# Table of Contents



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

# 3-D Datenrekonstruktion mit generative Adverserial Networks

Andreas Wiegand

Master These in künstlicher 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 dadurch anschließend neue Samples aus dieser Wahrscheinlichkeitsverteilung zu generieren. Man erhofft sich, den hohen Datenaufwand beim trainieren von ANN zu umgehen und durch GANS neue Trainingsdaten zu generieren. Ziel dieser Arbeit ist es das Konzept von GANs auf 3D-Scans von Tabakblättern anzuwenden um die Wahrscheinlichkeitsverteilung von 3D-Daten zu erlernen. Ein weiteres Ziel ist es herraus zu finden ob mit Hilfe von GANs es möglich ist 3D-Daten welche beispielsweiße beim Scannen unvollständig sind ergänzen zu können. Im Folgenden wird auf die Theorie von GANs eingegangen, wie deren Aufbau und deren zugrunde liegendes mathematische Modell. Anschließend werden die Methoden des Experimentes präsentiert sowie die Ergebnisse diskutiert.

## 1 Einleitung

Um genaurer Prognossen über Erntemenge und frühzeitiger Erkennung von Krankheiten zu treffen werden 3D-Scan verfahren eingesetzt welche Pflanzen scannen und damit 3D-Daten liefern welche zur weiterverarbeitung von Machine-Learning Ansätzen benötigt werden. [EVLT BILD EINBAUEN UND GENAUER AUF SCANVERFAHRENE INGEHEN]. Jedoch sind diese Scanverfahren wie jede Informationsübertragung mit sogenannten rauschen verbunden das heißt die Information vom Empfänger zum Sender wird verändert und behält nicht die ursprünglichen Inhalt. Beispiele dafür waren ein Blatt verdeckt ein anderes Blatt und lässt die Sensoren des Scanner nicht zu das unter trunder liegende Blatt zu erfassen. Um nun nun diesen Verlust von Daten zu verhindert muss geprüft werden ob die Möglichkeit besteht diese Daten zu reparieren in dem sie ergänzt werden. In dieser Arbeit wird ein Ansatz überprüft welcher es Möglich machen kann dieses Verhalten zu erhalten. Ein Ansatz der dafür Verwendet werden kann ist Deep Learning bei dem man das Deep Learning Modell den unterschied zwischen Reparierten und nicht reparierten Daten erlernt kann es rückschlüsse zeihen und selber Daten reparieren.

Deep learning, however, gained much popularity across many academic disciplines in recent years and has been used in computer vision successfully to

produce state-of-the-art results for various tasks. It is described as the application of multiple processing layers to produce multiple levels of representation. It is therefore capable of learning higher level representations of raw data, that can be used for the intended task, e.g. classification of an image. The most common realization are artificial neural networks (ANN) and especially convolutional neural networks (CNN) for processing data with a grid-like topology, e.g. images. Several publications have shown the effectiveness of CNNs for instance segmentation of plant leaves on images (e.g. Ren and Zemel, 2016). But there is a lack of research in applying deep neural networks to 3D representations of plants.

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text Daten zu bestimmte Klassen zuordnen. Der Grund für das steigende 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 ANN 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. Es lernt eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann und somit kann das Modell auch für discriminativen 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 diesem Paper wird speziel 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. 2017 wurden alleine 227 neue Paper zu diesem Thema vorgestellt. Ein Grund weshalb GAN an Popularität gewinnt ist der, dass neuronale Netzwerke mit Zunahme der Trainingsdatenanzahl eine Erhöhung der Performance für die sie Trainiert werden zeigen. Was bedeutet,dass sich mit Zunahme der Datenanzahl die Chance auf eine bessere Performance der Neuronalen Netzwerke ergibt (Halevy, Norvig, & Pereira, 2009). An diesem Punkt erhofft man sich durch GANS mehr qualitative Daten zu erzeugen und somit das Trainingsergebnis zu discriminativen Modelle zu verbessern.

### 1.1 Objectives

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text Daten zu bestimmte Klassen zuordnen. Der Grund für das steigende Interesse liegt in der niedrigen Fehlerrate bei discriminativen Aufgaben, im Vergleich zu anderen Maschine Learning Ansätzen, wie Desicion Trees oder Markov Chains(Goodfellow et al., 2016). Die Modelle lernen eine Funktion welche Input Daten X auf ein Klassen Label Y abbildet. Die Modelle werden dabei von ANN

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. Es lernt eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann und somit kann das Modell auch für discriminativen 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 diesem Paper wird speziel 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. 2017 wurden alleine 227 neue Paper zu diesem Thema vorgestellt. Ein Grund weshalb GAN an Popularität gewinnt ist der, dass neuronale Netzwerke mit Zunahme der Trainingsdatenanzahl eine Erhöhung der Performance für die sie Trainiert werden zeigen. Was bedeutet,dass sich mit Zunahme der Datenanzahl die Chance auf eine bessere Performance der Neuronalen Netzwerke ergibt (Halevy et al., 2009). An diesem Punkt erhofft man sich durch GANS mehr qualitative Daten zu erzeugen und somit das Trainingsergebnis zu discriminativen Modelle zu verbessern.

### 1.2 Outline

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

Kapitel 3 - stellt die Arbeit an sich vor. Welche Methoden gewählt wurden ob diese zu Entwickeln und Auszuführen. Die Trainingsdaten werden vorgestellt und Modelle aufgezeigt.

Kapitel 4 - Gibts ausführliche Angaben über die Ergebnisse von den Experimenten und evaluiert sie für ihre Aussagekraft.

Kapitel 5 - Gibt eine Zusammenfassung über die Arbeit und wird sie kritisch Diskutieren. Desweiteren wird ein Ausblick erstellt 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 die 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 den Grundstein für das in dieser Arbeit vorgestellte verfahren zur 3D-Datenrekonstruktion von 3D-Modellen aus Tabakblättern.

### 2.1 Künstliche Neuronale Netzwerke

Künstliche Neuronale Netzwerke(KNN) ist eine Datenstruktur welche von biologischen neuronalen Netzen wie sie bei Lebewesen vorkommen inspiriert sind. KNN haben das Ziel ein Funktion f* zu approximieren. Dabei werden Parameter $\Theta$ eines Models so angepasst, um die Abbildung von $y = f(x;\Theta)$ 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 einen Optimierungsproblem. KNN können aus mehreren Schichten sogenannten Hidden Layer n besteht 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 welche den Input in das Netzwerk aufnimmt(Goodfellow et al., 2016). Ein ANN mit mehr als nur 1 Layer und wird dies 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.

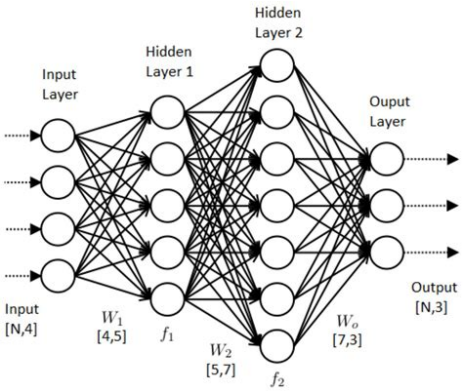


**Abb. 1.** Künstliches Neuronales Netzwerk

Jeder Layer besteht aus künstliche Neuronen diese haben ihren Namensgebung von aus der Natur stammende Neuron in Gehirnen von Lebewesen. Neuronen sind die Bausteine aus denen die Gehirne von Lebewesen, wie Fische, Vögel, Säugetiere zusammen gesetzt sind. Neuronen oder auch Nervenzellen bestehen in eine Zellkern der Zentrum der Zelle um sie herum sind Dendriten. Neuronen sind untereinander mit den Axon verbunden, diese haben an den enden Synapsen welche die grenze von Axon zur Nervenzelle einen Spalt bilden. Dieser Spalt kann überwunden werden indem von der Synapse Botenstoffe abgesendet werden welche sich dann an Rezeptor der gegenüberliegenden Synapse anhaften. Diese Übertragung findet statt wenn an der Synapse ein bestimmter Schwellenwert überschritten hat wurde von elektrischen reizen welche die Zelle abfeuert lässt. Künstliche Neuronen haben diese Schwellenwert durch sogenannte Gewichte $w_{ij}$ diese sind auf den Verbindungen zwischen den Neuronen in den unterschiedlichen Layern im ANN.Dabei steht i für die Layersposition im Netzwerk und j für das jeweilige Neuron im Netzwerk.

