Master Thesis

Learning Shape Completion from Bounding Boxes with CAD Shape Priors

vorgelegt von

David Stutz
13.09.2017

Erstgutachter: Prof. Dr. Bastian Leibe
Zweitgutachter: Prof. Dr. Andreas Geiger
# Contents

1 Introduction  
   1.1 Contributions .................................................. 3  
   1.2 Outline .................................................................. 3  

2 Problem  
   2.1 Proposed Approach ................................................. 6  

3 Related Work  
   3.1 Shape Completion .................................................. 9  
      3.1.1 Shape Priors ..................................................... 10  
   3.2 3D Deep Learning and Amortized Inference .................... 10  
      3.2.1 3D Deep Learning .............................................. 11  
      3.2.2 Amortized Inference ............................................ 11  

4 Deep Learning ......................................................... 13  
   4.1 Tensors .................................................................. 13  
      4.1.1 Basic Tensor Operations ....................................... 14  
   4.2 Layered Neural Networks .......................................... 15  
   4.3 Convolutional Neural Networks ................................... 16  
      4.3.1 Convolution ....................................................... 17  
      4.3.2 Convolutional Layer ............................................ 17  
      4.3.3 Pooling and Upsampling ....................................... 18  
   4.4 Auto-encoders ....................................................... 19  
   4.5 Training .............................................................. 21  
      4.5.1 Losses ............................................................ 21  
      4.5.2 Stochastic Gradient Descent .................................. 22  
      4.5.3 Weight Initialization .......................................... 24  
      4.5.4 Error Backpropagation ....................................... 24  
      4.5.5 Regularization .................................................. 25  
      4.5.6 Data Augmentation ............................................. 27  
   4.6 Discussion ........................................................... 27  

5 Shape Representation .................................................. 29  
   5.1 Point Clouds ........................................................ 29  
   5.2 Meshes ................................................................ 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Occupancy Grids</td>
<td>31</td>
</tr>
<tr>
<td>5.4 Signed Distance Functions</td>
<td>31</td>
</tr>
<tr>
<td><strong>6 Shape Prior</strong></td>
<td>33</td>
</tr>
<tr>
<td>6.1 Variational Inference</td>
<td>34</td>
</tr>
<tr>
<td>6.2 Gaussian Variational Auto-Encoder</td>
<td>35</td>
</tr>
<tr>
<td>6.2.1 Practical Considerations</td>
<td>36</td>
</tr>
<tr>
<td>6.3 Bernoulli Variational Auto-Encoder</td>
<td>39</td>
</tr>
<tr>
<td>6.3.1 Practical Considerations</td>
<td>41</td>
</tr>
<tr>
<td>6.4 Discussion</td>
<td>42</td>
</tr>
<tr>
<td><strong>7 Shape Inference</strong></td>
<td>43</td>
</tr>
<tr>
<td>7.1 Maximum Likelihood</td>
<td>44</td>
</tr>
<tr>
<td>7.1.1 Bernoulli Maximum Likelihood</td>
<td>44</td>
</tr>
<tr>
<td>7.1.2 Practical Considerations</td>
<td>45</td>
</tr>
<tr>
<td>7.2 Amortized Maximum Likelihood</td>
<td>45</td>
</tr>
<tr>
<td>7.2.1 Bernoulli Amortized Maximum Likelihood</td>
<td>46</td>
</tr>
<tr>
<td>7.2.2 Gaussian Amortized Maximum Likelihood</td>
<td>46</td>
</tr>
<tr>
<td>7.2.3 Practical Considerations</td>
<td>48</td>
</tr>
<tr>
<td>7.3 Extended Variational Auto-Encoder</td>
<td>49</td>
</tr>
<tr>
<td>7.3.1 Practical Considerations</td>
<td>52</td>
</tr>
<tr>
<td>7.4 Discussion</td>
<td>53</td>
</tr>
<tr>
<td><strong>8 Data</strong></td>
<td>55</td>
</tr>
<tr>
<td>8.1 3D Example and ShapeNet</td>
<td>55</td>
</tr>
<tr>
<td>8.1.1 Mesh Pre-Processing and Voxelization</td>
<td>56</td>
</tr>
<tr>
<td>8.1.2 Mesh Filling</td>
<td>58</td>
</tr>
<tr>
<td>8.1.3 Mesh Rendering and Observation Voxelization</td>
<td>58</td>
</tr>
<tr>
<td>8.1.4 Noise</td>
<td>59</td>
</tr>
<tr>
<td>8.1.5 Discussion</td>
<td>60</td>
</tr>
<tr>
<td>8.2 KITTI</td>
<td>61</td>
</tr>
<tr>
<td>8.2.1 Point Cloud Voxelization</td>
<td>61</td>
</tr>
<tr>
<td>8.2.2 Free Space Voxelization</td>
<td>61</td>
</tr>
<tr>
<td>8.2.3 Filtering</td>
<td>62</td>
</tr>
<tr>
<td>8.2.4 Discussion</td>
<td>62</td>
</tr>
<tr>
<td><strong>9 Experiments</strong></td>
<td>63</td>
</tr>
<tr>
<td>9.1 Experimental Setup</td>
<td>63</td>
</tr>
<tr>
<td>9.1.1 Architecture and Training</td>
<td>64</td>
</tr>
<tr>
<td>9.1.2 Evaluation</td>
<td>64</td>
</tr>
<tr>
<td>9.2 3D Example</td>
<td>65</td>
</tr>
<tr>
<td>9.2.1 Shape Prior</td>
<td>66</td>
</tr>
<tr>
<td>9.2.2 Amortized Maximum Likelihood</td>
<td>68</td>
</tr>
<tr>
<td>9.2.3 Extended Variational Auto-Encoder</td>
<td>71</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>E.3.2</td>
<td>Maximum Likelihood</td>
</tr>
<tr>
<td>E.3.3</td>
<td>Non-Probabilistic Approach</td>
</tr>
<tr>
<td>E.3.4</td>
<td>Amortized Maximum Likelihood</td>
</tr>
<tr>
<td>E.3.5</td>
<td>Extended Variational Auto-Encoder</td>
</tr>
<tr>
<td>E.3.6</td>
<td>Discussion</td>
</tr>
<tr>
<td>E.4</td>
<td>3D Example</td>
</tr>
<tr>
<td>E.5</td>
<td>ShapeNet</td>
</tr>
<tr>
<td>E.6</td>
<td>KITTI</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Shape perception is a long-standing and fundamental problem both in human [Piz07, Piz10] and computer vision [FH15]. In both disciplines, a large body of work focuses on 3D reconstruction, e.g. reconstructing objects or scenes from one or more views. The problem of reconstructing 3D scenes or objects is an inherently ill-posed inverse problem because many configurations of shape, color, texture and lighting may give rise to the very same views [FH15]. In human vision, one of the fundamental problems is understanding how the human visual system accomplishes such tasks; in computer vision, in contrast, the goal is to develop 3D reconstruction systems. Both disciplines are related through insights regarding the cues and constraints used by humans to perceive 3D shapes. Results from human vision [Piz07, Piz10] suggest that these priors as well as the ability to process the provided cues is innate and not learned. In computer vision, as well, cues and priors are commonly built into 3D reconstruction pipelines through explicit assumptions. Recently, however – leveraging the success of deep learning – researchers started to learn shape models from data. Predominantly generative models have been used to learn how to generate, manipulate and reason about shapes, e.g. [GFRG16, DQN16, SGF16, BLRW16, WSK+15, WZX+16]. Learning such shape models offers many interesting possibilities for a wide variety of problems in 3D computer vision.

In this context, we focus on a specific problem in the realm of 3D reconstruction, namely shape completion from point clouds, as illustrated in Figure 1.1. This problem occurs when only a single view of an individual object is provided and large parts of the object are not observed or occluded. The problem is, however, also closely related to surface reconstruction [BTS+14] and, thus, has relevant applications in computer graphics, as well. Motivated by the success of learning shape models, we intend to tackle shape completion using a learning-based approach where we make use of shape priors learned from large datasets of shapes such as ModelNet [WSK+15] or ShapeNet [CFG+15]. This idea, i.e. learning-based shape completion, has recently gained traction by works such as [RUBG17, SM17, DQN16, SGF16, FSG16] or [REM+16]. Similarly, shape
Figure 1.1.: Illustration of the shape completion problem on KITTI [GLU12, GLSU13] where we intend to learn shape completion of cars. We show a bird’s eye view of a point cloud from KITTI in the background. On top, we show the same point cloud from a more natural viewpoint showing the points corresponding to cars in blue and the corresponding 3D bounding boxes, in particular their corners, highlighted in green. Below we show an illustration of the shape completion task. We extract point clouds of cars using the corresponding 3D bounding boxes and subsequently voxelize these points to obtain occupancy grids. Shape completion then describes the task of predicting a full shape which is also illustrated using occupancy grids, i.e. in voxelized form.

Prior s have already been applied to many different problems in computer vision including 3D pose estimation [PSR12, DSYT08, SDYT09, MHG15], tracking [MS14, LM09] or classical 3D reconstruction [GG15, DPRR13, BCLS13]. In the case of shape completion, however, most learning-based approaches still require full supervision; this means that observations are either synthesized from known models, or datasets need to be annotated. On real data, e.g. on KITTI [GLU12, GLSU13], shape completion without supervision can be posed as energy minimization problem over a latent space of shapes [ESL16, BCLS13, DPRR13]. In this case, shape completion usually involves solving a complex minimization problem using iterative approaches. Deep learning-based approaches, in contrast, can complete shapes using a single forward pass of the learned network. We find that both problems, the required supervision on the one hand and the computationally expensive optimization problems at the other, constrain the applicability of these approaches to real data considerably.

In this thesis, we propose a probabilistic framework for shape completion that tries to mitigate both problems: the need of annotated training data and the drawbacks of computationally involved optimization problems. Following related
work, e.g. [SM17, GFRG16, DQN16, SGF16, BLRW16, WSK15] or [WZX16], we utilize ShapeNet to learn a strong shape prior using deep generative models, specifically variational auto-encoders [KW13]. We hypothesize that using a strong prior of possible shapes, we are able to learn how to complete shapes under weak supervision, i.e. only given knowledge about the object category at hand. On real data, we additionally require knowledge about the object location in the form of bounding boxes, e.g. provided by an object detector. By learning shape completion using deep networks, we additionally reduce the problem to a forward pass of the trained network.

1.1 Contributions

We propose two different probabilistic frameworks enabling us to learn shape completion with weak supervision. In both cases, we first train a shape prior, particularly a variational auto-encoder. In the spirit of [ESL16], we can then formulate shape completion as maximum likelihood problem over the learned latent space. Instead of maximizing the likelihood independently for distinct observations, however, we follow the idea of amortized inference [GG14] and learn to predict the maximum likelihood solution directly given the corresponding observations. Specifically, we train an encoder, which embeds the observations in the same latent space, using an unsupervised, maximum likelihood loss between the observations and the corresponding shapes. This variant of amortized maximum likelihood allows us to learn shape completion under real conditions, e.g. on KITTI, and is able to compete with a fully-supervised baseline on a ShapeNet-based, synthetic dataset used for evaluation.

As alternative approach, we extend the general framework of latent space models, as implemented by variational auto-encoders, to specifically account for the observations. Applied to a pre-trained variational auto-encoder – representing the required shape prior – we derive the evidence lower bound of this extended variational auto-encoder which we then optimize in an unsupervised fashion, i.e. only given the observations. We also show that the underlying objective is closely related to our amortized maximum likelihood approach. On our synthetic, ShapeNet-based dataset, we experimentally demonstrate the applicability of the extended variational auto-encoder regarding shape completion. Overall, we present two approaches in favor of our claim that shape priors allow to learn shape completion in an unsupervised fashion, thereby also introducing many interesting directions for future research.

