Master's thesis

2019

Sondre Bø Kongsgård

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Sondre Bø Kongsgård

A Deep Learning-Based 3D Vision Pipeline for Shape Completion of 3D Objects

June 2019

NTNU
Norwegian University of
Science and Technology

A Deep Learning-Based 3D Vision Pipeline for Shape Completion of 3D Objects

Sondre Bø Kongsgård

Cybernetics and Robotics
Submission date: June 2019
Supervisor: Annette Stahl
Co-supervisor: Ekrem Misimi

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

In order for robots to operate safely and effectively, they must be aware of their surroundings. This includes knowledge of the 3D geometry and position of objects in the scene. The purpose of this thesis is to develop a pipeline which processes a single RGB-D image in real time, and outputs a more complete rendition of the scene around the robot. This problem is concerned with (1) detection and segmentation of objects, (2) point cloud registration, and (3) shape completion of the objects of interest. The objects can be both deformable and compliant, and could potentially be different in various scenarios, which calls for a data-driven approach that can use prior information to infer more details than what is directly given in an image. A series of deep learning networks combined into an overall shape completion pipeline is therefore proposed in this MSc thesis.

Abstract

Single-view shape completion entails the problem of estimating the complete geometry of objects from a single partial observation, and is at the core of many vision and robotics applications. This task could be considered an upsampling process, as the goal is to introduce new data that cannot be directly inferred from the given data, but which can be implied by comparing the partial scan to similar geometries in a prior dataset. Existing shape completion methods use structural assumptions about the underlying shape, but this limits the potential usage and the generalization of the resulting perception system.

In recent years, deep learning has significantly improved the performance of many applications in computer vision, including 3D machine learning problems such as shape completion. Even though its popularity has escalated, it is not always easy to apply deep learning to real-world problems. Especially in the related work for deep learning on shape completion there has so far only been limited results in successfully transferring the final agents from inferring complete shapes of the objects used in the training dataset to those found in depth scans of the real world. Either the resulting geometries are very coarse (i.e. considerably low resolution) or they do not fully match the given partial observations.

In this thesis, I investigate whether splitting the shape completion agent into multiple deep learning architectures might be helpful in adapting the learning-based methods to the real-world domain. By defining the problem as a series of components including semantic instance segmentation, point cloud registration, and shape completion in isolation, I am able to fully utilize the latest advances in the state-of-the art of the respective fields. Consequently, the input and output from each component is made more similar to the data used during training (which is customized for each component) and thus narrows the gap between the training environment and real-world applications. This approach also constitutes a framework which can be further built upon. In particular, each component in the perception system can be exchanged for future improved methods in the literature.

The proposed perception system predicts the 3D shape of the objects of interest in a scene. Since predictions are inferred directly from neural networks and no optimization methods are used, shape completion is done in the order of about a second, which means that the system can be used for online processing applications. Furthermore, training of the models can be done using a simulation environment, which simplifies the generation of data and reduces cost of adapting the system to new applications. These advantages are inherent to the presented approach, and thus strongly motivates future research on the proposed framework.

Keywords: *3D reconstruction, single-view shape completion, robot vision, semantic instance segmentation, point cloud registration, RGB-D imaging*

Sammendrag

Enkelt-vis formfullføring omhandler problemet hvor målet er å estimere den komplette geometrien fra delvise observasjoner av objekter, noe som er essensielt i mange applikasjoner innen datasyn og robotikk. Denne oppgaven kan anses som en prosess som oppjusterer oppløsningen til den gitte dataen. Siden den manglende dataen ikke kan finnes direkte må den istedet finnes implisitt ved å sammenligne den delvise observasjonen med lignende geometrier i et datasett. Nåværende formfullføringsmetoder bruker strukturelle antakelser om den underliggende formen, men dette begrenser de potensielle bruksområdene og generaliserbarheten til det resulterende persepsjonssystemet.

I de senere år har dyp læring forbedret ytelsen betydelig av mange applikasjoner innen datasyn, inkludert 3D maskinlæring anvendt på formfullføring. Selv om populariteten til dyp læring har økt markant, så er det ikke alltid lett å anvende disse metodene til applikasjoner i den virkelige verden. Relatert arbeid innen dyp læring for formfullføring har foreløpig gitt særdeles begrensede resultater for overføring av de lærte metodene til den virkelige verden. Enten er den resulterende geometrien ganske grov (dvs. svært lav oppløsning) eller så passer den ikke helt med den gitte delvise observasjonen.

I denne avhandlingen undersøker jeg om det å dele opp formfullførings-oppgaven til flere dype læringsarkitekturer kan være gunstig for å tilpasse de læringsbaserte metodene til den virkelige verden. Ved å definere problemet som en serie med komponenter (semantisk forekomstsegmentering, punktskyregistrering og isolert formfullføring) kan jeg fullt utnytte de siste fremskrittene innen hvert felt. Dermed vil inngangs- og utgangsdataen for hver komponent være mer lik den dataen som er brukt under treningen, og som er tilpasset hver komponent, og dermed minke gapet mellom treningsmiljøet og virkelige anvendelser. Denne tilnærmingen danner også et grunnlag for et rammeverk som kan påbygges videre. Hver komponent i det resulterende persepsjonssystemet kan byttes ut med framtidige utbedrede metoder.

Det foreslåtte persepsjonssystemet kan forutsi 3D-formen til de utvalgte objektene i en scene. Siden prediksjonene kommer direkte fra nevrale nettverk uten bruk av noen optimeringsmetoder, så kan hele formfullføringen gjøres på omtrent ett sekund, noe som betyr at systemet kan brukes i sanntidsapplikasjoner. Videre kan treningen av modellene gjøres i et simuleringsmiljø, slik at datagenerering forenkles og kostnader vedrørende tilpassing av systemet til nye anvendelser reduseres. Disse fordelene er iboende til den nevnte metoden, og motiverer derfor sterkt framtidig forskning på det foreslåtte persepsjonssystemet.

Nøkkelord: *3D-rekonstruksjon, formfullføring, robot vision, semantisk forekomstsegmentering, punktskyregistrering, RGB-D bildebehandling*

Preface

This MSc thesis concludes my Master of Science in Cybernetics and Robotics at the Norwegian University of Science and Technology, at the Department of Engineering Cybernetics. The thesis is a continuation of my specialization project completed during the fall of 2018.

The assignment is formulated in collaboration with SINTEF Ocean in the project called iProcess (`www.iprocessproject.com`) where SINTEF Ocean is working with robotic manipulation of compliant food objects based on visual information from an RGB-D camera in an eye-in-hand configuration on a 7-DOF manipulator. Robot grasping necessitates a detailed understanding of the scene, including the complete 3D geometry of the objects of interest, and thus motivates the idea for this thesis.

I would like to thank my supervisors Annette Stahl and Ekrem Misimi for their support, guidance, and feedback during the completion of this project. They have steered me in the right direction, and helped me focus on the most important problems at hand. I would also like to thank PhD candidate Jonatan Dyrstad for conversations on 3D modelling and for recommending the PyTorch framework, and PhD candidate Simen Haugo for discussions on signed distance functions. Additionally, I find SINTEF Ocean to be very generous in allowing me to use the tools and workstations in their robot lab MARO.

This work was completed on a workstation with a Nvidia GeForce GTX 1080 graphics card and an Intel RealSense D435 RGB-D camera. A variety of software was used for 3D modelling and editing; including Meshlab, Meshroom, CloudCompare, Blender, and UnrealROX. Python is used as the programming language of choice, with the following core frameworks and libraries: PyTorch, Open3D, Scikit-Learn, and Trimesh. All the mentioned software are open-source.

Trondheim, June 17, 2019

Sondre Bø Kongsgård

Sondre Bø Kongsgård

Contents

Problem Description **i**

Abstract **ii**

Sammendrag **iii**

Preface **iv**

Contents **v**

1 Introduction **3**
- 1.1 Motivation 3
- 1.2 Background 4
- 1.3 Problem Formulation 4
- 1.4 Research Questions 6
- 1.5 Contributions 7
- 1.6 Thesis Outline 8

2 Theoretical Background **9**
- 2.1 3D Shape Reconstruction 9
  - 2.1.1 Capturing Depth 9
  - 2.1.2 3D Representations 15
  - 2.1.3 Metrics 17
- 2.2 Deep Learning 20
  - 2.2.1 Neural Networks 21
  - 2.2.2 Convolutional Neural Networks 27
  - 2.2.3 Representation Learning Techniques 30

3 Related Work **38**
- 3.1 Depth Image Denoising and Enhancement 38
- 3.2 Semantic Segmentation and Object Detection 39
- 3.3 Point Cloud Registration 43
- 3.4 3D Shape Completion 44
  - 3.4.1 Geometry-based shape completion 44
  - 3.4.2 Alignment-based shape completion 47
  - 3.4.3 Learning-based shape completion 48
- 3.5 Datasets 51

4 Methodology, Implementation and Results **54**

4.1 Preparation of Datasets . . . 54
4.1.1 3D Model Dataset . . . 54
4.1.2 3D Scene Dataset . . . 58
4.2 RGB-D Semantic Segmentation . . . 59
4.2.1 Network Architecture . . . 59
4.2.2 Implementation Details . . . 60
4.2.3 Evaluation . . . 62
4.3 Shape Completion . . . 63
4.3.1 Network Architectures . . . 64
4.3.2 Implementation Details . . . 70
4.3.3 Evaluation . . . 71
4.4 Point Cloud Registration . . . 72
4.4.1 Network Architecture . . . 74
4.4.2 Implementation Details . . . 76
4.4.3 Evaluation . . . 76
4.5 Combining the Building Blocks to Create a 3D Vision Pipeline . . . 78

5 Conclusion 85
5.1 Summary . . . 85
5.2 Future Work . . . 86

A Tools 89
A.1 Hardware Environment . . . 89
A.2 Software Environment . . . 89
A.2.1 Python Libraries . . . 89
A.2.2 3D Computer Graphics Software . . . 90

Bibliography 92

Abbreviations

AI	Artificial Intelligence
CD	Chamfer Distance
CE	Cross-Entropy
CNN	Convolutional Neural Network
COS	Center of Symmetry
CPU	Central Processing Unit
DGCNN	Dynamic Graph Convolutional Neural Network
EMA	Exponential Moving Average
EMD	Earth Mover Distance
FoV	Field of View
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IoU	Intersection over Union
JIT	Just-in-Time
JSD	Jensen-Shannon Divergence
MAP	Maximum-a-Posteriori
MISE	Multiresolution Isosurface Extraction
MSE	Mean Squared Error
NAG	Nesterov Accelerated Gradient
PDA	Principal Symmetry Axes
PRST	Planar Reflective Symmetry Transform
R-CNN	Region-based Convolutional Neural Network

R-FCN	Region-based Fully Convolutional Network
ReLU	Rectified Linear Unit
RoI	Region of Interest
RPN	Region Proposal Network
SDF	Signed Distance Function
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
TSDF	Truncated Signed Distance Fields
VAD	Variational Autodecoder
VAE	Variational Autoencoder
VR	Virtual Reality

CHAPTER 1 Introduction

1.1 Motivation

In any interactions with the physical world, 3D geometry and physics information is mandatory. For a robot to plan object manipulation, it needs a representation of the 3D scene. Finding a good such representation is a difficult problem, and the requirements could also be affected by the task at hand. For instance, how quickly the robot needs to operate can depend on the surrounding workstation. Consequently, there are various levels of accuracy and performance requirements of the perception system that will process the visual information gained from the robot's sensors.

SINTEF Ocean through projects iProcess (`www.iprocessproject.com`), MarineRobotics and activeVision has developed an active vision methodology for an eye-in-hand robot configuration based on an RGB-D camera, rich point clouds and visual servoing [99]. The approach developed by Dr. Ekrem Misimi is validated for different objects and is currently under preparation for publication. The approach involves both simulation and real-world implementation and validation. The simulation is done by importing rich point clouds of objects scanned with the RGB-D camera to a simulation environment, implementing classical machine learning methods for active vision. The real-world setup is developed to scan objects in real time with the RGB-D camera, process the point clouds in real-time, and by robot control by means of visual servoing, active vision is achieved in an automatic manner.

While giving a detailed and high-resolution 3D model of the scene, the aforementioned method has some limitations. The robot is often unable to gather all necessary views in order to generate the full 3D models of the objects in the scene. This limitation could be due to obstructions in the working space, such as overlapping objects or when the objects are placed such that the robot cannot move to scan them from all views, e.g. when they are too far away or in the corner of a box. Additionally, scanning the objects in a scene from all views is time consuming. The active vision method by Misimi et al. estimates an inference time of up to 10 seconds per object. For applications such as handling objects on a conveyor belt, this is too slow.

Recent studies indicate that single-view 3D reconstruction is possible, which would address all the problems inherent in active vision (i.e. multi-view) methods. Even though single-view 3D reconstruction is heavily under-constrained, the most likely completion of the 3D geometries in a scene can be inferred from priors encoded in a model. By representing the scene as a collection of 3D objects with rich attributes, the problem of reconstructing the whole scene can be split into smaller sub-problems where the task is to infer the shape completion of each individual object. This would not only allowing for the completion of arbitrary scenes, but also potentially give a direction on how to create an artificial system with rich compositional visual intelligence. Moreover, the inference time would be reduced to less than a second, allowing for high-paced real-time applications.

1.2 Background

Computer vision has evolved tremendously in the past several years. From simple image processing techniques, the field now has merit in sub-domains such as scene reconstruction, object recognition, and 3D pose estimation. In some cases, image classification has been shown to outperform humans [54, 134], indicating an enormous potential for future endeavours. Recent technological advances such as high-resolution imaging sensors and powerful programmable GPU hardware offer new possibilities for researchers. Whereas traditionally the field has required rigorous mathematical models, the advent of large-scale image datasets and deep convolutional neural networks moved the field into being more data-driven, where labeled data can supervise the learning of an optimal inference system.

iProcess, a SINTEF Ocean research project, has a main objective of developing novel concepts and methods for flexible and sustainable food processing. One of their work packages, Flexible Processing Automation, focuses on robots capable of using computer vision to recognize and localize compliant objects [60, 110, 100, 61]. Specifically, handling of food objects like meat and fish is of special interest. Currently, the food processing industry requires manual labor to tasks such as removing the breast fillet from a chicken. iProcess envisions assigning the responsibility of such tasks to robots, with potentially higher accuracy than humans so more of the raw materials could be preserved. Consequently, food production will be more sustainable, both in terms of profitability and reduction of food waste.

In order to sensibly interact with the 3D physical environment, the robot must possess the ability to sense and understand its surroundings. In this context, this means that it needs to capture physical signals in the environment and record them digitally. Additionally, this information must be processed to infer representations of the environment, which could then be used for decision making such as how to move the robot so it can grasp an object. Therefore, robots are typically equipped with a camera in an eye-in-hand configuration, i.e. the camera is placed on the end joint of the robot. At SINTEF Ocean, they employ the *manipulator* (robot arm) Franka Emika Panda with the RGB-D camera Intel RealSense D435. This camera captures depth images in addition to color images, and thus encodes 3D information of the scene.

1.3 Problem Formulation

The main aim of this thesis is to complete the 3D geometries of the objects in a scene given a partial observation, such as a single view from an RGB-D camera. The resulting perception system should segment the objects from the rest of the scene, infer their full 3D shapes from the partial observation, and render a complete 3D representation of the scene where the objects of interest have a complete geometry. The robot can then determine how it should move to position its end-effector (i.e. gripper) around the objects for grasping.

For instance, the scene could be some objects laying on top of a table as in Figure 1.1. While the complete geometry of the orange in Figure 1.1(a) could be represented as a point cloud, as in Figure 1.2(a), only what is presented in Figure 1.2(b) can be directly inferred from a single-view partial scan from an RGB-D camera. This illustrates the natural occlusion inherent of physical objects – it is not possible to see the back of an object. However, other physical occlusions are often present in a scene as well. Objects could partially cover each other, as in Figure 1.1(b). It could be desirable to complete the geometry of each of these

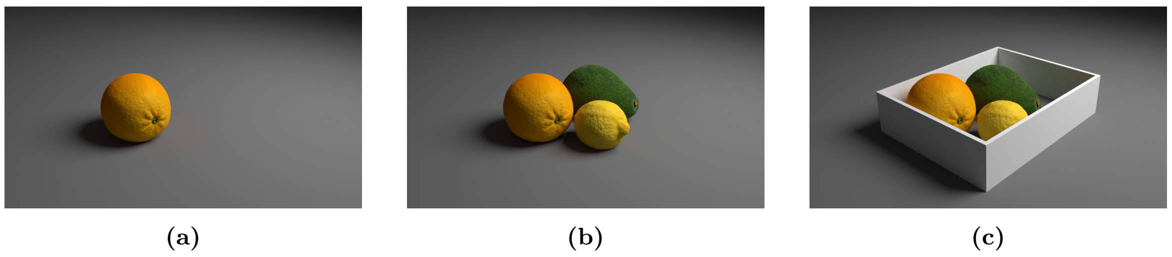


Figure 1.1: A simple representation of objects in a scene illustrating some examples scenarios on how an object can be occluded from the view in the camera direction. Figure 1.1(a) shows an example of natural occlusion – there is no way for the camera to see the backside of the orange from the current view. The object can also be partially occluded by other objects in the vicinity, such as in Figure 1.1(b). Moreover, an interesting scenario where single view shape completion is very much desired is when the object is in the corner of a box. Here, even if we would chose active vision or multi-view shape completion, the physical barrier of the box makes it impossible to see the occluded part of the object at the corner of the box. Physical occlusions such as the walls of a box could hinder the camera to see the objects, as illustrated in Figure 1.1(c). The same effect would also occur if a hand or an end-effector picked up an object, as it would most likely cover some part of it to be able to grasp it. The scenes in this illustration consist of objects from a custom dataset compiled as part of this thesis, and were rendered in Blender.

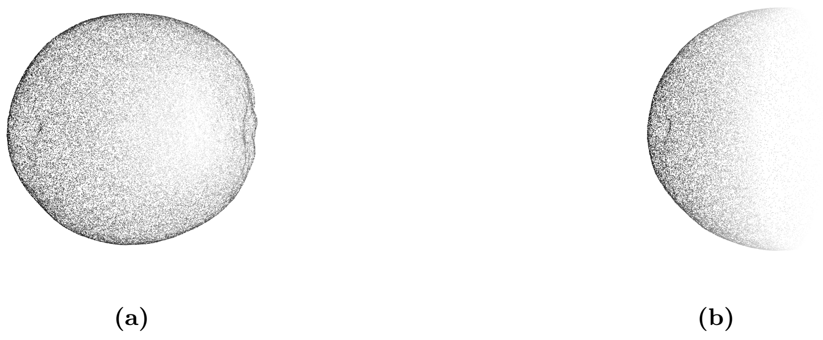


Figure 1.2: **Left:** A complete point cloud of an orange. **Right:** A partial point cloud of an orange, which would be what a camera system would see if it was viewing the orange from the left side of the image. The color information has been removed and replaced by a grayscale map to better illustrate the depth as seen from the camera.

objects, which is something that needs to be addressed. The objects of interest could also be partially covered by other structures such as walls, as illustrated in Figure 1.1(c), or the end-effector of the robot.

In this work, the problem is considered as a consecutive set of various subproblems, including segmentation, alignment, and completion. Deep learning methods have proven to give overwhelmingly better results than traditional methods, and are therefore of interest to research further in the context of this work. Their performance will be necessary in order to find satisfactory approaches to each of these problems, as they are highly complex. For segmentation, state-of-the art approaches have only come to a level where they solve the problems in a satisfactory manner. On the other hand, state-of-the art shape completion agents are still to be considered only proof-of-concepts, as currently scans of real-world objects are not considered for completion. Further, they are limited to completing only partial scans which are in canonical pose (the scale, translation, and rotation is known prior to completion) and doesn't take real-world partial scans into consideration. Therefore, in the light of the state-of-the art, it is clear that this MSc thesis has an exceedingly ambitious goal. The way the problem is defined puts the goal several steps ahead of the current related work.

To emphasize, the work presented in this MSc thesis is a report over research investigations carried out to inventively improve and adapt the recent advances in deep learning on 3D data to perform shape completion of scans of real-world objects. Therefore, the presented applications to shape completion are novel and innovative.

Aim of the Study

The main aim of this thesis is to complete the 3D geometries of the objects in a scene given a partial observation, such as a single view from an RGB-D camera.

1.4 Research Questions

There is an increasing number of published papers on shape completion, where the goal is to find the complete 3D geometry given a partial observation of an object. Traditionally, the approaches have consisted of aligning a dataset model to fit the partial observation. However, recent efforts have introduced deep learning as a plausible method. These have currently only been applied to curated training datasets, and not real depth scans. Thus, the first research question becomes:

> *(R1): Is shape completion of real-world objects based on a single-view partial scan feasible?*

To address this question with data-driven deep learning models, it is necessary to find a suitable method of transforming the partial scan of an object such that it has similar properties to those given in the dataset used to train the model. This leads to the second research question:

> *(R2): How can the partial scan of an object be digitally represented in a similar manner as the objects used in the prior dataset?*

Real-world shape completion turns out to consist of multiple subproblems, such as semantic segmentation and point cloud registration, that are not considered in the current methods on shape completion. These subproblems, however, have been addressed in isolation where deep learning is used in the state-of-the art methods. Several deep learning models, one for each sub-problem, must therefore be designed and implemented. They will then have to be combined into an overall pipeline to solve the main aim, but this sequencing of deep learning models is uncommon in the literature. The final research question is therefore the following:

> *(R3): Can multiple deep learning methods be combined in a succeeding fashion to create a robust perception system, or will image noise and limited prior knowledge hinder this?*

1.5 Contributions

The main research interest in this work is to develop a perception system which can accurately predict the complete geometry of objects in a partial scan of a scene. The idea is to do this by improving, adapting and combining the state-of-the art 3D deep learning methods. To the best of the author's knowledge, this work is the first effort to perform real-world single-view shape completion using such methods.

Contributions introduced in this thesis are described below:

- **Semantic Segmentation Network:** A state-of-the art ResNet-like architecture was adapted to high resolution color and depth images. By using two separate branches in the encoder, both the color and depth information are utilized fully, which has been a challenge in previous network architectures. With this network, accurate masks can be achieved of any scene which resemble the images used in the training dataset. Further, the network has shown a high generalizability in allowing training used on images from a simulation environment to also predict masks on real-world images.

- **Shape Completion Network:** A learning-based shape completion method that operates on sampled level sets, and which can take partial observations of objects in the form of point clouds as inputs to produce smooth complete objects was designed as part of this work. By representing the output as a continuous function, the problem of shape completion is restated as a boundary decision problem. Thus, the surface can be represented implicitly, and this is easier for a neural network to learn. The results show that the shape completions using the network closely follow the given partial observations on test data of the classes used during training.

- **Point Cloud Registration Network:** One of the hardest problems in this thesis was to find a robust way to align the segmented point clouds of objects from world coordinates into a representation where they are axis-aligned and unit-scaled, and thus similar to the objects in the prior dataset the shape completion agent had trained on. The reason for the complexity is that the objects can be of different scale, rotation, and translation. Additionally, only partial scans of the objects are given, and an exact match might not exist in the dataset of which they should be aligned to. In the literature, this problem is referred to as point cloud registration, and current methods are only able to perform local optimizations when only rotation and translation are considered. In this work, I adapt a state-of-the art neural network to learn mappings between point clouds. This network is able to avoid converging the alignment to local optimas, and instead align the point clouds regardless of their initial orientation. A heuristic assumption is added to account for the scaling factor.

- **3D Vision Pipeline:** A 3D vision pipeline which combines the methods mentioned above into a coherent perception system. It takes an RGB-D image of a scene as input, for then to segment and complete objects of interest in the scene. This constitutes a novel framework which adapts deep learning networks into a system that can be used on real-world applications such as robotic vision.

Additionally, a dataset preparation and preprocessing approach has been developed, which can generate aligned and unit-sphere scaled models as point clouds or signed distance fields from arbitrary compilations of CAD or mesh models. This was used in this work to create a dataset consisting of food objects, which can be used in conjunction with the popular ShapeNet [13] or ModelNet [160] datasets.
The terminology and concepts used above are described in detail in Chapter 2.

1.6 Thesis Outline

The remainder of the thesis is organized into the following chapters.

Chapter 2, Theoretical Background, explains the necessary theoretical foundation to build upon. It presents the relevant literature from computer vision and deep learning.

Chapter 3, Related Work, presents a thorough study of related work. Several fields are covered, and each section includes a discussion on the fields' relevance and use for the problem in this work. The most recently published papers are highlighted, as these fields have developed rapidly in the last years, and the most recent papers present the ideas that will be built upon in this work. Additionally, extra attention is given to related work on shape completion, as that is the most important subproblem in this thesis.

Chapter 4, Methodology, Implementation and Results, presents the details and performance of the implemented models. It covers how both scene and model datasets were prepared and converted into the required format for the chosen network agents. Furthermore, it presents details on how the deep learning networks for semantic segmentation, shape completion, and point cloud registration were implemented. It also gives a reflection on how the proposed 3D vision pipeline approach can be used in a real-world setting.

Chapter 5, Conclusion, provides a short summary and serves as the conclusion of this thesis.

CHAPTER 2

Theoretical Background

This chapter presents the theoretical background of 3D reconstruction and deep learning in connection to the work completed as part of this thesis. The next chapter builds upon this theory to present the related work

3D shape reconstruction refers to inferring 3D information from RGB- and/or depth-images. Depth images can be acquired from a range of various depth sensing techniques, and this is explained in Section 2.1.1. Further, 3D information can be stored in multiple formats such as meshes, voxels, point clouds, and more. An overview of these 3D representations is given in Section 2.1.2. Finally, the quality of the 3D reconstruction can be quantitatively measured by using a metric to determine the similarity between the reconstruction and a ground truth, which is described in Section 2.1.3.

Deep-learning methods are data-driven approaches that have the property of being universal function approximators, which will in this work be used for finding a mapping from a partial scan of an object to its complete geometry. It will also be used for segmentation of RGB-D images. In this chapter, an in-depth presentation of deep learning will be given, including feed-forward networks, convolutional neural networks, and generative models. A literature review on the state-of-the art approaches on segmentation and shape completion will be presented in the next chapter.

2.1 3D Shape Reconstruction

2.1.1 Capturing Depth

Depth is important in computer vision for many applications. With depth maps, a scene can easily be segmented into foreground and background and track objects. There are many ways to capture depth from a scene, but they can be roughly categorized into two main classes: Either the depth is directly inferred from the scene through a depth camera system (with a full overview given in Table 2.1), or the depth information is reconstructed from one or more images. Here, the most frequently used techniques from each class are presented: structured lighting and stereo vision. The theory for these methods are mostly extracted from the book *Concise Computer vision* [73], and with some details from *Computer Vision Metrics: Survey, Taxonomy and Analysis* [79] and *Computer Vision: Algorithms and Applications* [145].

Structured lighting: An example of a depth camera system, where 3D information is inferred by the projection of a light pattern under calibrated geometric conditions onto an object. Typically, one camera and one light source are used. Calibration includes determining the pose of the light source with respect to the camera. Refer to Figure 2.1, where it is assumed

Depth Sensing Technique	# of Sensors	Illumination Method	Characteristics
Parallax and Hybrid Parallax	2/1/array	Passive – Normal lighting	Positional shift measurement in FoV between two camera positions, such as stereo, multi-view stereo, or array cameras
Size Mapping	1	Passive – Normal lighting	Utilizes color tags of specific size to determine range and position
Depth of Focus	1	Passive – Normal lighting	Multi-frame with scanned focus
Differential Magnification	1	Passive – Normal lighting	Two-frame image capture at different magnifications, creating a distance-based offset
Structured light	1	Active – Projected lighting	Multi-frame pattern projection
Time of Flight	1	Active – Pulsed lighting	High-speed light pulse with special pixels measuring return time of reflected light
Shading shift	1	Active – Alternating lighting	Two-frame shadow differential measurement between two light sources at different positions
Pattern spreading	1	Active – Multi-beam lighting	Projected 2D spot pattern exapanding at different rate from camera lens field spread
Beam tracking	1	Active – Lighting on object(s)	Two-point light sources mounted on objectes in FoV to be tracked
Spectral Focal Sweep	1	Passive – Normal lighting	Focal length varies for each color wavelength, with focal sweep to focus on each color and compute depth
Diffraction Gratings	1	Passive – Normal lighting	Light passing through sets of gratings or light guides provides depth information
Conical Radial Mirror	1	Passive – Normal lighting	Light from a conical mirror is imaged at different depths as a toroid shape, depth is extracted from the toroid

Table 2.1: Some methods for capturing depth information. Adapted from Computer Vision Metrics [79].

that the base distance b is given by light source calibration and that the angle α by controlled light plane sweeping.

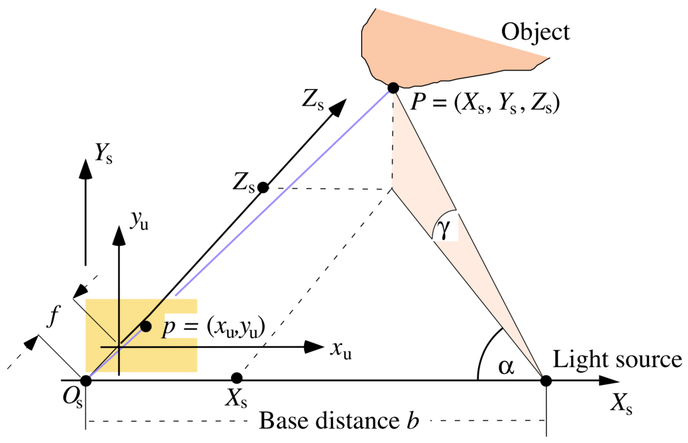


Figure 2.1: Structured lighting system. The projection centre is located at the origin O_s in a left-hand $X_sY_sX_s$ coordinate system. The image plane is parallel to the X_sY_s plane at focal distance f. For simplification, it is assumed that the light source is on the X_s axis. Courtesy of Concise Computer Vision [73].

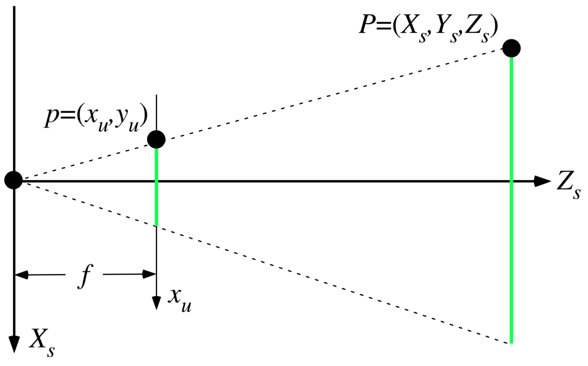


Figure 2.2: The central projection of a point $P = (X_s, Y_s, Z_s)$ into a pixel location $p = (x_u, y_u)$. The corresponding sides of the highlighted triangles are all in the same proportion. Courtesy of Concise Computer Vision [73].

The camera is positioned in a 3D space, with a left-hand $X_sY_sZ_s$ camera coordinate system. O_s is the projection centre, and the optical axis of the camera coincides with the Z_s-axis. The image plane is parallel to the X_sY_S plane, at focal distance $Z_s = f$. The ray theorem of elementary geometry says that f to Z_s is the same as x_u to X_s, and with analogous ratios in the Y_sZ_s plane (Figure 2.2), resulting in the equations

$$\begin{aligned} x_u &= \frac{fX_s}{Z_s} \\ y_u &= \frac{fY_s}{Z_s} \end{aligned} \tag{2.1}$$

Further, from the trigonometry of right triangles the following is true

$$\tan\alpha = \frac{Z_s}{b - X_s} \tag{2.2}$$

It then follows that

$$\begin{aligned} Z &= \frac{X}{x} \cdot f = \tan\alpha \cdot (b - X) \\ X \cdot \left(\frac{f}{x} + \tan\alpha\right) &= \tan\alpha \cdot b \end{aligned} \tag{2.3}$$

which gives the solution

$$\begin{aligned} X &= x \cdot \frac{\tan\alpha \cdot b}{f + x \cdot \tan\alpha} \\ Y &= y \cdot \frac{\tan\alpha \cdot b}{f + x \cdot \tan\alpha} \\ Z &= f \cdot \frac{\tan\alpha \cdot b}{f + x \cdot \tan\alpha} \end{aligned} \tag{2.4}$$

Stereo Vision: A binary vision method similar to the human visual system. Left and right views of a scene are captured, and the depth or distance to the visual points are calculated by first determining the corresponding points in each view. In Figure 2.3 two cameras are considered to follow the pinhole-camera model and positioned in general poses in projection centres O_1 and O_2. A 3D point P in $X_w Y_w Z_w$ world coordinates is projected into a point p_1 in image 1 and onto a corresponding point in p_2. The task is then to locate the point p_2 by starting at point p_1 in the first image. By looking at Figure 2.3, one can see that the points O_1, O_2, and the point P defines a plane which is called the *epipolar plane*. The same plane is defined by O_1, O_2, and p_1 in image 1, and is also intersected by the image plane of image 1 to form the *epipolar line*. It is evident that p_2 can only be found on this line in image 2, which means that searching for p_2 is reduced to a simpler problem of finding a corresponding problem on a line, rather than the entire image plane.

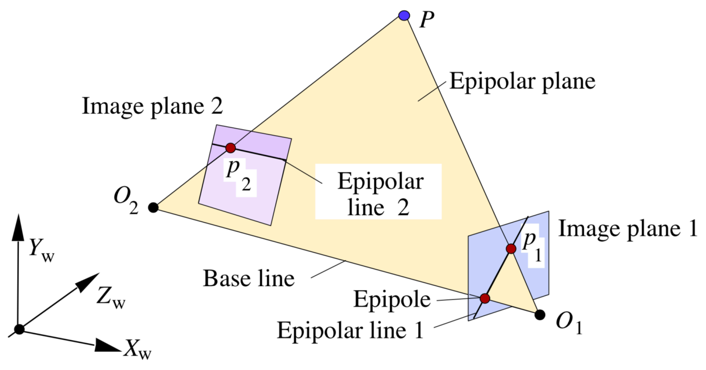


Figure 2.3: Epipolar geometry for two cameras in general poses. Courtesy of Concise Computer Vision [73].

Assume that the two cameras are virtually identical, and oriented the same way apart from one camera shifted by the distance b along the X_s-axis of the $X_sY_sZ_s$ camera coordinate system of the left camera. This setup is referred to as *canonical stereo geometry*, and results in two coplanar images of identical size $N_{\text{cols}} \times N_{\text{rows}}$, with parallel optic axes, identical effective focal length f, and collinear image rows. By utilizing the equations (2.1), the point $P = (X_s, Y_s, Z_s)$ is mapped to the undistorted image points

$$p_{uL} = (x_{uL}, y_{uL}) = \left(\frac{f \cdot X_s}{Z_s}, \frac{f \cdot Y_s}{Z_s}\right) \tag{2.5a}$$

$$p_{uR} = (x_{uR}, y_{uR}) = \left(\frac{f \cdot (X_s - b)}{Z_s}, \frac{f \cdot Y_s}{Z_s}\right) \tag{2.5b}$$

For undistorted coordinates, a pixel $p_L = (x_L, y)$ and its corresponding pixel $p_R = (x_R, y)$ have x-values where $x_R \leq x_L$. With the base distance denoted as $b > 0$ and f the focal length, Z_s can be eliminated from (2.5a) and (2.5b) by

$$Z_s = \frac{f \cdot X_s}{x_{uL}} = \frac{f \cdot (X_s - b)}{x_{uR}} \tag{2.6}$$

This equation can further be solved for X_s to yield

$$X_s = \frac{b \cdot X_{uL}}{x_{uL} - x_{uR}} \tag{2.7}$$

which can be inserted back into (2.6) to get

$$Z_s = \frac{b \cdot f}{x_{uL} - x_{uR}} \tag{2.8}$$

The final coordinate, Y_s, can be obtained by using this value of Z_s in (2.5a) and (2.5b):

$$Y_s = \frac{b \cdot y_u}{x_{uL} - x_{uR}} \tag{2.9}$$

Here, $y_u = y_{uL} = y_{uR}$.