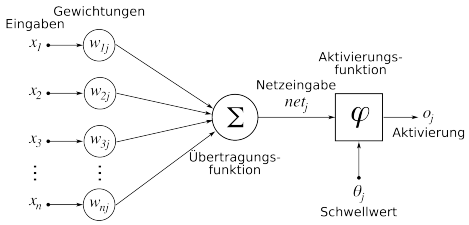


**Abb. 2.** künstliches Neuron

Jeder Layer eines KNN besteht aus mehreren Neuronen. Ein Neuron $\theta$ kann mehrere Input $X_n$ erhalten und produziert einen Output $\omega$Die Berechnungen welche von den Neuron durchgeführt wird ist 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 $\lambda$ gegeben. Ein Neuron ist definiert durch:

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

Die Aufgabe der Aktivierungsfunktion $\lambda$ ist es eine nicht liniere 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 in Abb.11 entnommen werden. Diese veranschaulichen den Wertebereich welcher von den Aktivierungsfunktion angenommen werden kann und ihren Verlauf.

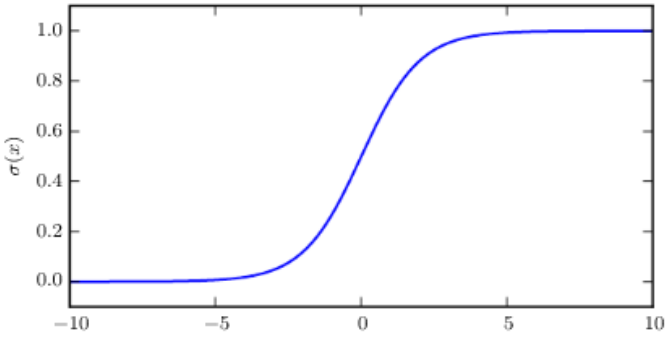


**Abb. 3.** Sigmoid Funktion

### 2.1.1 Ziel-Funktion

Die Ziel-Funktion oder auch Loss-Funktion genannt J($\theta$) muss Differenzierbar sein. Das heißt unseren Funktion f: D $\rightarrow$ R ist differenzierbar an der Stelle $x_0$. Wenn nun $f'$: x $\rightarrow$ f(x) an jeden Punkt $x^n$ ableitbar ist, ist f differenzierbar. Die Aufgabe der Ziel-Funktion ist es zu messen wie gut unsere Model $\theta$ f*(x) approximiert. Man verwendet auch gerne den Begriff Kosten, also wie viel Kosten unser Modell erzeugt bei den lösen der zugewiesenen Aufgabe. Die Wahl welche Funktion gewählt wird ergibt sich aus der Aufgabe des KNN. Diese kann beispielsweise für Klassifikation oder Regression Aufgaben hergenommen werden. Bei Regression soll eine kontinuierlicher Variablen von $\Theta$ als Output generiert werden wohin gegen bei Klassifikationsproblemen der Output Klassen-Labels darstellen(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:

$$\hat{q}(c \mid x) = \arg\min_{q(c|x)} \{-\sum_{n} \log q(c_n \mid x_n)\}$$

Wobei $x_n$:n=1,...,N die Trainingsdaten sind ist und $c_n$:n=1,...,N die mögliche 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 werden dieser zählt zu den Optimierungs Algorithmen für KNN. Er konnte zeigen das dieser Algorithmus schneller und effizienter auf Neuronalen Netzwerken arbeitet als andere Optimierungsalgorithmen vor ihm. Das mathematische Zugrundeliegende Konzept ist ein Optimierungsproblem der die Partitiellen Ableitung von $\frac{\partial Z}{\partial w}$, wobei Z 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 an Punkt x ist man nun in der Lage einer Änderung von der Ableitung x dahin gehen 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. Das Verfahren wird in 4 veranschaulicht.

Wenn nun $f'$(x) = 0 haben wir keinerlei Information über die Steigung erreicht. Dies ist aber kein Indiz dafür sein das $f$ ein Optimum erreicht hat. Es könnten wie in Abbildung 5 unten dargestellt ein lokales Minimum sein dass heißt das wir nur an diesen Punkt ein Minimum erreicht haben aber im Funktionsverlauf ein noch niedrigeres Minimum vorhanden ist. Oder einen Sattelpunkt welche

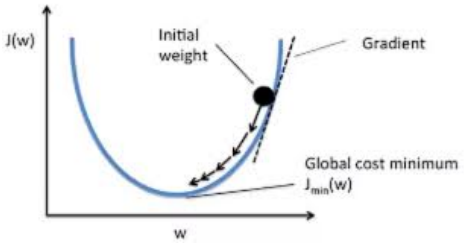


**Abb. 4.** LossFunktion

ein Übergang zu einen anstieg der Funktion bildet. Ist nun die die Funktion $f$ definiert als $f\colon \mathbb{R}^{\mathrm{n}} \to \mathbb{R}$ und hat als Input mehrere Variablen. Die partielle Ableitung $\frac{\partial f(x)}{\partial x_{\mathrm{i}}}$ zeigt an wie sehr sich $f(x)$ ändert wenn wir $\mathrm{x_i}$ ändern. Der Gradient $\nabla_{\mathrm{x}} f(x)$ ist ein Vektor welche alle partiellen Ableitungen von $f$ enthält. Nun kann man f optimieren in dem man in die Richtung des Gradienten geht welche negativ ist. Dieses Verfahren wird Gradient Descent genannt (Goodfellow et al., 2016).

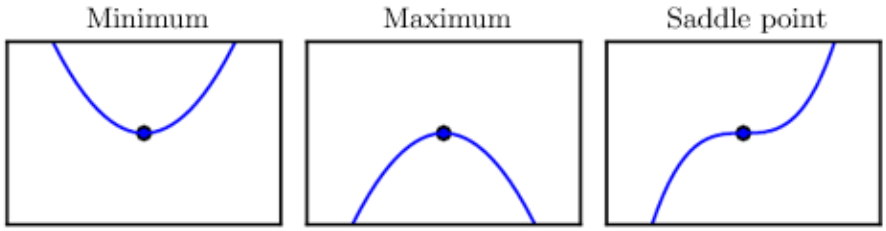


**Abb. 5.** Minimum Maximum Saddle Point

### 2.1.3 Momentum

Momentum hat das Ziel das Gradientenverfahren des Backpropagation Algorithmus 2.1.2 zu beschleunigen um eine effizienter Ergebnis herbei zu führen und ein schnelleres Lernen erzielt zu erreichen. Definiert ist dieses 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 und $\mu \in [0,1]$ das Momentum ist und $\nabla \mathrm{f}(\theta_{\mathrm{t}}$ der Gradient von $\theta_{\mathrm{t}}$ ist. Je größter das Momentum desto schneller bewegt sich der

Gradient abwärts. Da am Anfang eines Lernphase der Gradient üblicherweise hoch ist empfiehlt sich zunächst mit einen niedrigen Momentum zu Arbeiten da sonnst die Gefahr besteht über das globale Optimum hinüber zuschießen. Wenn nun das Training stagniert welches aus Gründe der Aufbau der Loss-Funktion 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 je länger das Training voran schreitet. Und sich somit schneller zu einen globalen Optimum zu bewegen oder aber aus einen lokalen Optimum hinaus Richtung eines globalen Optimum(Sutskever, Martens, Dahl, & Hinton, 2013).

### 2.1.4 Regularization

Ein Problem was alle Machine Learning Anwendungen Teilen ist das Problem wenn der Trainingsalgorithmus eine gutes Trainingsergebnis erzielt aber dann auf den Testdatensatz ein schlechtes Ergebnis erlangt dies wird Overfitting genannt. Es gibt einige Möglichkeiten welche eingesetzt werden können welche dann aber auch das Trainingsergebnis verschlechtern. Man spricht davon das diese Versuchen zu Regularizieren.

**Data Argumentation**

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

**Dropout**

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

**L2-Regularization**

Dabei wird das Gewicht des Neurons während des Trainings nähe zu seinen Ursprung, dass heißt den Wert seiner Initialisierung gedrängt. Dies passiert indem ein Regularisierungswert $\Omega = \frac{1}{2}||w||\frac{1}{2}$ an den Zielfunktion gehängt wird. Dieses sorgt dafür das während dem Training der Gewichts Vektor welche durch das Gradienten Verfahren ermittelt wird schrumpft dieses sorgt für eine höhere Varianz während des Trainings was wiederum das ANN dazu

zwingt mehr zu generalisieren(Goodfellow et al., 2016).

