Tabakblättern keine Rolle und nehmen keinen Einfluss auf die Verwendbarkeit des Ergebnisses. Abgespeichert werden die Daten im .ply Datenformat.

Abb. 22: 3D Punktwolke einer Tabakpflanze

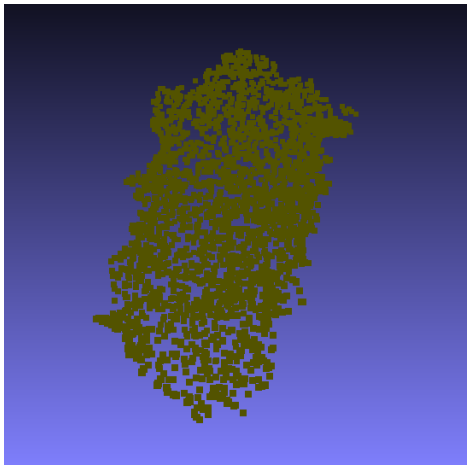

Abb. 23: 3D Punktwolke einer Tabakpflanze

3.2 Datensatz 2. Versuchsaufbau

Für den zweiten Versuchsaufbau wird auf den in Datensatz Versuchsaufbau 1. Blatt Datensatz "Blätterßurück gegriffen. Das heißt der Grunddatensatz besteht aus 422 unterschiedlichen Blättern. Dieser Datensatz wird nun dahingehen verändert um das Rekonstruieren von Tabakblättern welche durch Artefakte von ihren Urzustand verändert wurden Rückgengig zu machen. Die Artefakte simulieren dabei das Verdecken von anderen Blättern beim Scanverfahren was zu unvollständigen Blättern führen kann. Dieses verdecken wir durch 3D-Sphären simuliert wie in Abb. **??** dargestellt. Die Sphären bestehen aus 100 000 Punkte welche alle einen maximalen Radius von 15 mm haben und welcher jeder Punkt aus einer Normalverteilung entnommen wird um eine gleichmäßige Verteilung der Punkte zu gewährleisten. Die Sphären werden nun vom Koordinatenursprung in euklidischen Raum bewegt das sich 46 unterschiedlichen Sphären im Raum ergeben. Diese werden dahin gehen Bewegt um möglichst eine hohe Differenzmenge mit den 422 Blättern im Datensatz zu haben. In Abb 24 sind alle Sphären symbolisch eingefügt werden um dieses Vorgehen zu Veranschaulichen. Um nun das Verdecken zu Simulieren wird die Differenzmenge je Blatt mit einer der Sphären genommen. Wobei die Differenzmenge eines Punktes der nähst Nachbar in einen Radius von 3mm befindet. Durch die dieses Verfahren entstehen Kreisförmige Löcher in den Blättern welches in Abb. **??** dargestellt ist. Anschließend werden alle erzeugten ßerstörten"Blätter nach der Anzahl ihrer übrig geblieben Punkte gefiltert um zu gewährleisten das genügen Punkte aus dem Urblatt entfernt wurden sind und eine visuelle Diskrepanz zwischen den Blätter vorhanden sind. Was letztendlich zu 2047 zerstörten Blättern geführt hat mit jeweils 2048 Punkten.

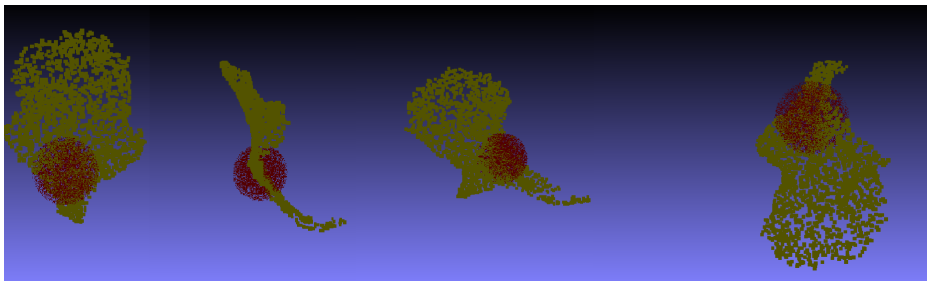

Abb. 24: Tabakblatt mit Sphäre welche die Differenzmenge bilden

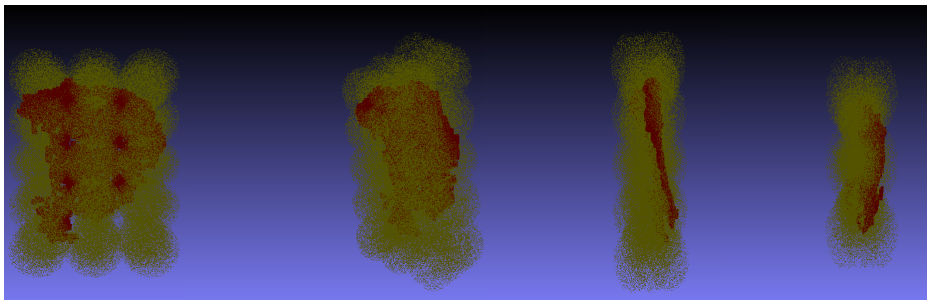

Abb. 25: Sphäre im Raum welche das verdecken der Blätter simulieren

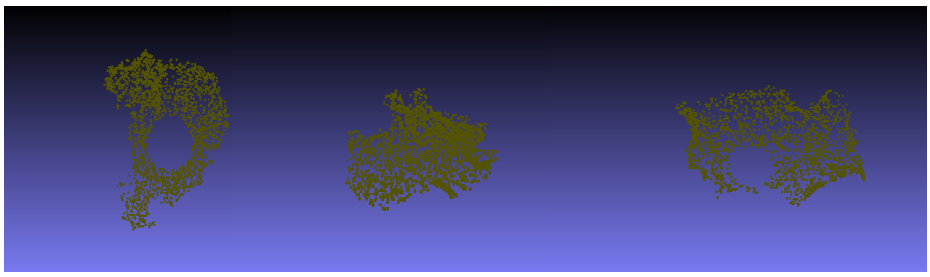

Abb. 26: 3D Punktwolke einer Tabakpflanze

3.3 Versuchsaufbau 1 - GAN

Der Versuchsaufbau 1 besteht aus zwei unterschiedlichen Versuchen. Aufbau 1.1 ist das RAW-GAN((Achlioptas et al., 2018)), dabei wird auf den Vanilla-GAN Aufbau zurückgegriffen. Die Meta Trainingsvariablen für diesen Versuch sind Learningrate mit 0,0005 und einen Adam Optimizer welcher zu den stochastischen Gradient Descent Verfahren aus Kapitel 2.1.2 und dabei auf das aus Kapitel 2.1.3 zurückgreift, mit einem Beta1 von 0,5 und einem Beta2 von 0,5. Die Batchgröße ist 64. Der Discriminator besteht aus 5-Layern welche aus 1-Dimensionalen Convolutionen-Layer bestehen mit einer Filteranzahl [64, 128, 256, 256, 512] je Layer. Mit einer Kernel Größe von 1 und Stride von 1. Also Aktivierungsfunktion wird die ReLu-Funktion genutzt. Darauf folgen 3 Fully-Connected-Layer mit der Größe [128, 64, 1] alle mit einer ReLu-Funktion. Der Generator des RAW-GAN besteht auf 6 Fully-Connected-Layern mit [64, 128, 512, 1024 ,1536 ,6144] Neuronen, jeweils mit der Relu Aktivierungsfunktion. Der Noisevektor z wird aus einer 128-D Normal Verteilung entnommen mit $\mu = 0$ und $\sigma = 0{,}2$. Als Zielfunktion wird die Wasserstein Metric gewählt da diese bessere Ergebnisse im Versuchsaufbau von Achlioptas, Panos und Diamanti(Achlioptas et al., 2018) erzielt hat. Das Training erfolgt jeweils mit den Datensätze SStuhlünd Tabakblätter"beschrieben in Kapitel 3.1.