It is generally desirable, however, to tilt the cameras slightly towards each other, such that the space of objects being visible in both cameras is maximized. This is referred to as a convergent setup, illustrated in Figure 2.4.
The Cartesian coordinate system XYZ can be transformed into the $X_LY_LZ_L$ system by a rotation $-\theta$ about the Y-axis followed by a translation $b/2$ to the left on the $X - axis$

$$\begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 \cos(\theta) & \end{bmatrix} \begin{bmatrix} X - \frac{b}{2} \\ Y \\ Z \end{bmatrix} \tag{2.10}$$

$X_RY_RZ_R$ can be acquired in the same manner,

$$\begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 \cos(\theta) & \end{bmatrix} \begin{bmatrix} X + \frac{b}{2} \\ Y \\ Z \end{bmatrix} \tag{2.11}$$

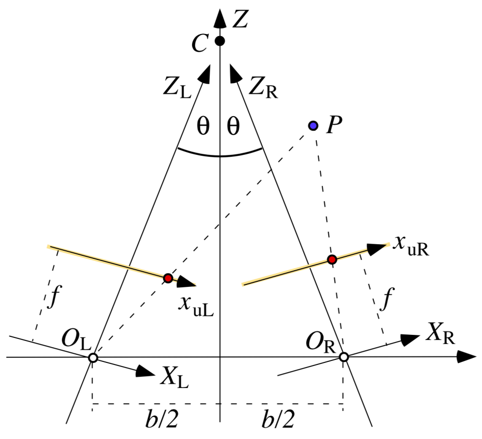


Figure 2.4: Convergent camera system. The left and right cameras are symmetric to the Z-axis, but tilted towards each other such that the optical axes intersect at the point of convergence C on the Z-axis. Courtesy of Concise Computer Vision [73].

A point $P = (X, Y, Z)$ projected into the left $x_{uL}y_u$ and the right $x_{uR}y_u$ coordinate systems can then be represented in the centred XYZ-coordinate system based on the central projection equations:

$$x_{uL} = \frac{f \cdot X_L}{Z_L} = f\frac{\cos(\theta)(X - \frac{b}{2}) + \sin(\theta) \cdot Z}{-\sin(\theta)(X - \frac{b}{2}) + \cos(\theta) \cdot Z} \tag{2.12a}$$

$$x_{uR} = \frac{f \cdot X_R}{Z_R} = f\frac{\cos(\theta)(X + \frac{b}{2}) - \sin(\theta) \cdot Z}{\sin(\theta)(X + \frac{b}{2}) + \cos(\theta) \cdot Z} \tag{2.12b}$$

$$y_u = \frac{f \cdot Y_L}{Z_L} = \frac{f \cdot Y_R}{Z_R} = f\frac{Y}{-\sin(\theta)(X - \frac{b}{2}) + \cos(\theta) \cdot Z} \tag{2.12c}$$

The following system of equations follows:

$$[-x_{uL} \cdot \sin\theta - f \cdot (\theta)]X + [x_{uL} \cdot \cos(\theta) - f \cdot \sin(\theta)]Z = -[\frac{b}{2}x_{uL} \cdot \sin(\theta) + \frac{b}{2}f] \tag{2.13a}$$

$$[x_{uR} \cdot \sin(\theta) - f \cdot \cos(\theta)]X + [x_{uR} \cdot \cos(\theta) + f \cdot \sin(\theta)]Z = -[\frac{b}{2}x_{uR}\sin(\theta) - \frac{b}{2}f] \tag{2.13b}$$

$$[-y_u \cdot \sin(\theta)]X + [-f]Y + [y_u \cdot \cos(\theta)]Z = [\frac{b}{2}y_u \cdot \sin(\theta)] \tag{2.13c}$$

which can easily be solved for X, Y, and Z. Thus, convergent stereo ends up being no more complex than canonical stereo in terms of calculating the world coordinates of a point.

As an aside, *disparity* is in the general case defined as the virtual shifts of a point $P = (X, Y, Z)$ projected into a point p_i in one image i to the point p_j in another image j,

$$\begin{aligned} \boldsymbol{d}_{ij} &= p_i - p_j \\ \boldsymbol{d}_{ji} &= p_j - p_i \end{aligned} \tag{2.14}$$

where the disparity value d_{ij} is defined as the magnitude $||\boldsymbol{d}_{ij}||_2$. For the special case of canonical stereo geometry, the disparity value is $x_L - x_R$ due to the restriction that $x_R \leq x_L$.

2.1.2 3D Representations

Depending on the intended use, shapes can be represented in a variety of convertible formats. Some of the most important 3D representations are illustrated in Figure 2.5.

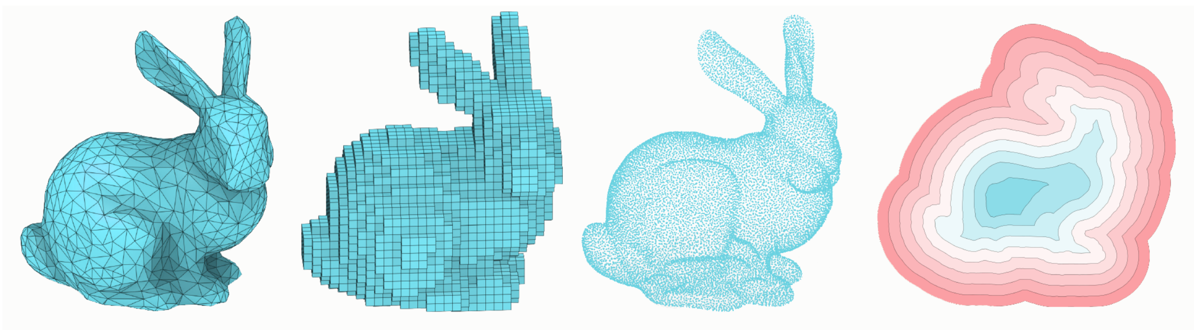


Figure 2.5: 3D representations of the Stanford bunny. Meshes are sparse water-tight representations which are slightly complicated by consisting of different components (vertices, edges, faces). Voxel representations are regular structures, but which have a high memory cost, are computationally inefficient to perform operations on, and generally are not able to represent detailed geometric information. Point clouds are simple sparse representations which do not encode any information about the surface of objects. Signed distance functions implicitly define the surface of an object with a continuous function, which theoretically means that they are representations with infinite resolution.

Polygon meshes store 3D information in a collection of connected vertices, edges, and faces. Each vertex is associated with neighboring vertices in a standard pattern such as a triangle describing the surface, each having a surface normal, 3D coordinates (x, y, z), color, and texture.

Meshes can also be stored in the *AABB tree* data structure, which is useful for performing efficient intersection and distance queries against sets of finite 3D geometric objects. The data structure is created by converting a mesh into primitives, of which a hierarchy of axis-aligned bounding boxes (AABBs) is constructed.

A *voxel volume* is a 3D volumetric data structure composed of a 3D array of voxels, which are volumetric elements like pixels are picture elements. Since the voxels are regularly spaced in the 3D array, their depth coordinates are not explicitly encoded. Each voxel may contain color and normal information.

An extension to the voxel representation is the *octree*, which has an adaptive grid size. This allows for lossless reduction of memory consumption compared to a regular voxel grid.

Point clouds are sparse representations of geometric shapes, where the surface is given by a set of 3D points. Each point in the cloud may contain information about 3D location (x, y, z), color, and normal.

Point clouds can also be represented as k-dimensional trees or *k-d trees* – binary search trees that are very useful for range and nearest neighbor searches. In this case, k will always be 3. Each level of a k-d tree splits all children along a specific dimension, using a hyperplane that is perpendicular to the corresponding axis. At the root of the tree all children will be split based on the first dimension. Each level down in the tree divides on the next dimension, returning to the first dimension once all others have been exhausted.

A *signed distance function* (SDF) is a continuous function that, for a given spatial point, outputs the point's distance to the closest surface, whose sign encodes whether the point is inside (negative) or outside (positive) of the watertight surface:

$$\mathrm{SDF}(\boldsymbol{x}) = s : \boldsymbol{x} \in \mathbb{R}^3,\ s \in \mathbb{R} \tag{2.15}$$

The underlying surface is implicitly represented by the *isosurface* of $\mathrm{SDF}(\cdot) = 0$.

Additionally, a *depth map image* can represent the 3D information. Each point in a 2D pixel array may contain color and depth information. This is typically the format of the raw depth data from RGB-D cameras.

2.1.2.1 Conversion between 3D Representations

It can often be of use to convert one 3D representation to another, as depending on the application a water-tight representation such as meshes, a regular representation such as voxels, a sparse representations such as point clouds, or a continuous representations such as SDFs might be desirable. The conversions are unfortunately not straight-forward and most often not lossless. In the following two paragraphs, the most common methods to convert from meshes to point clouds and from SDFs to meshes are presented, as they are used in the methodology in this work.

Mesh to Point Cloud: A mesh could intuitively be converted to a point cloud simply by selecting the vertices of the mesh. While this is a feasible strategy, it has several limitations. First, this only allows for a predefined number of points in the point cloud. Second, more points will be aggregated in areas of the mesh with a higher face density, resulting in an uneven and poor representation of the object.

Instead, points could be uniformly sampled on the mesh faces. The process consists of simply choosing a random triangle, and then generate a point within its bounds until a desired number of samples is reached. This sampling process still has the second issue mentioned above, i.e. it is more likely to randomly select a face in the higher density areas of the mesh.

One of the most common methods to sample points from a mesh is called *Poisson disk sampling*, which guarantees the minimum distance between each sample to be $2r$ [24]. Therefore, a disk of radius r centered on each sample does not overlap any other disk, which implicitly means that biased distributions of points are avoided. Since this method is implemented and used in this work, the pseudo-code of the algorithm is summarized in Algorithm 1.

Point clouds may also store the normal of the mesh face the point is sampled from. This can easily be added by identifying the face the point is sampled from, and calculate the normal from the vector product of two of its edges. Some mesh file formats such as .off also explicitly stores the normal for each mesh, which provides an alternative way of how the normal can be acquired. The resulting point cloud datastructure is a list of [x-coordinate, y-coordinate, z-coordinate, x-normal, y-normal, z-normal] for each point.

Signed Distance Function to Mesh: *Marching cubes* is the reference algorithm for converting a scalar field such as SDFs to a 2D surface mesh [91]. It requires a data volume

Algorithm 1 PoissonDiskSampling

```
 1: Given: number of points N
 2: SamplePool Pool = GenerateSamplePool(N)          ▷ Pre-generate samples on the mesh
 3: SpatialHashTable Cells = FillSpatialHashTable(Pool)    ▷ Fill a spatial index for fast
    access to samples
 4: RandomShuffle(Cells)
 5: Samples Samples                                  ▷ Random shuffle of the cells
 6: while Cells.IsNotEmpty() do                      ▷ Main loop
 7:     Cell Cell = ExtractCell(Cells)    ▷ Choose a cell with a probability proportional to the
    number of samples contained in it
 8:     Sample P = ExtractFromSamplePool(Cell, Pool)    ▷ Generate a valid sample inside
    the current cell by extracting it from the pre-computed sample pool
 9:     Samples.Add(P)
10:     if IsValid(P) then              ▷ Subdivide cell if necessary and update active cells
11:         RemoveCell(Cell, Cells)
12:     else
13:         SubdivideCell(Cell, Cells)
14:     end if
15: end while
16: return Samples
```

(a cuberille grid, equivalent to a voxel volume) and an isosurface value. The vertices on each cube in the volume are classified as as positive or negative, according to their comparison with a given isovalue, which in the case of SDFs is where SDF($\boldsymbol{x} = 0$). Then it uses a lookup table to tile the surface inside the cube. However, this simple algorithm can lead to cracks in the resulting mesh. Therefore, a variant called Lewiner marching cubes [82] is always used. It ensures a topologically correct result, i.e. a manifold mesh for any input data.

As an alternative to marching cubes, *raycasting* could be used for visualizing SDFs in a direct manner. This method does not produce any conversion loss except for the limiting factor of the number of pixels in the resulting image, but it is generally much slower than the marching cubes algorithm and does not output a mesh. In raycasting, one ray for each of the pixels in the resulting image is traced from a virtual camera, and the intersection of the closes object blocking the path of that ray is found, determining the resulting value for each pixel.

2.1.3 Metrics

A *metric* is a measure of a distance between two sets. Generally, metrics should satisfy the following axioms:

$$d(f,g) \geq 0 \text{ (non-negativity)} \tag{2.16a}$$

$$f = g \text{ iff } d(f,g) = 0 \text{ (identity of indiscernibles)} \tag{2.16b}$$

$$d(f,g) = d(g,f) \text{ (symmetry)} \tag{2.16c}$$

$$d(f,g) \leq d(f,h) + d(h,g) \text{ (triangle inequality)} \tag{2.16d}$$

Here, d is the distance between two real-valued functions f, g defined on the same discrete domain, and h is a third real-valued function. However, a proposed metric does not necessarily have to fulfill all these properties. If so, it is sometimes called a *semimetric*.

The two foundational distance functions are the $\mathcal{L}_1$ and $\mathcal{L}_2$ norms, which are defined as

$$\mathcal{L}_1 = d_1(f, g) = \frac{1}{T} \sum_{x=1}^{T} |f(x) - g(x)| \tag{2.17a}$$

$$\mathcal{L}_2 = d_2(f, g) = \frac{1}{T} \sqrt{\sum_{x=1}^{T} \left(f(x) - g(x)\right)^2} \tag{2.17b}$$

on the same discrete domain $1, 2, \ldots, T$ and satisfy all the properties in (2.16).

Metrics based on point-wise distances There are two permutation-invariant metric for comparing unordered sets that have been proposed in the literature [32].

The *Chamfer pseudo-distance* (CD) metric calculates the squared distance between each element in one set $S_1 \subseteq \mathbb{R}^3$ to the nearest neighbor in the other set $S_2 \subseteq \mathbb{R}^3$:

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_2^2 \tag{2.18}$$

The triangle-inequality does not hold for d_{CD}, but it is a non-negative function that is continuous and piecewise smooth. For each element, d_{CD} gives the nearest neighbor in the other set and sums up the squared distances. The search for each element is independent, and therefore this calculation is easily parallelizable. Even better acceleration of the nearest neighbor search can be used by using k-dimensional trees (binary search trees) to represent the sets, resulting in a $O(n \log n)$ complexity.

The other metric is named *Earth Mover's Distance* (EMD) – also called *Wasserstein distance* in the field of mathematics – after it was originally discovered as a measure on how to transport soil from one place to another with minimal effort. It requires the two sets $S_1, S_2 \subseteq \mathbb{R}^3$ to be of equal size $s = |S_1| = |S_2|$, and is defined as

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} ||x - \phi(x)||_2 \tag{2.19}$$

where $\phi : S_1 \to S_2$ is a bijection (a mapping that is both one-to-one and onto) which minimizes the average distance between the corresponding points. Even with modern graphics hardware, finding the exact computation of Earth Mover's distance is too expensive. Therefore, an $(1+\epsilon)$ approximation scheme given by Bertsekas et al. [8] is typically used for finding the optimal ϕ. This algorithm gives an approximation error around 1% for most typical inputs, and is easily parallelizable. However, the complexity of calculating d_{EMD} is still as high as $O(n^2)$, making it too expensive to use when n is large. Compared to Chamfer Distance, Earth Mover's distance favors distributions of points that are similarly evenly distributed as the ground truth distribution. A low Chamfer distance may be ahcieved by assigning just one element in S_2 to a cluster of points in S_1. On the other hand, to achieve a low Earth Mover's distance, each cluster of elements in S_1 requires a comparably sized cluster of elements in S_2.

Another evaluative metric is the *Jensen-Shannon Divergence* (JSD) between marginal distributions defined in the Euclidean 3D space, which can be used for comparing two point clouds. By having point-cloud data that are axis aligned, one can assume a canonical voxel grid in the ambient space and count the number of point lying within each voxel [1]. JSD is then a measure of the distance between the two empirical distributions P and Q,

$$\text{JSD}(P||Q) = \frac{1}{2} D_{\text{KL}}(P||M) + \frac{1}{2} D_{\text{KL}}(Q||M) \tag{2.20}$$

where $M = \frac{1}{2}(P+Q)$ and $D_{\mathrm{KL}}(\cdot||\cdot)$ is the Kullback-Leibler divergence, which calculates exactly how much information is lost when one distribution P is approximated by another distribution Q, and is defined as

$$D_{\mathrm{KL}}(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{2.21}$$

The metrics *mesh accuracy* and *mesh completion* compare the point-wise distances between generated points and the ground truth mesh [135]. Specifically, mesh accuracy is the minimum distance d such that 90% of generated points are within d of the ground truth mesh. Mesh completion, on the other hand, is the fraction of points sampled from the ground truth mesh that are within some distance parameter Δ, e.g. $\Delta = 0.01$ to the generated mesh. Higher values are better for both of these metrics. Since a good mesh accuracy can be achieved by generating one small portion of the ground truth and ignoring the rest, mesh accuracy does not measure how complete the generated mesh is. Therefore, the two metrics should be paired together.

Metrics based on geometric alignment Several metrics based on geometric alignment are available, including *Intersection over Union*, *cross entropy*, and *mean squared error.*

Intersection over Union (IoU), also known as the Jaccard index, is defined as the size of the intersection between two sets S_1 and S_2 divided by the size of the union of the sets:

$$\mathrm{IoU}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{|S_1| + |S_2| - |S_1 \cap S_2|}{|S_1 \cap S_2|} \tag{2.22}$$

Specifically, for voxels the IoU between a predicted 3D voxel grid and its true voxel grid is

$$\mathrm{IoU} = \frac{\sum_{ijk} \left[I(y'_{ijk} > p) * I(y_{ijk}) \right]}{\sum_{ijk} \left[I\left(I(y'_{ijk} > p) + I(y_{ijk}) \right) \right]} \tag{2.23}$$

where y'_{ijk} is the predicted value at the (i, j, k) voxel, y_{ijk} is the ground truth value at the (i, j, k) voxel, and p is the threshold for voxelization (typically set to $p = 0.5$), and $I(\cdot)$ is an indicator function. The indicator function of a subset A of a set X is defined as

$$I(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \tag{2.24}$$

The IoU value ranges between 0 and 1, with small values indicating low similarity and large values indicating high degrees of similarity.

Cross-entropy is defined as

$$\mathrm{CE} = \frac{1}{IJK} \sum_{ijk} \left[y_{ijk} \log(y'_{ijk}) + (1 - y_{ijk}) \log(1 - y'_{ijk}) \right] \tag{2.25}$$

A low cross-entropy means that the predicted value is close to the ground-truth value.

Mean squared error (MSE) is defined as

$$\mathrm{MSE} = \frac{1}{IJK} \sum_{ijk} \left(y_{ijk} - y'_{ijk} \right)^2 \tag{2.26}$$

which is always non-negative, with a perfect value being 0, i.e. MSE values close to zero indicate that the prediction is close to the ground-truth.

Metrics based on visual similarity The *light field descriptor* is inspired by the human vision system. It considers a set of rendered views of a 3D shape from various camera angles [14]. The dissimilarity D_A between two 3D models is defined as

$$D_A = \min_i \sum_{k=1}^{10} d(I_{1k}, I_{2k}), \quad i = 1, 2, \ldots, 60 \tag{2.27}$$

where i denotes different rotations between camera positions of two 3D models. The camera is positioned on the vertices of a dodecahedron surrounding each model. A dodecahedron has 20 vertices, each connected by 3 edges, which results in a total of 60 different rotations for each camera system. I_{1k} and I_{2k} are corresponding images under the i-th rotation. Finally, d denotes the dissimilarity between two images,

$$d(I_1, I_2) = \sum_i |C_{1_i} - C_{2_i}| \tag{2.28}$$

This is the $\mathcal{L}_1$ distance between the coefficients C_1 and C_2 of two images, which are the combined coefficients of the Zernike moment descriptors and Fourier descriptors of the images. There are 35 coefficients for the Zernike moment descriptor and 10 coefficients for the Fourier descriptor, totaling 45 coefficients for each image. i denotes the index of their coefficients.

2.2 Deep Learning

Deep learning is an approach to solving problems through allowing computers to learn from experience, and understand their environment through a hierarchy of concepts. By acquiring knowledge from experience, the computers can make decisions without humans formally specifying the exact rules needed. The hierarchy is organized such that simple concepts are combined to build increasingly complex representations. A graph can be drawn showing how these concepts relate to one another. It would be a deep graph with many layers, which explains the term 'deep learning'.

In general, deep learning is a specific approach in the field of *artificial intelligence* (AI) [43]. One of the main desires among computer scientists have been to create machines that are intelligent. Several efforts have been made to hard-code knowledge in a formal language, but they have deemed unsuccessful. Instead, machines seem to require a learning based approach, by extracting patterns from data. This approach is called *machine learning*. A set of features is fed into a machine learning agent, which then learns how the features correlate with various outcomes. Figure 2.6 gives a naive schematic on this distinct difference between the two programming dogmas. *Representation learning* further builds upon this by using a machine learning approach for both mapping a representation to an output and the representation itself. However, many factors influence the data that can be observed, while not affecting the outcome. Therefore, high-level features are desired. Deep learning solves this problem by building up a hierarchy of representations, gradually abstracting the lower-level concepts. The relationship between the AI disciplines is illustrated in Figure 2.7.

Recently, deep learning has seen an immense success due to several factors such as dataset sizes, model sizes, and real-world applications. Since computers have become increasingly interconnected, data is readily available from online sources. Datasets of significant sizes, such as ImageNet [30], were first released in the late 2000s, which have enabled deep learning to

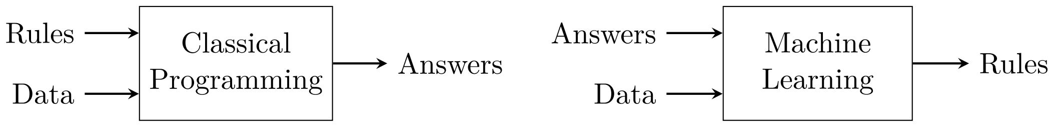


Figure 2.6: The different programming paradigms of machine learning and traditional programming. With machine learning, data as well as the answers expected from the data are given as the input, and the output from the program are the rules. These rules can then be applied to new data to produce original answers.

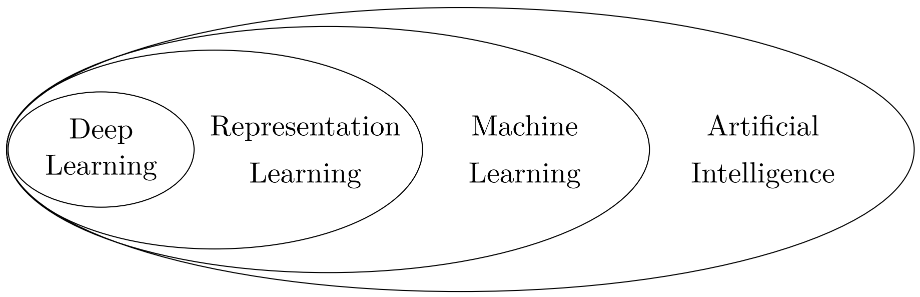


Figure 2.7: A Venn diagram illustrating the relation between artificial intelligence, machine learning, representation learning, and deep learning.

solve problems that were infeasible earlier. Moreover, faster CPUs and GPUs, in addition to better software infrastructure have allowed for the deep learning models to grow in size. A breakthrough came in 2012 when a deep learning approach won the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), the largest contest in object recognition [134], and in all consecutive years deep learning approaches has won the competition. Similar results have been presented in other fields, rendering deep learning one of the most promising tools in numerous fields.

2.2.1 Neural Networks

The word *neuron* is used to describe the computations

$$z = \sum_{k=1}^{n} x_k w_k + b = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{2.29a}$$

$$y = \phi(z) \tag{2.29b}$$

which are also illustrated in Figure 2.8.

Equation (2.29a) is a weighted sum of its inputs $\boldsymbol{x}$, i.e. the inputs $\boldsymbol{x}$ are multiplied by variables $\boldsymbol{w}$ called weights. The weights signify strengths between the input and output. For example, x_1 influences the output more than x_2 if w_1 has a higher value than w_2. The value b is the bias, which is not multiplied by a weight. With this term, the function is not constrained to the origin. The weighted sum and the bias are the linear part of the neuron. z is then passed through a nonlinear activation function ϕ to get the output y in (2.29b). This is necessary for the neuron to be able to model a non-linear problem. Example activation functions are given in Figure 2.9.

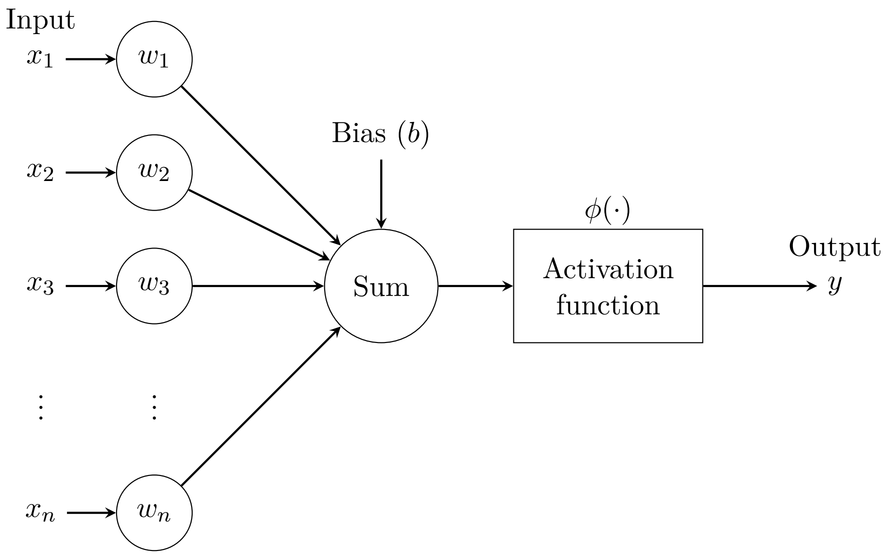


Figure 2.8: Neuron structure. The output y is defined as the sum of a bias b and the weighted sum of it inputs $\sum_{k=1}^{n} x_k w_k$ passed through an activation function ϕ.

As an example, consider a binary classification problem where y is defined as the true label ($y = 0$ or $y = 1$) and $\hat{y}$ is the predicted output. Put another way, $\hat{y}$ is the probability that $y = 1$ given inputs $\boldsymbol{w}$ and $\boldsymbol{x}$, which give the equations

$$P(y = 1|\boldsymbol{w}, \boldsymbol{x}) = \hat{y} \tag{2.30a}$$

$$P(y = 0|\boldsymbol{w}, \boldsymbol{x}) = 1 - \hat{y} \tag{2.30b}$$

This can be written more compactly as

$$p(y|\boldsymbol{w}, \boldsymbol{x}) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{2.31}$$

Further, by taking the logarithm a *loss function* can be defined as

$$L(y, \hat{y}) = -(y \log \hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{2.32}$$

The goal of the loss function is to minimize the error between the predicted and desired output and thus arrive at an optimal solution for one training sample. The logarithm operation was added to make the following derivations simpler, and it could be done since the goal is to minimize (2.31) which is equivalent to minimizing any strictly increasing function of (2.31). However, to get useful results the average of the loss over a training set that contains m independently generated training samples is needed. This is defined as the cost function $J(w, b)$, where the goal is to find w and b that minimize it:

$$\begin{aligned} J(w, b) &= \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}_i, y_i\right) \\ &= -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right] \end{aligned} \tag{2.33}$$

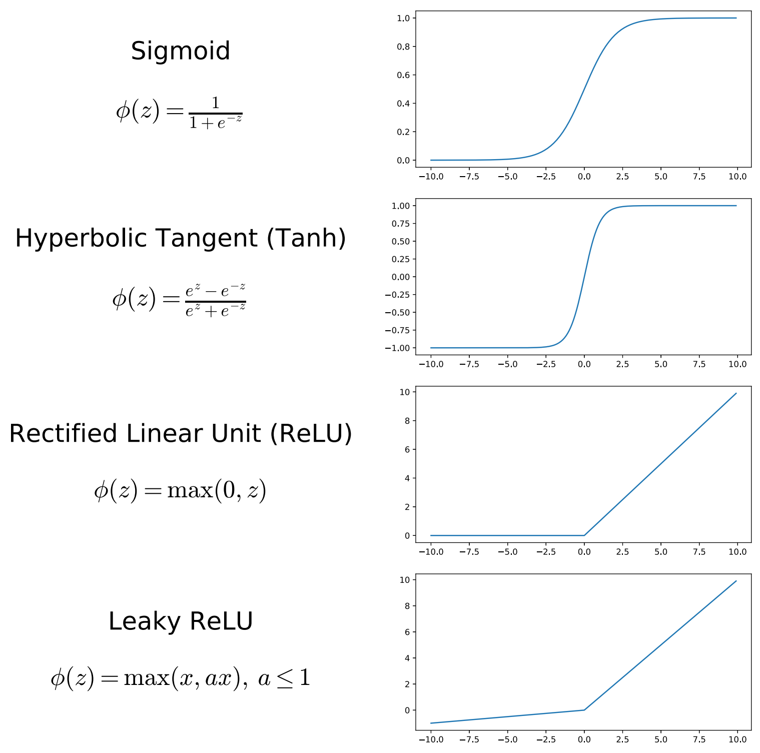


Figure 2.9: Some alternatives which can be used as activation functions. Most literature recommend using ReLU (or its variations like leaky ReLU) for all layers except the output layer, where a sigmoid or tanh are more suitable because of their symmetric property.

$J(w, b)$ is the binary cross entropy between the target and the output, and is a convex function with a single global optimum. Therefore, by moving in the direction of the steepest slope from any point on the function will iteratively give a value that is closer to the minima. This method is called *gradient descent*. In this case, the formulas are

$$w = w - \alpha \frac{\partial J(w, b)}{\partial w} \tag{2.34a}$$

$$b = b - \alpha \frac{\partial J(w, b)}{\partial b} \tag{2.34b}$$

The term α is called the *learning rate*, and is a measure on the step size in each iteration. This is an important parameter to tune, as if it is too small the model will take numerous steps before finding the minima. On the other hand, if it is too large the model will simply overshoot the minima and fail to converge.

The training process is twofold: *forward propagation* and *backward propagation*. First the input is propagated through the model as given in (2.29) to get an output $\hat{y}$. Backpropagation, however, is the process of calculating the partial derivatives from the loss function back to the inputs, and thereby updating the values of w and b:

$$\frac{\partial L(y, \hat{y})}{\partial w} = \frac{\partial L(y, \hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} \tag{2.35a}$$

$$\frac{\partial L(y,\hat{y})}{\partial b} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} \tag{2.35b}$$

One *epoch* of learning has been completed if the procedure above has been performed for the entire dataset.

Generally, a *neural network* combines several neurons together, and the same components as above – a model, a cost function, an optimizer, and a dataset – apply to all networks. These components can be replaced almost independently from the others.

Models *Feedforward neural networks* are the most important deep learning models. These models are called feedforward because information flows through some function f evaluated from $\boldsymbol{x}$ as defined in (2.29), and to the output $\boldsymbol{y}$. For instance, there might be four functions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, and $f^{(4)}$ connected in a chain $f(\boldsymbol{x}) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))))$. This gives a network structure that can be illustrated as given in Figure 2.10. In this case, $f^{(1)}$ is called the *input layer*, $f^{(2)}$ and $f^{(3)}$ are called *hidden layers*, and $f^{(4)}$ is the *output layer*.

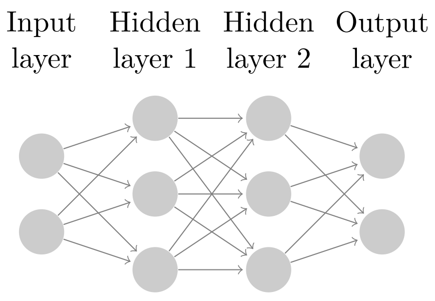


Figure 2.10: Feedforward neural network with 2 hidden layers. The nodes in this graph are simplified representations of the neuron structure as given in Figure 2.8.

Each function $f^{(i)}$, where i denotes the layer, is a combination of neurons given as

$$f^{(i)} = \phi(\boldsymbol{W}_i \boldsymbol{x}_i + \boldsymbol{b}_i) = \boldsymbol{y}_i \tag{2.36}$$

which is the generalized version of (2.29). With a total of m weights and n neurons, the weight matrix $\boldsymbol{W}_i$

$$\boldsymbol{W}_i = \begin{bmatrix} w_{i_{1,1}} & w_{i_{1,2}} & \cdots & w_{i_{1,n}} \\ w_{i_{2,1}} & w_{i_{2,2}} & \cdots & w_{i_{2,n}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i_{m,1}} & w_{i_{m,2}} & \cdots & w_{i_{m,n}} \end{bmatrix} \tag{2.37}$$

$\boldsymbol{y}_i$ is the outputs for layer i, $\boldsymbol{x}_i$ is either the input to the model or the outputs from layer $i-1$, and $\boldsymbol{b}_i$ is the biases:

$$\boldsymbol{y}_i = \begin{bmatrix} y_{i_1} \\ y_{i_2} \\ \vdots \\ y_{i_n} \end{bmatrix} \quad \boldsymbol{x}_i = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_n} \end{bmatrix} \quad \boldsymbol{b}_i = \begin{bmatrix} b_{i_1} \\ b_{i_2} \\ \vdots \\ b_{i_n} \end{bmatrix} \tag{2.38}$$

Other networks, including convolutional neural networks and autoencoders, will be presented in Section 2.2.2 and Section 2.2.3.

Cost functions A cost function is needed to give a quantitative measurement on the model's predictions, and the choice of cost function is an important part of network design. With (2.33) in the example above, the cross-entropy between the training data and the model's predictions is used. Alternatively, some statistic of the output can be predicted based on the input.

Two requirements must be satisfied by a cost function [107]. First, the cost function J must be able to be written as an average

$$J = \frac{1}{m}\sum_{i=1}^{m} C_i \tag{2.39}$$

This requirement allows the gradient to be computed for a single training example. Second, the cost function must not be dependent on any outputs except for the last layer. Otherwise, backpropagation would not be possible.

The cost function may also include additional regularization terms. *Weight decay* is one of the most common regularization terms, as it adds a criterion to minimize the weights to have a smaller norm. For $\mathcal{L}_2$ regularization, the cost function is

$$\hat{J}(\boldsymbol{w}, b) = J(\boldsymbol{w}, b) + \lambda \boldsymbol{w}^\top \boldsymbol{w} \tag{2.40}$$

where λ determines the strength of preference for small weights.

Optimizers The optimizer updates the model in response to the output of the cost function, such that the predictions are as correct as possible. Gradient descent, as presented previously, is the most straightforward optimizer. It computes the gradient of the cost function with respect to the parameters θ (includes both the weights w and the biases b) for the entire dataset, and is therefore also called *batch gradient descent*:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta) \tag{2.41}$$

For convex surfaces, batch gradient descent is guaranteed to converge to the global minima, while for non-convex surfaces it will converge to a local minimum. As the gradients for the entire dataset is needed to perform a single update, batch gradient descent can be very slow. Additionally, a dataset of substantial size will not fit in memory at once, making batch gradient descent infeasible.