### 2.1.5 Batch Normalisation

Wie in Kapitel 2.1.2 gezeigt das stochastischen Gradienten Verfahren Vorteile gegenüber des normalen Gradienten Ermittlung 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 das 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 $\hat{x}_{iB} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}}$
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 Input von Mini-Batch B berechnet und In Schritt 3 die Varianze. In Schritt 3 wird nun der der normalisierte $x_i$ berechnet welche dann mit denn $\beta$ und $\gamma$ multipliziert werden. Diese Werte sind neue Gewichte im Neurnonalen Netzwerk welche während des Trainingsprozesses angepasst werden. $\epsilon$ 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 den der Minibatch des Trainings ersetzt wird.

**Algorithm 2:** Training mit Batch-Normalisierungs Netzwerk

1 Input: Netzwerk N mit trainierbaren Parameter $\theta$;$\{x^{(k)}\}$k=1
2 Output: Batchnormalisiertes Netzwerk
3
4
5
6
7
8
9
10
11
12

Schritt 1 bis 5 des Algorithmus 2 baut das ANN durch die Transformationen aus Algorithmus 1 um. In Schritt 6 und 7 geht es darum die Parameter $\gamma$ und $\beta$ zu trainieren. Dieses passiert mit den üblichen Backpropagation Algorithmus wären des allgemeinen Training des Netzwerkes.

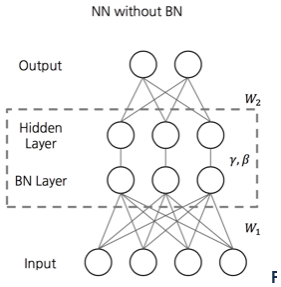


**Abb. 6.** Neuronales Netzwerk mit Batch Normalication Layer

### 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 diese wird Vertex List genannt. 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

Diese kann 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.

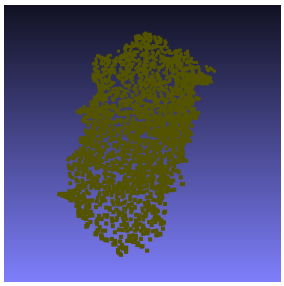

**Abb. 8.** Punktwolke eines Tabakblattes

### 2.2.1 Datenaufnahme von Tabakpflanzen

Die Daten stammen von TERRA-REF Feld Scanner von der University von Arizona Maricopa Agricultural Center and USD Arid Land Researh Station in Maricopa. Es ist der größe Feld crop scnanner. In Abb. **??** ist dieser abgebildet. In Abb. 9 ist der Scankopf des TERRA-REF dargestellt mit den unterschiedlichen Scanmöglichkeiten für das du scannende Objekt.

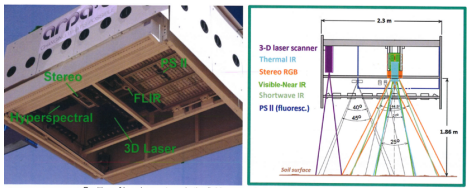


**Abb. 9.** Scankopf

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. Durch die Sammlung eizelner Scanpunkte entsteht eine 3D-Repräsentation eines Gegenstandes.

### 2.2.2 Das Problem mit 3D-Data bei Machine Learning Ansätzen

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

Wie im Kapitel "Woher stammen die daten "gezeigt, sind Punktwolken als Menge gespeichert in denen keine Relation untereinander besteht, dass heißt es ist für den Punktwolkencompiler nicht von Relevanz auf welchen Platz die einzelnen Punkte abgespeichert werden, die Punktwolke wird egal in welcher Permutation der einzelnen Punkte abgespeichert, immer gleich angezeigt. Vergleicht man nun ein Bild mit 512 Pixeln und 3 RGB-Farbkanälen ist einer Dimension von 391680 erreicht. Vergleichen wir nun das mit einer Punktwolke in einen 125 cm $_3$ großen Bereich. Da die einzelen Koordianten eines Punktes als Rationale Zahlen dargestellt wird und rationalen Zahlen abzählbar Unendlich ist unser 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 ist, dass die aus Kapitel Convolutional Neural Network beschrieben Convolutionel-Layer einen großen Beitrag bei den 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 diese Tools bei Punktwolken einzusetzen(Achlioptas, Diamanti, Mitliagkas, & Guibas, 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 Convolution oder auch Kernels genannt, zusammengesetzt ist. Ein Aufbau kann aus Abbildung 10 entnommen werden(Goodfellow et al., 2016).

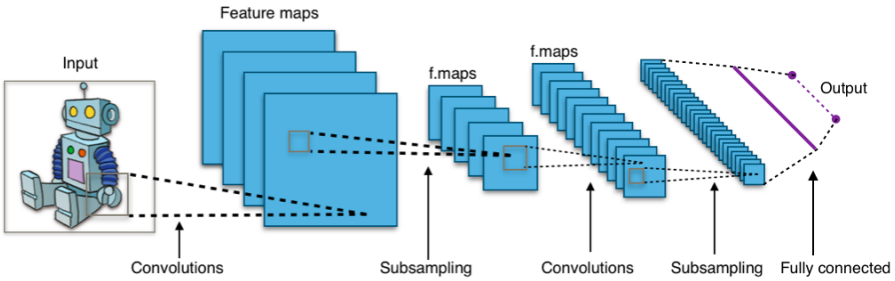


**Abb. 10.** Convolutional Neural Network

Die Kernels, also die einzelnen Filter, von den jeder der N-Layer k besitzt sind $K^{n \cdot n}$ Matrizen jedes $k_{ij}$ in einem Filter einspricht einen aus der üblichen Neuronalen Netzwerk Architektur bekannten Gewichte. Diese Gewichte werden dann durch den Backprobagation-Algorithmus in der Trainingsphase des Netzwerkes angepasst um den Verlust der Loss-Funktion durch bestimmten des Gradianten zu minimieren. Das durch die Abbildung 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 Kernel Größe und den Input des jeweiligen Kernel sowie des Strides. Die Output Größ W kann berechnet werden durch W = (W-F+2P)/s+1. Wobei F für die Größe des Kernel steht (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. Würde 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 ver-

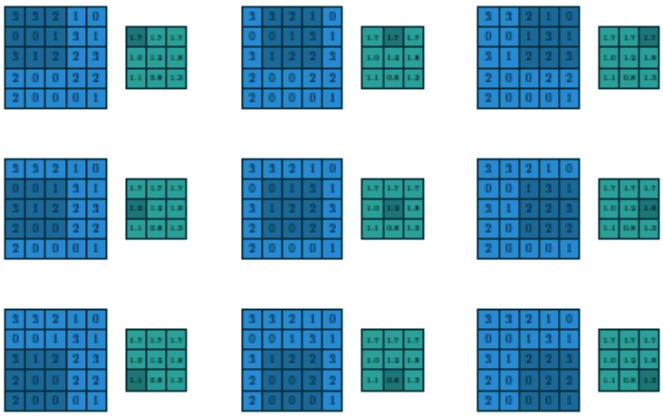

**Abb. 11.** Convolution Beispiel

wendet 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).

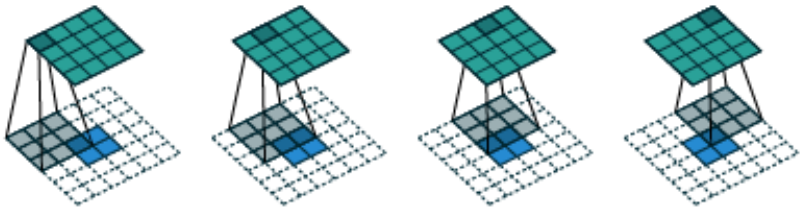

**Abb. 12.** Deconvolution Beispiel

### 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 und 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 ein 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 werden die Dimension der Daten so reduziert und Informationen bei zu behalten welche als relevant gelten. Diese Technik finden auch in anderen Machine Learning Anwendung wie in beispielsweise der Principale component anaysis(PCA) Anwendung(Hinton & Salakhutdinov, 2006). Ein Autoencoder besteht aus 2 Bestandteilen. Erstens ein Encoder e parametrisiert mit $\phi$ welcher einen Input x $\in$ von $x^{\mathrm{i}}$ wo x ein Vektor der Länge i ist und damit die Input Dimension bestimmt. Dieser wird durch den Encoder auf einen Vektor $z^{\mathrm{k}}$ abgebildet wobei k¡i ist. Zweitens der Decoder d parametrisiert durch $\theta$ bekommt als Input $z^{\mathrm{k}}$ und bildet z auf $x^{\mathrm{l}}$ ab wobei l = i. Und somit die gleiche Dimension wie der Input. Aufgabe ist es nun das der Encoder den Input z so gut komprimiert das der Encoder es schafft das x $\approx$ z. Eine grafische Darstellung kann aus Abb. 13 entnommen werden(Goodfellow et al., 2016).