Der Versuchsaufbau 1.2 ist das Latent-GAN welches in Kapitel 2.7 vorgestellt wurde. Es wurde die Implementierung von (Achlioptas et al., 2018) verwendet ($www.github.com/optas/latent_3d_points$). Zunächst wird dabei der Autoencoder mit den Trainingsdaten trainiert. Das Training erfolgt einer Learningrate von 0,0005 und einer Batchsize von 50. Der Encoder besteht dabei aus 4 1-D Convolutional-Layern mit [64, 128, 246, 1024] Filtern, eine Stride von 1 und Size von 1. Die Batchgröße ist 64. Als Aktivierungsfunktion wir die Relu verwendet. Der Encoder besteht aus 3 Fully-Connected-Layern mit [256, 256, 6144] Neuronen, alle mit der ReLu-Zielfunktion. Das Training erfolgt jeweils mit den Datensätze SStuhlünd Tabakblätter"beschrieben in Kapitel 3.1 und wurde auf 500 Epochen durchgeführt.

Nach dem Training des Autoencoder kann der komprimierte Latent Code welcher von Encoder erstellt wird dazu verwendet werden das Latent-GAN zu trainieren. Alle Trainingsdaten werden nun durch den Encoder komprimiert um ihre 128-D Latenten Code y zu erzeugen. Mit diesen wird nun das GAN trainiert um eigene Latente Code zu produzieren welche von den Decoder wieder auf ihre ursprüngliche Dimension von 2048 Punkten zu projizieren. In Abb. 20 ist dieser Prozess dargestellt. Der Generator des Latent-GAN besteht aus 2 Fully-Connected-Layer mit [128, 128] Neuronen welche als Aktivierungsfunktion eine ReLu-Funktion benutzen. Der Noisevektor z wird aus einer 128-D Normal Verteilung entnommen mit $\mu = 0$ und $\sigma = 0{,}2$. Der Discriminator besteht aus [128, 64, 1] Fully-Connected-Layern welche ebenfalls ReLu-Funktion nutzen.Als Zielfunktion wird die Wasserstein Metrik gewählt da diese bessere Ergebnisse im

Versuchsaufbau von Achlioptas, Panos und Diamanti(Achlioptas et al., 2018) erzielt hat.

3.4 Versuchsaufbau 2 - CGAN für Punktwolkenrekonstruktion

Da sich im Versuchsaufbau 1 das Latent-CGAN bessere Ergebnisse liefert bei erlernen von Punktwolken Daten. Wird im Versuchsaufbau 2.1 Latent-CGAN das komprimieren von Trainingsdaten übernommen und dahin gehen verändert, das Ziel von C-GAN zu übernehmen und bedingte Wahrscheinlichkeiten zu lernen. Zunächst werden wie bei Versuchsaufbau 1 die Trainingsdaten vgl.3.1 an einen Autoencoder trainiert. Dabei wird jeweils ein Autoencoder für zerstörte Blätter heran genommen und einer für unzerstörte im Urzustand befindende Blätter.

Die Autoencoder folgen dabei den gleichen Aufbau wie in Versuchsaufbau 1.2. Der Encoder besteht dabei aus 4 1-D Convolutional-Layern mit [64, 128, 246, 1024] Filtern, eine Stride von 1 und Size von 1. Als Aktivierungsfunktion wir die Relu verwendet. Der Decoder besteht aus 3 Fully-Connected-Layern mit [256, 256, 6144] Neuronen, alle mit der ReLu-Zielfunktion. Als Batchsize wird 64 genommen. Als Zielfunktion wird jeweils einmal die Chamfer Distance getestet und die Earthmover Distance. Für den Autoencoder wurde die Implementierung von (Achlioptas et al., 2018) verwendet ($www.github.com/optas/latent_3d_points$)

Der Generator des Latent-CGAN bekommt als Input z einen 128-D Vektor, welcher aus einer Normal Verteilung entnommen mit $\mu = 0$ und $\sigma = 0{,}2$. Y welches zusätzlich zu z den Generator zum Training eingespeist wird sind die von Autoencoder codierten 128-D komprimierten ßerstörteTabakblätter. Der Generator des Latent-GAN besteht aus 2-Fully-Connected-Layer mit [128, 128] Neuronen welche als Aktivierungsfunktion eine Leaky-ReLu-Funktion benutzen. Der Discriminator besteht aus 2 Fully-Connectected-Layern mit der Größe [128, 128]. Der Discriminator bekommt als Input entweder den vom Generator erzeugten latenten Code x' welcher vom Encoder des Autoencoder zu einen unzerstörten Blatt gemappt werden kann und zusätzlich y also (x‘,y). Oder ein aus unseren Trainingsdaten stammendes (x,y) Paar welches als reale-Datensätze gekennzeichnet ist. Als Ziel wird jeweils das Wasserstein Metrik und die Vanilla - GAN Metrik Loss-Function benutzt.

Beim Trainingsaufbau 2.2 RAW-CGAN bekommt der Generator als Input Noisevektor z einen 128-D Vektor aus einer Normal Verteilung entnommen mit $\mu = 0$ und $\sigma = 0{,}2$. Des weiteren wird als y den Generator die zerstörte Blatt Punktwolken welche als ein 6144-D Vektor dargestellt werden als Input übergeben. Der Generator besteht aus Fully-Connected-Layern mit [6272, 3163, 1568, 3136, 4850, 5680, 6144] Neuronen je Layer. Als Aktivierungsfunktion wird die Leaky-ReLu genommen. Im letzten Layern wird keine Aktivierungsfunktion verwendet um den Output linear Verarbeitbar zu machen und damit Euklidische Koordinaten generiert werden können. Des weiteren wird nach jedem Layer ein Batch-Normalisation-Layer eingesetzt mit Momentum 0,9 und einen Beta1 und Beta2

von 0,5 eingesetzt. Des Discriminator besteht aus [12288, 6144, 3072,1536, 768, 384, 128, 32, 1] Fully-Connected-Layern mit Aktivierungsfunktion Leaky-ReLu bis auf den letzten Layer in den wird die Sigmoid-Funktion verwendet, wenn als Zielfunktion die Vanilla-GAN gewählt wird. Bei Verwendung der Wasserstein Metrik wird auf eine Aktivierungsfunktion verzichtet. Des weiteren wird nach jedem Layer ein Batch-Normalisation-Layer eingesetzt mit Momentum 0.9 und einen Beta1 und Beta2 von 0,5. Der Input des Discriminator sind die von Generator erstellten (x',y) oder die direkt aus dem Trainingsdaten stammenden Punktwolken Paare zerstörtes Blatt y und Urzustand Blatt x als Tupel (x,y).Als Zielfunktion wird die Vanilla-GAN und W-GAN Zielfunktion getestet.

Im weiteren Versuchsaufbau für den RAW-CGAN wird der Discriminator umgebaut um ihn durch Convolutional-Layer bessere Lernfähigkeiten geben kann dies sorgt nicht nur für Vorteile für D dadurch kann auch der Generator durch den Gradianten des Discriminators einen besseren latenten Raum lernen soll. Der Generator bleibt in diesen Aufbau gleich der Discriminator bekommt [64, 128, 256, 256, 512] 1-D-Convolutional-Layer mit einer Kernel-Größe von 1 und Stride von 1 als Aktivierungsfunktion wird eine Leaky-ReLu verwendet. Die Batchsize beträgt 64. Anschließend folgen [128, 32, 1] Fully-Conncected Layer mit Leaky-Relu. Zwischen jeden Layer befindet sich ein Batch-Normalisation-Layer mit Momentum von 0,9 und einen Beta1 und Beta2 von 0,5. In Abb. 27 kann der Aufbau und Konnektivät der einzelnen Komponenten entnommen werden.