To alleviate these challenges, *stochastic gradient descent* (SGD) can be used instead. It performs a parameter update for each training sample $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta, x^{(i)}, y^{(i)}) \tag{2.42}$$

As only one update is done at a time, this algorithm is much faster than gradient descent. However, that property also causes SGD to have a high variance, such that the cost function can fluctuate heavily. On the one hand, this gives SGD the opportunity of jumping to another and potentially better local minima for non-convex problems. On the other hand, SGD will keep overshooting the minima if the learning rate is not set low enough.

A third version of a gradient descent optimizer is *mini-batch gradient descent*, which combines the best properties of the two aforementioned optimizers:

$$\theta = \theta - \alpha \nabla_\theta J(\theta, x^{(i:i+m)}, y^{(i:i+m)}) \tag{2.43}$$

By performing an update for every mini-batch of m training samples, the variance of the parameter updates is reduced, which can lead to a more stable convergence. Mini-batch gradient descent is almost always used instead of batch gradient descent and SGD, and the term SGD is usually employed also when mini-batches are used. Therefore, this thesis will use the term SGD for mini-batch gradient descent, and specify the batch size where applicable. Further, the parameters $x^{(i:i+m)}$ and $y^{(i:i+m)}$ are left out of the modifications of SGD for simplicity.

SGD performs poorly when the surface of one dimension is much steeper than another, which led to the introduction of *momentum* as a method to accelerate the gradient descent in the relevant direction. Momentum does this by adding a fraction γ, usually set to around $\gamma = 0.9$, of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta J(\theta) \tag{2.44a}$$

$$\theta = \theta - v_t \tag{2.44b}$$

The momentum term v_t increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. This results in faster convergence and reduced oscillations.

Still, further improvements can be made, such as giving the optimizer a notion of the slope ahead. *Nesterov accelerated gradient* (NAG) computes $\theta - \gamma v_{t-1}$, which gives an approximation on where the parameters will be for the next time step. The gradient is then computed not with respect to the current parameters, but with respect to approximate future location of the parameters:

$$v_t = \gamma v_{t-1} \alpha \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{2.45a}$$

$$\theta = \theta - v_t \tag{2.45b}$$

Adagrad is an optimizer that additionally adapts the learning rate to the parameters. It performs smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features. This makes Adagrad well suited for sparse data. Since there is a different learning rate for each parameter θ_i, the notation becomes slightly more complex than for the previously mentioned optimizers. g_t is used to denote the gradient at time step t, and $g_{t,i}$ is then

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{2.46}$$

In the update rule for Adagrad, the learning rate α is updated for every parameter θ_i based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{2.47}$$

Here, $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element i, i is the sum of the squares with respect to θ_i up to time step t. To avoid division by zero, ϵ is added as a smoothing term and is typically around $\epsilon = 10^{-8}$. One of the main benefits with this algorithm is that manual tuning of the learning rate is not needed. On the other hand, the accumulated sum of the gradients in the denominator keeps growing during training since every term is positive. Consequently, the learning rate will eventually shrink to become too small to further update the position of the parameters.

Adam is an optimizer that seeks to resolve this flaw with Adagrad. It stores an exponentially decaying average of past gradients m_t and v_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.48a}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.48b}$$

Both m_t and v_t are initialized as vectors of 0's, and therefore have a bias towards zero. To counteract these biases, the bias-corrected estimates are first calculated as

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.49a}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.49b}$$

which are then used in the Adam update rule,

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} + \hat{m}_t \tag{2.50}$$

Generally, the suggested values for the constants are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Several other optimizers exist, but they have limited or no improved performance over the Adam optimizer.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) is a kind of network that is specifically adapted to data that has a known grid-like topology. They have been immensely successful in practical application, and especially in applications where image data is used. In place of general matrix multiplication, convolution is used in some or all layers of the network. These convolution layers can be broken down into three stages: the convolution stage, the detector stage, and the pooling stage as illustrated in Figure 2.11. The convolution stage performs several convolutions in parallel to create a set of linear activations. In the following detector stage, each linear activation is passed through a set of nonlinear activation functions, such as the ones previously presented in Figure 2.9. Finally, the output is further modified in the pooling stage before it is passed on to the next layer. What the convolution and pooling stages specifically entails will be presented in more detail in the following paragraphs.

The convolution operator is defined as

$$s(t) = \int x(a) w(t - a) da \tag{2.51}$$

where $x(t)$ is the input and $w(t)$ is the *kernel*, if the terminology for convolutional neural networks is used. $s(t)$ is then called a *feature map*. Typically, the convolution operator is denoted as

$$s(t) = (x * w)(t) \tag{2.52}$$

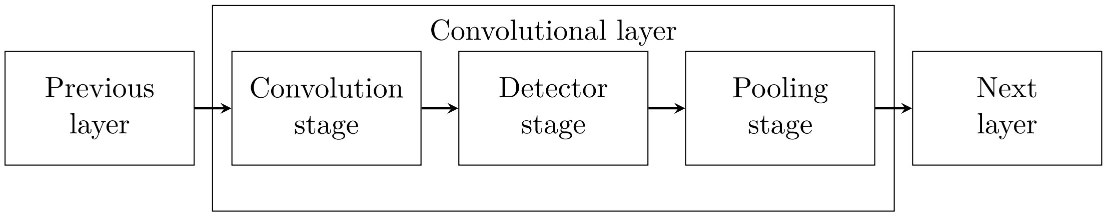


Figure 2.11: The stages of a convolutional layer in a neural network. Freely adapted from the book Deep Learning [43].

In the discrete domain, the integral reduces to a sum,

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.53}$$

which for M dimensions is

$$s(n_1, n_2, \dots, n_M) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \cdots \sum_{k_M=-\infty}^{\infty} x(k_1, k_2, \dots, k_M)w(n_1-k_1, n_2-k_2, \dots, n_M-k_M) \tag{2.54}$$

For a two-dimensional image (i.e. a grayscale image), the convolution operator should be applied over two axes at a time. Using a two-dimensional kernel K on an image I, the convolution operator becomes

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.55}$$

$$= (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(i,j) \tag{2.56}$$

where the third equality follows from convolution being a commutative operator.

The kernel is typically a 3×3, 5×5 or 7×7 matrix, which is likely much smaller than the image. The two-dimensional convolution with a 2×2 kernel is illustrated in Figure 2.12. This kernel is slided over the input image, producing the feature map. How far the kernel moves when traversing the image is called the *stride*. The default for this value is usually 1, but a stride of 2 could be used if for instance downsampling of the input is desirable. Without any *padding*, which defines how the border of the sample is handled, the feature map will be $k-1$ pixels smaller than the input if a $k \times k$ kernel is used. Instead, if the dimensionality of the input should be maintained in the output, the input can be surrounded (or padded) with e.g. zeros with a size of $(k-1)/2$ if a stride of 1 is chosen. In general, the formula for calculating the outpus size for any convolution layer is

$$o = \frac{w - k + 2p}{s} + 1 \tag{2.57}$$

where o is the output height/length, w is the input height/length, k is the kernel size, p is the padding, and s is the stride. Note that the input size has to be considered when selecting the stride and padding, as the convolution operator is not defined if the kernel does not cover input units for all of its elements.

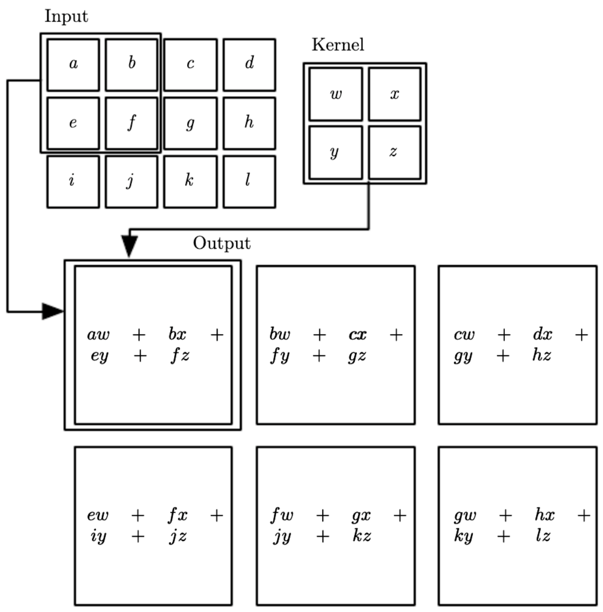


Figure 2.12: A schematic illustration of 2D convolution with a 2×2 kernel on a 3×4 image grid. In this figure the stride is 1 and the output is restricted to positions where the kernel lies entirely within the image. Courtesy of Deep Learning [43].

In most networks, the images used are actually three-dimensional, and thus these convolutions are performed in 3D. This is because an image is represented as a 3D *tensor* (i.e. multidimensional array) with dimensions of height, width and depth, where depth corresponds to the color channels RGB. A kernel, by design, covers the entire depth of its input and should therefore be three-dimensional. However, multiple kernels can be used in one layer which would output just as many disjoint feature maps stacked along the depth dimension, as illustrated in Figure 2.13. In the next layer the kernel depth dimension must then match the number of kernels used in the previous layer.

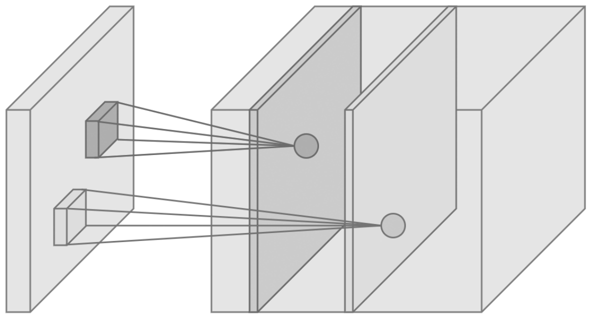


Figure 2.13: Illustration of how two feature maps can be stacked along the depth dimension.

In the pooling stage, a pooling operator replaces the output of the net at a certain location with a summary statistic of the nearby outputs. This leads to a reduction in the number of parameters, which could be necessary if the images are large. It will therefore also reduce the training time, and counteract overfitting of the network. More importantly, pooling helps make the representation nearly invariant to small translations of the input. If it is more important to know whether a feature exists in an image, rather than the location of that feature, the invariance property could be useful. Common pooling functions include max pooling, average pooling, and sum pooling. Max pooling is illustrated in Figure 2.14.

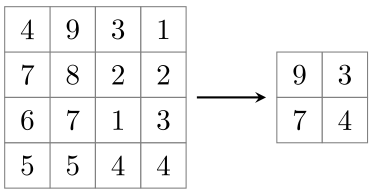


Figure 2.14: Max pooling outputs the maximum value within a rectangular neighborhood. This example uses max pooling with a 2×2 filter with a stride of 2.

In the stages presented above, the CNN utilizes three ideas: *sparse interactions*, *parameter sharing*, and *equivariant representations*.

Sparse interactions refer to a property of CNNs; they store fewer parameters than their fully connected feed-forward network counterparts. It is accomplished by making the kernel smaller than the input. If there are m inputs and n outputs, forward propagation using a fully-connected layer will have $O(m \times n)$ runtime. On the other hand, if the number of connections each output may have is limited to k, the number of parameters is reduced to $k \times n$ which results in a runtime of $O(k \times n)$. Good performance can in many applications still be obtained with k several orders of magnitude smaller than m.

Second, using the same parameter for more than one function in a model is referred to as parameter sharing. Specifically, the convolution operator uses parameter sharing to learn only one set of parameters, instead of separate set for every location. This has no impact on the runtime of forward propagation, but it reduces the memory requirements of the layer to k parameters.

Third, equivariance refers to a property for a function where the output changes the same way as the input. In mathematical terms, two functions $f(x)$ and $g(x)$ are equivariant if $f(g(x)) = g(f(x))$. For a convolutional layer, the parameter sharing causes the layer to have the additional property of being equivariant to translation. As an example, consider a function f which maps one image function I describing a grayscale image to another image function I', such that $I'(x,y) = I(x-1,y)$. This operation shifts every pixel of I one unit to the right. The same result will be obtained irrespectively of the order of convolution and this transformation to I is applied. However, convolutional layers are not inherently invariant to changes such as scaling or rotation.

2.2.3 Representation Learning Techniques

The goal of *representation learning*, or *feature learning*, is to automatically discover a set of features of the data that make it easier to extract useful information when building classifiers or other predictors [7]. A good representation is both one that captures the posterior distribution of the underlying explanatory factors for the observed input, and one that is useful as input to a supervised predictor. In deep learning, the representations are formed by composition of multiple non-linear transformations of the input data. One should note that representation learning is here referring to the characteristics of the transformed input, and not the model that is causal to it.

The representation of the data has a vital role in effective deep learning, and representation learning provides a data-driven approach for achieving the best data transformations. Generally, the performance of a model is critically dependent on the representations it learns to output, which in turn is dependent on the model and on what is fed as input. In the context of deep learning, this helps explain the reasoning for stacking a linear layer on top of one or more complex blocks with many non-linear layers of different kinds. These complex blocks transform the input to a rich representation which then only require a simple linear layer to do the desired task. Without the transformation performed by the complex blocks it would not be possible to extract key abstract features.

There are several priors that can impact the accuracy of a representation, and Bengio et al. [7] cover these comprehensively. In the following f denotes a function that maps input x to output representation y. Models may implement one or more of these priors to learn output representations suited to a specific task.

- Smoothness: It is assumed that small changes in x leads to small changes in y: $x_1 \approx x_2$ implies $f(x_1) \approx f(x_2)$. Some models require training examples to map out all the wrinkles in the training function, or otherwise those features will not be represented by the model as it is interpolating between neighboring samples for generalization.

- Multiple explanatory features: A model could potentially generalize without requiring as many examples as there are variations in the underlying function f. This compactness can only be achieved if the features are reused across examples that are not necessarily in a local neighborhood, which is unlike the smoothness prior mentioned above. Deep learning models can learn distributed representations of size $O(N)$ to distinguish $O(2^k)$ input regions where $k = N$ in a densely distributed representation and $k < N$ in a sparsely distributed representation. In distributed representations, k features are used to represent a concept. Moreover, each feature participates in the representation of multiple concepts. Sparse distributed representations, specifically, add a restriction where only $k < N$ features can be changed at any time. Distributed representations allows for exponential combinations to represent the input.

- Depth - a hierarchical organization of explanatory factors: This hierarchy is beneficial for mainly two reasons. First, deep learning models add increasing levels of abstraction as they learn functions that transform input to output using a composition of non-linear functions stacked in layers. Second, the depth allow for feature reuse across layers, which adds numerous paths in the computational graph within the network.

- Semi-supervised learning: Representations that are useful for $P(X)$ tend to be useful when learning $P(Y|X)$ (where X is the input and Y is the target to predict), since a subset of the factors explaining X's distribution explain much of Y, given X.

- Shared features across tasks. This refers to sharing of learned representation across tasks, where $P(Y|X, \text{task})$ are explained by facors that are shared with other tasks.

- Manifold representations: Probability mass concentrates near regions that have a much smaller dimensionality than the original space where the data resides. Autoencoders explicitly exploit this hypothesis to learn lower dimensional representations of high dimensional data. For example, a 28×28 black and white image (such as the images in MNIST [80], one of the simplest datasets in machine learning) has 784 degrees of freedom which yield 2^{784} possible images, but most of them would not be naturally occurring images. The hypothesis is that small variations in the naturally occurring images, such as rotations, are mapped to corresponding changes in the learned representation.

- Natural clustering: Different values of categorical variables such as object classes tend to be associated with separate manifolds, where each manifold is composed of learned representation of an object class. That is, $P(X|Y = i)$ for different i do not have much overlap.

- Temporal and spatial coherence: Consecutive or spatially nearby observations tend to be associated with the same value of relevant categorical concepts, or result in a small move on the surface of the high-density manifold. Even though different features change at different spatial and temporal scales, the values of the categorical variables of interest tend to change slowly. Consequently, this prior can be used as a mechanism to force the representatiosn to change slowly.

- Sparsity: For any given observation x, only a small fraction of the possible features are relevant. In terms of representation, this could be represented by features that are often zero or by the fact that extracted features are insensitive to variations of x.

- Simplicity of factor dependencies: With a high-level representation, the features that may relate to each other through simple, typically linear dependencies. This can be seen several laws of physics. For deep learning models, this explains why several state-of-the art network architectures which have one or more linear layers near the output perform well.

The approaches that are used for the purpose of learning data distributions of the training set to generate new data points are also called *generative models*. In brief, some of the most important generative models are the following: *Generative Adversarial Networks* (GANs) aim to achieve an equilibrium between a generator and a discriminator, *Variational Autoencoders* (VAEs) focus on maximizing a variational lower bound of the data log-likelihood, and *Variational Autodecoders* (VADs) which are similar to VAEs but allow for inference to be performed with gradient descent.

One thing these models have in common is that they are optimizing a *latent vector*, which is a set of variables that are not directly observed but rather inferred through a mathematical model. The neural network maps the input to this typically low-dimensional latent representation. However, the exact way the latent vector is generated is unique to each network architecture.

Generative model learning can roughly be grouped into four classes [111]: variational learning, wake-sleep learning, joint-stochastic-approximation learning, and adversarial learning. Only variational learning (of which VAEs and VADs belong) and adversarial learning (of which GANs belong) will be presented here.

2.2.3.1 Variational Learning

Variational learning is mainly applied to prescribed models, which are models that provide an explicit parametric specification of the distribution of the observed random variable x, and where the likelihood function or target model density $p_\theta(x)$ is specified with the parameter θ. Further, the auxiliary density introduced in training is denoted by $q_\phi(\cdot)$ with parameter ϕ.

Specifically, variational learning uses the variational lower bound of the marginal log-likelihood as the single objective function to optimize the target models and auxiliary model [111].

Variational Autoencoders (VAEs) The most important class of models in variational learning is VAEs, which aims to learn the marginal likelihood of the data in a generative process [71]. This section explains the concepts behind VAEs.

A prescribed latent variable model could be generally defined as

$$p_\theta(x,h) := p_\theta(h)p_\theta(x|h) \tag{2.58}$$

where h are hidden variables, which are also called the latent code. It is difficult to directly evaluate and maximize the marginal log-likelihood $\log p_\theta(x)$. An auxiliary inference model $q_\phi(h|x)$ with parameters ϕ is introduced in the variational inference approach, to be an approximation to the exact posterior $p_\theta(h|x)$. It is often called *variational distribution* in the context of variational inference [174]. Technically, the implementation of $q_\phi(h|x)$ is similar to implementing $p_\theta(x|h)$ as a prescribed model.

$$\begin{aligned} \log p_\theta(x) &= E_{q_\theta(h|x)} \log\left(\frac{p_\theta(x,h)}{q_\theta(h|x)}\right) + D_{\mathrm{KL}}(q_\theta(h|x)||p_\theta(h|x)) \\ &\geq E_{q_\theta(h|x)} \log\left(\frac{p_\theta(x,h)}{q_\theta(h|x)}\right) =: \mathcal{L}(x;\theta,\phi) \end{aligned} \tag{2.59}$$

where $D_{\mathrm{KL}}(\cdot||\cdot)$ is the Kullback-Leibler divergence introduced in (2.21). $\mathcal{L}(x;\theta,\phi)$ is called the variational lower bound, which could be rewritten as

$$\mathcal{L}(x;\theta,\phi) = E_{q_\phi(h|x)}[\log p_\theta(x,h)] - D_{\mathrm{KL}}(q_\theta(h|x)||p_\theta(h|x)) \tag{2.60a}$$

$$= E_{q_\phi(h,x)}[\log p_\theta(x,h)] + H(q_\phi(h|x)) \tag{2.60b}$$

The two terms in (2.60a) are the expected negative reconstruction error and the KL divergence between the approximate posterior and the prior which is a regularizer, respectively. Presented this way, it becomes clear why $p_\theta(x|h)$ and $q_\phi(h|x)$ are referred to as *encoder* and *decoder*. Further, the combination of the two terms in (2.60a) hints at performing *autoencoding*, i.e. automatically finding encodings for input x.

Variational learning aims to maximize (2.60) over the entire training data:

$$\max_{\theta,\phi} \mathcal{L}(\theta,\phi) = \max_{\theta,\phi} E_{\tilde{p}(x)q_\phi(h|x)} \log\left(\frac{p_\theta(x,h)}{q_\theta(h|x)}\right) \tag{2.61}$$

In words, the objective itself is aimed to be maximized by maximizing a lower bound to the true maximum-likelihood objective in maximum likelihood learning, which is defined by maximizing the data log-likelihood:

$$\max_\theta = \sum_{k=1}^{n} \log p_\theta(x_k) \tag{2.62}$$

The above optimization problem in (2.61) can be solved by finding the root of the following system if the zeros of the gradients of the objective with respect to (θ, ϕ) are set to zero:

$$\begin{cases} E_{\tilde{p}(x)q_\theta(h|x)}[\nabla_\theta \log p_\theta(x,h)] = 0 \\ E_{\tilde{p}(x)q_\theta(h|x)}\left[\log\left(\frac{p_\theta(x,h)}{q_\phi(h|x)}\right) \times \nabla_\phi \log q_\phi(h|x)\right] = 0 \end{cases} \tag{2.63}$$

A simple schematic illustrating the encoder-decoder architecture above is given in Figure 2.15.

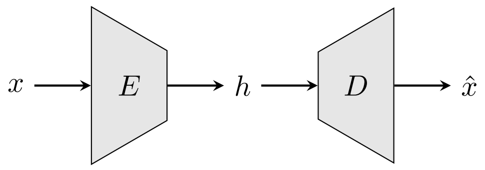


Figure 2.15: Autoencoder. A neural network E (encoder) learns a mapping from the input data x to its latent representation h. D (decoder) can then reproduce a reconstruction $\hat{x}$ based on h.

Variational Autodecoders (VADs) In contrast to VAEs, a VAD does not use any encoder during the learning process [171], as illustrated in Figure 2.16. In brief, it is in the inference step that the major distinction between VAEs and VADs become the most apparent.

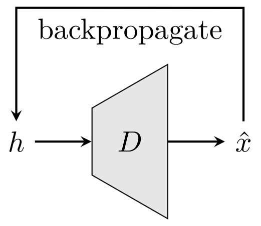


Figure 2.16: Autodecoder. A neural network D (decoder) directly accepts a latent vector as an input, which is optimized through backpropagation.

VADs perform inference by a multi-step procedure. The input $x_i \in X$ is given to a probabilistic decoder $\mathcal{F}(h;\theta)$ with parameters θ^* to give the posterior distribution,

$$p(h|x_i, \Lambda_i) = \mathcal{N}(\mathcal{F}(z;\theta^*); x_i, \Lambda_i) \tag{2.64}$$

where Λ_i is the covariance defined as a diagonal positive semi-definite matrix with random variables α_i on its main diagonal whenever $\alpha_i \neq 0$. Given the sample h_t given at time t, the possible steps at time step $t+1$ are chosen to follow a normal distribution centered around z_t. This leads to a higher chance of visiting points closer to z_t next, which makes the sampling seem like a random walk when they are taken from the posterior $p(h|x_i, \Lambda_i)$. The following acceptance criteria is used to determine which samples are chosen:

$$c = \min\left(\frac{p(z_{t+1}|x_i, \Lambda_i)}{p(z_t|x_i, \Lambda_i)}, 1\right) \tag{2.65}$$

where c is the probability of 1 from a Bernoulli distribution. If the result of the test is 1, the step is accepted. Otherwise, if the result is 0, the step is rejected and $z_{t+1} = z_t$. In the end, a set of points $H = h_1, \ldots, h_T$ is produced, where T is the number of samples drawn from a posterior $p(h|x_i, \Lambda_i)$. Note that this procedure to generate the latent representation is much slower than using an encoder as in VAEs, but potentially give a superior performance in dealing with partial data.

For an approximate posterior $q(h|x_i)$ from a set of known simpler distributions where sampling can be done more easily, the following variational lower bound is maximized:

$$\arg\max_{\phi} E_{q_\phi(z|x_i)}[\log p_{\theta^*}(h, x_i) - \log q_\phi(h|x_i)] \tag{2.66}$$

This equation can be maximized using the expectation maximization algorithm to learn ϕ in such a way that samples from $p(h|x_i, \Lambda_i)$ are reasonably generated by the approximate posterior. The inference output is $q_\phi(h|x_i)$.

The training procedure of VAD models can be summarized in two steps that are iteratively taken until no further improvement in the variational lower bound in (2.60) is achieved. In the first step, inference as defined in the previous paragraph is performed. This maximizes the lower bound with respect to ϕ. Then, in the next step, the lower bound is maximized with respect to θ. In the end, a trained decoder $\mathcal{F}(h; \theta^*)$ with parameters θ^* is obtained.

2.2.3.2 Adversarial learning

Adversarial learning uses a technique where an auxiliary model is introduced to act like a discriminator and optimized simultaneously as an implicit model is learned. The implicit model uses a latent variable ϵ, and transform it using a deterministic function $G_\theta(\epsilon)$ to define

$$\begin{cases} x = G_\theta(\epsilon) \\ \epsilon \sim p(\epsilon) \end{cases} \tag{2.67}$$

Quite often, a feed-forward network is used to represent $G_\theta(\epsilon)$, which is called a *generator* in this context. ϵ is also referred to as an input noise variable, and assumed to obey a simple priori $p(\epsilon)$ such as the standard normal $\mathcal{N}[0, 1]$ or the uniform distribution. A distribution $p_\theta(x)$ is thus implicitly defined by the transformation.

In the following, the adversarial learning methods Generative Adversarial Networks and Variational Divergence Minimization learning are presented.

Generative Adversarial Networks (GANs): GAN training is formulated as playing a two-player minimax game where the goal is to find $p_\theta(x)$ that best describes the true distribution given a set of independent and identically distributed samples $\mathcal{D} = \{x_1, \ldots, x_n\}$ from an unknown distribution $p_0(x)$ [42]:

$$\min_{\theta} \max_{\psi} \mathcal{F}_{\mathrm{GAN}}(\theta, \psi) := E_{x \sim p_0(x)}[\log D_\psi(x)] + E_{\epsilon \sim p(\epsilon)}[\log(1 - D_\psi(G_\theta(\epsilon)))] \tag{2.68}$$

Here, $D_\phi(x)$ represents the *discriminator*, which calculates the probability that x comes from the data $p_0(x)$ rather than $p_\theta(x)$. $D_\phi(x)$ is trained to maximize the probability of assigning the correct labels to both training examples and generated examples, while $G_\theta(\epsilon)$ is simultaneously trained to minimize the probability of the correct labeling of generated samples. In other words,

$G_\theta(\epsilon)$ tries to trick the discriminator into believing that $p_\theta(x)$ actually comes from the true distribution $p_0(x)$. For any fixed generator $G_\theta(\epsilon)$ the optimal discriminator is

$$D_{\psi^*}(x) = \frac{p_0(x)}{p_0(x) + p_\theta(x)} \tag{2.69}$$

When the discriminator is optimal, the objective in (2.68) becomes

$$\begin{aligned}\mathcal{F}_{\text{GAN}}(\theta, \psi^*) &= E_{x\sim p_0(x)}\left[\log\left(\frac{p_0(x)}{p_0(x) + p_\theta(x)}\right)\right] + E_{\epsilon\sim p(\epsilon)}\left[\log\left(\frac{p_\theta(x)}{p_0(x) + p_\theta(x)}\right)\right] \\ &= -\log(4) + 2\cdot \text{JSD}(p_0(x)||p_\theta(x))\end{aligned} \tag{2.70}$$

where JSD$(\cdot)$ is the Jensen-Shannon divergence defined in (2.20).

After the training is finished, hopefully a saddle-point ($p_\theta(x) = p_0(x)$) of (2.68) is found. A schematic of a GAN is given in Figure 2.17.

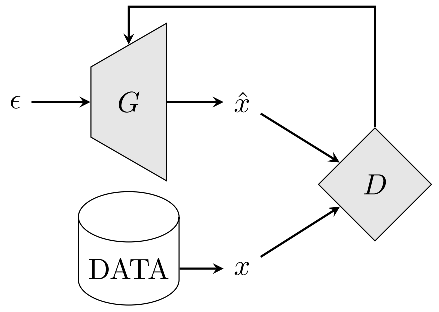


Figure 2.17: Generative Adversarial Network. A neural network G (generator) synthesizes samples that look indistinguishable from real data based on a distribution ϵ. D (discriminator) tries to distinguish the real samples x from the synthesized samples $\hat{x}$.

Variational Divergence Minimization Nowozin et al. [109] showed that the learning objective of GANs are actually variational bounds on a divergence between two distributions. This enables extending the GAN objective to general f-divergences, which is defined as follows:

$$D_f(p_0||p_\theta) := \int_{\mathcal{X}} p_\theta(x) f\left(\frac{p_0(x)}{p_\theta(x)}\right) dx \tag{2.71}$$

f is a generator function $f : \mathcal{R}_+ \rightarrow \mathcal{R}$ which is convex, lower-semicontinuous and satisfying $f(1) = 0$, which leads to $D_f(p_0||p_0) = 0$ for all distributions p_0. Various choices of f are available, including the Kullback-Leibler divergence. This generalization of GANs is called variational divergence minimization.

To see how one can learn by variational divergence minimization, the lower bound for the f-divergence has to be derived. First, it is important to note that every convex, lower-semicontinuous function f has a convex conjugate function $f^\dagger$. $f^{\dagger\dagger} = f$, which means that the pair $(f, f^\dagger)$ is dual. Furthermore,

$$\begin{cases} f^\dagger(t) = \sup\limits_{u \in \mathrm{dom}_f} \{ut - f(u)\} \\ f(u) = \sup\limits_{t \in \mathrm{dom}_{f^\dagger}} \{tu - f^\dagger(t)\} \end{cases} \tag{2.72}$$

A lower bound on the f-divergence can be obtained by using the variational representation of f:

$$\begin{aligned} D_f(p_0 || p_\theta) &= \int_{\mathcal{X}} p_\theta(x) \sup_{t \in \mathrm{dom}_{f^\dagger}} \left\{ t \frac{p_0(x)}{p_\theta(x)} - f^\dagger(t) \right\} dx \\ &\geq \sup_{T \in \mathcal{T}} \left(\int_{\mathcal{X}} p_0(x) T(x) dx - \int_{\mathcal{X}} p_\theta(x) f^\dagger(T(x)) dx \right) \\ &= \sup_{T \in \mathcal{T}} (E_{p_0(x)}[T(x)] - E_{p_\theta(x)}[f^\dagger(T(x))]) \end{aligned} \tag{2.73}$$

where T is an arbitrary class of functions $T : \mathcal{X} \rightarrow \mathrm{dom}_{f^\dagger}$. Two reasons can be stated for why the above derivation yields a lower bound. First, because of Jensen's inequality (which states that if a function f is convex and x is a random variable then $E[f(x)] \geq f(E[x])$) when swapping the integration and supremum operations. Second, the class of functions T may contain only a subset of all possible functions. The bound is tight for

$$T^*(x) = f' \left(\frac{p_0(x)}{p_\theta(x)} \right) \tag{2.74}$$

when taking the variation of the lower bound of (2.73) with respect to T Here, f' denotes the first-order derivative of f.

The variational lower bound in (2.73) can be used on the f-divergence $D_f(p_0||p_\theta)$ to estimate the implicit generative model $p_\theta(x)$ given the true distribution $p_0(x)$:

$$\min_\theta \max_\psi \mathcal{F}_f(\theta, \psi) := E_{p_0(x)}[T_\psi(x)] - E_{p_\theta(x)}[f^\dagger(T_\psi(x))] \tag{2.75}$$

Here, $T_\psi(x)$ is a parametrized vector of the variational function T. To find the saddle points of $\mathcal{F}(\theta, \psi)$, it is necessary to set the gradients of it to zero with respect to (θ, ψ) and then find the root of the resulting system.

CHAPTER 3

Related Work

As the goal of this work is to create a 3D vision pipeline which fills in the occluded parts of the objects of interest in a scene or a workspace, ideas from several fields will have to be combined. Therefore, for a holistic understanding, this chapter includes a literature review on as many as four fields related to RGB-D imaging and 3D vision; denoising and filtering (Section 3.1), segmentation and object detection (Section 3.2), point cloud registration (Section 3.3), and shape completion (Section 3.4).Extra attention is given to the current state-of-the art, in addition to some methods which could potentially be used in a novel way.

Additionally, a section on available 3D datasets is included (Section 3.5). Compiling and annotating datasets is an intensive process which requires numerous man-hours. Since deep learning models in particular require thousands of objects in the datasets used for training, the design of such models are enabled by the availability of public datasets.

3.1 Depth Image Denoising and Enhancement

RGB-D cameras and related sensor technology have recently become more widely available and affordable, and thus enabled new applications in computer vision in the 3D domain. Unfortunately, for most methods of depth acquisition, the resolution and quality of the depth component are significantly lower than that of the RGB component. Additionally, the depth cameras still suffer from heavy sensor noises. This results in the need for post-processing depth images acquired from RGB-D cameras, by using methods in denoising and upsampling, and other enhancements.

In the color image domain, the most widely accepted framework employs the combination of transform domain techniques and nonlocal similarity characteristics of natural images [76]. Based on this framework, many competitive methods model the correlation of RGB channels with pre-defined or adaptively learned transforms. In the depth image domain, however, different approaches are needed.

Since the RGB-D camera captures images that are of higher resolution and lower noise than the depth sensors, it is a reasonable approach to use the color information to improve the depth data. Some of these methods use a heuristic assumption between color and depth [31, 78, 90, 112, 167, 180], but typically produce depth maps with artifacts and which are not metrically accurate. RGB images can also be used to improve depth quality by investigating the depth transport process [6, 49, 169, 175]. For instance, shape-from-shading techniques can be used to refine the geometry of the structures in the depth map via inverse rendering optimization. However, such optimization is immensely challenging with traditional optimization methods. Additionally, artifacts may appear if the geometry is not precisely estimated.

Another approach is to use multiple depth frames as input, and fuse them together to improve the depth quality. Fusion methods combine feature extraction and matching for succeeding RGB-D scans of a scene [26, 48, 105], which in addition to reducing noise also performs 3D reconstruction of the scene. By integrating signed distance functions, some methods are able to effectively reduce noise [15, 108]. The limitation with these approaches is that they require multiple frames to improve the accuracy of the original depth map.