Die Parameter $\phi$ und $\theta$ werden durch den in Kapitel 2.1.2 vorgestellten Algorithmus trainiert und erlernt. Und 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.

Bei Punktwolken als Datentyp erzielt die Cross-Entropy-Funktion keine gute Ergebnisse da diese invariant zu ihrer Permutation sind. Dies Da in der Arbeit der Input 3d Pointcloud sind und diese Sets invariant zu ihren Permutationen sind Das heißt ändere ich die Anordnung meiner einzelnen Punkte in mein Set bleibt das Ergebnis unverändert. Kann auf übliche Zielfunktionen welche für strukturierte Daten wie Bilder verwendet werden nicht zurück gegriffen. Das Problem dabei besteht das zwischen zwei unterschiedliche Sets von Punkten heraus

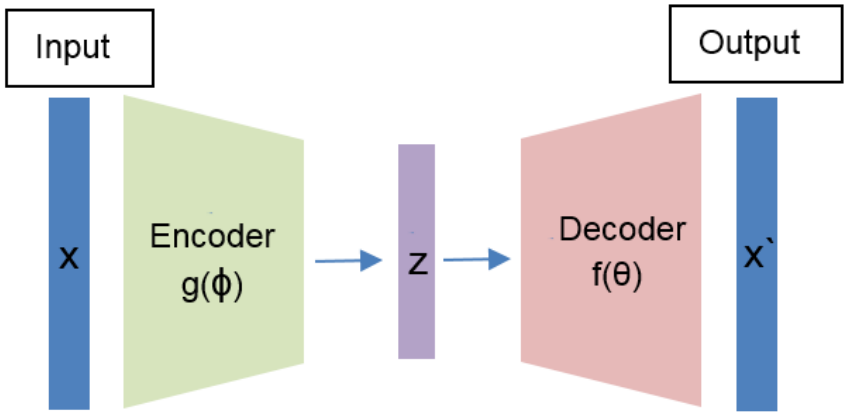


**Abb. 13.** Autoencoder

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:

$$d^{\text{CD}}(X_1, X_2) = \sum_{x \in X_1} \min_{y \in X_2} ||x - y|| + \sum_{y \in X_2} \min_{x \in X_2} ||x - y||$$

Der Unterschied ergibt sich zwischen den beiden das 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).

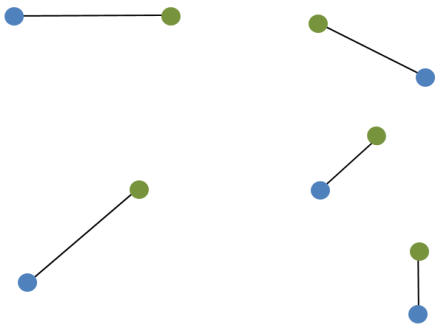

**Abb. 14.** Earth Mover Distance

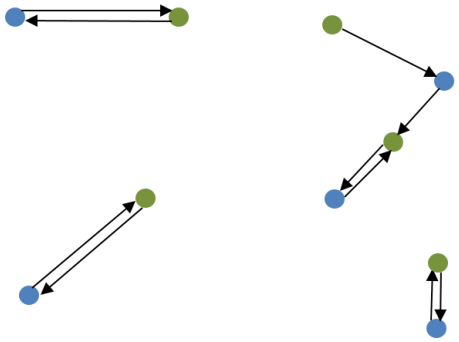

**Abb. 15.** Chamfer Distance

### 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 der Maximum Likelihood Schätzverfahren(ML-Schätzer) in dem die Parameter $\theta$ 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 $\Phi_g$ 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.

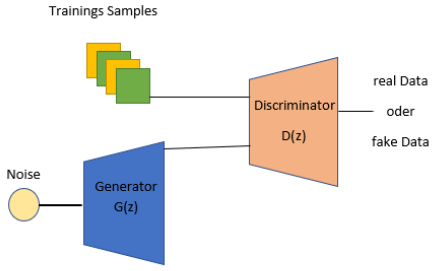


**Abb. 16.** Generativ Adverserial Network

Der Discriminator ist definiert durch $D(x;\Phi_d)$. Wobei $\Phi_d$ 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 $\Phi_d$ so angepasst, dass die Wahrscheinlichkeit Trainingsbeispiele richtig zu klassifizieren maximiert wird. Und $\Phi_g$ 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 3:** 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).

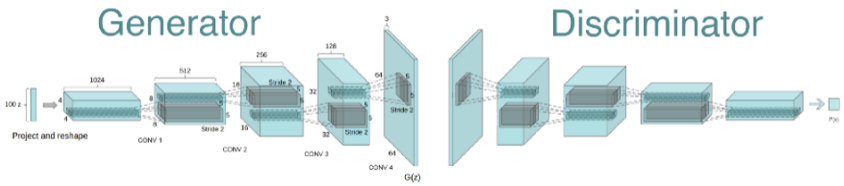


**Abb. 17.** Deep Convolutional GAN

Wie unter Generativen Modellen gezeigt wurde kann 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).

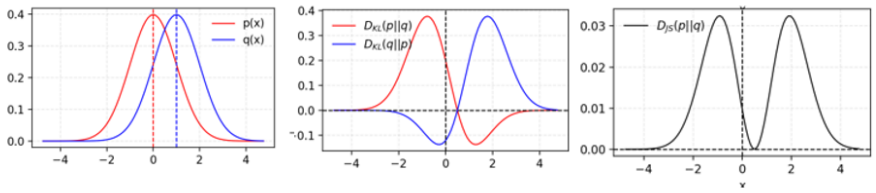


**Abb. 18.** KL Divergenz und JS Divergenz

### 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 $\gamma$ 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 $P_\theta$ 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. Dem entsprechend muss die Zielfunktion dahin gehen angepasst werden bedingte Wahrscheinlichkeiten zu lernen

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

Im Abb. 19 kann der Informationsfluss und die Konnektivät der einzelnen Module entnommen werden. Die Module Generator und Discriminator bleiben gleich und können von ihren Aufbau für die jeweiligen Datentyp verändert werden und Beispielsweiße durch Convolutional-Layer, Deconvolutional-Layer oder Fully-Connected-Layer bestehen.(Radford et al., 2015).

### 2.7 3D-GAN

Das besondere an den 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.

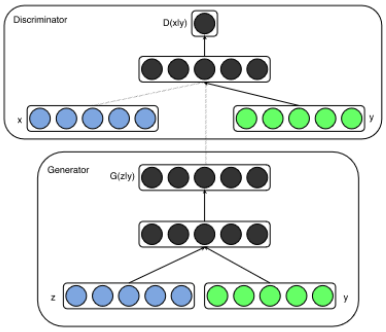


**Abb. 19.** Conditional Adverserial Network

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 oder 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 in dieser Arbeit als 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 Gausischen 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).

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(siehe Autoencoder) trainiert. Der Aufbau ist der selbe einen üblichen 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 wird nun meine Trainingsdaten durch den vorher trainierten Encoder des 3D-Autoencoder auf die festgelegte Output-Dimension komprimiert. Anschließend werden diese komprimierten

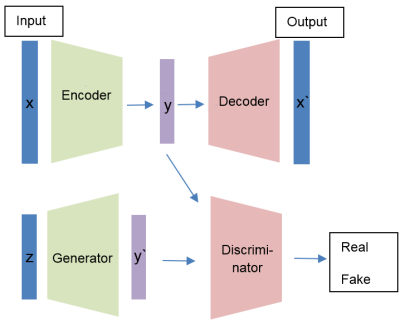


**Abb. 20.** Latent-GAN

Daten verwendet um das GAN zu trainierten. Dabei erlernt das GAN komprimierte Code zu produzieren welche anschließend durch den Decoder des 3D-Autoencoder wieder auf die ursprüngliche Größe gebracht werden kann. Durch dieses Verfahren war es ihnen möglich Daten in einer guten Qualität zu produzieren und eine Datenverteilung des zugrundeliegenden Models zu erlernen. Der gesamte Ablauf kann in 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 Texte. Bei der Rekonstruktion geht es darum das Daten welche von ihren Urzustand verändert wurden, sei es durch Artefakte oder manuelle Bearbeitung wieder dahin zurück zu führen. In Abb. 21 ist eine Rekonstruktion eines Bildes dargestellt welches eine Wiederherstellung eines Hundes zeigt. Deep Learning hat in diesen Bereich besonders bei Bilder große Erfolge erlangt, dadurch das diese als Matrizen dargestellt werden können liefern sie eine Datenstruktur auf welche KNN arbeiten können. Auch durch Bearbeitung mit Convolutionen-Layer auf Bilddaten zählt als Erfolg für die Weiterverarbeitung(Dong, Loy, He, & Tang, 2016a). Durch die genannten Methoden werden bessere Strukturen für das jeweilige Ziel, in diesem Fall der Rekonstruktion gelernt und ermöglicht es wie in verschieden Papern wie (Dong, Loy, He, & Tang, 2016b) welche auf super-hochauflösenden Bildern Artefakte entfernt und das Bild wieder zum Urzustand zurück führt.