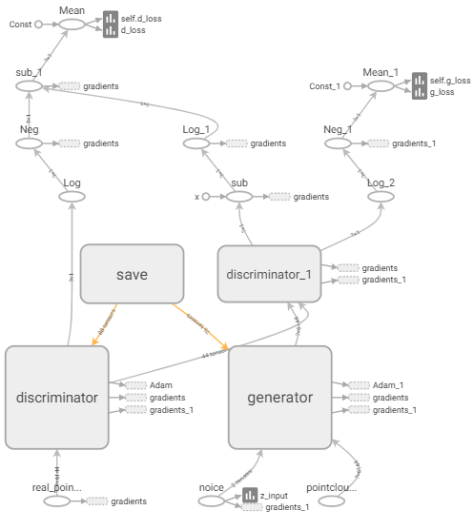

Abb. 27: Aufbau des RAW-CGAN

4 Evaluation und Ergebnisse

Es wird nun in diesen Kapitel auf die Ergebnisse von Versuchsaufbau 1 und 2 eingegangen. Zunächst wird Versuchsaufbau 1 vgl. 3.3 des erlernen von latenten Raum von Punktwolken gezeigt. Dabei wird ein Vergleich zwischen den Modeldaten Satz Stühle sowie einen aus der Praxis Stammenden Datensatz Tabakblätter aufgezeigt. Anschließend wird auf die Ergebnisse aus Versuchsaufbau 2 vgl. 3.4 eingegangen. Dabei wird evaluiert ob es möglich ist die zerstörten Blattdaten auf ihren Urzustand wieder herzustellen und aussagekräftige Ergebnisse für die Anwendbarkeit in der Praxis vorhanden sind. Da es keine gängige Validierungsmethode beziehungsweise Validierungsmetrik gibt wird bei der Prüfung der Testergebnisse auf das menschliche Auge verlassen und die Prüfung erfolgt durch eine Sichtprüfung.

4.1 Ergebnisse - Versuchsaufbau 1

Von RAW-GAN vom Versuchsaufbau 1.1 mit den Blattdaten welches nach 500 Epochen Training lassen sich in Abb.29 Beispieldatensätze welche vom Generator erzeugten wurden entnehmen. Wie zu erkennen ist sind die Blätter nicht gut genug für eine Weiterverarbeitung. Der Latente Raum welcher von den GAN gelernt wurde und am Ende durch den Generator produziert wurde zeigt keine Ähnlichkeit mit normalen Blättern welche im Datensatz enthalten waren. Die Daten erinnern eher an einen hohlen Ball als an Blättern. Es Formen sich Punktansammlungen nähe des Koordinatenursprungs und es lässt keine Anzeichen von Bilden einer Fläche da. Die Daten erinnern eher an einen hohlen Ball als an Blättern. Wobei eine starke Punkthäufung am Koordinaten Urpsprung festzustellen ist und einzelne verteile Punkte außen herum. Die Starke Punkthäufung am Koordinaten Ursprung ergibt dadurch das die reale Blattdaten an dieser stelle häufig vertreten sind und sich dann an den Blattkanten unterscheiden. Dadurch lernt der Generator an diesen hohen Schnittpunkt mehr Punkte zu generieren als Außerhalb. Dieser Effekt ist in Abb. 54 zu erkennen. Dargestellt sind 24 Datensätze aus den Blattdatensatz welche in einen Koordinaten System geladen wurden sind aus verschiedenen Blickwinkeln. Die Punkte häufen sich in der nähe des Koordinatenursprungs und sind geringer Verteilt im äußeren Bereich. Vergleicht man das nun mit Abb.**??** sind diese dichten Ansammlungen ebenfalls zu erkennen. Der W-GAN Loss für den Generator und Discriminator können in Abbildung 52 entnommen werden. Der Loss des Discriminator und Generator bleibt konstant über mehrere Epochen es ist kein Anzeichen von Verbesserung zu erkennen sollte das Training länger als 500 Epochen laufen.

Ähnliche Ergebnisse sind auch beim RAW-GAN vom Versuchsaufbau 1.1 mit Stuhldaten zu beobachten. Wobei die Qualität der Stühle mehr an die im Trainingsdatensatz stammende Daten erinnert. Es zeichnen sich Lehne und Stuhlbeine von dem GAN trainierten Generator erzeugten Daten ab. Der Trainingsverlauf ist in Abb. 31 zu entnehmen. Dabei ist zu sehen das der Discriminator sich nicht mehr verbessert und der Generator keine besseren Ergebnisse liefern kann. Diese Ergebnisse decken sich mit den von Achlioptas, Panos und Diamanti

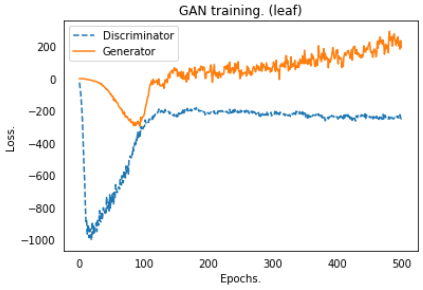

Abb. 28: Trainingsverlauf des RAW-GAN mit den Blattdaten

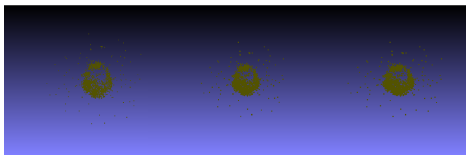

Abb. 29: Generierte Beispieldaten von Generator des RAW-GAN mit den Blattdaten

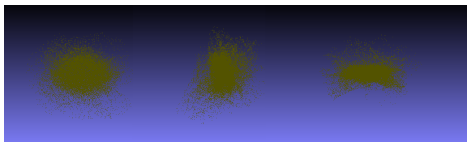

Abb. 30: 24 Pointclouds aus den Blattdatensatz in einen Koordinatensystem

(Achlioptas et al., 2018) festgestellten Ergebnissen in den das RAW-GAN keine qualitativ gute Stuhldaten erzeugen konnte.

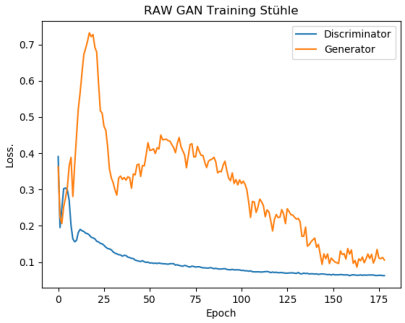

Abb. 31: Trainingsverlauf des RAW-GAN mit den Stuhldaten

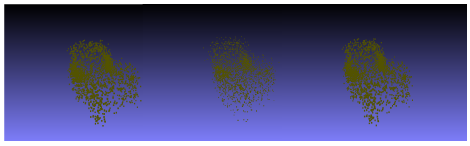

Abb. 32: Generierte Beispieldaten von Generator des RAW-GAN mit den Stuhldaten

Um nun die Ursache genauer zu begründen, welche die Gründe für den Qualitätsunterschied zwischen den beiden generierten Daten auszumachen und festzustellen ob durch mehr Blattdaten bessere Ergebnisse erzeugt werden können. Wurden die Stuhldaten auf 422 Trainingsdaten reduziert und ein erneutes Training des RAW-GAN durchgeführt. Die erzeugte Beispieldaten können aus Abb. 33 entnommen werden. Vergleicht man nun die Qualität von den RAW-GAN mit 422 Stuhldaten Trainingsdaten und 6778 Stuhldaten Abb.34 ist eine Steigerung der Qualität klar festzustellen. Der Trainingsverlauf in Abb. 34 ähnelt der von Abb .33. Der Discriminator ist zu schnell perfekt trainiert und gibt den Generator keine Chance mehr nachzuziehen. Es ist eine Tendenz festzustellen das durch eine Erhöhung der Blattdaten auch die Qualität des RAW-GAN steigern könnte.

