Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Deep-Learning-Methoden zur Erstellung von 3D-Modellen

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Markus Paschke

Betreuer: M. Sc. Benedict Stephan
Dr.-Ing. Steffen Müller

Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 22.07.2020 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

Erklärung:

„Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet."

Ilmenau, 22.07.2020 .....

Markus Paschke

Zusammenfassung

Das Teilgebiet der Computer Vision, dass sich mit der bildbasierten 3D-Rekonstruktion aus einem oder mehreren 2D-Bildern beschäftigt, ist ein hochaktuelles Forschungsthema und grundlegend für eine Vielzahl von Anwendungen. In den letzten Jahren wurden viele Ansätze vorgestellt, die vermehrt auf die Nutzung von tiefen neuronalen Netzwerken setzen. Im Rahmen dieser Arbeit werden drei State of the Art Methoden vorgestellt, die sich in der Art ihrer Netzwerkarchitektur, sowie der Ausgabe-Repräsentation stark unterscheiden. Darunter fallen das *Point Set Generation Network (PSG)*, das *Octree Generation Network (OGN)* und das *Pixel2Mesh Network (P2M)*, welche als Ausgabe eine Punktwolke, einen Octree für eine Voxeldarstellung und ein Dreiecksmesh generieren. Das P2M-Netzwerk wurde als vielversprechendster Kandidat für das nachfolgend definierte Szenario ausgewählt und in weiteren Experimenten u. a. bezüglich Verdeckung und den Einfluss des Blickwinkels auf einem eigen erstellten Handwerkzeug-Datensatz trainiert und evaluiert. Die finale Auswertung bestätigt die Nutzung dieses Verfahrens bezüglich des Anwendungsszenarios zur 3D-Rekonstruktion anhand eines RGB(D)-Bildes für einen mobilen Roboter zur Bestimmung der Greifposition auf theoretischer Basis und gibt einen Ausblick auf Problemfelder und Optimierungsmöglichkeiten.

Abstract

The subfield of computer vision, which deals with image-based 3D reconstruction from one or more 2D images, is a highly topical research topic and fundamental for a variety of applications. In recent years, many approaches have been presented that increasingly rely on the use of deep neural networks. In this thesis, three state-of-the-art methods are presented, which differ strongly in their network architecture and output representation. Among them are the *Point Set Generation Network (PSG)*, the *Octree Generation Network (OGN)* and the *Pixel2Mesh Network (P2M)*, which generate as output a point cloud, an octree for a voxel representation and a triangle mesh. The P2M network was selected as the most promising candidate for the scenario defined below and was trained and evaluated in further experiments, e.g. regarding occlusion and the influence of the angle of view on a self-generated hand-tool dataset. The final evaluation confirms the use of this method regarding the application scenario for 3D reconstruction using an RGB(D) image for a mobile robot to determine the gripping position on a theoretical basis and gives an outlook on problem areas and optimization possibilities.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CNN	Convolutional Neural Network
GAN	Generative Adversarial Networks
cGAN	Conditional Generative Adversarial Networks
VAE	Variational Autoencoder
GCN	Graph Convolutional Network
OGN	Octree Generation Network
PSG	Point Set Generation
P2M	Pixel2Mesh
MVS	MultiView Stereo
VGG	Visual Geometry Group
FC	Fully Connected
GT	Ground Truth
SDF	Signed Distance Function
BPA	Ball Pivoting Algorithm
IoU	Intersection over Union
MCE	Mean of Cross Entropy
MSE	Mean Squared Error
CD	Chamfer Distance
EMD	Earth Mover Distance
SN	ShapeNet
3DW	3D Warehouse
CVPR	Conference on Computer Vision and Pattern Recognition

Kapitel 1

Einleitung

Ein grundlegendes Problem in der Computer Vision ist das Verstehen der Struktur und Extraktion von geometrischen Informationen einer realen Szene anhand von einem oder mehreren Bildern. Die Rekonstruktion von dreidimensionalen Modellen (im Folgenden 3D-Modelle genannt) ist ein großes Teilgebiet dieser Wissenschaft und hat in den letzten Jahren immer mehr an Bedeutung gewonnen. In den letzten Jahrzehnten erfolgte die Gewinnung der fehlenden Dimension aus 2D-Bildern vorwiegend durch klassische `multiview stereo` (MVS) und `shape-from-X` Ansätze. Dabei wird beim MVS versucht, ein Korrespondenzproblem einzelner Pixel zwischen zwei Bildern zu lösen [Hartley und Zisserman, 2003]. Die Berechnung guter Modelle benötigt jedoch viele Bilder aus unterschiedlichen Perspektiven und kalibrierte Kameras.

Menschen sind besonders gut darin, die 3D-Struktur eines Objektes aus einem Bild abzuleiten. Wir können Erfahrungen und bereits bekannte Objekte in Betracht ziehen, um uns ein mentales Modell des Objektes, sogar aus einem anderen Blickwinkel, vorzustellen. Diese a priori Informationen und Erfahrungen bezüglich der Objekte und Formen, können durch ein Training eines neuronalen Netzwerks ähnlich gut abgebildet werden. Aufgrund der nun zur Verfügung stehenden notwendigen Rechenkapazität konnte in den letzten Jahren die Bildererkennung große Fortschritte erzielen. Vor allem jedoch die Verfügbarkeit von großen 3D-Modell-Datensätzen ermöglicht die Nutzung von NN im Kontext der Geometriewiederherstellung.

1.1 Motivation

Die Möglichkeit dreidimensionale Modelle anhand von einem oder mehreren Bildern zu erstellen ist von grundlegenden Nutzen für eine Reihe von Applikationen. Anwendungsszenarien beinhalten u. a. die Roboter-Navigation, Assistenzsysteme, Industrie 4.0, Medizinische Erkennung und Hilfe (assistierte Operationen) sowie das Greifen und Reichen von Objekten in der Pflege.

Für eine Mensch-Roboter-Kollaboration ist das Schätzen einer geeigneten Greifpose für bekannte und unbekannte Objekte notwendig. Kameras und Tiefensensoren des Roboters liefern bereits eine Teilansicht des Objektes, dessen abgewandte Seite konstruiert werden muss. Auch wenn das Gebiet der 3D-Rekonstruktion hochaktuell ist und daran aktiv geforscht wird, so existieren bereits eine Vielzahl von Ansätzen und Methoden zur Erstellung von 3D-Geometrien aus RGB-Bildern, die im Folgenden genauer untersucht werden.

Diese Arbeit fokussiert sich insbesondere auf das Anwendungsszenario eines mobilen Roboters, der sowohl über RGB-Kameras, als auch Tiefendaten dank einer Kinect-Kamera verfügt. Die Hauptaufgabe sieht vor, Handwerkzeuge einer Person entgegenzunehmen oder zu reichen. Daraus resultiert als zusätzliche Aufgabe neben der Evaluierung eines geeigneten neuronalen Netzwerks die Erstellung eines solchen Datensatzes. Wie gut schneidet ein trainiertes Netzwerk auf diesen Daten ab und ist es für die Implementierung auf dem Roboter geeignet?

1.2 Ziel der Arbeit

Diese Arbeit dient mit einer Analyse und Evaluierung verschiedener State of the Art Techniken als Grundlage hinsichtlich der Frage, ob ein untersuchter Ansatz für die Implementierung in einen mobilen Roboter zur Bestimmung der Greifposition in Betracht gezogen werden kann. Dafür werden die Netzwerkarchitekturen analysiert, anschaulich dargestellt und bezüglich der Ausgabe-Repräsentation für die Nutzbarkeit eingeordnet. Darauffolgend wird der erfolgversprechendste Ansatz auf einem selbst erstellten Handwerkzeug-Datensatz trainiert und anhand von mehreren Fehlermetriken untersucht. Abschließend werden weitere Experimente, darunter der Einfluss von

Verdeckung und eine Vertex Analyse genutzt, die Struktur und das Verhalten des neuronalen Netzwerks genauer zu bestimmen und greifbar zu machen.

1.3 Aufbau der Arbeit

Dieses Kapitel leitet den Anfang der Arbeit, die in insgesamt 6 Kapitel gegliedert ist, ein. Das folgende Kapitel geht ausführlich auf die theoretischen Grundlagen verschiedener neuronalen Netzwerkarchitekturen für eine 3D-Rekonstruktion ein. Ein besonderes Augenmerk wird auf das Graph Convolution Network (GCN) gelegt. Kapitel 3 zeigt mögliche Formen der Eingabe, der Netzwerkarchitektur und der Ausgabe für neuronale Netze bezüglich relevanter Merkmale im Kontext der 3D-Rekonstruktion auf. Zudem wird auf den aktuellen Stand der Technik eingegangen und eine detaillierte Analyse des Point Set Generation Network(PSG), des Octree Generation Network (OGN) und des Pixel2Mesh Network (P2M) geliefert. In Kapitel 4 werden die benötigten Schritte erläutert, die zur Erstellung eines eigenen Handwerkzeug-Datensatzes genutzt wurden. Auf diesen Daten wurde anschließend Pixel2Mesh trainiert und durch eine Reihe von Experimenten evaluiert. Die Ergebnisse hierzu können in Kapitel 5 entnommen werden. Abschließend wird in Kapitel 6 die Arbeit zusammengefasst, festgestellte Probleme erläutert und ein Ausblick auf Optimierungen und mögliche zukünftige Arbeiten gegeben.

Kapitel 2

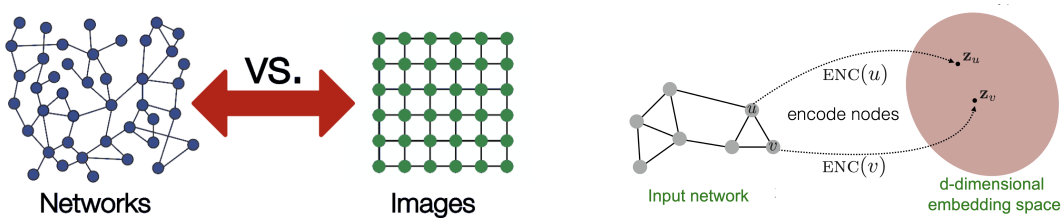
Grundlagen

Für das Verständnis der im State of the Art Kapitel aufgegriffenen Verfahren werden zunächst Grundlagen vorgestellt. Basiswissen bezüglich künstlichen neuronalen Netzwerken, genauer Convolutional Neural Networks (CNN), wird jedoch als gegeben vorausgesetzt. Dabei wird besonders auf die Idee von Graph Convolution Networks (GCN) eingegangen, welche Faltungen auf Knoten in einem Graphen definieren.

2.1 Graph Convolutional Network (GCN)

Viele der heutigen Datenbanken besitzen Abhängigkeiten zwischen Einträgen, die besonders gut durch einen Graphen abgebildet werden können. Darunter fallen u. a. soziale Netzwerke, Wissensgraphen, Bioinformatik und geometrischen Strukturen. Graphen sind irreguläre Strukturen und besitzen keinen fixen Start- bzw. Bezugspunkt (siehe Abbildung 2.1a). Die Anzahl der Knoten und Kanten ist nicht limitiert, sodass komplexe topologische Strukturen entstehen können. Während auf regulären Gittern, z.B. einem Bild, Merkmale durch Anwendung einer sogenannten Faltungsmatrix über dem Bildausschnitt extrahiert werden können, ist es nicht möglich, dies mit einem naiven Ansatz direkt auf Graphen zu übertragen.

Die grundsätzliche Idee ist, das originale Netzwerk bzw. den Graphen durch eine Encoding Funktion ENC in ein niedrig dimensionalen Raum (embedding space) abzubilden, wobei die Ähnlichkeit und die räumliche Nähe im embedding space die Struktur, Nachbarschaftsbeziehung und Distanz der einzelnen Knoten des originalen Graphen



(a) (b)

Abbildung 2.1: Vergleich einer Graph-Struktur mit einem 2D-Raster (a) und eine visuelle Darstellung einer Encoding Funktion zur Einbettung eines Knoten des Graphen in ein niedrig dimensionalen Raum (b) [Jure Leskovec, 2018].

möglichst genau abbilden. Die Encoding Funktion $ENC(v)$, die einen Knoten v in ein d-dimensionalen Raum einbettet, ist in Abbildung 2.1b zu sehen. Auf diese Weise wird jedem Knoten ein Feature-Vektor zugewiesen. Das Verfahren wird in [Jure Leskovec, 2018] detailliert beschrieben.

In einer klassischen 2D-Faltung entstehen Feature-Maps für ein Bild, indem die Eingabe von einer festgelegten Anzahl von Filtern analysiert wird. Die Dimensionen der Feature-Maps hängen von der Größe des Kernels, dem Padding und der Schrittweite ab. Durch die Größe des Filters und die Tiefe des Netzwerks, können gleichermaßen high-level und low-level Bildmerkmale extrahiert werden. Beispiele für low-level Features bei einem Bild eines menschlichen Kopfes sind u. a. Ecken, Kanten und Kreise, während high-level Features ein Auge, die Nase oder den Mund abbilden. Diese lokalen Abhängigkeiten können durch eine Faltung auf den Knoten des Graphen ebenfalls genutzt werden. Abbildung 2.2 vergleicht einen klassischen Faltungsschritt auf einem Grid (links) mit einem Faltungsschritt auf Graphen (rechts) durch einen 3×3 Filter.

Eine Faltung auf einem Graphen kann wie eine klassische 2D-Faltung verstanden werden. Jeder Knoten verrechnet seine eigenen Features mit den Features seiner Nachbarn. Die genaue Formel wird im Folgenden genauer analysiert, jedoch ist zu beachten, dass die Operation zur Aggregierung aller Knoten-Features order-invariant, d. h. invariant bezüglich der Reihenfolge sein muss. Der Durchschnitt, die Summe oder das Maximum aller zu betrachtenden Knoteninformationen ist order-invariant, da eine Permutation der Knotenreihenfolge kein anderes Ergebnis zur Folge hat.

In diesem Beispiel entspricht der 3×3 Filter auf dem Bild der Aggregierung des mittleren Knotens und seiner Nachbarknoten von einem Hop Entfernung. Diese Reichweite kann als Hyperparameter beliebig groß gewählt werden, sollte jedoch für Graphen, mit einer hohen Anzahl an ausgehenden Kanten der Knoten, zwischen zwei und sechs Hops liegen. Dies liegt an dem Kleine-Welt-Phänomen, der für u. a. für Social-Network-Graphen zutrifft. Das Phänomen, auch bekannt unter dem Namen *Six Degrees of Separation*, besagt, dass jeder Mensch zu einem beliebig anderen über eine kurze Kette von sechs Bekanntschaftsbeziehungen verbunden ist. Eine zu groß gewählte Reichweite berechnet, abhängig von dem gewählten Operator z. B. den Durchschnitt oder das Maximum des gesamten Netzwerks für jeden einzelnen Knoten. Im Falle eines Dreiecksmeshs mit sechs ausgehenden Kanten eines Knotens kann jedoch eine höhere Hop Anzahl genutzt werden.

Abbildung 2.3 skizziert den Berechnungsgraphen für jeden einzelnen Knoten des Eingabe-Graphen mit einer Reichweite von zwei Hops. Zur Berechnung des Feature-Vektors des Knoten A nach einem Faltungsschritt müssen die Nachbarknoten und dessen Nachbarknoten aufsummiert werden. Zur Berechnung werden zunächst die Feature-Vektoren der Nachbarknoten von B,C,D aufsummiert und zu den jeweils eigenen Feature-Vektor addiert. Die Summe von B,C,D und A bildet nun den finalen Feature-Vektor nach einem Faltungsschritt für den Knoten A. Dies wird für jeden

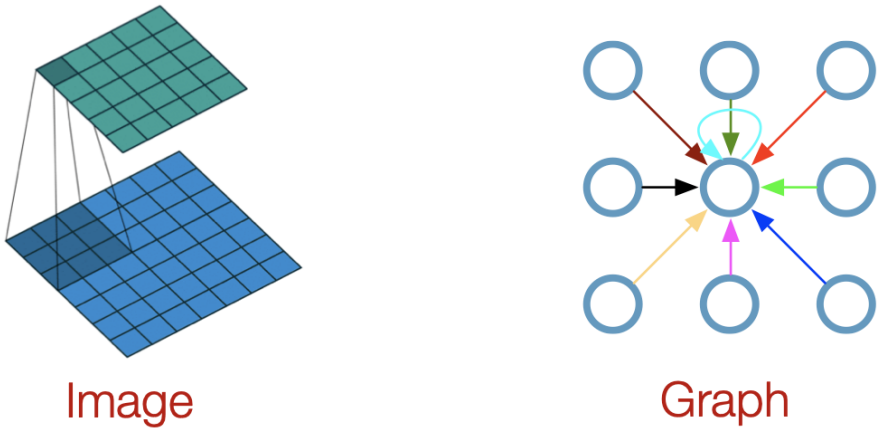


Abbildung 2.2: Einzelne CNN und GCN Schicht mit einem 3×3 Filter [Jure Leskovec, 2018]

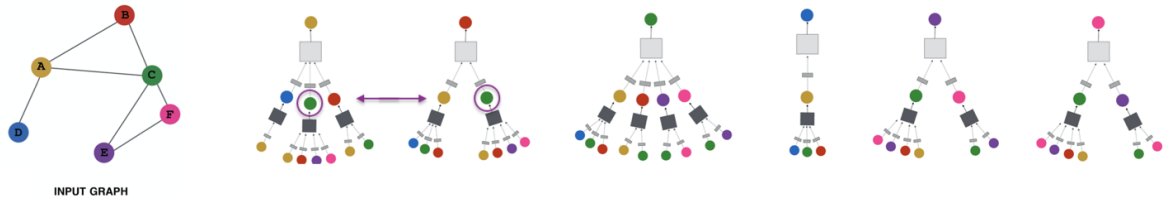


Abbildung 2.3: Berechnungsgraph jedes Knoten für einen Graphen mit einer Reichweite von zwei Hops [Jure Leskovec, 2018]

Knoten individuell berechnet, jedoch können Teilergebnisse wiederverwendet werden. So kann zum Beispiel für Knoten B das Teilergebnis von Knoten C aus der Berechnung für Knoten A im vorherigen Schritt genutzt werden.

Eine Faltung für Knoten v mit Hop Anzahl K (im folgenden Layer genannt) berechnet sich wie folgt:

$$
\begin{aligned}
h_v^0 &= x_v \\
h_v^k &= \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}), \forall k \in \{1,...,K\} \\
z_v &= h_v^K
\end{aligned}
\tag{2.1}
$$

mit:
x_v = Initialer Feature-Vektor für Knoten v
h_v^k = Feature-Vektor für Knoten v auf Layer k
σ = Aktivierungsfunktion
$N(v)$ = Nachbarknoten von v
W_k = Gewichtsmatrix für Nachbarknotenaggregation
B_k = Gewichtsmatrix für eigene Knoteninformation
z_v = Feature-Vektor von v nach Faltung

Auf initial Layer 0, bestehen die Knoteninformationen von Knoten v aus seinen eigenen Features x_v. Im Layer k werden die Knoteninformationen von Knoten v, durch eine Anwendung einer Aktivierungsfunktion σ (meistens ReLU) auf die gewichtete Summe über den Durchschnitt aller Nachbarknoteninformationen aus dem vorherigen Layer mit den eigenen gewichteten Knoteninformationen aus dem vorherigen Layer berechnet. Nach K Layern der Nachbarschaftsaggregation, bildet z_v den finalen Feature-Vektor

des Knotens v im embedding space. Die Gewichtsmatrizen W_k und B_k steuern den Einfluss der eigenen und der Nachbarschafts-Knoteninformationen und können durch Gradientenabstieg trainiert werden.

Das grundlegende Verständnis von Graph-basierten Netzwerken, sowie der einzelnen Schritte in der Rekonstruktion eines Dreiecksmeshs sollte nun soweit gefestigt sein, sodass im Abschnitt 3.5 im folgenden Kapitel Pixel2Mesh genauer untersucht werden kann.

2.2 Metriken und Verlustfunktionen

Metriken und Verlustfunktionen werden sowohl als Fehlerfunktion im Training als auch in der Evaluation und dem direkten Vergleich von Punktwolken oder volumenbasierten Repräsentationen genutzt. In diesem Abschnitt wird ein Überblick über existierende Metriken gegeben, mit Fokus auf der Chamfer- und Earth-Mover-Distanz.

2.2.1 Punktwolken basierte Repräsentationen

[Fan et al., 2017] führt drei Bedingungen ein, die eine geeignete Verlustfunktion erfüllen muss, um für ein neuronales Netz in Frage zu kommen:

1. Robust gegenüber Ausreißer-Punkten.
2. Differenzierbar in Bezug auf Punktpositionen.
3. Effizient zu berechnen, da die Funktion an vielen Stellen des Netzes während des Trainings berechnet werden muss.

Als Metrik kann für Punktwolken basierte Repräsentationen sowohl die Chamfer Distance (CD) als auch mit die Earth Mover's Distance (EMD) [Fan et al., 2017] genutzt werden.

Die **Chamfer Distance** zwischen zwei Punktwolken $S_1, S_2 \subseteq \mathbb{R}^3$ ist definiert als:

$$d_{CD}(S_1,S_2) = \frac{1}{S_1} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \frac{1}{S_2} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2 \tag{2.2}$$

Die **CD** ist kontinuierlich und verwendet suboptimales Matching, um die nächstliegenden Nachbarpunktpaare in beiden Sätzen zu bestimmen. Dafür wird für jeden

Punkt $x \in S_1$ bzw. $x \in S_2$ ein Punkt aus der jeweils anderen Punktwolke durch eine Nächste-Nachbar-Suche bestimmt und die quadratische euklidische Distanz zwischen beider Punkte gebildet. Die Summe des Mittelwerts aller berechneten Punktdistanzen bildet anschließend die Chamfer-Distanz. Alternativen ohne Mittelwertbildung existieren ebenfalls in der Literatur. Da die Berechnung der Distanz eines Punktpaares unabhängig von vorangegangen Berechnungen ist, kann dies leicht parallelisiert werden.

Aufgrund der leichten Berechnung wird sie häufig als Fehlerfunktion genutzt, ist jedoch sehr sensibel bezüglich der Skalierung oder Rotation eines Objektes und der dazugehörigen Punktwolke. Aus diesem Grund wurden alle Objekte der nachfolgenden Experimente sowie die Schere und der Hammer in der Abbildung 2.4 anhand ihrer Bounding Box auf den Einheitswürfel skaliert. Das Beispiel soll einen praktischen Eindruck über die Qualitätsmetrik der Chamfer- und Earth-Mover-Distanz zwischen einem Mesh und einer Punktwolke vermitteln. Dazu werden auf dem Mesh 16384 Punkte abgetastet und mit den 16384 Punkten der Punktwolke verglichen. Die Schere in der Mitte bildet die GT-Punktwolke sehr gut ab, besitzt jedoch nicht die zwei Löcher am Kopfende. Daraus resultiert eine CD von 0.07. Eine weitaus schlechtere CD mit 0.35 erzeugt die rechte Schere, welche die GT-Form nicht perfekt abbildet und etwas verschoben ist. Aufgrund der fehlenden Genus-0-Form der Schere, wird in einem zweiten Beispiel der Vergleich durch einen Hammer aufgezeigt. Unter dem Geschlecht (Genus) einer Fläche versteht man in der Topologie die Anzahl der Löcher einer Fläche. Auch hier wird der Unterschied zwischen einer guten und schlechten Rekonstruktion sichtbar. Der Hammer (im Bild rechts) mit einer CD von 0.19 besitzt keinen guten Übergang vom Stiel zum Hammerkopf. Zu viele Knoten des Polygons befinden sich außerhalb der GT-Punktwolke, was zu einer schlechteren CD und EMD führt.

Die **Earth Mover Distance** versucht ein Transportproblem zu lösen. Der Name kommt daher, dass die Arbeit gemessen wird, die nötig ist, einen Erdhügel in einen anderen zu transformieren. Das Prinzip kann ebenfalls auf Punktwolken angewandt werden. Gegeben sind zwei Punktwolken $S1, S2$ mit $|S1| = |S2|$. Ferner ist die EMD zwischen $S1$ und $S2$ definiert als die minimalen Kosten eines perfekten Matchings zwischen $S1$ und $S2$, d.h.:

$$d_{EMD}(S_1,S_2) = \min_{\phi:S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \tag{2.3}$$

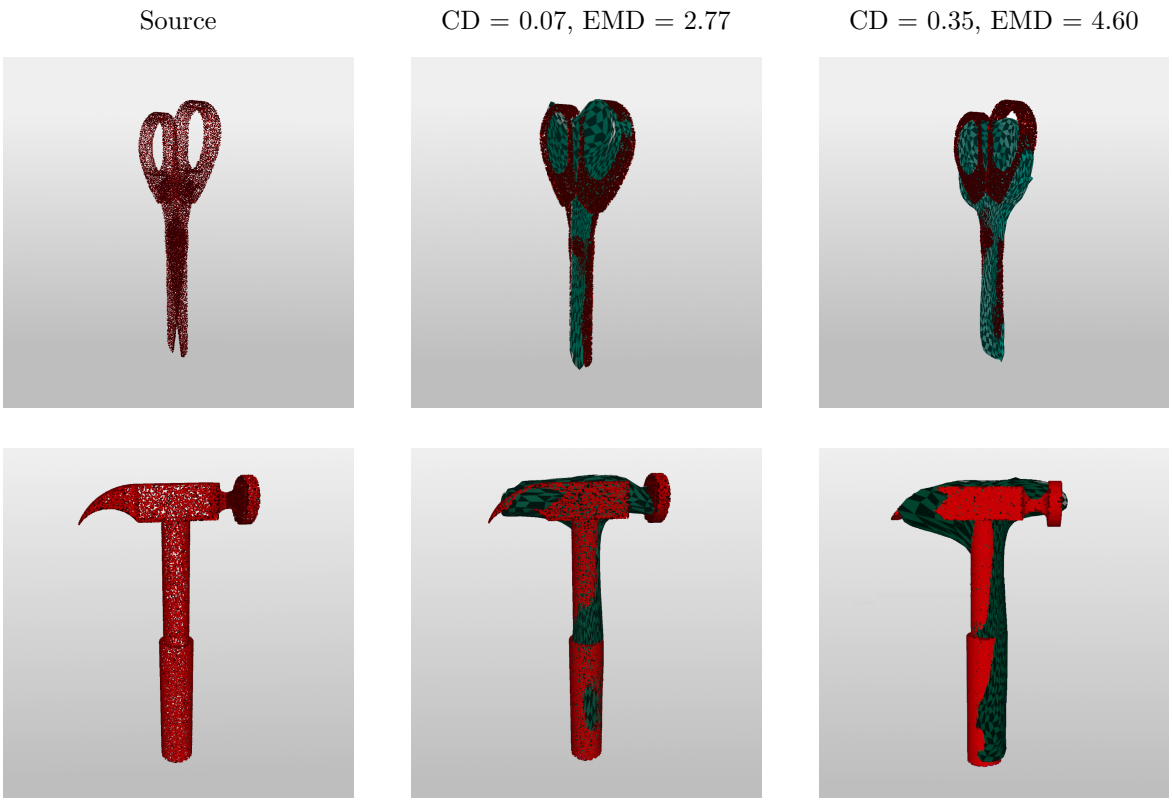


Abbildung 2.4: Vergleich der Chamfer- und Earth-Mover-Distanz für zwei Scheren bzw. Hammer und einer GT-Punktwolke

Die EMD ist die durchschnittliche Abweichung aller gegebenen Punkte, genauer die optimale Bijektion φ der Punkte $S_1 \to S_2$ unter infinitesimaler Bewegung. Da alle Permutationen berechnet werden müssen, ist diese Verlustfunktion teurer als die CD.