1.2 Outline

We formally introduce the problem of shape completion under weak supervision in Chapter 2. Subsequently, Chapter 3 reviews relevant related work. In Chapter 4, we introduce the necessary background of training 3D convolutional neural
networks. We then discuss appropriate shape representations in Chapter 5, before introducing the variational auto-encoder which is used as shape prior. In Chapter 7, we discuss how to use these models to learn shape completion with weak supervision. The datasets used for our experiments are introduced in Chapter 8. Finally, in Chapter 9, we present experiments and conclude in Chapter 10 including a discussion of future work.
We define shape completion as the surface reconstruction of a single object from a known object category given a partial observation of its shape – usually from one view only. As indicated in the introduction, we follow a learning-based approach. In its simplest form, i.e. the supervised case, the problem can be stated as:

**Problem 2.1** Given (partial) observations \( X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^R \) and shapes \( \mathcal{Y}^* = \{y_1^*, \ldots, y_N^*\} \subseteq \mathbb{R}^R \) of a specific object category such that \( y_n^* \) represents the ground truth shape of \( x_n \), the task is to learn a mapping \( x_n \mapsto y_n^* \) that is able to generalize to previously unseen observations.

Here, we assume the observations \( x_n \) and shapes \( y_n^* \) to be vectors in some high-dimensional space \( \mathbb{R}^R \) for brevity of notation. In practice, we will resort to occupancy grids or signed distance functions in three spatial dimensions, e.g. \( x_n, y_n^* \in \mathbb{R}^{H \times W \times D} \approx \mathbb{R}^R \). The observations \( x_n \) correspond to observed points – either occupied or non-occupied – and unobserved points. In order to make explicit that these are merely partial observations, we use \( x_n \in \{0, 1, \perp\}^R \) where \( \perp \) corresponds to unobserved points.

Obtaining the ground truth shapes \( \mathcal{Y}^* \) for real-world observations \( X \) is labor-intensive (cf. [GAGM15, MG15]). Thus, the unsupervised version of Problem 2.1 can be formulated as follows:

**Problem 2.2** Given (partial) observations \( X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^R \) of a specific object category, the task is to learn a mapping \( x_n \mapsto y(x_n) \in \mathbb{R}^R \) where \( y(x_n) \) represents a shape that matches the unknown ground truth shape \( y_n^* \in \mathbb{R}^R \) as close as possible.

Problem 2.2 can also be interpreted as weakly-supervised problem as we assume knowledge about the object category. However, this knowledge is not used explicitly yet. Furthermore, both Problem 2.1 and Problem 2.2 imply that each observation \( x_n \) corresponds to one individual object. This means that we assume an object detector to be given in order to extract the observations \( x_n \) from full point clouds as provided in real-world applications, e.g. on KITTI [GLU12, GLSU13].
Chapter 2. Problem 2.1. Proposed Approach

Shapes $y_m$  Observation $x_n$  Shape Completion  Shape $y^*_n$

Figure 2.1.: An illustration of Problem 2.3; the set of reference shapes $\mathcal{Y}$ is usually provided in the form of triangular meshes, e.g., from ShapeNet [CFG+15]. After voxelization, these are used to learn a shape prior. The problem of shape completion is then illustrated by showing a voxelized observation $x_n$ and the corresponding ground truth, i.e., completed, shape $y^*_n$.

2.1 Proposed Approach

Problem 2.2 is inherently ambiguous; in practice, infinitely many shapes may match a given observation and even the ground truth shape does not need to match the observation perfectly due to noise. We propose a Bayesian approach in order to model this uncertainty appropriately. As we assume knowledge of the object category, we collect a set $\mathcal{Y} = \{y_1, \ldots, y_M\} \subseteq \mathbb{R}^R$ of representative shapes corresponding to this object category. In practice, this might correspond to a set of Computer-Aided Design (CAD) models, i.e., triangular meshes. Problem 2.2 is then re-stated below:

**Problem 2.3** Given (partial) observations $\mathcal{X} = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^R$ and shapes $\mathcal{Y} = \{y_1, \ldots, y_M\} \subseteq \mathbb{R}^R$ of a specific object category, the task is to learn a mapping $x_n \mapsto y(x_n) \in \mathbb{R}^R$ such that $y(x_n)$ matches the unknown ground truth shape $y^*_n \in \mathbb{R}^R$ as close as possible.

We propose to use the set $\mathcal{Y}$ to learn a prior of possible shapes, e.g., using a variational auto-encoder. Shape completion can then be formulated as maximum likelihood problem over a lower-dimensional latent space $\mathcal{Z} = \mathbb{R}^Q$, $Q \ll R$. To this end, we interpret the individual voxels $y_i$, for $y \in \mathbb{R}^R$, as random variables and $x_i$, for $x \in \mathbb{R}^R$, as the corresponding observations. Then, $\mathcal{Y}$ is used to learn a prior model $p(y, z)$ which decomposes into a generative model and a prior: $p(y, z) = p(y|z)p(z)$. Using the set of observations $\mathcal{X}$, we then train an inference model that directly learns the mapping

$$x \mapsto \tilde{z} = \arg\max_z p(y = x|z)p(z). \tag{2.1}$$

thereby implicitly solving Problem 2.3. We interpret this approach as amortized maximum likelihood following the idea of [GG14]. In particular, we do not consider the maximum likelihood problem for each observation $x$ independently. Instead, we understand the maximum likelihood problem as loss between observations and shapes allowing to learn the embedding $x \mapsto \tilde{z}$ in an unsupervised fashion. In the context of variational auto-encoders, this boils down to training a new encoder to represent the embedding of observations within the latent space.
Alternatively, we can consider the observations $x_i$ to be random variables, as well. For the joint probability of the model, *i.e.* $p(x, y, z)$, we can subsequently derive the evidence lower bound. Using the simplifying assumption that $y$ and $z$ are statistically independent given $x$, we can use an extended variational autoencoder to implicitly learn an approximate model $q(z|x)$ which, again, implicitly solves Problem 2.3. Here, instead of directly optimizing the maximum likelihood in Equation (2.1) (involving $p(y = x|z)$), the observations are tied to the corresponding shapes through a Kullback-Leibler divergence. The learned model $q(z|x)$ can then be interpreted as a probabilistic embedding of the observations in the latent shape space, in analogy to the mapping $x \mapsto \tilde{z}$ learned when amortizing maximum likelihood. Overall, both approaches require a shape prior defining a latent space of shapes and subsequently learn to embed the partial observations within this space.
Chapter 3

Related Work

In this chapter, we intend to provide a concise but structured discussion of related work. We first consider the shape completion task itself, focusing on data-driven and learning-based approaches that are most related to our approach. We also discuss the use of shape priors in general, \textit{e.g.} for tasks including 3D reconstruction, 3D pose estimation or tracking. Finally, we give a brief overview of 3D deep learning, specifically regarding generative shape models, as well as the idea of amortized inference. We note, however, that we do not intend to provide a comprehensive discussion of these areas due to time and space constraints.

3.1 Shape Completion

Shape completion is closely related to 3D reconstruction as well as surface reconstruction; however, the addressed problems do usually not exactly match the problem of shape completion. Therefore, we refer to recent textbooks \cite{FP02, Sze11} and surveys \cite{MGV10, FH15, BTS14, BF05} for a general overview of these fields.

In contrast to surface reconstruction and 3D reconstruction, shape completion is usually performed on partial scans of individual objects. Following \cite{SKAG15}, shape completion approaches can roughly be categorized into data-driven methods and symmetry-based methods. The latter detect symmetry in non-occluded parts in order to complete shapes by completing the corresponding symmetry in occluded parts; representative works are \cite{TW05, PMW08, ZSW10, KAEP12} or \cite{LA11}. These approaches also found application in robotics, \textit{e.g.} for robot grasping \cite{KAEP12, MGRB09}. The data-driven case is more interesting in relation to the proposed approaches; in early work, Pauly \textit{et al.} \cite{PMG05} pose shape completion as retrieval and alignment problem. Further work in this direction includes \textit{e.g.} \cite{LDGN15, NXS12, GAGM15, DPRR13, ESL16, ESL17} where local shape features \cite{LDGN15, NXS12}, object knowledge \cite{GAGM15, DPRR13} or shape priors \cite{ESL16, ESL17, DPRR13} have found application. Model fitting of-
ten follows a scheme similar to the iterative closest point (ICP) algorithm [BM92] or is posed as energy minimization. We also refer to [BTS+14, Table 1].

Recently, several learning-based approaches have been proposed [RUBG17, FMJB16, SM17, DQN16, SGF16, REM+16, FSG16], most of them using deep learning. Shape completion is posed as supervised learning problem, e.g. from single RGB(-D) images [FSG16, GFRG16], partial observations [RUBG17, DQN16, REM+16, SM17], or noisy shapes [SGF16]. In comparison with data-driven approaches, shape completion usually involves a single forward pass of the trained network. However, these approaches can only be applied on synthetic datasets such as ModelNet [WSK+15] or ShapeNet [CFG+15] as explicit supervision is required. We also intend to leverage deep neural networks, however, we want to learn shape completion in a weakly supervised setting only using the given observations and knowledge about the object category. This allows us to learn shape completion on real data, e.g. on KITTI [GLU12, GLSU13], while still providing efficient inference through the trained network.

3.1.1 Shape Priors

As indicated above, shape priors play an important role in shape completion. For example, learning-based approaches can be understood as implicitly learning a shape prior from supervised training data [RUBG17, SM17, DQN16, SGF16, REM+16, FSG16]. Only [GFRG16] uses the learned shape model explicitly to constrain the space of predicted shapes. Data-driven methods may also make use of learned shape priors, e.g. using principal component analysis (PCA) [ESL16, ESL17] or Gaussian process latent variable models (GP-LVM) [DPRR13]. However, shape priors have also been applied in different domains, e.g. in 3D pose estimation – often tackled in conjunction with tasks such as image segmentation [PSR12, DSYT08, SDYT09] or scene flow [MHG15]. Shape priors have also become a standard tool in tracking, see [MS14] or [LM09] for related work in this domain. In classical 3D reconstruction, shape priors are also commonly used to resolve e.g. ambiguities [GG15], specularities [DPRR13] or the lack of texture [BCLS13]. In most cases, shape priors are “learned” from full shapes; this is in contrast to non-data-driven priors such as local or global smoothness priors, planarity assumptions etc., see [BTS+14]. We mostly follow the idea of [ESL16, ESL17, DPRR13] or [GFRG16] and use a shape prior explicitly to define and constraint the space of possible shapes.

3.2 3D Deep Learning and Amortized Inference

In computer vision, deep learning often refers to the practice of using deep convolutional neural networks [LBD+89] to train discriminative and more recently also generative models. While neural networks have been used and researched for several decades now (e.g. see [Bis95, Hay05, Ros58, RHW86, HSW89]), they only
recently received considerable attention. Among several milestones is the substantial performance increase on the ImageNet classification challenge \([DDS+09]\) in 2012 \([KSH12]\). Afterwards, deep convolutional neural networks have been successfully applied to a wide range of computer vision tasks – more complete overviews can be found in \([Sch15, WRX17, Li17, BCV12, LBH15, GBC16]\).