Für Versuchsaufbau 1.2 Latent-GAN mit Stühlen kann der Trainingsverlauf aus Abb. 35 entnommen werden. Der Generator Loss verbessert sich zwar gering über mehre Epochen der Discriminator jedoch bleibt konstant auf einen guten Ergebnisse und lässt keine starken Verbesserung des Generators mehr

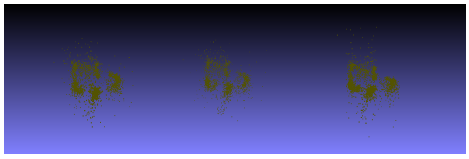

Abb. 33: Trainingsbeispiele des RAW-GAN mit den 422 Stuhldaten

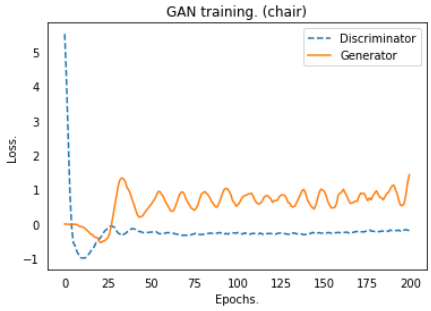

Abb. 34: Trainingsverlauf des RAW-GAN mit den 422 Stuhldaten

zu. Beispieldatensätze welche von Generator nach Beendigung des Trainings erzeugt wurden in dem der latente-Code von G in den Decoder als Input gegeben wurde sind in Abb. 36 zu entnehmen. Diese sind qualitativ hochwertige Daten welche eine Ähnlichkeit mit denen aus dem Trainingsdatensatz widerspiegeln. Die Ergebnisse decken sich mit den von (Achlioptas et al., 2018) festgestellten Ergebnissen in den das RAW-GAN keine qualitativ gute Stuhldaten erzeugen konnte.

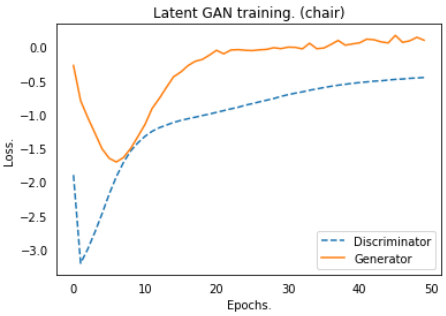

Abb. 35: Trainingsverlauf des Latent-GAN mit den Stuhldaten

Das Latent-GAN welches mit den Blattdaten trainiert wurde können exemplarisch generierte Daten aus der Abb. 38 entnommen werden. Diese zeigen eine

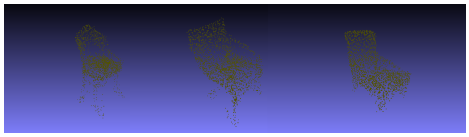

Abb. 36: Trainingsergebnisse des Latent-GAN mit den Stuhldaten

qualitativ gute Grundfläche welche in der Trainingsdatengesamtheit enthalten ist. Jedoch ist festzustellen das es starke Punkthäufungen an gewissen Bereichen je Blatt gibt, diese sind beispielsweise im 2. Blatt von Rechts in Abb. 38 zu erkennen und mit einen roten Kreis markiert. Der Trainingsverlauf in Abb. 37 zu entnehmen sind keine starken Veränderungen im weiteren Trainingsverlauf zu entnehmen das Training stagniert schon über mehrere Epochen.

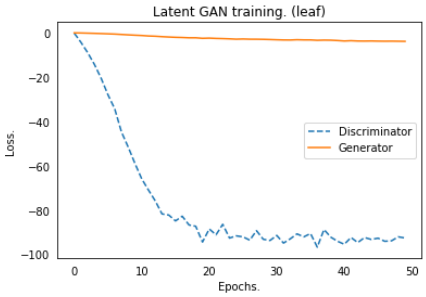

Abb. 37: Trainingsverlauf des Latent-GAN mit den Blattdaten

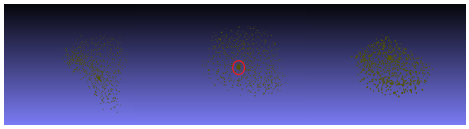

Abb. 38: Trainingsergebnisse des Latent-GAN mit den Blattdaten

Um die Ergebnisse von Testaufbau 1.2 Latent-GAN mit Stuhldaten besser zu Vergleichen und um ersichtlich zu machen ob eine Erhöhung der Anzahl der Tabakblätter Daten zu einen besseren Ergebniss führen kann und die Punktanhäufung sich reproduzieren lässt, wurden 422 Stuhldaten genommen um das Latent-GAN zu trainieren. In Abb. 40 sind die exemplarische erzeugte Daten von Generator zu entnehmen. Zu erkennen ist ein ähnlich Phonemen wie bei den Blattdaten. Hohe Punktanhäufungen an Kanten der Sitzfläche zum Übergang der Stuhlbeine. Dadurch lässt sich Rückschlüsse ziehen das eine Erhöhung der

Daten zu einer besseren Qualität der Daten führen kann. Der Trainingsverlauf kann aus Abb. 39 entnommen werden auch dieser zeigt eine Stagnation des Verlaufs schon über mehre Epochen an und verspricht keine Erhöhung der Qualität.

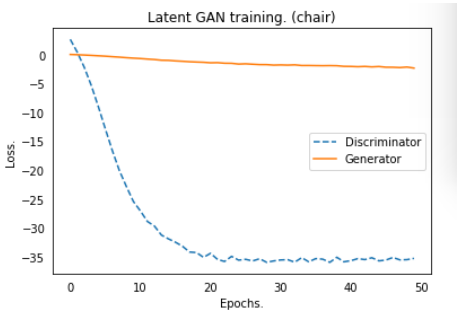

Abb. 39: Trainingsverlauf des Latent-GAN mit den Stuhldaten und 400 Trainingsdaten

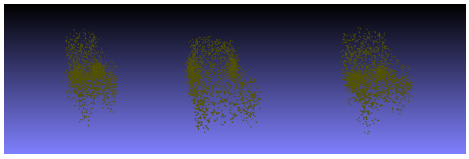

Abb. 40: Traingsergebnisse des Latent-GAN mit den Stuhldaten und 400 Trainingsdaten

Zusammengefasst besonders auf den Hinblick des Blattdatensatzes lässt sich feststellen das bei den Stuhldatensatz die Ergebnisse von Achlioptas, Panos und Diamanti(Achlioptas et al., 2018) reproduziert werden konnte. Bei einen Datensatz welcher durch Scanverfahren von realen Objekten in diesen Beispiel Tabakblättern konnte gezeigt werden das durch das Latent-GAN eine erste Annäherung an qualitativ Hochwertig generierten Daten zu sehen ist. Durch eine Erhöhung der Trainingsdaten lässt sich auch eine Erhöhung der Qualität feststellen wie bei den Stuhldaten gezeigt wurde, sowie bei RAW-GAN und Latent-GAN. Des weiteren können aber keine Aussagen getroffen werden um welche Höhe die Qualitätssteigerung statt finden kann. Die Datensätze unterscheiden sich in ihrer Aufbereitung. Die Stuhldaten sind besser auf einen Koordinaten Ursprung geeicht wie in Abb. 41 festzustellen ist, in dieser sind mehre Stühle in ein Koordinaten System geladen zum Vergleich die Blattdaten im Abb. 54 Zu erkennen ist einen enorme Überschneidung der einzelnen Punktwolken bei Stühlen im Vergleich zu Blattdaten dies hilft den Suchraum für die GANS ein-