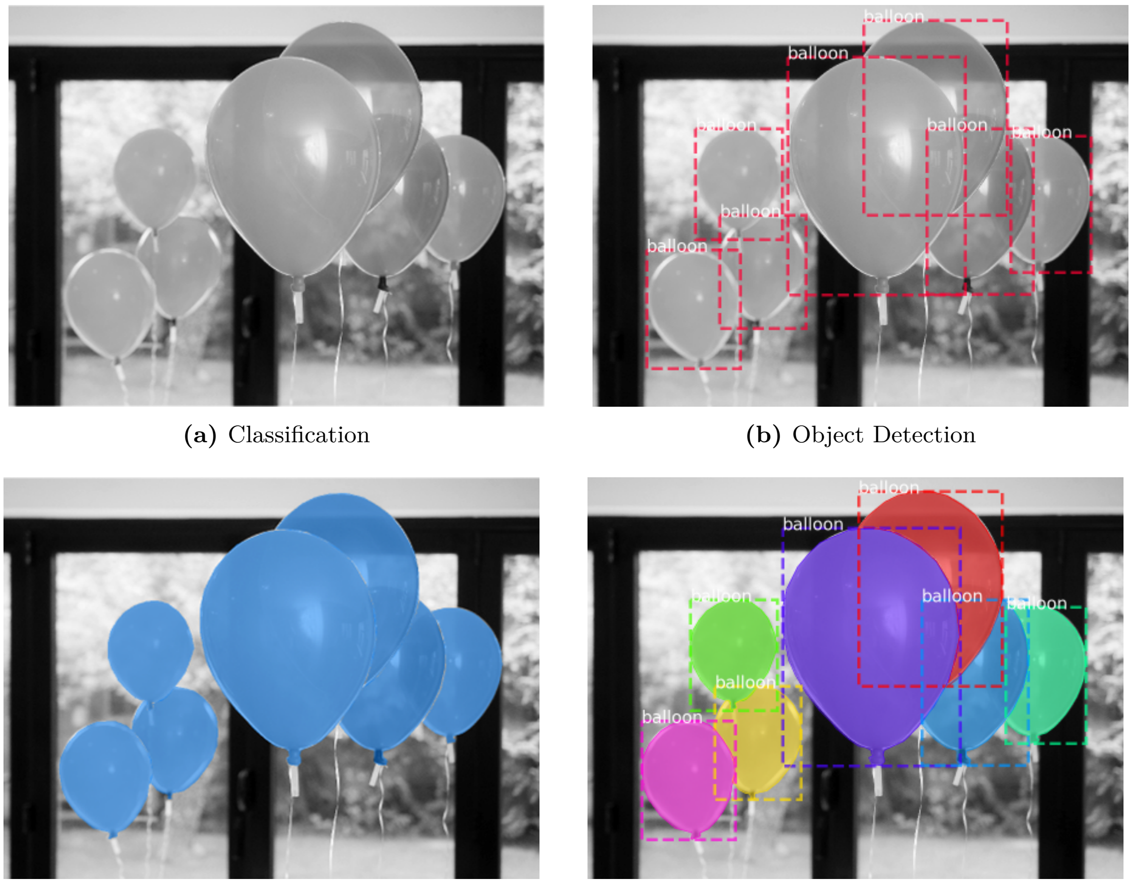
Data-driven methods may be used to improve the solutions of the previously mentioned approaches. For instance, Barron et al. [5] use a statistical method where priors derived from images are used to find reflectance, shape and illumination in the inverse rendering problem. Methods like these do not require any learning, but they typically require segmentation of foreground objects and depend heavily on the quality of such segmentation. On the other hand, neural networks such as CNNs [161, 162, 85] are able to produce impressive results that are generalizable to a wide range of scenarios. These methods also upsample the resolution of the depth images to comparable sizes to those of the RGB images. Hybrid methods [128, 141] have also been proposed, where subsequent optimization stages are applied to the CNN output to produce sharper results. Voinov et al. [153] demonstrate that using a visual appearance-based loss, used with e.g. a CNN, yields significantly improved 3D structures with less artifacts than the previously mentioned methods.

Another option is to denoise the resulting point clouds after conversion from the depth maps. In this domain, deep learning models are by far producing the best results [44, 57, 81, 124]. There does not seem, however, to be any advantage of denoising the point clouds rather than the depth maps.

3.2 Semantic Segmentation and Object Detection

Perhaps the most important task when working with images is to identify what the image represents. To gain a complete image understanding, effort should not only be put on classifying images, but also to estimate which objects are represented and their locations in the image. These tasks have different names depending on the exact formulation of the problem. *Object detection* refers to the task of detecting all objects in an image and localize them with a bounding box. The bounding box could also be labeled to add information on the object inside. Segmentation, however, refers to a partition of an image into several coherent parts. Specifically, *semantic segmentation* both partitions the image and classify each part into one set of predefined classes, which in practice means that the goal is to classify each pixel in the image. *Instance segmentation* is an even more ambitious task, where each instance of objects belonging to the same class should be identified. The difference between all these tasks is illustrated in Figure 3.1. It should be noted that these tasks can also be applied to other data than images. For instance, bounding boxes can be drawn around three-dimensional data which could also be segmented into its constituent parts.

Recent surveys and reviews [38, 148, 178] conclude that deep learning methods is the current state-of-the art, and that they have many promising directions for future work. These methods are able to extract high-level features and are thus able to address the issues existing with classical methods. Most notably, popular classical methods such as gray level segmentation, which hard-codes rules (gray level intensities) a region must satisfy for it to be assigned a particular label. It might work somewhat well for depth images, where objects in the foreground might have distinctly different depth than their surroundings. However, it will fail for cases such as a tabletop which can span a relatively high length in the depth dimension, and thus



(a) Classification **(b)** Object Detection

(c) Semantic Segmentation **(d)** Instance Segmentation

Figure 3.1: The difference between classification (identify which object is in the scene), object detection (find tight bounding boxes around each instance of an object in a scene), semantic segmentation (identify the pixel locations of each object class in a scene), and instance segmentation (identify the pixel locations of each *instance* of each class in a scene).

be falsely be segmented into multiple partitions. Deep learning methods, on the other hand, are producing results of impressive quality in cases like these.

In this work, RGB-D images will be the raw data which should be segmented in order to extract the objects of interest. Since each object has to be individually identified, instance segmentation methods in particular will be of special interest. The additional data in the depth image should intuitively give a vastly improved information foundation compared to images with only color data. However, the state-of-the art approaches which use only color images as the training data could possibly be transferred to corresponding approaches where RGB-D data is used instead, as the only difference is an added channel to the neural network input (i.e. the depth).

Current instance segmentation methods can be roughly categorized into two classes; detection based methods and segmentation based methods. Detection-based methods [16, 27, 55, 86, 119] use detectors from object-detection networks such as Faster R-CNN [126] or

R-FCN [28] to get the region of each instance, and then predict the mask for each region. A common limitation with these methods is that mask quality is only measured by the classification scores, which only distinguishes the semantic categories of proposals, and do not control the quality of the instance mask. High classification scores can be achieved for the localized bounding boxes, but the corresponding masks may be inaccurate. The other class, segmentation based methods [12, 34, 51, 64, 72, 87, 106], predict the category labels of each pixel first and then group them together to form the region masks. However, a drawback with both of the aforementioned classes is that they do not consider the alignment between mask score and mask quality. A mask hypothesis could be ranked with low priority if it has a low mask score, even though its IoU against ground truth is high.

Huang et al. [59] addressed these issues by introducing a detection score correction to the detection-based method Mask R-CNN [55], which resulted in a network that is the current state-of-the art. Conceptually, as shown in Figure 3.2 the network is composed of Mask R-CNN with a MaskIoU head, which predicts the intersection over union between the input mask (the instance feature and the predicted mask) and the ground truth mask. Mask R-CNN further consists of two stages. The first stage is called the Region Proposal Network (RPN), which regardless of object categories propose candidate object bounding boxes. The second stage is referred to as the R-CNN stage, which for each proposal extracts features using RoIAlign as more precisely explained in Figure 3.3. It also performs classification, bounding box regression and mask prediction.

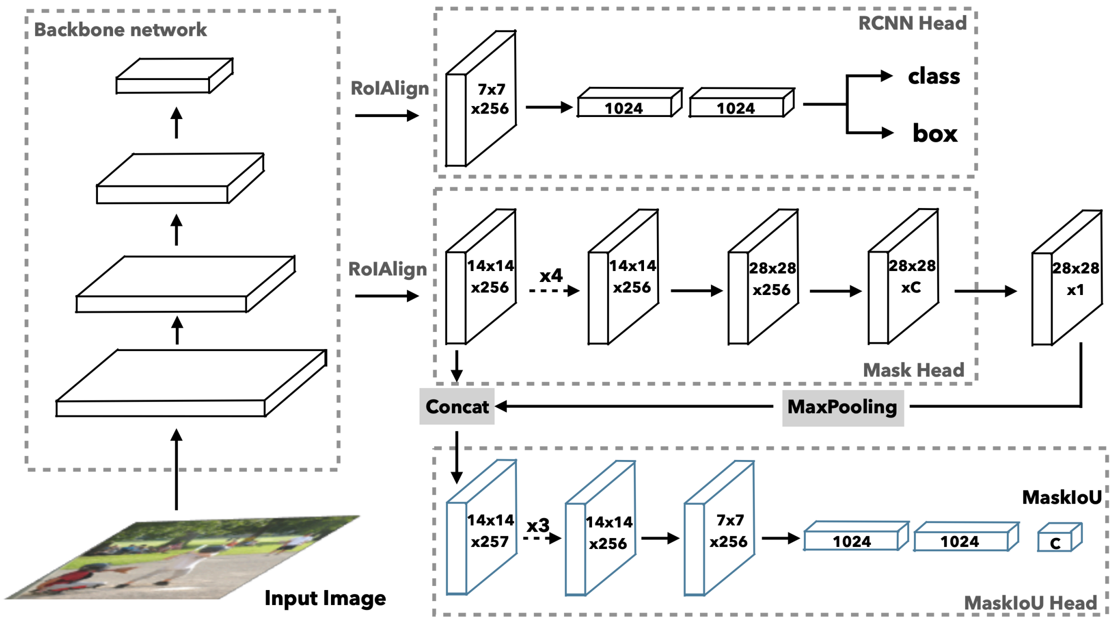


Figure 3.2: Mask Scoring R-CNN architecture. A backbone network is using RPN and RoIAlign on the input image to generate RoI features. Both the RCNN Head and Mask Head are components extracted from Mask R-CNN. The MaskIoU Head is used to predict the MaskIoU. The layers with numbers $k \times k \times d$ are convolutional layers, while the layers with only one number are fully connected. Courtesy of Huang et al. [59].

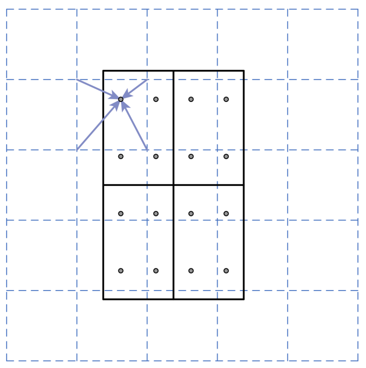


Figure 3.3: RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. In the figure, the dashed grid is represented as a feature map, while RoI is here represented by solid lines with 2×2 bins, and the dots are the four sampling points in each bin. Courtesy of He et al. [55].

The scoring of predicted mask is defined as $s_{\text{mask}} = s_{\text{cls}} \cdot s_{\text{iou}}$, where s_{cls} is the score of the classification proposal and s_{iou} is the score for regressing the MaskIoU. This decomposition of s_{mask} directly follows from two criteria: It should be equal to the pixel-level IoU between the predicted mask and its matched ground truth mask. Since a mask only belongs to one class, it should also be positive for ground truth category, and zero for other classes.

Previous work in RGB-D image segmentation combined two networks for RGB and data. A common approach with these methods is to pretrain a model on a large RGB dataset and then fine tune it on the target RGB and depth dataset. This has been a necessary approach as until recently RGB-D datasets have been limited in terms of size (see Table 3.2 in Section 4.1.2). However, the approach has several limitations. The low-level filters learned from the RGB data are not used for the depth data, and thus the network cannot properly exploit depth-specific patterns. Further, RGB and depth features are only combined a high levels (i.e. the last layers).

A second option on including depth data in segmentation networks is to encode them as RGB data, such as using the HHA approach by Gupta et al. [46]. With this encoding, the depth image is represented with three channels at each pixel: horizontal disparity, height above the ground, and the angle the pixel's local surface normal makes with the inferred gravity direction. Hazirbas et al. [52] showed that RGB-HHA outperforms other input forms such as RGB-D, which have similar accuracy as only RGB input. However, they also showed that the HHA encoding does not hold more information than the depth image, but hinted that splitting up the RGB-D data into RGB and depth images would fully utilize the depth information.

More recent RGB-D image segmentation networks have successfully trained encoder-decoder-like networks on combined RGB and depth data [52, 63, 136]. The encoder in these networks consists of multiple branches that simultaneously extract features from RGB and depth images. These features are then fused together after each layer. The network RedNet by Jiang et al. [63] in particular achieve state-of-the art on the SUN RGB-D benchmark dataset by also applying depth fusion structure on the downsample part of the network, and apply skip-connections to bypass the fused information to the decoder for full-resolution semantic prediction. A main limitation with RedNet is that it needs to be modified to do instance segmentation, which is a non-trivial task.

In the 3D domain, segmentation and object detection on point clouds has shown promising results [118, 121, 139, 163]. As point clouds could be directly inferred from depth images, such methods could potentially be of interest in this work. However, they discard any color information in the data and are thus more relevant for LiDAR scans in terms of real-world applications.

3.3 Point Cloud Registration

Point cloud registration is a key problem in computer vision which involves finding a rigid transformation from one point cloud to another so that they align. Use cases include scanning a scene, where it is important to transform the point clouds to the same reference coordinates, or aligning a partial scan of an object to that of a reference point cloud acquired from a dataset, which is one of the core tasks in this work.

Iterative Closest Point (ICP) [9] is the most popular registration algorithm. This algorithm iteratively estimates the point corresponcence in the following manner: For each point in the first point cloud, the closest point in the second point clouds is searched for. Then, solve for an affine transformation which minimizes the distance between these point pairs. The first point cloud is then updated with this affine transform, and the process is repeated until the change in mean-square error falls below a preset threshold. The are multiple variations of ICP [133], which handles these steps slightly differently to address issues regarding e.g. outliers in the point clouds. For instance, the search for closest points could be done using k-d tree nearest-neighbor search, and the transforms could be solved using singular value decomposition (SVD). However, all variations of ICP methods have some essential drawbacks: They are sensitive to initialization, in that they are prone to a local minima due to non-convexity. Further, the closest point correspondences are explicitly estimated, which causes the complexity not to scale well with increasing point cloud sizes. It would also be useful to incorporate these methods into a deep learning context, but this is nontrivial due to issues with differentiability.

When a global solution is desired, i.e. when the point clouds are not roughly aligned prior to registration, Go-ICP [166] could be used. It is a branch and bound-based optimization approach to search the motion space $SE(3)$ to obtain a globally optimal pose. Other methods which attempt to find the globally optimal estimates are based on Riemannian optimization [132], semi-definite programming [58, 92], and mixed integer programming [62]. All these methods are severely limited in terms of real-time usefulness due to their large computation time.

Some approaches estimate interest points to help with registration [40, 41]. These methods might increase the computational speed, but are not generalizable to all applications.

Recent efforts have applied graph neural networks to address difficulties in the classical ICP pipeline. For instance, Aoki et al. [2] use a PointNet-architecture to learn point cloud representations [123], while Wang et al [155] use DGCNN [156] for this purpose. The latter method, named Deep Closest Point (DCP), constitutes the current state-of-the art on point cloud registration and will form the basis for the registration part of this work. If extra accuracy is desired, ICP can be used to the output from DCP to refine the result.

A major drawback with all the aforementioned approaches is that they only align point clouds that are of the same size. That is, they are able to correctly rotate and translate the point clouds to match, but not scale them appropriately. Most applications include point clouds of different scales, which means that this is a very important issue. However, this research in this area still seems to be in the early stages. Some recent work [17, 18, 89] presents a few approaches to the issue, but they give a limited accuracy on point clouds which are of widely different scales, or when they differ in terms of completeness.

3.4 3D Shape Completion

3D shape completion is a sub-field in computer vision and computer graphics which is concerned with the problem of estimating a complete 3D structure of an object given sparse or partial input observations of an object. Such partial observations could be depth images from one viewing angle or point clouds from LiDAR scans. This problem corresponds to image-inpainting in 2D computer vision. In three dimensions, the problem is even more daunting as in theory an infinite number of 3D models can be associated with a partial observation. For reference, a related problem is 3D reconstruction, which is the problem of recovering object shape from either a single image or multiple images. Zollhöfer et al. [181] recently published a review on state of the art on 3D reconstruction with RGB-D cameras, but their main focus is on reconstructing complete scenes, and not individual objects. Therefore, the most relevant papers to 3D shape completion are missing in that review.

In general, existing methods for 3D shape completion can roughly be categorized into geometry-based, alignment-based and learning-based approaches. These approaches will be discussed in detail in the following sections, with a focus on the learning-based approaches as the most recent literature and state-of-the art belong to this category.

3.4.1 Geometry-based shape completion

Geometry-based shape completion use geometric properties from the partial input without any external data to output a complete shape.

In geometry processing, shape completion has been a long-studied problem where hole infilling in particular has been one of the main areas of focus. Several traditional algorithms aim to fit in local surface primitives such as planes or quadrics. For instance, minimum area triangulation [4] finds the triangulation of a mesh that has the smallest measure in terms of metrics such as area or edge length. Liepa's hole-filling algorithm [88] instead looks for a triangulation with the smallest dihedral angle (the angle between two intersecting planes). A volumetric approach is introduced in the robust repair [65] algorithm, which determines grid edges of a regular voxel grid that intersect the mesh, and then uses marching cubes [91] to reconstruct a complete polygon surface. These algorithms finish in up to $O(n^3)$ time, making them too slow for real-time applications.

Alternatively, a continuous energy minimization approach such as Laplacian smoothing [104, 142, 177] could be applied. This is equivalent to formulating the problem as an optimization problem. These methods represent a mesh as a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, with vertices $\boldsymbol{V}$ and edges $\boldsymbol{E}$, where $\boldsymbol{V} = [\boldsymbol{v}_1^\top, \boldsymbol{v}_2^\top, \ldots, \boldsymbol{v}_n^\top]^\top$, $\boldsymbol{v}_i = [v_{ix}, v_{iy}, v_{iz}]^\top \in \mathbb{R}^3$ is the original geometry, and $\boldsymbol{V'}$ is the displaced geometry. Furthermore, δ_i is the Laplacian of $\boldsymbol{v}_i$, the result of applying the discrete Laplace operator to $\boldsymbol{v}_i$, i.e.

$$\delta_i = \sum_{\{i,j\} \in \boldsymbol{E}} w_{ij}(\boldsymbol{v}_j - \boldsymbol{v}_i) = \left[\sum_{\{i,j\} \in \boldsymbol{E}} w_{ij} \boldsymbol{v}_j \right] - \boldsymbol{v}_i \tag{3.1}$$

where $\sum_{\{i,j\}\in\boldsymbol{E}} w_{ij} = 1$, and weights

$$w_{ij} = \frac{\omega_{ij}}{\sum_{\{i,j\}\in\boldsymbol{E}} \omega_{ij}} \tag{3.2}$$

where ω_{ij} can be chosen to be e.g. $\omega_{ij} = 1$. The Laplacian for the entire mesh is obtained by using the $n \times n$ Laplacian matrix $\boldsymbol{L}$ with elements

$$\boldsymbol{L}_{ij} = \begin{cases} -1 & i = j \\ w_{ij} & (i,j) \in \boldsymbol{E} \\ 0 & otherwise \end{cases} \tag{3.3}$$

This Laplacian matrix is used in the algorithm least square meshes by Sorkine et al. [142], who demonstrate how a mesh can be reconstructed from only connectivity information. The positions x, y and z are solved for separately by minimizing the quadratic energy

$$\left\|\boldsymbol{L}_u\boldsymbol{V}'_d\right\|^2 + \sum_{s\in\boldsymbol{C}} w_s^2\left|v'_{sd} - v_{sd}\right|^2 \tag{3.4}$$

Here, $\boldsymbol{C} \subset \boldsymbol{V}$ of m geometrically constrained vertices (anchors) and $\boldsymbol{V}'_d = [v'_{1d}, v'_{2d}, \ldots, v'_{nd}]^\top, d \in \{x, y, z\}$. The scalars v_{sd} are the stored anchor positions and w_s^2 are weighting factors. The overdetermined linear system

$$\underbrace{\begin{bmatrix} \boldsymbol{L}_u \\ \hline \boldsymbol{I_{m\times m}}|\boldsymbol{0} \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{V}'_d = \underbrace{\begin{bmatrix} \boldsymbol{0} \\ \hline \boldsymbol{V}_{(1\ldots m)d} \end{bmatrix}}_{\boldsymbol{b}} \tag{3.5}$$

where $\boldsymbol{I_{m\times m}}$ is the identity matrix and $\boldsymbol{0}$ is the zero-vector is then solved with least squares as $\boldsymbol{V}'_d = (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{b}$.

Laplacian mesh smoothing [104] generalizes the previous method by modifying it to include all vertices as both Laplacian and positional constraints.

A third option is Poisson surface reconstruction [68, 69], which is a technique for creating watertight surfaces from oriented point samples, such as point clouds acquired from 3D range scanners. The watertight mesh can be obtained by transforming the oriented point samples into a continuous vector field in 3D, for then to find a scalar function whose gradients best match the vector field, and finally extract the appropriate *isosurface*, which is a surface that represents points of a constant value. In this context, the isosurface implies the surface of a 3D object.

The original Poisson surface reconstruction approach reconstruct the surface of the model M by solving for an indicator function χ of the shape:

$$\chi_M = \begin{cases} 1 & \text{if } p \in M \\ 0 & \text{if } p \notin M \end{cases} \tag{3.6}$$

This indicator function is defined as 1 inside the model, and 0 outside. The problem is then rephrased into how the indicator function can be constructed. Given a point cloud with oriented points, there is a relationship between the normal of the field and the gradient of the indicator function $\nabla\chi_M$. By representing the points by a vector field $\overrightarrow{V}$, and finding the function χ whose gradient best approximates $\overrightarrow{V}$, the problem reduces to inverting the gradient operator, i.e. $\min_{\chi}\left\|\nabla\chi - \overrightarrow{V}\right\|$. If the divergence operator is applied, this problem can further be transformed into a Poisson problem:

$$\nabla \cdot (\nabla \chi) = \nabla \cdot \overrightarrow{V} \Leftrightarrow \nabla\chi = \nabla \cdot \overrightarrow{V} \tag{3.7}$$

The entire procedure is illustrated in Figure 3.4

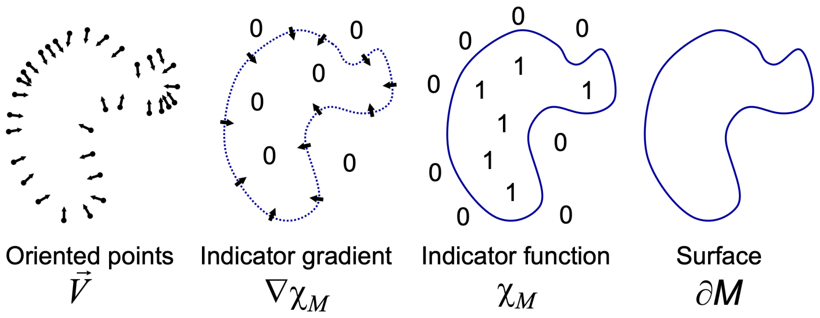


Figure 3.4: Illustration of Poisson reconstruction in 2D. Courtesy of Kazhdan et al. [68].

Symmetry-driven methods [101, 102, 115, 120, 140, 144, 149] constitute another subcategory of geometry-based shape completion, where the approach is to identify symmetry axes and repeating regular structures in the partial input in order to copy parts from observed regions to unobserved regions. For instance, Zabrodsky et al. [56] and Podolak et al. [120] propose the *planar reflective symmetry transform* (PRST) to encode a continuous notion of symmetry of an object about any any reflective plane in 3D. Given a reflective plane γ, the PRST for a function f is defined as:

$$\mathrm{PRST}^2(f,\gamma) := 1 - \frac{d(f,\gamma(f))^2}{\|f\|^2} \tag{3.8}$$

where the function $d(f,\gamma(f))$ is defined as

$$d(f,\gamma(f)) := \min_{g}\|f-g\|^2 \tag{3.9}$$

which is the distance of the function f to the closest function g that is symmetric with respect to the transform defined by γ. Hence, $\gamma(g) = g$. Equation (3.9) can be simplified to

$$d(f,\gamma(f)) = \|f - (f+\gamma(f))/2\| \tag{3.10}$$

$$= \frac{\|f-\gamma(f)\|}{2} \tag{3.11}$$

since the closest function is the average of f and its reflection γf, and thus the equation for PRST reduces to

$$\mathrm{PRST}^2(f,\gamma) = \frac{1 + f\cdot\gamma(f)}{2} \tag{3.12}$$

where f is assumed to be normalized, i.e. $\|f\| = 1$.

PRST is a general-purpose transform with several potential applications such as alignment, matching, segmentation, and viewpoint selection. In the context of shape completion, alignment using PRST is the most relevant. Three *principal symmetry axes* (PSA), which the normals of the orthogonal set of planes with maximal symmetry, are found in a sequential manner. The first principal symmetry axis is found by finding the plane with maximal symmetry, and the second axis is found by searching for maximal symmetry on the planes that are perpendicular to the first. The final axis is found in the same way. With these axes, the *center of symmetry* (COS) is also defined as the intersection of those three planes. With multiple partial scans of an object, each partial scan can be aligned based on its PSA and COS, providing a unified model. However, for individual scans, PRST provides an automatic alignment of the object into a canonical coordinate frame, which could be of use for other shape completion methods. Additionally, the same approach could be of use in the preparation of aligned 3D model datasets.

To summarize, multiple caveats apply to the methods presented in this section. They mainly only fill in smaller holes, and thus require surrounding geometry around the part that is to be reconstructed. Therefore, they are more applicable to completing human-designed meshes that are not watertight or contain undesired internal structures than partial observations or incomplete data from the real world, as they are generally too sparse for such approaches to give any satisfying completions.

3.4.2 Alignment-based shape completion

Alignment-based approaches complete shapes by matching the input with template models from a large database.

An idea is to fit models from a shape database to the partial observation from a range scan. The matched models could either be complete objects [47, 84, 103, 116, 137], or object parts that are then assembled to obtain the complete shape [66, 70, 94, 138, 144].

For better generalizability, models in the dataset could be deformed to match the partial observation [11, 37, 45, 83, 130, 131]. Particularly for models such as humans and animals, approaches like these would be beneficial. Such models might pose in numerous ways, but the core geometry constituting the models are still the same. Therefore, representing e.g. humans with a single model, the completion of a partial observation amounts to identifying the joint angles (i.e. elbows, knees, etc.) and characteristics such as height to deform the base model appropriately.

Alternatively, geometric primitives such as planes and quadrics (ellipsoids, paraboloids, hyperboloids, etc.) could be used in place of a shape database.

If an exact match of the object is in the database, alignment-based approaches might give the best completion, but otherwise these methods fall short. Therefore, in an industrial application such as an assembly line where the geometry of objects are known at millimeter precision, shape completion with an alignment-based approach could be the method of choice. On the other hand, since these methods do not easily generalize to previously unseen shapes, they are of limited use in open environments. Additionally, for the methods that deform or assemble parts of objects, the completion procedure requires expensive optimization. This would in turn make them too slow for online, i.e. real-time, applications. Finally, the methods are sensitive to noise. This could be addressed by filtering the partial observation first, but this adds extra complexity.

3.4.3 Learning-based shape completion

Learning-based approaches complete shapes with a parameterized model, such as a deep learning network, which directly maps the partial input to a complete shape. This property of these approaches consequently enable fast inference and better generalization. In the literature, most approaches adapt an autoencoder approach such as variational autoencoders. In the following, the literature review is therefore organized according to the data representation of 3D models, as that seems to be the main concept to differentiate between the various methods. Still, there are some variations in the architectures for each data representation, and the main novel ideas will be highlighted.

Voxel Representations: Early works introduced learning-based models to 3D shape completion using voxels as the chosen data representation. Wu et al. [160] proposed a CNN which complete models from depth images, where they generalized the convolution operator to three dimensions in order to convolve volumes. Further, it was shown that networks such as GANs also can be utilized in the 3D domain [158], although the training is heavily unstable. A third paper formulated a recurrent neural network approach to the problem [20], but this approach seems to be more rare in the literature.

However, these approaches were limited to very small grid sizes, such as 32^3. This creates a very blocky model, which can only complete simple man-made objects such as tables and chairs somewhat accurately. The reason behind this limitation is explained by the cubic increase in the voxel grid, causing the memory requirements to be very high. More recent papers try to address these caveats by designing more complex architectures [159, 157, 164, 165, 176]. Yang et al. [165], for instance, adds linear upsampling at the output from the discriminator in their GAN-architecture. Although these networks support resolutions up to 256^3, it is only made possible by having shallow architectures or small batch sizes, which leads to either poor shape completions or slow training.

Alternatively, the grids could be represented in tree-structures such as octrees [50, 127, 146]. This alleviates the compute and memory limitations of regular voxel methods, and can support resolutions up to 512^3 [146]. Unfortunately, these methods are difficult to implement, and they require multiple passes over the input to generate the final 3D model. Therefore, they could also become quite slow for inference, rendering them unusable for online applications. In the end, octrees only allow for slightly better resolution compared to voxels, and therefore only partly solves the memory issues.

While still keeping voxel grids as the data representation, sub-voxel precision could be introduced by predicting *truncated signed distance fields* (TSDF) [25, 129, 143, 173] where each point in a 3D grid stores the truncated signed distance to the closest 3D surface point. Since these methods store distance functions in addition to the voxel grids, networks that work with this kind of data representation have to learn additional parameters making the learning process harder. The TSDFs are also confined to a voxel grid, and thus the resolution is still limited by the underlying grid structure, while the memory usage is even higher than regular voxels.

Recent research indicate that these limitations with voxel-based approaches can not be solved by designing better network architectures, as the 3D domain introduce challenges that are not present in the two dimensional domain of images on which these methods draw inspiration. The main problem with voxels is that for high resolutions, a substantial fraction of the data structure does not represent any part of the 3D object, as it is inside the model.

For most applications, only the surface of the model is of interest. As the memory limitations of the voxel grids hinders the resolution to be set high, finer details can not be preserved. The only advantage with a voxel-based learning method seems to be that it can be used to fairly directly adapt novel methods presented in 2D computer vision to the 3D domain.

Mesh Representations: Meshes have been considered as the data representation, but suffer from being overly complex in a deep learning perspective. In the computer graphics community, however, meshes have always been the main representation. There are a few key differences between the two fields that explains this discrepancy. For computer graphics, describing geometry with vertices, edges, and faces allow for simple descriptions of objects. A flat surface such as a wall can be coded as a single triangle, while objects that are in focus can be devoted more triangles. On the other hand, a neural network will have a higher difficulty in understanding such optimal encodings of objects. This insight helps explain why existing mesh-based deep learning efforts are only able to generate shapes with simple topology [154], require a reference template from the same object class [67, 75, 125], or cannot guarantee closed surfaces [150]. They are also prone to generating self-intersecting meshes, which would require advanced post-processing to be of use.

Other works introduce novel convolution and pooling operations for meshes [152], of which the recently released work by Feng et al. [35] is of special interest. They claim to solve the complexity and irregularity problem of meshes, by introducing new descriptors for meshes and a novel mesh convolution operator. Currently, their presented network MeshNet only solves tasks such as classification. It will be interesting to see their mesh convolution task applied to the task of shape completion, and how it compares to the state-of-the art in that area.

Point Representations: Learning on point clouds was pioneered by the PointNet architecture [123, 122], which was first applied to the 3D reconstruction problem by Achlioptas et al. [1] The main concept they introduced was achieving permutation invariance by applying 1x1 convolution followed by a global pooling operation, which might sound somewhat peculiar. Intuitively, convolution cannot happen on a single 1x1 cell. For layers with only one channel, 1x1 convolution would simply be multiplication. However, if there are multiple channels, two effects can be applied with this operation. Non-linearities are introduced to the network due to the activation functions included in the convolution. Additionally, the depth of the input volume can be either increased or decreased by choosing the appropriate number of filters in the 1x1 convolution. It turns out that this is enough for a network to learn 3D representations, as otherwise fairly standard autoencoder architectures are used.

Several more recent approaches build upon the PointNet architecture for shape completion [33, 168, 172], but the Point Completion Network by Yuan et al. [170] seems to be the current state-of-the art of the point-based approaches. It combines two networks into one: an autoencoder that generates a coarse output with a resolution of 1024 points, and a folding network which learns how to deform 4x4 point patches around the points from the coarse output, effectively upsampling the point cloud 16 times to a fine-detailed output. The folding network is simply a feed forward network that deforms the 4x4 patches into a smooth 2D manifold in 3D space. The idea to combine several networks have been used in several other approaches, including the voxel-based network proposed by Yang et al. [165] mentioned earlier. It seems like this is an effective way of guiding or helping the network to learn more easily in the training phase. A common problem in deep networks with many layers, which is a necessity to be able to represent complex functions, is the *vanishing gradient problem.* For each layer, the gradient during backpropagation becomes smaller for the first layers, which makes it more difficult for their weights to update during training. Therefore, by introducing checkpoints such as comparing the first network to a simpler ground-truth and the second to the complete shape, each network can be designed to be smaller. Phrased this way, the idea can be perceived similar to the concept of *skip-connections* [53], where neurons from some of the first layers in a network are directly connected to some of the layers closer to the output.

Since point clouds as a 3D representation is sparse and lightweight, and match the output from typical sensors such as LiDAR or depth cameras, it would seem to be the ideal data representation for applying learning-based methods. However, point clouds do not describe topology, i.e. they do not explicitly encode the inside and outside of objects. Converting them to other data representations such as meshes requires non-trivial post-processing steps, which would also add to the total inference time.

Surface Representations: Interestingly, four papers were independently published in december 2018 or january 2019, which all argued for using signed distance functions as the data representation for 3D shape completion [19, 96, 98, 113]. Prior to these papers, there were no existing literature on using this data representation for learning-based approaches. Although these methods have resemblance to the TSDF methods mentioned as part of the voxel-based representations, the distance functions are not restricted to a grid. Therefore, the memory usage of these methods is quite low. In fact, Park et al. explains that their models only use 7.4 MB of memory to represent entire classes of shapes [113], which is less than half of a single 512^3 voxel model.

Park et al. also introduces the concept of *auto-decoders* to the 3D domain, a method which seems to have several advantages compared to other representation learning techniques such as GANs or VAEs. While VAEs perturb the bottleneck features with Gaussian noise to achieve smooth and complete latent spaces, auto-decoders do not use any encoder. Instead they optimize the latent codes and the decoder weights simultaneously through back-propagation during training. The decoder weights are then fixed for inference, while the optimal latent vector is searched for to match the partial observation. The complete method can be explained as follows.

A set of K point samples $\boldsymbol{x}_j$ and their signed distance values s_j are prepared from a dataset of N shapes represented with signed distance function $\mathrm{SDF}^{i\,N}_{i=1}$:

$$X_i = \{(\boldsymbol{x}_j, s_j) : s_j = \mathrm{SDF}^i(\boldsymbol{x}_j)\} \tag{3.13}$$

Each latent code $\boldsymbol{z}_i$ is paired with training shape X_i, which gives the posterior probability

$$p_\theta(\boldsymbol{z}_i|X_i) = p(\boldsymbol{z}_i) \prod_{(\boldsymbol{x}_j,\boldsymbol{s}_j)\in X_i} p_\theta(\boldsymbol{s}_j|z_i;\boldsymbol{x}_j) \tag{3.14}$$

where the prior distribution over codes $p(\boldsymbol{z}_i)$ is assumed to be a zero-mean multivariate-Gaussian with a speherical covariance of $\sigma^2 I$, and θ parameterizes the SDF likelihood expressed by a deep feed-forward network $f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j)$, which is assumed to take the form

$$p_\theta(\boldsymbol{s}_j|z_i;\boldsymbol{x}_j) = \exp(-\mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j)) \tag{3.15}$$

$\mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j)$ is a loss function which could be chosen to be for instance the $\mathcal{L}_1$ or $\mathcal{L}_2$ norm. During training, the joint log posterior over all training shapes are maximimized with respect to the individial shape codes $\{z_i\}_{i=1}^N$ and the network parameters θ:

$$\underset{\theta,\{\boldsymbol{z}\}_{i=1}^N}{\arg\min} \sum_{i=1}^{N} \left(\sum_{j=1}^{K} \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2} \|\boldsymbol{z}_i\|_2^2 \right) \tag{3.16}$$

Finally, for inference a shape code $\boldsymbol{z}_i$ for shape X_i can be estimated via *Maximum-a-Posterior* (MAP) estimation as

$$\hat{\boldsymbol{z}} = \underset{x}{\arg\min} \sum_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X} \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2} \|\boldsymbol{z}\|_2^2 \tag{3.17}$$

In this work, an approach based on signed distance functions is chosen due to its apparent advantages. Networks where signed distance functions are the chosen data representation seem to be able to represent fine details in the 3D models, unlike any of the other voxel-, mesh-, or point-based networks. The signed distance functions' continuous and implicit description of the surface of an object with the notion of inside or outside is ideal for neural networks, as they effectively define decision boundaries, i.e. the isosurfaces. In fact, using this data representation, the problem of representing shapes resembles that of regression problems. For instance, the training data could be in the form of points $[x, y, z, d]$ sampled from meshes, where x, y, and z denote the position and d denotes the distance to the closest part of the mesh (negative if inside, positive otherwise). By sampling points closely around the surface of the mesh, the fine details of the object will then be preserved. On the downside, the isosurface encoded by the signed distance function has to be acquired through either ray-casting or approximated with the marching cubes algorithm. This is necessary to include the completed shape in the acquired partial observation of a scene. However, isosurface extraction should still be fast enough to allow for online applications of methods where signed distance functions are used.