In der Praxis wird meist eine Approximation inspiriert durch den Auktionsalgorithmus von [Bertsekas, 1985] implementiert. Es wird angenommen, dass N Personen N Waren untereinander aufteilen möchten. Ware j wird genau einer Person i zugeordnet, die einen Preis p_j, der noch berechnet wird, zahlt. Der Wert a_{ij} drückt aus wie viel die Ware j für Bieter i wert ist und jede Person i möchte seinen Profit $a_{ij} - p_j$ maximieren. Gesucht ist also ein perfektes Matching M im bipartiten Graphen, sodass $a(M) = \sum_{i,j \in M} a_{ij}$ maximiert wird. Zunächst wird ein initialer Preis p_j für jedes Objekt erstellt. Die anschließende Auktion teilt sich in zwei Phasen auf, die sich abwechseln. In der **Bieten Phase** berechnet jede Person ihre erste und zweite Wahl einer Ware $(j^*, \hat{j})$ mit den

kleinsten Kosten:

$$\begin{aligned} c_{ij} &= p_j - a_{ij} \\ c_{ij^*} &= \min_j c_{ij} \\ c_{i\hat{j}} &= \min_{j \neq j^*} c_{ij} \end{aligned} \tag{2.4}$$

mit: c_{ij} = Aktuelle Kosten für Ware j für Person i
c_{ij^*} = Niedrigste Kosten - Beste Ware j^* für Person i
$c_{i\hat{j}}$ = Zweitniedrigste Kosten - Zweitbeste Ware $\hat{j}$ für Person i

Person i, die noch keine Ware besitzt, bietet auf Ware j^*, indem der Preis dieser Ware um eine festgelegte Konstante ϵ erhöht wird. In der **Zuordnung Phase** wird für jedes Objekt j, auf das geboten wurde, der Preis p_j auf das höchste Gebot erhöht. Person i^*, die das höchste Gebot abgegeben hat, bekommt temporär die Ware, während vorher zugeordneten Bietern, diese wieder entzogen wird. Der Algorithmus terminiert nachdem alle Bieter eine Ware besitzen mit einer Zuordnung, die $(N-1)\epsilon$ optimal ist. Eine asynchrone Implementierung, die paralleles Bieten und Zuordnen ermöglicht, wird von [Bertsekas, 1985] ebenfalls bereitgestellt. Im Kontext der Earth-Mover-Distanz wird die Wahl der Ware anhand der zwei nächsten Nachbarn in der anderen Punktwolke festgelegt. Zudem wird die Terminierung durch eine Begrenzung der Zeit pro Phase garantiert Fan et al., 2017.

2.2.2 Volumenbasierte Repräsentationen

Für volumenbasierte Darstellungen gibt es verschiedene Qualitätsmetriken, darunter der Jaccard-Koeffizient auch Intersection over Union (IoU) genannt, der Mittelwert der Kreuzentropie – Mean of Cross Entropy (MCE) oder der mittlere quadratischer Fehler – Mean Squared Error (MSE).

Die **Intersection over Union** (nach [Leibe et al., 2016]) wird, wie der Name schon sagt, durch das Verhältnis der Volumina der Schnittmenge über ihre Vereinigung

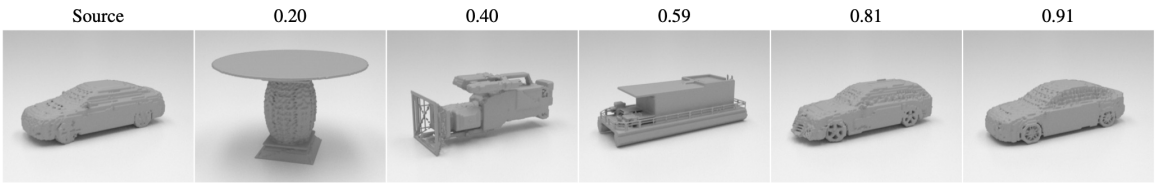


Abbildung 2.5: IoU zwischen einem GT-Voxel-Objekt und verschiedenen anderen Objekten. Je höher die IoU, desto ähnlicher sind sich die Objekte. [Tatarchenko et al., 2019]

definiert:

$$d_{IoU}(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\sum_i \{I(\hat{V}_i \geq \epsilon) * I(V_i)\}}{\sum_i \{I(I(\hat{V}_i \geq \epsilon) + I(V_i))\}} \tag{2.5}$$

mit: $I(\cdot)$ = Indikator-Funktion
$\hat{V}_i$ = Vorhergesagter Wert für den Voxel an der i-ten Stelle
V_i = GT-Wert für den Voxel an der i-ten Stelle

Anhand eines festgelegten Schwellenwerts ϵ werden die Voxel `binarized`, d. h. auf 0 oder 1 abgebildet. Die Indikator-Funktion $I(\cdot)$ gibt an, ob ein Voxel belegt oder frei ist. Auf diese Weise wird der Schnitt aller belegten Voxel über der Vereinigung aller belegten Voxel gebildet. Abbildung 2.5 verdeutlicht das Ähnlichkeitsmaß der Intersection over Union für ein GT-Voxel-Objekt mit diversen anderen Objekten.

Die **Mean of Cross Entropy** (nach [Su et al., 2015]) für Voxel oder Punktwolken ist wie folgt definiert:

$$d_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \{V_i \; log \; \hat{V}_i + (1 - V_i) \; log \; (1 - \hat{V}_i)\} \tag{2.6}$$

mit: N = Anzahl der Voxel oder Punkte
$\hat{V}_i, V_i$ = Siehe Gleichung 2.5

Der **Mean Squared Error** (nach [Pontes et al., 2018]) ist wie folgt definiert:

$$d_{MSE} = \frac{1}{n_X} \sum_{p \in X} d(p, \hat{X}) + \frac{1}{n_{\hat{X}}} \sum_{\hat{p} \in \hat{X}} d(\hat{p},X) \tag{2.7}$$

mit: $n_{\hat{X}}$ = Anzahl der abgetasteten Punkte auf dem vorhergesagten Mesh $\hat{X}$
n_X = Anzahl der abgetasteten Punkte auf dem GT-Mesh X
$\hat{p}$ = Abgetasteter Punkt auf dem vorhergesagten Mesh $\hat{X}$
p = Abgetasteter Punkt auf dem GT-Mesh X
$d(\cdot,\cdot)$ = L_1 oder L_2 Distanz

Zur Bestimmung der symmetrischen Oberflächendistanz zwischen zwei Meshs werden auf den Facetten zufällig Punkte abgetastet. Anhand des Normalenvektors kann die Manhatten Distanz (L_1) oder die Euklidische Distanzt (L_2) von einem Punkt zur Oberfläche des anderen Meshs bestimmt werden. Die mittlere Summe über alle Punkte beider Oberflächendistanzen ergibt den mittleren quadratischen Fehler.

Kapitel 3

State of the Art

In diesem Kapitel werden zunächst die taxonomischen Aspekte hinsichtlich der 3D-Rekonstruktion anhand von Deep-Learning-Netzwerken analysiert. Viele der in den letzten Jahren vorgestellten Verfahren teilen sich Eigenschaften bezüglich der Eingabe- und Ausgabe-Repräsentation oder die Netzwerkarchitektur per se. Diese vorangegangene Klassifizierung hilft im Anschluss eine Auswahl möglicher Kandidaten für den Anwendungsfall zur Konstruktion eines 3D-Modells zur Bestimmung der Greifposition eines Roboters zu treffen. Aus drei im Detail vorgestellten Verfahren wird ein Ansatz für weitere Experimente ausgewählt.

3.1 Taxonomie

In diesem Kapitel werden die möglichen Arten und Strukturen der Eingabe (Abschnitt 3.1.1), der Netzwerkarchitektur (Abschnitt 3.1.2) und der Ausgabe (Abschnitt 3.1.3) aktueller 3D-Rekonstruktionsverfahren genauer beleuchtet. Diese Datenformate und Module werden gezielt aufeinander abgestimmt. So ist zum Beispiel die Darstellung der Ausgabe entscheidend für die Wahl der Netzwerkarchitektur und hat Auswirkungen auf die Berechnungseffizienz und die Qualität der Rekonstruktion.

3.1.1 Eingabe

Mögliche Formen der Eingabe sind in Abbildung 3.1 aufgezeigt und haben starken Einfluss auf den Prozess zur Konstruktion einer 3D-Form. Folglich erfordern verschiedene

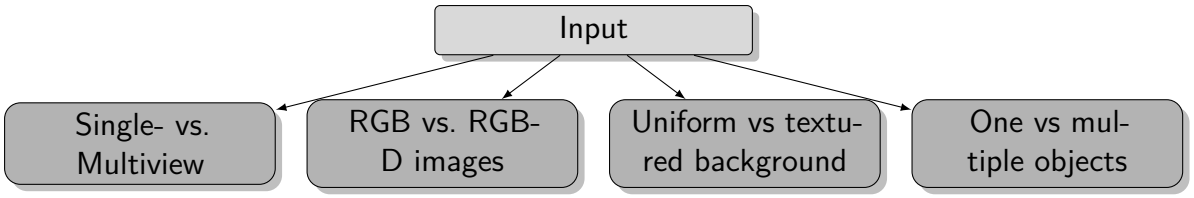


Abbildung 3.1: Mögliche Formen der Eingabe

Eingaben entsprechende Methoden und Algorithmen oder müssen vorher transformiert und angepasst werden. Die Wahl des richtigen Eingabeformats für das Netzwerk hängt von der Art der Anwendung ab. Im Falle eines sich selbst bewegenden Roboters könnten ein bis viele RGB-D-Bilder mit einem texturierten Hintergrund, der ein oder mehrere Objekte enthält, erzeugt werden. Ein Möbelgeschäft hingegen würde wahrscheinlich eine stationäre Kamera nutzen und ein RGB(D)-Bild mit einem einheitlichen Hintergrund, der genau ein Objekt enthält, erstellen.

Einzel- vs. Mehrfachansicht

Die Verwendung eines einzigen Bildes und damit eines einzigen Blickwinkels als Eingabe für die Rekonstruktion von 3D-Objekten führt zu vielfältigen Problemen. So muss die komplette geometrische Struktur der abgewandten Seite anhand eines Bildes vorhergesagt werden. Falls in diesem Bild außerdem eine Teilverdeckung auftritt oder die Umgebungsbeleuchtung nicht ausreicht, wird die Rekonstruktion zusätzlich erschwert. Ein positiver Aspekt ist jedoch zum einen die Laufzeit der Rekonstruktion und eine einfachere Netzwerkarchitektur. Dieser Methode steht die Mehrfachansicht `(MultiView)` 3D-Rekonstruktion gegenüber, die mehrere Bilder aus verschiedenen Blickwinkeln nutzt um eine einzelne 3D-Rekonstruktion eines Objektes herzustellen. Eine mögliche Implementierung liefert u. a. das Pix2Vox Netzwerk [Xie et al., 2019], das für jedes Eingabe-Bild eine volumetrische Repräsentation erstellt, welche im Anschluss zu einem finalen 3D-Modell fusioniert werden.

RGB vs. RGB-D-Bild

Die Nutzung von Tiefeninformationen eines Bildes als zusätzlicher Parameter für einen Encoder führt zu einer besseren und präziseren Repräsentation des dargestellten Objekts. Durch den Einsatz von Kinect-Kameras oder Lidar-Sensoren lassen sich 2,5D-

Bilder einfach erstellen. Dennoch spiegeln synthetisch erstellte Tiefendaten anhand eines Wavefront- oder CAD-Modells oftmals nicht die Tiefendaten einer realen Szene wider, da kein Sensorrauschen auftritt.

Einheitlicher- vs. texturierter Hintergrund

Die meisten 3D-Datensätze, die für das Training neuronaler Netzwerke genutzt werden, besitzen 3D-Modelle mit einheitlichen Hintergrund oder maskieren diese. Das Netzwerk wird daher nicht lernen Hintergrundinformationen zu ignorieren, wenn das Testbild solche besitzt. Wir als Menschen können gleichzeitig das Objekt aus dem Hintergrund segmentieren, klassifizieren und ein 3D-Modell aufbauen. Für das Training von derzeitigen State of the Art Verfahren an realen Daten wird daher oftmals das zu rekonstruierende Objekt zunächst freigestellt. Die Segmentierung kann mithilfe von Tiefeninformationen, einer Objektklassifizierung oder durch Bounding Boxes erreicht werden.

Ein Objekt vs. mehrere Objekte

Hier muss das gleiche Problem wie in Abschnitt 3.1.1 gelöst werden. Das Entfernen von zusätzlichen Objekten vor der Rekonstruktion des Hauptobjektes ist die erste Aufgabe in der Prozesskette. Das Training eines Netzwerks mit mehreren Objekten kann zu mehreren Objektrekonstruktionen führen, die nicht erwünscht sind. Andererseits führt ein Training des Netzwerks mit nur einzelnen Objekten und anschließenden Testen mit mehreren Objekten dazu, dass alle Objekte zu einem einzigen Objekt zusammengeführt werden. In aktuellen State of the Art Ansätzen wird in der Regel das Netzwerk mit einem einzelnen Objekt trainiert und entsprechend auch getestet.

3.1.2 Netzwerkarchitektur

Encoder-Decoder

Die Vanilla Netzwerkarchitektur besteht aus zwei Hauptkomponenten, dem Encoder und dem Decoder. Bei der Kodierung von Bildern sind Faltungsnetze die erste Wahl, seit dem das CNN AlexNet im Wettbewerb um die ImageNet Large Scale Visual Recognition Challenge 2012 [Krizhevsky et al., 2012] gewonnen hat. In dieser Architektur

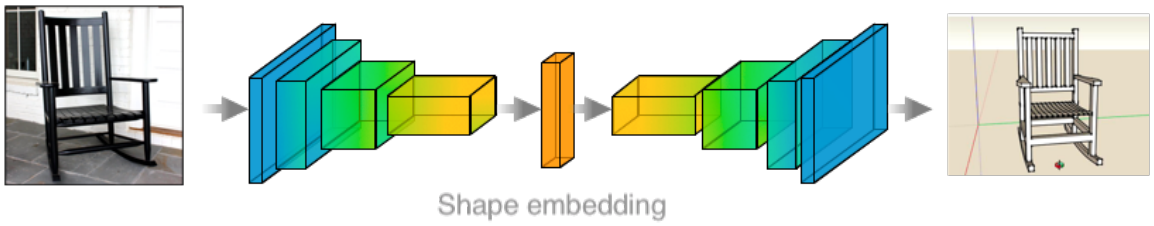


Abbildung 3.2: Encoder-Decoder Architektur[LÊ, 2018]

durchläuft das Eingangsbild mehrere Prozeduren, einschließlich Faltungs-, Pooling- und Regularisierungsoperationen, gefolgt von vollständig verschalteten Schichten zu einer typisch niederdimensionalen Darstellung. Anschließend dekomprimiert der Decoder diese Merkmalsvektoren in die Wahl der Ausgabedarstellung. Die Gewichte für den Encoder und Decoder werden durch Minimierung einer Kostenfunktion und einer Rückpropagierung des Fehler gesetzt. Die Generierung einer höherdimensionalen Ausgabedarstellung erfolgt hauptsächlich durch Deconvolution (inverse Faltung), Upsampling, Unpooling oder vollständig verschalteten Schichten.

TL-embedding networks

Die Idee eines TL-embedding network ist es, eine Vektordarstellung zu erlernen, die generativ und gut vorhersagbar ist. In Abbildung 3.3 zeigt der obere Teil die Encoder-Decoder Struktur, die zunächst in einem dreistufigen Ansatz trainiert wird. Die Eingabe ist eine $20 \times 20 \times 20$ Voxel-Map, die in einer 64D-Darstellung durch 4 Faltungsschichten kodiert wird, gefolgt von einer vollständig verschalteten Schicht. Die folgenden 3D-Faltungen erzeugen eine Voxelausgabe. Das Netzwerk wird mit einer Kreuzentropie als Verlustfunktion bezüglich der Eingangs-Voxel-Map trainiert. Zweitens wird der untere Teil des TL-embedding network, das ConvNet, so trainiert, dass es das 2D-Bild als die gleiche 64D-Darstellung aus dem ersten Teil darstellt. Zum Training wird hier eine euklidische Verlustfunktion gewählt. Im dritten Schritt wird das Netzwerk feinjustiert, wobei beide Zweige parallel trainiert werden. Das Training dieses Netzwerks erfordert eine vorherige Transformation der CAD-Modelle in 2D-Bilder und Voxel-Maps mit dem Voxelizer von [Wu et al., 2015] und ist in [Girdhar et al., 2016] zu finden.

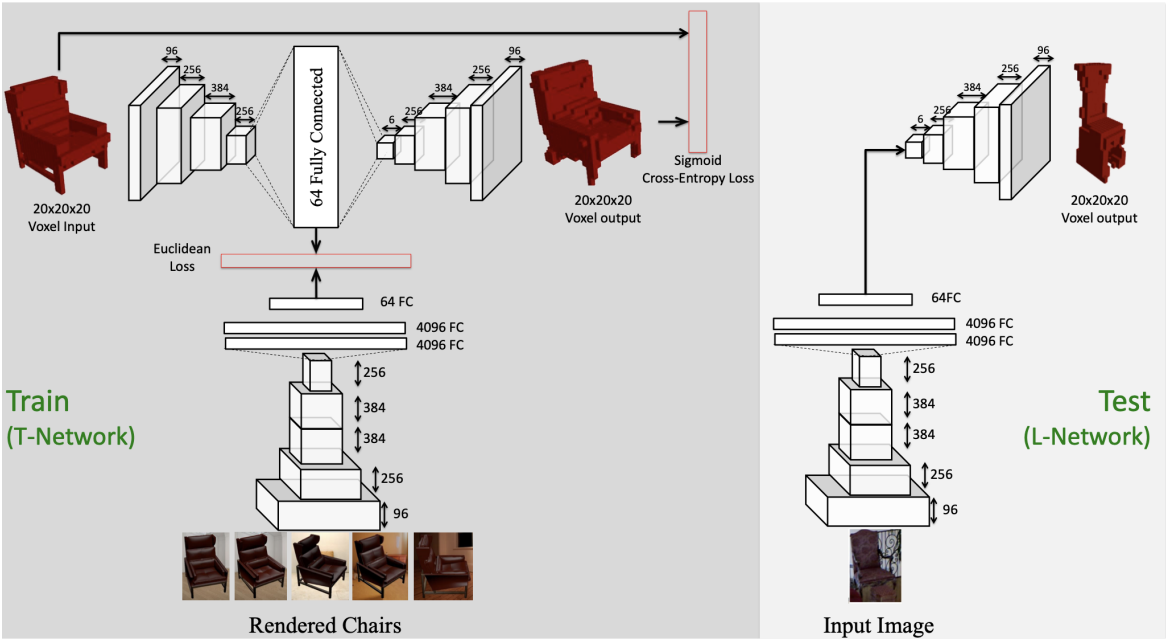


Abbildung 3.3: T-Netz: Zur Trainingszeit nimmt das Netzwerk zwei Eingaben, bestehend aus RGB-Bildern, die unten in ConvNet eingespeist werden und 3D-Voxel-Maps, die links in den Autoencoder gegeben werden, entgegen. Zwei Verlustfunktionen werden während des Trainings angewandt. Zum einen die Kreuzentropie für die Rekonstruktion der Voxel-Ausgabe und die Euklidische Distanz für den 64-D-Vektor in der Mitte. b) L-Netz: Während des Tests wird der Encoder Teil entfernt und das Bild wird als Eingabe verwendet. Das ConvNet erzeugt die Feature-Maps und der Decoder die Voxelrepräsentation.[Girdhar et al., 2016]

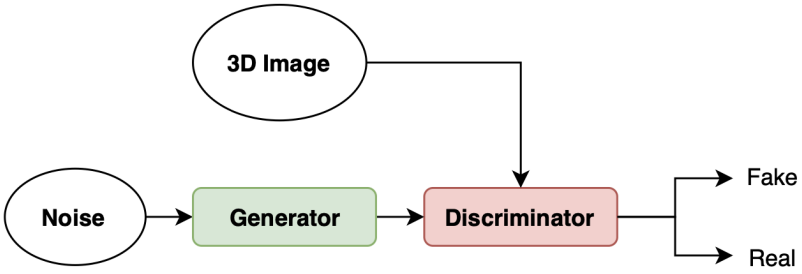


Abbildung 3.4: Aufbau einer GAN Architektur mit Generator und Diskriminator [Volkhonskiy et al., 2019]

GAN, 3D-GAN-VAE

In den letzten Jahren sind Deep-Learning-basierte generative Modelle in den Bereichen der Erzeugung von realistischen Inhalten, wie Audio, Text und Bild, interessant geworden. Generative Adversarial Networks (GANs) und Variational Autoencoders (VAEs) sind die am weitesten verbreiteten. [Han et al., 2019]

GANs bestehen aus zwei künstlichen neuronalen Netzwerken, dem Generator und Diskriminator. Der Generator ist eine lernfähige, differenzierbare Transformationsfunktion eines Rauschens z von einer vorher festgelegten Verteilung $p_{noise}(z)$, die synthetische Objekte erzeugt und versucht, den Diskriminator zu täuschen. Während des Trainings versucht der Generator, möglichst realistische Bilder zu erzeugen, während der Diskriminator reale von synthetischen Bildern korrekt klassifizieren und die Täuschung des Generators erkennen will, was zu einem Nullsummenspiel, siehe Gleichung 3.1, führt. Auf diese Weise können 3D-Geometrien rekonstruiert werden, die dem Trainingset sehr nahe kommen, aber vorher noch nicht gesehen wurden. Weitere Informationen über GANs finden sich in [Goodfellow et al., 2014].

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{3.1}$$

3.1.3 Ausgabe

Die Ausgabedarstellung des Netzes hat einen großen Einfluss auf die Recheneffizienz, die Speichergröße und die Qualität der Rekonstruktion. Zwei Hauptfamilien können

unterschieden werden: Volumetrische- und Oberflächendarstellungen. Die erste umfasst reguläre Voxelgitter, die zweite verwendet Polygon Meshs und Punktwolken.

Abbildung 3.5 zeigt verschiedene Darstellungen desselben 3D-Objekts [Agarwal und Prabhakaran, 2009, Brimkov et al., 2006, Hamidi et al., 2019]. Die erste Repräsentation ist eine Punktwolke, die aus einer Menge von 3D-Datenpunkten $P \in \mathbb{R}^3$ in einem geeigneten Koordinatensystem, wie dem dreidimensionalen kartesischen Koordinatensystem, besteht. Jeder Punkt enthält die (x,y,z)-Komponente und zusätzliche Daten wie z. B. Farbwerte, Intensität oder Reflexionsgrad. Die zweite Repräsentation sind volumetrisch Ausgaben (d. h. Voxel) und wurden in den ersten tiefen neuronalen Netzwerken adaptiert. Die Erweiterung von 2D-Faltungen auf 3D-Faltungen und dementsprechend die Abbildung der Ausgabe des CNN auf ein 3D Occupancy Grid, welches eine Wahrscheinlichkeit für ein freien, belegten oder unbekannten Voxel ermöglicht, wurde mit großem Erfolg von [Maturana und Scherer, 2015] vorgestellt. Die dritte Repräsentation, ein Polygonnetz, ist eine der beliebtesten Darstellungen für 3D-Formen. Die Ausgabe-Repräsentation eines Meshs für neuronale Netze wird vermehrt in aktuellen Ansätzen verfolgt.

Volumetrisch

Volumetrische Darstellungen können als ein Gitter im dreidimensionalen Raum definiert werden, wobei Voxel die jeweiligen Felder darstellen, die nicht homogen gefüllt sind. Die

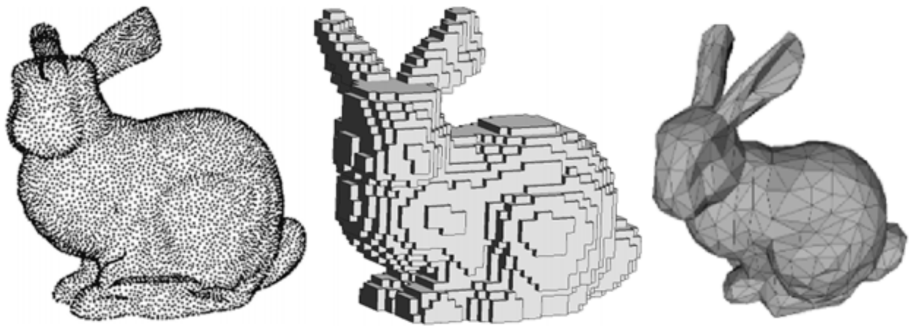


Abbildung 3.5: 3D-Datenformen des Standford-Hasen-Modells: Punktwolke (links), Voxeldarstellung (mitte), Dreiecksmesh (rechts).

Verwendung von 3D-Faltungen und Pooling-Operationen werden mit großem Erfolg in bestehende Deep-Learning-Methoden übernommen.

Unabhängig von der Simplizität der Implementierung sind Voxel-basierte Darstellungen speicherplatzintensiv und wachsen kubisch mit der Gitterauflösung. Die ersten Ansätze nutzen Auflösungen von 32^3 [Choy et al., 2016]. Aufgrund dessen, dass die meisten Informationen an der Oberfläche liegen, speichern diese Verfahren zu viele Informationen im Inneren des Objektes ab. Jeder Voxel speichert seinen Abstand zum nächsten Oberflächenpunkt (SDF) oder seinen binären bzw. probabilistischen Zustand, ob er im dreidimensionalen Gitter besetzt, unbesetzt oder unbekannt ist. Neuere Verfahren nutzen Octrees als Datenstruktur, um den Raum zu partitionieren und Gitter mit unterschiedlichen Voxelgrößen zu ermöglichen. Auf diese Weise können Auflösungen von 256^3 und 512^3 [Wang et al., 2017, Tatarchenko et al., 2017, Wang et al., 2019] erreicht werden.

Abbildung 3.6 und 3.7 illustrieren den Nutzen für die Verwendung von Quad- bzw. Octrees in Voxeldarstellungen. Diese effiziente Datenstruktur führt zu einem quadratischen Speicherwachstum, allerdings muss die spezifische objektabhängige Oktreestruktur während der Laufzeit generiert werden. Die Implementierung solcher Strukturen in neuronalen Netzwerken ist komplexer und schwieriger zu parallelisieren als reguläre 3D-Faltungen. Da die meisten Objektinformationen an die Oberfläche gebunden sind, sind oberflächenbasierte Datenstrukturen effizienter bezüglich des Speicherverbrauchs.