zuschränken und erleichtert das Training. Es müssten sich bessere Datenvorverarbeitungsschritte bei den Blattdaten erhoben werden. Außerdem sind die 422 Blattdaten zum Vergleich zu den 6778 Stuhldaten sehr gering und hilft den Latenten-GAN und RAW-GAN den latenten Raum besser zu erlernen da die Grundgesamtheit mehr abgedeckt ist. Aus technischer Sicht könnte bessere Layer strukturen für bessere Ergebnisse. Die Bearbeitung mit Convolutionen-Layer auf Bilddaten hat erst den großen Durchbruch bei der Rekonstruktion bei Bilddaten möglich gemacht(Dong et al., 2016b). Da nun unser Generator nur aus Fully-Connected-Layern besteht und 3D-Punktwolken komplexer sind als Bilder kann an dieser Stelle definitiv noch nachgearbeitet werden. Beispielsweise könnte ein 1D-Deconvolutional-Layer implementiert werden. Welche bei Tensorflow 1.12 als Prototyp zur Verfügung steht. Eine andere Möglichkeit wäre auf die Arbeit von Cai, Zhongang und Yu (Cai, Yu, & Pham, 2018) anzusetzen dieser postuliert eine Methode für 3D-Convolution Methode welche auf 3D-Punktwolken arbeitet und eine Invarianz im Euklidischen Raum lernt mit deren Hilfe es leicht sein soll Rotationen der Punktwolke zu erlernen.

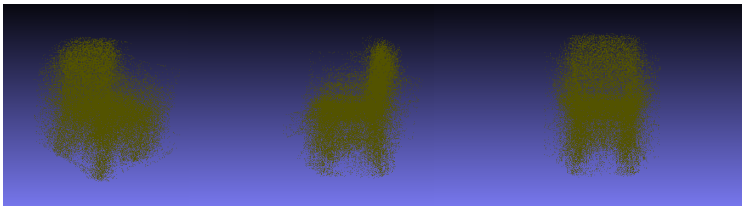

Abb. 41: 24 Stuhldaten in einen Koordinaten System geladen

4.2 Ergebnisse - Versuchsaufbau 2

Für Versuchsaufbau 2.1 vgl. 3.4 wurde zunächst der Autoencoder mit den zerstörten Blattdaten auf 500 Epochen trainiert. Um eine komprimierte Version auf 128-D zu erlernen. Dabei wurde zunächst die in Kapitel 2.4 vorgestellte Chamfer Distanz als Distanzmass genommen. Beispiel Trainingsdatensätze können Abb. 47 entnommen werden mit ihren den jeweiligen erzeugten Output des Decoders in Abb. 43. Wie zu erkennen ist vervollständigt der Autoencoder mit der Chamfer Zielfunktion die Blätter selbstständig obwohl das nicht als Ziel in diesen Bearbeitungsschritt ist. Es ist zu erkennen, dass die Dichte in welche die Punkte beim Output auf der Höhe der Löcher angeordnet ist stark von den üblichen Bereichen auf den Blatt abweicht. Auch lässt der Trainingsverlauf in Abb. 42 darauf schließend das selbst beim weiteren Training der Autoencoder keine bessere Codierung erlernen wird, welche die Löcher in den Blättern erzeugt, da seit Epoche 300 das Training stagniert. Außerdem lassen das durch die Chamfer Distanz berechnete Abstand zwischen den beiden Punktwolken keine starke Verbesserung mehr zu da die durchschnittliche Diskrepanz nach dieser Metrik, zwischen den Input x und den von Decoder erzeugten Output x‘ nur noch im 0.0005 cm Bereich liegt nach wie in Abb.42 in Epoche 500 abzulesen ist. Da keine vernünftige Trainingsdaten für den Latenten C-GAN Versuchsaufbau 2.1 generiert werden können kann an dieser Stelle der Versuchsaufbau 2.1 Latent C-GAN mit Chamfer Distanz nicht weiter geführt können.

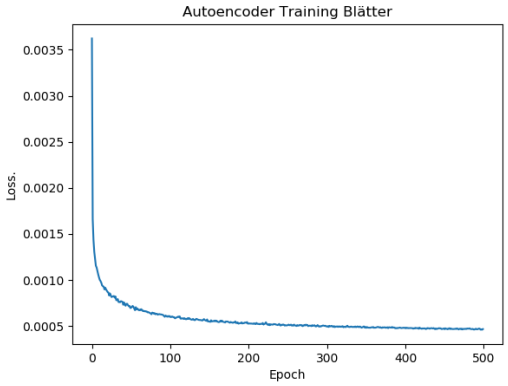

Abb. 42: Trainingsverlauf des Autoencoder mit CD für zerstörte Blattdaten

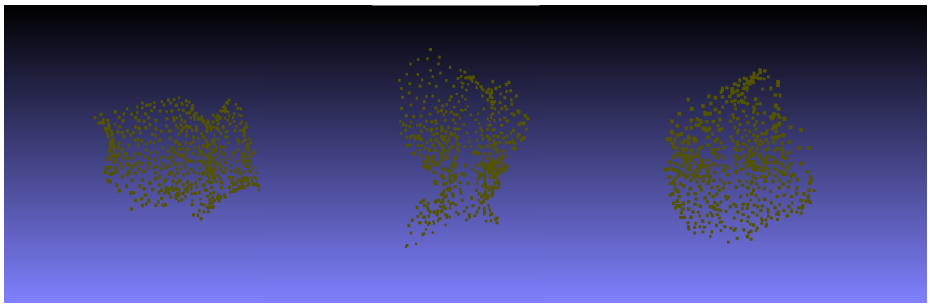

Abb. 43: Trainingsbeispiele des Autoencoder mit CD für zerstörte Blattdaten - Output

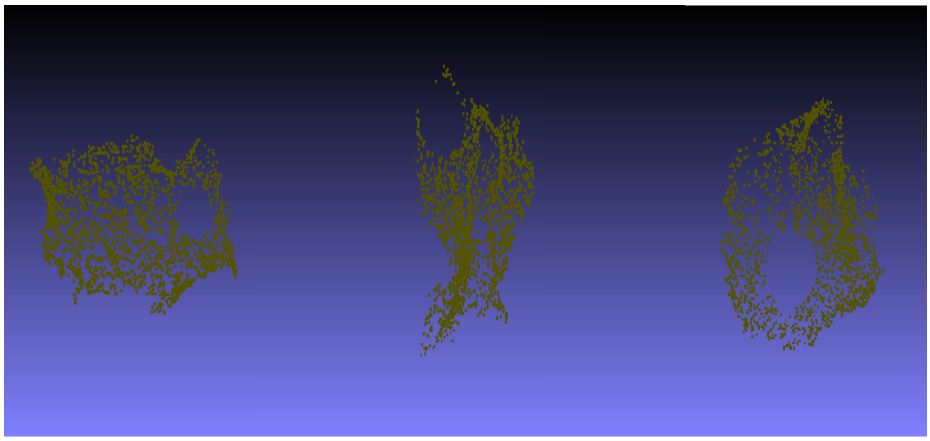

Abb. 44: Trainingsbeispiele des Autoencoder mit CD für zerstörte Blattdaten - Input

Um nun bessere Validierungsergebnisse zu erlangen wurde Versuchaufbau 2.1 latent-CGAN noch mit EMD Zielfunktion für Autoencoder getestet. Es wurden 800 Epochen trainiert. Die Ergebnisse sind recht ähnlich zu der mit der Chamfer Distanc vgl.2.4. Beispiel Trainingsinput Trainingsdatensätze können in Abb. 45 entnommen werden mit den jeweiligen erzeugten Output des Decoders in Abb. 46 entnommen. Wie zu erkennen ist, vervollständigt der Autoencoder mit der EMD die Blattflächen selbstständig obwohl das nicht als Ziel des Autoencoder ist, sondern nur das lernen einer Kodierung. Im Vergleich dazu sind die Blätter aber an der Stelle des Loches gleichmäßiger Verteilt und es gibt keine höheren Punktansammmlungen auf der Blattfläche. Wenn man die Ergebnisse mit der Hinsicht der Zielfunktion berücksichtigt liefert die CD ein besseres Ergebnis. Mit der Begründung das auf der Blattoberfläche die Blätter gleichmässiger Verteilt wurden sind. Möchte man nun die beiden Ergebnisse ebenfalls auf die Qualität des Rekonstruieren messen obwohl dies nicht das eigentliche Ziel ist sticht die EMD hervor da auch auf in den Bereich der Löcher die Punkte gleichmäßiger Verteilt sind.