Additionally, the recently introduced auto-decoder will provide the starting point on how the shape completion network will be designed, as it has some benefits that are particularly useful in this domain. The auto-decoder can handle any form of partial observations, since the latent code can be found via a MAP estimation. This removes the need to prepare training data of partial shapes, which is necessary for all autoencoder architectures. Additionally, the training time is reduced compared to VAEs, since only a decoder is used. However, as the goal is to create an online application, the duration of the inference stage is desired to be as low as possible. Therefore, the time to search for a latent code that best matches a partial observation has to be optimized. Consideration has to be given to these potential issues in order to keep the inference time low for this method.

3.5 Datasets

3D datasets is a necessity in bringing the experimentation of real-world applications to a simulated environment. Recent advances in robotics and computer vision required huge datasets, and the trend seems to be that research in these fields tend to be increasingly data-driven. However, compiling such datasets is a labor-intensive process. Researchers have used several techniques such as synthesising 3D models using 3D design software or scanning real world 3D objects from multiple views with RGB-D cameras. Some of these curated datasets have been released publicly to simplify and lower the threshold for other researchers to work with 3D data. A non-extensive overview of synthetic and real-world datasets is presented in Table 3.1 and Table 3.2.

Dataset	Thousands of Objects	Year	Comments
Synthetic Datasets			
IKEA	0.2	2013	
PASCAL3D+	36	2014	
ModelNet	128	2015	
ShapeNet	3000	2016	Only a subset, ShapeNetCore (with 51 000 objects), is publicly available
ObjectNet3D	44	2016	
Thingi10K	10	2016	
PartNet	3000	2018	
ABC	1000	2019	
Real World Datasets			
BigBird	0.1	2014	
YCB	0.6	2015	Physical objects in the dataset can be requested
3D Scan	10	2016	
T-LESS	0.03	2017	
RBO	0.02	2018	Has articulation that allows it to be animated kinematically in physics engines

Table 3.1: Synthetic and real world datasets of 3D models.

The tables 3.1 and 3.2 show that there are significant differences in the sizes between the datasets. Real world datasets, in particular, are several orders of magnitude smaller than the synthetic datasets. However, there are also several similarities. Household objects and scenes containing such objects are the main classes in the majority of the datasets. These datasets also have huge class imbalances, where objects such as chairs and tables have a relatively higher number of instances.

Many challenges are present in the curation of a synthetic dataset. Even if the objects are publicly available from an online repository, there are challenges that have to be addressed. Objects from such sources often require manual filtering, including removing duplicate vertices and normals. Some models might have missing textures, or the wrapping of the textures is not as expected. Moreover, a significant percentage of the models are unfinished, and it is hard to identify them using an automatic process. The creators of the 3D objects could also have forgotten to provide information about scale or position the model in a canonical reference frame. Additionally, for objects to be used in a physics engine, a collision shape is required. Lastly, skeletonization is required for meshes that have dense point clouds.

Dataset	Thousands of Frames	Thousands of Layouts	Year	Comments
Synthetic Datasets				
PBR Princeton	500	45	2016	
SUNCG	130	45	2017	
Facebook House3D	-	45	2017	Sourced from the SUNCG dataset
HoME	-	45	2017	
SceneNet	5000	0.057	2017	
The RobotriX	8000	0.016	2018	Generated using UnrealROX
Real World Datasets				
NYU Depth Dataset	1.5	0.464	2012	
SUN RGB-D	10	-	2017	
ScanNet	2500	1.5	2017	
Matterport3D	200	0.09	2017	Includes 10,800 panoramic views

Table 3.2: Synthetic and real world datasets of 3D scenes.

The curation of real world datasets has been made possible by the recent advances in multi-view stereo methods and online real-time 3D reconstructions, and have therefore been introduced to the community more recently. Working with real world objects is even more labour intensive than synthetic objects, which provides additional limitations on scalability. As the data gathering is completed by scanning the objects, it can be difficult to get watertight 3D models with crisp textures. Several tries or cumbersome methods might be necessary to keep the textures from becoming blurry. Real world scanning is also a question about practicalities, as scanning rigs require multiple cameras or moving a single camera in a pre-determined path, where only slight perturbations could result in an unsuccessful scan.

The ideal object dataset is a synthetic dataset with a large variety in the types of objects included. It should be a uniform distribution between all kinds of classes to avoid any bias towards a particular set, as in the ABC dataset[74]. Due to the extra challenges in acquiring real world datasets, without the gain of any advantages, writing efficient synthetic data-processing techniques will provide higher-quality results. In terms of data formats for 3D models, CAD is preferred [13, 74]. This data format allows for sampling at an arbitrary resolution without any loss of data, and it provides the option of using 3D printing to have an exact real world replica of the model. As a bonus, physical parameters could be attached to each object, so that it could be realistically manipulated in a physics engine.

CHAPTER 4

Methodology, Implementation and Results

This chapter presents approaches and concepts in the implementation of the perception system developed as part of this work. The overall system, called the 3D vision pipeline, is given in Section 4.5 and presents how RGB-D imaging, semantic segmentation, point cloud registration, and shape completion are combined to infer the complete geometry of real-world objects.

Sections 4.2 to 4.4 cover the deep learning methods used in the 3D vision pipeline and present the perception system components in more detail. These sections are organized as follows: First, the network architecture is presented, such that the design intentions and decisions are clearly stated. Second, the implementation details are given, which includes loss function, optimizer, training parameters, etc., necessary for reproducing the results. Finally, I give an evaluation on the network's quantitative and qualitative performance, with the latter being the most important factor in determining the visual appeal and practical usage of the method.

A crucial step in the development of these deep learning methods is the compilation and curation of datasets. Therefore, a discussion on how the dataset preparation was done in this work was presented in Section 3.5, which also gives a reasoning behind the chosen training data for the models.

4.1 Preparation of Datasets

As a learning-based data-driven approach is chosen for the two core parts – semantic instance segmentation and shape completion – of the 3D vision pipeline, datasets for both objects and scenes are needed. The datasets are presented in the two following sections, with information about acquiring, compiling, and converting the datasets so that they have a suitable size and format for the chosen network architectures they will be used for.

4.1.1 3D Model Dataset

Generally, the research on shape completion uses pre-made datasets tailored for learning-based methods. An overview of the most notable datasets was presented in Table 3.1, with ModelNet and ShapeNet possibly being the most frequently used. These datasets contain mostly man-made objects, such as cars, airplanes, chairs, couches, etc., in mesh formats. Since they are used so frequently, they also make a good baseline on which to compare deep learning methods.

In this work, however, it is desirable to have a custom dataset specifically containing 3D models of food objects such as fruit and vegetables, fish and meat, etc. These are objects the robot manipulator in a food processing plant would handle, and it is therefore of interest to see whether completion can be done for these objects as well. In contrast to man-made objects, the same class of an organic object can have widely different sizes and features, but still be identifiable as a specific class. This will inherently increase the difficulty of shape completion.

Effort was made to produce a dataset based on real-world scans, but this unfortunately deemed unsatisfactory results. The approach taken was to use a mobile camera to take 20-30 images of a still-life object (for instance a strawberry attached to a thin straw) from various views, and stitch them together in the software Meshroom [97], which infers the geometry of a scene from a set of unordered photographs or videos. In most cases, even for scans of whole scenes, this software is able to predict the 3D information with errors in the order of centimeters or even millimeters. It turns out, however, that this is still too big for creating realistic 3D objects from scans of smaller objects such as fruit and vegetables. Although the 3D reconstructions do not have to be an exact match with the real-world object used for the scan in creating a dataset, it should still have a similar topology.

Therefore, models of fruits and vegetables, bread varieties, and some processed food were sourced from online providers of 3D models. These models are highly realistic and accurate representations of their real-world counterparts, as can be seen in the collage Figure 4.1 with some objects extracted from the compiled dataset. In fact, each model contain up to as many as 3 million faces. This is an extremely high number of faces, much higher than any neural network has a capacity to represent. It would also take too long to process such objects during training. Subsampling is therefore needed in order to reduce the complexity, which was done using *quadric edge collapse decimation* in a software called Meshlab[21]. This algorithm iteratively picks a mesh region and applies a decimation operator. Decimation involves selecting a vertex in the mesh region, then selecting and removing all the faces sharing that vertex, for then to fill in the resulting hole with new faces. Further, the algorithm then adds the quality of the mesh region after decimation to a queue, and gets the best mesh region from that queue based on an error metric of the difference between the simplified and original mesh region until no further reduction in the number of faces is possible.

The resulting mesh was decided to consist of about 100 000 faces, which based on experimentation turned out to give a satisfactory representation of the object while still not taking too long to process for further use. In this process, it was also verified that the meshes are water-tight, i.e. that no holes or double-sided faces are present in the 3D models. They were also converted and saved as .off-files, which is the filetype used in the ModelNet dataset.

While this results in a dataset that has 3D models in the same format as ModelNet, some further processing was done to adapt the dataset to neural network architectures that do not necessarily take meshes as input, but rather representations such as point clouds. Therefore, the approach described in Section 2.1.2 was implemented in C++ to convert meshes into a point cloud with a pre-defined number of points. Since it is straight-forward to subsample points later, as many as 250000 points were sampled for each mesh. The C++ program first uniformly samples points on the surface of a given mesh, for then to redistribute the points using the Poisson disk sampling method, which gives a good distribution of points on the surface of the 3D model. Moreover, the normal of the face where the point is sampled from is also stored. The normal is calculated by taking the vector product of two of the edges of the

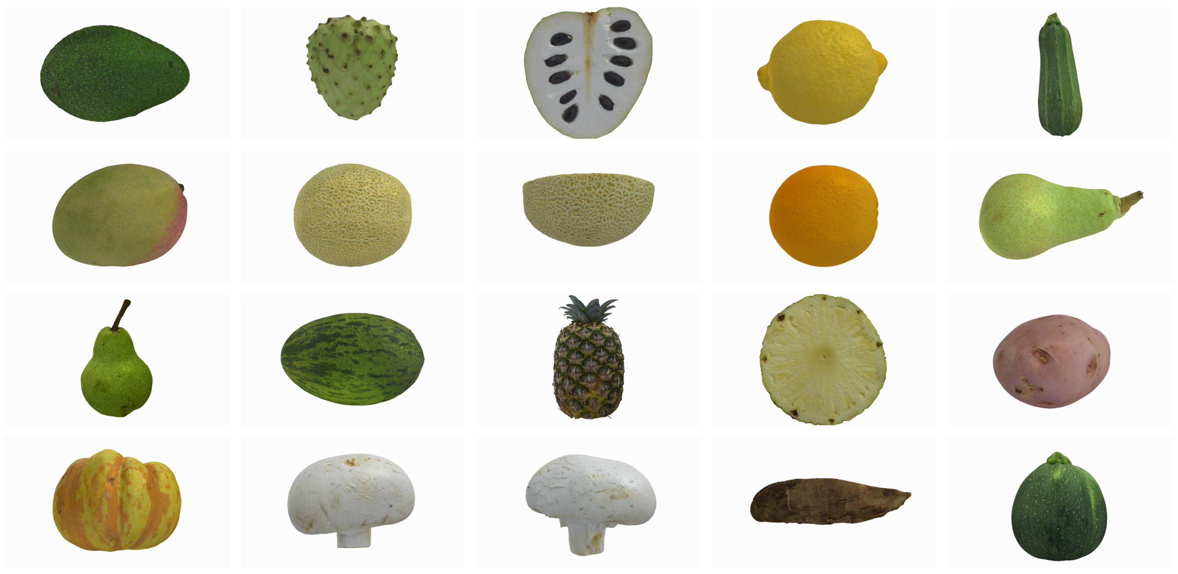


Figure 4.1: The vegetables in the custom-made dataset compiled from highly realistic 3D models, including vegetables such as avocado, lemon, orange, pear, potato, etc.

face. Lastly, the point clouds were scaled to fit within the unit sphere, i.e. the maximum value along any dimension is 1. This is important, as neural networks require scaled data as input. There will also be less variation between the different objects which belong to the same class.

Moreover, a third variation of the dataset was made in order to be able to train a network which would implicitly define the surface of an object. The idea was that using SDF samples for a set of meshes, a network would be able to learn a continuous SDF of the surface. To get these samples, both the point cloud and the mesh representations of the dataset were used. The points in the point cloud dataset was first duplicated where each half was perturbed along all xyz axes with mean-zero Gaussian noise with a variance of either 0.0025 or 0.00025. This resulted in 500000 sampled near the surface of the mesh. It is important that sampling is done to such a high number for points near the surface, since this potentially allows for a model to accurately predict the zero-crossings. However, 25000 points were uniformly sampled within the unit sphere to also more globally indicate the true SDF. For each of these points, the distance to the closest face on the mesh is stored.

A close estimate of the distance could be calculated by constructing a KD-tree of the points and find the closest point in the original point cloud with the L2-norm. The sign of the distance, i.e. whether the point is inside or outside the original point cloud, could be calculated by calculating the dot product between the normal of the point and the vector difference between the point and the closest point from the point cloud. The dot product would be positive if the point is outside the original point cloud, and negative otherwise. However, while being a fast approach in terms of runtime as processing each 3D model is done on the order of a couple seconds, this approach fails for objects that have relatively sharp or thin structures. Figure 4.2 illustrates this problem. As a consequence, deciding the sign of the distance is more complicated than what one would intuitively think. One solution could perhaps be to use the numerically greatest dot product based on the vector difference from the point to a set of points in the original point cloud, but this also fails in some special cases. Bærentzen et al. [3] builds further upon this idea by introducing a pseudonormal which they call the angle weighted normal, and prove that this always determines the correct sign.

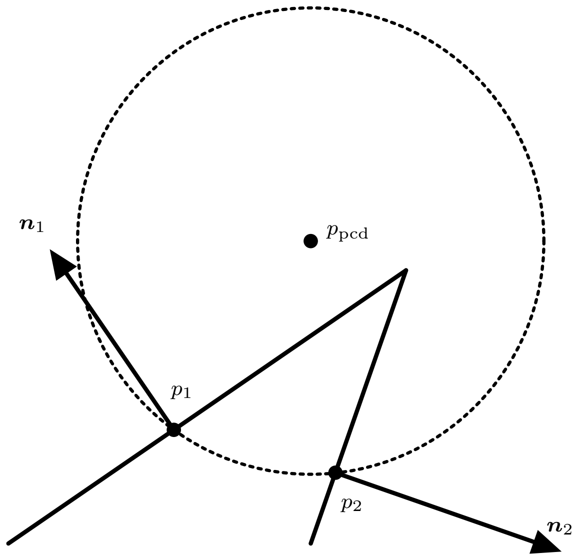
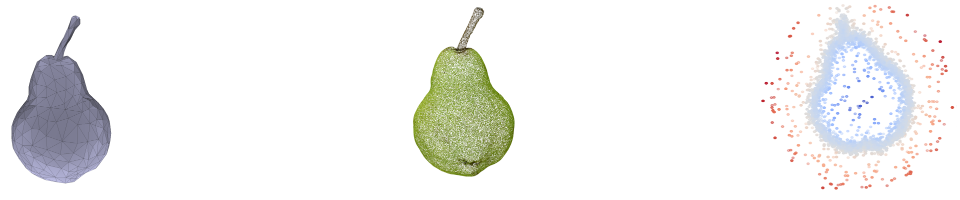


Figure 4.2: Consider the lines in this illustration to be faces of a mesh, and $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ to be their normals. The schematic then shows that it is possible to have the same distance from a point p_{pcd} to two points p_1 and p_2, while the sign of the dot product of the direction vector is not the same for the two normals.

In this work however, I utilize the fact that I have both a point cloud and a mesh representing the same object, and thus the sign can be correctly calculated by the dot product of the point normal and the normal of the closest face in the mesh. Independently of how the points are sampled, this method will always give the correct sign. However, a naive implementation of this method where all the faces are iterated through to find the closest face results in a very long runtime. For instance, calculating the sign for 250000 points sampled around a mesh that is represented with 100000 faces takes about 2 days. Fortunately, similarly to how KD-trees efficiently can be used to calculate distances for point clouds, AABB trees can be used to perform distance queries on meshes. By using this structure to find the nearest mesh face to a point with the Python library trimesh [29], the runtime is reduced to about 5 minutes. This is still a fairly long time, but low enough to convert all the models for one class in a reasonable time-frame. Further improvements could be done by parallelizing the process, for instance by using multiple threads or by moving the process to the GPU.

It is difficult to give a good illustration of the resulting data structure, since the points are scattered uniformly around in a unit sphere. An attempt is made in Figure 4.3(c), where a slice along one dimension is flattened to give a two-dimensional scatter plot with the point colors indicating their SDF values. The surface of the object can be seen by the high number of points in that area. There are also only points with positive SDF values outside the surface (and vice versa), indicating that the method used to generate this data representation is successful. Figure 4.3 also illustrates its corresponding mesh and point cloud representations.



(a) Mesh **(b)** Point Cloud **(c)** Signed Distance Samples

Figure 4.3: Example object in the custom-made 3D object set consisting of fruit and vegetables, processed food, and an assortment of bread. **Left:** A subsampled mesh with the color information removed. **Middle:** A point cloud generated from the mesh using Poisson disk sampling. **Right:** Signed distance samples generated by perturbing the point cloud with Gaussian noise, calculating the distance to the unperturbed points, and finally finding the sign of the distance by registering whether the perturbed points are inside or outside the mesh.

4.1.2 3D Scene Dataset

A 3D scene dataset consisting of color and depth images with groundtruth masks is a necessity in this work. It constitutes the foundation a neural network uses to learn a good segmentation of objects in a scene. For that reason, it should include images that are similar to the given use case – both in terms of the scene setup (structures, lighting, etc.) and in terms of the resolution and quality of the images. Most segmentation networks do not consider any depth component, and therefore use datasets with only color images. In the recent literature on RGB-D segmentation, however, it seems that the SUN RGB-D dataset [] is most frequently used. The segmentation network is therefore also trained on this dataset for comparison purposes with other methods.

While a real-world 3D scene dataset might be the most realistic option to choose, simulated datasets have properties that now outweigh that factor. Additionally, with the UnrealROX [93] engine it is fairly easy to create a highly realistic rendition of any scene. Compared to the real-world datasets, datasets made with this engine have the following advantages: (1) The simulator automatically generates ground-truth masks, which greatly simplifies the generation process. For real-world datasets, researchers have outsourced the mask generation to hundreds of workers using companies such as Amazon Mechanical Turk. These workers draw the mask by freehand, which is a method that is prone to errors and coarse masks. Such a process is clearly tedious and expensive. (2) Any scene can be imaged. Especially for scenes containing food items which decay over time, this can be very helpful. In many cases, it might be impossible to gather real-world images. (3) An unlimited number of virtual cameras can be used to capture a process from multiple views.

In this work, I use the RobotriX [39] studio apartment dataset made with the UnrealROX engine. This dataset was generated by a person wearing a VR-headset which was manipulating objects such as an apple and an orange in a simulated kitchen environment. An example scene is given in Figure 4.4. Since this scene is fairly similar to the designated use case of this work, and the RobotriX dataset contains thousands of images from various views, it is an ideal choice for the final dataset to be used with the segmentation network presented in Section 4.2.

There are, however, some challenges associated with using this dataset. Generally, synthetic

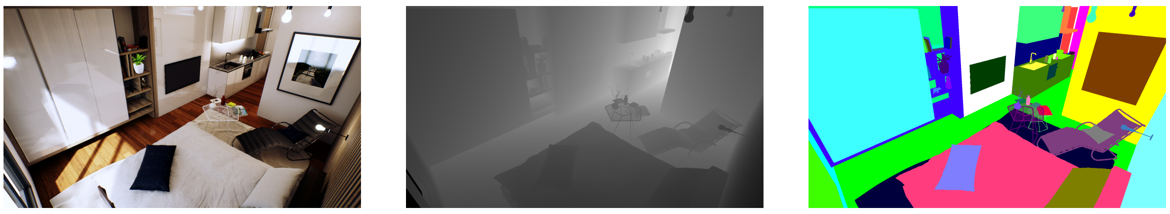


Figure 4.4: Example scene called 'Studio Apartment' from the RobotriX dataset [39], where the ground truth mask is automatically generated for each frame by the UnrealROX simulator [93].

datasets feature numerous discrepancies between them and real-world datasets. This is known as the reality gap. While Robotrix dataset is realistic in terms of the rendition of the scenes, the image capturing process is too perfect compared to particularly the real-world depth sensors, which has numerous faults such as holes occuring in the resulting depth images taken of a scene. Future work on the UnrealROX engine could therefore include adding simulated physical behavior of real RGB-D cameras.

4.2 RGB-D Semantic Segmentation

The related work on semantic segmentation clearly shows that deep learning architectures are by far the methods with the highest accuracy. While the Mask-scoring R-CNN approach seems to be the state-of-the art approach on color images, this network does not take depth into consideration. While depth could be added as another channel to the images propagated to such a network, this would not fully utilize the depth information. Therefore, I've chosen to build my segmentation network upon RedNet [63], as this network seems to have found a good way to incorporate depth into the network architecture. Although it does not perform instance segmentation, this is of lesser importance as most scenes to be segmented in this work only contain unique objects. The output from a semantic segmentation network should then be the same as that from a semantic instance segmentation network.

4.2.1 Network Architecture

An overview of the network architecture for the segmentation network is given in Figure 4.5. This is an encoder-decoder network, which first adds several downsample operations to the input in the encoder, for then to upsample the resulting latent representation. It is strongly inspired by the general ResNet [53] architecture, but has been extended to two branches in the encoder part of the network (the upper half of the flowchart); the color branch and the depth branch. These two branches have the same configuration except for the very first convolution layer, which has 3 channels for the color image input, and only 1 channel for the depth image input. The lower half of the flowchart illustrates the decoder. Several skip-connections, i.e. direct connections from early layers to later layers, are added (illustrated by the Conv2D layers in the left half of the flowchart) to aid with back-propagation and avoid the vanishing-gradient problem often present in networks of this depth. These 2D convolutional layers have kernel sizes 1×1 and strides equal to 1. Note that batch normalization and ReLU activation functions are applied after each convolution operation, but that they are omitted from the figure to avoid too much cluttering.

The encoder consists of a convolution layer followed by a series of four residual layers, each with a different number of residual units, as illustrated in Figure 4.6. The convolution layer encodes the depth image, and increases the feature channel from 3 (or 1 for the depth layer) to 64. The layers one through four have 3, 4, 6, and 3 residual units, respectively. As illustrated in Figure 4.7(a), the residual unit consist of two branches of which the outputs are added together before being propagated to the next layer. However, only the first unit in each residual layer has the right branch with the single convolutional layer. The left branch consists of a 1×1 convolution with stride 1, a 3×3 convolution with stride 2, and a 1×1 convolution with stride 1. In the right branch, the input is propagated through only one convolutional layer (with kernel size 1×1 and stride 2).

The decoder mirrors to a large degree the encoder by also consisting of a series of residual layers. These layers, however, have upsample residual units as illustrated in Figure 4.7(b). ”ConvTranspose2D” in this figure refers to the transposed convolution operator, which is an upsampling operator. It can be seen as the gradient of 2D convolution (”Conv2d”) with respect to its input. The left branch consists of 3×3 convolution with stride 1, and 3×3 transposed convolution with stride 2, while the right branch consists of a single 2×2 transposed convolution with stride 2. Similarly to the residual layers for the encoder, the right branch is only included in one residual unit, but in this case it is for the final unit in the layer. The residual upsampling layers one through five have 6, 4, 3, 3, and 3 units, respectively.

4.2.2 Implementation Details

The network is implemented in PyTorch, with the encoder pretrained on the ImageNet dataset (the network weights are available through the PyTorch interface). For the optimizer, SGD is chosen with momentum $\gamma = 0.9$ and a weight decay ($\mathcal{L}_2$ regularization) of 10^{-4}. The initial learning rate is set to $\alpha = 2 \cdot 10^{-2}$, and decayed with a factor of 0.8 every 80 epochs to slow down training and avoid overshooting as the global optimum is closed in on.

The loss function used takes into consideration all the intermediate outputs shown in Figure 4.5:

$$L(\boldsymbol{s}, \boldsymbol{g}) = \frac{1}{N} \sum_i - \log \left(\frac{\exp(s_i[g_i])}{\sum_k \exp(s_i[k])} \right) \tag{4.1}$$

This is the summation of five 2D cross entropy losses between the output and the groundtruth where $g_i \in \mathbb{R}$ denote the class index of the groundtruth mask on location i (acquired by downsampling the full resolution groundtruth using nearest-neighbor interpolation) and $s_i \in \mathbb{R}^{N_c}$ denote the score vector of the network output on location i with N_c being the number of classes in the dataset. The reasoning behind using this loss function is to address the gradient vanishing problem, which might occur for networks of this depth. In fact, the downsampled cross entropies are given more weight in (4.1) to shorten the backpropagation distance, and incentivize the lower layers in the decoder part of the network to learn the global features, while the higher layers only refine and upsample the output.

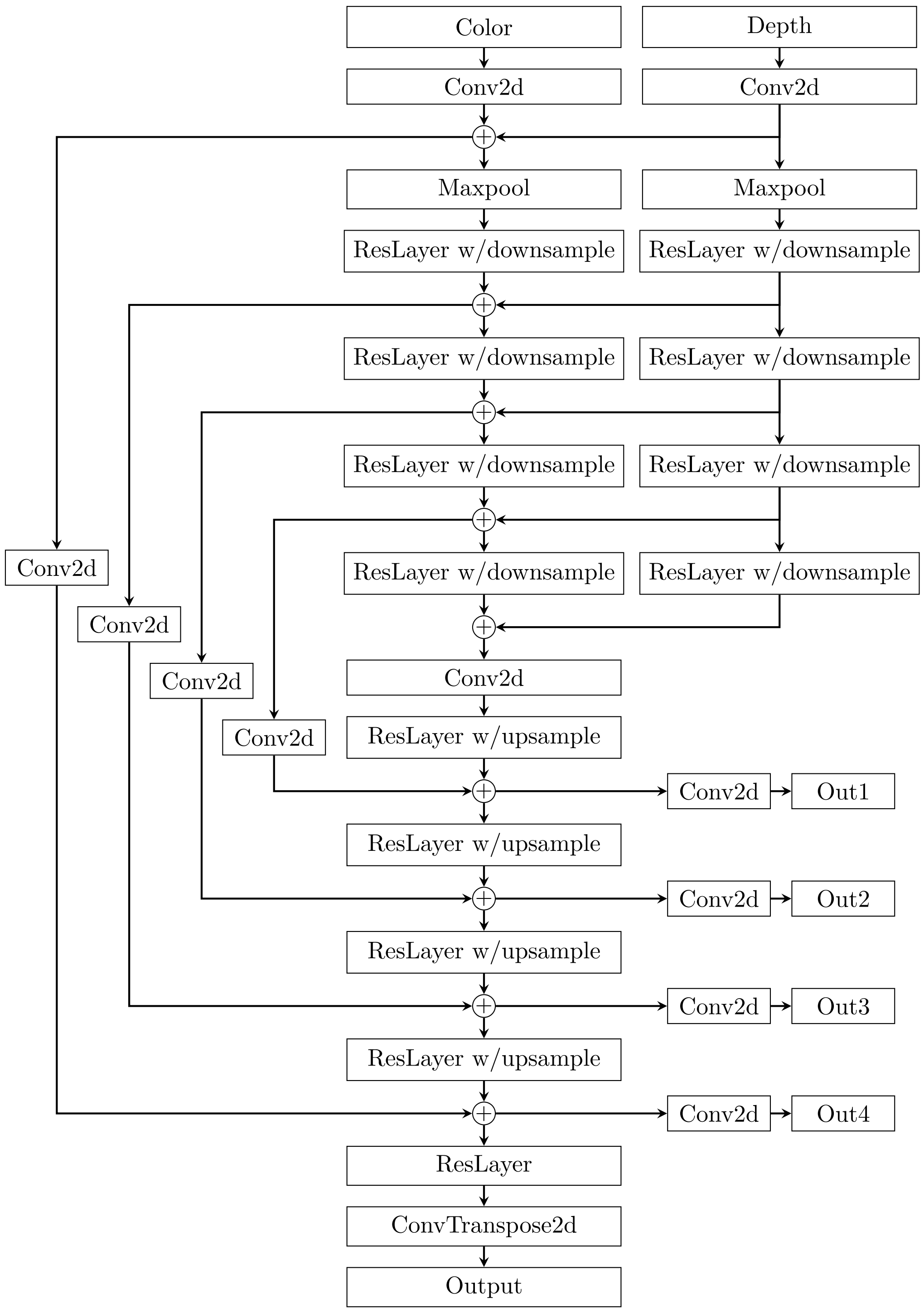


Figure 4.5: Overview of the semantic segmentation network architecture.

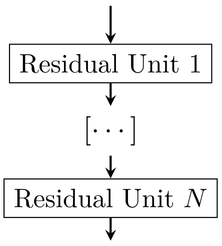
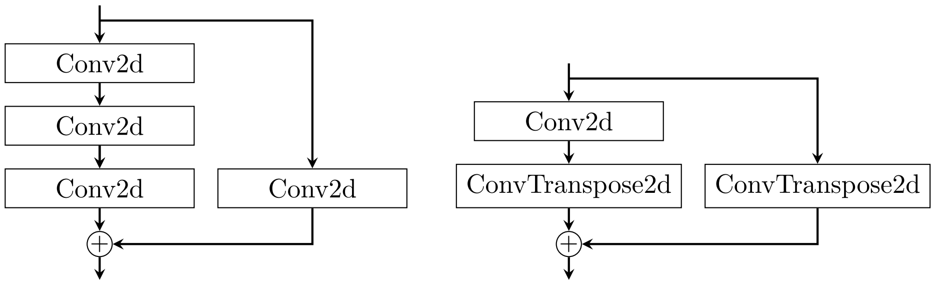


Figure 4.6: Residual unit sequence.



(a) Downsample residual unit

(b) Upsample residual unit

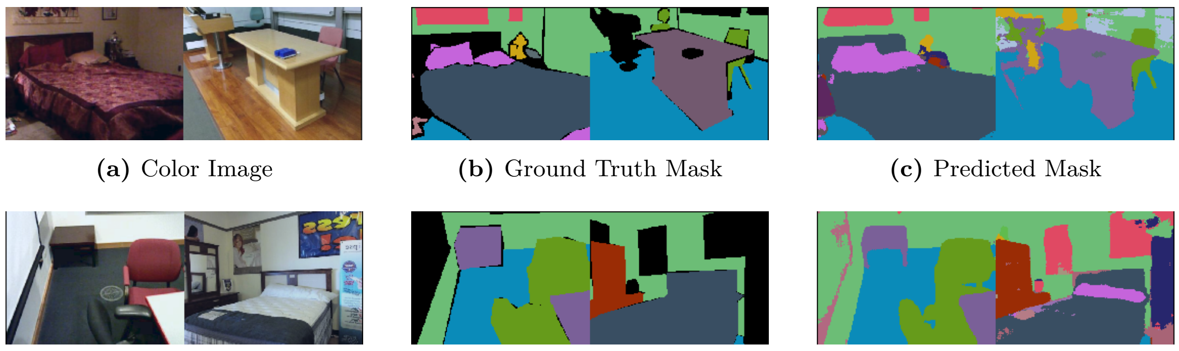
Figure 4.7: Residual units.

4.2.3 Evaluation

By using the semantic segmentation network on the SUN RGB-D dataset I get segmentation masks as shown in Figure 4.8. The network is clearly able to find good masks corresponding to what a human annotator would assign to the images. It achieves a mean IoU of 0.45 on the test partition of the dataset, which is better or on par with other semantic segmentation networks on RGB-D data. This indicates that having a separate branch for color and depth data works as intended, as it seems to effectively exploit RGB-D information where the feature distributions of RGB and depth images vary significantly in different scenes. A qualitative evaluation is needed to better understand the numerical value achieved for the mean IoU, and this will be discussed in the following paragraphs.

Note that the network output mask does not have any black pixels. In the color chart used to visualize the images, I used black to denote the label ”unknown”, i.e. pixels which have not been annotated in the groundtruth masks. The designed network architecture, however, does only have a representation of the class labels, and therefore learns a mapping which fills in the gaps shown in Figure 4.8(b).

In some cases, the network seems to have learned a better mapping than what is given in the ground truth from the SUN RGB-D dataset. One such instance is presented in Figures 4.8(d) to 4.8(f). Here, the table in the upper left corner is annotated as a box in the ground truth mask. Although it might be hard to see in Figure 4.8(d), this annotation is not accurate, as the area under the tabletop is actually the floor darkened by a shadow from the table. The network, however, correctly assigns the mask to the tabletop and its legs. This is also a good



(a) Color Image **(b)** Ground Truth Mask **(c)** Predicted Mask

(d) Color Image **(e)** Ground Truth Mask **(f)** Predicted Mask

Figure 4.8: Example outputs from the semantic segmentation network when trained on the SUN RGB-D dataset with image resolution of 640×480. In the illustration, two and two images are grouped together. The depth image is omitted from this figure, but still used in the network.

example of illustrating the generalizability of this network. Moreover, this adds another reason for using auto-generated datasets – which can only be made in a simulation environment – instead of human annotated datasets.

In images with many objects, such as the background part of the image given in Figure 4.8(a), the network output becomes noisy. The groundtruth mask for these areas are either missing (i.e. assigned the ”unknown” label), or only coarsely follow the outlines of objects in the scene. It should also be mentioned that the number of classes is restricted to 37 for this dataset, which complicates the annotation when objects outside of this list are present (such as the whiteboard eraser on the table in this example). However, the network output for images with such areas do not seem to be overall affected by the mismatched masks of those locations. That is, as in Figure 4.8(c), the table and chair masks are nicely aligned to the groundtruth even though the upper left part of the mask is noisy. This indicates that the network's class label probabilities for the noisy locations are approximately equal. Human annotators would also have trouble giving a correct mask for these occurences, and the network has no opportunity to learn the groundtruth masks as in this dataset they are either too coarse or not present (i.e. the areas are assigned the ”unknown” label). As a consequence of this observation, further extensions should be added to the network such that the network is not forced to output a label for every pixel in the image. This will, however, possibly be a major change as it might require the concept of Bayesian neural networks to be added to the network architecture.