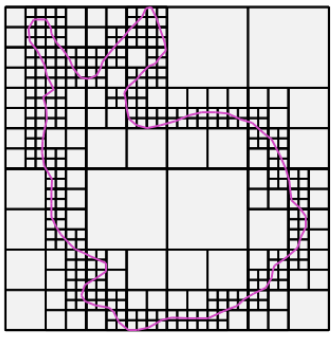


Abbildung 3.6: 2D-Illustration des Standford-Hasens als Quadtree

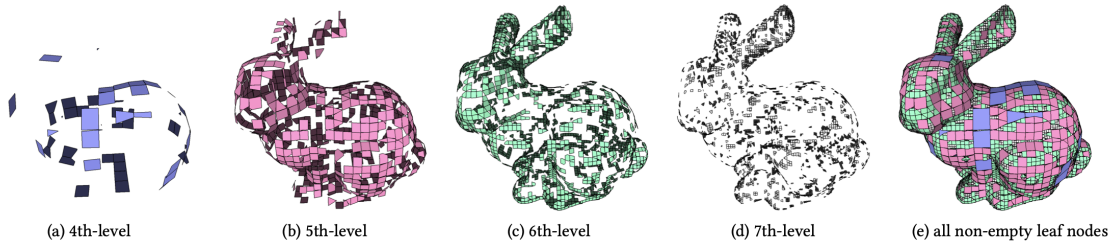


Abbildung 3.7: Octree Level mit planaren Flicken auf den nicht leeren Blattknoten der entsprechenden Ebene [Wang et al., 2019]

Oberflächen Repräsentationen

Im Gegensatz zu volumetrischen Daten sind Oberflächendaten nicht euklidisch und irregulär. Dementsprechend fehlt ein gemeinsames Koordinatensystem und eine geeignete Vektorraumstruktur. Dennoch sind Polygon Meshs und Punktwolken beliebte Ausgabedarstellungen, die jedoch aufgrund ihrer unregelmäßigen Struktur nicht ohne weiteres für klassische 2D- oder 3D-Faltungen in CNNs genutzt werden können.

Für diese geometrische Domäne müssen daher die konventionellen Deep-Learning Operationen, wie Faltung und Pooling speziell angepasst werden. Graph Convolution Networks (GCN) werden häufig im Zusammenhang mit der Polygon Mesh-Erzeugung verwendet. Hier können Oberflächeninformationen und die 3D-Form anhand von Graphen nachgebildet werden. Die Knoten im Graphen bilden auf diese Weise auf die Knoten des Meshs ab. Die spektralen Eigenschaften von Graphen repräsentieren die Nachbarschaftsbeziehungen von Kanten und Knoten eines Polygonnetzes.

Mesh

Polygon Meshs haben im Vergleich zu Voxeln oder Punktwolken mehrere Vorteile. Ihre zugrunde liegende Datenstruktur ist kompakt und kann durch einfache Operationen auf den Vertices leicht transformiert werden, einschließlich Rotation, Translation und Skalierung. Die Größe eines Objektes für 3D-Modelle ist wesentlich kleiner. Eine große Pyramide besteht beispielsweise nur aus vier Eckpunkten und vier Flächen, während Voxel und Punktwolken viele Abtastpunkte auf und in der geometrischen Form benötigen.

Aufgrund ihrer komplexen Definition wurde die Verwendung von Polygonen als geometrische Darstellung erst in letzter Zeit für neuronale Netze übernommen. Die meisten Mesh-Rekonstruktionsansätze nutzen die durch [Kipf und Welling, 2016] eingeführten Graph-Faltungen, um den Gradientenabstieg für das Lernen und die Backpropagation zu ermöglichen. Diese GCNs erzeugen gleichmäßig verteilte Knoten durch einer Verformung eines Template- oder Domänenspezifischen-Netzes [Wang et al., 2018, Pontes et al., 2018]. Die Rekonstruktion von geometrischen Formen mit deformationsbasierten Techniken kann hochauflösende Polygonnetze erzeugen, ist aber an die Topologie des Templates oder Genus-0-Formen gebunden.

Punktwolken

Eine Punktwolke ist eine Menge unstrukturierter (x,y,z) Punkte, die eine höhere Speichereffizienz als volumetrische Ansätze aufweist, indem nur Punkte auf der Oberfläche kodiert werden. Einfache lineare Transformationen an allen Punkten werden verwendet, um die punktbasierte 3D-Darstellung zu drehen und zu skalieren. Mit dem geringen Speicherbedarf ist es eine gute Alternative im Vergleich zu Voxelausgaben, speichert jedoch weder lokale noch globale Nachbarschaftsbeziehungen. Um eine hohe Auflösung

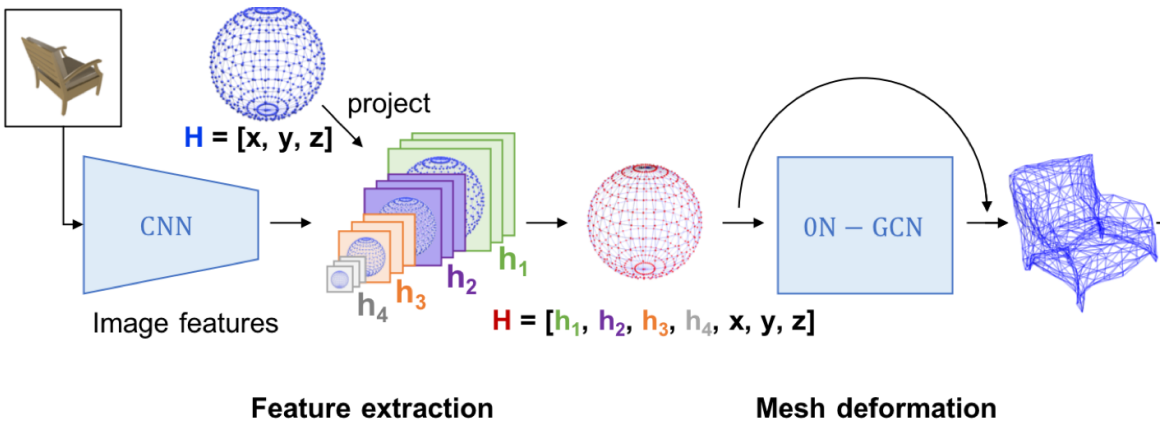


Abbildung 3.8: Mesh-Rekonstruktionsmodul mit drei Hauptkomponenten. Die Feature-Extraktion beschreibt den Prozess durch den Bildmerkmale für jeden Knoten extrahiert werden. Die Mesh Deformierung nutzt eine Reihe von Graph-Faltungen auf diesen Knoten und seinen Merkmalen, um die finale geometrische Form zu erstellen. [Smith et al., 2019]

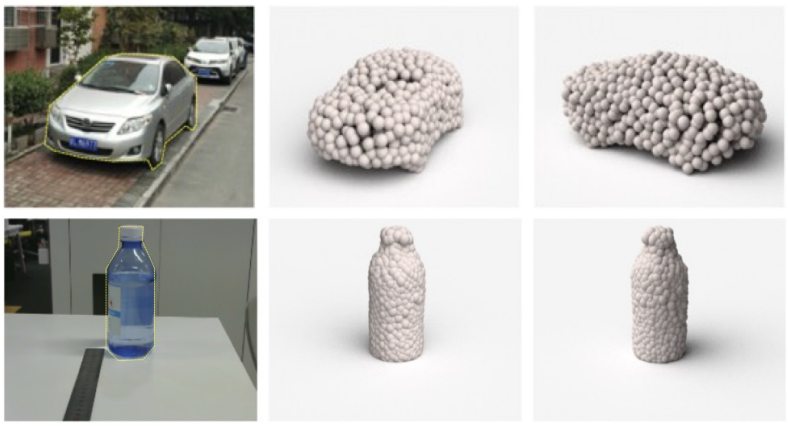


Abbildung 3.9: Aus dem 2D-Bild wird eine 3D-Punktwolke erstellt. Jeder Punkt ist als kleine Sphäre visualisiert. Die rekonstruierte Punktwolke ist aus zwei Blickwinkeln (0° und 90° entlang des Azimut) gezeigt. Eine Segmentierungsmaske wird verwendet, um das Bild von dem Hintergrund abzugrenzen. [Fan et al., 2017]

zu erreichen, muss die Abtastdichte über die gesamte Fläche erhöht werden. Die gebräuchlichsten Netzwerkarchitekturen zur Erzeugung einer Punktmenge aus einem 2D-Bild verwenden ein Encoder-Decoder-Modell. [Fan et al., 2017] verwendet zwei parallele Prädiktorzweige. Einen vollständig verschalteten Fully Connected (FC) Zweig und einen Faltungszweig, wobei die jeweiligen Ergebnisse am Ende zusammengeführt werden. Der FC-Zweig erzeugt jeden Punkt unabhängig voneinander und kann eine hohe Detailgenauigkeit erzeugen, während der Faltungszweig gut für die Vorhersage großer und glatter geometrischer Oberflächen geeignet ist. Wie bereits hervorgehoben, sind Punktwolken unregelmäßige Strukturen und enthalten keine Oberflächen. Die Abbildung von Texturen muss mit Postprocessing-Methoden wie den Marching Cubes, der Poissionsrekonstruktion oder dem Ball Pivoting erfolgen.

3.2 Überblick über ausgewählte Verfahren

Ungeachtet der Vielzahl von Ansätzen, ein 3D-Modell aus einem 2D-Bild zu rekonstruieren, besteht die überwiegende Mehrheit der Architekturen aus einer Encoder-Decoder Architektur. Um ein breites Spektrum an Architekturen abzudecken, werden PSG, OGN

Methode	Eingabe		Ausgabe		Training	
	Training	Testing	Typ	Auflösung	Supervision	Netzwerk
[1]	$n = 1$ RGB, 3D GT	1 RGB	Point Set	1024	3D	Multibranch Encoder-Decoder Block
[2]	$n \geq 1$ RGB, 3D GT	1 RGB	Volumetric (Voxel)	32^3 - 512^3	3D	Encoder + OGN
[3]	$n = 1$ RGB, 3D GT	1 RGB	Mesh	2466	3D	Encoder + GCN

Tabelle 3.1: Taxonomy of state of the Art Networks

und P2M aufgrund ihrer Ausgabe-Repräsentation und Netzwerk-Diversität vorgestellt.

1. A Point Set Generation Network for 3D Object Reconstruction from a Single Image (PSG) [Fan et al., 2017]
2. Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs (OGN) [Tatarchenko et al., 2017]
3. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images [Wang et al., 2018]

3.3 Point Set Generation Network (PSG)

Das Point Set Generation Network war eines der ersten CNNs, dass gute Ergebnisse bei der Rekonstruktion (mehrerer) Punktwolken aus einem einzigen Bild lieferte. [Fan et al., 2017] generiert mit seinem Ansatz aus einem realen oder synthetischen RGB-Bild eine Menge von 1024 Punkten. Punktwolken eignen sich insbesondere für Formen mit Genus > 0 und feinen Details.

3.3.1 Netzwerkarchitektur

Abbildung 3.10 zeigt in einem groben Überblick die Evolution in der Entwicklung des PSG. In der `vanilla` Version besteht das neuronale Netzwerk aus einem Encoder, welcher mehrere Merkmalsvektoren aus dem Eingabe-Bild erstellt. Der Zufallsvektor, wird während des Trainings mit dem Bild konkateniert, um eine leicht veränderte Eingabe zu generieren und eine Überanpassung an die Trainingsdaten zu verhindern. In Conditional Generative Adversarial Nets (cGAN), wird ein solcher Vektor genutzt, um die Konstruktion von neuen Ausgaben in eine bestimmte Richtung zu lenken. Der Prädiktor besteht aus vollständig verschalteten Schichten und erzeugt einen Vektor der Größe 256×3.

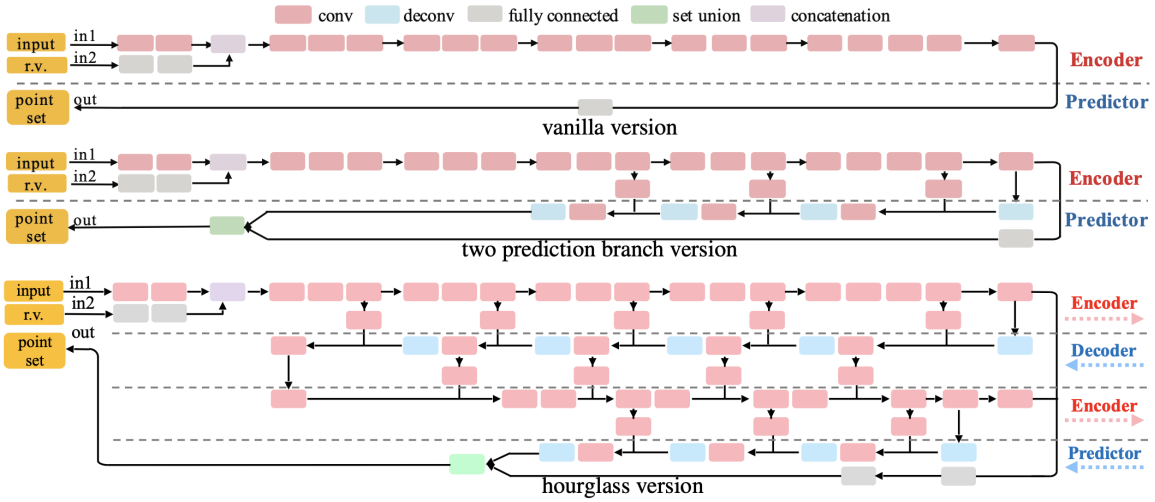


Abbildung 3.10: Überblick über die Evolution der Netzwerkarchitektur des PSG-Netzwerks, mit der `vanilla` Version, der `two prediction branch` Version und der `hourglass` Version.

Die `two prediction branch` Version erweitert den Prädiktor des ersteren Netzwerks, indem sie einen zusätzlichen parallelen Faltungszweig nutzt, der zusätzlich 768 Punkte generiert. Im Anschluss werden diese mit den 256 Punkten der Ausgabe der vollständig verschalteten Schicht vereinigt um eine Punktwolke der Größe 1024×3 zu generieren.

Die letzte Version nutzt ein sogenanntes `stacked hourglasses` für eine bessere Performance. Dabei werden zwischen den bisherigen Encoder und Prädiktor zwei weitere Schichten eingeschoben, die ebenfalls Faltungen, inverse Faltungen und Shortcut Verbindungen besitzen. Je tiefer Netzwerke gehen, desto wichtiger werden diese Shortcut Verbindungen, welche das Problem des verschwindenden Gradienten verringern.

Im Folgenden wird die Netzwerkarchitektur der zweiten Version im Detail erläutert, da sie sehr anschaulich die Kombination von Faltungsschichten und vollständig verschalteten Schichten aufzeigt.

In Abbildung 3.11 ist zunächst die Eingabe, welche aus der Segmentierungsmaske und dem RGB-Bild besteht, zu sehen. Die Konkatentation beider Eingaben ergibt einen $192 \times 256 \times 4$ Vektor. Aus diesem Vektor werden durch mehrere Faltungsschichten Bildmerkmale extrahiert. Während die ursprüngliche Bilddimension in den roten Faltungen durch eine Filterschrittweite von 2 halbiert wird, verdoppelt sich die entsprechende Anzahl an Filtern. Aus Performance Gründen findet die Dimensionsreduktion mit Faltungen und nicht mit Pooling Operationen statt. Die Kernelgröße beläuft sich in den meisten Faltungen auf eine 3×3 Matrix. Nach der letzten Faltungsschicht wurde die ursprüngliche Repräsentation des Eingabebildes auf eine kompakte niederdimensionale Darstellung der Dimension $3 \times 4 \times 512$ heruntergebrochen. Dieser Tensor bildet nun die Eingabe sowohl für den ersten Zweig, welcher aus drei vollständig verschalteten Schichten mit $2048, 1024$ und 768 Neuronen besteht, als auch für den zweiten Zweig, der auf inversen Faltungsschichten aufbaut.

Zweig 1: Vollständig verschaltete Schichten

Der kompakte Vektor wird mithilfe von drei vollständig verschalteten Schichten auf einen 768×1 Vektor reduziert und anschließend in die Form 256×3 transformiert, welche den ersten 256 Koordinaten entspricht.

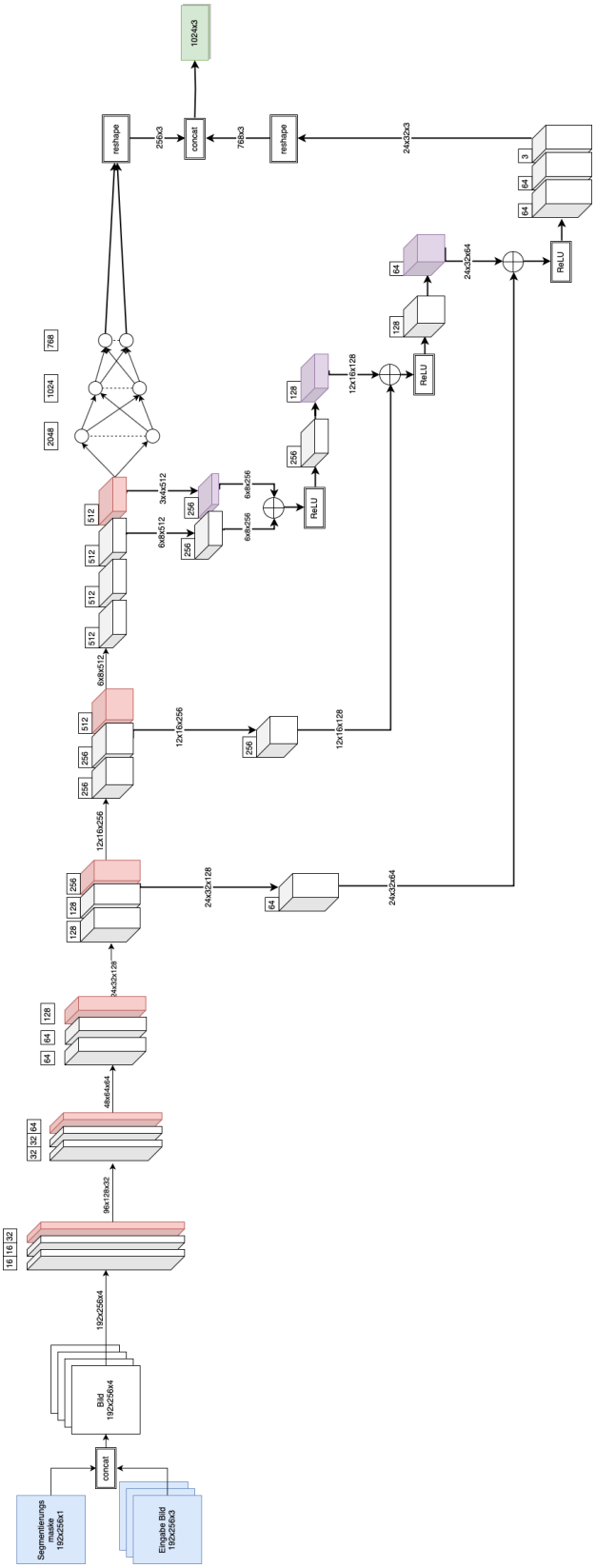


Abbildung 3.11: Detail-Architektur des PSG. Blau markiert ist sowohl die Eingabe mit der Segmentierungsmaske und dem RGB-Bild. Rot markiert sind Faltungsschichten mit Schrittweite 2. Lila markiert sind inverse Faltungschichten. Die Ausgabe als 1024×3 Vektor ist grün markiert.

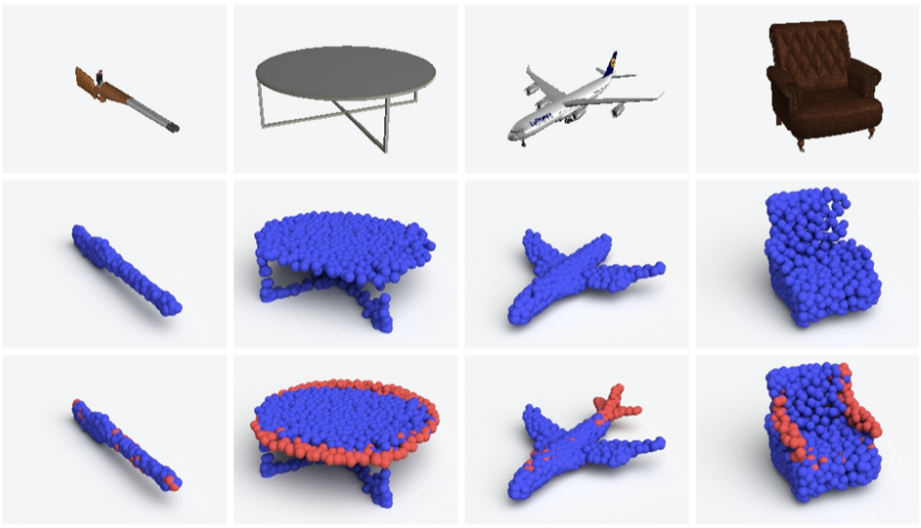


Abbildung 3.12: Visualisierung von Punkten, die durch die vollständig verschalteten Schichten (rot) und den Faltungsschichten erzeugt wurden (blau).

Zweig 2: Inverse Faltungen

Zur Erzeugung der restlichen 768 Koordinaten wird ein Vektor der Dimension $24 \times 32 \times 3$ benötigt. Inverse Faltungen, die in Abbildung 3.11 in lila dargestellt sind, verdoppeln die Dimensionen des Vektors und halbieren die Anzahl der Schichten der Feature-Maps.

Tiefe neuronale Netzwerke besitzen ein sehr großes rezeptives Feld und rufen das Problem von verschwindenden Gradienten hervor. Aus diesem Grund werden aus den Schichten $11, 14$ und 18 Shortcut Verbindungen hinzugefügt. Die Ergebnisse dieser Faltungen werden mit dem aktuellen Tensor elementweise aufsummiert und mit der Rectifier Linear Unit (ReLU) transformiert. Aus den erzeugten 768 Punkten aus diesem Zweig und den 256 Punkten aus dem ersten Zweig entsteht die finale Punktmenge.

Abbildung 3.12 verdeutlicht den Vorteil einer Kombination der zwei vorgestellten Zweige. Feine Details der endgültigen Punktwolke werden durch die voll verschalteten Schichten erzeugt, während glatte und lange Formen aus den Faltungschichten generiert werden. Das Netzwerk ist zudem in der Lage, mehrere plausible Punktwolken zu erzeugen, indem der Zufallsvektor r als weiterer Parameter zu der Segmentierungsmaske und dem RGB-Bild gegeben wird.

category		plane	bench	cabinet	car	chair	monitor	lamp	speaker	firearm	couch	table	cellphone	watercraft	mean
PSG	1 view	**0.601**	**0.550**	0.771	0.831	0.544	0.552	**0.462**	**0.7337**	**0.604**	**0.708**	**0.606**	0.749	**0.611**	**0.640**
3D-R2N2	1 view	0.513	0.421	0.716	0.798	0.466	0.468	0.381	0.662	0.544	0.628	0.513	0.661	0.513	0.560
	3 views	0.549	0.502	0.763	0.829	0.533	0.545	0.415	0.708	0.593	0.690	0.564	0.732	0.596	0.617
	5 views	0.561	0.527	**0.772**	**0.836**	**0.550**	**0.565**	0.421	0.717	0.600	0.706	0.580	**0.754**	0.610	0.631

Tabelle 3.2: Vergleich der Güte der 3D-Rekonstruktion bezüglich der IoU zwischen dem PSG-Netzwerk und dem 3D-R2N2-Netzwerk auf dem ShapeNet-Testdatensatz.

3.3.2 Vergleich mit State of the Art Verfahren

In diesem Abschnitt wird das PSG-Netzwerk durch die IoU mit dem damaligen State of the Art Netzwerk 3D-R2N2 verglichen. Beide Verfahren wurden laut [Fan et al., 2017] auf dem 3D-R2N2-Datensatz mit einem 80% - 20% Split trainiert. Für die Auswertung 3.2 werden die aus den Papern gelieferten Werte übernommen und eingeordnet. Es wird deutlich, dass das PSG-Netzwerk aus einem einzelnen Bild in 8 von 13 Fällen bessere Ergebnisse als das 3D-R2N2-Netzwerk mit einer Eingabe von 5 RGB-Bildern erzielt.

3.4 Octree Generation Network (OGN)

Das Octree Generation Netzwerk kann ebenfalls in die Kategorie der tiefen neuronalen Netzwerke eingeordnet werden. Der Decoder besteht aus einer Reihe von Faltungsschichten und produziert ein volumetrisches 3D-Modell. Im Vergleich zu typischen Voxel-basierten Repräsentationen ist die Octree basierte Darstellung rechen- und speichereffizient. Die genauen Vorteile wurden bereits in Abschnitt 3.1.3 erläutert. Zusammenfassend lässt sich sagen, dass die Verwendung einer Datenstruktur mit adaptiver Zellgröße bezüglich eines Voxels eine performante Kodierung sehr großer als auch sehr detaillierter Oberflächen zulässt. In den ersten Phasen der Dekodierung werden zunächst große Flächen erzeugt, die anschließend genauer untersucht werden, sodass es in nachfolgenden Phasen zu einer Erhöhung der Auflösung kommt. Das Prinzip von einer groben Struktur in eine feine überzugehen wird auch im Pixel2Mesh Netzwerk verwendet. Auf diese Weise können Gittergrößen von 256^3 oder 512^3 erreicht werden.

Die Idee ist es den gesamten Raum initial als eine Zelle bzw. Voxel darzustellen, die in rekursiven Schritten jeweils in 8 Oktanten unterteilt wird. Eine Baumstruktur wird genutzt, um diese Aufteilung technisch abzubilden. Eine Zelle wird zu einem Blatt im Baum, wenn jeder Voxel in der Zelle den gleichen Funktionswert besitzt. Die Menge aller Zellen auf einer Auflösungsstufe gehören einem Octree Level an. Wenn eine Zelle geteilt wird, speichert sie einen Zeiger auf seine 8 Kindknoten. Die Zugriffszeit auf ein

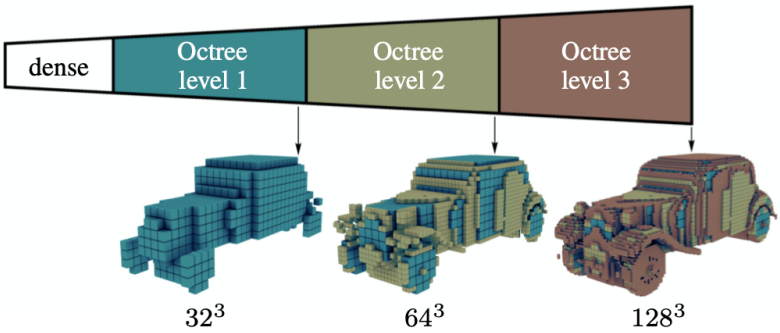


Abbildung 3.13: Die volumetrische Ausgabe eines Fahrzeuges ist anhand von drei Leveln einer Octree-Datenstruktur abgebildet. Die Auflösung wird inkrementell von 32^3 bis 128^3 erhöht.

spezielles Blatt im Baum würde mit einer naiven Implementierung der Baumstruktur zu einer linearen Zugriffszeit führen. [Tatarchenko et al., 2017] hat dieses Problem durch die Implementierung eines hashbasierten Ansatzes überwunden.

Die Nutzung von eine Octree-Datenstruktur zur Vorhersage, welche Regionen des Objekts hochauflösende Oberflächeninformationen beinhalten, ist mit den Kosten für die Konstruktion eines Baumes und seines Inhaltes verbunden. Sowohl die Werte der einzelnen Zellen als auch die Aufteilung dieses Raumes sind sehr objektspezifisch. Ansätze einer vordefinierten Baumstruktur für Objektkategorien wurden u. a. von [Riegler et al., 2017] genutzt.