Recently, deep generative models, \(e.g.\) \([vdOKK16, vdOKE+16, GPM+14, MSJG15, KW13, LSLW16]\) and related work, have received considerable attention. Such models offer to generate realistic images or shapes and to learn strong prior models which are interesting for semi-, weakly- or unsupervised tasks, \(e.g.\) in \([KMRW14]\). In this thesis, we will use variational auto-encoders and subsequent work \([IAMB17, JGP16, MMT16]\) as they include both an inference and a generative model. This is also motivated by the fact that generative adversarial networks \([GPM+14]\) are known to be hard to train \([MNG17, NCT16, SGZ+16, GAA+17, ACB17]\), a problem that we anticipated to be amplified on 3D data. Generative models have also been applied to shape modeling, \(e.g.\) \([SM17, GRFG16, BLRW16, WSK+15, WZX+16]\), allowing shape interpolation, manipulation and generation.

### 3.2.1 3D Deep Learning

Most works, including \([WZX+16, SM17, WSK+15, BLRW16, SGF16, DQN16, GFRG16, GGR+16, MS15]\), naively generalize convolutional neural networks to 3D data, \(e.g.\) in the form of voxelized shapes. Unfortunately, this significantly limits the used resolution; usually resolutions between \(32^3\) and \(64^3\) are used. Some approaches, however, exploit the sparsity of 3D data through octrees, \(e.g.\) \([WLG+17, RUG16, RUBG17, TDB17]\), to reduce memory and time consumption and allow higher resolutions, \(e.g.\) up to \(256^3\) \([RUG16]\). Alternatively, sparse convolution schemes have been defined to speed up training \([LPS+16, ERW+17, Gra15]\). There is also a line of work applying convolutional neural networks directly to meshes \([GZC15, BMM+15, BZSL13]\) or point clouds \([QSMG16, FSG16, QYSG17]\). While we also use simple 3D convolutional neural networks for the presented experiments, which we also conduct on a resolution of \(32^3\), the above references provide interesting possibilities for future work.

### 3.2.2 Amortized Inference

To the best of our knowledge, the notion of amortized inference was first introduced in \([GG14]\), however, has been used repeatedly in recent work \([KW13, RM15, WL16, RMW14, RHG16]\). While a clear, coherent definition is still missing, amortized inference generally describes the idea of learning inference, \(e.g.\) in the form of “learning to sample” \([WL16]\) or amortized, \(i.e.\) learned, variational inference \([KW13, RM15]\). The idea includes that drawn samples – or past inferences – are not considered independently; instead, inference can profit from insights
obtained from past inferences. In our case, we learn, *i.e.* amortize, the maximum likelihood problem for shape completion.
Deep learning generally describes the use of deep neural networks \cite{GBC16} – where depth may refer to both the number of model parameters as well as the number of used layers, \textit{i.e.} processing steps. In the context of this thesis, neural networks are used in the framework of variational auto-encoders to learn the shape prior, \textit{i.e.} the embedding of shapes $\mathcal{Y}$ within a latent space $\mathcal{Z}$. Specifically, both the generative model $p(y|z)$ and the approximate recognition model $q(z|y)$ are implemented using neural networks. This is accomplished by choosing appropriate parameterizations and predicting the corresponding parameters – \textit{e.g.} mean and variance for Gaussian distributions. In the proposed approaches, neural networks will also be used to learn the embedding of observations $x$ in the latent shape space. For amortized maximum likelihood, this will be a deterministic embedding $x \mapsto z(x; w)$ – here, we made the parameters $w$ of the neural network explicit. For the proposed extended variational auto-encoder, the embedding is represented by the approximate recognition model $q(z|x)$.

In general, neural networks can be introduced from different viewpoints (cf. \cite{Bis06, DHS01, Bis95, Hay05, GBC16}). In this chapter, we follow \cite{GBC16} and introduce neural networks as series (or acyclic graph, in general) of tensor operations which use a set of parameters $w$ in order to – given input $x$ – compute a function $y(x; w)$ that may approximate some target function. We will first introduce the used concept of tensors – the data structure underlying all modern deep learning frameworks – before gradually defining more complex networks and finally discussing network training.

\section{4.1 Tensors}

In our context, real tensors are generally understood as multi-dimensional arrays \cite[Section 2.1]{GBC16} of real numbers:

\begin{definition}
A (real) tensor is an element $t \in \mathbb{R}^{n_1 \times \cdots \times n_r}$ where $(n_1, \ldots, n_r) \in \mathbb{N}^r$ and $r$ is called the rank of the tensor. We write $r_i := r_{(i_1, \ldots, i_r)}$ to index a tensor; in this case, $i = (i_1, \ldots, i_r)$ is called multi-index.
\end{definition}
In practice, the rank is often referred to as dimension. While this is mathematically imprecise, we will use both interchangeably. Thus, a tensor $t \in \mathbb{R}^{n_1 \times \ldots \times n_r}$ has dimension (=rank) $r$ and $n_i$ is called the size of dimension $i$. We also allow so-called singleton dimensions, i.e. dimensions of size 1. Then, the terminology can be applied one-to-one to many popular deep learning frameworks such as Torch, PyTorch, Theano, Tensorflow, Caffe and more\(^1\).

**Example 4.1** Examples include trivial cases such as scalars, i.e. rank-0 tensors, vectors, i.e. rank-1 tensors, and matrices, i.e. rank-2 tensors. Examples specific to our problem include:

**Multi-Channel Image** An image is a tensor of rank 3: $t \in \mathbb{R}^{C \times H \times W}$ where $C$ is the number of channels, $H$ the height of the image and $W$ the width of the image. For grayscale (i.e. single-channel) images, $C = 1$, while color images use $C = 3$ channels usually corresponding to red, green and blue. Grayscale as well as color is usually encoded using 8-bit each.

**Multi-Channel Volumes** A tensor $t \in \mathbb{R}^{C \times H \times W \times D}$ of rank 4 can be interpreted as $C$-channel volume where $H$, $W$ and $D$ represent height, width and depth, respectively. Volumes are commonly used in medical imaging (i.e. MRI or CT volumes). In our case, we use volumes to represent shapes using occupancy grids or (signed) distance functions, see Chapter 5.

**Batch of Multi-Channel Volumes** A batch, i.e. a set, of multi-channel volumes can be interpreted as tensor of rank 5: $t \in \mathbb{R}^{B \times C \times H \times W \times D}$ where $B$ is the size of a set of multi-channel volumes. When training neural networks, $B$ will be the batch-size.

In this thesis, we will mostly work with volumes and multi-channel volumes and – during training – with batches of these. Therefore, we always consider the rank-5 case and usually refer to the dimensions as $B \times C \times H \times W \times D$.

### 4.1.1 Basic Tensor Operations

For tensors of rank 1 or 2 we can apply all the operations known from linear algebra (e.g. matrix-vector or matrix-matrix multiplications). Additionally, we define element-wise operations which are commonly used in deep learning:

**Definition 4.2** Let $t$, $s$ be two tensors; let $i \in \mathbb{N}^5$ be a multi-index. Further, let $\times$ be any binary arithmetic operation on $\mathbb{R}$. Then the corresponding element-wise operation is defined by $(t \times s)_i = t_i \times s_i$.

**Definition 4.3** Let $t$ be a tensor and $i \in \mathbb{N}^5$ be a multi-index; further, let $h : \mathbb{R} \mapsto \mathbb{R}$ be a real-valued function. Then, $h$ may operate element-wise on $t$, i.e. $h(t)_i = h(t_i)$.

4.2 Layered Neural Networks

In the following, we introduce layered neural networks – models consisting of several layers, i.e. a sequence, of tensor operations. We follow the introduction of [GBC16, Chapter 6]:

**Definition 4.4** A (feed-forward, layered) neural network is a directed, acyclic graph \( G = (V, E) \) where each node \( v \in V \) corresponds to an operation mapping one or more input tensors to one or more output tensors. The edges \((v_i, v_j) \in E\) characterize the information flow, i.e. one or more of the output tensors of operation \( v_i \) are fed as input into operation \( v_j \). The operations \( v_i \) are also called layers and may have a finite number of adjustable weights.

Given a topological ordering – see [CLRS09, Chapter 22] – of the vertices \( V = \{v_1, \ldots, v_{|V|}\} \), the network computes its output by sequentially computing the layers’ outputs given an input \( x \), i.e. \( v_i(x), v_{i+1}(v_i(x), x), v_{i+2}(v_{i+1}(v_i(x)), v_i(x), x), \ldots \). An early example of a one-layer neural network is the perceptron [Ros58]:

**Example 4.2** The perceptron is a linear classifier (cf. [Bis06, Chapter 4]):

\[
y(x; w) = x w^T \quad \text{where} \quad x \in \mathbb{R}^{|C_{in}|} \text{ is an input tensor and } w \in \mathbb{R}^{|C_{out}|} \text{ is a weight vector.}
\]

Classification is done by considering \( \text{sign}(y(x; w)) \) with class labels in \( \{-1, 1\} \). While the inner product \( x w^T \) might be unintuitive, it allows to easily apply the decision function to a batch of samples \( x_1, \ldots, x_B \) stacked into \( x \in \mathbb{R}^{B \times |C_{in}|} \).

Today the perceptron is more important than ever because it describes the basic building blocks of neural networks, the fully connected layer:

**Definition 4.5** A fully connected layer, denoted \( \text{fc}_{C_{in} \times C_{out}} \), takes as input a tensor \( x \in \mathbb{R}^{B \times |C_{in}|} \) and computes:

\[
\text{fc}_{C_{in} \times C_{out}}(x) = x w^T \in \mathbb{R}^{B \times |C_{out}|}
\]

where \( w \in \mathbb{R}^{|C_{out}| \times |C_{in}|} \) is the corresponding weight matrix.

Usually, the fully connected layer is combined with an additive bias term:

\[
y(x; w) = x w^T + \begin{bmatrix} b \\ \vdots \end{bmatrix}
\]

with \( x \in \mathbb{R}^{B \times |C_{in}|} \), \( b \in \mathbb{R}^{|C_{out}|} \) (which is repeated \( B \) times to allow operations on batches) and, as before, \( w \in \mathbb{R}^{|C_{out}| \times |C_{in}|} \). To make this additive term explicit, we use a separate layer. The bias layer can easily be generalized to tensors of higher rank and is usually followed by non-linearity:

**Definition 4.6** Let \( h : \mathbb{R} \rightarrow \mathbb{R} \) be a real-valued function. Then, \( h \) also denotes a layer taking as input an arbitrary tensor \( x \) and computing \( h(x) \) element-wise.

These non-linearities are also called activation or transfer functions (cf. [Bis95, Hay05, GBC16]) and are motivated by the wish to approximate complex non-linear functions [HSW89] using a combination of simpler, non-linear functions.
Chapter 4. Deep Learning 4.3. Convolutional Neural Networks

**Figure 4.1.** Illustration of a multi-layer perceptron with $L = 3$ fully-connected layers followed by bias layers and non-linearities. The sizes $C_1$ and $C_2$ are hyper-parameters while $C_0$ and $C_3$ are determined by the problem at hand. Overall, the multi-layer perceptron represents a function $y(x; w)$ parameterized by the weights $w$ in the fully-connected and bias layers.

**Example 4.3** Common non-linearities for neural networks include the logistic sigmoid

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

and the rectified linear unit $\text{ReLU}(x) = \max(x, 0) = [x]_+$. The logistic sigmoid allows to model probabilities and ReLU non-linearities have been shown to reduce training time and improve performance \cite{KSH12,SLJ15} while being a good model of biological neurons \cite{GBB11}.