Insgesamt sind die Distanzen niedriger und die Dichte an den Löchern nimmt ab. Jedoch liefert die EMD bessere den besseren Nebeneffekt das die Blätter realistischer Rekonstruiert werden. 42 darauf schließend das selbst beim weiteren Training nicht die Löcher welche in den Blättern sind herzustellen sind, da seit Epoche 300 das Training stagniert. Außerdem lassen das durch die Chamfer Distanz berechnete Abstand zwischen den beiden Punktwolken keine starke Verbesserung mehr zu da die durchschnittliche Diskrepanz zwischen den Input x und den von Decoder erzeugten Output x‘ nur noch im 0.03 cm Bereich liegt wie in Abb.42 in Epoche 500 abzulesen ist. Das Training entwickelt sich jedoch seit mehren Epochen nicht mehr und stagniert. Auch mit dieser Anwendung kann das latente C-GAN nicht weitergeführt werden da keine erfolgreichen Trainingsdaten erzeugt werden konnten.

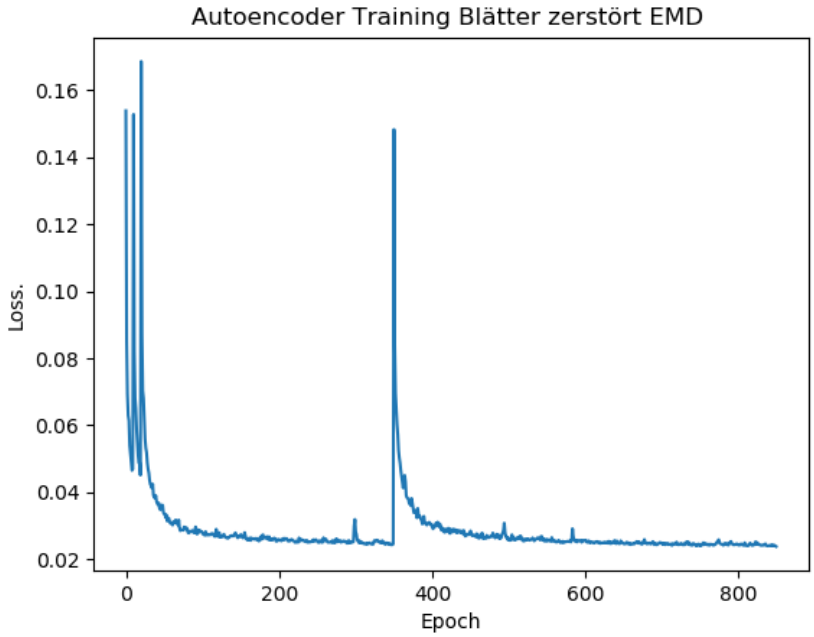

Abb. 45: Trainingsverlauf des Autoencoder mit EMD für zerstörte Blattdaten

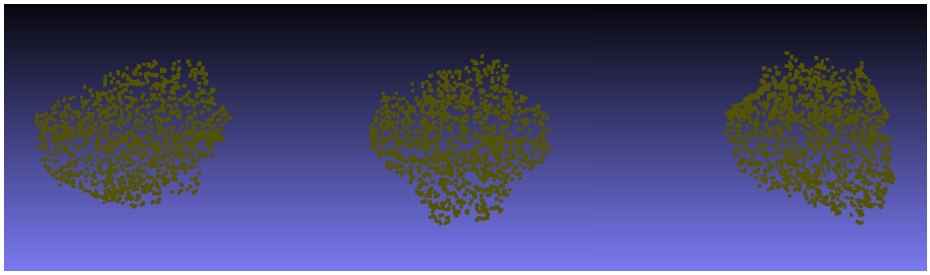

Abb. 46: Trainingsbeispiele des Autoencoder mit EMD für zerstörte Blattdaten - Output

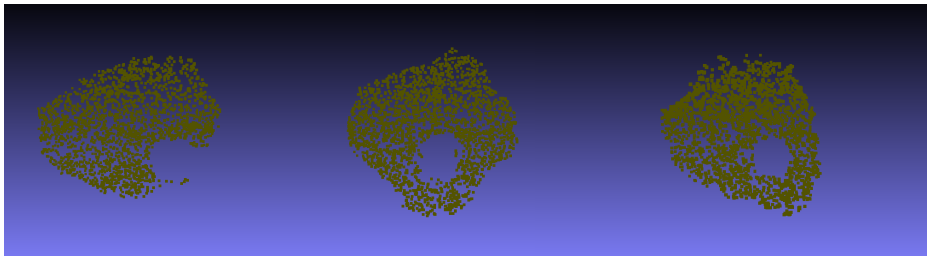

Abb. 47: Trainingsbeispiele des Autoencoder mit EMD-Distanz für zerstörte Blattdaten - Input

Für den Versuchsaufbau 2.2 RAW-CGAN mit Fully-Connected-Layern mit Wasserstein Metric ist in Abb. 48 ist rechts der Loss des Discriminators abgebildet und links der Loss des Generators. Der Loss des Generator sinkt zwar aber die generierten Trainingsbeispiele in Abb. 50 ähneln keine reparierten Blätter. Der dazugehörige Input ist in Abb.49 zu sehen. Da der Generator schon an sein Limit gerät und keine erhebliche Verbesserung mehr Möglich ist, ist festzustellen das durch einen Generator mit Fully-Connected-Layern keine Blattdatenrekonstruktion statt finden kann.

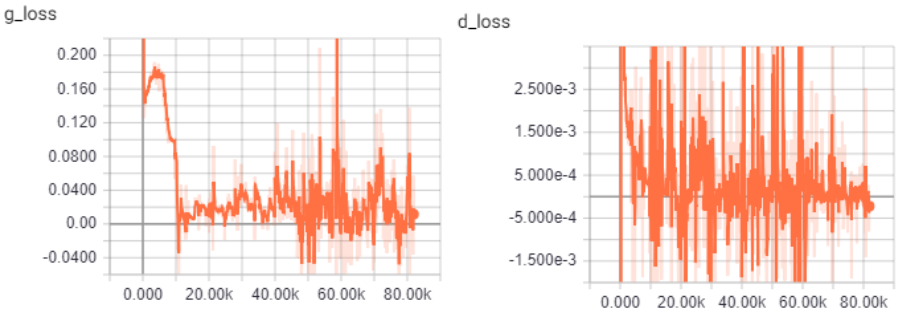

Abb. 48: Generator und Discriminator Loss des RAW-CGAN mit Wasserstein Distanz

Als nächtest erfolgen die Ergebnisse von Testaufbau 2.2 RAW-CGAN mit Convolutional Layer im Discriminator und der Vanilla-GAN-Loss. Der Trainingsverlauf vom Generator und Discriminator kann in Abb. 51 entnommen werden. Die Sprünge schauen auf den ersten Blick sehr stark aus beachtet man aber die Scalenwerte auf der Y-Achse des Generators bewegt sich der Loss zwischen 0.691 und 0,696. Dies lässt auf keine starke Verbesserung schließen. Das Training läuft