3.4.1 Netzwerkarchitektur

Die Netzwerkarchitektur der Autoren des OGN ([Tatarchenko et al., 2017]) konzentriert sich primär auf den Decoder zur Generierung von 3D-Formen und verweist für die Encoder-Struktur auf bekannte CNN-Architekturen wie AlexNet, VGG oder ResNet zur 2D-Feature-Extraktion, sowie für Voxel-basierte Eingaben auf [Choy et al., 2016, Girdhar et al., 2016, Häne et al., 2017]. Um die vollständige Rekonstruktion aus einem RGB-Bild zu einer Octree-basierten Voxel-Repräsentation zu verstehen, werden jedoch nachfolgend beide Teile in Kürze skizziert.

Ähnlich wie im bereits vorangegangenen Abschnitt des Point Set Generation Netzwerks, wird in einem ersten Schritt eine kompakte Feature-Map des Eingabe-Bildes erstellt. Die Bildmerkmalsextraktion und Dimensionsreduktion findet durch normale 2D-Faltungen statt. Anschließend folgen eine Reihe von inversen 3D-Faltungen, die eine initiale Voxel-Repräsentation mit der Dimensionen 32^3 erstellt. Diese Auflösung kann beliebig gewählt werden, weil sie als Eingabe für das OGN dient. Jeder der 32768 Voxel auf dem Gitter besitzt einen Wert, der angibt, ob dieser Voxel belegt (1) oder frei (0) ist. Der Name von `Occupancy Grid Maps` stammt genau aus dieser Zuordnung.

Die Faltungen auf einem Octree spiegeln einen neuartigen Ansatz zur Generierung eines 3D-Modells wieder. Im Kern fungieren sie jedoch gleichartig zu Standard 3D-Faltungen auf Voxelgittern. Jede Zelle mit den kartesischen Koordinaten $\boldsymbol{x} = (x,y,z)$ auf Level l wird als Index-Wertepaar (m,v) beschrieben, wobei m der berechnete Hash

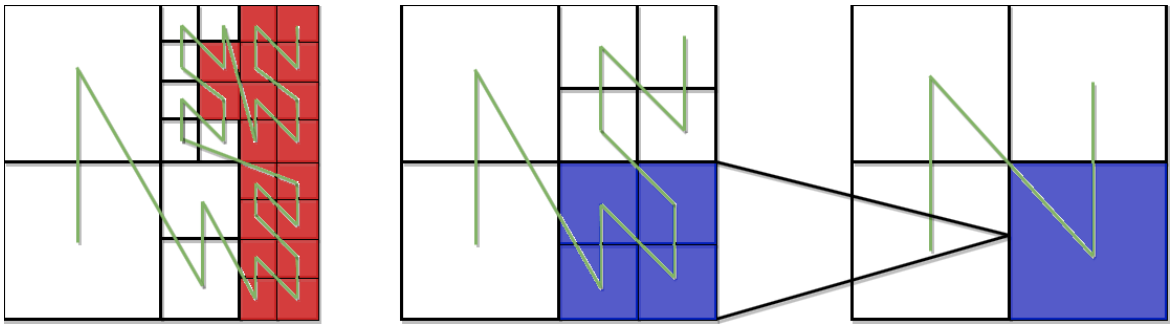


Abbildung 3.14: Beispiel der Z-Kurven-Abbildungsfunktion für ein Quadtree, sowie ein Pooling bzw. Unpooling zwischen zwei Leveln. Diese lineare Ordnung ist wichtig wird für die Hashfunktion genutzt, um schnell die Werte benachbarter Zellen zu finden. Das Unpooling wird hier anhand der Teilung einer Zelle in 4 Sub-Zellen in blau dargestellt.

ist. Dazu wird die Z-Kurve genutzt, welche nachbarschaftserhaltende Eigenschaften besitzt und alle Voxel aus dem dreidimensionalen Raum in eine lineare Ordnung bringt [Gargantini, 1982]. Nachbar-Voxel mit ihrem Feature-Vektor v können auf diese Weise für eine Faltung schnell ermittelt werden. Abbildung 3.14 zeigt die Bildung einer Z-Kurve durch einen Quadtree sowie die Pooling bzw. Unpooling Operation zwischen zwei Leveln.

Tatarchenko et. al. definiert den Index Hash m als

$$m = Z(\boldsymbol{x},l), \tag{3.2}$$

und den Octree O als Menge von Paaren

$$O = \{(m, v)\}. \tag{3.3}$$

Aus dem dichten Voxelgitter, dass mit einer Auflösung von 32^3 als Eingabe für das OGN dient, wird zunächst eine Hashtabelle mit Feature-Vektoren für jeden Voxel, anhand von herkömmlichen 3D-Faltungen und inversen Faltungen, für Level 1 generiert. Jeder Feature-Vektor besitzt die Dimension $d_1 \times d_2 \times d_3 \times c$, zusammengesetzt aus den räumlichen Koordinaten und der Anzahl der Kanäle. Abbildung 3.15 zeigt der Einfachheit halber einen einzelnen Block zur Vorhersage der Feature-Vektoren der

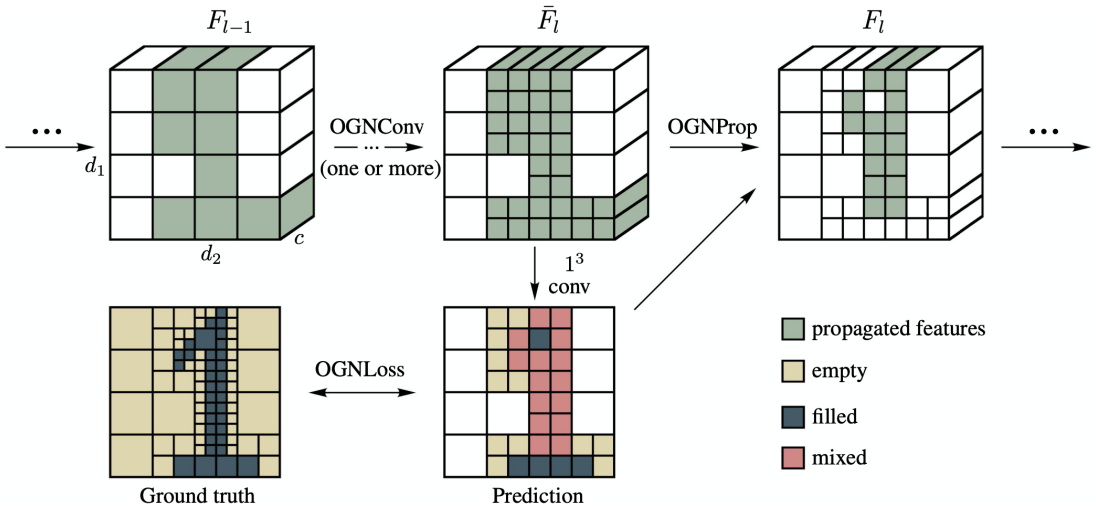


Abbildung 3.15: Einzelner Block eines OGN, der Einfachheit halber als 2D-Quadtree dargestellt. Nachdem die Merkmale in der Hashtabelle F_{l-1} des vorherigen Blocks mit Gewichtsmatrizen gefaltet wurden, werden direkt die Belegungswerte der Voxel auf Level l vorhergesagt. Nur Features aus `mixed` Zellen werden weiter propagiert und bilden die Ausgabe dieses Blocks.

Hashtabelle der Ebene l aus der Hashtabelle der Ebene $l-1$ als Quadtree. Aus der temporären Hashtabelle $\overline{F_l}$ wird durch eine 1^3-Faltung gefolgt von einer Drei-Wege-Softmax-Funktion eine Klassifizierung jeder einzelnen Zelle erstellt. Dabei kann eine Zelle einen von drei Zuständen einnehmen:

- **empty** - Zustand ist nicht im Octree gespeichert.
- **filled** - Zustand ist im Octree gespeichert, wird aber in nachfolgenden Faltungen nicht weiter berücksichtigt.
- **mixed** - Zustand ist im Octree gespeichert und wird für nachfolgende Faltungen in Betracht gezogen, um die Auflösung weiter zu erhöhen.

Zum Training des Klassifikators wird die Kreuzentropie-Fehlerfunktion zwischen dem GT-Octree und dem vorhergesagten Octrees minimiert. Einzelheiten hierzu können in [Tatarchenko et al., 2017] nachgelesen werden. Die Features der Zellen, die einen `mixed` Zustand besitzen, werden an die finale Hashtabelle für Level l weiter propagiert, die wiederum als Eingabe für den nächsten Block dienen.

category		plane	bench	cabinet	car	chair	monitor	lamp	speaker	firearm	couch	table	cellphone	watercraft	mean
3D-R2N2	1 view	0.513	0.421	0.716	0.798	0.466	0.468	0.381	0.662	0.544	0.628	0.513	0.661	0.513	0.560
PSG	1 view	**0.601**	**0.550**	**0.771**	**0.831**	**0.544**	**0.552**	**0.462**	**0.7337**	**0.604**	**0.708**	**0.606**	**0.749**	0.611	**0.640**
Dense	1 view	0.570	0.481	0.747	0.828	0.481	0.509	0.371	0.650	0.576	0.668	0.545	0.698	0.550	0.590
OGN	1 view	0.587	0.481	0.729	0.816	0.483	0.502	0.398	0.63**7**	0.593	0.646	0.536	0.702	**0.632**	0.596

Tabelle 3.3: Vergleich der IoU zwischen dem OGN, dem 3D-R2N2-Netzwerk und dem PSG-Netzwerk auf dem ShapeNet-Testdatensatz.

Diese Blöcke wiederholen sich solange, bis die gewünschte Auflösung erreicht wurde. Im Vergleich zu konventionellen Voxel-basierten neuronalen Netzwerken können weitaus detailliertere Modelle rekonstruiert werden.

3.4.2 Vergleich mit State of the Art Verfahren

Wie auch bei dem PSG-Netzwerk wird in diesem Abschnitt die Performance des OGN mit bereits bekannten Netzwerken verglichen. Die IoU-Werte für das auf dem 3D-R2N2-Datensatz trainierte OGN werden ebenfalls aus dem offiziellen Paper entnommen. Für einen besseren Vergleich werden in der Tabelle 3.3 die Werte des MultiView-Verfahrens von 3D-R2N2 entfernt und dafür IoU-Werte von einem Netzwerk, dass einen Decoder mit der gleichen Konfiguration wie das des OGN besitzt und nicht auf Octrees arbeitet hinzugefügt. Dieses `Dense` Netzwerk gibt ein reguläres Voxel-Gitter mit mit einer Auflösung 32^3 aus. Diese speicherintensivere Ausgabe besitzt mehr, aber eventuell redundante Informationen, die im Folgenden als Basislinie zum Vergleich mit dem OGN (Level 1 Ausgabe 32^3) herangezogen wird. Deutlich wird, dass das OGN minimal besser als das `Dense` Netzwerk abschneidet, obwohl es durch den Octree in der Regel weniger Informationen abspeichert, jedoch nicht an die IoU-Werte des PSG-Netzwerk herankommt.

3.5 Pixel2Mesh (P2M)

Das Pixel2Mesh-Netzwerk generiert durch eine Encoder-Decoder-Struktur Dreiecksmeshs aus einem 2D-Bild. Das zugrundeliegende Prinzip, um löcherfreie Polygon Meshs zu erzeugen, nutzt ein GCN aus und deformiert ein Ellipsoid in mehreren Schritten zu der gewünschten Form.

3.5.1 Netzwerkarchitektur

In dem Grundlagenkapitel wurden bereits Graph-basierte neuronale Netzwerke eingeführt, welche in den folgenden Abschnitten bezüglich Pixel2Mesh genauer erläutert werden. Dabei wird vor allem auf die technische Umsetzung zur Rekonstruktion eines 3D-Modells aus einem RGB-Bild Wert gelegt. Die genaue Implementierung aller Teilarchitekturen kann in [Wang et al., 2018] entnommen werden.

Die Netzwerkarchitektur von Pixel2Mesh kann in kleine Teilarchitekturen herunter gebrochen werden. Für die Encoder-Struktur wurde ein CNN, welches an die VGG-16 (Visual Geometry Group) Architektur angelehnt ist, genutzt [Simonyan und Zisserman, 2014]. Die Decoder-Struktur ist in drei Deformationsblöcke aufgeteilt, die aus einer Reihe von Faltungs-, Projektions- und Pooling-Schichten bestehen. Abbildung 3.16 gibt einen groben Überblick über die Architektur. In den folgenden Abschnitten wird auf jeden dieser Teile genauer eingegangen.

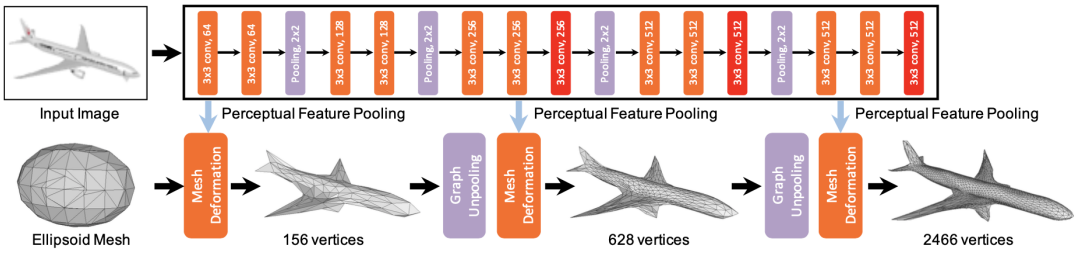


Abbildung 3.16: Das kaskadierende Mesh-Deformations-Netzwerk beinhaltet drei sequentiell geschaltete Mesh-Deformationsblöcke. Jeder Block erhöht die Auflösung des Meshs und bestimmt neue Knotenkoordinaten. Diese werden anschließend genutzt, um neue Bildmerkmale zu extrahieren, welche wiederum für die Faltungsschichten als Eingabe dienen. [Wang et al., 2018]

Encoder - CNN Bildmerkmal Extraktion

Wie auch bei der Klassifizierung oder der Segmentierung von Objekten in Bildern werden auch bei Pixel2Mesh in der ersten Phase Merkmale durch einen CNN-Encoder extrahiert. Diese Merkmale, in Abbildung 3.17 `img_feat` genannt, bestehen aus den vier Ausgaben der Schichten 8, 11, 14 und 18. Eine Konkatenation des Eingabebildes mit einem Tiefenbild ist optional und hat keinerlei Einfluss auf die restlichen Komponenten des CNN. Als Eingabe dient ein RGB-D-Bild mit den Maßen $224 \times 224 \times 4$. Anhand von 18 Faltungsschichten werden mehrere nieder-dimensionale Repräsentationen erzeugt. Im Detail handelt es sich um normale 2D-Faltungen mit Kernel Größen von 3×3 und 5×5. Dabei wird die Rectified Linear Unit (ReLU) als Aktivierungsfunktion genutzt. Die Anzahl der Kernelfilter verdoppelt sich circa alle drei Schichten, indem die Schrittweite des Filters von eins auf zwei gesetzt wird. Zur Vermeidung von Überanpassung werden die Parameter mit der L2–Norm regularisiert. Die rot markierten Schichten in Abbildung 3.17 kennzeichnen die Filter mit Schrittweite 2, die zu einer Dimensionsreduktion der eigenen Eingabe und einer Verdoppelung der Anzahl der Kernelfilter führt. Die finale Liste von Bildmerkmalen (img_feat) wird in der Graph-Projektions-Schicht zur Erzeugung der Features für einen Knoten genutzt.

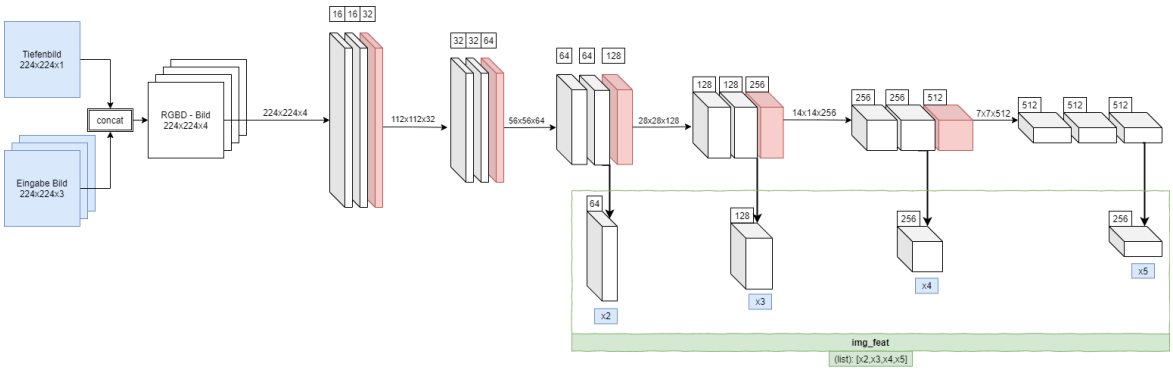
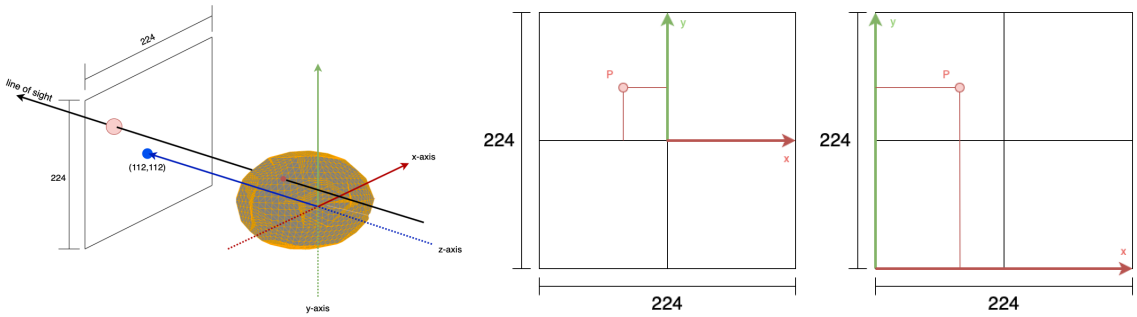


Abbildung 3.17: CNN-Encoder-Architektur des Pixel2Mesh-Netzwerks. Aus der Konkatenation des Eingabebildes und der Tiefendaten werden anhand von 2D-Faltungen mehrdimensionale Feature-Maps erstellt.



(a) (b) (c)

Abbildung 3.18: Projektion eines Knotens auf das Eingabebild

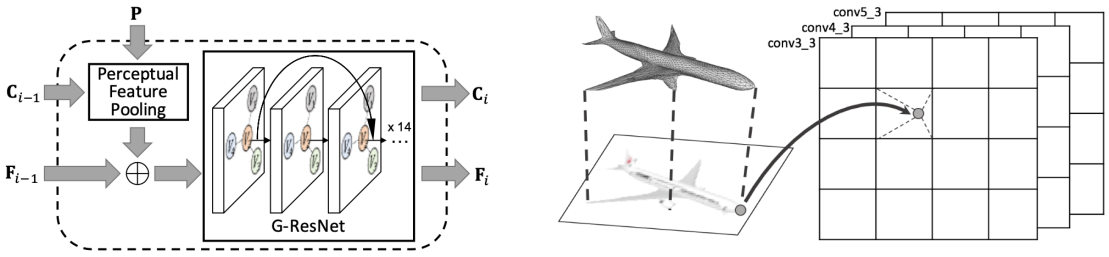
Decoder

Der Decoder besteht aus der Graph Projection Phase, der Graph Convolution Phase und der Graph Unpooling Phase, die im Folgenden genauer untersucht werden.

Graph Projection

Damit auf Graphen, genauer den Knoten eines Graphen, Faltungen ausgeführt werden können, müssen zu diesen zunächst Features hinzugefügt werden. Im vorherigen Schritt wurden bereits für alle Pixel des Eingabebildes Merkmale extrahiert, jedoch müssen diese noch richtig assoziiert werden. Dafür ist zunächst eine Projektion eines Knotens des (Ellipsoiden) Meshs zurück auf das Eingabebild notwendig. Den korrespondierenden 2D-Pixel aus einem 3D-Punkt zu finden ist rein geometrischer Natur und wird durch das perspektivische Projektionsverfahren umgesetzt. Die Transformation eines Punktes aus der Weltposition in ihre endgültige Rasterposition (ihre Position im Bild in Form von Pixelkoordinaten) ist in Abbildung 3.18a skizziert. Der Knoten des Ellipsoiden wird auf der z-Achse, die den Betrachtungswinkel darstellt, auf ein 224×224 Raster, welches den Maßen des Eingabebildes entspricht, projiziert.

Abbildung 3.18b und 3.18c zeigen den geometrischen Vorgang einer perspektivischen Projektion. Da das Eingabebild und das Objekt die gleichen Kameraparameter teilen, wird zunächst x und y durch z geteilt. Diese perspektivische oder auch Z-Teilung genannt, spiegelt eine der fundamentalen Relationen in der Computer-Grafik dar. Anschließend werden die entsprechenden Komponenten auf die neue Breite des Rasters durch Multiplikation der Breite bzw. Höhe skaliert und anhand des unterschiedlichen



(a) Mesh Deformation Block **(b)** Feature Pooling

Abbildung 3.19: (a) zeigt die Konkatenation von eigenen und erhaltenen Features aus dem Perceptual Feature Pooling, sowie Shortcut-Connections in den Faltungsschichten, um das rezeptive Feld und die dementsprechend eingeschränkte Nachbarschaft zu erweitern. (b) verdeutlicht die Generierung der Features aus den Knotenkoordinaten, dem 2D-Bild und dem `img_feat` Tensor [Wang et al., 2018]

Koordinatenursprungs durch die Hälfte der Rasterbreite bzw. Rasterhöhe verschoben. Als letzter Schritt wird jeder Punkt, der noch außerhalb des Rasters liegt, durch die *min* und *max* Funktion auf den Rand des Rasters gesetzt. Die $P_{raster}y$ und $P_{raster}x$ Komponenten werden wie folgt berechnet:

$$\begin{aligned} P_{raster}y &= min(max(R_h * \frac{-P_{object}y}{-P_{object}z} + (R_h/2), 0), R_h) \\ P_{raster}x &= min(max(R_w * \frac{P_{object}x}{-P_{object}z} + (R_w/2), 0), R_w) \end{aligned} \tag{3.4}$$

mit: R_h = Pixel-Höhe des Rasters (Eingabebild)
R_w = Pixel-Breite des Rasters (Eingabebild)

Nachdem der exakte 2D-Pixel für den korrespondierenden 3D-Knoten berechnet wurde, werden aus dem `img_feat` für jeden der vier angrenzenden Pixel die berechneten Features entnommen und durch eine bilineare Interpolation wird der Zwischenwert gebildet. Die Ausgabe der Graph-Projektions-Schicht ist dementsprechend ein 963 (3 + 64 + 128 + 256 + 512) dimensionaler Vektor für jeden Knoten.

Graph Convolution

Die Faltung auf Knoten eines Graphen bildet das Kernstück eines jeden GCN und wurde bereits in Abschnitt 2.1 eingeführt. Die gleichen Prinzipien gelten auch für die

Implementierung bei Pixel2Mesh, nur dass hier eine Hop-Weite von 1 genutzt wird. Es werden nur die unmittelbaren Nachbarn für einen Faltungsschritt in Betracht gezogen. Da jeder Knoten im Ellipsoid den Ausgangsgrad von 6 hat, muss zudem nicht über die Anzahl der Ausgangskanten normiert werden. Gleichung 4.1 zeigt die Faltung auf einem Knoten p durch Aggregation sowohl der eigenen Features, als auch der Features der Nachbarknoten.

$$f_p^{l+1} = w_0 f_p^l + \sum_{q \in N(p)} w_1 f_q^l \tag{3.5}$$

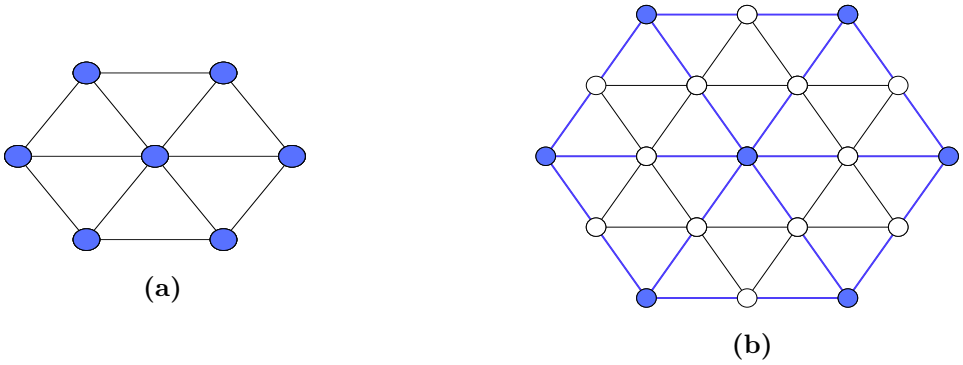
mit: f_p^l, f_p^{l+1} = Feature-Vektor des Knoten p vor und nach einer Faltung
$N(p)$ = Nachbarknoten von p
w_0, w_1 = Gewichtsmatrizen für die Eigen- und die Nachbar-Knotenmatrix

Tiefe Netzwerke sind wegen des verschwindenden Gradientenproblems schwer zu trainieren [Zhang et al., 2018], zudem werden nur unmittelbare Nachbarknoteninformationen zur Faltung in Betracht gezogen. Um diese Probleme zu vermeiden, werden Shortcut Verbindungen genutzt um Ergebnisse einer Schicht mit den Ergebnissen der übernächsten Schicht zu addieren. In diesem Netzwerk gehen beide Ergebnisse mit jeweils einem Gewicht von 0,5 in das finale Ergebnis ein. Schematisch ist dies in 3.19a zu sehen. Auf diese Weise wird das rezeptive Feld eines einzelnen Knotens erhöht.

Der Graph-Faltungsblock besitzt insgesamt 14 Schichten. Die erste Faltungsschicht bricht den 963 dimensionalen Vektor auf 256 Dimensionen herunter, auf denen in 12 konsekutiven Schichten weitere Faltungen stattfinden. Die letzte Schicht bricht diesen wiederum auf einen drei dimensionalen Vektor herunter, der für die (x,y,z)-Komponenten eines Knotens stehen. Jede einzelne Graph Faltung ist mit einer Verformung des Meshs gleichzusetzen, da die (x,y,z)-Komponenten zu jedem Zeitpunkt in dem 963-, 256- oder 3-dimensionalen Vektor enkodiert sind.

Graph Unpooling

Die Graph Unpooling Phase folgt direkt nach der Faltungs-Phase und verfeinert das aktuelle Mesh, indem auf der Kante zwischen zwei Knoten ein weiterer Knoten eingeführt wird. Falls drei Knoten auf einem Dreieck liegen, werden diese ebenfalls



(a) **(b)**

Abbildung 3.20: Erweiterung eines Polygons durch Hinzunahme von neuen Knoten

untereinander mit neuen Kanten verbunden. Auf diese Weise ändert sich der ausgehende Grad der Kanten eines Knotens nicht. Abbildung 3.20a und 3.20b zeigen ein Polygon mit 7 Knoten und 19 Kanten sowie ein Polygon mit 19 Knoten und 42 Kanten. Auf den Kanten zwischen den blauen Knoten werden jeweils neue Knoten eingefügt.