Finally we discuss our first layered neural network whose network graph is illustrated in Figure 4.1: the multi-layer perceptron \cite[Chapter 4]{Bis95} which consists of a sequence of fully connected, bias and non-linearity layers. The name comes from the fact that it can be interpreted as a multi-layer extension of the perceptron:

**Example 4.4** A multi-layer perceptron consists of several stages of computation as illustrated in Figure 4.1. Let $L > 0$ be the number of fully connected layers; $x \in \mathbb{R}^{1 \times C_0}$ be the input. Then, the first stage computes:

$$
y^{(1)} = h \left( x(w^{(1)})^T + b^{(1)} \right)
$$

with $w^{(1)} \in \mathbb{R}^{C_1 \times C_0}$ and $b^{(1)} \in \mathbb{R}^{1 \times C_1}$. As shown in Figure 4.1, this already subsumes the first fully connected, bias and non-linearity layers. Subsequently, for $l > 1$ we define:

$$
y^{(l)} = h \left( y^{(l-1)}(w^{(l)})^T + b^{(l)} \right).
\tag{4.1}
$$

with $w^{(l)} \in \mathbb{R}^{C_{l+1} \times C_l}$ and $b^{(l)} \in \mathbb{R}^{1 \times C_l}$. Finally, $y^{(L)}$ is the output of the network. Again, this formulation extends to the case of multiple $x_1, \ldots, x_B$ stacked into one tensor $x \in \mathbb{R}^{B \times C}$.

### 4.3 Convolutional Neural Networks

Convolutional neural networks were introduced in \cite{LBD89} and replace the fully connected layers with discrete convolutions. This has the advantage that spatial information within images can be used explicitly while simultaneously reducing the number of parameters. Additionally, discrete convolution is inherently invariant to translations, can be implemented efficiently and is still a linear operation with respect to both input and parameters. Overall, convolutional neural net-
4.3. Convolutional Neural Networks

works played an important role in the recent success of deep learning in computer vision.

4.3.1 Convolution

The discrete convolution is a well-known technique in signal processing. The following example introduces the general concept. More details can be found in most signal processing textbooks, e.g. [GW06].

Example 4.5 For simplicity, let \( x \in \mathbb{R}^C \) be a one-dimensional input signal. Further, let \( w \in \mathbb{R}^{2K+1} \) be a so-called kernel. For notational convenience, we index \( w \) using \( w_i, -K \leq i \leq K \). Then, the discrete convolution is defined as:

\[
(x * w)_i := \sum_{j=-K}^{K} x_{i-j}w_j
\]  

(4.2)

where the size of the result will depend on how the convolution operation is defined on the boundaries of the signal. In particular, for \( i < K \) or \( i > n - K \), Equation (4.2) is not well-defined. Usually, we assume a padded version of \( x \) such that \( x_i = 0 \) for all \( i \notin \{1, \ldots, n\} \). We note that it is straightforward to show that the discrete convolution is linear in both \( x \) and \( w \).

Definition 4.7 In general, let \( x \in \mathbb{R}^{n_1 \times \ldots \times n_r} \) be a tensor of rank \( r \) and \( w \in \mathbb{R}^{2K_1+1 \times \ldots \times 2K_r+1} \) be a kernel of rank \( r \). Then, the convolution of rank \( r \) is defined as follows:

\[
(x * w)_i = \sum_{j_1=-K_1}^{K_1} \ldots \sum_{j_r=-K_r}^{K_r} x_{i-j_1-s_1} \ldots x_{i-j_r-s_r} \]

where \( w \) is indexed with \( -K_1 \leq j_1 \leq K_1, \ldots, -K_r \leq j_r \leq K_r \) for simplicity of notation and multi-indices \( i \) and \( j \) can be added and subtracted element-wise.

The above definition is specific to our use cases, i.e. convolutional neural networks, where we assume odd kernel sizes for simplicity. As in the one-dimensional case, the input signal is assumed to be padded with zeros; this is commonly done when using convolutions in neural networks [GBC16, Section 9.5].

4.3.2 Convolutional Layer

Based on the convolution as described above, the convolutional layer can be introduced as follows:

Definition 4.8 A convolutional layer \( \text{conv}_{C_{in}, C_{out}, K} \) takes as input a tensor \( x \in \mathbb{R}^{B \times C_{in} \times H \times W \times D} \) and computes

\[
(\text{conv}_{C_{in}, C_{out}, K}(x))_{b,c_{out}} = \sum_{c_{in}=1}^{C_{in}} w_{c_{out}, c_{in}} * t_{b,c_{in}}
\]

where \( W \in \mathbb{R}^{C_{out} \times C_{in} \times K \times K \times K} \) is the corresponding weight tensor with \( K \) being odd.
In words, a convolutional layer takes as input a tensor consisting of \( C_{\text{in}} \) channels, convolves each of these channels with the corresponding kernel \( w_{\text{out},c_{\text{in}}} \in \mathbb{R}^{K \times K \times K} \) and sums over the results. This is done \( C_{\text{out}} \) times, thereby creating \( C_{\text{out}} \) new channels. The convolutional layer comes in many different flavors, e.g. using (fractional) strides [DV16], as deconvolution [ZKTF10] or as dilated convolution [YK15] to name just a few; we will, however, use the introduced, simple variant.

### 4.3.3 Pooling and Upsampling

In [LBD+89], strided convolution was applied to subsample the input image; alternatively, max pooling [ZC88] computes the maximum value within non-overlapping windows. Today, max pooling is popular to incorporate robustness against noise and small transformations:

**Definition 4.9** A max pooling layer \( \text{pool}_K \) takes as input a tensor \( x \in \mathbb{R}^{B \times C \times H \times W \times D} \) and computes a tensor of size \( B \times C \times \frac{H}{K} \times \frac{W}{K} \times \frac{D}{K} \) as

\[
(p_{\text{pool}})_b,c,i = \max_{K(i_{1}-1) \leq j_1 < K(i_{1}), ..., K(i_{3}-1) \leq j_3 < K(i_{3})} x_{b,c,j}
\]

where \( i = (i_1, i_2, i_3) \) and \( j = (j_1, j_2, j_3) \) are multi-indices and we assume \( H \), \( W \) and \( D \) to be divisible by \( K \).

Pooling essentially computes the maximum value over non-overlapping 3-dimensional cubes of edge length \( K \), thereby reducing the spatial size of the input tensor. Many other types of pooling have been proposed, see [BPL10] for a discussion and related work, and the idea has been used to incorporate invariances against other transformations – such as rotations in 3D [QSN+16,HZ16].

Instead of reducing the size of the tensor, it is also interesting to spatially increase the size of the tensor. To this end, upsampling techniques from image processing are commonly used, for example using bilinear interpolation. Another option is deconvolution (or fractionally strided convolution) [ZKTF10,DV16]. Alternatively, max pooling can also be “inverted” resulting in unpooling layers [ZF14]. Instead, we follow [ODO16] and use simple nearest neighbor upsampling where each element is duplicated within fixed-sized windows to increase the spatial size:

**Definition 4.10** A nearest neighbor upsampling layer \( \text{nnup}_K \) takes as input a tensor \( x \in \mathbb{R}^{B \times C \times H \times W \times D} \) and computes a tensor of size \( B \times C \times KH \times KW \times KD \) as

\[
(n_{\text{nnup}}(x)_K)_b,c,i = x_{b,c,\left\lfloor \frac{i_1}{K} \right\rfloor, \left\lfloor \frac{i_2}{K} \right\rfloor, \left\lfloor \frac{i_3}{K} \right\rfloor}
\]

for \( 1 \leq i_1 \leq KH, \ldots, 1 \leq i_3 \leq KD \).
4.4 Auto-encoders

As examples we chose a popular class of (convolutional) neural networks: auto-encoders. These models were – to the best of our knowledge – first introduced in [RHW86] but gained more attention with the use for pre-training deep neural networks [HS06, HOT06, EBC10]:

**Example 4.6** The main goal of an auto-encoder is to reconstruct its input through a lower-dimensional latent representation. Auto-encoders can be split into an encoder, computing the mapping \( z(x; w) \) from input \( x \) to low-dimensional code \( z \), and a decoder, computing the reconstruction \( \hat{x}(z; w) \approx x \) from the latent code \( z \). Encoder and decoder often “mirror” each other, i.e. both can be implemented using the same multi-layer perceptron as illustrated in Figure 4.2. Here, both consist of three fully connected layers, each followed by a bias layer and a non-linearity. The input \( x \in \mathbb{R}^R \) is transformed to a \( Q \ll R \) dimensional latent code \( z \) which itself is used to estimate a reconstruction \( \hat{x} \approx x \). The exact dimensionalities \( C_1, C_2 \) and \( Q \) as well as the used non-linearity \( h \) are hyper-parameters that can be adapted depending on the application.

When translating auto-encoders to the convolutional case, we usually apply several stages of convolutional layers (often followed by non-linearities and pooling). However, to compute a one-dimensional latent code \( z \), the output tensor from the final pooling layer needs to be reshaped.

**Definition 4.11** A view layer \( \text{view}_{B',C',H',W',D'} \) views a given input tensor of size \( B \times C \times H \times W \times D \) as a tensor of size \( B' \times C' \times H' \times W' \times D' \) given that \( BCHWD = B'C'H'W'D' \).
Figure 4.3.: Illustration of a convolutional auto-encoder consisting of encoder (top) and decoder (bottom). Both are modeled using two stages of convolutional layers each followed by a bias layer and a non-linearity layer. The encoder uses max pooling to decrease the spatial size of the input; the decoder uses upsampling to increase it again. The number of channels $C_1$, $C_2$ and $C_3$ as well as the size $Q$ are hyper parameters. We assume the input to comprise one channel. Again, the reconstruction loss $L_{\tilde{x},x}$ quantifies the quality of the reconstruction and is minimized during training.

In practice, a tensor is always implemented using a one-dimensional array such that it represents a contiguous set of memory blocks, e.g.,

$$t_{i_1,i_2,i_3,i_4,i_5} := \hat{t}_{((i_1 C + i_2) H + i_3) W + i_4) D + i_5}$$

for $t \in \mathbb{R}^{B \times C \times H \times W \times D}$ and $\hat{t} \in \mathbb{R}^R$ with $R = BCHWD$ the underlying one-dimensional tensor. Thus, reshaping may not involve any actual data movement but just represents a different "view" on the data.

**Example 4.7** The example of a general auto-encoder can be extended to the convolutional case. Instead of a succession of fully connected layers followed by bias and non-linearity layers, the encoder consists of a sequence of convolutional layers followed by bias layers, non-linearity layers and max pooling layers. The decoder, again, mirrors the encoder while replacing max pooling layers with upsampling layers. The latent code is computed using a fully connected layer. The full model is illustrated in Figure 4.3.

In both examples, the mappings $x \mapsto z(x;w)$ and $z \mapsto \tilde{x}(z;w) \approx x$ are deterministic. In Chapter 6 we will see that auto-encoders can also be re-formulated in a probabilistic, non-deterministic way [KW13] where instead of direct mappings, the corresponding probability distributions $q(z|x)$ and $p(x|z)$ are modeled. We will use such a model to learn our shape prior in Chapter 6.
4.5 Training

Training (convolutional) neural networks involves adjusting the collected weights \( w \) of the constructed model \( y(x; w) \) in order to approximate an unknown target function. The target function, however, is often known for individual input values \( \mathcal{X} = \{x_1, \ldots, x_N\} \), i.e. \( y(x_n; w) \). In this supervised case, i.e. where both inputs \( x_n \) and targets \( y_n^* \) are given, training usually proceeds in two steps: formulating a loss function \( L(y(x_n; w), y_n^*) \) to measure the quality of approximation, and selecting an appropriate algorithm to minimize it.