4.3 Shape Completion

Shape completion is at the core of this work, as the methods in this field perform the completion of 3D geometries. Initially, the aim was to adapt the relevant approaches presented in the literature to be applicable to real-world shape completion, but this turned out to be infeasible. The range of possible completions is too large if there are no restrictions on the locations, dimensions and transforms of the objects in the scene, and the scene as a whole. This is why the preceding semantic segmentation network and the following point registration network were introduced. With these two networks processing the RGB-D scan of a scene, only partial scans of objects of interest in a canonical pose are selected as the input to the shape completion agent. Therefore, the shape completion agent should be designed to maximize the accuracy and resolution of the resulting geometry given such an input.

The field of shape completion is still in its early stages, with many new methods emerging recently. Therefore, the design and development of shape completion agents in this work has been an iterative process. Starting from voxel-based models, which were state-of-the art a year ago, the current models output a continuous representation of the resulting geometry, which immensely increases generalizability and resolution. In Section 4.3.1, this design process is presented. The final selected network architecture is the last one given. This network represents the output with a continuous function through occupancy functions such that high-resolution 3D geometries can be achieved.

4.3.1 Network Architectures

Several network architectures have been developed and tested as part of this work. However, most of them did not give satisfactory results. Although these architectures were unfit to be used for real-world shape completion, there is still interesting insight to be gathered from those architectures, such as the reasoning behind their design. Therefore a short presentation of them are given in this section, but specific implementation details will not be addressed except for the final selected shape completion network.

GAN based on voxels The first network developed as part of this work was implemented during my specialization project [77]. It is a generative adversarial network which is able to generate voxelized chairs with a resolution of 32^3 based on a latent vector. A schematic of the architecture is given in Figure 4.9. The discriminator consists of five consecutive 3D 4×4 convolution layers with stride 2, where 3D batch normalization and leaky ReLU activation functions are applied to each layer except the output layer, where a sigmoid activation function is used instead. The generator is almost a mirrored version of the discriminator. It consists of five consecutive transpose of 3D 4×4 convolution layers with stride 2, which function as upsampling operators. In contrast to the discriminator, ReLU activation functions are used instead of the leaky variants.

Although this architecture does not encode a latent vector based on a partial observation, it is able to generate objects that look similar to the ones given in the dataset, as illustrated in Figure 4.10. Technically, it could be extended with an encoder, resulting in an architecture as in the simplified illustration given in Figure 4.11. However, the output resolution with a model based on voxels is too low to be of any practical use, and therefore I did not look further into this alternative.

Encoder-decoder architecture based on point clouds Since the partial observation of a scene is a partial scan represented as a point cloud (alternatively a depth map, which can be directly converted to a point cloud), it could be argued that point clouds should be the 3D representation of choice throughout the network, as all other representations will alternatively be derived from point clouds. This reasoning was the foundation behind researching an improved network architecture compared to the voxel-based network mentioned previously.

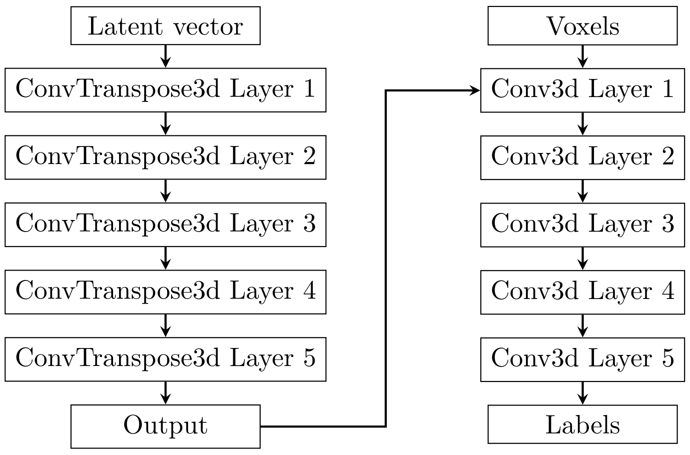


Figure 4.9: 3D-GAN generator and discriminator network architecture.

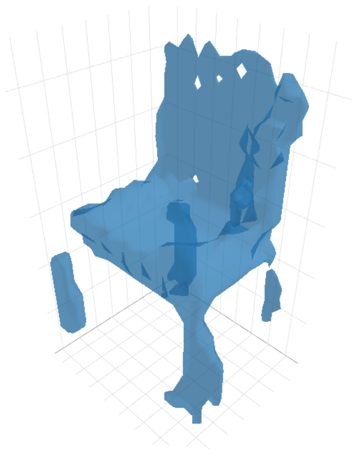


Figure 4.10: A chair generated by the 3D-GAN implemented as part of the specialization project [77]. This network architecture accepts a randomly generated latent vector as input, and is able to generate various objects based on identifying small features in the vector that it has learned to map to some specific object during training. In this case 3D-GAN was trained on a subset of ShapeNet, with 10668 chairs.

Related work by Achlioptas et al. [1] and Yuan et al. [170] seemed to get promising results with point cloud based network architectures.

Therefore, I tested the network given in Figure 4.12. This network is based on an encoder-decoder architecture, where the encoder consists of a series of convolution layers, and the decoder is a series of fully connected layers. The main issue with point clouds in a deep learning setting, is that the data is both sparse and unordered, i.e. the order of the points do not have any inherent meaning. As a result, using the standard convolution operator on point cloud data is not possible for kernel sizes larger than 1. However, as Qi et al. [123] showed, information in point clouds can in fact be encoded with convolution with kernel size 1. In the used encoder, I have four consecutive 1D convolution layers, with 128, 512, 1024,

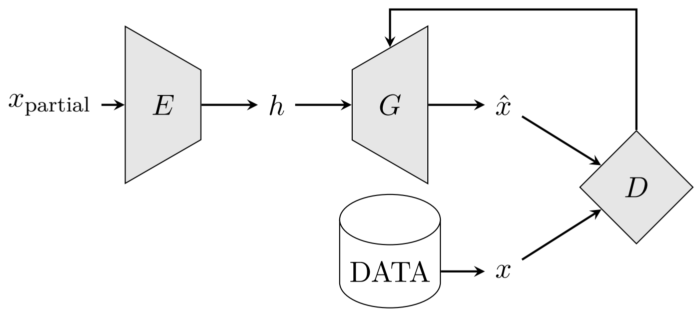


Figure 4.11: An extended version of a GAN, where the latent vector fed to the generator is made by an encoder. Since the encoder-generator pair resembles that of an encoder-decoder pair in a VAE (and could be designed in the same manner), this architecture is referred to as VAE-GAN.

and 1024 output channels, respectively. ReLU activation is used in each convolution layer. I also add a point-wise maxpool branch which finds a global feature of the output from the second layer, and repeat this to match the vector dimension of the output such that they can be concatenated before being passed to the next layer. The intent behind this branch is to increase the network's invariance to permutation and tolerance to noise. The decoder, on the other hand, consists of two fully connected layers each with 1024 output features and a final fully connected layer with 3 times the desired number of points in the resulting point cloud. Leaky ReLU activation is used for each layer, except for the final layer which uses a sigmoid layer to avoid any bias in the output.

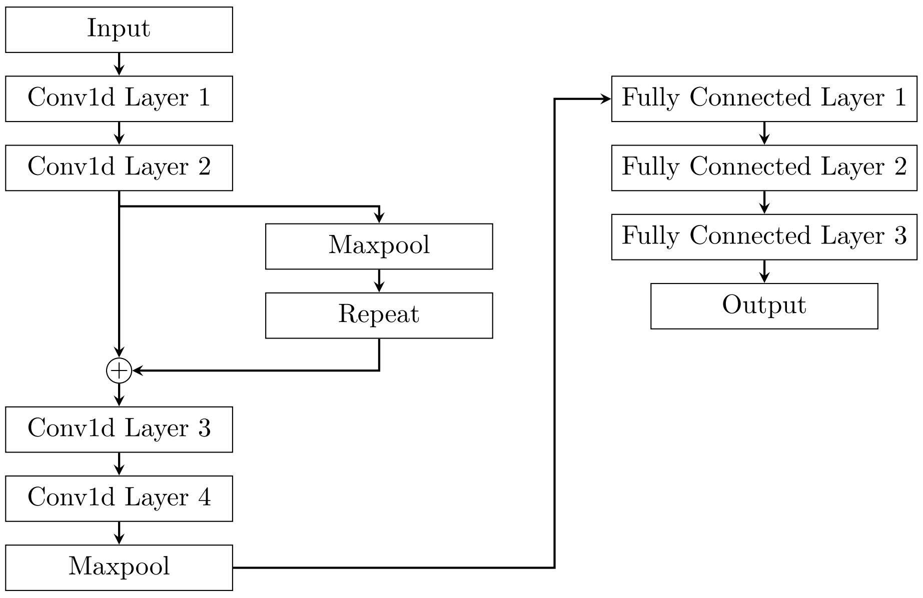
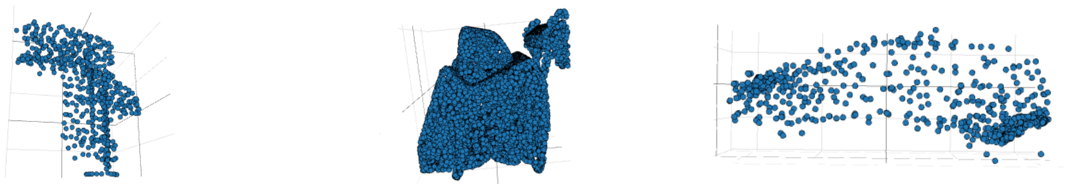


Figure 4.12: Network architecture for the encoder-decoder architecture based on point clouds.

By using the Adam optimizer and Chamfer distance (presented in Section 2.1.3) as the loss function, I get shape completions as shown in Figure 4.13. Here, the network has been trained on a subset of the ShapeNet dataset which consists of a variety of cars. As seen in the figure, the completed point cloud does not resemble either the input or the ground truth. The completed point cloud, in isolation, is a good point cloud representation of a sedan or a coupe. It turns out that these two types of car have by far the most instances of 3D models in the dataset. Therefore, it seems like the network has only been able to learn a general representation of the dataset, and outputs variations of this for each shape completion. This is a possible indication of the encoder not being able to produce a fully detailed representation. Alternatively, only the decoder part of the overall network was trained, possibly caused by a vanishing gradient problem. However, inspection of the weights during training indicated that they were changing for all layers. These challenges have been encountered by other researchers too, and was recently extensively studied by Tatarchenko et al. [147]. They claim that several state-of-the art single-view reconstruction and completion networks does not actually perform reconstruction, but image classification.



(a) Input **(b)** Ground Truth **(c)** Completion

Figure 4.13: Shape completion using the encoder-decoder architecture based on point clouds. The network was in this case trained on a dataset consisting of cars which were mostly of the type sedan and coupe. **Left:** A partial observation of a trolley. **Middle:** The completed ground truth point cloud of the partial observation. **Right:** A sedan is outputted from the network, which does not match the observation.

Autodecoder based on signed distance functions Early this year, as discussed in Section 3.4.3, multiple papers were published which argued for learning a continuous function as the output from the network. In this way, the problem is simplified into a binary classification problem where the surface of an object is represented implicitly by a decision surface. One of these network architectures, DeepSDF [113], is based on the idea to use the network to map SDF samples (generated by the dataset preprocessing described in Section 4.1.1) to a continuous SDF. By propagating samples from various objects, including variations within the same object, a neural network should in theory be able to estimate a smooth function that basically functions as a binary decision surface, separating the inside of the objects from the outside.

In my experimentation, I tried to replicate their results by imitating their proposed network architecture. It is a series of fully-connected layers, each with 512 output channels, as illustrated in Figure 4.14. Additionally, a skip-connection is added where the latent vector is concatenated to the output from the fourth fully connected layer before being passed into the fifth layer. This is done to improve the gradient flow in the network, which is quite deep due to its eight layers.

Unfortunately, I could not get the loss to decrease during the training of this network. The

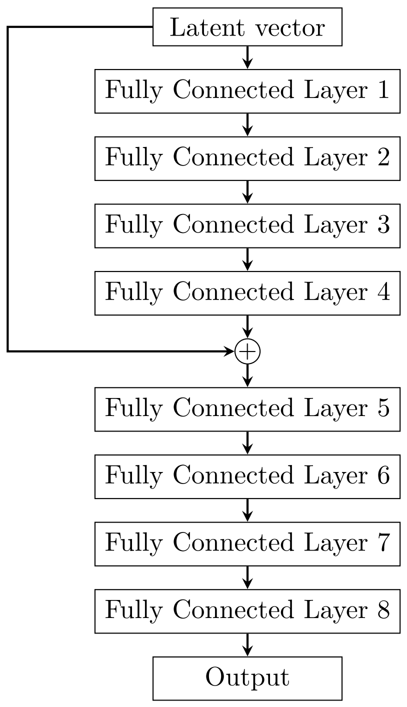


Figure 4.14: A feed-forward autodecoder network.

authors of DeepSDF did indicate that training could become unstable and that they had to tune the training parameters, but they did not disclose all their values. I could not find any indication of a better loss curve regardless of my choices of training parameters, and therefore moved on from this network architecture. It should also be noted that a fully connected network has the apparent disadvantages of not using any spatial information which could be present in the data, which also motivates a more complex network architecture.

Autoencoder based on occupancy functions The overall network architecture of the occupancy function network is presented in Figure 4.15. It is the network considered for the remainder of this section on shape completion. As seen in the overview, the network accepts three components as the input, and passes them through three subnetworks to produce an output. The inputs include the ground-truth points (i.e. the full-resolution point cloud), the partial input (a subset of the ground-truth points, alternatively a different set of points sampled from the mesh), and the occupancies which state whether the ground-truth points are inside or outside the surface of the corresponding mesh.

The encoder is based on a PointNet architecture with ResNet blocks, as illustrated in Figure 4.16(a). This is a fully-connected network, as it only consists of fully-connected layers. In total, there are 5 ResNet units, where the output from each unit is concatenated with the maxpooled output before being passed into the next unit. Each ResNet unit shown in Figure 4.17(a) consists of two branches of fully-connected layers, where the output is the sum of the input passed through either two or one fully-connected layers. ReLU is used as the activation function for all layers.

The decoder, on the other hand, is a series of 1D convolutional layers of which some are organized as ResNet Units, as illustrated in Figure 4.16(b). All convolutional layers use a kernel

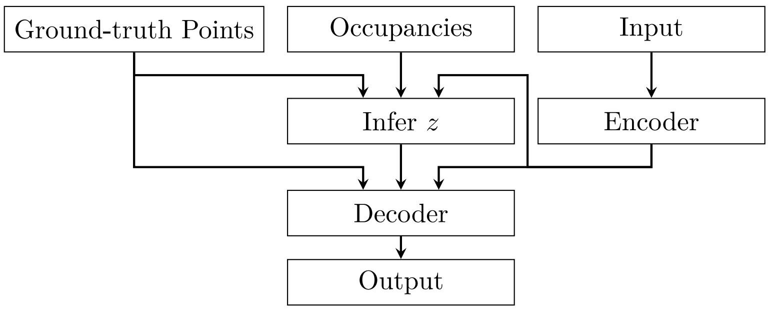


Figure 4.15: Shape completion network architecture. Three inputs are given to the network which consists of three subcomponents. The *Infer c* component is just a Normal distribution that is trained with the rest of the network.

size of 1, and are followed by ReLU activation. The ResNet units shown in Figure 4.17(b) have the same architecture as the ResNet units used in the encoder, except that convolutional layers are used instead of fully-connected layers.

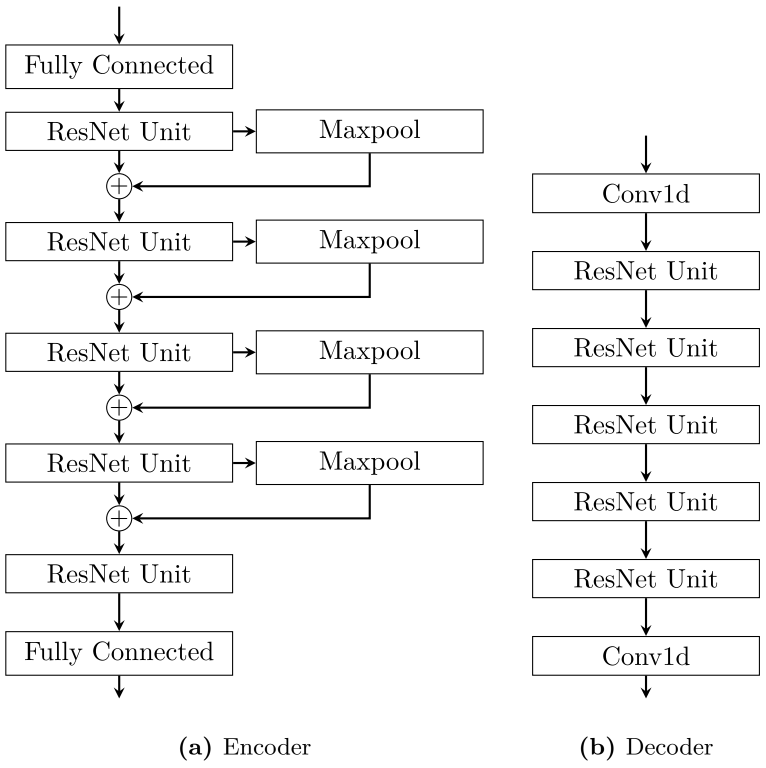
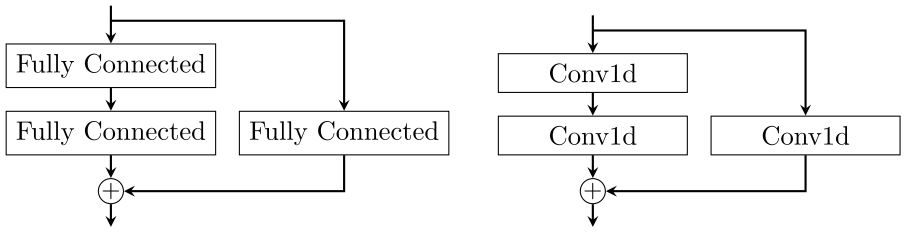


Figure 4.16: Shape completion encoder and decoder architecture.



(a) Residual unit with fully connected layers used in the PointNet encoder for the shape completion network

(b) Residual unit with convolutional layers used in the decoder

Figure 4.17: Residual units used in the encoders and decoders of the shape completion network.

4.3.2 Implementation Details

The implemented PyTorch model follows the final network architecture discussed in Section 4.3.1; where the aim is to model the 3D objects as occupancy functions. The Adam optimizer with an initial learning rate of 10^{-4} is chosen for updating the network parameters. Additionally, a multistep learning decay is added to slow down the learning rate with a factor of 10 after 75, 150, and 225 epochs.

The chosen loss function used to learn the parameters of the neural network $f_\theta(p, x)$ is a cross-entropy classification loss averaged over the batch $\mathcal{B}$:

$$L(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{\mathcal{B}|} \sum_{j=1}^{K} L(f_\theta(p_{ij}, x_i), o(p_{ij})) \tag{4.2}$$

Here, x_i is the i'th observation of the batch, $p_{ij} \in \mathbb{R}$, $j = 1, \ldots, K$ are points sampled from the i'th observation, and $o(p_{ij})$ denotes the true occupancy at point p_{ij}.

For inference, the surface of the occupancy function needs to be extracted. This could be done with marching cubes, but here I use an extension to this method presented by Mescheder et al. [96]. This method is called *multiresolution isosurface extraction* (MISE), and is able to efficiently produce high resolution meshes from the occupancy network. The volumetric space is first discretized at an initial resolution of e.g. 32^3, and then the occupancy network $f_\theta(p, x)$ is evaluated for all points p on this grid. The points p for which $f_\theta(p, x)$ is greater than some threshold τ are marked as occupied. Then the voxels for which two adjacent grid points have differing occupancy predictions are marked as *active* voxels, i.e. these will be subdivided into eight subvoxels for which new grid points are introduced to improve the resolution. These steps are then repeated until the final desired resolution is reached, for which the original marching cubes algorithm is applied:

$$\{p \in \mathbb{R}^3 \mid f_\theta(p, x) = \tau\} \tag{4.3}$$

In total, this inference stage takes 2-3 seconds, which can be too long for some applications. This algorithm could, however, be parallelizable on the GPU. Alternatively, the final resolution could be adjusted to render a coarser output.

4.3.3 Evaluation

By training the shape completion network on a subset of the ShapeNet dataset I get accurate predictions of the complete geometry of the test objects. Table 4.1 gives a quantitative evaluation of the network, along with equivalent metrics reported for two former state-of-the art networks. These networks are trained on the same subset as the network implemented as part of this work, and can thus be compared directly. The shape completion network that I implemented gains a few percentage points improvement compared to the other networks. The papers on shape completion discussed in Section 3.4.3 have not reported the same metrics or used the same dataset, and are therefore omitted from this overview.

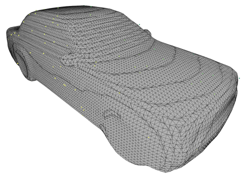
Model	IoU	IoU (voxelized)
This network	0.62	0.69
3D-R2N2	-	0.57
DMC	-	0.65

Table 4.1: Test metrics on unseen partial input for this network and two other former state-of-the-art networks. I measure the intersection over union (IoU) and the IoU with a voxelized representation. All networks are trained on the same subset of ShapeNet, such that their metrics can be compared directly.

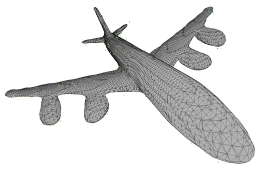
A qualitative evaluation can be done based on example outputs from the network, such as the ones given in Figure 4.18. In this figure, the completed shapes are converted to meshes using the MISE approach described in the previous section. The meshes could be further converted to point clouds, if desirable. For the input to the network, a point cloud with as few as 300 points is used. It turns out that this is enough points to uniquely represent most objects. A resulting advantage of this observation, is that inference can be done in a short amount of time, as only a few points are propagated through the network. The main bottleneck during inference is in fact the conversion step from the occupancy function to the mesh. On the other hand, while the network accepts any number of points as input, it does not seem like the completed object has a noticeable improved geometry if more points are used in the point cloud used as the partial observation.

Based on the meshes, and the partial observation (point clouds), it is clear that for all classes the completion is detailed and accurate. The network is able to capture small details, such as the side mirrors of the car in Figure 4.18(a). This means that if the side mirrors are not visible from a certain view, the shape of the remainder of the car indicates that they should be present, and the network has learned this connection. Additionally, holes such as the ones underneat the armrests in Figure 4.18(d), are correctly preserved in the completed geometry. The network also has the capacity to simulateneously represent all classes (12 in total) in the training dataset, indicating its extensibility to learn representations for a wide range of objects.

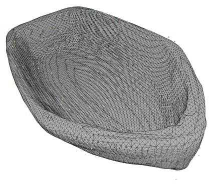
It should also be noted that all resulting meshes are water-tight. This is an inherent result inforced by having the network architecture output occupancy functions.



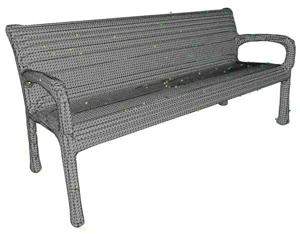
(a) Car



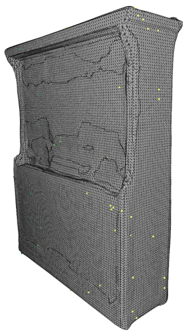
(b) Airplane



(c) Vessel



(d) Bench



(e) Cabinet

Figure 4.18: Shape completion of unseen objects from the ShapeNet dataset, but of the same categories as used for training. In this illustration, the input partial observation is overlaid as yellow points on the resulting mesh. The wireframes of the meshes are also visible, to showcase the high resolution (i.e. number of faces).

4.4 Point Cloud Registration

Point cloud registration is a crucial component in adapting shape completion neural networks to be usable for real-world problems, as these networks require the partial observation to be approximately in a canonical pose (defined by the objects used in the prior dataset). However, achieving this pose is a strictly non-trivial task, as the alignment of the partial scan requires finding a transformation that includes scaling, rotation, and translation – resulting in 7 degrees of freedom. Ideally, this alignment of a partial scan should be done compared to a unit scale axis. However, this seems to be an unsolvable problem as it is highly under-constrained, and the optimal alignment is not uniquely defined. Moreover, only part of the object is given, which means that several standard algorithms fail.

On the other hand, if a complete point cloud of the object was given, the optimal translation and rotation between two point clouds $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\}$ (input object) and $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i, \ldots, \boldsymbol{y}_N\}$ (corresponding object in the dataset) of the same object could be found in the following manner. First, the centroid of both clouds are found by averaging their points:

$$\begin{aligned}\text{centroid}_{\mathcal{X}} &= \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i \\ \text{centroid}_{\mathcal{Y}} &= \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{y}_i\end{aligned} \tag{4.4}$$

Then the two matrices are multiplied together to form an intermediary matrix,

$$\boldsymbol{H} = \sum_{i=1}^{N}(\boldsymbol{x}_i - \text{centroid}_{\mathcal{X}})(\boldsymbol{y}_i - \text{centroid}_{\mathcal{Y}})^\top \tag{4.5}$$

By taking the singular value decomposition of this matrix,

$$[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \text{SVD}(\boldsymbol{H}) \tag{4.6}$$

the rotation between the two clouds is given as

$$\boldsymbol{R}_{\mathcal{X}\mathcal{Y}} = \boldsymbol{V}\boldsymbol{U}^\top \tag{4.7}$$

and the translation as

$$\boldsymbol{t}_{\mathcal{X}\mathcal{Y}} = -\boldsymbol{R}_{\mathcal{X}\mathcal{Y}} \times \text{centroid}_{\mathcal{X}} + \text{centroid}_{\mathcal{Y}} \tag{4.8}$$

This method assumes that both point clouds have the corresponding points in the same order (i.e. that $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are paired), are of the same scale, and with limited noise. If there are more than three points in the point cloud, this method minimizes the mean-squared error

$$E(\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}) = \frac{1}{N}\sum_{i}^{N}||\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{y}_i||^2 \tag{4.9}$$

Since this method obviously fails in the general case, a more sophisticated approach is needed. ICP is generally the method of choice, but it fails in this case as it is only able to converge to a local optimization, which would require the two point clouds to be roughly aligned initially. Go-ICP, which tries to find a global optimum, often fails by converging to anti-symmetries in the point clouds. Surprisingly, it turns out that the only feasible method is a deep learning network which can find global mappings between point cloud alignments. ICP could optionally be used with the network's output to refine the alignment.

The network implemented as part of this work is based on Deep Closest Point [155], which performs accurate alignment of point clouds. With this approach, the goal is to learn embeddings which recover a matching $m(\cdot)$,

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \arg\min_j ||\mathcal{R}_{\mathcal{X},\mathcal{Y}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{X},\mathcal{Y}} - \boldsymbol{y}_j|| \tag{4.10}$$

which finds the corresponding $\boldsymbol{y}_{m(x_i)}$ to each $\boldsymbol{x}_i$. The objective function (4.9) is then slightly modified to account for matching:

$$E(\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}) = \frac{1}{N}\sum_{i}^{N}||\boldsymbol{R}_{\mathcal{X},\mathcal{Y}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{y}_{m(x_i)}||^2 \tag{4.11}$$

However, this network assumes the point clouds to be of the same scale, i.e. it only performs a rigid transformation with respect to rotation and translation. Therefore, I add a heuristic assumption on scaling where the point clouds are scaled to the unit sphere, which makes them roughly the same size. Further, I also change one of of the input point clouds during training to only contain a subset of the points, which is done to emulate partial scans. The other point cloud to align the partial scan to is determined based on a corresponding object from the class determined by the segmentation network.

4.4.1 Network Architecture

The overall network architecture of the point registration network is presented in Figure 4.19. In short, the network consists of two branches – one for the source point cloud and one for the target point cloud which the source point cloud should be transformed to by rotation and translation. Each branch has an embedding network, which learns a high-dimensional representation of the point clouds, and a transformer, which encodes contextual information. The branches are intertwined, as seen in the figure, and finally joined together in the SVD layer, which estimates the alignment between the point clouds. These three types of layers are presented in the following paragraphs.

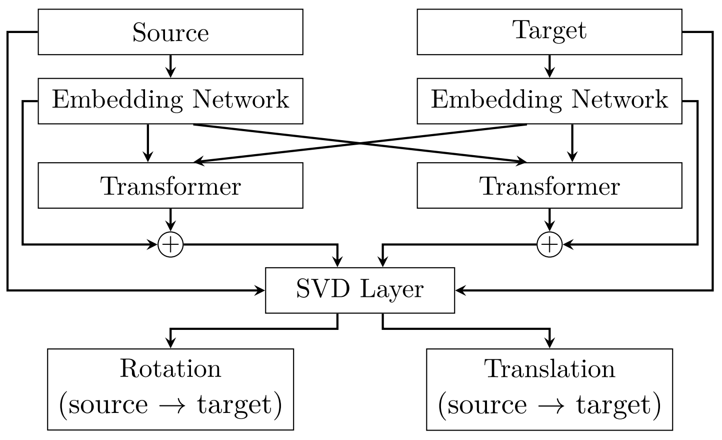


Figure 4.19: Overview of the point cloud registration network architecture.

The embedding network, which is based on a module called DGCNN [156], seeks to find a feature per point in the source and target point clouds. DGCNN has proven to produce a good representation by explicitly incorporating local geometry. It constructs a nearest neighbor graph $\mathcal{G}$, that is used as the input to the network, as illustrated in Figure 4.20. The convolutional layers all have kernel size equal to 1 and stride 1, which means that they only apply a nonlinearity to the input. The series of operations can be summarized with the following equation:

$$\boldsymbol{x}_i^l = f(\{h_\theta^l(\boldsymbol{x}_i^{l-1}, \boldsymbol{x}_j^{l-1})\,\forall j \in \mathcal{N}_i\}) \tag{4.12}$$

where $\mathcal{N}_i$ denotes the neighbors of vertex i in graph $\mathcal{G}$. PointNet [123] could potentially also be used as the embedding network, but DGCNN is able to find local neighborhood information which is favorable for later steps in the overall point cloud registration network.

A transformer is a network architecture based on a self-attention mechanism that especially in the natural language processing domain outperforms both recurrent and convolutional

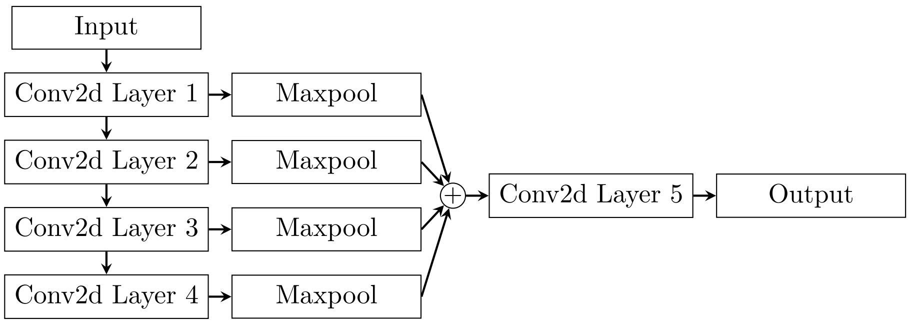


Figure 4.20: DGCNN embedding network.

models [151]. The network architecture is given in Figure 4.21. As this figure shows, the architecture includes two subcomponents; simple feed-forward networks and multi-head attention layers.

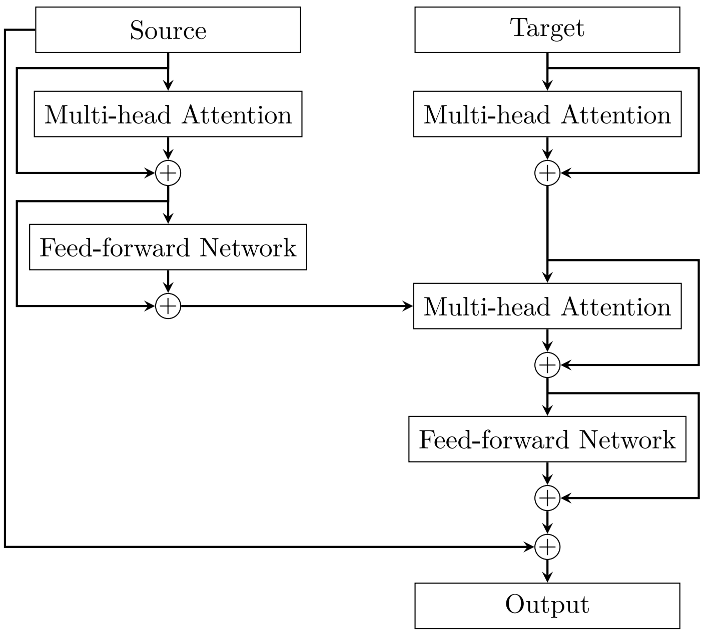


Figure 4.21: Transformer network architecture.

In Figure 4.19 the transformer is used to learn a function $\phi : \mathbb{R}^{N\times P} \times \mathbb{R}^{N\times P} \mapsto \mathbb{R}^{N\times P}$, where P is the embedding dimension, which provides new embeddings for the points clouds,

$$\begin{aligned} \Phi_{\mathcal{X}} &= \mathcal{F}_{\mathcal{X}} + \phi(\mathcal{F}_{\mathcal{X}}, \mathcal{F}_{\mathcal{Y}}) \\ \Phi_{\mathcal{Y}} &= \mathcal{F}_{\mathcal{Y}} + \phi(\mathcal{F}_{\mathcal{Y}}, \mathcal{F}_{\mathcal{X}}) \end{aligned} \tag{4.13}$$

Here, $\mathcal{F}_{\mathcal{X}}$ and $\mathcal{F}_{\mathcal{Y}}$ are the embeddings produced by the previous layer. The intent is to modify the original embeddings by adding the residual term ϕ which has encoded the structure of the other point cloud. The updated embeddings are used to find a probability vector which maps one point cloud to the other, i.e. each $\boldsymbol{x}_i \in \mathcal{X}$ is assigned a probability vector over elements of $\mathcal{Y}$ given by

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \operatorname{softmax}(\Phi_{\mathcal{Y}} \Phi_{\boldsymbol{x}_i}^{\top}) \tag{4.14}$$

The SVD layer extracts the matched points in the point cloud $\mathcal{Y}$ for each point in $\mathcal{X}$ from (4.14):

$$\hat{\boldsymbol{y}}_i = Y^\top m(\boldsymbol{x}_i, \mathcal{Y}) \in \mathbb{R}^3 \tag{4.15}$$

Here, $Y \in \mathbb{R}^{N\times 3}$ is defined to be a matrix containing the points in $\mathcal{Y}$. The rotation $\boldsymbol{R}_{\mathcal{XY}}$ and translation $\boldsymbol{t}_{\mathcal{XY}}$ are then found with (4.7) and (4.8) based on the pairing $\boldsymbol{x_i} \mapsto \hat{\boldsymbol{y}}_i \forall i$.

4.4.2 Implementation Details

Like the other networks previously discussed, the point cloud registration network is also implemented in PyTorch. The Adam algorithm with weight decay 10^{-4} is chosen to optimize the network parameters. The learning rate is initially set to $\alpha = 10^{-3}$, and divided by 10 every 75 epochs.

The chosen loss function measures the deviation of the rotation and translation of a pair of point clouds from the ground truth using mean squared error:

$$L(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}) = ||\boldsymbol{R}_{\mathcal{XY}}^\top \boldsymbol{R}_{\mathcal{XY}}^g - \boldsymbol{I}||^2 + ||\boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{t}_{\mathcal{XY}}^g||^2 \tag{4.16}$$

In this equation, $\mathcal{X}$ and $\mathcal{Y}$ are a pair of point clouds which are mapped with a rigid motion $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ (rotation, translation) that aligns them together. The ground-truth is denoted by the superscript g.