Die Features der benachbarten Knoten gehen aufsummiert jeweils zur Hälfte in die Features des neuen Knotens ein. Ein Informationsgewinn ist aus dieser Phase nicht zu erwarten, da die neuen Knoten exakt zwischen den bisherigen Knoten auf der korrespondierenden Facette liegen. Eine Translation der neuen (und existierenden) Knoten findet durch wiederholte Anwendung der Phasen 2 und 3 statt.

Die Pixel2Mesh-Architektur baut auf mehreren Deformationsblöcken auf, um von einer groben zu einer feinen Mesh-Struktur zu gelangen. Ein initialer Ellipsoid mit vielen Knoten würde zu einem unflexiblen Polygon führen, denn ein Informationsaustausch weit benachbarter Knoten ist nur durch eine Reihe von Faltungen möglich. Fehler, die zu Beginn der Deformation entstehen, können im Nachhinein nicht verbessert werden. Aus diesem Grund startet das originale Mesh mit 156 Knoten, die im zweiten Graph Unpooling Block auf 618 erweitert werden. Nach dem dritten und letzten Block besteht das finale Mesh aus 2466 Knoten mit 34500 Kanten und kann Formen mit Genus-0 darstellen.

3.5.2 Trainingsdetails

In diesem Abschnitt werden noch einmal die Details eines Trainings des P2M-Netzwerks aufgeführt. Die Hyperparameter des Netzwerks wurden nicht verändert und können aus [Wang et al., 2018] entnommen werden. Das Training belief sich für jedes Experiment auf 50 Epochen, um einen Vergleich zwischen den einzelnen Evaluationen ziehen zu können. Die Ergebnisse dieser sind in Kapitel 5.4 zu finden. Die 3D-Rekonstruktion eines Meshs durch eine Verformung eines Ellipsoiden aus einem 2D-Bild erfordert einige Regularisierungstechniken, die in die Trainingsfehlerfunktion einfließen und im Folgenden kurz erläutert werden.

Pixel2Mesh nutzt eine Kombination aus vier verschiedenen Fehlerfunktionen zur Bestimmung des Fehlers des rekonstruierten Meshs zum GT-Objekt während des Trainings. Zum einen ist dies die Chamfer-Distanz (`chamfer loss`), um die Knoten des Polygons möglichst nah an dem GT-Objekt auszurichten, ein `normal loss`, dass die Konsistenz der Normalenvektoren gewährleistet, die `laplacian regularization`, welche die örtliche Beziehung zwischen Nachbarknoten forciert und die `edge length regularization` zur Verhinderung von Ausreißer-Knoten.

Die Chamfer-Distanz wurde bereits im Kapitel 2 vorgestellt, wird jedoch als Trainingsfehlerfunktion leicht abgeändert. Sie wird ohne Mittelwertbildung genutzt und als l_c noch einmal kurz aufgeführt. Sofern nicht anders angegeben, wird p für einen Knoten des vorhergesagten Meshs, q für einen Knoten aus dem GT-Mesh und $N(p)$ für einen Nachbarknoten von p verwendet.

$$l_c = \sum_p min_q \|p - q\|_2^2 + \sum_q min_p \|p - q\|_2^2 \tag{3.6}$$

Als Zweites wird die Fehlerfunktion l_n bezüglich der Normalenvektoren der Knoten vorgestellt.

$$l_n = \sum_p \sum_{q=argmin_q(\|p-q\|_2^2)} \|\langle p - k, n_q \rangle\|_2^2, k \in N(p) \tag{3.7}$$

mit: q = Nähester Knoten für p im GT-Mesh anhand der Chamfer-Distanz
k = Nachbarknoten von p
$\langle \cdot, \cdot \rangle$ = Skalarprodukt
n_q = Normalenvektor des Knotens q

Die Fehlerfunktion in Gleichung 3.7 zwingt die ausgehenden Kanten eines Knotens in dem erzeugten Meshs, sich senkrecht zu dem Normalenvektor des nähesten Knoten des GT-Meshs auszurichten. Dies unterstützt die Angleichung der Orientierung der Oberfläche (die Facetten des rekonstruierten Meshs) an die Oberfläche des GT-Meshs. Die Funktion ist differenzierbar und ähnlich wie die Chamfer-Distanz leicht zu parallelisieren.

Die Kombination der zwei bisher vorgestellten Fehlerfunktionen verbessert die Qualität des rekonstruierten Meshs, jedoch können weiterhin Teilbereiche des Polygons priorisiert werden, sodass an kleinen Arealen starke Verformungen auftreten. Veranlasst durch das Prinzip des Übergangs von einem groben zu einem feinen Mesh, können ursprünglich starke und eventuell falsche Deformierungen nach der Pooling-Phase nur schlecht rückgängig gemacht werden, weshalb zwei weitere Regularisierungen eingeführt werden. Die Nutzung einer Laplace Fehlerfunktion l_{lap} beschränkt den Umfang der Verschiebung eines Knotens des Polygonnetzes bezüglich eines Deformationsblocks und garantiert, dass Knoten sich nicht zu weit von ihrer Ursprungskoordinate entfernen. Auf diese Weise wandern Nachbarknoten in eine ähnliche Richtung. Es kommt seltener zu Knoten, die sich überkreuzen und infolgedessen auch zu einer glatteren Oberfläche. Die Laplace Koordinaten für einen Knoten p sind definiert als:

$$\delta_p = p - \sum_{k \in N(p)} \frac{1}{\|N(p)\|} k \tag{3.8}$$

Die Laplace Fehlerfunktion l_{lap} ist definiert als:

$$l_{lap} = \sum_{p} \|\delta'_p - \delta_p\|_2^2 \tag{3.9}$$

mit: δ'_p = Laplace Koordinaten des Knoten p nach dem Deformationsblock
δ_p = Laplace Koordinaten des Knoten p vor dem Deformationsblock

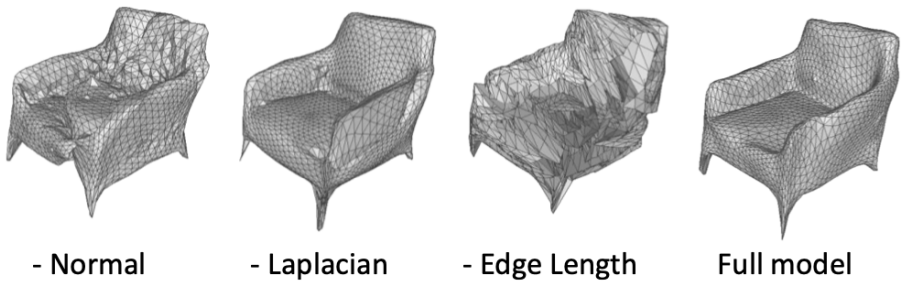


Abbildung 3.21: Visuelle Darstellung des Einflusses der einzelnen Fehlerfunktionen für ein rekonstruiertes Mesh. [WANG et al., 2018]

Schließlich wird eine Funktion eingeführt, die lange Kanten im Mesh bestraft. Auf diese Weise sollen zum einen Ausreißer-Knoten unterbunden werden, zum anderen eine gleichmäßige Verteilung aller Knoten über das Polygon sichergestellt werden. Die Kantenlängen Fehlerfunktion l_{loc} ist definiert als:

$$l_{loc} = \sum_{p} \sum_{k \in N(p)} \|p - k\|_2^2 \tag{3.10}$$

Der gesamte Trainingsfehler l_{all} ist eine gewichtete Summer aus den vier vorgestellten Fehlerfunktionen, mit den konstanten Hyperparametern $\lambda_1, \lambda_2, \lambda_3$. Der Einfluss einer einzelnen Fehlerfunktionen auf das rekonstruierte Mesh ist in Abbildung 3.21 demonstriert. Besonders deutlich zeigt sich die Wirkung der Laplace Regularisierung durch die glatte Oberfläche des Sessels. Der Trainingsfehler ist definiert als:

$$l_{all} = l_c + \lambda_1 l_n + \lambda_2 l_{lap} + \lambda_3 l_{loc} \tag{3.11}$$

mit: $\lambda_1 = 1.6 \times 10^{-4}$
$\lambda_2 = 0.3$
$\lambda_3 = 0.1$

3.5.3 Vergleich mit State of the Art Verfahren

Für eine Bewertung und Einordnung der 3D-Rekonstruktion mit aktuellen State of the Art Verfahren, liefern die Autoren von Pixel2Mesh die Chamfer- und Earth-Mover-Distanzen für das Netzwerk, welches auf dem ShapeNet-Datensatz trainiert

	category	plane	bench	cabinet	car	chair	monitor	lamp	speaker	firearm	couch	table	cellphone	watercraft	mean
CD	3D-R2N2	0.895	1.891	0.735	0.845	1.432	1.707	4.009	1.507	0.993	1.135	1.116	1.137	1.215	1.445
	PSG	**0.430**	0.629	0.439	0.333	0.645	**0.722**	**1.193**	0.756	**0.423**	0.549	0.517	0.438	**0.633**	0.593
	P2M	0.477	**0.624**	**0.381**	**0.268**	**0.610**	0.755	1.295	**0.739**	0.453	**0.490**	**0.498**	**0.421**	0.670	**0.591**
EMD	3D-R2N2	0.606	1.136	**2.520**	1.670	1.466	1.667	1.424	**2.732**	0.688	2.114	1.641	0.912	0.935	1.501
	PSG	**0.396**	1.113	2.986	1.747	1.946	1.891	**1.222**	3.490	**0.397**	2.207	2.121	1.019	0.945	1.653
	P2M	0.579	**0.965**	2.563	**1.297**	**1.399**	**1.536**	1.314	2.951	0.667	**1.642**	**1.480**	**0.724**	**0.814**	**1.380**

Tabelle 3.4: P2M - Vergleich der Chamfer- und Earth-Mover-Distanz mit dem 3D-R2N2- und dem PSG-Netzwerk auf dem ShapeNet-Testdatensatz.

und evaluiert wurde. Die in dem Grundlagen Abschnitt 2.2.1 eingeführte CD wird mit einem Faktor von 1000 und die EMD mit einem Faktor von 0.01 skaliert. In Tabelle 3.4 sind die bekannten 13 Kategorien des ShapeNet Datensatzes mit den jeweiligen Qualitätsmetriken bezüglich des 3D-R2N2-, PSG- und P2M-Netzwerks aufgeführt. Das P2M-Netzwerk schlägt das PSG-Netzwerk in der CD und EMD hinsichtlich des Testdatensatzes. Das PSG Netzwerk als direkter Konkurrent, genießt mit einer Punktwolke als Ausgabe-Repräsentation einen hohen Freiheitsgrad, da Punkte beliebig im Raum verteilt sind und keine lokalen Abhängigkeiten, wie dies bei den Knoten eines Meshs ist, besitzen. Dieser hohe Freiheitsgrad spiegelt sich in einer niedrigen CD und EMD wider, bedeutet jedoch nicht automatisch, dass durch ein Nachbearbeitungsschritt, wie dem Ball-Pivoting Algorithm (BPA), ein qualitativ hochwertiges Mesh entsteht.

Nachdem drei Verfahren im Detail vorgestellt wurden, ergibt sich die Frage, welches dieser Ansätze für das Anwendungsszenario zur Implementierung auf einem mobilen Roboter am geeignetsten erscheint. Der Roboter im Fachgebiet erwartet zur Bestimmung der Greifposition ein Mesh als Eingabe. Pixel2Mesh besitzt als einzig vorgestelltes Netzwerk ein Polygon als Ausgabe-Repräsentation und wurde deshalb für die nachfolgenden Experimente ausgewählt. Beide anderen Verfahren schneiden zudem schlechter bezüglich der CD und EMD ab und benötigen Nachbearbeitungsschritte zur Erstellung eines Ausgabe-Meshs.

Kapitel 4

Erzeugung der Trainingsdaten

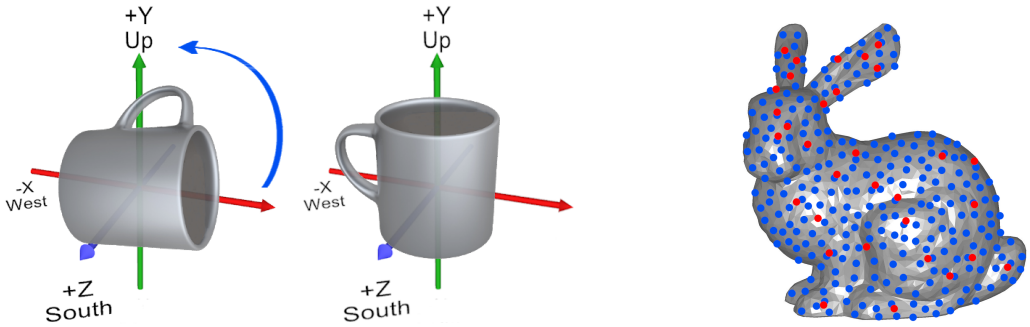
In diesem Kapitel wird die Erzeugung eine Trainings- und Testdatensatzes aufgezeigt, einschließlich einer detaillierten Vorgehensweise zur Generierung der RGB(D)-Bilder und korrespondierenden GT-Punktwolke aus verschiedenen Blickwinkeln anhand eines Wavefront-3D-Modells und Blender. Erstgenannte werden als Eingabe für das Netzwerk benötigt, letztere zur Berechnung der Fehlerfunktion.

4.1 Datensatz

Da sich unser Anwendungsfall weitestgehend auf Handwerkzeuge beschränkt, wurden hauptsächlich zwei Quellen zur Datensatzerstellung herangezogen. Zum einen der ShapeNet-Datensatz [Chang et al., 2015], der sowohl aus Wavefront-, als auch aus Collada-Objekten besteht und die Datenbank von 3D Warehouse [LLC, 2020], welche weitestgehend Objekte in dem Sketchup und Collada Format anbietet.

4.2 Vorverarbeitung der Objekte

Zuerst wurden die Collada-Objekte aus der ShapeNet Datenbank ausgerichtet, falls die entsprechenden Rotationsmatrizen für das Objekt mitgeliefert wurden. Dies soll garantieren, dass die natürliche Ausrichtung einer Form gewährleistet ist. Eine Tasse sollte in der Regel die Öffnung in der positiven Y-Richtung haben (siehe Abbildung 4.1a). Anschließend wurden zur Vereinheitlichung alle Collada-Objekte ebenfalls in das Wavefront-Format umgewandelt.



(a) Ausrichtung einer Tasse

(b) Gleichmäßig abgetastete Punkte

Abbildung 4.1: (a) zeigt die Rotation einer Tasse um die Z-Achse, um die Tassenöffnung in positive Y-Richtung zu drehen. (b) demonstriert die zufällig verteilten, abgetasteten Punkte des Standford Bunny Modells. Rote Punkte werden aufgrund der Nähe zu benachbarten Punkten verworfen.

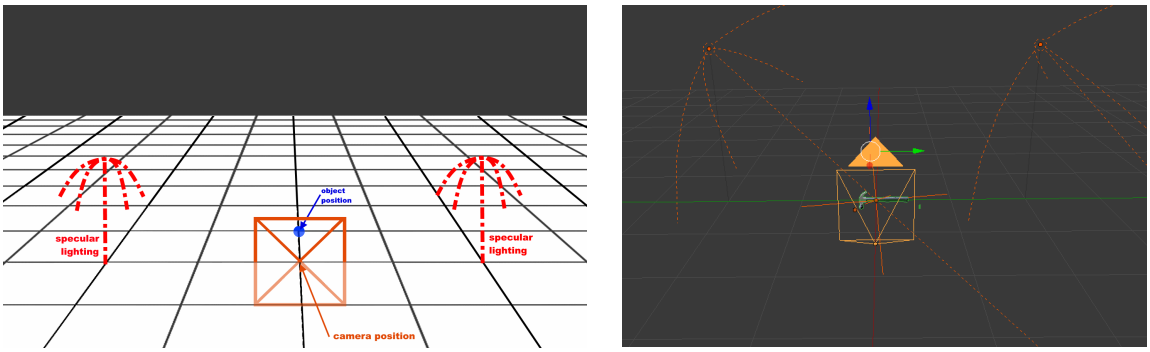
4.3 Ground Truth Punktwolke

Die GT-Punktwolke, welche als Tupel (X,Y,Z,N_X,N_Y,N_Z) mit Koordinaten X,Y,Z und Normalenvektoren N_X,N_Y,N_Z besteht, wurde durch Abtastung von 16384 Punkten über das Mesh erstellt. Hierfür wurde die Funktion `sample_surface_even(mesh, count, radius=None)` der Python-Bibliothek Trimesh [Dawson-Haggerty, 2019] genutzt, welche gleichmäßig verteilte Punkte auf dem Objekt erzeugt, indem es Punkte verwirft, die innerhalb des Radius eines anderen Punktes liegen (schematisch in Abbildung 4.1b aufgezeigt). Aus diesem Grund können weniger Punkte als angefordert zurückgegeben werden. In unserem Fall übergeben wir keinen Radius und nutzen den Fallback der Bibliothek, sodass intern folgende Formel zur Berechnung genutzt wird:

$$radius = \sqrt{(mesh.area/(3 * count))} \tag{4.1}$$

4.4 Rendering

Die RGB-Bilder aus unterschiedlichen Perspektiven wurden mithilfe des GT-Objekts und Blender erstellt. Der Aufbau der Szene in Blender besteht wesentlich aus der Kamera und den zwei Glanzlichtquellen, die in Blender mit einer Stärke von 0.18 initialisiert wurden. Das geladene 3D Objekt wird in einem ersten Schritt mit einer Bounding Box umgeben und bezüglich der längsten Kante isotrop auf den Einheitswürfel



(a) Schematische Darstellung **(b)** Blender Darstellung

Abbildung 4.2: Räumliche Darstellung der zwei Glanzlichtquellen (−2,2,0), (2,2,0) und der Kamera (0,0,1) im euklidischen Koordinatensystem.

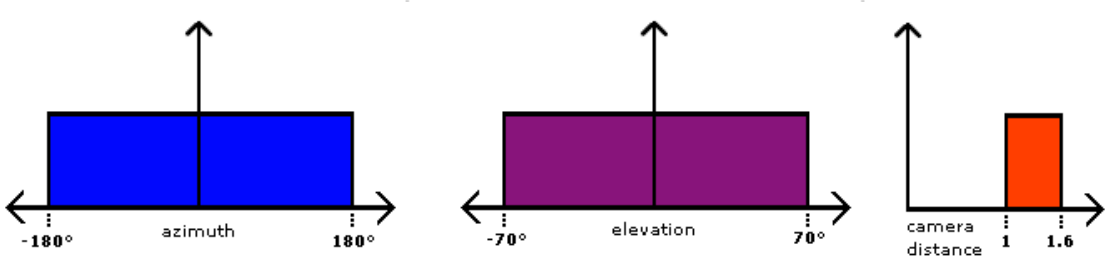


Abbildung 4.3: Stochastische Verteilung der Kameraparameter

skaliert. Die Mittelpunktkoordinaten des Würfels werden nachfolgend genutzt, um das Objekt in den Koordinatenursprung zu setzten. Auf diese Weise wird garantiert, dass das komplette Objekt von der Kamera erfasst werden kann. Dieser Ablauf kann anhand des Code-Schnipsels A.1 im Anhang nachempfunden werden.

Aus Abbildung 4.4 wird deutlich, wie sich die Kamera um das Objekt bewegt und Bilder aus verschiedenen Blickwinkeln rendert. Der Azimutwinkel liegt zwischen 0° und 360°. Der Höhenwinkel ist gleichverteilt im Intervall $[-70°, 70°]$, um Eigenverdeckung von oben und unten zu verhindern. Eine Variation der Objektgröße wird durch eine variable Distanz der Kamera zum Objekt ermöglicht. Die Lichtquellen bleiben stationär und befinden sich an den Koordinaten $(0, -2, 2)$ und $(0, 2, 2)$.

Der Ablauf der Generierung von mehreren RGB-Bildern und den korrespondierenden GT-Punktwolken anhand des originalen Meshs ist in Abbildung 4.5 erkennbar. Zunächst

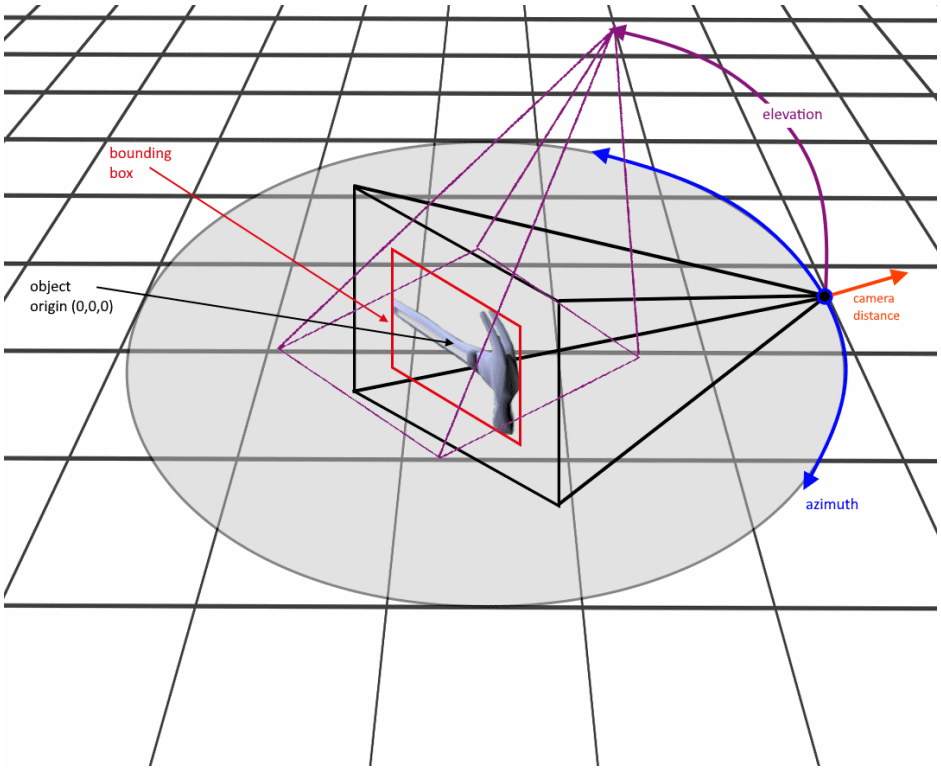


Abbildung 4.4: Im Koordinatenursprung befindet sich das Hammer-Mesh, dass von einer roten Bounding Box umgeben ist. Die ausgerichtete Kamera ist in schwarz dargestellt. Die Kameraparameter beinhalten die Distanz zum Objekt (`camera distance`), den Azimutwinkel (`azimuth`) und den Höhenwinkel (`elevation`).

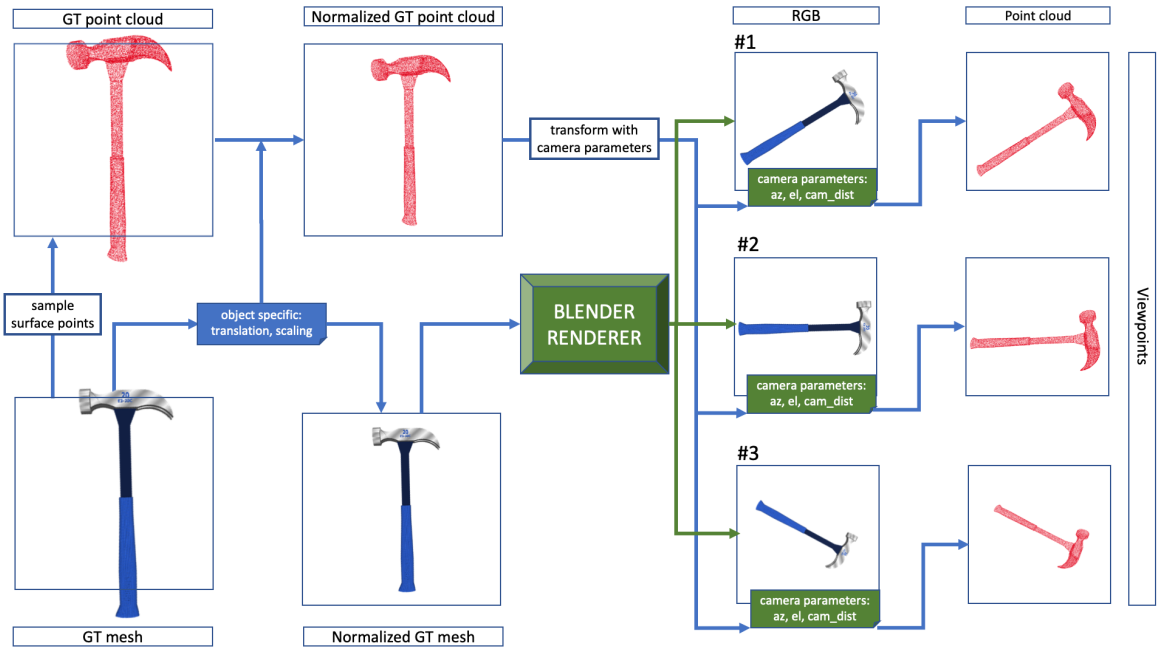


Abbildung 4.5: Überblick über die Generierung von RGB und Ground Truth (GT) Punktwolken durch algebraische Transformationen und Blender

wird das GT-Mesh wie bereits erwähnt auf den Einheitswürfel skaliert und in den Koordinatenursprung verschoben. Diese Objektspezifischen Translations- und Skalierungsparameter werden für die GT-Punktwolke ebenfalls genutzt. Die beiden Schritte werden in der Abbildung 4.5 als Normalisierung bezeichnet. Nun folgt die Erstellung der RGB-Bilder durch das Kernmodul Blender aus unterschiedlichen Blickwinkeln. Auch hier werden wieder die Kameraparameter pro Blickwinkel gespeichert, um die normalisierte Punktwolke ebenfalls anzupassen. Für jeden Blickwinkel wurde auf diese Weise ein RGB-Bild und die dazugehörige Punktwolke erstellt.

In Blender kann durch das Verbinden eines `RenderLayers` mit einem `ViewerNode`, zusätzlich zu dem RGB-Bild auch ein Tiefenbild erzeugt werden. Die `Map Value Node` wird genutzt, um den Tiefenwert auf einen Bereich von 0 bis 255 einzugrenzen.