4.5.1 Losses

Revisiting our approach to Problem 2.3, it has the form of the more general problem \( \arg\min_y -\ln p(y|x) \) which we will use to derive two loss functions by considering different parametric forms of \( p(y|x) \). In particular, we introduce both the binary cross entropy error as well as the sum-of-squared loss; a more detailed discussion can be found in [Bis95, Chapter 6] or [GBC16, Section 6.2].

**Example 4.8** Considering binary classification, we are interested in the posterior \( p(y|x_n) \). The neural network \( y(x_n; w) \) models the probability of class one, i.e. \( y(x_n; w) \approx p(y = 1|x_n) \) such that \( p(y = 0|x_n) = 1 - p(y = 1|x_n) \). The distribution of \( p(y = y_n^*|x_n) \), \( y_n^* \in \{0, 1\} \), can then be written as Bernoulli distribution:

\[
p(y = y_n^*|x_n) = y(x_n; w)^{y_n^*} (1 - y(x_n; w))^{1-y_n^*}.
\]

Taking the negative log-likelihood leads to the binary cross entropy error:

\[
L_{BCE}(w) = -\ln p(y = y_n^*|x_n) = -y_n^* \ln y(x_n; w) - (1 - y_n^*) \ln(1 - y(x_n; w))
\]

where we directly wrote \( L_{BCE} \) in terms of the weights \( w \) to be found.

Overall, using \( y(x; w) \) to model binary distributions, i.e. Bernoulli distributions, the negative log-likelihood leads to the binary cross entropy error. We can formalize this in the following two definitions:

**Definition 4.12** Let \( y \in \{0, 1\} \) be a random variable; then \( y \) is distributed according to a Bernoulli distribution, \( y \sim \text{Ber}(y; \theta) \), with parameter \( \theta \) if the probability density function takes the form:

\[
p(y) = \theta^y (1 - \theta)^{1-y}.
\]

**Definition 4.13** Given a set of samples with associated labels \( \{x_n, y_n^*\}_{n=1}^{N} \), \( y_n^* \in \{0, 1\} \), and a neural network \( y(x; w) \) modeling the posterior distribution \( y(x; w) \approx p(y = 1|x) \), the binary cross entropy error \( L_{BCE}(w) \) is defined as the negative log-likelihood assuming the samples \( x_n \) are independent:

\[
L_{BCE}(w) = -\sum_{n=1}^{N} y_n^* \ln y(x_n; w) + (1 - y_n^*) \ln(1 - y(x_n; w))
\]
For regression tasks, \( p(y|x) \) cannot be modeled as discrete distribution. In this case, \( p(y|x) \) is usually modeled as Gaussian distribution which, following the same derivation as in Example (4.8), leads to the sum-of-squared loss:

**Definition 4.14** Let \( y \in \mathbb{R} \) be a random variable; then \( y \) is distributed according to a Gaussian distribution, i.e. \( y \sim \mathcal{N}(y; \mu, \sigma^2) \), with mean \( \mu \) and variance \( \sigma^2 \) if its probability density function is given by

\[
p(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - \mu)^2}{\sigma^2}\right).
\]

**Definition 4.15** Given a set of samples \( \{x_n, y^*_n\}_{n=1}^N \) with real-valued labels, \( y^*_n \in \mathbb{R} \), and a neural network \( y(x; w) \) modeling the mean of a Gaussian distribution, the sum-of-squared loss \( \mathcal{L}_{\text{SSE}}(w) \) is defined as the corresponding negative log-likelihood assuming independent samples:

\[
\mathcal{L}_{\text{SSE}}(w) = \sum_{n=1}^N \ln 2\pi + \ln \sigma^2 + \frac{(y^*_n - y(x_n; w))^2}{\sigma^2}.
\] (4.3)

In Equation (4.3), we kept both the constant term and \( \sigma^2 \) – which are usually dropped for convenience – as it is also possible to let the neural network model both the mean and its variance. Both, the binary cross entropy loss and the sum-of-squared loss are, in practice, often applied per voxel for dense prediction tasks, i.e. when modeling a distribution per voxel. Also note that we usually assume continuously differentiable losses.

### 4.5.2 Stochastic Gradient Descent

After having set up the network \( y(x; w) \), acquired a set of samples \( \{x_n, y^*_n\}_{n=1}^N \) and chosen a loss \( \mathcal{L}(w) \), gradient-based optimization algorithms usually proceed as follows [NW06, Section 2.2]. Given an initial value of the weights \( w^{(0)} \), we iteratively compute weight updates \( \Delta w^{(t)} \) defining a sequence

\[
w^{(t+1)} = w^{(t)} + \gamma \Delta w^{(t)}
\]

using step size – or learning rate – \( \gamma \). The overall goal is to find a stationary point corresponding to a (local) minimum:

**Definition 4.16** We call \( w^* \) a stationary point of \( \mathcal{L} \), if \( \nabla \mathcal{L}(w^*) = 0 \).

**Lemma 4.1** If \( w^* \) is a local minimum of \( \mathcal{L} \), then \( \nabla \mathcal{L}(w^*) = 0 \).

**Proof:** A proof can be found in [NW06, Theorem 2.2].

In deep learning, we usually do not get any guarantees beyond the first-order necessary condition in Lemma 4.1 as neural networks are highly non-convex and using second-order information, i.e. the Hessian of \( \mathcal{L} \), is computationally expensive.
**Algorithm 4.1** The general gradient descent algorithm; different choices of the learning rate \( \gamma \) and the estimation technique for \( \nabla L(w) \) may lead to different implementations.

**Input:** initial weights \( w^{(0)} \), number of iterations \( T \)

**Output:** final weights \( w^{(T)} \)

1. for \( t = 0 \) to \( T - 1 \)
2. estimate \( \nabla L(w^{(t)}) \)
3. compute \( \Delta w^{(t)} = -\nabla L(w^{(t)}) \)
4. select learning rate \( \gamma \)
5. \( w^{(t+1)} := w^{(t)} + \gamma \Delta w^{(t)} \)
6. return \( w^{(T)} \)

Gradient descent is one of the best known first-order optimization algorithms and has been studied extensively in the context of neural networks [Bot12]. The underlying idea is to move in the direction of the negative gradient; this means
\[
\Delta w^{(t)} = -\nabla L(w^{(t)}).
\]

Gradient descent, as depicted in Algorithm 1 is the basis for many different optimization algorithms used in deep learning, e.g. [KB14, Zei12].

The difficulties of implementing Algorithm 1 include choosing the learning rate \( \gamma \) and estimating \( \nabla L(w^{(t)}) \) in each iteration. The learning rate is usually set experimentally and might decrease automatically during training. The gradient \( \nabla L(w^{(t)}) \) can be estimated over the whole dataset, i.e.
\[
\nabla L(w^{(t)}) \approx \sum_{n=1}^{N} \nabla L(y(x_n; w^{(t)}), y_n^*)
\]

In practice, however, this is computationally too expensive, even for datasets of moderate size. A well-studied alternative is stochastic gradient descent where, in each iterations, the gradient is estimated on a random subset of the samples:

**Definition 4.17** Stochastic gradient descent is an implementation of Algorithm 1 where the gradient \( \nabla L(w^{(t)}) \) is estimated on a randomly chosen subset \( B \subseteq \{1, \ldots, N\} \) of samples, i.e.
\[
\nabla L(w^{(t)}) \approx \sum_{n \in B} \nabla L(y(x_n; w^{(t)}), y_n^*)
\]

and the learning rate \( \gamma = \gamma^{(t)} \) decreases during the iterative process:
\[
\gamma^{(t)} = \max(\gamma_{\min}, \gamma^{(0)} \alpha_{\gamma}^{T_{\gamma}})
\]
where \( \gamma^{(0)} \) is given and \( \alpha_{\gamma} \in (0, 1) \), \( T_{\gamma} \in \mathbb{N} \) define an exponential decay.

The subset \( B \) is also called mini-batch; in the following we will also use \( B \) as the size of this mini-batch. This directly relates to our standard tensor size for
multi-channel volumes where \( B \) represents the first dimension. The parameters \( \gamma^{(0)} \) and \( T_{\gamma} \) are usually set experimentally.

### 4.5.2.1 Momentum

Gradient descent is well known to have difficulties with highly non-linear objective functions including local minima, plateaus as well as steep areas. Common problems include slow convergence along plateaus or oscillations near steep areas. A popular approach introduces a so-called momentum term [Bis95, Section 7.5] to replace line 3 in Algorithm 1 with

\[
\Delta w^{(t)} = -\beta \Delta w^{(t-1)} - \nabla \mathcal{L}(w^{(t)})
\]

where \( \beta \) is the momentum parameter. It controls how fast the optimization scheme adapts to changes in the weight landscape such that oscillations and slow convergence can be avoided. In practice, the momentum parameter is often increased during training in analogy to Equation (4.4) using an \( \alpha \beta > 1 \).

### 4.5.3 Weight Initialization

Stochastic gradient descent is crucially dependent on a good initial guess for \( w^{(0)} \). In practice, weight initialization usually follows a simple scheme: a hyper parameter \( \sigma \) is chosen (either globally or per layer) and the weights are initialized as random values from \( U[-\sigma, \sigma] \) or \( \mathcal{N}(0, \sigma) \). While \( \sigma \) can be chosen by hand (typically in the order of \( \sigma \approx 0.05 \)), [GB10] suggests choosing \( \sigma = \frac{\sqrt{\mathcal{C}_{in}} \mathcal{C}_{out}}{\mathcal{C}_{in} \mathcal{C}_{out}} \) for a fully connected layer fc\( \mathcal{C}_{in}, \mathcal{C}_{out} \). This scheme can intuitively be applied to convolutional layers, as well, and performs well in practice.

### 4.5.4 Error Backpropagation

After introducing stochastic gradient descent and discussing weight initialization, the remaining problem is the computation of \( \nabla \mathcal{L} (y(x_n; w^{(t)}), y_n^*) \) for an individual sample \( x_n \) and its label \( y_n^* \). The error backpropagation algorithm [RHW86] solves this problem by recursively applying the chain rule; we follow [Bis95, Section 4.8] to introduce the main idea as example before stating the general algorithm.

**Example 4.9** Considering Example 4.4, we use \( \mathcal{L}_n := \mathcal{L} (y(x_n; w^{(t)}), y_n^*) \) for simplicity and neglect the iteration \( w = w^{(t)} \) for notational convenience. Exemplary taking an individual weight \( w_{i,j}^{(t)} \) in layer \( \ell \) which is involved in computing the output \( y_i^{(t)} \) according to Equation (4.1), the chain rule applied to \( \mathcal{L}_n \) yields:

\[
\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(t)}} = \frac{\partial \mathcal{L}_n}{\partial y_i^{(t)}} \frac{\partial y_i^{(t)}}{\partial w_{i,j}^{(t)}}.
\]
4.5. Training Chapter 4. Deep Learning

To compute the first part, the chain rule can be applied recursively to give:

$$\frac{\partial L_n}{\partial y_i^{(l)}} = \frac{\partial L_n}{\partial y^{(L)}} \frac{\partial y^{(L)}}{\partial y_i^{(l)}} = \ldots = \frac{\partial L_n}{\partial y^{(L)}} \left( \prod_{l=0}^{L} \frac{\partial y^{(l+1)}}{\partial y_i^{(l)}} \right) \frac{\partial y^{(l+1)}}{\partial y_i^{(l)}}$$

Here, we exploit the hierarchical structure of the multi-layer perceptron, i.e. $w_{i,j}^{(l)}$ is involved in computing $y_i^{(l)}$ which itself is the input for $y^{(l+1)}$ and so forth. The second part of Equation (4.5) can easily be computed from Equation (4.1):

$$\frac{\partial y_i^{(l)}}{\partial w_{i,j}^{(l)}} = h'(y^{(l-1)}(w^{(l)})^T + b^{(l)})_i \frac{\partial y^{(l-1)}(w^{(l)})^T + b^{(l)}}{\partial w_{i,j}^{(l)}}$$

where both parts can easily be computed given the input $y^{(l-1)}$ to layer $l$ and the derivative $h'$ of the non-linearity $h$. Overall, the derivatives can be calculated easily if, for each layer, we are able to compute its derivatives with respect to its input and with respect to its weights.