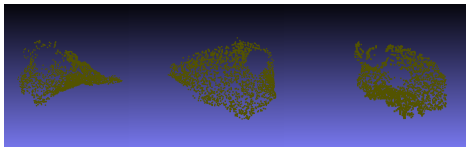

Abb. 49: Der Input aus welchen vom RAW-CGAN die Daten rekonstruiert wurden

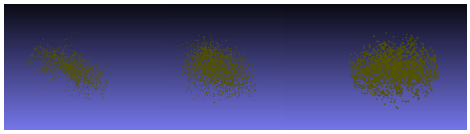

Abb. 50: Vom Generator des RAW-CGAN rekonstruierte Daten

über mehre Epoche konstant hinweg und lässt auch für eine längeren Durchlauf auf keine Verbesserung schließen. Einige erstellten Beispieltabakblätter welche vom Generator erstellt wurden sind können aus Abb. 52 zu entnehmen so wie der dazu gehörige Input aus Abb. 53. Diese sind jedoch keine hohe Ähnlichkeit zu Blätter zu erkennen. Und da es nach der Loss Funktion keine gravierenden Veränderungen mehr zu erwarten sind kann festgehalten werden das keine Rekonstruktion von zerstörten Blattdaten mit den in Kapitel 3.4 beschriebenen RAW-CGAN mit Convolutional-Layer und Vanilla-GAN-Loss möglich ist.

Abb. 51: RAW-CGAN Loss

Zuletzt erfolgen die Ergebnisse von Testaufbau 1.2 RAW-CGAN mit Deconvolutional-Layer mit der Wasserstein-Metric, welches mit den nicht komprimierten Daten trainiert wird. Dass heißt wie in 3.2 beschrieben Blattdaten. Der Trainingsverlauf vom Generator und Discriminator kann in Abb. 54 entnommen werden. Aus den Verlauf des Discriminator Loss arbeitet sich konstant nach unten hat zwei Ausbrecher welche aber kein Einfluss auf den weiteren Trainingsverlauf hat. Der Generator springt zwar noch am Ende des Trainings aber die Sprünge sind in einen so niedrigen Intervall das keine große Veränderung mehr erwartet werden kann. Die vom Generator generierten Outputs in Abb. 55 und deren dazugehöriger Input in Abb. 56 zeigen schon die Umrisse der Blätter auf jedoch sind die Blattflächen nicht gebildet.

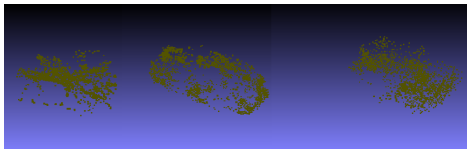

Abb. 52: Der Output aus welchen vom RAW-CGAN mit Convolutional Layern die Daten rekonstruiert wurde

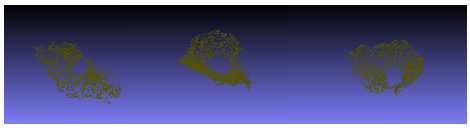

Abb. 53: Der Input aus welchen vom RAW-CGAN mit Convolutional Layern die Daten rekonstruiert wurden

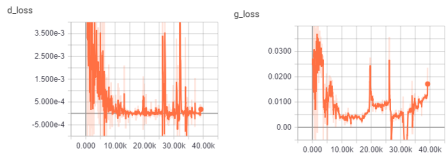

Abb. 54: Discriminator und Generator Loss RAW-DGAN mit Convolutional-Layer und Wasserstein Distanz

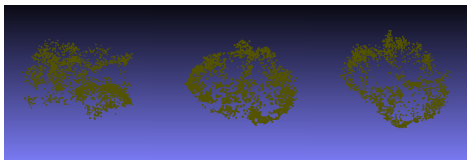

Abb. 55: Der Output aus welchen vom RAW-CGAN mit Convolutional Layern die Daten rekonstruiert wurde

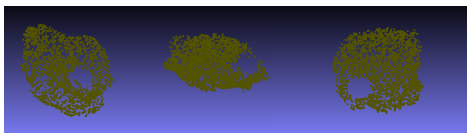

Abb. 56: Der Input aus welchen vom RAW-CGAN mit Convolutional Layern die Daten rekonstruiert wurden

Fast man nun Versuchsaufbau 2 zusammen. Ist festzustellen das in Versuchsaufbau 2.1 abgebrochen werden musste da die Autoencoder nicht fähig waren die zerstörten Trainingsdaten zu erlernen. Jedoch schafften sie die Daten zu rekonstruieren obwohl dies nicht als Ziel für die Autoencoder definiert wurde. Ob eine praktische Anwendung dadurch ermöglicht ist und ob diese Rekonstruktion nur für die Form für Blätter gilt oder auch Beispielsweise ähnliche Ergebnisse beim Stuhldaten Satz liefert muss geprüft werden. Die beiden Zielfunktionen des Autoencoder, CD und EMD, hatten beide ein ähnliches Problem welcher Punkt in Punktwolke x ist die dazugehörige Punkt in Punktwolke y. Beide lösten dies indem sie die nähsten Nachbarn eines Punktes suchen aus der anderen Punktwolke. Jedoch für dieses Mapping zu Problemen führen wie man in Abb. erkennen kann. Die rote Punktwolke ist der erste Versuch des Autoencoder die grüne Punktwolke zu kopieren. Eine Zuordnung in diesem Stadium welcher Punkt in rot zu grün gehört ist sehr schwierig. Dadurch resultiert das Punkte in Zentrum eher als nächster Nachbar identifiziert werden. Und lassen in weiteren Trainingsverlauf die Punktwolke wie einen Teig außernander gehen und lassen es nicht zu die Löcher zu entstehen. Für eine weitere Nutzung könnte an dieser Stelle angestetzt werden und andere Zielfunktionen für Autoencoder mit Punktwolken entwickelt werden.

Im Versuchsaufbau 2.2 zeigen sich bessere Ergebnisse mit der Verwendung von Convolutional-Layer im Discriminator. Ein qualitativer Unterschied zwischen den Versuchsaufbau 2.2 RAW-CGAN mit Convolutional-Layer und Vanilla-GAN oder Wasserstein Metrik lässt sich nicht feststellen. Jedoch lässt durch die Convolutional-Layer das erlernen der Latenten-Struktur besser gelingen. Eine Rekonstruktion ist aber auch mit diesen Model nicht möglich. Jedoch kann nicht ausgeschlossen sein das GAN durch andere Aufbauten wie Beispielsweise mit 1D-Deconvolutional-Layer implementiert werden. Welche bei Tensorflow 1.12 als Prototyp zur Verfügung steht. Eine andere Möglichkeit wäre auf die Arbeit von Cai, Zhongang und Yu (Cai et al., 2018) anzusetzen dieser postuliert eine Methode für 3D-Convolution Methode welche auf 3D-Punktwolken arbeitet und eine Invarianz im Euklidischen Raum lernt mit deren Hilfe es leicht sein soll Rotationen der Punktwolke zu erlernen. Durch diese neuen Module könnte das erlernen des latenten Raumes von 3d-Punktwolken möglich gemacht werden und das letztendliche Ziel, 3D-Punktwolken zu rekonstruieren möglich gemacht werden.

5 Zusammenfassung und Diskussion

3D-Daten bleiben eine Herausforderung zu den vergleichsweise anderen Daten wie Bilder oder Audio da ihr Suchraum erheblich höher ist, dieses Thema wurde auch schon in Kapitel 2.2.2 thematisiert. Die beiden Ziele:

Ziel 1 Können durch GANs 3D-Punktwolken von Tabakblättern erlernt werden um neue Datensätze zu generieren?