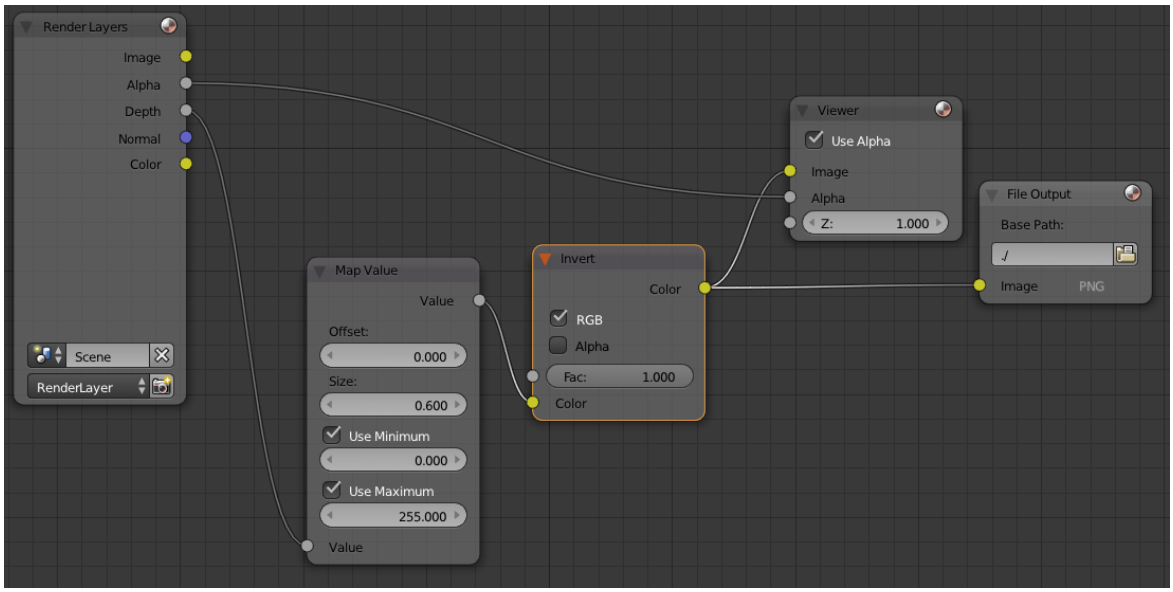


Abbildung 4.6: Aufbau und Parameter der `Compositing Nodes` zur Tiefenbilderstellung in Blender

4.5 Qualität des erzeugten Trainingssatzes

In diesem Abschnitt wird die Qualität des Rendering-Skripts überprüft, indem zunächst Pixel2Mesh mit den original gelieferten ShapeNet Daten von [Wang et al., 2018] trainiert und evaluiert wird. Dieser Datensatz beinhaltet 50k Objekte aus 13 Kategorien und fünf verschiedenen Blickwinkeln. Parallel dazu wird das Netzwerk aus den von uns generierten Daten trainiert und auf dem Original-Testdatensatz evaluiert. Der Mittelwert aus fünf Trainings wurde genutzt, um den Einfluss der zufälligen Initialisierung der Startgewichte des Netzwerks zu verringern.

Aus Tabelle 4.1 kann geschlossen werden, dass die Chamfer- und Earth-Mover-Distanz im Vergleich mit den originalen Daten, um einen Faktor von circa 1,2 schlechter ist. Diese Abweichung ist tolerierbar, da in Betracht gezogen werden muss, dass ein Training auf den originalen Daten und einer Evaluation auf unseren Testdaten ebenfalls eine Verschlechterung als Folge hätte. Dies ist auf die unterschiedliche Blender-Konfiguration der Szene zurückzuführen.

Trainingssatz	ShapeNetP2M_ORIGINAL
Testsatz	ShapeNetP2M_ORIGINAL
Objekte	43783
Kategorien	13
Trainingsbilder	218915
Blickwinkel	5
Epochen	50
CD	0.601762
EMD	3.393307

(a) Originaler Datensatz

Trainingssatz	ShapeNetP2M_OWN
Testsatz	ShapeNetP2M_ORIGINAL
Objekte	43783
Kategorien	13
Trainingsbilder	218915
Blickwinkel	5
Epochen	50
CD	0.773262
EMD	4.138422

(b) Eigener Datensatz

Tabelle 4.1: Evaluation auf dem originalen und dem eigenen gerenderten ShapeNet Datensatz bezüglich der Chamfer- und Earth-Mover-Distanz.

4.6 Handwerkzeug-Datensatz

Bestimmt durch die Beschränkung der Rekonstruktion, zur Ermittlung der Greifposition eines Roboters, auf Handwerkzeuge und Kleinteile, werden folgende Kategorien zur Datensatzerstellung festgelegt. Dabei wird darauf geachtet, ein starkes Ungleichgewicht innerhalb des Trainingsdatensatzes zwischen den Klassen zu vermeiden. Die Anzahl der Zangenobjekte (`pliers`) ist hierbei die kleinste Kategorie mit 46 Modellen. In unserem Fall wird jedes Objekt aus 20 verschiedenen Blickwinkeln gerendert, wodurch sich der Unterschied der Objektanzahl zwischen den einzelnen Klassen relativiert. Sowohl bei unserer P2M-, als auch bei der originalen Implementierung wurde auf eine gewichtete Fehlerfunktion bezüglich der Größe der Kategorien verzichtet.

4.7 Einordnung für Pixel2Mesh

Abbildung 4.7 zeigt auf an welchen Stellen die erzeugten RGB-Bilder und die transformierten Punktwolken in das neuronale Netzwerk einfließen. Die aus verschiedenen

category	bolt	bottle	bulb	calculator	flashlight	hammer	key	knife	mouse
#items	102	474	169	136	122	204	100	288	61
category	mug	nail	pencil	pliers	scissors	screwdriver	spoon	usb_stick	wrench
#items	205	61	451	46	68	81	84	84	105

Tabelle 4.2: Anzahl der Objekte pro Klasse des erstellten Datensatzes

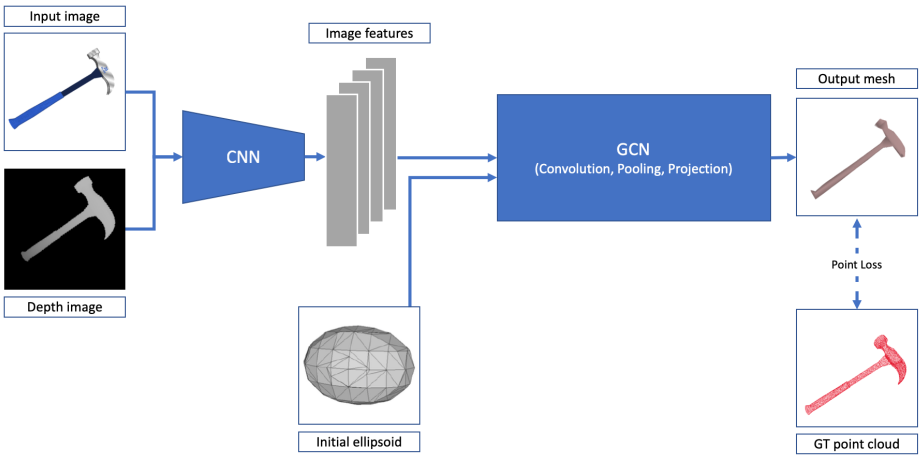


Abbildung 4.7: Die gerenderten RGB-Bilder und die dazugehörigen transformierten Punktwolken werden als Eingabe und zur Fehlerbestimmung für Pixel2Mesh genutzt.

Blickwinkeln generierten RGB-D-Bilder werden als $224 \times 224 \times 4$ Vektor für die Eingabe genutzt. Die Punktpaardistanz (`point loss`), im P2M-Netzwerk als Chamfer-Distanz implementiert, nutzt die Nearest Neighbor-Suche, um für jeden Knoten des rekonstruierten Meshs den nächstgelegenen Nachbarn in der transformierten GT-Punktwolke zu bestimmen. Anhand dieser und weiterer Fehlermetriken zwischen dem erzeugten Mesh und der GT-Punktwolke, wird der Trainingsfehler berechnet.

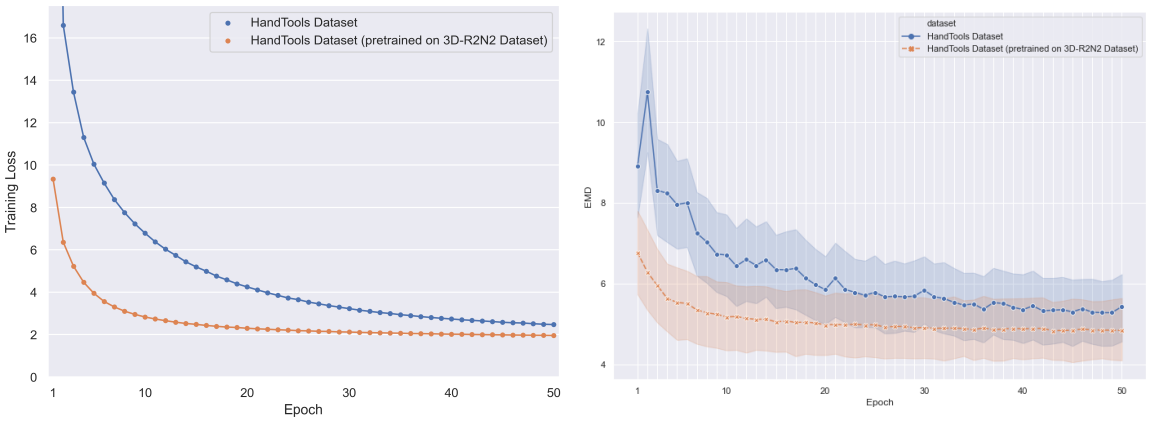
Kapitel 5

Experimente

Im Weiteren schließt dieses Kapitel mit den durchgeführten Trainings und Auswertungen an. Alle Trainings beziehen sich auf Pixel2Mesh und den im vorherigen Kapitel dargestellten Handwerkzeug-Datensatz. Begonnen wird mit dem grundlegenden Aufbau eines Experimentes, gefolgt von einem Vergleich zwischen einem vortrainierten Netzwerk und einem alleinig auf dem Handwerkzeug-Datensatz trainierten Netzwerks (Abschnitt 5.2), der Vorteil von einer Hinzunahme von Tiefendaten (Abschnitt 5.3), einer Vertex Analyse (Abschnitt 5.4), die Performance des Netzwerks bezüglich einer Rekonstruktion, wenn eine Teilverdeckung in dem Eingabebild vorliegt (Abschnitt 5.5) und schließlich der Einfluss des Blickwinkels (Abschnitt 5.6). Diese Auswertungen bilden die Argumentationsgrundlage für eine Bewertung des Netzwerks im Kontext des Anwendungsszenarios im nächsten Kapitel.

5.1 Aufbau

In diesem Abschnitt wird der Versuchsaufbau in Kürze skizziert. Verglichen mit dem 3D-R2N2-Datensatz, der 43783 Objekte enthält, beläuft sich die Objektanzahl des Handwerkzeug-Datensatzes auf 2760. Zur Vergrößerung des Datensatzes werden anstelle von 5, 20 verschiedene Blickwinkel zur Generierung der RGB-Bilder genutzt. Anschließend wird der Datensatz nach dem Pareto-Prinzip zu 80% in Trainingsdaten und zu 20% in Testdaten aufgeteilt. Um einzelne Experimente untereinander zu vergleichen, wird jedes Training auf 50 Epochen festgesetzt. Parameter des Pixel2Mesh werden nicht geändert und können aus der Publikation von [Wang et al., 2018] entnommen werden.



(a) Trainingsfehler

(b) Testfehler (EMD)

Abbildung 5.1: Trainingsfehler (a) und Testfehler (b) zwischen vortrainierten- und nicht vortrainierten Netzwerk

5.2 Vortrainieren mittels 3D-R2N2-Datensatz

Auf der Grundlage, dass der erstellte Handwerkzeug-Datensatz im Vergleich zum 3D-R2N2-Datensatz eine geringe Anzahl an Objekten besitzt, welche zum Training eines tiefen neuronalen Netzwerks notwendig sind, wird nachfolgend ein Vergleich zwischen einem, auf einem großen Datensatz vortrainierten- und einem nicht vortrainierten Netzwerk aufgezeigt. In der Literatur [Martinez und Gill, 2019], [Shima et al., 2017] hat sich zudem gezeigt, dass ein neuronales Netzwerk, welches auf Gewichten von einem vortrainierten Netzwerk initialisiert wird, bessere Ergebnisse als ein Netzwerk auf zufällig initialisierten Gewichten erzielt. Durch das sogenannte `transfer learning` [Yang et al., 2016] stabilisiert sich außerdem das Training und es zeigen sich in der Regel drei positive Faktoren:

1. Der initiale Trainingsfehler ist niedriger.
2. Schnellerer Abfall der Kurve.
3. Der konvergierte Trainingsfehler ist niedriger.

Abbildung 5.1 verdeutlicht diese Punkte. Die gelbe Kurve zeigt das auf dem Datensatz 3D-R2N2 vortrainierte Netzwerk, welches schneller gegen die Asymptote konvergiert. Nach circa 20 Epochen zeigt sich keine gravierende Verbesserung des Trainingsfehlers, während das Netzwerk mit zufällig initialisierten Gewichten bis Epoche 40 einen stark

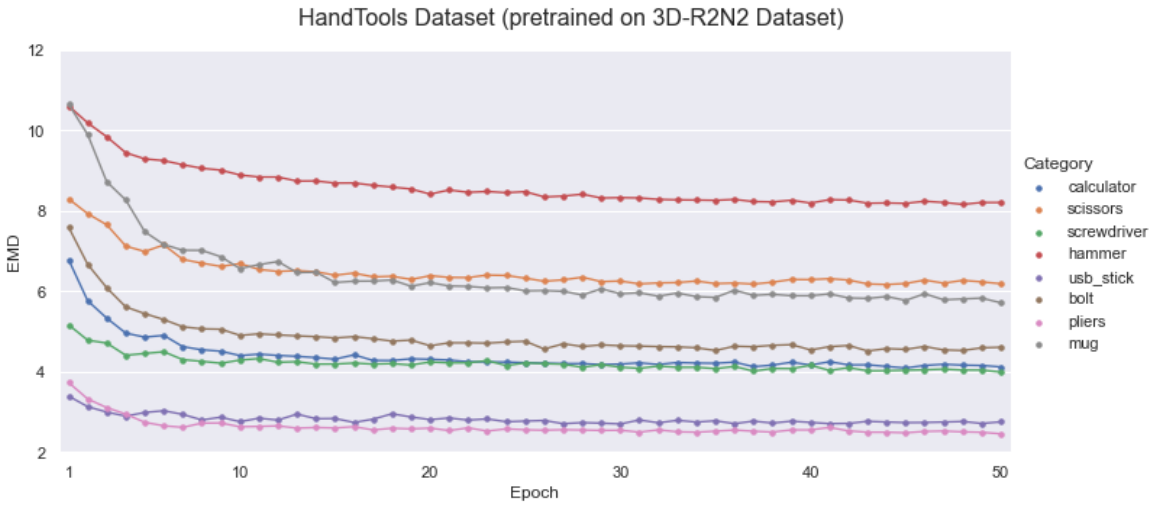


Abbildung 5.2: Testfehler bezüglich der EMD über 50 Epochen anhand der Objektkategorien eines vortrainierten Netzwerks

fallenden Trainingsfehler besitzt. Das gleiche Verhalten bestätigt sich im Testfehler, in diesem Fall als EMD aufgeführt. Auch hier schlägt das vortrainierte Netzwerk das zufällig initialisierte Netzwerk. In den ersten Epochen schwankt der Testfehler in dem nicht vortrainierten Netzwerk stark. Die schattierten Bereiche um beide Kurven bilden das 95% Konfidenzintervall über alle 13 Objektkategorien ab und unterscheiden sich nicht auffällig voneinander.

Abbildungen 5.2 und 5.3 zeigen den Testfehler innerhalb der einzelnen Objektkategorien gleichermaßen von einem vortrainierten- und nicht vortrainierten Netzwerk auf. Interessant ist, dass sich einzelne Kategorien untereinander deutlich in der EMD unterscheiden. Ein Hammer wird im Schnitt schlechter rekonstruiert als ein USB-Stick, da die Hammer-Kategorie 204 Objekte enthält, die sich bezüglich ihrer Form stark voneinander unterscheiden. Dagegen beinhaltet die USB-Stick-Kategorie nur 84 Objekte, die sich in ihrer Form sehr ähneln.

Da in den 3D-Datenbanken oftmals nicht genug Objekte pro Kategorie zu finden sind, wurden 3D-Modelle in allen möglichen Ausprägungen genutzt. Im Fall des Hammers bedeutet dies Formen mit kleinen oder großen Hammerkopf, kurzen oder langen Stielen und anderen Merkmalen. Ein USB-Stick ist in der Regel jedoch sehr homogen in seiner Form und weist keinerlei Variationen auf. Aus Abbildungen 5.2 und 5.3 ist zusätzlich

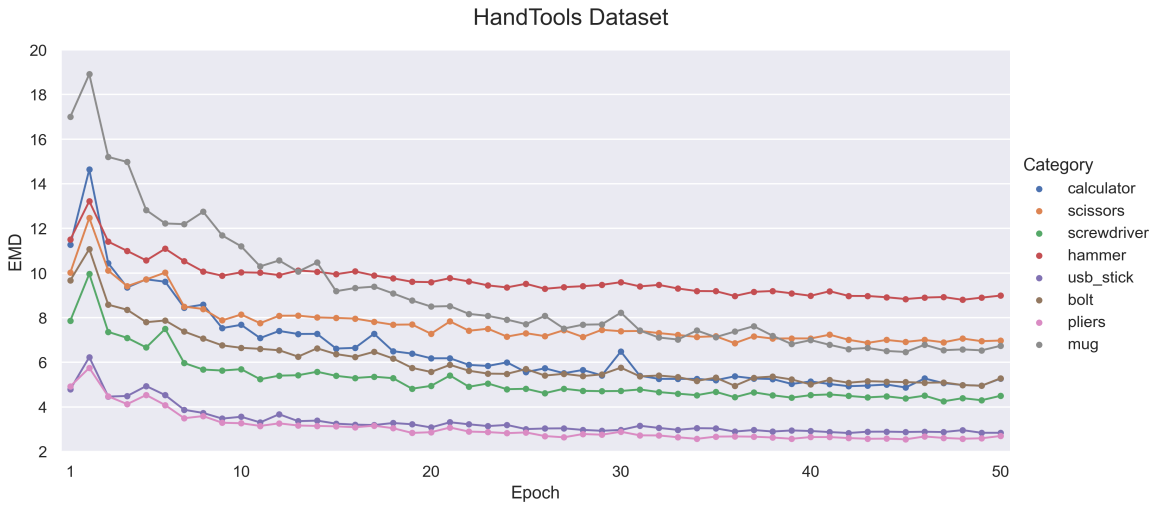


Abbildung 5.3: Testfehler bezüglich der EMD über 50 Epochen anhand der Objektkategorien eines nicht vortrainierten Netzwerks

erkennbar, dass ein Vortraining keiner Objektkategorie im Speziellen hilft und zu einer drastischen Verbesserung des Testfehlers führt. Dies könnte ebenfalls auf die Diversität der Objekte in den Kategorien zurückzuführen sein.

In den folgenden Experimenten wird, falls nicht ausschließlich anders deklariert, auf den Ergebnissen des vortrainierten Netzwerks weitergearbeitet, aus welchen die bis zu diesem Zeitpunkt besten Ergebnisse in der Rekonstruktion erzielt wurden.

5.3 Tiefendaten

Aus dem vorherigen Abschnitt resultiert die Erkenntnis, dass ein, auf einem großen Datensatz vortrainiertes Netzwerk die besten Ergebnisse in der Rekonstruktion erzielt. Eine weitere Verfeinerung von Pixel2Mesh ist die Einbindung von Tiefendaten als Eingabevektor. Dieser Schritt ist sinnvoll, weil der mobile Roboter bereits über Tiefenbildsensoren verfügt. Die Feature-Vektoren werden nun aus der Kombination von Farbbildern und Tiefenbildern erstellt. Dies hat den Vorteil, dass die Distanz vom Betrachter zu dem Objekt (Z-Achse) korrekt abgebildet werden kann und texturfreie Areale auf dem Eingabebild besser interpretiert werden können [Lasang et al., 2014]. Die nötigen Tiefenbilder für das Training werden ebenfalls durch Blender aus allen Blickwinkeln generiert und spiegeln die Distanzen zwischen der Kamera und dem Objekt perfekt wider. In einem realen Szenario, beinhalten die Daten aus den Sensoren ein gewisses Grundrauschen und können nicht die Qualität der synthetischen Daten erreichen. Die Abweichung einer Rekonstruktion mit einem realen System muss in der Praxis zusätzlich evaluiert werden. Der Aufbau der benötigten `Compositing Nodes` für die Blender-Konfiguration zur Erstellung der synthetischen Tiefenbilder ist in Abbildung 4.6 dargelegt. In der Recherche der State of the Art Verfahren ist aufgefallen, dass oftmals keine Tiefenbild Informationen, sondern lediglich `Multi-View` Ansätze, genutzt werden. Weshalb dies der Fall ist, wurde jedoch nicht direkt ersichtlich.

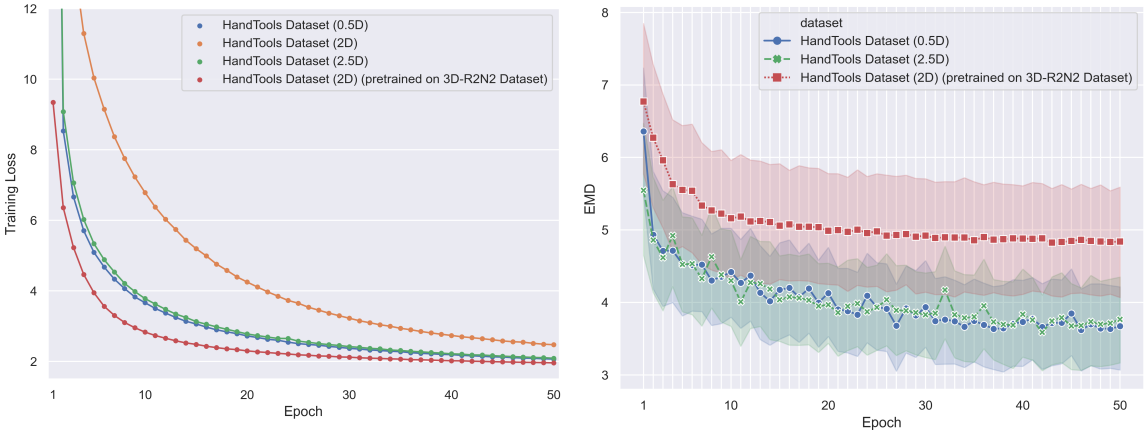
Als Basislinie dient das auf dem 3D-R2N2 vortrainierte Modell. In Abbildung 5.4b wird auf den Testfehler des nicht vortrainierten 2D-Modells aufgrund der besseren Lesbarkeit des Graphen verzichtet. Die Ergebnisse können zum Vergleich aus Abbildung 5.1b entnommen werden. Sowohl das Modell, dass nur auf Tiefenbildern, als auch das Modell, dass auf 2.5D-Bildern trainiert und evaluiert wird, weisen im Vergleich mit einem auf 2D-Bildern trainierten Modell deutlich niedrigere EMD Distanzen auf.

Aus dem niedrigen Testfehler des auf den 0.5D-Daten trainierten Modells zeigt sich, dass das Netzwerk keine Klassifizierung der Objekte anhand von Textureigenschaften vornimmt oder Lichtverhältnisse aufgrund der stationären Lichtquellen aus Blender als Ankerpunkt zur Berechnung der Rotation des Objektes mit einbezieht. Lediglich die erkannte Form dient zur Konstruktion des 3D Meshs. Aus den beiden Kurven der auf 0.5D- und 2.5D-Daten trainierten Modelle wird ersichtlich, dass das Farbbild keinen

großen Einfluss auf die Rekonstruktion hat und als Eingabevektor entfernt werden kann.

Abbildung 5.5 zeigt eine 3D-Rekonstruktion eines Hammers aus zwei unterschiedlichen Perspektiven. Ein auf 2D-Daten trainiertes Netzwerk, welches als Eingabe ein RGB-Bild bekommt, kann zwar die Form des Objektes detailgetreu nachbilden, jedoch nicht die Entfernung auf der Z-Achse richtig einschätzen. Die Verschiebung ist in dem rechten Bild zu sehen. Eine mögliche Ursache dieses systematischen Fehlers könnte durch die variable Kamera Entfernung entstehen, die im Kapitel 4.4 beschrieben ist. Dieses Problem wird jedoch durch Nutzung von Tiefeninformationen gelöst, da der Abstand der Kamera als Eingabeparameter eingeht.

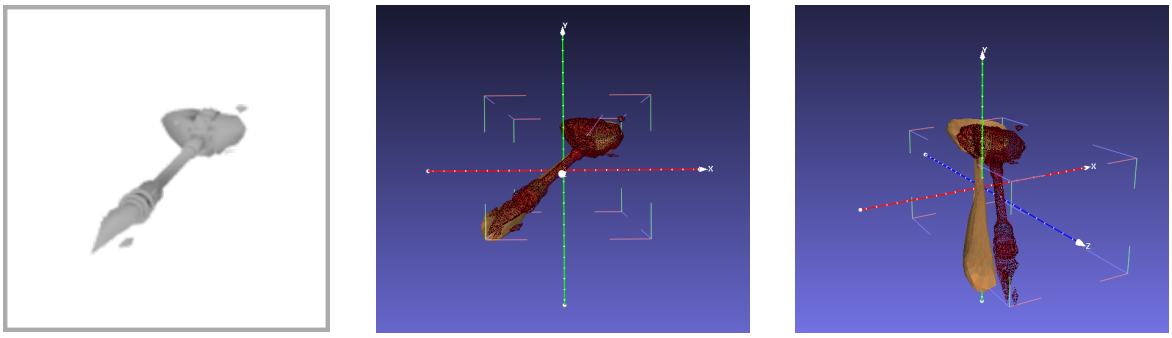
Weiterhin ist zu beobachten, dass in dem Eingabebild zwei kleine Formen neben dem Hammer zu sehen sind. Die Ursache dafür liegt in dem Handwerkzeug-Datensatz, bei dessen Erstellung zwar Objekte mit großen Artefakten entfernt wurden, jedoch von einer Aussortierung von Objekten mit sehr kleinen Artefakten abgesehen werden, da sonst die Anzahl der 3D-Modelle pro Kategorie nicht ausreichend für ein Training eines neuronalen Netzwerks ist. Unabhängig davon trägt dies zu einer unmerklich schlechteren Rekonstruktion bei und kann in diesen Fällen vernachlässigt werden. Die



(a) Trainingsfehler

(b) Testfehler (EMD)

Abbildung 5.4: (a) verdeutlicht die Trainingsfehlerkurven zwischen einem Training auf 0.5D-, 2D- und 2.5D-Trainingsdaten. (b) bezieht sich in diesem Zusammenhang auf den Testfehler.



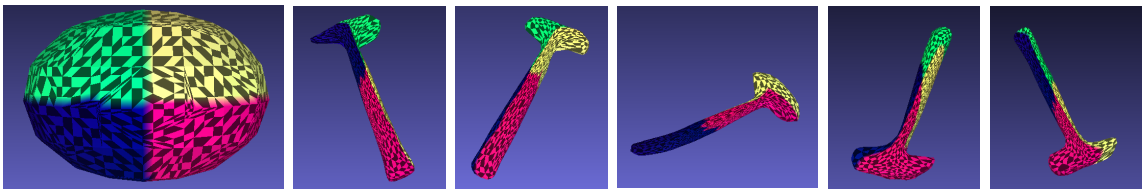
(a) Eingabebild **(b)** Blick aus Richtung Z-Achse **(c)** Andere Perspektive

Abbildung 5.5: Vergleich der Mesh Rekonstruktion aus zwei verschiedenen Blickwinkeln.

zwei Teilobjekte werden, wie man anhand des orangenen Meshs erkennt, in das finale Objekt mit eingebunden, denn nur Genus-0-Formen können konstruiert werden.