During training, 1024 points are sampled from objects in the ModelNet40 dataset and the custom 3D dataset, and then centered and scaled to fit in the unit sphere. This source point cloud $\mathcal{X}$ is then applied a random rigid transformation including a rotation between 0 and 45 degrees, and a translation between -0.5 and 0.5. The transformed point cloud is used as the target.

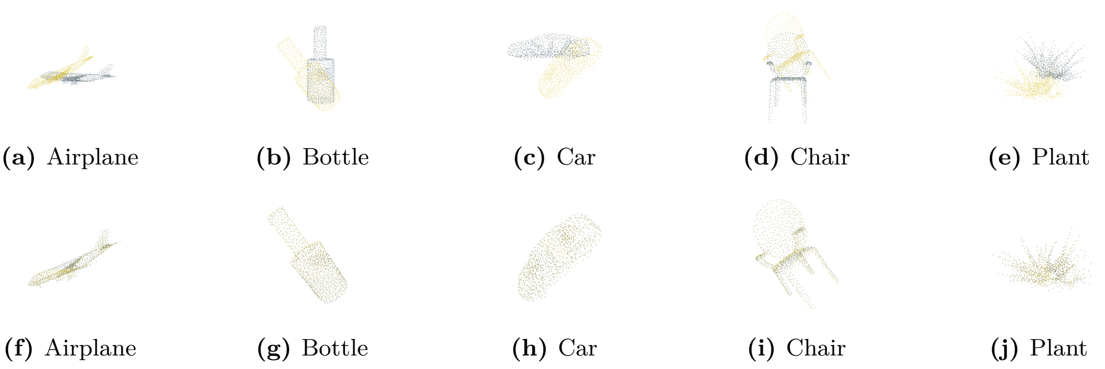
4.4.3 Evaluation

The point cloud registration network as presented outperforms the other alternative methods on point cloud registration. A quantitative summary of the test metrics for the network is given in Table 4.2, along with corresponding metrics for ICP and Go-ICP. It should also be noted that the inference time for the network on a point cloud with 1000 points is less than 0.1 seconds, which is faster than both of the other methods (ICP is slightly slower, while Go-ICP's inference time is on the order of about 15 seconds).

Model	MSE (rotation)	MAE (rotation)	MSE (translation)	MAE (translation)
This network	3.3	1.2	0.00001	0.002
ICP	895	25	0.08	0.25
Go-ICP	140	2.5	0.0007	0.007

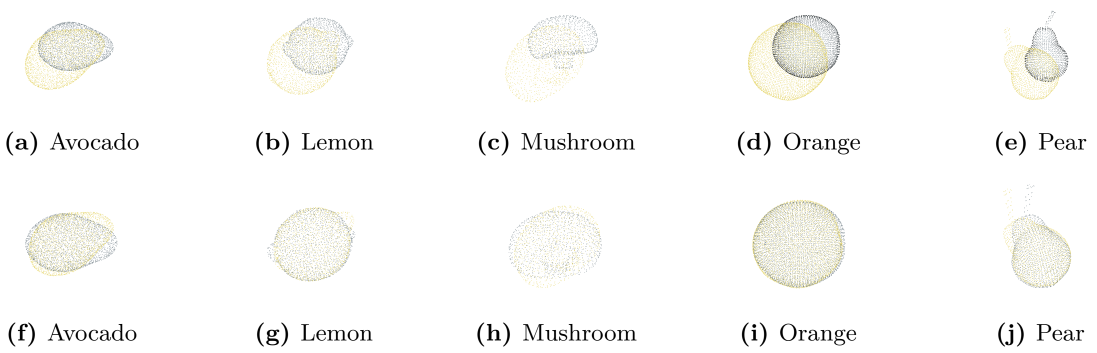
Table 4.2: Test metrics on unseen point clouds for this network, ICP, and Go-ICP. I measure the mean squared error (MSE) and mean absolute error (MAE).

One of the core design requirements of the network is that it should be flexible in terms of how many points are passed as the input. The network satisfies this requirement, and the number of points in the source and target point clouds don't even have to be the same. There is a practical limitation, however, due to the GPU memory limitations (8 GB for the Nvidia GeForce GTX 1080) which restrains the maximum points in either cloud to be less than 4000 points. For this network to be used in a production environment, it might therefore be necessary to use a modern high-end GPU, or use an array of GPUs linked together. Since the core OS functions of the host computer use about 3 GB memory, slightly increasing the GPU memory will potentially double the number of points that could be passed through the network.


(a) Airplane **(b)** Bottle **(c)** Car **(d)** Chair **(e)** Plant

(f) Airplane **(g)** Bottle **(h)** Car **(i)** Chair **(j)** Plant

Figure 4.22: Point cloud registration of objects from the ModelNet40 dataset. **Top row:** Unaligned point clouds. **Bottom row:** Aligned point clouds.


(a) Avocado **(b)** Lemon **(c)** Mushroom **(d)** Orange **(e)** Pear

(f) Avocado **(g)** Lemon **(h)** Mushroom **(i)** Orange **(j)** Pear

Figure 4.23: Point cloud registration of objects from the custom food object dataset. **Top row:** Unaligned point clouds. **Bottom row:** Aligned point clouds.

By evaluating the sample inferences given in Figure 4.22 and Figure 4.23, it seems like the network is able to generalize to unseen cases. All these point clouds are not in the training set, and therefore have not been affecting the learning stage of the network. Nevertheless, for the objects in the ModelNet40 dataset (top row) the alignment is close to perfect, while for the custom dataset consisting of food objects (middle row) the alignment is an improvement compared to the original unaligned cases, but with some offsets. The reason behind the lesser alignment of the food objects could partially be explained by these objects being more different from most of the objects in the training dataset, as they constitute a smaller number of the

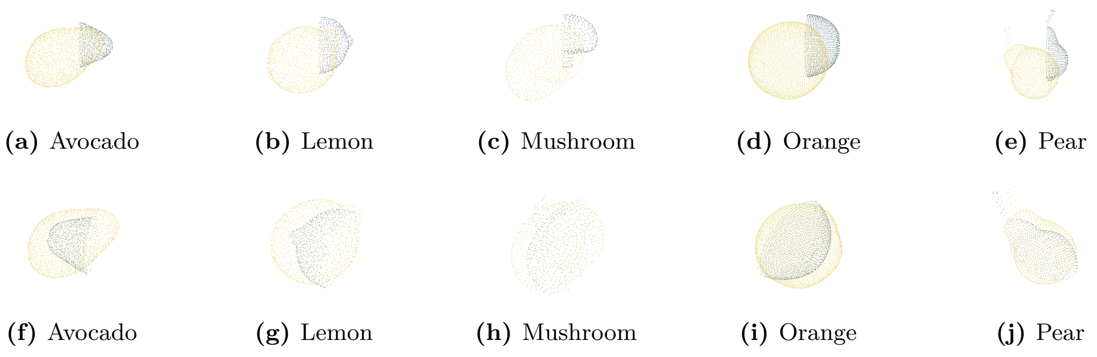


Figure 4.24: Point cloud registration of objects from the custom food object dataset, where half of the points in the source point cloud is removed. **Top row:** Unaligned point clouds. **Bottom row:** Aligned point clouds.

training samples. Additionally, it could indicate that the intraclass variance among the food objects are bigger than the classes and objects given in the ModelNet40 dataset. Considering this caveat, the alignment is performing fairly well. Therefore, effort should be done to make a bigger dataset which covers most variations of fruit and vegetables, as this would most likely solve these alignments issues.

Figure 4.24 depicts the same objects as in Figure 4.23, and with the same initial transformation applied to the target point cloud, but with about half of the source points missing. With these conditions, the network output is still able to rotate and translate the two point clouds in the right direction, but the source point cloud (corresponding to the partial input) is centered inside the target point cloud, instead of being slightly offset to match up with the target points. The reasoning for this result might be due to the network optimizing the mean distance between all the points in both point clouds, instead of just the points in the point cloud with the lesser number of points. Thus, the nearest-neighbor distance should be aimed to be minimized, but only for the points in the source point cloud. This necessitates a change in the network architecture, that would imply making it less symmetrical than the current structure. For an improvement to be made, the goal should be to only minimize the distances between the points in the point cloud with the least amount of points to the other point cloud, and ignore any optimization regarding the latter point cloud. Although these idea might prove a promising direction for future work, currently ICP should be used on the output of the network to produce a better alignment, as the network makes a global optimization that enables ICP to perform its local optimization.

4.5 Combining the Building Blocks to Create a 3D Vision Pipeline

All the previous methods are interesting on their own, but they have to be combined in order to solve the main aim of this study. Figure 4.25 gives a schematic overview on the proposed program flow for how the entire process from capturing an RGB-D image to inferring the complete geometry of the objects in a scene. During this work it became apparent that the necessary subproblems for real-world shape completion is naturally a series of succeeding steps, where the output from one step is the desired input of the next. The pipeline is therefore an end-to-end framework for completing 3D geometry, which combines and automates all the necessary processes needed for optimal completion.

In the following, I discuss the responsibility of the steps and how they relate to the methods presented earlier in this chapter. Some of the steps are more strongly related, and will be discussed jointly. I will also add more specific schematics to some steps, to illustrate more in detail what the abstract steps in Figure 4.25 involve.

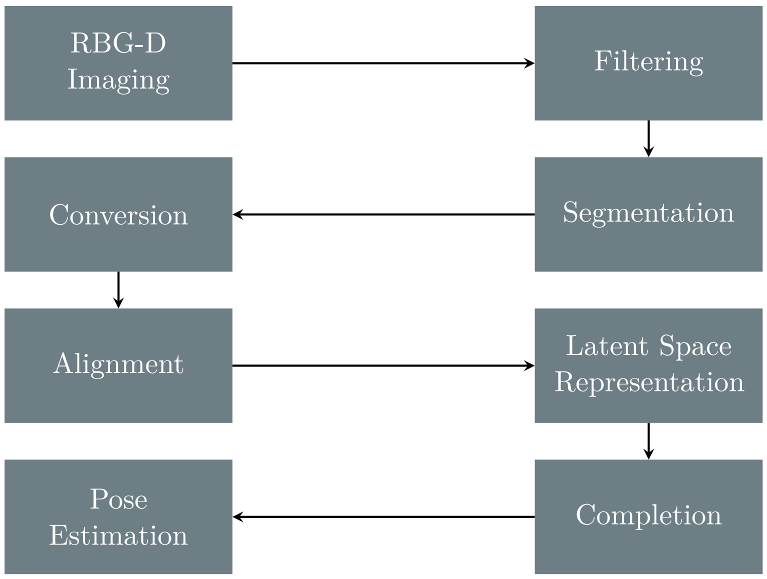


Figure 4.25: 3D reconstruction end-to-end pipeline from a single-view raw RGB-D image to a complete 3D model with a given pose. The output from one block is the input to the next block.

Imaging and Filtering: In Figure 4.26, the process of imaging and filtering is illustrated. It portrays a real-world scene of some fruit on a flat surface. The Intel RealSense D435 camera, which is used in this work, captures RGB images with resolution 1920×1080 and depth images with resolution 1280×720. It is clear that the color image does not need further processing before being sent to the segmentation network. On the other hand, the depth image clearly needs some post-processing to remove noise. In addition to being used for improving the accuracy of segmentation, the depth information is used for the completion network. A good completion accuracy depends on the quality of the raw depth data.

The post-processing used to generate the right depth image in Figure 4.26 includes a threshold filter, a spatial filter, and a temporal filter. The threshold filter specifies the minimum and maximum distance for depth information to be included in the depth image. In scenes like this one, the objects of interest could be in the range from right next to the camera sensor and up to typically less than 1.5 meters away. Further, the spatial filter applies edge-preserving smoothing of the depth data. The amount of smoothing is determined by the one-dimensional

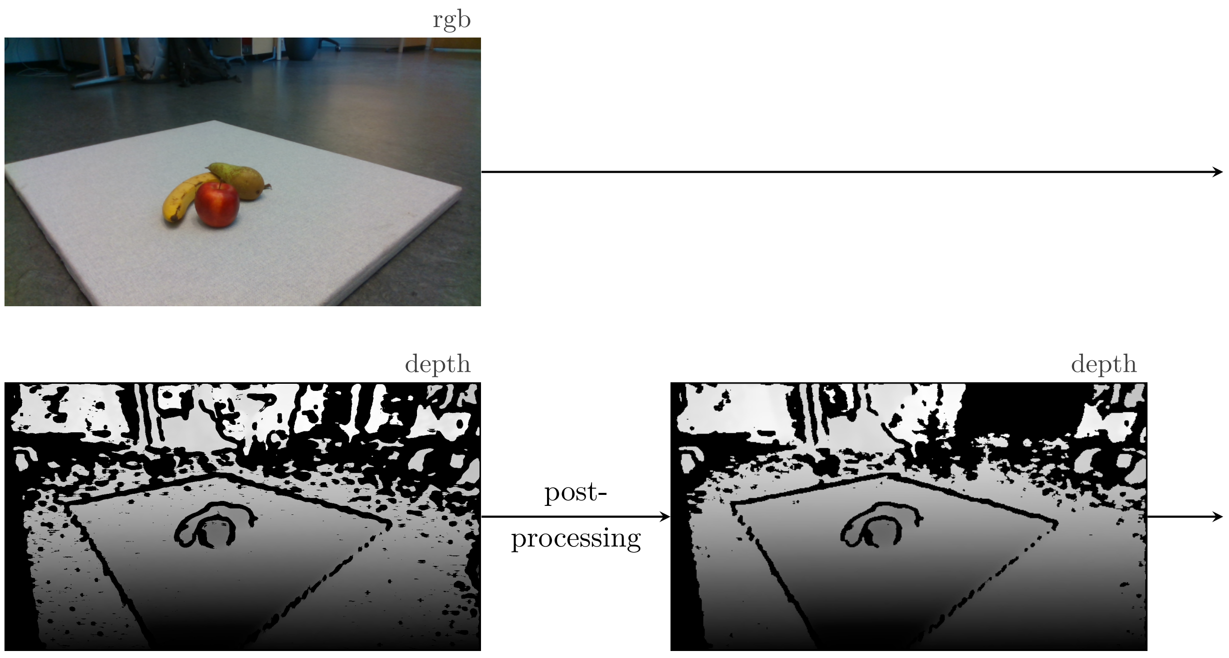


Figure 4.26: RGB and depth images captured with Intel RealSense D435 at its highest density setting. The depth image to the left has been post-processed to give the right image, where some noise has been removed. The low quality of the depth image illustrates a hitherto unmentioned advantage of shape completion; it is not only useful for filling in the parts of the object that can't be seen, but also for upsampling the depth range and filling in missing details of the observed parts of the object. The arrows which point right from the images in the illustration signify that they are propagated as input to the next step in the pipeline.

exponential moving average (EMA) equation, given as

$$S_t = \begin{cases} Y_1 & t = 1 \\ \alpha Y_t \ (1-\alpha) S_{t-1} & t > 1 \text{ and } \mathit{\Delta} = |S_t - S_{t-1}| < \delta_{\text{thresh}} \\ Y_t & t > 1 \text{ and } \mathit{\Delta} = |S_t - S_{t-1}| > \delta_{\text{thresh}} \end{cases} \tag{4.17}$$

where S_t is the value of the EMA at any time period t, Y_t is the newly recorded instantaneous value of disparity or depth, and α represents the degree of weighting decrease (chosen to be $\alpha = 0.5$). δ_{thresh} (chosen to be $\delta_{\text{thresh}} = 8/32$ disparities). Lastly, the temporal filter filters depth data by looking into previous frames, which is a form of time averaging of the depth data. EMA is also used for calculating this filter, where setting $\alpha = 1$ amounts to no filtering, while reducing α will increase averaging and smoothing.

Holes, patches of the depth image where $z = 0$, are still present by design in the post-processed image. These holes commonly result from one or more of the following four reasons:

- *Occlusions:* Shadowing causes the left and the right images to not see the same object.

- *Lack of texture:* Since the stereo matching relies on matching feature descriptors in the left and right images, depth estimation for texture-less surfaces will be challenging.
- *Multiple matches:* This is a case where features in one image can be matched with multiple features in the other image, which could happen if the surface has a uniform periodic structure.
- *No signal:* If the exposure is too high or too low, or the object is too far away there might not be any signal to calculate depth from. It could also occur if the object is closer than the minimum search-range for the stereo vision algorithm.

While an image with no holes is more visually appealing, for the application of shape completion it is better to leave the holes without doing anything as hole filtering might introduce erroneous depth information. Anyhow, the shape completion network will fill in the missing parts of the objects of interest – including any holes they might have. This showcases another use for shape completion, as it can also improve the depth quality of the partial scan.

More effort could be done to add more advanced filters, but the main issue is the quality of the current consumer-grade depth cameras such as Intel RealSense. The reality is that the current models are limited in their ability to correctly infer the depth information of a scene. This is also one of the reasons behind using a simulated environment for evaluating the overall 3D vision pipeline. However, it is clear that the development of better camera hardware is evolving at a high pace. With the field of developing RGB-D camera hardware being highly competitive, the quality of the depth cameras will probably be much better in the near future.

Segmentation and Conversion: A schematic of the process of next couple of steps in the pipeline, segmentation and conversion, is given in Figure 4.27. It uses the color and depth images to segment and isolate the object of interest, for then to convert it to a point cloud.

Converting the depth map to a point cloud is done using Open3d [179]. It requires the intrinsic camera parameters of the camera used to correctly calculate the depth scale of the resulting point cloud. With the UnrealROX engine used to generate the RobotriX dataset (which I used), the field of view (FOV) is specified as 90°, and the parameters can be calculated using the following equations:

$$\begin{aligned} f_x = f_y &= \frac{w}{2} \\ c_x &= \frac{w}{2} \\ c_y &= \frac{h}{2} \end{aligned} \tag{4.18}$$

where w and h are the image width and height, respectively, and f_x and f_y are the x-axis and y-axis focal length. The principal point offset (the location of the principal point relative to the image origin) are given by the pixel locations c_x and c_y. For completeness, in general the focal length should be calculated with

$$f_x = f_y = \frac{w}{2\tan\left(\frac{\text{FOV}}{2}\right)} \tag{4.19}$$

The segmentation network presented in Section 4.2 is used to create the segmentation mask in Figure 4.27 from the color and depth images. This mask is then used to isolate a

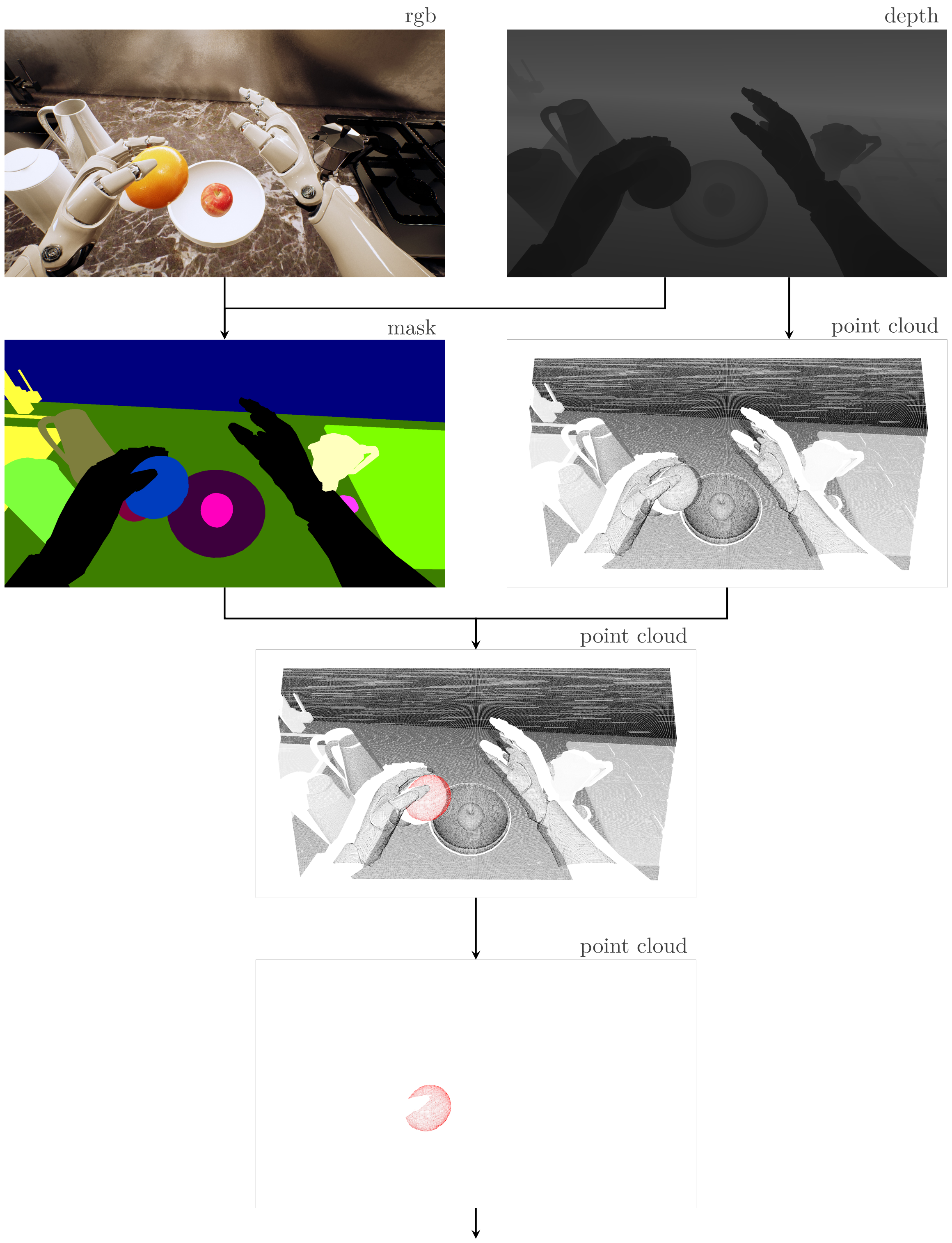


Figure 4.27: A flowchart of the proposed segmentation process. Both the color and depth image are used to find a mask which best segments the objects using the segmentation network. The depth image is also converted into a point cloud, which constitutes the 3D representation of the scene. Each object of interest are identified and isolated by extracting points based on a selected class in the mask. Then each object are passed on to the point cloud registration network for alignment with its reference object in the prior object dataset.

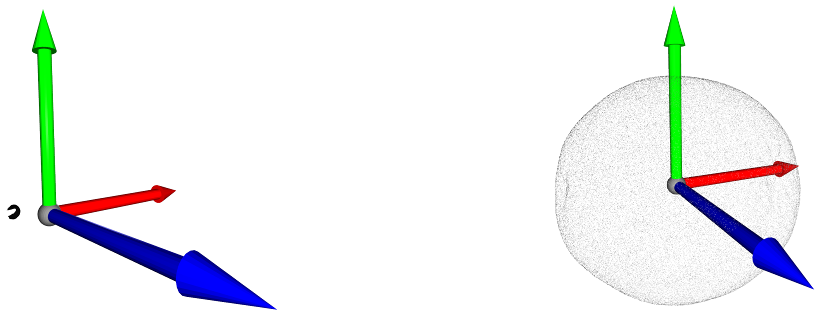
point cloud of the segmented object. Specifically, the whole depth image is converted to a point cloud where the mask is used as the color value of each point. Then, each point is compared to an array with all the possible mask colors (each class has its own color) based on its color value. By following this procedure, a boolean array is achieved where the truthy values indicate the indexes of the points in the point cloud which belong to the segmented object. Based on this array, the object point cloud can be created. The runtime to complete this step is less than 0.2 seconds for images with a resolution of 1920×1080.

Alignment: Continuing from the isolated object point cloud (from now on referred to as the source point cloud), it now needs to be aligned to the world coordinate system in the same way as the objects in the dataset used to train the shape completion network. This process is also called point cloud registration. There are several issues that arise as part of this challenge, which turned out to be one of the toughest problems in this work. As can be seen in Figure 4.28, the source point cloud has a different scale, rotation, and translation than one of its matches in the dataset, which is defined to be in a canonical pose. While the object point cloud acquired from the dataset (from now on referred to as the target point cloud) is of a known size, i.e. it is constrained to be within the unit sphere, the number of unknown poses increases the complexity of finding a good alignment between the two point clouds, especially considering only a partial scan is given of the source point cloud.

As a small digression, consider the input point cloud to be complete, i.e. it has all the missing parts already filled in. This would tremendously simplify the problem, as the scale of the point cloud can then be determined in a few simple steps; find the maximum distance between two points in the point cloud, and globally scale the point cloud such that this distance would be 2 (twice the unit length). By subtracting the mean of the point cloud, it is now centered within the unit sphere. Consequently, the scale is the same as the target point cloud and the approach given in Section 4.4, which correctly identifies the needed rotation and translation transformation to align the input point cloud with the target point cloud. However, this approach to find the correct scale will fail when the input point cloud is a partial scan. The maximum distance between two points in a partial scan will be less than that of the maximum distance of the complete object, resulting in finding a scale factor which is too large.

These difficulties are addressed in Section 4.4, and the approach mentioned in that section is used in this step of the pipeline. The end result from this step is a scale, rotation, and translation transformation of the partial point cloud. The transform is saved until the shape completion is done, as then the inverse transform will be applied to the completed point cloud to place it in the correct pose in the scene.

Completion: With the partial object scan aligned to its corresponding canonical object in the dataset, the shape completion network presented in Section 4.3 can infer its full shape. Of course, it would be possible to simply use the canonical object for the shape completion as well, but the network may be able to make the resulting object geometry a better fit by using the observed data. Since the output from the completion network is an implicit function, it is first converted to a mesh with marching cubes and then to a point cloud using the Poisson disk sampling method. By applying the inverse transform of the previous alignment of the partial scan to the complete object point cloud, the object is placed back into its correct size and pose in the scene.



(a) Unaligned Partial Scan **(b)** Aligned Ground-truth

Figure 4.28: Left: Segmented partial point cloud scan (small black disk) plotted with an axis cross with axes equal to 1. **Right:** Complete point cloud from the dataset, which is scaled to be within the unit sphere. The partial point cloud has to be aligned to some part of the complete scan for the completion network to be able to infer its full geometry.

An extension to this part of the pipeline could be to calculate a weighting average of the prior dataset object and the resulting object from the shape completion network. By calculating the distances between the partial scans and the completed geometries using one of the point metrics described in Section 2.1.3, one could determine whether the optimal completion is the one from the shape completion network or the corresponding dataset object, or a combination of the two. The combination of multiple point clouds in this way is done by simply appending the points from one cloud to the other.

3D object pose: Identifying the pose of the object is out of the scope for this work, but it is added to the pipeline schematic in Figure 4.25 as it is considered the final step for a robotic vision system to have a complete understanding of the scene. Additionally, with the full geometry of the object, which is also segmented from the rest of the scene, this becomes almost a trivial task compared to the current approaches for pose estimation.

CHAPTER 5

Conclusion

5.1 Summary

In this chapter the essential points of this thesis are summarized. Based on the motivation – to complete the 3D geometry of objects in a scene based on a partial scan – the main aim was to explore and validate the feasibility of using deep learning methods to utilize prior knowledge from a dataset to perform shape completion. A custom dataset and three deep-learning based methods were introduced: one for semantic segmentation of RGB-D images, one for shape completion of point clouds, and one for point cloud registration. Furthermore, a 3D vision pipeline architecture was proposed to consolidate the various subtasks into a coherent whole.

There are three main limitations with current active vision and multi-view methods which can be overcome by employing a single-view agent, and thus represent a motivation for such an approach. First, current multi-view and active vision methods may take as much as 10 seconds to infer a complete 3D model of the object of interest for grasping. In an industrial setting, this would be a severe bottleneck in the overall process. Moreover, for some instances, the objects that should be picked by the robot might be moving on a conveyor belt and pass by it in that time-frame. Second, most objects are partially covered in some way from the view of a camera. A single object will always cover other parts of the same object, in what is called natural occlusion. Furthermore, physical occlusions such as other objects or physical obstructions often hinder the full object from being visible. Third, if the object of interest is too far away from the robot agent, the robot can only move such that it sees the object from approximately the same view. Hence, it is not possible to use multi-view methods to infer the geometry of the object.
Based on these limitations, in this MSc thesis the focus has been to develop and implement a pipeline based on deep learning approaches to achieve a robust and accurate perception system. This work includes the following:

- **Semantic Segmentation Network:** In Section 4.2 a deep learning model with a ResNet-like architecture was used to efficiently and accurately predict an output for each pixel in the RGB-D image of a scene, resulting in a mask with semantic classification.
- **Shape Completion Network:** The most important method was covered in Section 4.3, where a shape completion network which uses continuous implicit functions to define a complete shape for a partial observation was proposed.
- **Point Cloud Registration Network:** Section 4.4 presented an approach to align the resulting isolated objects from the partial scan into their canonical pose corresponding to similar objects in the prior dataset.
- **3D Vision Pipeline:** Section 4.5 ties the aforementioned methods together into the overall 3D vision pipeline.

In connection to the outlined research questions, and based on the research investigations carried out and presented in this MSc thesis it can also be concluded that real-world shape completion is feasible (R1). The partial scan should be segmented with the object of interest carefully isolated (R2), such that only the relevant information is used in the shape completion stage. Since point clouds are the data representation that most closely follows that of the output from the depth sensors, it is reasonable to use this data representation when comparing 3D objects. It also turns out that deep learning models cannot only be pipelined (R3), but that this actually improves results and reduces noise compared to a one-for-all approach. Although it is in its early stages, the work presented here provides a framework to be built further upon in future work.

The key features of the resulting pipeline implemented in this MSc thesis are provided below:

- **Adaptability:** The perception system is highly adaptable. It will produce reasonable outputs for any scene, as long as the objects to be completed are in the prior dataset (or similar objects) used to train the agent.
- **Online Processing:** The pipeline performs real-time completion of objects, as the inference time is about a second. This removes lag and other delays in the robot system, effectively addressing the main bottleneck in current processes.
- **Compositional Visual Intelligence:** The classification, segmentation and completion of objects is inherently tasks that add to the ”intelligence” of the robot. These are tasks that are usually tagged as human-only, and especially the combination of them brings the robot's understanding of a scene closer to that of humans – increasing its possibilities to do general tasks such as grasping and manipulation of objects.
- **Component-based System:** The components in the 3D vision pipeline can be exchanged independently of one another. For instance, the segmentation method could be exchanged with a future improved version without affecting the remainder of the pipeline. This enables continuous improvement, as each component in the pipeline can be updated with the future state-of-the art for the specific task that component performs.
- **Simulation-based Learning:** All the data, both for the objects and the scenes, are acquired from digital models and simulations. The learning does not seem to be limited by not using real-world data, which leads to some opportunities by using such a simulated-to-real approach: First, acquiring a sizeable dataset is simplified and takes a fraction of the time. Second, scenes and objects can be digitally represented even though they are not readily available in the real world, and which would otherwise be difficult to image and create a dataset of. This also reduces the economical cost of compiling datasets.

5.2 Future Work

This section outlines several ideas which were out of the scope for this thesis, but which are worth to be investigated in the future and might improve and expand the techniques and approaches used in this work.

- **Prepare for Production-Readiness:** For using this perception system in the industry, which is important for the commercialization of the method, optimization of the production code is of vital importance. Further work should be put on identifying and removing potential bugs and errors in the code. The deep learning models could make use of PyTorch's just-in-time (JIT) compiler, to be usable in most software and hardware environments, including those of robotic facilities. A general vision system such as this work presents is an enabling factor in the future multi-billion dollar industry of robotic automation. In that regard, it could be of interest to look into patenting the approach presented in this thesis. In practice, however, it possibly builds too much upon related open-source work for this to be feasible.

- **Use Sequence of Images:** A robot with an eye-in-hand camera configuration performing manipulation would mostly be moving, and capturing a video-stream at all times. This calls for a merged approach between multi-view and single-view approaches, where the sequence of images could be stitched together and continuously updated by the multi-view system, while the single-view agent performs inference and shape completion on a more detailed partial scan of the scene. Additionally, an image sequence allows for models such as recurrent neural networks (which use their internal state to process sequences of inputs) where the next inference could also take into account previous inferences. Deep reinforcement learning could also be added to the pipeline in the same fashion.

- **Solve the Next-Best-View Problem:** Based on the current observation of a scene, information could be gained to determine the next optimal sensor pose that maximizes the reconstructed surface of the object of interest. In the literature, this is referred to as the next-best-view problem. A recent effort by Mendoza et al. [95] introduced supervised learning as a means to find the next-best-view, but it still remains an open problem.

- **Reduce Inference Time:** Some parts of the methods in the pipeline could be further optimized to further reduce the overall inference time. For example, there is a potential of adding modifications marching cubes algorithm used to render the result from the completion network.

- **Verify the Inner Workings of the Shape Completion Network:** The neural networks used as part of the 3D vision pipeline, and particularly the shape completion network, should be further studied. Do they perform the task they are supposed to do, or do they only *seem* to do the task? Insight could be gained by interpolation in the latent space, testing on specific objects with certain geometries, and comparing with other neural networks based on different concepts than the ones used.

- **Increase the Complexity of the Neural Networks:** In the design of the network architectures used in the pipeline, performance and accuracy improvements could be gained by using more intricate convolutional operators, such as the ones included in PyTorch Geometric [36]. It has compiled implementations of convolutional operators specifically designed for graph representations such as meshes and point clouds. Moreover, it could be of interest to design a Bayesian neural network using a framework such as Pyro [10]. Especially for the shape completion agent, this allows the pipeline to have the concept of being unsure of what the object to be completed is. This could occur either in cases where the object was not in the prior dataset used for training the agent, or in cases where the view is from an angle where the object class is ambiguous.

- **Improve the Simulation Environment:** While the simulation environment provided by UnrealROX [93] renders ultra-realistic scenes, it is limited by the design of the models. Further work could be devoted to recreating the work environment of the robots at SINTEF etc. in this tool, which then ends up being a digital twin. In this manner, one can make a vast dataset with relatively few man-hours. Moreover, the physical properties of the robot and the rest of the scene could be represented, allowing for realistic manipulation of objects in the simulation environment. Today, some problems are potentially only solvable by big tech companies who source data from their users. Simulation could, however, possibly make the same research possible with comparatively limited computational resources.

- **Extend the Model Flexibility to Account for Compliant Objects** Most organic objects are deformable, and a robotic manipulation agent which assumes that objects are rigid might fail to grasp these. This necessitates a 3D representation which addresses the properties of compliant objects, as the data representations presented in Section 2.1.2 cannot handle such characteristics. In terms of the shape completion agent, it could also predict the ”skeleton” of the object in addition to its complete geometry. Moreover, it could be useful to infer whether an object is hard/soft, dry/wet, coarse/sticky, etc., as these properties affect the optimal grasping policy.

APPENDIX A
Tools

This appendix includes an overview on the used hardware and software environment as part of the work in this thesis.

A.1 Hardware Environment

- *Intel Realsense D435* is an RGB-D camera which uses stereo vision to calculate depth. It has an infrared projector which projects a non-visible static infrared pattern to improve depth accuracy in scenes with low texture. In this work, this camera was used to capture real-world color and depth images of a scene.
- *Nvidia Geforce GTX 1080* is a GPU which can be used for faster training and inferencing of neural networks than what is possible with CPUs. In this work, the GPU was used to enable parallelization during training of the neural networks.
- *Franka Emika Panda 7 DOF* is a robot manipulator that can have a payload up to 3 kg with a maximum reach of 855mm. In this work, this manipulator was only used as a reference to see the target system for the developed perception framework.