Ebenso wie die Arbeit von Liu, Gulin und Reda (Liu et al., 2018) welche ähnliche Ergebnisse liefern konnte. 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. Bei wurde mit Hilfe von Autoencoder gearbeitet welche auf für die folgenden Arbeit von Bedeutung sind.(Yi et al., 2017).

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

## 3 Methoden

In diesen Kapitel werden die Methoden für die Versuchsaufbauten aufgezeigt welche die Ziele 1 und 2 überprüfen sollen. Dabei geht es darum ob mit hilfe von GAN 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. Dieses Verfahren wurde in Kapitel 2.8 beschrieben. Die Trainingsdaten für diesen Versuchsaufbau können aus Kapitel 3.2 entnommen werden. Alle Versuche 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. Also 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).

### 3.1 Datensatz 1. Versuchsaufbau

Der erste Datensatz SStühle"besteht aus 4014 Punktwolken mit je 2056 Punkten. Der Datensatz wurde aus von den Shapenet Datensatz(www.shapenet.org) entnommen dieser ist als .ply Datenformat abgespeichert. Ein Beispielpunktwolke kann aus Abbildung entnommen werden. Dieser Datensatz wurde von in der Arbeit von Achlioptas, Diamanti, Mitliagkas und Guibas(Achlioptas et al., 2018) verwendet welches auch in Kapitel 2.7 vorgestellt wurde ist, dieser dient als Validierungsdatensatz um die Qualität der generierten Daten des GAN welches mit Tabakblättern trainiert wurde mit den welches die Stuhldaten bekommt zu vergleichen.

Die Daten für den zweiten Datensatz "Blätterßtammen 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 420 Punktwolken welche dann durch ein Punktreduktionsverfahren auf jeweils 2056 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

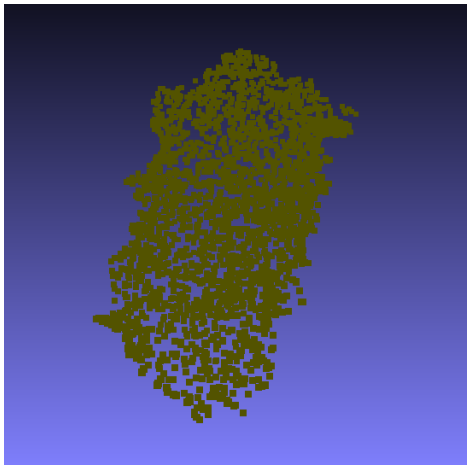

**Abb. 23.** 3D Punktwolke einer Tabakpflanze

### 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 420 unterschiedlichen Blättern. Welche nun durch einen Algorithmus dahin gehen verändert werden um dann Simulieren von verdecken des Blattes durch andere Objekte zu beschädigen. Dies Simuliert das Verdecken das Blattes durch Artefakte. Dabei werden 3D-Spheren erzeugt wie in Abb. **??** Rot dargestellt. Die Spheren bestehen aus 100 000 Punkte welche alle einen maximalen Radius von 10 mm haben und welcher jeder Punkt aus einer Normalverteilung entnommen wird um eine gleichmäßige Verteilung der Punkte zu gewährleisten. Die Spheren werden nun in euklidischen Raum bewegt das sich 46 unterschiedlichen Spheren ergeben diese sollen von unseren 420 Blättern mögliche Differenzmengen Bildern um Schnittmengen mit möglichst allen Bereich des Blattes zu haben. In Abb 24 sind alle Sphären symbolisch eingefügt werden um dieses Vorgehen zu Veranschaulichen. Dabei wird die Differenzmenge der beiden Punkte genommen wobei je Punkt der Euklidische Raum in einen Radius von 30 mm nach Nachbarn gesucht wird welche dann als Differenzmenge gelten. 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 ist. Was letztendlich zu 2047 zerstörten Blättern geführt hat mit jeweils mit 2056 Punkten. In Abbildung 26 können Beispieltrainingsdatensätze entnommen werden.

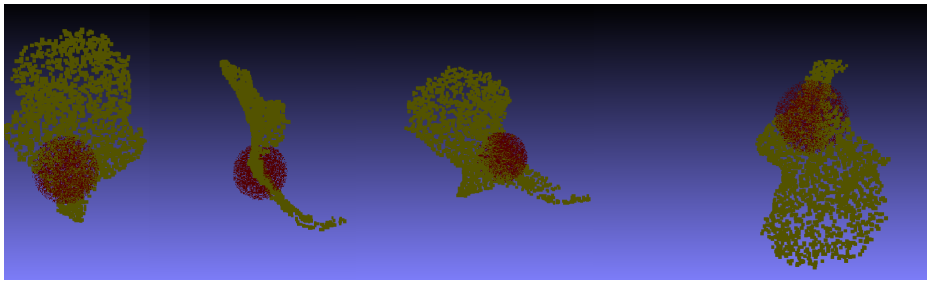

**Abb. 24.** 3D Punktwolke einer Tabakpflanze

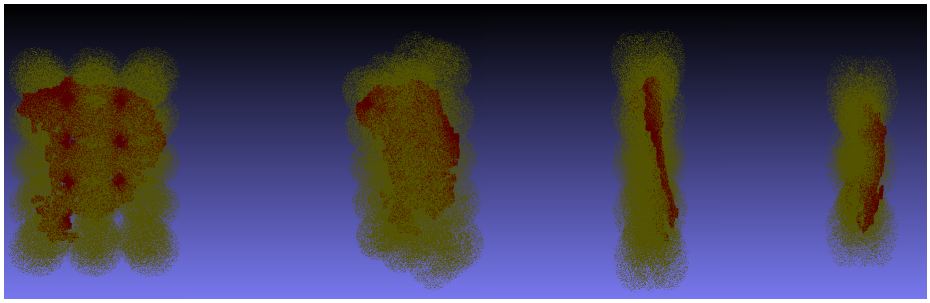

**Abb. 25.** 3D Punktwolke einer Tabakpflanze

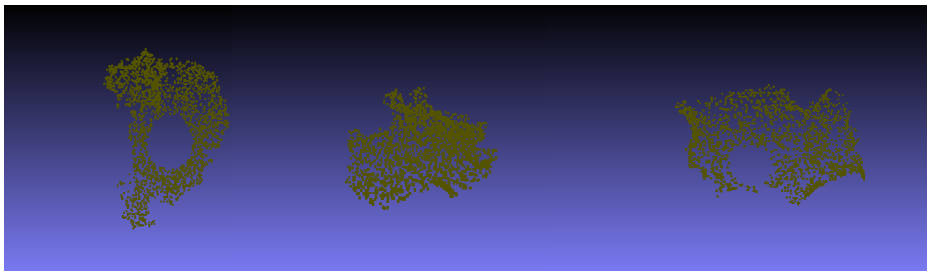

**Abb. 26.** 3D Punktwolke einer Tabakpflanze

### 3.3 Trainingsaufbau 1 - GAN

Der Trainingsaufaufbau besteht aus zwei unterschiedlichen Versuchen. Aufbau 1.1 ist das RAW-GAN. Dabei wird auf den herkömmlichen Aufbau von GAN zurückgegriffen. Die Algemeinen Meta-Trainingsvariablen sind Learningrate mit 0,005 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 beträgt 64. Der Discriminator besteht aus 4-Layern welche 1-Dimensionaler Convolutional Layer bestehen[64, 128, 256, 256, 512]. Mit einer Kernel Größe von 1 und Stride von 1. Die Aktivierungsfunktion wird die ReLu-Funktion genutzt. Darauf folgen 3 fully-connected-Layer mit der Größe [128, 64, 1] alle mit einer Relu-Aktivierungsfunktion. Dropout and Batchnormalisation wird auch angewendet. Das Training erfolgt jeweils mit den den Datensätze SStuhlünd Tabakblätter"beschrieben in Kapitel 3.1.