5.4 Vertex Analyse

Dieser Abschnitt verdeutlicht die Transformation der einzelnen Knoten des initialen Ellipsoid Meshs zum rekonstruierten Mesh. In Abbildung 5.6a ist das initiale Ellipsoid mit 2466 Knoten und 4928 Kanten ersichtlich. Jeweils zwei Oktanten (Vorder- und Rückseite) werden zusammengefasst und mit einer Farbe markiert. Im rekonstruierten Mesh werden die korrespondierenden Knoten mit den gleichen Farben eingefärbt, sodass ein Zusammenhang festgestellt werden kann und Rückschlüsse über Pixel2Mesh getroffen werden können.



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Abbildung 5.6: Farbkodierte Transformation der Knoten des initialen Ellipsoid (a) zu einem finalen Mesh aus unterschiedlichen Blickwinkeln (b) - (f)

(a) Verdeckung in der Mitte **(b)** Verdeckung am Rand

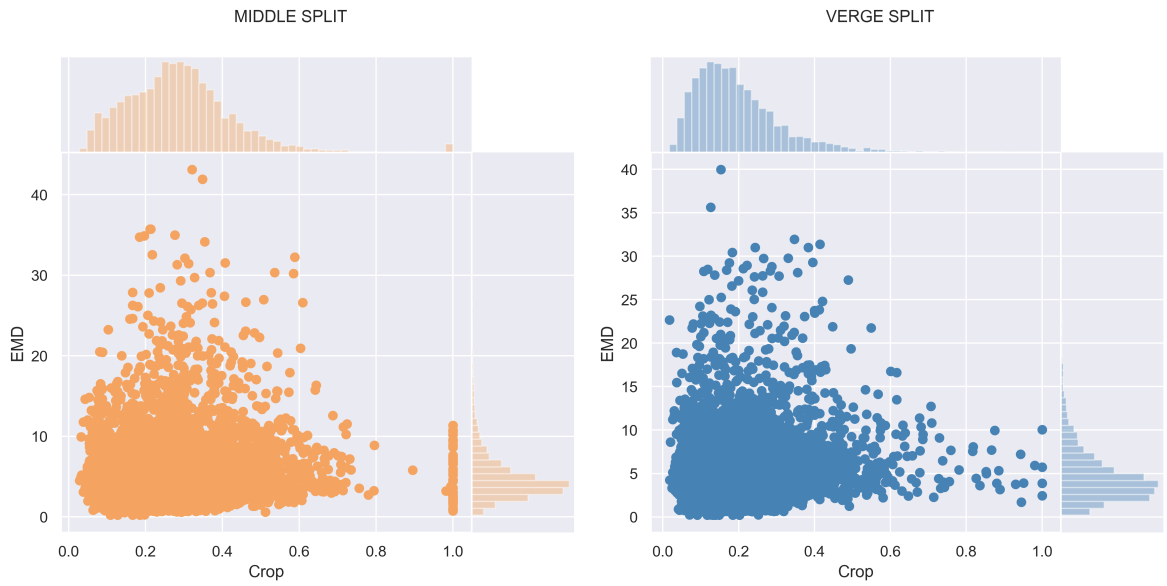
Abbildung 5.7: Teilverdeckung einer Schraube in der Mitte (a) und am Rand (b)

Besonders auffällig ist, dass jede Farbe räumlich gebunden ist und in jedem der Bilder (b) - (f) an einer ähnlichen Stelle auftaucht. Dies lässt darauf schließen, dass das Netzwerk intern keine Klassifikation der Eingabe bezüglich einer Objektkategorie vornimmt und kein Basisobjekt erstellt, das daraufhin gedreht wird. Wäre dies der Fall, so würde z. B. der Hammerkopf in jedem rekonstruierten Mesh aus unterschiedlichen Blickwinkeln die gleiche Farbe besitzen. Daraus ergibt sich, dass jede Orientierung eines Objekts neu gelernt werden muss und nicht anhand anderer Blickwinkel inferiert werden kann.

5.5 Verdeckung

In bisherigen Publikationen werden weitestgehend vollständig und alleinstehende Objekte betrachtet. Im Anwendungsfall zur Bestimmung der Greifposition eines überreichten Gegenstandes kann es jedoch zu Teilverdeckungen kommen. Eine wichtige Eigenschaft, die das Netzwerk besitzen muss, ist die korrekte Rekonstruktion des 3D-Objektes trotz Verdeckung.

Zuerst wurde die Annahmen getroffen, dass eine Verdeckung am Rand und in der Mitte für das Netzwerk unterschiedliche Probleme erzeugt. Für den ersten Fall, einer Verdeckung in der Mitte, entsteht ein Loch, welches im Prozess der Rekonstruktion bestmöglich geschlossen werden muss. Beim zweiten Fall, einer Verdeckung am Rand, muss das Netzwerk entscheiden, ob das Objekt vollständig ist und eine reguläre 3D-Rekonstruktion genutzt wird oder ob ein Teil des Objekts fehlt, was zu einer erweiterten Rekonstruktion führt. Eine Schraube mit einer Verdeckung in der Mitte und am Rand aus zwei verschiedenen Blickwinkeln ist in Abbildung 5.7 aufgeführt. In (a) ist die



(a) **(b)**

Abbildung 5.8: Zusammenhang zwischen Verdeckungsgrad und Testfehler für Verdeckung in der Mitte (a) und Verdeckung am Rand (b)

Schraube in der Mitte verdeckt, während in (b) die Schraube an der Spitze (am Rand des Objekts) verdeckt wird. Im Folgenden wird der Ablauf zur Erstellung eines Verdeckungsdatensatzes (nachfolgend auch HandTools Dataset Occlusion genannt) aus dem Handwerkzeug-Datensatz aufgezeigt.

Das Code Snippet A.2 im Anhang zeigt die wichtigsten Schritte den Verdeckungsdatensatz zu generieren. Der Ablauf kann jedoch wie folgt zusammengefasst werden. Zunächst wird jedes Bild eingelesen und der RGB-Wert pro Pixel aufsummiert. Anschließend wird zufällig ein Punkt auf dem Bild, der nicht Null ist und dementsprechend dem Objekt angehört, gezogen. Im letzten Schritt wird ein Kreis um diesen Punkt gezeichnet und jegliche Werte innerhalb dieses Kreises mit Null überschrieben, sodass dieser Teil transparent erscheint. Der `occultation_factor` kann beliebig gewählt werden und bestimmt die Größe des Verdeckungsgebiets. Die Klassifizierung, ob ein Bild eine Verdeckung am Rand oder in der Mitte besitzt, wird durch die Kontur bestimmt. Objekte, die nicht in zwei oder mehr zusammenhangslose Teile durch Verdeckung getrennt werden, besitzen genau eine Kontur. Sobald ein Objekt jedoch zerteilt wird, entstehen mehrere Konturen. Die Schraube in Abbildung 5.7a wird in zwei Stücke geteilt und besitzt

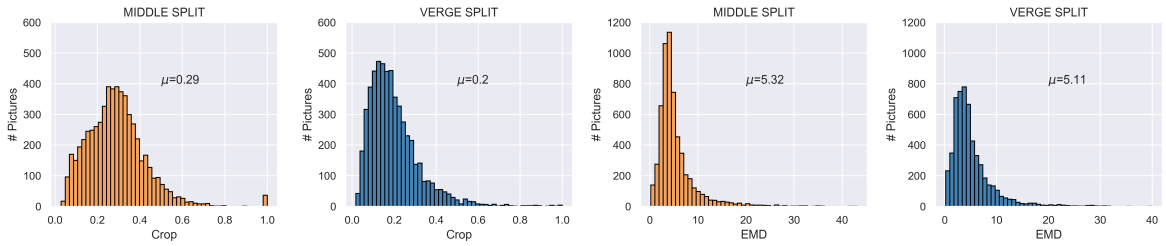


Abbildung 5.9: Histogramm zur Verteilung des Verdeckungsgrades und der EMD über den Testdatensatz

dementsprechend zwei Konturen. Aus einigen Blickwinkeln kann ein Objekt sehr klein erscheinen, z. B. ein Nagel von oben, was zu einer Verdeckung von 100% führt. In Abbildung 5.8 ist auf der X-Achse die Höhe der Verdeckung und auf der Y-Achse der Testfehler mit der EMD angegeben. Eine vollständige Verdeckung ist bei einer kleinen Menge von Objekten im Datensatz aufgetreten. Diese sind im Diagramm anhand der Punkte mit einem Crop-Wert von 1.0 zu erkennen.

Die folgenden Experimente beziehen sich nun auf die Trainings von Pixel2Mesh auf dem Verdeckungsdatensatz. In Abbildung 5.8 ist der Zusammenhang zwischen dem Verdeckungsgrad und der EMD dargestellt. Aufbauend auf dem derzeitig besten Modell (vortrainiert auf 3D-R2N2- und Handwerkzeug-Datensatz) wird Pixel2Mesh zusätzlich auf dem Verdeckungsdatensatz trainiert und anschließen evaluiert. Folgende Aussagen können anhand des Diagramms getroffen werden:

1. Der Verdeckungsgrad (`Crop`) unterscheidet sich nicht stark zwischen der Verdeckung in der Mitte und am Rand. Die genaue Verteilung kann in Abbildung 5.9 abgelesen werden.
2. Der Unterschied des Mittelwerts der EMD zwischen einer Verdeckung in der Mitte ($\mu = 5.32$) und Verdeckung am Rand ($\mu = 5.11$) kann ebenfalls vernachlässigt werden.
3. Es kann eine komplette Verdeckung der Objekte stattfinden.

Bei einer kompletten Verdeckung wird das Eingabebild als 0-Array gelesen. Zumal auch hier der Alpha Channel nur 0er enthält, wird jeder RGB-Wert des betroffenen

Pixel durch 255 ersetzt und auf das Intervall zwischen [0,1] skaliert. Der resultierende Vektor enthält dementsprechend nur 1er und wird zur Extraktion der Bildmerkmale in den CNN Schichten als Eingabe verwendet. Die Bildmerkmalsvektoren werden anschließend mit den Gewichten des GCN verrechnet und für die Verschiebung der Knoten des Ellipsoiden genutzt.Angesichts keiner vorhandenen Features gleicht auch das rekonstruierte Mesh keiner bekannten Form. Auffällig ist, dass die EMD für diese hohen Verdeckungsgrade relativ niedrig ist. Inwiefern die Ausgabe ein Mittelwert-Mesh aller gelernten Objekte ist, müsste in weiteren Untersuchungen gezeigt werden, ist jedoch aufgrund der zeitlichen Beschränkung dieser Arbeit nicht weiter verfolgt worden. Zudem findet in der Praxis eine 3D-Rekonstruktion aus einem leeren Bild nicht statt.

Abschließend wird untersucht, welchen Einfluss das zusätzliche Training auf einem Verdeckungsdatensatz (im Folgenden als VD-Modell benannt) gegenüber einem Modell hat, dass nicht auf diesem trainiert wurde (im Folgenden als ND-Modell benannt). Die blaue Kurve in Abbildung 5.10 zeigt den Verlauf des Testfehlers für ND-Modell auf den Verdeckungsdaten. Die EMD beläuft sich bei Epoche 50 auf 7.6. Diese unzufriedenstellende Rekonstruktion ist nicht verwunderlich, da das trainierte Modell zu keinem Zeitpunkt im Training eine Vervollständigung des verdeckten Areals vollziehen musste. Der Testfehler des VD-Modells ist weitaus niedriger und durch die grüne Kurve

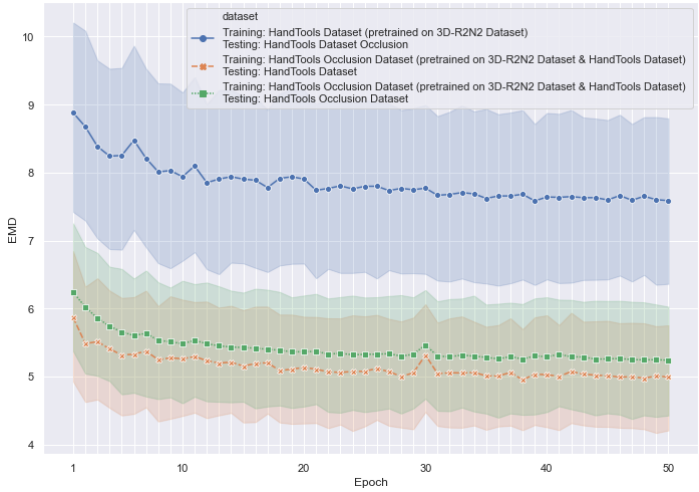


Abbildung 5.10: Verlauf der Testfehler (EMD) über 50 Epochen auf dem Verdeckungs- und Handwerkzeug-Datensatz

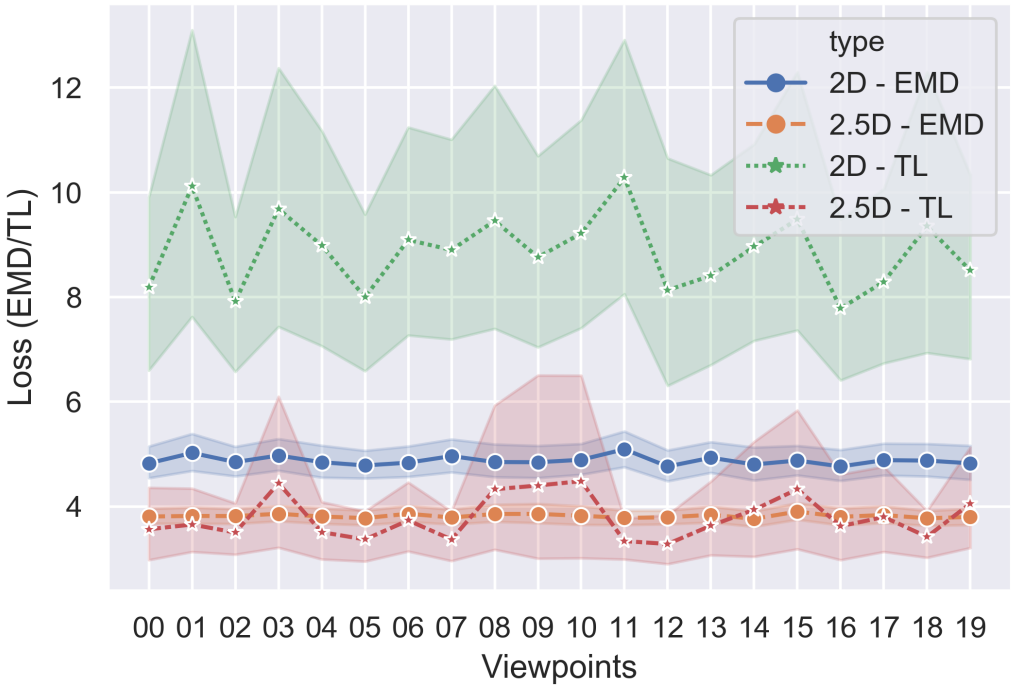
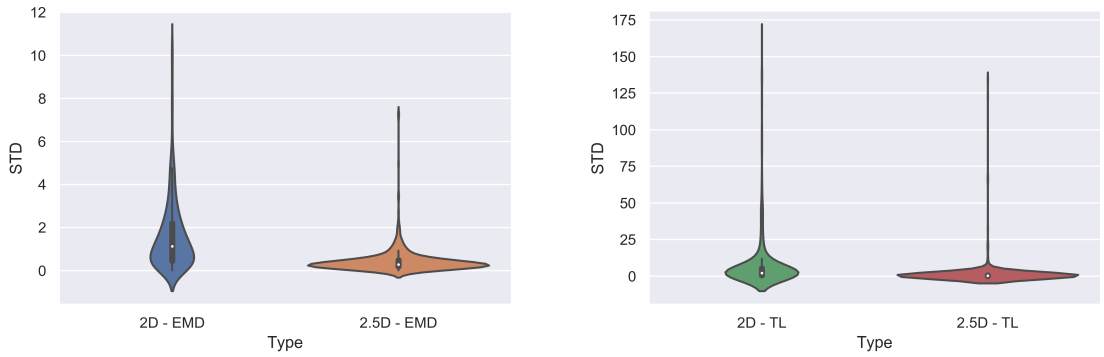


Abbildung 5.11: Fehlerverlauf der EMD und des Trainingsfehlers über 20 Blickwinkel

gekennzeichnet. Mithin wird ein zusätzliches Training auf dem Verdeckungsdatensatz notwendig für ein Szenario, in dem Teilverdeckung auftreten kann. Abschließend wird zusätzlich getestet, ob das VD-Modell den normalen Handwerkzeug-Datensatz gleichermaßen gut rekonstruieren kann und sich nicht auf Verdeckungen spezialisiert hat. Der gelbe Kurvenverlauf gleicht dem aus Abbildung 5.4b in Abschnitt 5.3 und bekräftigt damit die These.

5.6 Einfluss der Blickwinkel

Im letzten Abschnitt der Experimente wird der Zusammenhang zwischen Blickwinkel und Testfehler untersucht. Für einen mobilen Roboter könnte aus einer solchen Analyse eine geeignete Kameraposition bzw. der Winkel von der Kamera zum Objekt inferiert werden. Für die folgenden Versuche wird das auf dem 2D- und 2.5D-Handwerkzeug-



(a) **(b)**

Abbildung 5.12: Standardabweichung des Testfehlers für Pixel2Mesh trainiert auf 2D- und 2.5D-Daten bezüglich der EMD (a) und des Trainingsfehlers (b) über 20 Blickwinkel

Datensatz trainierte Modell genutzt. Die Trainingsfehlerfunktion wird zusätzlich als Testfehlermetrik hinzugezogen, um herauszufinden, ob ein Ausreißer (im Sinne einer hohen EMD im Testfehler bezüglich eines Blickwinkels) auch einen hohen Trainingsfehler produziert. Ein erster Einblick wird exemplarisch in Tabelle 5.1 geliefert. Daraus wird ersichtlich, dass es einzelne Blickwinkel pro Objekt gibt, die eine sehr viel schlechtere EMD aufweisen. Dies äußert sich sowohl in der Testfehler-Metrik als auch in der Trainingsfehler-Metrik. Gründe hierfür kann Eigenverdeckung oder die bereits vorgestellte Z-Verschiebung sein, die bei den rot markierten Objekten nur auf dem 2D-Datensatz auftreten.

Deutlich wird anhand Abbildung 5.11 die kontinuierliche Stabilität der EMD über alle Blickwinkel. Ferner existieren keinerlei Abhängigkeiten zwischen genau einer Perspektive und dem Testfehler. Als Nachtrag für den Abschnitt 5.3 bekräftigt auch der EMD Kurvenverlauf zwischen 2D und 2.5D eine weitaus bessere Rekonstruktion mit der Hinzunahme von Tiefenbildern. Das 95% Konfidenzintervall ist ebenfalls kleiner für das 2.5D-Modell im Gegensatz zu dem des 2D-Modells. Die genaue Standardverteilung kann anhand des Violingraphen in Abbildung 5.12 entnommen werden.

Unter Berücksichtigung aller Aspekte liegt kein Zusammenhang zwischen dem Testfehler und einem bestimmten Blickwinkel vor. Ursache dafür ist die zufällige Generierung der

object	viewpoints																				type type	metrics		
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19		mean	std	cv
A	0.96	0.73	0.78	0.97	0.84	0.95	0.89	0.77	0.70	0.69	0.94	0.88	0.94	0.75	0.91	0.92	0.99	0.81	0.97	1.04	2D - EMD	0.87	0.10	0.11
	3.20	2.39	2.54	3.37	2.86	3.40	3.08	2.38	2.18	2.15	3.43	3.04	3.31	2.28	3.20	3.23	3.51	2.51	3.51	3.70	2D - TL	2.97	0.49	0.16
	0.95	0.85	0.90	0.94	1.01	0.98	1.00	0.88	0.90	0.88	1.01	1.04	0.96	0.85	0.97	0.94	0.94	0.91	1.00	1.04	2.5D - EMD	0.95	0.05	0.06
	2.59	2.31	2.45	2.56	2.73	2.58	2.69	2.33	2.41	2.31	2.69	2.80	2.57	2.31	2.63	2.51	2.48	2.40	2.67	2.73	2.5D - TL	2.54	0.15	0.06
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	GT	0	0	0
B	3.30	4.51	3.82	2.88	4.29	2.41	2.56	4.05	2.89	3.92	5.41	5.85	3.91	2.70	3.15	4.08	3.03	2.36	2.80	2.30	2D - EMD	3.51	0.97	0.27
	3.13	5.17	3.57	4.09	7.62	3.03	3.83	5.52	4.73	4.32	5.70	7.57	4.98	3.09	3.29	3.73	2.72	2.63	3.58	2.41	2D - TL	4.23	1.45	0.34
	2.52	2.22	2.42	2.84	2.60	2.52	2.34	2.72	2.81	2.57	2.74	2.71	2.29	2.88	2.08	2.48	2.59	2.71	2.85	2.53	2.5D - EMD	2.57	0.21	0.08
	2.47	2.66	2.79	3.54	3.71	2.96	3.26	2.80	3.71	3.46	3.19	2.74	2.29	2.45	2.51	3.08	2.43	2.88	4.04	2.35	2.5D - TL	2.97	0.50	0.16
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	GT	0	0	0

Tabelle 5.1: Fehlermetriken aus unterschiedlichen Blickwinkeln

Kameraparameter jeder einzelnen Perspektive. Ausreißer liegen vorwiegend im 2D-Fall vor, da die Z-Verschiebung des Objektes nicht korrekt berechnet werden kann. Weitere perspektivische Probleme treten entweder unter Eigenverdeckung auf oder wenn das Netzwerk die Form zu keiner bereits erlernten Objektstruktur einordnen kann.

Kapitel 6

Bewertung und Ausblick

Die im Rahmen dieser Arbeit untersuchten Verfahren zur Rekonstruktion eines 3D-Modells aus einem RGB(D)-Bild, sowie die dargestellten Experimente auf Pixel2Mesh werden in den nachfolgenden Abschnitten noch einmal kritisch hinterfragt. Aufgetretene Probleme, mögliche Optimierungsmöglichkeiten und eine Bewertung bezüglich der Nutzbarkeit für das vorgestellte Anwendungsszenario werden aufgezeigt. Eine kurze Zusammenfassung und ein Ausblick für zukünftige Arbeiten bilden den Abschluss.

6.1 Probleme und Grenzen der Rekonstruktion

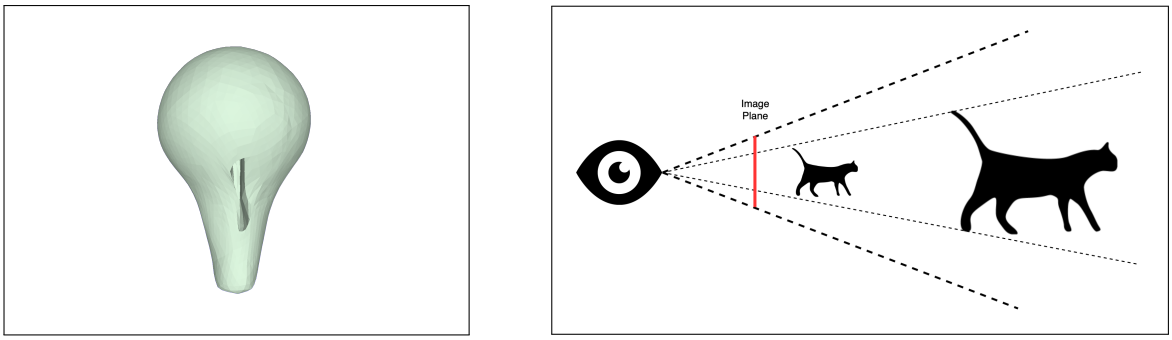
Die Erstellung eines guten Datensatzes ist maßgeblich an dem Ausgang und der Güte der Rekonstruktion eines neuronalen Netzwerks beteiligt. Pixel2Mesh wurde anhand des 3D-R2N2-Datensatzes vortrainiert, welches wie auch unser Handwerkzeug-Datensatz rein aus synthetischen Daten besteht. Der Prozess zur Generierung von RGB(D)-Bildern wurde durch Blender realisiert. Die Konfiguration beinhaltet eine vordefinierte Szene mit statischer Beleuchtung und einer Kamera, die sich um das Objekt dreht. Die gerenderten Bilder sind abhängig von den gesetzten Kameraparametern sowie der Position der Lichtquellen. Zum Finden geeigneter Parameter wurde versucht, die gerenderten Bilder aus dem 3D-R2N2-Datensatz anhand der mitgelieferten Wavefront Objekte möglichst gut zu reproduzieren. Dazu wurde in einem ersten Schritt Pixel2Mesh auf dem originalen 3D-R2N2-Datensatz trainiert und auf dem Testdatensatz evaluiert. Die Parameter des eigenen Rendering-Skripts wurden in mehreren Zyklen so angepasst, dass ein Training von P2M auf dem reproduzierten 3D-R2N2-Datensatz ähnlich gute

Werte, wie auf dem mitgelieferten Testdatensatz erreicht. Die Details hierzu wurden bereits in Tabelle 4.1 aufgezeigt. Die Objekte in den synthetischen Bildern weisen jedoch nicht die Diversität und Detailreiche gegenüber einem Bild auf, welches in der Realität aufgenommen wurde. Die Texturen der 3D-Modelle in den Datensätzen sind oftmals sehr simpel und können durch das Rendering nicht die Qualität eines Realbildes erreichen.

Ein weiteres Problem ist die sogenannte Eigenverdeckung, die aus bestimmten Kamerawinkeln auftreten kann, weil viele Objekte im Datensatz nicht ausgerichtet sind. Nimmt man beispielsweise eine Tasse, so kommt diese in der Realität vorwiegend in einer mit dem Boden nach unten (in negative Y-Achse) und mit der Öffnung nach oben (in positive Y-Achse) gerichteten Form vor. Der Höhenwinkel der Kamera in der Blender-Konfiguration ist auf das Intervall $[-70°, 70°]$ gesetzt, sodass das 2D-Bild des Objekts nicht direkt von oben oder unten erstellt wird. Da die 3D-Modelle im Datensatz jedoch in jeglicher Rotationslage vorkommen können, kann diese Problem nicht vermieden werden, außer es wird eine manuelle Ausrichtung jedes einzelnen Objekts im Datensatz vorgenommen.

Der finale Handwerkzeug-Datensatz besteht aus 2760 Objekten, die auf 18 Kategorien aufgeteilt wurden. Mit 153 Objekten pro Kategorie stellt dies ein unteres Limit für ein Training in tiefen neuronalen Netzwerken dar. Obwohl die Anzahl der Objekte in öffentlichen 3D-Modell-Datenbanken in den letzten Jahren stark gestiegen ist, konnten in unserem Fall aus ShapeNet und 3D Warehouse nicht genügend Formen für die benötigten Kategorien gefunden werden. Ein Vortraining auf einem großen Datensatz, wie bereits in den Experimenten aufgeführt, sowie der Erhöhung der Anzahl der gerenderten Bilder pro Objekt, kann dieses Problem jedoch nur teilweise verbessern.