The above successive application of the chain rule can be formalized into the error backpropagation algorithm (see [Bis95, Section 4.8] and [GBC16, Section 6.5] for details and illustrations):

**Algorithm 4.1** Error backpropagation algorithm for a layered neural network represented as computation graph $G = (V,E)$ as in Definition 4.4.

1. For a sample $(x_n, y_n^*)$, propagate the input $x_n$ through the network to compute the outputs $(v_{i_1}, \ldots, v_{i_{|V|}})$ (in topological order).
2. Compute the loss $L_n := L(v_{i_{|V|}}, y_n^*)$ and its gradient

$$\frac{\partial L_n}{\partial v_{i_{|V|}}}.$$ 

3. For each $j = |V|, \ldots, 1$ compute

$$\frac{\partial L_n}{\partial w_j} = \frac{\partial L_n}{\partial v_{i_{|V|}}} \prod_{k=j+1}^{V} \frac{\partial v_{i_k}}{\partial v_{i_{k-1}}} \frac{\partial v_{i_j}}{\partial w_j}.$$ 

where $w_j$ refers to the weights in node $i_j$.

4.5.5 Regularization

As for any other machine learning technique, regularization plays an important role in deep learning, especially as deep neural networks are known to be universal approximators [HSW89] and tend to overfit the given data [Ben09]. A general
form of regularization augments the loss $\mathcal{L}$ using a weighted regularization term [Bis95, Section 9.2]:

$$\hat{\mathcal{L}}(w) = \mathcal{L}(w) + \kappa \mathcal{P}(w)$$

Often, $\mathcal{P}(w)$ is defined as $L_p$ norm [Bis95], e.g. $\mathcal{P}(w) = \|w\|_2^2$ for the Euclidean norm – this regularization is also referred to as weight decay as, neglecting $\mathcal{L}(w)$, the weights will tend exponentially towards zero [Bis95, Section 9.2].

In contrast to explicit regularization, researchers are commonly using implicit forms of regularization. Among these are early stopping [Bis95, Section 9.2], unsupervised pre-training [EBC+10], batch normalization [IS15], dropout [SHK+14] and data augmentation [GBC16, Section 7.4]. In the following, we will discuss some of them in more detail.

### 4.5.5.1 Early Stopping

A neural network is said to overfit to a training set if the training loss is particularly low while the loss on an held-out validation set is comparably large. During training, overfitting occurs at the point when the training loss continues to decrease while the loss on the validation set starts increasing again. Therefore, it seems reasonable to stop training as soon as the validation loss increases again – this method is called early stopping [Bis95, Section 9.2]. The network can then be re-trained on the full data for the same number of iterations.

### 4.5.5.2 Batch Normalization

Batch normalization [IS15] was not explicitly introduced as regularization technique but to simplify training. To this end, it is worth considering intermediate layers during training – i.e. the layers $y^{(l)}$ for $1 < l < L$ in the formalism of the multi-layer perceptron of Example 4.4. During training, the distribution of the values in one of these layers is not supposed to change significantly as subsequent layers assume them to be fixed for a local gradient descent step. Batch normalization tries to control these distributions through normalization. We directly introduce the “convolutional” variant:

**Remark 4.1** A batch normalization layer $bn$ that follows a convolutional layer takes as input a tensor $x \in \mathbb{R}^{B \times C \times H \times W \times D}$ and computes

$$\text{(bn}(x))_{b,c,i} = w_c \frac{x_{b,c,i} - \mu_c}{\sigma_c^2 + \epsilon} + b_c$$

where $i = (i_1, i_2, i_3)$, $\epsilon > 0$ is small, $w, b \in \mathbb{R}^C$ are weight parameters, and

$$\mu_c = \frac{1}{BHWD} \sum_{b=1}^{B} \sum_{i_1=1}^{H} \sum_{i_2=1}^{W} \sum_{i_3=1}^{D} x_{b,c,i}$$

$$\sigma_c^2 = \frac{1}{BHWD} \sum_{b=1}^{B} \sum_{i_1=1}^{H} \sum_{i_2=1}^{W} \sum_{i_3=1}^{D} (x_{b,c,i} - \mu_c)^2.$$
Intuitively, batch normalization normalizes the individual channels of the input tensor by the mean and variance over the mini-batch and all spatial dimensions. The output values would then be distributed according to a unit Gaussian. However, this would be a constraint on what the preceding convolutional layer can represent, e.g. it would not be able to compute the identity of its inputs. Therefore, Ioffe and Szegedy [IS15] introduce the weights $w$ and $b$ allowing to shift and scale the unit Gaussian. The batch normalization layer is usually added after a convolutional layer but before the corresponding non-linearity layer. Due to Equation (4.6), bias layers can be omitted. For fully connected layers, batch normalization usually normalizes all values independently [IS15].

4.5.6 Data Augmentation

Data augmentation, see [GBC16, Section 7.4], describes the practice of artificially generating new training samples from existing ones using random transformations in order to incorporate invariances to these transformations. Apart from different noise models, color transformations (for images) as well as geometric transformations including translations, rotations, shear etc. are popular. This also generalizes to our case of deep learning in 3D where e.g. rotational invariance is desired [QSN+16, HZ16].

4.6 Discussion

Unfortunately, the scope of this thesis does not allow an extensive discussion of recent developments in deep learning. Most of the presented approaches – except for e.g. batch normalization, the introduced weight initialization scheme or deconvolution/upsampling – have been used for several decades already, at least in similar forms. However, many interesting ideas have only been proposed recently and, as a result, deep learning has found wide applicability in computer vision. Therefore, we refer the reader to some recent surveys and textbooks: [Sch15, WRX17, Li17, BCV12, LBH15] and [GBC16].

Although we only introduced the basic concepts of convolutional neural networks, these have been shown to be sufficient to tackle complex 3D tasks, including shape completion [FMJB16, SM17, DQN16, SGF16, REM+16, FSG16] and generative shape modeling [SM17, GFRG16, BLRW16, WSK+15, WZX+16]. Additionally, computational resources are still a limiting factor for convolutional neural networks in 3D; more complex networks are very difficult to train, especially in high resolutions. Therefore, some works also introduced sparse variants of the discussed concepts [WLG+17, RUG16, RUBG17, TDB17, LPS+16, ERW+17, Gra15]. The introduced layers, due to their simplicity, are also available in most deep learning frameworks allowing to easily reproduce and judge the presented experiments. Finally, convolutional auto-encoders, as introduced in Example 4.7, can easily be “upgraded” to powerful generative models such as variational auto-
encoders [KW13, IAMB17, JGP16, MMT16]. In the following chapters, we will discuss the use of variational auto-encoders for learning shape priors in more detail.
Chapter 5

Shape Representation

Coming back to the original problem of this thesis, the goal is to complete shapes. As we want to learn both the shape prior as well as the proposed shape completion approaches using 3D convolutional neural networks, we require grid-based representations of the observations, i.e. point clouds from KITTI [GLU12, GLSU13], and the shapes, i.e. meshes from ShapeNet [CFG+15]. As foreshadowed in Chapter 2, we primarily rely on occupancy grids for representing the observations and both occupancy grids and signed distance functions for representing shapes. In the following we briefly formalize both modalities in detail.

5.1 Point Clouds

Point clouds are unordered sets of 3D points. In our formulation of Problem 2.3, point clouds correspond to a raw version of the observations $\mathcal{X}$:

**Definition 5.1** A point cloud $\mathcal{P} = \{p_1, \ldots, p_N\} \subseteq \mathbb{R}^3$ is a set of points with dimensions corresponding to width (horizontal), height (vertical) and depth in this order.

We note that the dimensions are not coherent with the introduced tensor dimensions, i.e. $H \times W \times D$. While this might seem confusing, we decided to follow common practice in computer vision and deep learning where the first dimension of tensors refers to the height of an image or a volume\(^1\). This means, that height and width axes need to be swapped when voxelizing point clouds into occupancy grids, see Section 5.3. Point clouds from KITTI have already been illustrated in Figure 1.1.

\(^1\)This is motivated by popular computer vision libraries such as OpenCV (http://opencv.org/) and tools such as SciPy (https://www.scipy.org/) as well as deep learning frameworks, e.g. Torch (http://torch.ch/).
Chapter 5. Shape Representation 5.2. Meshes

Figure 5.1.: Illustration of the used modalities, i.e. occupancy grids and signed distance functions. In particular, a given triangular mesh is first simplified in order to reduce its complexity and obtain a watertight mesh which is required for voxelization, i.e. to obtain an occupancy grid. In this thesis, we derive the corresponding signed distance function from the occupancy grid using a distance transform – as illustrated by the dashed line. We note, however, that signed distance functions should in general be derived from the original mesh in order to obtain sub-voxel accuracy.

5.2 Meshes

Another source of information for our problem are datasets of Computer-Aided Design (CAD) models. In this thesis, we assume that CAD models are provided as triangular meshes – we refer to [BKP*10] for a detailed introduction. Regarding Problem 2.3, these correspond to a raw version of the shape set $\mathcal{Y}$:

**Definition 5.2** A triangular mesh $\mathcal{M} = (V, F)$ is defined by a set of vertices $V \subseteq \mathbb{R}^3$ and a set of triangular faces $F \subseteq \{1, \ldots, |V|\}^3$ such that $f = (f_1, f_2, f_3) \in F$ defines a triangular face enclosed by the corresponding vertices $v_{f_1}$, $v_{f_2}$ and $v_{f_3}$. Faces implicitly also define the edges $E(F)$ between the vertices.

An illustration of a triangular mesh from ShapeNet can be found in Figure 5.1. The concepts of adjacency and incidency are naturally extended to triangular meshes. We note that a triangular mesh only defines the surface of an object. Without additional constraints it is generally hard to reason about the interior and exterior of the surface – and whether the surface is closed. As we also see in Chapter 8, this question naturally leads to the concept of watertight meshes. In the literature [BKP*10, Section 1.3], watertight meshes are usually defined as 2-manifold meshes without boundary edges, see Appendix A for details. Another problem, illustrated in Figure 5.1, concerns very detailed meshes, i.e. meshes consisting of a large number of faces and vertices, often going into the tenth of thousands. Therefore, mesh simplification [BKP*10, Chapter 6] is an important problem in computer graphics. In Chapter 8 we will present a very practical
approach to compute very rough, simplified meshes. In our case, this algorithm solves both problems – obtaining watertight and simplified meshes.

### 5.3 Occupancy Grids

Given point clouds or watertight meshes, occupancy grids are a natural representation for applying deep learning techniques. In particular, 3D convolutional neural networks are able to operate directly on the provided topology and thereby utilize spatial information. Operating directly on triangular meshes or point clouds, in contrast, is less straightforward regarding a proper representation (e.g. for encoding the faces) which would need to be invariant to the order of faces or points \([\text{GGR}^{+}16, \text{FSG}16]\) and allow to leverage local information \([\text{QYS}G17, \text{GZC}15, \text{BMM}^{+}15, \text{BZSL}13]\). Occupancy grids assume a voxelization of the space and determine, for each voxel, its occupancy. Specifically, a voxel is considered occupied if it lies inside the interior of the shape or intersects with its surface. For point clouds, a voxel is considered occupied if it contains at least one point.