Der Generator besteht auf 5 fully-connected Layern mit [64,128,512,1024,1536] Neuronen jeweils mit einer Relu Aktivationsfunktion. Der Noisevektor z wird aus einer 128-D Gaussischen Verteilung entnommen mit $\mu = 0$ und $\sigma = 0.2$. Der Aufbau kann in Abbildung entnommen werden. Die Zielfunktion ist die Wasserstein Metric und die Vanila-GAN Zielfunktion von Goodfellow verwendet(Goodfellow et al., 2014). Das Training erfolgt jeweils mit den den Datensätze SStuhlünd Tabakblätter"beschrieben in Kapitel 3.1 und wurde auf 500 Epochen durchgeführt.

Der Versuchsaufbau 1.2 ist das Latent-GAN welches kurz in Kapitel 2.7 vorgestellt wurde. Es wurde die Implementierung von (Achlioptas et al., 2018) benutzt(www.github/3d) Zunächst wird dabei der Autoencoder mit den Trainingsdaten trainiert. Der Encoder des Autoencoders wird mit einer Learning Rate von 0.0005 trainiert und einer Batchgröße von 50. Der Encoder besteht dabei besteht dabei aus 4 1-D Convolutional Layern mit 64,128,246,1024] Filtern Stride von 1 und Size von 1. Die Activierungsfunktion ist relu. Der Letzte Layer ist ein Max Layer. Der Decoder besteht aus aus [256,256,[614]. Das Training erfolgt jeweils mit den den Datensätze SStuhlünd Tabakblätter"beschrieben in Kapitel 3.1 und wurde auf 500 Epochen durchgeführt.

Das GAN hat einen Generator mit [128,128] fully-connected Layer welche als Aktivierungsfunktion eine ReLu-Funktion benutzen. Der Noisevektor z wird aus einer 128-D Gaussischen Verteilung entnommen mit $\mu = 0$ und $\sigma = 0.2$. Der Discriminator besteht aus [128,64,1] Fully-Connected Layern welche ebenfalls ReLu-Funktion nutzen.

### 3.4 Trainingsaufbau 2 - CGAN für Punktwolkenrekunstroktion

Da sich im Trainingsaufbau 1 das L-GAN bessere Ergebnisse liefert bei erlernen von Punktwolken Daten. Wird der Aufbau von L-GAN übernommen und dahin gehen verändert, das Ziel zu erfüllen. Zunächst werden wie bei Trainingsaufbau 1 die Trainingsdaten(siehe Trainingsdaten Aufbau 2) trainiert. Der Aufbau des Autoencoder ist dabei gleich dem von Aufbau 1. Der Encoder des Autoencoders wird mit einer Learning Rate von 0.0005 trainiert. und einer Batch Größe von 50. Der Encoder besteht dabei besteht dabei aus 4 1-D Convolutional Layern mit 64,128,246,1024] Filtern Stride von 1 und Size von 1. Die Activierungsfunktion ist relu. Der Letzte Layer ist ein Max Layer. Der Decoder besteht aus aus [256,256,[614]. Die Zielfunktion ist die Champfer Disdance da sie die besten Ergebnisse bei citiere PGAN besten geliefert hat. Für den Autoencoder wurde die Implementierung von (Achlioptas et al., 2018) benutzt(www.github/3d).

Für das C-GAN wurde als Der Noisevektor z wird aus einer 128-D Gaussischen Verteilung entnommen mit $\mu = 0$ und $\sigma = 0.2$. y sind von Autoencoder codierten 128-D komprimierten ßerstörteTabakblätter das Netzwerk wird nun lernen zu den passenden komprimierten zerstörten Blättern die unzerstörten kodierten Blätter zu erlernen.

Anschließend werden zerstörten BlatDaten x mit den trainierten Encoder zu den latenten Code y komprimiert, der einen Vektor von 128-D entspricht. Das selbe wird mit den unzerstörten Blattdaten x‘gemacht welche mit den Encoder zu dem latenten Code y‘komprimiert werden welcher einen Vektor von 128-D entspricht. Wobei es jeweils (x,x‘) paare Gibt welche den

Der Aufbau des C-GANS entspricht der in im 3.4 beschrieben C-GAN üblichen Aufbau. der Generator bekommt als Input x’ und x welches einen 128-D Vektor welcher aus einer Gausischen Verteilung gesamplet wird. Der Generator besteht aus 3 layern [256,128,128] der Diskriminator besteht aus 2 fully Connectected Layer mit der Größe [128,128]
Bei generator und Diskriminator wird beim Trainieren jeweils Batchnormalization eingesetzt.

Als Zielfunktion wird wieder das W-GAN verwendet da es im Versuchsaufbau 1 die besseren Ergebnisse liefert.

## 4 Evaluation und Ergebnise

Es wird nun in diesen Kapitel auf die Ergebnisse von Versuchsaufbau 1 und 2 eingeganngen. Zunächst wird Versuchsaufbau 1 mit des erlernen von latenten Raum von Punktwolken gezeigt. Anschließend geht es um die Ergebnisse von Versuchsaufbau 2 für die Rekunstruntion von zerstörten Blättern.

### 4.1 Ergebnisse - Versuchsaufbau 1

Die Ergebnisse von RAW-GAN mit den Blattdaten welches nach 500 Epochen Training lassen sich in Abb.28 entnehmen. Wie zu sehen 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 Zeit keine Ähnlichkeit mit normalen Blättern. Der WGAN Loss für den Generator und Discriminator können in Abbildung 27 entnommen werden. 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.

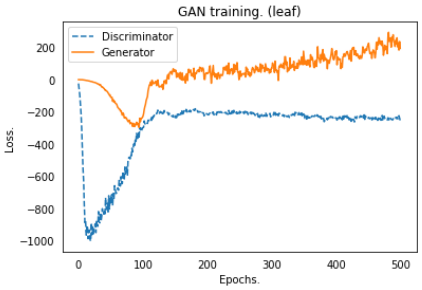


**Abb. 27.** Trainingsverlauf des RAW-GAN mit den Blattdaten

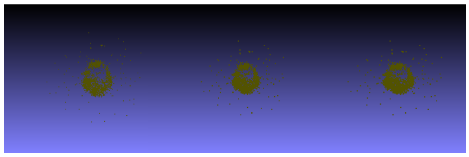

**Abb. 28.** Traingsverlauf des RAW-GAN mit den Blattdaten

Ähnliche Ergebnisse sind auch mit den Stuhldaten zu beobachten. Wobei die Qualität der Stühle mehr an Stühlen erinnert. Es zeichnen sich Lehne und Stuhlbeine von dem GAN trainierten Generator erzeugten Daten ab. Der Trainingsverlauf ist in Abb. 29 zu entnehmen. Dabei ist zu sehen das der Discriminator sich nicht mehr verbessert und der Generator keine besseren Ergebnisse liefern kann. Die Ergebnisse decksen sich mit den von (Achlioptas et al., 2018) festgestellten Ergebnissen in den das RAW-GAN keine gut aussehenden Stuhldaten erzeugen kann. Um nun die Ursache für die unterschiede in der Qualität auszumachen und festzustellen ob durch mehr Blattdaten bessere Ergebnisse erzeugt werden können, da es einen Qualitätiven Unterschied zwischen den beiden erzeugten Daten gibt. Wurden die Stuhldaten auf 422 Trainingsdaten reduziert und ein erneutes Training des RAW-GAN durchgeführt. Erzeugte Beispieldaten können aus Abb. **??** entnommen werden. Vergleicht man nun die Qualität von den RAW-GAN mit 42 Stuhldaten Trainingsdaten und 4014 Stuhldaten ist ei-

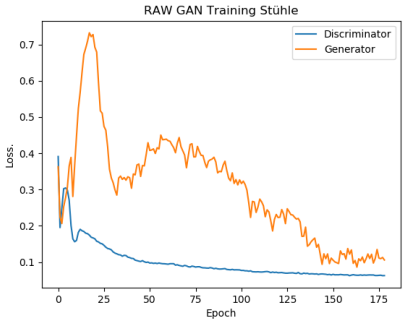


**Abb. 29.** Traingsverlauf des RAW-GAN mit den Stuhldaten

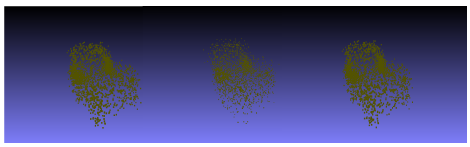

**Abb. 30.** Traingsverlauf des RAW-GAN mit den Stuhldaten

ne Steigerung der Qualität klar festzustellen. Der Trainingsverlauf in Abb. 32 ähnelt der von Abb **??**.