Die Evaluierung der Experimente aus Kapitel 5 haben verschiedene Grenzen der Rekonstruktion mit Hilfe von Pixel2Mesh aufgezeigt. Die größte Einschränkung betrifft die Verformung eines initialen Ellipsoiden, aus dem nur Genus-0 Formen erstellt werden können. Löcher im Ausgabe-Mesh können nicht reproduziert werden, was für die Bestimmung einer Greifposition ein Problem darstellen kann. Die Auswertung der Experimente zeigt daher zum Teil schlechte Chamfer- und Earth-Mover-Distanzen, da der Handwerkzeug-Datensatz oftmals Formen mit Löchern (siehe Schere - Abbildung 2.4)



(a) **(b)**

Abbildung 6.1: (a) zeigt ein rekonstruiertes 3D-Modell einer Glühbirne mit einem Loch. (b) verdeutlicht die Abhängigkeit zwischen der Größe eines Objektes und der Distanz zum Betrachter.

enthält. Interessant ist zudem, dass einige rekonstruierte 3D-Modelle einen Einschnitt aufweisen. Abbildung 6.1a zeigt eine Genus-0-Form einer Glühbirne, die einen vertikalen Einschnitt aufweist, der das Polygon nicht komplett durchstößt. Die Knoten auf der Vorderseite wurden auf eine Weise in das Mesh deformiert, sodass ein kleiner Hohlraum im Inneren entstanden ist. Solche Fehlkonstruktionen verringern die Qualität eines Meshs und müssen in `post-processing` Schritten geschlossen werden, falls die Anwendung dies erfordert.

Weiterhin zeichnete sich ab, dass es mit einer Eingabe von einem RGB-Bild teilweise zu einer Verschiebung entlang der Z-Achse (Betrachtungs-Achse) kommen kann. Abbildung 6.1b verdeutlicht dieses Verhalten mit einem simplen Beispiel. Nahe und kleine Objekte erscheinen auf dem Bild mit gleicher Größe wie ein großes Objekt in weiter Ferne. Ein möglicher Ursprung des Problems bei RGB-Bildern wurde bereits aufgezeigt, das aus der variablen Kamera-Distanz zum Objekt in der Erstellung des Datensatzes entsteht. Dennoch kann diese Entfernung nicht allein aus dem Bild inferiert werden, weshalb die Hinzunahme von Tiefendaten notwendig ist. Da auf dem Roboter Tiefensensoren verbaut sind, ist auch dieses Problem zu vernachlässigen.

Die Auswahl von Fehlermetriken für das Training des neuronalen Netzwerks spielen für die Qualität des generierten 3D-Modells ebenfalls eine wichtige Rolle. So kann zwischen einem *besten* Mesh bezüglich einer Distanz-Metrik zwischen dem GT- und dem

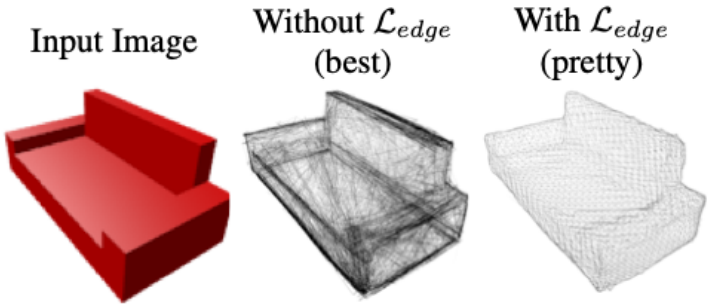


Abbildung 6.2: Training des Netzwerks ohne `edge length regularizer` L_{edge} produziert Polygone mit überlappenden Kanten. Das Hinzufügen dieser Fehlermetrik zum Trainingsfehler verringert dieses Problem, führt jedoch zu einer schlechteren Übereinstimmung mit dem GT-Objekt bezüglich der Chamfer- oder Earth-Mover-Distanz [Gkioxari et al., 2019].

rekonstruierten Mesh und einem subjektiv schönsten Mesh des Betrachters unterschieden werden [Gkioxari et al., 2019]. Pixel2Mesh nutzt u. a. eine Fehlermetrik, die lange Kanten zwischen Knoten unterbinden soll. Auf diese Weise soll ein Polygonnetz erzeugt werden, dass eine gleichermaßen äquivalente Aufteilung aller Knoten über die ganze Form gewährleistet. Ein Vergleich beider Rekonstruktionen ist in Abbildung 6.2 zu sehen. Zudem werden überlappende Knoten bestraft, wodurch eine glatte Oberfläche entsteht. Für den Betrachter führt dies zu einer schöneren Form, während es für die Chamfer- oder Earth-Mover-Distanz als Fehlermetrik zwischen dem GT- und dem vorhergesagten Modell einen Nachteil aufweist, da auf diese Weise alle Knoten über das Mesh verteilt sind und sich nicht an Arealen klumpen, an denen die Distanz zum GT-Objekt am kürzesten ist. Wichtige Stellen des GT-Objekts, die eine höhere Auflösung benötigen, können bei einer Verteilung aller Knoten über das Polygon nicht anhand einer höheren Knotendichte repräsentiert werden. Eine geeignete Gewichtung aller Qualitätsmetriken muss für einen Anwendungszweck speziell evaluiert werden.

6.2 Optimierungspotenzial

Einige der im vorherigen Abschnitt eingeführten Probleme können durch den Einsatz von Tiefendaten oder Nachberarbeitungsschritten behandelt werden. Weitere Optimierungen bezüglich der Netzwerkarchitektur, der Berechnung der Qualitätsmetriken und des

Datensatzes werden nun in Kürze aufgelistet.

Eine einfache Möglichkeit zur Verbesserung der Rekonstruktion von ungesehen Objekten ist es, den Datensatz bezüglich der Anzahl der Modelle pro Kategorie zu erweitern. Der 3D-R2N2-Datensatz besteht aus 50k Objekten, wobei jedes Objekt aus 5 verschiedenen Blickwinkeln gerendert wurde. Aufgrund der geringeren Objektanzahl im Handwerkzeug-Datensatz, wurde ein RGB-Bild für jedes Objekt aus 20 verschiedenen Blickwinkeln erzeugt. Eine Erhöhung der Objektdiversität führt meist auch zu einer verbesserten Rekonstruktion von ungesehen Formen. Ein weiterer Optimierungspunkt wäre es, gleichmäßig abgetastete Punkte auf dem Mesh anstelle der Knotenkoordinaten für die Berechnung der Chamfer-Distanz zwischen dem GT- und dem vorhergesagten Modell zu nutzen. Diese müssten zwar zusätzlich während der Laufzeit gesampled werden, würden aber die Form des Meshs besser wiedergeben.

Für die Erstellung der Bildmerkmale wird derzeit eine VGG-16 Encoder-Architektur genutzt. Durch ein tieferes residuales Netzwerk (ResNet) oder das Inception-Modul von GoogLeNet, könnten Features besser und teilweise schneller extrahiert werden. Eine weitere Idee behandelt die Projektion von Features aus dem 2D-Bild in die Knoten des Polygons. Hierfür muss getestet werden ob eine bessere Rekonstruktion auftritt, wenn nur Features des 2D-Bildes auf die sichtbaren Knoten des Meshs übertragen werden, da die abgewandte Seite des Objektes im Bild keinerlei Informationen für die Graph-Projektion bereitstellt. Die abgewandten Knoten bleiben leer und erhalten ihre Features bzw. Punktkoordinaten anschließend allein über Faltungen aus den Nachbarknoteninformationen. Damit einher gehen würde jedoch auch die Erhöhung der Hop-Reichweite, damit sich die Features schnell über den kompletten Graphen ausbreiten. Ein MultiView-Ansatz ließe sich mit dieser Idee ebenfalls kombinieren.

6.3 Einordnung der Nutzbarkeit für den Anwendungsfall

Das Ziel dieses Abschnittes ist die Betrachtung der Nutzbarkeit des Pixel2Mesh-Netzwerks für eine Implementierung auf einem mobilen Roboter im Anwendungsfall: Rekonstruktion eines 3D-Modells anhand eines oder mehrerer RGB-D-Bilder zur Be-

stimmung der Greifposition. Notwendige Vor- und Nachbearbeitungsschritte werden ebenfalls analysiert.

Das Pixel2Mesh-Netzwerk erwartet als Eingabe ein RGB(D)-Bild mit einem freigestellten Objekt auf einem weißen bzw. transparenten Hintergrund. Die Höhe und Breite des Bildes kann beliebig gewählt werden, da das Netzwerk dieses in einem ersten Schritt auf die Dimension 224×224 skaliert. Das Objekt nimmt in der Regel nur einen geringen Bereich des Bildes ein und sollte in einem ersten Schritt detektiert, segmentiert und auf circa 80% - 90% des Bildausschnittes skaliert werden. Ein möglicher State of the Art Detektions-Algorithmus ist `Detectron2` [Wu et al., 2019], welcher eine Weiterentwicklung des Mask R-CNN [He et al., 2017] ist. Die Verwendung von Tiefenbildinformationen kann diesen Prozess erleichtern. Damit das erkannte und freigestellte Objekt 80% des Bildausschnittes einnimmt und in der Mitte zentriert ist, muss es skaliert und verschoben werden. Die Skalierungs- und Translationsparameter werden nach der Rekonstruktion des 3D-Modells wieder benötigt, um das Modell bezüglich der Raumkoordinaten des Ursprungsobjektes zurück zu transformieren. Nach diesen Schritten liegt ein 3D-Modell mit korrekter Ausrichtung, Größe und Position aus dem Realbild vor. Falls als Eingabe ein RGB-Bild genutzt wurde, kann das Skalierungsproblem mit zugehöriger Z-Verschiebung des 3D-Modells auftreten (siehe Abbildung 6.1b).

Die Farb- und Tiefenbildkameras auf dem Roboter nehmen mit einer Framerate von 30 fps auf. Im Folgenden wird untersucht, ob eine Rekonstruktion in einer Zeitbeschränkung von 33ms möglich ist. Wie für das Training, wurde auch für diesen Test eine NVIDIA Titan X genutzt. Das Laden aller nötigen Bibliotheken zu Beginn dauerte 6961ms. Im Anschluss wurden sequentiell 100 Objekte anhand von RGB-D-Bildern rekonstruiert. Die vollständige Zeit belief sich auf 2713ms, die sich aus dem Laden eines RGB-D-Bildes von der Festplatte (743ms) und der Rekonstruktion eines 2466 Knoten Polygons durch einen `forward-pass` durch das Pixel2Mesh Netzwerk (1970ms) zusammensetzt. Im Durchschnitt ergibt dies eine Rekonstruktionszeit für ein einzelnes Bild von 27,13ms. Falls das Bild bereits im Speicher vorliegt, verringert sich die Zeit auf 19,70ms. Die Zeitschranke für eine Rekonstruktion in unter 33ms wurde damit erfüllt und ermöglicht die Anwendung in 30 fps Systemen.

Aus den vorgestellten Ergebnissen folgt, dass eine Implementierung auf einem mobilen Roboter in dem vorgestellten Anwendungsszenario möglich ist und alle notwendigen Kriterien erfüllt. Das Problem der Limitierung auf Genus-0 Formen kann leider nicht überwunden werden und es muss sich in der Praxis zeigen, inwiefern dies zu einer Beeinträchtigung zur Bestimmung der Greifposition anhand des konstruierten Modells führt. Neue Verfahren in diesem aktiven Forschungsfeld zeigen, dass es Ansätze gibt, Polygone mit Genus > 0 aus einem einzelnen Bild zu erzeugen. Darunter fällt u. a. das Mesh R-CNN-Netzwerk [Gkioxari et al., 2019]. Benötigte Vor- und Nachbearbeitungsschritte der Kameraaufnahmen und des erstellten 3D-Modells wurden ebenfalls aufgezeigt und müssen zusätzlich realisiert werden. Falls gewünscht kann der Handwerkzeug-Datensatz anhand der gelieferten Blender-Konfiguration erweitert werden.

6.4 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde untersucht, ob ein aktueller State of the Art Ansatz die Anforderungen an eine 3D-Rekonstruktion anhand eines RGB(D)-Bildes zur Bestimmung der Greifposition in Betracht gezogen werden kann. Dazu wurden zunächst Grundlagen für Graph-basierte neuronale Netzwerke erläutert. Drei Verfahren wurden bezüglich ihrer Netzwerkarchitektur und Ausgabe-Repräsentation detailgetreu vorgestellt und auf ihre Eignung für das Anwendungsszenario hin untersucht. Aufgrund der Tatsache, dass die Ausgabe von Pixel2Mesh keinerlei Nachbearbeitungsschritte zur Erstellung eines Meshs erfordert, wurde dieses Verfahren für die vorgestellten Experimente ausgewählt.

Zu den Anforderungen dieser Arbeit galt es zudem ein Handwerkzeug-Datensatz zu generieren, auf dem das Training des neuronalen Netzwerks ausgeführt werden sollte. Eine besondere Schwierigkeit war eine geeignete Datenbank mit genügend 3D-Modellen zu finden. Ein Zusammenschluss von Objekten aus der ShapeNet-Datenbank und der 3D-Warehouse-Datenbank lieferten insgesamt 2760 Objekte für 18 Kategorien. Ein Blender-Render-Skript wurde an Anlehung der geometrischen Vorgaben von den Autoren von 3D-R2N2 implementiert und benötigte viele iterative Zyklen zur Wahl der korrekten Szenenparameter. Nachdem der Datensatz mit den RGB(D)-Bildern und der entsprechend transformierten GT-Punktewolken erstellt wurde, folgten Experimente

bezüglich einer Optimierung der Rekonstruktion. Ein vortrainiertes Netzwerk anhand des ShapeNet-Datensatzes, sowie eine Hinzunahme von Tiefendaten verbesserten die Performance des Netzwerks erheblich. Im Kontext der Mensch-Roboter-Interaktion kommt es im Regelfall zu Teilverdeckung eines entgegengenommenen Objektes. Dieses Problem wurde ebenfalls weitgehend untersucht und konnte durch ein zusätzliches Training auf einem Verdeckungsdatensatz verringert werden. Die Beurteilung, ob dieses Netzwerk für das genannte Anwendungsszenario genutzt werden kann, wurde in der Theorie bestätigt und muss sich in der Praxis noch zeigen.

Aufbauend auf den hier vorgestellten trainierten Netzwerken und dem gelieferten Handwerkzeug-Datensatz, gilt es die Entwicklung zur Implementierung auf dem Roboter weiterzuführen. Die in dieser Arbeit vorgestellten Qualitätsmetriken beziehen sich auf perfekte Farb- und Tiefendaten, welche in der Praxis zusätzliches Sensorrauschen aufweisen. Weitere Experimente in einem realen Szenario sind notwendig, um eine endgültige Aussage über die Genauigkeit der Rekonstruktion eines erkannten Objektes zu treffen.

Abschließend lässt sich sagen, dass an dem Gebiet der Computer Vision und der 3D-Rekonstruktion aktiv geforscht wird und sich in den letzten Jahren viele neue Ansätze entwickelt haben. Dies zeigt sich unter anderem an der Anzahl von Einsendungen wissenschaftlicher Arbeiten, die jedes Jahr bei der Konferenz für Computer Vision und Mustererkennung (CVPR) eingegangen sind.

Anhang A

Quellcode

In Abschnitt 4.4 wird der Ablauf zur Erstellung eines Datensatzes durch Blender erläutert. Damit das Mesh vollkommen von der Kamera erfasst werden kann, wird es zunächst anhand der längsten Kante seiner Bounding Box isotrop auf den Einheitswürfel skaliert und anschließend in den Koordinatenursprung verschoben. Dieser Vorgang findet für jedes Objekt im Datensatz statt.

```
1 meshBound = mesh.matrix_world.to_quaternion() * mesh.dimensions
2 x_ratio, y_ratio, z_ratio =  abs(meshBound.x), abs(meshBound.y), abs(meshBound.z)
3 mesh.scale *= 1 / max(x_ratio, y_ratio, z_ratio)
4 ...
5 local_bbox_center =  0.125 * sum((Vector(b) for b in mesh.bound_box), Vector())
6 global_bbox_center = o.matrix_world * local_bbox_center
7 mesh.location = mesh.location - global_bbox_center
```

Quellcode A.1: Das Mesh wird zunächst skaliert und anschließend anhand seiner Bounding Box in den Ursprung verschoben.

Dieser Code-Schnipsel zeigt die Berechnung der Größe der Verdeckung sowie die zufällige Platzierung auf dem Objekt. In Abschnitt 5.5 wird das Vorgehen im Detail erläutert.

```
1  im = Image.open(image).convert("RGBA")
2  ...
3  image_pixels = np.sum(pixel_list, axis=2)
4  ...
5  single_vector = np.reshape(image_pixels, (1,-1))[0]
6  idx_nonzero, = np.nonzero(single_vector)
7  random_pixel = np.random.choice(idx_nonzero)
8
9  x = random_pixel % image_pixels.shape[0]
10 y = int(np.floor(random_pixel / image_pixels.shape[0]))
11
12 radius = max(im.size[0], im.size[1]) / occultation_factor
13 half_rad = radius / 2
14 ImageDraw.Draw(im).ellipse((x - half_rad, y - half_rad, x + half_rad, y + half_rad), fill=0)
```

Quellcode A.2: Schritte zur Erstellung eines Verdeckungsdatensatzes. Aus dem Bild wird zufällig ein Punkt, der auf dem Objekt liegt, gezogen und anschließend durch einen transparenten Kreis gefüllt.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[Agarwal und Prabhakaran, 2009] Agarwal, Parag und B. Prabhakaran (2009). *Robust blind watermarking of point-sampled geometry.* IEEE Transactions on Information Forensics and Security, 4(1):36–48. *(Seite: 21)*

[Bertsekas, 1985] Bertsekas, Dimitri P (1985). *A distributed asynchronous relaxation algorithm for the assignment problem.* In: *1985 24th IEEE Conference on Decision and Control*, S. 1703–1704. IEEE. *(Seite: 11, 12)*

[Brimkov et al., 2006] Brimkov, Valentin E, R. P. Barneva und H. Hauptman (2006). *Combinatorial image analysis.* Lecture Notes in Computer Science, (4958). *(Seite: 21)*

[Chang et al., 2015] Chang, Angel X, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su et al. (2015). *Shapenet: An information-rich 3d model repository.* arXiv preprint arXiv:1512.03012. *(Seite: 47)*

[Choy et al., 2016] Choy, Christopher B, D. Xu, J. Gwak, K. Chen und S. Savarese (2016). *3d-r2n2: A unified approach for single and multi-view 3d object reconstruction.* In: *European conference on computer vision*, S. 628–644. Springer. *(Seite: 22, 33)*

[Dawson-Haggerty, 2019] Dawson-Haggerty, et al. (2019). *trimesh.* `https://trimsh.org/`. version = 3.2.0, Accessed: 2019-12-8. *(Seite: 48)*

[Fan et al., 2017] Fan, Haoqiang, H. Su und L. J. Guibas (2017). *A point set generation network for 3d object reconstruction from a single image.* In: *Proceedings*

of the IEEE conference on computer vision and pattern recognition, S. 605–613. *(Seite: 9, 12, 25, 26, 27, 31)*

[Gargantini, 1982] Gargantini, Irene (1982). *Linear octtrees for fast processing of three-dimensional objects.* Computer graphics and Image processing, 20(4):365–374. *(Seite: 34)*

[Girdhar et al., 2016] Girdhar, Rohit, D. F. Fouhey, M. Rodriguez und A. Gupta (2016). *Learning a predictable and generative vector representation for objects.* In: *European Conference on Computer Vision*, S. 484–499. Springer. *(Seite: 18, 19, 33)*

[Gkioxari et al., 2019] Gkioxari, Georgia, J. Malik und J. Johnson (2019). *Mesh r-cnn.* In: *Proceedings of the IEEE International Conference on Computer Vision*, S. 9785–9795. *(Seite: 72, 75)*

[Goodfellow et al., 2014] Goodfellow, Ian, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio (2014). *Generative adversarial nets.* In: *Advances in neural information processing systems*, S. 2672–2680. *(Seite: 20)*

[Hamidi et al., 2019] Hamidi, Mohamed, A. Chetouani, M. El Haziti, M. El Hassouni und H. Cherifi (2019). *Blind robust 3D mesh watermarking based on mesh saliency and wavelet transform for copyright protection.* Information, 10(2):67. *(Seite: 21)*

[Han et al., 2019] Han, Xian-Feng, H. Laga und M. Bennamoun (2019). *Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era.* arXiv preprint arXiv:1906.06543. *(Seite: 20)*

[Häne et al., 2017] Häne, Christian, S. Tulsiani und J. Malik (2017). *Hierarchical surface prediction for 3d object reconstruction.* In: *2017 International Conference on 3D Vision (3DV)*, S. 412–420. IEEE. *(Seite: 33)*

[Hartley und Zisserman, 2003] Hartley, Richard und A. Zisserman (2003). *Multiple view geometry in computer vision.* Cambridge university press. *(Seite: 1)*

[He et al., 2017] He, Kaiming, G. Gkioxari, P. Dollár und R. Girshick (2017). *Mask r-cnn*. In: *Proceedings of the IEEE international conference on computer vision*, S. 2961–2969. *(Seite: 74)*

[Jure Leskovec, 2018] Jure Leskovec, William L. Hamilton, Rex Ying Rok Sosic (2018). *Representation Learning on Networks*. `http://snap.stanford.edu/proj/embeddings-www/files/nrltutorial-part2-gnns.pdf`. Accessed: 2019-12-28. *(Seite: 6, 7, 8)*

[Kipf und Welling, 2016] Kipf, Thomas N und M. Welling (2016). *Semi-supervised classification with graph convolutional networks*. arXiv preprint arXiv:1609.02907. *(Seite: 24)*

[Krizhevsky et al., 2012] Krizhevsky, Alex, I. Sutskever und G. E. Hinton (2012). *Imagenet classification with deep convolutional neural networks*. In: *Advances in neural information processing systems*, S. 1097–1105. *(Seite: 17)*

[Lê, 2018] Lê, Phúc (2018). *Create 3D model from a single 2D image in PyTorch*. `https://medium.com/vitalify-asia/create-3d-model-from-a-single-2d-image-in-pytorch-917aca00bb07`. Accessed: 2019-11-18. *(Seite: 18)*

[Lasang et al., 2014] Lasang, P., S. M. Shen und W. Kumwilaisak (2014). *Combining high resolution color and depth images for dense 3D reconstruction*. In: *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, S. 331–334. *(Seite: 59)*

[Leibe et al., 2016] Leibe, Bastian, J. Matas, N. Sebe und M. Welling (2016). *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV*, Bd. 9908. Springer. *(Seite: 12)*

[LLC, 2020] LLC, Google (2020). *3D Warehouse*. `https://3dwarehouse.sketchup.com`. Accessed: 2020-03-01. *(Seite: 47)*

[Martinez und Gill, 2019] Martinez, J. B. und G. Gill (2019). *Comparison of Pre-Trained vs Domain-Specific Convolutional Neural Networks for Classification*

of Interstitial Lung Disease. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, S. 991–994. *(Seite: 56)*

[Maturana und Scherer, 2015] Maturana, Daniel und S. Scherer (2015). *Voxnet: A 3d convolutional neural network for real-time object recognition*. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, S. 922–928. IEEE. *(Seite: 21)*

[Pontes et al., 2018] Pontes, Jhony K, C. Kong, S. Sridharan, S. Lucey, A. Eriksson und C. Fookes (2018). *Image2mesh: A learning framework for single image 3d reconstruction*. In: *Asian Conference on Computer Vision*, S. 365–381. Springer. *(Seite: 13, 24)*

[Riegler et al., 2017] Riegler, Gernot, A. Osman Ulusoy und A. Geiger (2017). *Octnet: Learning deep 3d representations at high resolutions*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, S. 3577–3586. *(Seite: 33)*

[Shima et al., 2017] Shima, Y., Y. Nakashima und M. Yasuda (2017). *Pattern augmentation for handwritten digit classification based on combination of pre-trained CNN and SVM*. In: *2017 6th International Conference on Informatics, Electronics and Vision 2017 7th International Symposium in Computational Medical and Health Technology (ICIEV-ISCMHT)*, S. 1–6. *(Seite: 56)*

[Simonyan und Zisserman, 2014] Simonyan, Karen und A. Zisserman (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556. *(Seite: 37)*

[Smith et al., 2019] Smith, Edward J, S. Fujimoto, A. Romero und D. Meger (2019). *GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects*. arXiv preprint arXiv:1901.11461. *(Seite: 24)*

[Su et al., 2015] Su, Hao, C. R. Qi, Y. Li und L. J. Guibas (2015). *Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views*. In: *Proceedings of the IEEE International Conference on Computer Vision*, S. 2686–2694. *(Seite: 13)*

[Tatarchenko et al., 2017] Tatarchenko, Maxim, A. Dosovitskiy und T. Brox (2017). *Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs*. In: *Proceedings of the IEEE International Conference on Computer Vision*, S. 2088–2096. *(Seite: 22, 26, 33, 35)*

[Tatarchenko et al., 2019] Tatarchenko, Maxim, S. R. Richter, R. Ranftl, Z. Li, V. Koltun und T. Brox (2019). *What do single-view 3d reconstruction networks learn?*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, S. 3405–3414. *(Seite: 13)*

[Volkhonskiy et al., 2019] Volkhonskiy, Denis, E. Muravleva, O. Sudakov, D. Orlov, B. Belozerov, E. Burnaev und D. Koroteev (2019). *Reconstruction of 3D Porous Media From 2D Slices*. arXiv preprint arXiv:1901.10233. *(Seite: 20)*

[Wang et al., 2018] Wang, Nanyang, Y. Zhang, Z. Li, Y. Fu, W. Liu und Y.-G. Jiang (2018). *Pixel2mesh: Generating 3d mesh models from single rgb images*. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, S. 52–67. *(Seite: 24, 26, 37, 40, 43, 45, 52, 55)*

[Wang et al., 2017] Wang, Peng-Shuai, Y. Liu, Y.-X. Guo, C.-Y. Sun und X. Tong (2017). *O-cnn: Octree-based convolutional neural networks for 3d shape analysis*. ACM Transactions on Graphics (TOG), 36(4):72. *(Seite: 22)*

[Wang et al., 2019] Wang, Peng-Shuai, C.-Y. Sun, Y. Liu und X. Tong (2019). *Adaptive o-cnn: A patch-based deep representation of 3d shapes*. ACM Transactions on Graphics (TOG), 37(6):217. *(Seite: 22, 23)*

[Wu et al., 2019] Wu, Yuxin, A. Kirillov, F. Massa, W.-Y. Lo und R. Girshick (2019). *Detectron2*. `https://github.com/facebookresearch/detectron2`. *(Seite: 74)*

[Wu et al., 2015] Wu, Zhirong, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang und J. Xiao (2015). *3d shapenets: A deep representation for volumetric shapes*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 1912–1920. *(Seite: 18)*

[Xie et al., 2019] Xie, Haozhe, H. Yao, X. Sun, S. Zhou und S. Zhang (2019). *Pix2vox: Context-aware 3d reconstruction from single and multi-view images.* In: *Proceedings of the IEEE International Conference on Computer Vision*, S. 2690–2698. *(Seite: 16)*

[Yang et al., 2016] Yang, L., L. Jing, J. Yu und M. K. Ng (2016). *Learning Transferred Weights From Co-Occurrence Data for Heterogeneous Transfer Learning.* IEEE Transactions on Neural Networks and Learning Systems, 27(11):2187–2200. *(Seite: 56)*

[Zhang et al., 2018] Zhang, K., L. Guo und C. Gao (2018). *Optimization Method of Residual Networks of Residual Networks for Image Classification.* In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, S. 321–325. *(Seite: 41)*