Ziel2 Können durch GANs 3D-Punktwolken von Tabakblättern welche von Tabakblättern von ihren Urzustand abgebracht wurden rekonstruiert werden?

welche in dieser Arbeit überprüft wurden, ist festzuhalten das Ziel 1 durch GANs machbar ist jedoch noch Verbesserungen in der Qualität möglich sind. Diese können durch Beispielsweiße, erhöhung der Trainingsdaten, eine Datenvorverarbeitung welche den Suchraum einschränkt oder Verbesserung des GAN Modells durch verwendung von 1D- deconvolutional-Layer oder 3D-Convolutional-Layer für Punktwolken (Cai et al., 2018) möglicher weiße erreicht werden.

Für Ziel 2 welches in Kapitel 3.4 überprüft wurde ist festzuhalten das der Aufbau des Latent-CGAN abgebrochen werden musste, da der Autoencoder es nicht geschafft das die zerstörten Trainingsdaten zu erlernen.

auch zeigt beispielsweiße bei der Rekonstruction von Bildern das erst durch den Einsatz verbesserter Convolution Methoden es erst Möglich gemacht wurde Bilder zu rekonstruieren und zu verändert (Dong et al., 2016b).

Eine weitere Möglichkeit ist das Konzept welches Zhu, Jun-Yan und Park (Zhu, Park, Isola, & Efros, 2017), mit ihren Pix2Pix Model vorgestellt haben. In dieser Verwendet sich die C-GAN Loss Funktion und ändern diese um Zusätzlich zu der üblichen Loss-Funktion die Differenz zwischen Generator Output und den dazugehörigen Urzustand Trainingsdatensatz zu berechnen und addieren dies zu der Loss-Funktion. Der Optimizer versucht nun dieses Differenz zusätzlich zu Minimieren. Man könnte dieses Verfahren nun für Punktwolken übernehmen und mit der CD oder EMD implementieren.

Fast man nun Versuchsaufbau 2 zusammen. Ist festzustellen das in Versuchsaufbau 2.1 abgebrochen werden musste da die Autoencoder nicht fähig waren die zerstörten Trainingsdaten zu erlernen. Jedoch schafften sie die Daten zu rekonstruieren obwohl dies nicht als Ziel für die Autoencoder definiert wurde. Ob eine praktische Anwendung dadurch ermöglicht ist und ob diese Rekonstruktion nur für die Form für Blätter gilt oder auch Beispielsweise ähnliche Ergebnisse beim Stuhldaten Satz liefert muss geprüft werden. Die beiden Zielfunktionen des Autoencoder, CD und EMD, hatten beide ein ähnliches Problem welcher Punkt in Punktwolke x ist die dazugehörige Punkt in Punktwolke y. Beide lösten dies indem sie die nähsten Nachbarn eines Punktes suchen aus der anderen Punktwolke. Jedoch für dieses Mapping zu Problemen führen wie man in Abb. erkennen kann. Die rote Punktwolke ist der erste Versuch des Autoencoder die grüne Punktwolke zu kopieren. Eine Zuordnung in diesem Stadium

welcher Punkt in rot zu grün gehört ist sehr schwierig. Dadurch resultiert das Punkte in Zentrum eher als nächster Nachbar identifiziert werden. Und lassen in weiteren Trainingsverlauf die Punktwolke wie einen Teig außernander gehen und lassen es nicht zu die Löcher zu entstehen. Für eine weitere Nutzung könnte an dieser Stelle angestetzt werden und andere Zielfunktionen für Autoencoder mit Punktwolken entwickelt werden.

Im Versuchsaufbau 2.2 zeigen sich bessere Ergebnisse mit der Verwendung von Convolutional-Layer im Discriminator. Ein qualitativer Unterschied zwischen den Versuchsaufbau 2.2 RAW-CGAN mit Convolutional-Layer und Vanilla-GAN oder Wasserstein Metrik lässt sich nicht feststellen. Jedoch lässt durch die Convolutional-Layer das erlernen der Latenten-Struktur besser gelingen. Eine Rekonstruktion ist aber auch mit diesen Model nicht möglich. Jedoch kann nicht ausgeschlossen sein das GAN durch andere Aufbauten wie Beispielsweise mit 1D-Deconvolutional-Layer implementiert werden. Welche bei Tensorflow 1.12 als Prototyp zur Verfügung steht. Eine andere Möglichkeit wäre auf die Arbeit von Cai, Zhongang und Yu (Cai et al., 2018) anzusetzen dieser postuliert eine Methode für 3D-Convolution Methode welche auf 3D-Punktwolken arbeitet und eine Invarianz im Euklidischen Raum lernt mit deren Hilfe es leicht sein soll Rotationen der Punktwolke zu erlernen. Durch diese neuen Module könnte das erlernen des latenten Raumes von 3d-Punktwolken möglich gemacht werden und das letztendliche Ziel, 3D-Punktwolken zu rekonstruieren möglich gemacht werden.

Abbildungsverzeichnis

Literatur

Achlioptas, P., Diamanti, O., Mitliagkas, I., & Guibas, L. (2018). Learning representations and generative models for 3d point clouds.

Ahmed, E., Saint, A., Shabayek, A. E. R., Cherenkova, K., Das, R., Gusev, G., ... Ottersten, B. (2018). Deep learning advances on different 3d data representations: A survey. *arXiv preprint arXiv:1808.01462*.

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

batchnorm chun. (n.d.). `http://sanghyukchun.github.io/88/`. (Accessed: 2019-02-16)

Cai, Z., Yu, C., & Pham, Q.-C. (2018). 3d convolution on rgb-d point clouds for accurate model-free object pose estimation. *arXiv preprint arXiv:1812.11284*.

dc-gan online book. (n.d.). `https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html`. (Accessed: 2019-02-16)

Dong, C., Loy, C. C., He, K., & Tang, X. (2016a). Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, *38*(2), 295–307.

Dong, C., Loy, C. C., He, K., & Tang, X. (2016b). Image super-resolution using deep convolutional networks. , *38*(2), 295–307.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Fan, H., Su, H., & Guibas, L. (2017). A point set generation network for 3d object reconstruction from a single image. In *2017 ieee cosnference on computer vision and pattern recognition (cvpr)* (pp. 2463–2471).

gan lilian weng. (n.d.). `https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html`. (Accessed: 2019-02-16)

Golik, P., Doetsch, P., & Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech* (Vol. 13, pp. 1756–1760).

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Haque, M. R., Islam, M. M., Iqbal, H., Reza, M. S., & Hasan, M. K. (2018). Performance evaluation of random forests and artificial neural networks for the classification of liver disorder. In *2018 international conference on computer, communication, chemical, material and electronic engineering (ic4me2)* (pp. 1–5).

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Huang, G., Yuan, Y., Xu, Q., Guo, C., Sun, Y., Wu, F., & Weinberger, K. (2018). *An empirical study on evaluation metrics of generative adversarial*

networks. Retrieved from `https://openreview.net/forum?id=Sy1f0e-R-`

Huszár, F. (2015). How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A., & Catanzaro, B. (2018). Image inpainting for irregular holes using partial convolutions. *arXiv preprint arXiv:1804.07723*.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Morabbi, S., Keyvanpour, M., & Shojaedini, S. V. (2018). A new method for p300 detection in deep belief networks: Nesterov momentum and drop based learning rate. *Health and Technology*, 1–16.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems* (pp. 841–848).

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ravanbakhsh, S., Schneider, J., & Poczos, B. (2016). Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems* (pp. 2234–2242).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks form overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems* (pp. 82–90).

Yi, L., Shao, L., Savva, M., Huang, H., Zhou, Y., Wang, Q., ... others (2017). Large-scale 3d shape reconstruction and segmentation from shapenet core55. *arXiv preprint arXiv:1710.06104*.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.