**Definition 5.3** An occupancy grid is a tensor \(x \in \{0, 1\}^{H \times W \times D}\) where the elements \(x_i\) are called voxels and \(x_i = 1\) corresponds to an occupied voxel and \(x_i = 0\) determines an unoccupied voxel.

Occupancy grids can be interpreted as explicit shape representation. With \(H, W\) and \(D\) being large enough, i.e. using high resolution, even small details of shapes can be captured, e.g. see \([\text{TDB}17]\). In Figure 5.1, we illustrate the voxelization of the shown simplified mesh.

### 5.4 Signed Distance Functions

In contrast to occupancy grids, signed distance functions are used to define shapes implicitly. In the general case, a signed distance function can be defined as follows \([\text{BKP}^{+}10, \text{Chapter 1}]\):

**Definition 5.4** Let \(F : \mathbb{R}^3 \rightarrow \mathbb{R}\) be a continuous function. Then, the surface of a shape is implicitly defined by the zero level-set \(S := \{x \in \mathbb{R}^3 | F(x) = 0\}\) of \(F\). As convention, \(F\) is negative in the interior of the shape and positive in the exterior of the shape.

The implicit representation through distance functions simplifies the problem of determining the interior and exterior of a shape. The name stems from the fact that \(F\) is usually taken to be the signed distance to the nearest point on the surface, e.g.

\[ |F(x)| = \min_{x_S \in S} \|x - x_S\|_2.\]
In order to numerically work with signed distance functions, they are usually defined on a regular grid:

**Definition 5.5** A signed distance function is a tensor \( x \in \mathbb{R}^{H \times W \times D} \) such that \( x \) is the signed distance of the center of the corresponding voxel to the surface of the shape. It is negative inside the shape, and positive outside of it.

In practice, and as simplification, we are computing signed distance functions from occupancy grids. This is an important distinction; by using the occupancy grid representation, we lose accuracy depending on the used resolution. To regain higher accuracy, we could instead utilize a signed distance function obtained from the original surface [LC87, BKP+10]; we leave this for future work.

**Remark 5.1** Given an occupancy grid \( x \in \{0, 1\}^{H \times W \times D} \), we derive the corresponding distance function \( df(x) \in \mathbb{R}^{H \times W \times D} \) as

\[
    df_i(x) = \min_{j, x_j = 1} \|i - j\|_2.
\]

Similarly, the signed distance function \( sdf(x) \) can be defined by combining \( df(x) \) and\( df(1 - x) \) with the appropriate signs.

In practice, the conversion from occupancy grids to (signed) distance functions can be computed using generalized distance transforms [FH12]. To compute the sign, the interior and exterior must be known; for watertight meshes the interior and exterior can be determined using occupancy grids and connected components/flood filling algorithms [Sze11, Section 3.4]. As defined above, we assume the distance function \( df_i(x) \) to represent the distance to the next occupied voxel \( x_j = 1 \) such that \( df_i(1 - x) \) is the distance to the next unoccupied voxel. We also found that using the log signed distance function, i.e.

\[
    lsd(x)_i = \text{sign}(sdf(x)_i) \ln(1 + |sdf(x)_i|),
\]

aids neural network training – a strategy also employed for similar representations, e.g. for depth prediction [EF15, EPF14, LRB+16]. Intuitively, the log reduces the range of the prediction problem; in contrast to simple scaling, however, the range is reduced non-linearly such that the range of small distances (i.e. around the zero level set) is effectively increased at the expense of larger distances.
Chapter 6

Shape Prior

Given occupancy grids or signed distance functions of the shape set \( \mathcal{Y} \) and the observations \( \mathcal{X} \), our approach to Problem 2.3 involves two steps: first, we learn a shape prior defining the space of allowed shapes; and second, we learn an inference model to embed the observations within the same latent shape space. The shape prior defines a joint distribution \( p(y, z) = p(y|z)p(z) \) of shapes \( y \) and latent codes \( z \). Essentially, the shape prior represents an embedding of the shapes \( \mathcal{Y} \) within a low-dimensional latent space \( \mathcal{Z} \). By enforcing a prior \( p(z) \) on the latent space, we are able to generate shapes using \( y \sim p(y|z) \) for \( z \sim p(z) \). Additionally, shape completion can be defined over the low-dimensional latent space by also learning an embedding \( x \mapsto z \) of observations \( x \) within the latent space. In our formulation of amortized maximum likelihood, this embedding is deterministic. For the proposed extended variational auto-encoder, the embedding is also probabilistic, \( i.e. \) expressed as \( q(z|x) \).

In the following, we intend to learn the shape prior using variational auto-encoders on set of shapes \( \mathcal{Y} = \{y_1, \ldots, y_M\} \). For occupancy grids or signed distance functions, we assume a flattened version \( y \in \mathbb{R}^R \simeq \mathbb{R}^{H \times W \times D} \) for simplicity. The latent space is then given as \( \mathcal{Z} = \mathbb{R}^Q \) for small \( Q \ll R \). A variational

![Graphical model for recognition model](image-a)

![Graphical model for generative model](image-b)

Figure 6.1.: Illustration of the graphical models corresponding to encoder and decoder, \( i.e. \) recognition model \( q(z|y) \) and generative model \( p(y|z) \).
auto-encoder is an implementation of the more general continuous latent variable model, which can easily be summarized using the two graphical models in Figure 6.1. Although we are mainly interested in the generative model \( p(y|z) \) given a fixed prior \( p(z) \), training also requires learning the recognition model \( q(z|y) \) in order to maximize the overall likelihood \( p(y) \). For simple Gaussian models, the recognition model \( q(z|y) \) as well as the so-called marginal likelihood

\[
p(y) = \int p(y, z) dz = \int p(y|z)p(z)dz
\]

(6.1)
can usually be determined analytically, e.g. see the discussion of probabilistic principal component analysis in Appendix B. In the case of variational auto-encoders, both the generative and the recognition model are implemented using neural networks. Then, the recognition model as well as the marginal likelihood can only be approximated – mainly because the integral in Equation (6.1) becomes intractable. In this chapter, we first follow [BKM16] and [Doe16] to introduce the framework of variational inference which allows to maximize a lower bound on the likelihood. Subsequently, we discuss two variants of variational auto-encoders where the prior \( p(z) \) is either modeled using Gaussian distributions or based on Bernoulli distributions – corresponding to continuous latent variables or binary latent variables, respectively.

## 6.1 Variational Inference

In general, variational inference is posed as the problem of finding a model distribution \( q(z) \) to approximate the true posterior \( p(z|y) \)

\[
q(z) = \arg\min_q \text{KL}(q(z)|p(z|y))
\]

(6.2)

where the Kullback-Leibler divergence \( \text{KL} \) is a distance measure defined on probability distributions. The Kullback-Leibler divergence can then be rewritten to obtain a lower bound on the intractable marginal likelihood \( p(y) \). The Kullback-Leibler divergence is formally defined as (see [KF09, Section A.1] or [Bis06, Section 1.6]):

**Definition 6.1** The Kullback-Leibler divergence between two probability distributions \( q(z) \) and \( p(z|y) \) is defined as:

\[
\text{KL}(q(z)|p(z|y)) = \mathbb{E}_{q(z)} \left[ \ln \frac{q(z)}{p(z|y)} \right]
\]

where \( \mathbb{E}_{q(z)} \) denotes the expectation with respect to the distribution \( q(z) \).
A close look at the Kullback-Leibler divergence reveals that the optimization problem in Equation (6.2) involves computing the marginal likelihood:

\[
\text{KL}(q(z)p(z|y)) = \mathbb{E}_{q(z)} \left[ \ln \frac{q(z)}{p(z|y)} \right] \\
= \mathbb{E}_{q(z)}[\ln q(z)] - \mathbb{E}_{q(z)}[\ln p(z|y)] \\
= \mathbb{E}_{q(z)}[\ln q(z)] - \mathbb{E}_{q(z)}[\ln p(z, y)] + \ln(p(y)).
\]

Re-arranging left- and right-hand side leads to the evidence lower bound which is also referred to as variational lower bound [BKM16]:

\[
\ln p(y) = \text{KL}(q(z)p(z|y)) - \mathbb{E}_{q(z)}[\ln q(z)] + \mathbb{E}_{q(z)}[\ln p(z, y)] \\
\geq \mathbb{E}_{q(z)}[\ln q(z)] + \mathbb{E}_{q(z)}[\ln p(z, y)] \\
= \mathbb{E}_{q(z)}[\ln q(z)] + \mathbb{E}_{q(z)}[\ln p(z)] + \mathbb{E}_{q(z)}[\ln p(y|z)] \\
= -\text{KL}(q(z)p(z)) + \mathbb{E}_{q(z)}[\ln p(y|z)].
\]

The original problem of maximizing the intractable marginal likelihood \(p(y)\) in Equation (6.1) is then approximated by maximizing the evidence lower bound which we formally define as:

**Definition 6.2** The variational lower bound or evidence lower bound derived from Problem (6.2) is given by

\[
-\text{KL}(q(z)p(z)) + \mathbb{E}_{q(z)}[\ln p(y|z)] = \mathbb{E}_{q(z)} \left[ \ln \frac{p(y, z)}{q(z)} \right] \tag{6.3}
\]

In this general formulation of the evidence lower bound, the model distribution \(q(z)\) can be arbitrary. However, in the context of latent variable models, it makes sense to make the model distribution depend on \(y\) explicitly, i.e. \(q(z|y)\), as we want to be able to reconstruct any \(y\) from the corresponding latent code \(z\). Then, the evidence lower bound in Equation (6.3) takes the form of an auto-encoder where \(q(z|y)\) represents the encoder and \(p(y|z)\) the decoder. Thus, a variational auto-encoder can be trained by maximizing the right-hand-side of Equation (6.3) after choosing suitable parameterizations for the distributions \(p(z)\) and \(q(z|y)\). In the following, we first discuss the Gaussian case.

### 6.2 Gaussian Variational Auto-Encoder

In the framework of variational auto-encoders [KW13], the model distribution \(q(z|y)\) is implemented as neural network; similarly, \(p(y|z)\) is modeled as neural network. In the case of Gaussian variational auto-encoders, both \(q(z|x)\) and \(p(z)\) are assumed to be Gaussian, specifically

\[
q(z|y) = \mathcal{N}(z; \mu(y; w), \text{diag}(\sigma^2(y; w))) \\
p(z) = \mathcal{N}(z; 0, I_Q)
\]
where the dependence on the neural network weights \( w \) is made explicit; i.e. both the mean \( \mu(y; w) \in \mathbb{R}^Q \) and the covariance matrix \( \text{diag}(\sigma^2(y; w)) \in \mathbb{R}^{Q \times Q} \) are predicted using a neural network with parameters \( w \) that are to be optimized. In the following we will neglect the weights \( w \) for brevity.