A.2 Software Environment

A.2.1 Python Libraries

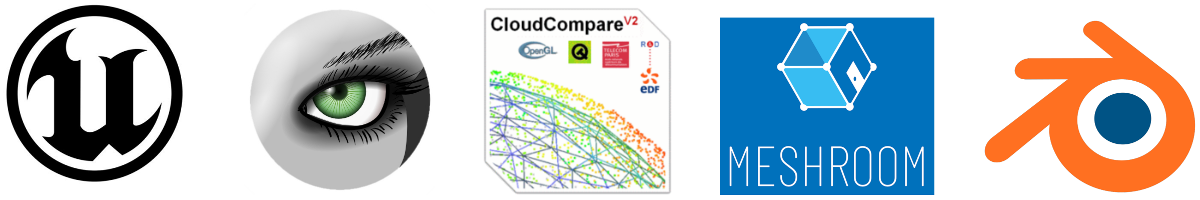
(a) PyTorch **(b)** Open3D **(c)** Trimesh **(d)** Scikit-learn

Figure A.1: Python libraries.

- *PyTorch* [114] is an open source deep learning platform that proces a seamless path from research prototyping to production deployment. It is a Facebook-backed Python-first library, with a rich ecosystem of tools and libraries which extend the core functionality. It is built upon the Lua-framework Torch, but is now Python-first and backed by Facebook. In this work, PyTorch was used in the implementation of all neural networks. While many alternatives exist, PyTorch was chosen in this work for its dynamic graph-building, simple API, and NumPy-like syntax. Dynamic graph-building allows the programmer to prototype a novel network as it is being built, which simplifies debugging. Furthermore, since its syntax is similar to that of NumPy, it is a low barrier for Python-developers to use the framework and its functionality can easily be combined with other libraries.

- *Open3D* [179] is a 3D data manipulation library with a built-in visualization tool. It also has support for basic 3D data processing algorithms. In this work, Open3D was used to visualize point clouds in the deep learning pipeline. It was also used to convert RGB-D images into point clouds, and to NumPy arrays for further manipulation. Disclosure: I am a contributor to this library.
- *Trimesh* [29] is a library for loading and using triangular meshes, and allows for easy manipulation and analysis. In this work, Trimesh was used to convert vertices and faces into meshes.
- *Scikit-Learn* [117] is a machine learning library designed to interoperate with the libraries NumPy and SciPy. In this work, Scikit-Learn was used for all non-deep machine learning algorithms.

A.2.2 3D Computer Graphics Software



(a) UnrealROX **(b)** Meshlab **(c)** CloudCompare **(d)** Meshroom **(e)** Blender

Figure A.2: 3D modeling and processing software.

- *UnrealROX* [93] is an environment built over Unreal Engine 4 which aims to reduce the reality gap by leveraging hyperrealistic indoor scenes that are explored by robot agents which also interact with objects in a visually realistic manner in that simulated world. In this work, the RobotriX dataset [39] was used, which is automatically generated by UnrealROX.
- *Meshlab* [21] is a software for processing and editing 3D triangular meshes, and provides a set of tools for editing, cleaning, healing, rendering, texturing and converting meshes. In this work, Meshlab was used to decimate/subsample meshes from the custom compiled dataset, which unedited have up to 3 million faces each.
- *CloudCompare* [22] is a 3D point cloud and mesh processing software. In this work, CloudCompare was used for working with 3D models with the .off format, and was also used for converting a mesh to a point cloud while keeping the color information for each point.

- *Meshroom* [97] is a photogrammetry software, i.e. it can infer the geometry of a scene from a set of unordered photographs or videos. In this work, an effort was made to use Meshroom to generate a dataset based on real-world scans of food objects. While Meshroom can generally reconstruct 3D geometry from multiple images (20 per scan) with a high accuracy, it was not able to give a satisfactory result for small objects such as fruit and vegetables.

- *Blender* [23] is a 3D creation suite which supports modeling, rigging, simulation, rendering, compositing, etc. In this work, Blender was used to generate specific still-life scenes, such as Figure 1.1.

Bibliography

[1] Panos Achlioptas et al. "Learning Representations and Generative Models for 3D Point Clouds". In: *arXiv e-prints* (July 2017), arXiv:1707.02392. arXiv: `1707.02392 [cs.CV]` (cited on pages 18, 49, 65).

[2] Yasuhiro Aoki et al. "PointNetLK: Robust & Efficient Point Cloud Registration using PointNet". In: *CoRR* abs/1903.0 (2019) (cited on page 43).

[3] Jakob Andreas Bærentzen and Henrik Aanæs. "Generating Signed Distance Fields From Triangle Meshes". In: (cited on page 57).

[4] Gill Barequet and Micha Sharir. "Filling gaps in the boundary of a polyhedron". In: *Computer Aided Geometric Design* 12.2 (March 1995), pages 207–229. DOI: `10.1016/0167-8396(94)00011-g`. URL: `https://doi.org/10.1016/0167-8396(94)00011-g` (cited on page 44).

[5] Jonathan T Barron and Jitendra Malik. "Shape, Illumination, and Reflectance from Shading Supplementary Material". In: 2013 (cited on page 39).

[6] Thabo Beeler et al. "Improved Reconstruction of Deforming Surfaces by Cancelling Ambient Occlusion". In: *ECCV*. 2012 (cited on page 38).

[7] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013), pages 1798–1828 (cited on pages 30, 31).

[8] D P Bertsekas. "A distributed asynchronous relaxation algorithm for the assignment problem". In: *1985 24th IEEE Conference on Decision and Control*. December 1985, pages 1703–1704. DOI: `10.1109/CDC.1985.268826` (cited on page 18).

[9] Paul J Besl and Neil D McKay. "A Method for Registration of 3-D Shapes". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14 (1992), pages 239–256 (cited on page 43).

[10] Eli Bingham et al. "Pyro: Deep Universal Probabilistic Programming". In: *arXiv preprint arXiv:1810.09538* (2018) (cited on page 87).

[11] Volker Blanz and Thomas Vetter. "A morphable model for the synthesis of 3D faces". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - {SIGGRAPH} {\textquotesingle}99*. {ACM} Press, 1999. DOI: `10.1145/311535.311556`. URL: `https://doi.org/10.1145/311535.311556` (cited on page 47).

[12] Bert De Brabandere, Davy Neven, and Luc Van Gool. "Semantic Instance Segmentation with a Discriminative Loss Function". In: *CoRR* abs/1708.0 (2017) (cited on page 41).

[13] Angel Xuan Chang et al. "ShapeNet: An Information-Rich 3D Model Repository". In: *CoRR* abs/1512.0 (2015) (cited on pages 8, 53).

[14] Ding-Yun Chen et al. “On Visual Similarity Based 3D Model Retrieval”. In: *Computer Graphics Forum* 22.3 (2003), pages 223–232. DOI: `10.1111/1467-8659.00669`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00669` (cited on page 20).

[15] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. “Scalable real-time volumetric surface reconstruction”. In: *ACM Trans. Graph.* 32 (2013), 113:1–113:16 (cited on page 39).

[16] Liang-Chieh Chen et al. “MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 4013–4022 (cited on page 40).

[17] Wenyu Chen et al. “Polynomial curve registration for matching point clouds of different scales”. In: *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)* (2017), pages 122–126 (cited on page 44).

[18] Wenyu Chen et al. “Scale registration based on descriptor analysis and B-spline matching”. In: *TENCON 2017 - 2017 IEEE Region 10 Conference* (2017), pages 1451–1456 (cited on page 44).

[19] Zhiqin Chen and Hao Zhang. “Learning Implicit Fields for Generative Shape Modeling”. In: *CoRR* abs/1812.0 (2018) (cited on page 50).

[20] Christopher Bongsoo Choy et al. “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction”. In: *ECCV*. 2016 (cited on page 48).

[21] Paolo Cignoni et al. “MeshLab: an Open-Source Mesh Processing Tool”. In: *Eurographics Italian Chapter Conference*. Edited by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5. DOI: `10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136` (cited on pages 55, 90).

[22] “CloudCompare [Computer software]”. In: URL: `https://github.com/cloudcompare/cloudcompare` (cited on page 90).

[23] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: `http://www.blender.org` (cited on page 91).

[24] Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. “Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes”. In: *IEEE Transaction on Visualization and Computer Graphics* 18.6 (2012), pages 914–924. URL: `http://vcg.isti.cnr.it/Publications/2012/CCS12` (cited on page 16).

[25] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. “Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 6545–6554 (cited on page 48).

[26] Angela Dai et al. “BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration”. In: *ACM Trans. Graph.* 36 (2017), 24:1–24:18 (cited on page 39).

[27] Jifeng Dai et al. “Instance-sensitive Fully Convolutional Networks”. In: *ECCV*. 2016 (cited on page 40).

[28] Jifeng Dai et al. “R-FCN: Object Detection via Region-based Fully Convolutional Networks”. In: *NIPS*. 2016 (cited on page 41).

[29] Michael Dawson-Haggerty. “Trimesh [Computer software]”. In: URL: `https://github.com/mikedh/trimesh` (cited on pages 57, 90).

[30] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pages 248–255 (cited on page 20).

[31] James Diebel and Sebastian Thrun. “An Application of Markov Random Fields to Range Sensing”. In: *NIPS*. 2005 (cited on page 38).

[32] Haoqiang Fan, Hao Su, and Leonidas Guibas. “A Point Set Generation Network for 3D Object Reconstruction from a Single Image”. In: *arXiv e-prints* (December 2016), arXiv:1612.00603. arXiv: `1612.00603 [cs.CV]` (cited on page 18).

[33] Haoqiang Fan, Hao Su, and Leonidas J Guibas. “A Point Set Generation Network for 3D Object Reconstruction from a Single Image”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 2463–2471 (cited on page 49).

[34] Alireza Fathi et al. “Semantic Instance Segmentation via Deep Metric Learning”. In: *CoRR* abs/1703.1 (2017) (cited on page 41).

[35] Yutong Feng et al. “MeshNet: Mesh Neural Network for 3D Shape Representation”. In: *AAAI 2019* (2018) (cited on page 49).

[36] Matthias Fey and Jan E Lenssen. “Fast Graph Representation Learning with {PyTorch Geometric}”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019 (cited on page 87).

[37] David A Forsyth. “Object Detection with Discriminatively Trained Part-Based Models”. In: *IEEE Computer* 47 (2014), pages 6–7 (cited on page 47).

[38] Alberto Garcia-Garcia et al. “A Review on Deep Learning Techniques Applied to Semantic Segmentation”. In: *CoRR* abs/1704.0 (2017) (cited on page 39).

[39] Alberto Garcia-Garcia et al. “The RobotriX: An Extremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pages 6790–6797 (cited on pages 58, 59, 90).

[40] Natasha Gelfand et al. “Robust Global Registration”. In: *Symposium on Geometry Processing*. 2005 (cited on page 43).

[41] Jared Glover, Radu Bogdan Rusu, and Gary R Bradski. “Monte Carlo Pose Estimation with Quaternion Kernels and the Bingham Distribution”. In: *Robotics: Science and Systems*. 2011 (cited on page 43).

[42] Ian J Goodfellow et al. “Generative Adversarial Nets”. In: *NIPS*. 2014 (cited on page 35).

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cited on pages 20, 28, 29).

[44] Paul Guerrero et al. “PCPNet Learning Local Shape Properties from Raw Point Clouds”. In: *Comput. Graph. Forum* 37 (2018), pages 75–85 (cited on page 39).

[45] Saurabh Gupta et al. “Aligning 3D models to RGB-D images of cluttered scenes”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pages 4731–4740 (cited on page 47).

[46] Saurabh Gupta et al. “Learning Rich Features from RGB-D Images for Object Detection and Segmentation”. In: *ECCV*. 2014 (cited on page 42).

[47] Feng Han and Song-Chun Zhu. “Bottom-Up/Top-Down Image Parsing with Attribute Grammar”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2009), pages 59–73 (cited on page 47).

[48] Lei Han and Lu Fang. “FlashFusion: Real-time Globally Consistent Dense 3D Reconstruction using CPU Computing”. In: *Robotics: Science and Systems*. 2018 (cited on page 39).

[49] Yudeog Han, Joon-Young Lee, and In-So Kweon. “High Quality Shape from a Single RGB-D Image under Uncalibrated Natural Illumination”. In: *2013 IEEE International Conference on Computer Vision* (2013), pages 1617–1624 (cited on page 38).

[50] Christian Häne, Shubham Tulsiani, and Jitendra Malik. “Hierarchical Surface Prediction for 3D Object Reconstruction”. In: *2017 International Conference on 3D Vision (3DV)* (2017), pages 412–420 (cited on page 48).

[51] Adam W Harley, Konstantinos G Derpanis, and Iasonas Kokkinos. “Segmentation-Aware Convolutional Networks Using Local Attention Masks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pages 5048–5057 (cited on page 41).

[52] Caner Hazirbas et al. “FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture”. In: *ACCV*. 2016 (cited on page 42).

[53] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pages 770–778 (cited on pages 50, 59).

[54] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pages 1026–1034 (cited on page 4).

[55] Kaiming He et al. “Mask R-CNN.” In: *IEEE transactions on pattern analysis and machine intelligence* (2018) (cited on pages 40–42).

[56] Hagit Hel-Or, Shmuel Peleg, and David Avnir. “Symmetry as a Continuous Feature”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 17 (1995), pages 1154–1166 (cited on page 46).

[57] Pedro Hermosilla, Tobias Ritschel, and Timo Ropinski. “Total Denoising: Unsupervised Learning of 3D Point Cloud Cleaning”. In: *CoRR* abs/1904.0 (2019) (cited on page 39).

[58] Matanya B Horowitz, Nikolai Matni, and Joel W Burdick. “Convex relaxations of SE(2) and SE(3) for visual pose estimation”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), pages 1148–1154 (cited on page 43).

[59] Zhaojin Huang et al. “Mask Scoring R-CNN”. In: *CoRR* abs/1903.0 (2019) (cited on page 41).

[60] IProcess. *iProcessProject, WP3*. 2018 (cited on page 4).

[61] U J Isachsen, T Theoharis, and E Misimi. “GPU Accelerated 3D Registration - Evaluation of 3D Registration Algorithms for Robotic Scanning of Compliant Objects”. In: *ICRA* (2018) (cited on page 4).

[62] Gregory Izatt and Russ Tedrake. “Globally Optimal Object Pose Estimation in Point Clouds with Mixed-Integer Programming”. In: 2017 (cited on page 43).

[63] Jindong Jiang et al. “RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation”. In: *CoRR* abs/1806.0 (2018) (cited on pages 42, 59).

[64] Long Jin, Zeyu Chen, and Zhuowen Tu. “Object Detection Free Instance Segmentation With Labeling Transformations”. In: *CoRR* abs/1611.0 (2016) (cited on page 41).

[65] Tao Ju. “Robust repair of polygonal models”. In: *{ACM} {SIGGRAPH} 2004 Papers on - {SIGGRAPH} {\textquotesingle}04*. {ACM} Press, 2004. DOI: `10.1145/1186562.1015815`. URL: `https://doi.org/10.1145/1186562.1015815` (cited on page 44).

[66] Evangelos Kalogerakis et al. “A probabilistic model for component-based shape synthesis”. In: *ACM Trans. Graph.* 31 (2012), 55:1–55:11 (cited on page 47).

[67] Angjoo Kanazawa et al. “End-to-End Recovery of Human Shape and Pose”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 7122–7131 (cited on page 49).

[68] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. *Poisson Surface Reconstruction*. 2006. DOI: `10.2312/sgp/sgp06/061-070`. URL: `http://diglib.eg.org/handle/10.2312/SGP.SGP06.061-070` (cited on pages 45, 46).

[69] Michael Kazhdan and Hugues Hoppe. “Screened poisson surface reconstruction”. In: *{ACM} Transactions on Graphics* 32.3 (June 2013), pages 1–13. DOI: `10.1145/2487228.2487237`. URL: `https://doi.org/10.1145/2487228.2487237` (cited on page 45).

[70] Vladimir G Kim et al. “Learning part-based templates from large collections of 3D shapes”. In: *ACM Trans. Graph.* 32 (2013), 70:1–70:12 (cited on page 47).

[71] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *CoRR* abs/1312.6 (2014) (cited on page 33).

[72] Alexander Kirillov et al. “InstanceCut: From Edges to Instances with MultiCut”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 7322–7331 (cited on page 41).

[73] Reinhard Klette. *Concise Computer Vision - An Introduction into Theory and Algorithms*. 2014. ISBN: 978-1-4471-6319-0. DOI: `10.1007/978-1-4471-6320-6` (cited on pages 9, 11, 12, 14).

[74] Sebastian Koch et al. “ABC: A Big CAD Model Dataset For Geometric Deep Learning”. In: *CoRR* abs/1812.0 (2018) (cited on page 53).

[75] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. “Using Locally Corresponding CAD Models for Dense 3D Reconstructions from a Single Image”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 5603–5611 (cited on page 49).

[76] Zhaoming Kong and Xiaowei Yang. “A Brief Review of Real-World Color Image Denoising”. In: *CoRR* abs/1809.0 (2018) (cited on page 38).

[77] Sondre Bø Kongsgård. “Learning Latent Space of Compliant Objects with Deep Learning for Robotic Grasping”. In: 2018 (cited on pages 64, 65).

[78] Johannes Kopf et al. “Joint bilateral upsampling”. In: *ACM Trans. Graph.* 26 (2007), page 96 (cited on page 38).

[79] Scott Krig. *Computer Vision Metrics: Survey, Taxonomy, and Analysis.* 1st edition. Apress, 2014. ISBN: 978-1-4302-5929-9,978-1-4302-5930-5 (cited on pages 9, 10).

[80] Yann LeCun. “Gradient-based learning applied to document recognition”. In: 1998 (cited on page 32).

[81] Jan Eric Lenssen, Christian Osendorfer, and Jonathan Masci. “Differentiable Iterative Surface Normal Estimation”. In: *CoRR* abs/1904.0 (2019) (cited on page 39).

[82] Thomas Lewiner et al. “Efficient Implementation of Marching Cubes’ Cases with Topological Guarantees”. In: *J. Graphics, GPU, & Game Tools* 8 (2003), pages 1–15 (cited on page 17).

[83] Guo Li et al. “Analysis, reconstruction and manipulation using arterial snakes”. In: *ACM Trans. Graph.* 29 (2010), 152:1–152:10 (cited on page 47).

[84] Yangyan Li et al. “Database-Assisted Object Retrieval for Real-Time 3D Reconstruction”. In: *Comput. Graph. Forum* 34 (2015), pages 435–446 (cited on page 47).

[85] Yijun Li et al. “Deep Joint Image Filtering”. In: *European Conference on Computer Vision.* 2016 (cited on page 39).

[86] Yi Li et al. “Fully Convolutional Instance-Aware Semantic Segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 4438–4446 (cited on page 40).

[87] Xiaodan Liang et al. “Proposal-Free Network for Instance-Level Object Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (2017), pages 2978–2991 (cited on page 41).

[88] Peter Liepa. “Filling Holes in Meshes”. In: *Symposium on Geometry Processing.* 2003 (cited on page 44).

[89] Baowei Lin et al. “Scale ratio ICP for 3D point clouds with different scales”. In: *2013 IEEE International Conference on Image Processing* (2013), pages 2217–2221 (cited on page 44).

[90] Marvin Lindner, Andreas Kolb, and Klaus Hartmann. “Data-Fusion of PMD-Based Distance-Information and High-Resolution RGB-Images”. In: *2007 International Symposium on Signals, Circuits and Systems* 1 (2007), pages 1–4 (cited on page 38).

[91] William E Lorensen and Harvey E Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *{ACM} {SIGGRAPH} Computer Graphics* 21.4 (August 1987), pages 163–169. DOI: `10.1145/37402.37422`. URL: `https://doi.org/10.1145/37402.37422` (cited on pages 16, 44).

[92] Haggai Maron et al. “Point registration via efficient convex relaxation”. In: *ACM Trans. Graph.* 35 (2016), 73:1–73:12 (cited on page 43).

[93] Pablo Martinez-Gonzalez et al. “{UnrealROX}: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation”. In: *ArXiv e-prints* (2018) (cited on pages 58, 59, 88, 90).

[94] Andelo Martinovic and Luc Van Gool. “Bayesian Grammar Learning for Inverse Procedural Modeling”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition* (2013), pages 201–208 (cited on page 47).

[95] Miguel Mendoza et al. “Supervised Learning of the Next-Best-View for 3D Object Reconstruction”. In: *ArXiv* abs/1905.0 (2019) (cited on page 87).

[96] Lars M Mescheder et al. “Occupancy Networks: Learning 3D Reconstruction in Function Space”. In: *CoRR* abs/1812.0 (2018) (cited on pages 50, 70).

[97] “Meshroom [Computer software]”. In: URL: `https://github.com/alicevision/meshroom` (cited on pages 55, 91).

[98] Mateusz Michalkiewicz et al. “Deep Level Sets: Implicit Surface Representations for 3D Shape Inference”. In: *CoRR* abs/1901.0 (2019) (cited on page 50).

[99] E Misimi. “Active Vision from Dense Point Clouds and Visual Servoing”. In: (2018) (cited on page 3).

[100] E Misimi et al. “Robotic Handling of Compliant Food Objects by Robust Learning from Demonstration”. In: *IROS Madrid* (2018) (cited on page 4).

[101] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. “Partial and approximate symmetry detection for 3D geometry”. In: *{ACM} {SIGGRAPH} 2006 Papers on - {SIGGRAPH} {\textquotesingle}06.* {ACM} Press, 2006. DOI: `10.1145/1179352.1141924`. URL: `https://doi.org/10.1145/1179352.1141924` (cited on page 46).

[102] Niloy Jyoti Mitra et al. “Symmetry in 3D Geometry: Extraction and Applications”. In: *Eurographics.* 2012 (cited on page 46).

[103] Liangliang Nan, Ke Xie, and Andrei Sharf. “A search-classify approach for cluttered indoor scene understanding”. In: *ACM Trans. Graph.* 31 (2012), 137:1–137:10 (cited on page 47).

[104] Andrew Nealen et al. “Laplacian mesh optimization”. In: *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia - {GRAPHITE} {\textquotesingle}06.* {ACM} Press, 2006. DOI: `10.1145/1174429.1174494`. URL: `https://doi.org/10.1145/1174429.1174494` (cited on pages 44, 45).

[105] Richard A Newcombe et al. “KinectFusion: Real-time dense surface mapping and tracking”. In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality* (2011), pages 127–136 (cited on page 39).

[106] Alejandro Newell and Jia Deng. “Associative Embedding: End-to-End Learning for Joint Detection and Grouping”. In: *NIPS.* 2017 (cited on page 41).

[107] Michael A Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015 (cited on page 25).

[108] Matthias Nießner et al. “Real-time 3D reconstruction at scale using voxel hashing”. In: *ACM Trans. Graph.* 32 (2013), 169:1–169:11 (cited on page 39).

[109] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization”. In: *NIPS.* 2016 (cited on page 36).

[110] T Olsen, B M Ottebø, and E Misimi. "Sim-to-Real transfer learning based on Deep Reinforcement Learning for gripper vector estimation and grasping of semi-compliant objects". In: *ICRA* (2018) (cited on page 4).

[111] Zhijian Ou. "A Review of Learning with Deep Generative Models from perspective of graphical modeling". In: *CoRR* abs/1808.0 (2018) (cited on pages 32, 33).

[112] Jaesik Park et al. "High quality depth map upsampling for 3D-TOF cameras". In: *2011 International Conference on Computer Vision* (2011), pages 1623–1630 (cited on page 38).

[113] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CoRR* abs/1901.0 (2019) (cited on pages 50, 67).

[114] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017 (cited on page 89).

[115] Mark Pauly et al. "Discovering structural regularity in 3D geometry". In: *ACM Trans. Graph.* 27 (2008), 43:1–43:11 (cited on page 46).

[116] Mark Pauly et al. *Example-Based 3D Scan Completion*. 2005. DOI: 10.2312/sgp/sgp05/023-032. URL: http://diglib.eg.org/handle/10.2312/SGP.SGP05.023-032 (cited on page 47).

[117] F Pedregosa et al. "Scikit-learn: Machine Learning in {P}ython". In: *Journal of Machine Learning Research* 12 (2011), pages 2825–2830 (cited on page 90).

[118] Quang-Hieu Pham et al. "JSIS3D: Joint Semantic-Instance Segmentation of 3D Point Clouds with Multi-Task Pointwise Networks and Multi-Value Conditional Random Fields". In: *CoRR* abs/1904.0 (2019) (cited on page 43).

[119] Pedro H O Pinheiro, Ronan Collobert, and Piotr Dollár. "Learning to Segment Object Candidates". In: *NIPS*. 2015 (cited on page 40).

[120] Joshua Podolak et al. "A planar-reflective symmetry transform for 3D shapes". In: *{ACM} {SIGGRAPH} 2006 Papers on - {SIGGRAPH} {\textquotesingle}06*. {ACM} Press, 2006. DOI: 10.1145/1179352.1141923. URL: https://doi.org/10.1145/1179352.1141923 (cited on page 46).

[121] Charles Ruizhongtai Qi et al. "Frustum PointNets for 3D Object Detection from RGB-D Data". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 918–927 (cited on page 43).

[122] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *NIPS*. 2017 (cited on page 49).

[123] Charles Ruizhongtai Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 77–85 (cited on pages 43, 49, 65, 74).

[124] Marie-Julie Rakotosaona et al. "POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds". In: *CoRR* abs/1901.0 (2019) (cited on page 39).

[125] Anurag Ranjan et al. "Generating 3D Faces Using Convolutional Mesh Autoencoders". In: *Computer Vision {\textendash} {ECCV} 2018*. Springer International Publishing, 2018, pages 725–741. DOI: 10.1007/978-3-030-01219-9_43. URL: https://doi.org/10.1007/978-3-030-01219-9%7B%5C_%7D43 (cited on page 49).

[126] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015), pages 1137–1149 (cited on page 40).

[127] Gernot Riegler, Ali O Ulusoy, and Andreas Geiger. “OctNet: Learning Deep 3D Representations at High Resolutions”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 6620–6629 (cited on page 48).

[128] Gernot Riegler et al. “A Deep Primal-Dual Network for Guided Depth Super-Resolution”. In: *CoRR* abs/1607.0 (2016) (cited on page 39).

[129] Gernot Riegler et al. “OctNetFusion: Learning Depth Fusion from Data”. In: *2017 International Conference on 3D Vision (3DV)* (2017), pages 57–66 (cited on page 48).

[130] Jason Rock et al. “Completing 3D object shape from one depth image”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pages 2484–2493 (cited on page 47).

[131] Diego Rodriguez et al. “Transferring Grasping Skills to Novel Instances by Latent Space Non-Rigid Registration”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pages 1–8 (cited on page 47).

[132] David M Rosen et al. “SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group”. In: *I. J. Robotics Res.* 38 (2019) (cited on page 43).

[133] Szymon Rusinkiewicz and Marc Levoy. “Efficient Variants of the ICP Algorithm”. In: *3DIM*. 2001 (cited on page 43).

[134] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115 (2015), pages 211–252 (cited on pages 4, 21).

[135] Steven M Seitz et al. “A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms”. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*. CVPR ’06. Washington, DC, USA: IEEE Computer Society, 2006, pages 519–528. ISBN: 0-7695-2597-0. DOI: `10.1109/CVPR.2006.19`. URL: `http://dx.doi.org/10.1109/CVPR.2006.19` (cited on page 19).

[136] Lin Shao, Ye Tian, and Jeannette Bohg. “ClusterNet: 3D Instance Segmentation in RGB-D Images”. In: 2018 (cited on page 42).

[137] Tianjia Shao et al. “An interactive approach to semantic modeling of indoor scenes with an RGBD camera”. In: *ACM Trans. Graph.* 31 (2012), 136:1–136:11 (cited on page 47).

[138] Chao-Hui Shen et al. “Structure recovery by part assembly”. In: *ACM Trans. Graph.* 31 (2012), 180:1–180:11 (cited on page 47).

[139] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. “PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud”. In: *CoRR* abs/1812.0 (2018) (cited on page 43).

[140] Ivan Sipiran, Robert Gregor, and Tobias Schreck. “Approximate Symmetry Detection in Partial 3D Meshes”. In: *Comput. Graph. Forum* 33 (2014), pages 131–140 (cited on page 46).

[141] Xibin Song, Yuchao Dai, and Xueying Qin. “Deep Depth Super-Resolution : Learning Depth Super-Resolution using Deep Convolutional Neural Network”. In: *ACCV*. 2016 (cited on page 39).

[142] O Sorkine and D Cohen-Or. “Least-squares meshes”. In: *Proceedings Shape Modeling Applications*. IEEE, 2004. DOI: `10.1109/smi.2004.1314506`. URL: `https://doi.org/10.1109/smi.2004.1314506` (cited on pages 44, 45).

[143] David Stutz and Andreas Geiger. “Learning 3D Shape Completion from Laser Scan Data with Weak Supervision”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 1955–1964 (cited on page 48).

[144] Minhyuk Sung et al. “Data-driven structural priors for shape completion”. In: *ACM Trans. Graph.* 34 (2015), 175:1–175:11 (cited on pages 46, 47).

[145] Richard Szeliski. *Computer Vision: Algorithms and Applications*. London New York: Springer, 2011. ISBN: 978-1-84882-934-3 (cited on page 9).

[146] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pages 2107–2115 (cited on page 48).

[147] Maxim Tatarchenko et al. “What Do Single-view 3D Reconstruction Networks Learn?” In: *ArXiv* abs/1905.0 (2019) (cited on page 67).

[148] Martin Thoma. “A Survey of Semantic Segmentation”. In: *CoRR* abs/1602.0 (2016) (cited on page 39).

[149] Sebastian Thrun and Ben Wegbreit. “Shape from symmetry”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1* 2 (2005), 1824–1831 Vol. 2 (cited on page 46).

[150] M Vakalopoulou et al. “{AtlasNet}: Multi-atlas Non-linear Deep Networks for Medical Image Segmentation”. In: *Medical Image Computing and Computer Assisted Intervention {\textendash} {MICCAI} 2018*. Springer International Publishing, 2018, pages 658–666. DOI: `10.1007/978-3-030-00937-3_75`. URL: `https://doi.org/10.1007/978-3-030-00937-3%7B%5C_%7D75` (cited on page 49).

[151] Ashish Vaswani et al. “Attention Is All You Need”. In: *NIPS*. 2017 (cited on page 75).

[152] Nitika Verma, Edmond Boyer, and Jakob J Verbeek. “FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 2598–2606 (cited on page 49).

[153] O Voinov et al. “Perceptual deep depth super-resolution.” In: 2019 (cited on page 39).

[154] Nanyang Wang et al. “Pixel2Mesh: Generating 3D Mesh Models from Single {RGB} Images”. In: *Computer Vision {\textendash} {ECCV} 2018*. Springer International Publishing, 2018, pages 55–71. DOI: `10.1007/978-3-030-01252-6_4`. URL: `https://doi.org/10.1007/978-3-030-01252-6%7B%5C_%7D4` (cited on page 49).

[155] Yue Wang and Justin M Solomon. “Deep Closest Point: Learning Representations for Point Cloud Registration”. In: 2019 (cited on pages 43, 73).

[156] Yue Wang et al. “Dynamic Graph CNN for Learning on Point Clouds”. In: *CoRR* abs/1801.0 (2018) (cited on pages 43, 74).

[157] Jiajun Wu et al. “Learning 3D Shape Priors for Shape Completion and Reconstruction”. In: *European Conference on Computer Vision (ECCV)*. 2018 (cited on page 48).

[158] Jiajun Wu et al. “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling”. In: *Advances in Neural Information Processing Systems*. 2016, pages 82–90 (cited on page 48).

[159] Jiajun Wu et al. “MarrNet: 3D Shape Reconstruction via 2.5D Sketches”. In: *NIPS*. 2017 (cited on page 48).

[160] Zhirong Wu et al. “3D ShapeNets: A deep representation for volumetric shapes”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pages 1912–1920 (cited on pages 8, 48).

[161] Yi Xiao et al. “Joint convolutional neural pyramid for depth map super-resolution”. In: *CoRR* abs/1801.0 (2018) (cited on page 39).

[162] Shi Yan et al. “DDRNet: Depth Map Denoising and Refinement for Consumer Depth Cameras Using Cascaded CNNs”. In: *ECCV*. 2018 (cited on page 39).

[163] Bin Yang et al. “PIXOR: Real-time 3D Object Detection from Point Clouds”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 7652–7660 (cited on page 43).

[164] Bo Yang et al. “3D Object Reconstruction from a Single Depth View with Adversarial Learning”. In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)* (2017), pages 679–688 (cited on page 48).

[165] Bo Yang et al. “Dense 3D Object Reconstruction from a Single Depth View.” In: *IEEE transactions on pattern analysis and machine intelligence* (2018) (cited on pages 48, 49).

[166] Jiaolong Yang, Hongdong Li, and Yunde Jia. “Go-ICP: Solving 3D Registration Efficiently and Globally Optimally”. In: *2013 IEEE International Conference on Computer Vision* (2013), pages 1457–1464 (cited on page 43).

[167] Qingxiong Yang et al. “Spatial-Depth Super Resolution for Range Images”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pages 1–8 (cited on page 38).

[168] Yaoqing Yang et al. “FoldingNet: Point Cloud Auto-Encoder via Deep Grid Deformation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pages 206–215 (cited on page 49).

[169] Lap-Fai Yu et al. “Shading-Based Shape Refinement of RGB-D Images”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition* (2013), pages 1415–1422 (cited on page 38).

[170] Wentao Yuan et al. “PCN: Point Completion Network”. In: *2018 International Conference on 3D Vision (3DV)* (2018), pages 728–737 (cited on pages 49, 65).

[171] Amir Zadeh et al. “Variational Auto-Decoder”. In: *CoRR* abs/1903.0 (2019) (cited on page 34).

[172] Maciej Zamorski et al. “Adversarial Autoencoders for Generating 3D Point Clouds”. In: *CoRR* abs/1811.0 (2018) (cited on page 49).

[173] Andy Zeng et al. “3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pages 199–208 (cited on page 48).

[174] Cheng Zhang et al. “Advances in Variational Inference”. In: *IEEE transactions on pattern analysis and machine intelligence* (2018) (cited on page 33).

[175] Ruo Zhang et al. “Shape from Shading: A Survey”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 21 (1999), pages 690–706 (cited on page 38).

[176] Xiuming Zhang et al. “Learning to Reconstruct Shapes from Unseen Classes”. In: *NeurIPS*. 2018 (cited on page 48).

[177] Wei Zhao, Shuming Gao, and Hongwei Lin. “A Robust Hole-Filling Algorithm for Triangular Mesh”. In: *The Visual Computer*. Volume 23. 2007, page 22. ISBN: 978-1-4244-1579-3. DOI: `10.1109/CADCG.2007.4407836` (cited on page 44).

[178] Zhong-Qiu Zhao et al. “Object Detection with Deep Learning: A Review”. In: *IEEE transactions on neural networks and learning systems* (2018) (cited on page 39).

[179] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “{Open3D}: {A} Modern Library for {3D} Data Processing”. In: *arXiv:1801.09847* (2018) (cited on pages 81, 90).

[180] Jiejie Zhu et al. “Fusion of time-of-flight depth and stereo for high accuracy depth maps”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition* (2008), pages 1–8 (cited on page 38).

[181] M Zollhöfer et al. “State of the Art on 3D Reconstruction with RGB-D Cameras”. In: *Computer Graphics Forum (Eurographics State of the Art Reports 2018)* 37.2 (2018) (cited on page 44).