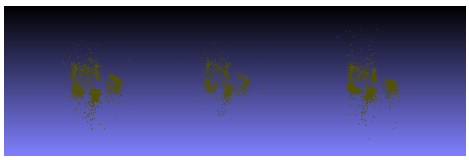

**Abb. 31.** Trainingsverlauf des RAW-GAN mit den 422 Stuhldaten

Die Ergebnisse des Versuchsaufbau 1.2 Latent-GAN für Stühle kann aus Abb. 33 und Trainingsverlauf von Generator 34

Das Latent-GAN welches mit den Blattdaten trainier wurde können exemplare welche vom generator erzeugt wurde sind aus Abb. 36 entnommen werden. Der Trainingsverlauf ist ins Abb. 36 zu entnehmen.

Um die Vorherige Ergebnisse besser zu Vergleichen und um ersichtlich zu machen ob eine erhöhung der Anzahl der Tabakblätter Daten zu einen besseren ergebnisse führen. Wurden 420 Stuhldaten genommen um das Latent-GAN zu trainieren. In Abb 37 sind die exemplarische erzeugte Daten von Generator zu entnehmen. Der Trainingsverlauf kann aus Abb. 37 entnommen werden.

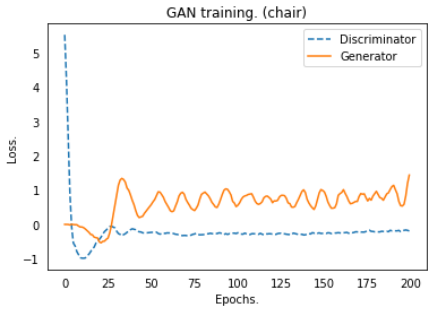


**Abb. 32.** Trainingsbeispiele des RAW-GAN mit den 422 Stuhldaten

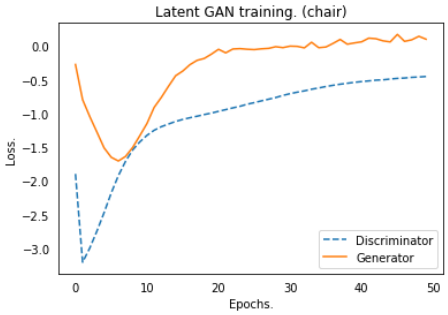


**Abb. 33.** Traingsverlauf des Latent-GAN mit den Stuhldaten

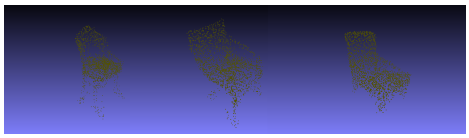

**Abb. 34.** Traingsverlauf des Latent-GAN mit den Stuhldaten

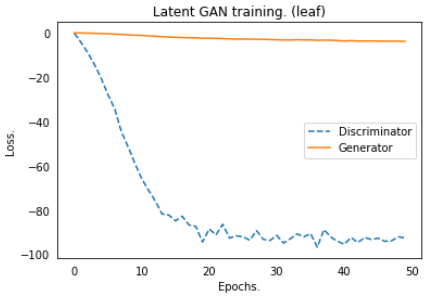


**Abb. 35.** Trainingsverlauf des Latent-GAN mit den Blattdaten

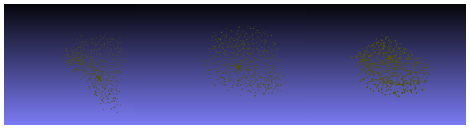

**Abb. 36.** Trainingsverlauf des Latent-GAN mit den Blattdaten

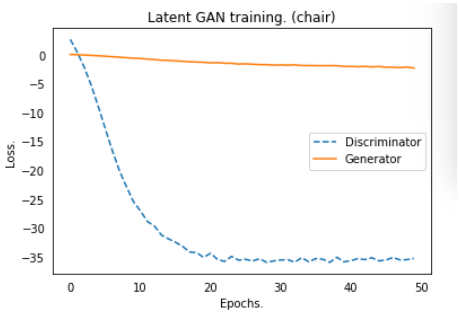


**Abb. 37.** Trainingsverlauf des Latent-GAN mit den Stuhldaten und 400 Trainingsdaten

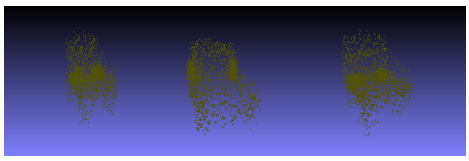

**Abb. 38.** Trainingsverlauf des Latent-GAN mit den Stuhldaten und 400 Trainingsdaten

### 4.2 Ergebnisse - Versuchsaufbau 2

Für Versuchsaufbau 2.1 wurde zunächst der Autoencoder mit den zerstörten Blattdaten auf 500 trainiert. Um eine komprimierte Version auf 128-D zu erlernen. Dabei wurde jeweils die in Kapitel 2.4 vorgestellte Chamfer Distanz als Distanzmass genommen. Beispiel Trainingsinput Trainingsdatensätze können Abb. 41 entnommen werden mit ihren den jeweiligen erzeugten Output des Decoders in Abb. 40 entnommen werden. Wie zu erkennen ist vervollständigt der Autoencoder mit der Chamfer Zielfunktion die Blätter selbstständig obwohl das nicht als Ziel in diesen Bearbeitungsschritt herbeizuführen ist. Es ist zu erkennen das die Dichte in welche die Punkte beim Output auf der Höhe der Löcher angeordnet ist stark von den üblichen Bereichen auf den Blatt ist. Auch lässt der Trainingsverlauf in Abb. 39 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 Distanz 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 0.0005 cm Bereich liegt wie in Abb.39 in Epoche 500 abzulesen ist. Der keine Vernünftigte Trainingsdaten für den Latenten C-GAN aufbau generiert werden können kann an dieser Stelle der Versuchsaufbau 2.1 Latent C-GAN nicht weiter geführt werden.

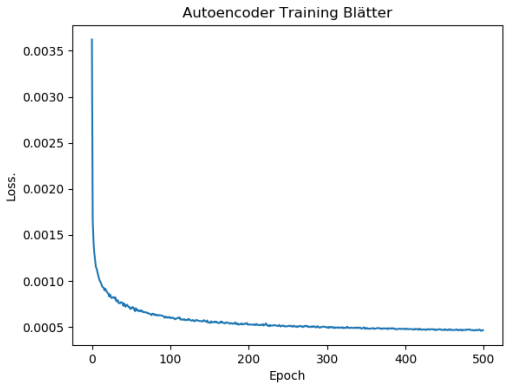


**Abb. 39.** Trainingsverlauf des Autoencoder mit CD-Distanz für zerstörte Blattdaten

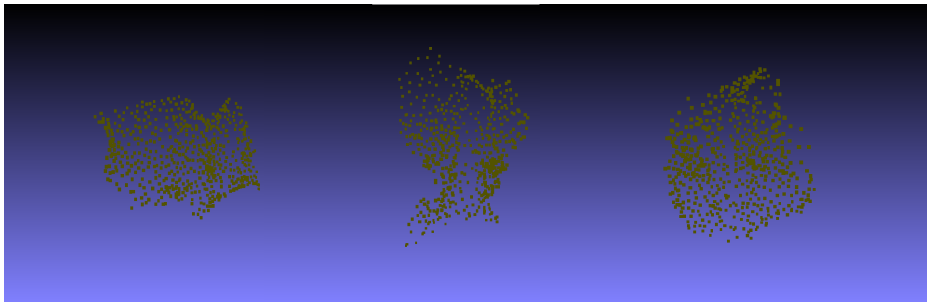

**Abb. 40.** Trainingsbeispiele des Autoencoder mit CD-Distanz für zerstörte Blattdaten - Output

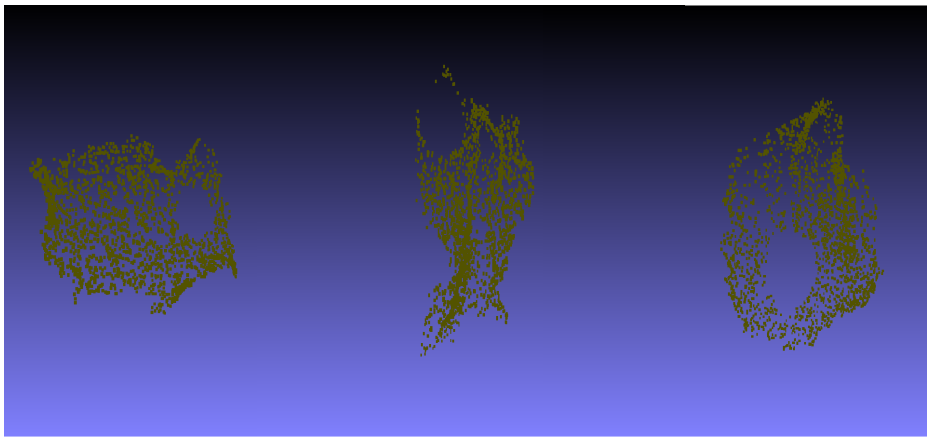

**Abb. 41.** Trainingsbeispiele des Autoencoder mit CD-Distanz für zerstörte Blattdaten - Input

Um nun bessere Validierungsergebnisse zu erlangen wurde Versuchaufbau 2.1 latent-CGAN nun noch mit EMD-Distanc für Autoencoder getestet. Es wurden 800 Epochen trainiert. Die Ergebnisse sind recht ähnlich zu der mit der CD-Distanc(vgl.2.4). Beispiel Trainingsinput Trainingsdatensätze können Abb. 43 entnommen werden mit ihren den jeweiligen erzeugten Output des Decoders in Abb. 44 entnommen werden. Wie zu erkennen ist vervollständigt der Autoencoder mit der EMD die Blätter selbstständig obwohl das nicht als Ziel in diesen Bearbeitungsschritt herbeizuführen ist. Im Vergleich dazu sind die Blätter aber an der Stelle des loches gleichmäßiger Verteilt und es gibt keine höhre Punktansammmlungen auf der Blattfläche. Wenn man die Ergebnisse mit der Hinsicht der Zielfunktion berücksichtigt liefert die CD ein besseres ERgebniss. 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. 39 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 Distanz 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.39 in Epoche 500 abzulesen ist. Das Training entwickelt sich jedoch seit mehren Epochen nicht mehr und stagniert. Auch mit dieser Anwendung kann das lantent C-GAN nicht weitergeführt werden da keine erfolgreichen Trainingsdaten erzeugt werden konnten. NOCH INPUT EINFÜGEN

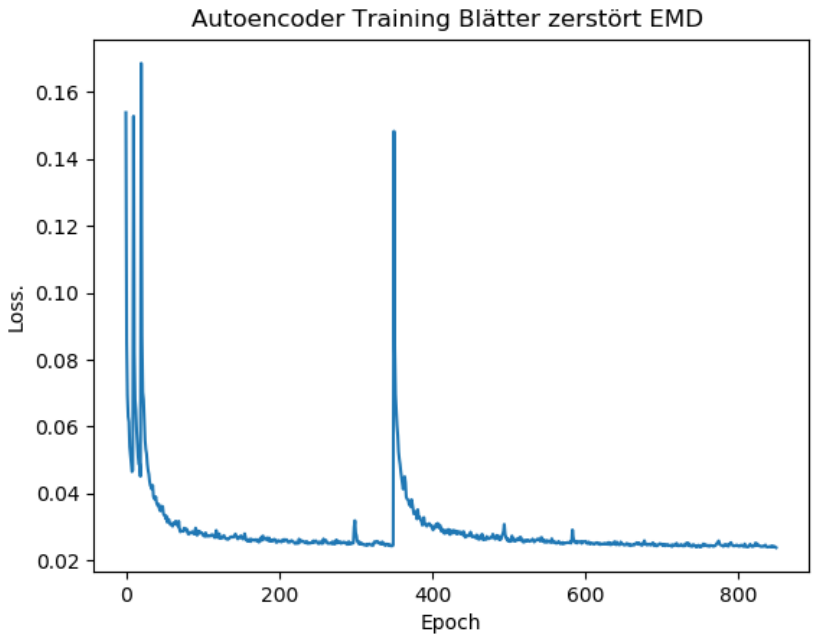


**Abb. 42.** Trainingsbeispiele des Autoencoder mit CD-Distanz für zerstörte Blattdaten - Output

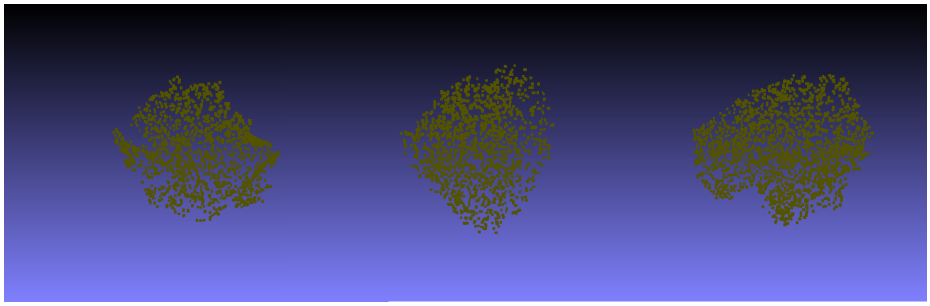

**Abb. 43.** Trainingsbeispiele des Autoencoder mit EMD für zerstörte Blattdaten - Output

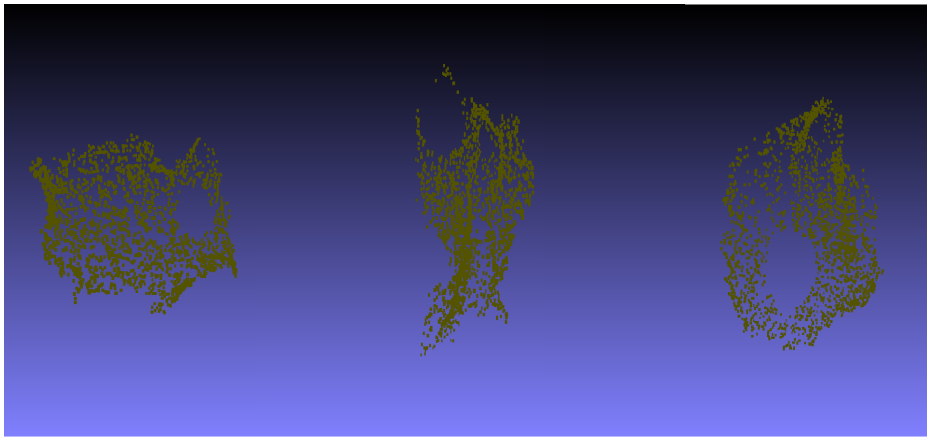

**Abb. 44.** Trainingsbeispiele des Autoencoder mit CD-Distanz für zerstörte Blattdaten - Input

## 5 Zusammenfassung und Diskussion

Ein GAN besteht aus zwei ANN, dem Discriminator und dem Generator. Das Ziel des G ist es Daten zu erzeugen, welche nicht von Trainingsdaten unterschieden werden können. Der Discriminator hat die Aufgabe zu unterscheiden ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von den Trainingsdaten stammt(Goodfellow et al., 2014). GANs werden derzeit noch erforscht. Es gibt noch einige offene Fragen, beispielsweise bezüglich der Performance hochauflösender Bilder(Wang et al., 2017). Es wurden in diesem Paper einige Probleme, welche beim Trainieren von GAN auftreten können und mögliche Lösungsansätze, vorgestellt. Es gibt derzeit einige praktische Ansätze, welche in der Anwendung auf GANs zurück greifen. Beispielsweise durch Textbeschreibung eigene Bilder als Output generiert werden (Reed et al., 2016), oder 3D Daten erzeugt werden(Achlioptas et al., 2018). GANs finden Anwendung in unterschiedlichen Bereichen des Deep Learnings, da sie als Lösung des Problems angesehen werden, dass Neuronale Netzwerke eine hohe Menge an Trainingsdaten benötigen und GANs dieses Problem durch ihre Fähigkeit, neue Daten zu generieren, umgehen. GANs lernen eine Art "versteckeRepräsentation von Klassen, was dazu beitragen kann auch Modelle von komplexen Prozessen zu erlernen. Es gibt erste Ansätze bei denen im Reinforcement Learning durch GANs versucht wird Modelle von der Umwelt eines Agenten zu erlernen, welche dann auf Grundlage dieser Modelle Vorhersagen über zukünftige Ereignisse treffen kann(Pinto, Davidson, Sukthankar, & Gupta, 2017).

## Abbildungsverzeichnis

## Tabellenverzeichnis

## 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*.

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

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

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).

Halevy, A., Norvig, P., & Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, *24*(2), 8–12.

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*.

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).

Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017). Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702*.

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*.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.

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).

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., & Catanzaro, B. (2017). High-resolution image synthesis and semantic manipulation with conditional gans. *arXiv preprint arXiv:1711.11585*.

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*.