Considering the evidence lower bound from Equation (6.3), i.e.

\[
(6.3) = -\text{KL}(q(z|y)p(z)) + \mathbb{E}_{q(z|y)}[\ln p(y|z)],
\]

the Kullback-Leibler divergence between \( q(z|y) \) and \( p(z) \) can be computed analytically. However, differentiating the lower bound with respect to the hidden weights in \( q(z|y) \) is problematic. A classical Monte-Carlo approximation of the expectation \( \mathbb{E}_{q(z|y)}[\ln p(y|z)] \) would require differentiating through the sampling process \( z \sim q(z|y) \). Therefore, Kingma and Welling [KW13] proposed the so-called reparameterization trick. In particular, the random variable \( z \sim q(z|y) \) is reparameterized using a differentiable transformation \( g(z, \epsilon) \) based on an auxiliary variable \( \epsilon \) drawn from a unit Gaussian:

\[
z_i = g_i(y, \epsilon_i) = \mu_i(y) + \epsilon_i \sigma^2_i(y)
\]

with \( \epsilon_i \sim \mathcal{N}(0, 1) \). Overall, given a sample \( y_m \in \mathbb{R}^R \), the objective to be minimized has the form

\[
\mathcal{L}_{\text{VAE}}(w) = \text{KL}(q(z|y_m)p(z)) - \frac{1}{L} \sum_{i=1}^{L} \ln p(y_m|z_{i,m})
\]

where \( z_{i,m} = g(\epsilon_{i,m}, y) \) and \( \epsilon_{i,m} \sim \mathcal{N}(0, I_Q) \). Here, \( L \) is the number of samples to use for the Monte Carlo estimator of the reconstruction error. In practice, the loss \( \mathcal{L}_{\text{VAE}} \) is applied on mini-batches and \( L = 1 \) is usually sufficient.

Given the neural network weights \( w \), the generation process can be summarized as follows: draw \( z \sim p(z) = \mathcal{N}(z; 0, I_Q) \), and draw \( y \sim p(y|z) \). Similarly, recognition is performed by drawing \( z \sim q(z|y) \), i.e. \( \epsilon \sim \mathcal{N}(0, I_Q) \) and \( z = g(y, \epsilon) \). For evaluation purposes, i.e. for measuring the recognition performance, \( z \) is usually set to \( z = \mathbb{E}_{q(z|y)}[z] \); this can be accomplished by directly considering \( \mu(y) \).

### 6.2.1 Practical Considerations

As mentioned above, the Kullback-Leibler divergence of two Gaussian distributions can easily be calculated directly. As we consider diagonal covariance matrices the Kullback-Leibler divergence is separable over \( 1 \leq i \leq Q \). Then, following [Kul59, Section 9] (also see [RW06, Appendix A]) we have

\[
\text{KL}(\mathcal{N}(z_i; \mu_{1,i}, \sigma^2_{1,i}) | \mathcal{N}(z_i; \mu_{2,i}, \sigma^2_{2,i})) = \frac{1}{2} \ln \frac{\sigma_{2,i}}{\sigma_{1,i}} + \frac{\sigma^2_{1,i}}{2\sigma^2_{2,i}} + \frac{(\mu_{1,i} - \mu_{2,i})^2}{2\sigma^2_{2,i}} - \frac{1}{2},
\]

and with \( \sigma_{2,i} = 1 \) and \( \mu_{2,i} = 0 \) (and for simplicity \( \sigma^2_i := \sigma^2_i(y) = \sigma^2_{1,i} \) and \( \mu_i := \mu_i(y) = \mu_{1,i} \)) it follows:

\[
\text{KL}(p(z_i|y)p(z_i)) = \frac{1}{2} \ln \sigma_i + \frac{1}{2} \sigma^2_i + \frac{1}{2} \mu^2_i - \frac{1}{2}.
\]
The remaining part of the objective is the reconstruction error, \( i.e. \) the negative log-likelihood \(- \ln p(y|z)\). This depends on how \( p(y|z) \) is modeled; in our case, \( p(y|z) \) is decomposed element-wise over voxels:

\[
p(y|z) = \prod_{i=1}^{R} p(y_i|z) \Rightarrow - \ln p(y|z) = - \sum_{i=1}^{R} \ln p(y_i|z)
\]

For shapes \( y \) in the form of occupancy grids, we use Bernoulli distributions to model individual voxels, \( i.e. \) \( p(y_i|z) = \text{Ber}(y_i; \theta_i(z)) \) where the probabilities of occupancy \( \theta_i(z) \) are predicted using the decoder. The negative log-likelihood then reduces to the binary cross entropy error. When working with signed distance functions, we use Gaussian distributions with fixed variance to model individual voxels, \( i.e. \) \( p(y_i|z) = \mathcal{N}(y_i; \mu_i(z), \sigma^2) \) where the \( \mu_i(z) \)'s are predicted by the decoder. In this case, the negative log-likelihood leads to the sum-of-squared error. See Section 4.5.1 for details.

For illustration and implementation purposes, we define two additional layers:

**Definition 6.3** A Gaussian Kullback-Leibler divergence layer \( \text{KLD}_\mathcal{N} \) takes as input two tensors \( \mu \in \mathbb{R}^{B \times Q} \) and \( \sigma^2 \in \mathbb{R}^{B \times Q} \) and computes the Kullback-Leibler divergence

\[
\sum_{b=1}^{B} \text{KL}(\mathcal{N}(z; \mu_b, \text{diag}(\sigma^2_b)))|\mathcal{N}(z; 0, I_Q)),
\]

before passing the predicted \( \mu_b, \sigma^2_b \in \mathbb{R}^Q \) on to the next layer (unchanged):

\[
\text{KLD}_\mathcal{N}(\mu, \sigma^2) = (\mu, \sigma^2).
\]

Essentially, this layer just makes the computation of the Kullback-Leibler divergence explicit – in a separate layer. This means that the forward pass is unaffected. However, it is important to consider the derivatives with respect to the predicted \( \mu(y) \) and \( \sigma^2(y) \) as these need to be added to the decoder’s gradients (coming from the reconstruction loss) during error backpropagation in order to account for the Kullback-Leibler divergence during training:

\[
\frac{\partial \text{KL}(p(z_i|y)|p(z_i))}{\partial \mu_i} = \mu_i \quad \text{and} \quad \frac{\partial \text{KL}(p(z_i|y)|p(z_i))}{\partial \sigma_i} = - \frac{1}{2\sigma_i} + \sigma_i
\]

After computing the Kullback-Leibler divergence, the reparameterization trick is applied:

**Definition 6.4** A Gaussian reparameterization layer \( \text{repa}_\mathcal{N} \) takes as input two tensors \( \mu \in \mathbb{R}^{B \times Q} \) and \( \sigma^2 \in \mathbb{R}^{B \times Q} \) and computes a single output tensor

\[
\text{repa}_\mathcal{N}(\mu, \sigma^2) = \mu + \epsilon \sigma
\]

where \( \epsilon \in \mathbb{R}^{B \times Q}, \epsilon_{b,i} \sim \mathcal{N}(\epsilon; 0, 1) \), and \( \mu \) are multiplied element-wise.

Again, we note that the primary purpose of the reparameterization layer is to sample from \( q(z|y) \) in a differentiable manner. Then, the following example
Chapter 6. Shape Prior 6.2. Gaussian Variational Auto-Encoder

![Diagram of a Gaussian variational auto-encoder]

Figure 6.2.: An illustration of a Gaussian variational auto-encoder with four convolutional stages in both encoder and decoder. This is also the architecture used in experiments in Chapter 9 where we assume volumes of size $32 \times 32 \times 32$ such that the spatial size just before the fully connected layers of the encoder is $2 \times 2 \times 2$ resulting in a 1024-dimensional representation when considering 128 channels. See Example 6.1 for details.

Illustrates how a convolutional auto-encoder such as the one from Example 4.7 can be “upgraded” to a variational auto-encoder using the newly introduced layers:

**Example 6.1** We consider Figure 6.2 of our implementation of a variational auto-encoder with four convolutional stages in the encoder and decoder. In the case of the encoder, two fully connected layers independently compute the mean $\mu(y) \in \mathbb{R}^Q$ and variance $\sigma^2(y) \in \mathbb{R}^Q$ given the same input. Then, the Kullback-Leibler divergence $KL(q(z|y)||p(z))$ is computed and mean and variance are passed into the reparameterization layer. Here, an auxiliary variable $\epsilon \sim \mathcal{N}(0; I_Q)$ is chosen in order to sample a latent code $z \sim q(z|y) = \mathcal{N}(z; \mu(y), \text{diag}(\sigma^2(y)))$ using $z = g(y, \epsilon)$ and then passed into the decoder. At test time, the reparameterization layer can be removed, and the predicted mean $\mu(y)$ can be directly passed into the decoder. In the illustration, we also make the reconstruction loss $\mathcal{L}(\hat{y}, y) = - \ln p(y|\hat{z})$ explicit.

In practice, letting the encoder predict $\sigma^2(y)$ directly is problematic as the variance $\sigma^2(y)$ may not be negative. Therefore, we follow publicly available implementations$^1$ and let the encoder predict log-variances instead, i.e. $l_i(y) := \ln \sigma^2(y)$.

---

$^1$ For example [https://github.com/y0ast/VAE-Torch](https://github.com/y0ast/VAE-Torch), [https://github.com/staturecranee/dcgan_vae_torch](https://github.com/staturecranee/dcgan_vae_torch) and [https://github.com/Kaixhin/Autoencoders](https://github.com/Kaixhin/Autoencoders).
This makes sure that the variance $\sigma^2(y) = \exp(l_i(y))$ will always be positive. The Kullback-Leibler divergence from Equation (6.4) and the corresponding derivative with respect to $l_i(y)$ as well as the reparameterization trick from Equation (6.5) are easily adapted.

Training a Gaussian variational auto-encoder in practice means balancing the – possibly conflicting – objectives corresponding to reconstruction loss and Kullback-Leibler divergence. While the reconstruction loss is intuitively interpretable, e.g. as binary cross entropy error or scaled sum-of-squared error in the Bernoulli and Gaussian cases, respectively, the Kullback-Leibler divergence is less intuitive. Therefore, it might be beneficial to monitor the latent space statistics over a held-out validation set. Concretely, this means monitoring the first two moments of the predicted means, i.e.

$$\bar{\mu} = \frac{1}{QM'} \sum_{i=1}^{Q} \sum_{m=1}^{M'} \mu_i(y_m)$$

and

$$\text{Var}[\mu] = \frac{1}{QM'} \sum_{i=1}^{Q} \sum_{m=1}^{M'} (\mu_i(y_m) - \bar{\mu})^2$$

where $M'$ is the size of the validation set. Additionally, it is worth monitoring the first moment of the predicted log-variances

$$\bar{l} = \frac{1}{QM'} \sum_{i=1}^{Q} \sum_{m=1}^{M'} \ln \sigma_i^2(y_m) = \frac{1}{QM'} \sum_{i=1}^{Q} \sum_{m=1}^{M'} l_i(y_m).$$

Note that these statistics are computed over all $Q$ dimensions of the latent space – this is appropriate as the prior $p(z)$ is a unit Gaussian such that all dimensions can be treated equally. During training, these statistics should converge to a unit Gaussian, i.e. $\bar{\mu} \approx 0$ and $\text{Var}[\mu] \approx 1$, and the network should be capable of making certain predictions, i.e. small $\bar{l}$.

### 6.3 Bernoulli Variational Auto-Encoder

The Gaussian variational auto-encoder will be used to learn the latent space $Z$, i.e. the shape prior. However, in the case of the extended variational autoencoder for shape completion we intend to model the joint probability $p(x, y, z)$ by introducing random variables for the observations $x_i$, as well. Here, both $y$ and $z$ are considered to be latent variables, therefore, we also need to be able to model discrete latent variables – particularly binary latent variables considering $y$ to be represented as occupancy grid. Unfortunately, this is not straight-forward in the framework presented so far.
Recently, however, researchers [MMT16, JGP16] were able to model both $p(z)$ and $q(z|y)$ using Bernoulli distributions, \textit{i.e.}

\[
p(z) = \prod_{i=1}^{Q} \text{Ber}(z_i; 0.5)
\]

\[
p(z|y) = \prod_{i=1}^{Q} \text{Ber}(z_i; \theta_i(y))
\]

where $\theta_i(y)$ are predicted using the encoder, given the input $y$. Again, the evidence lower bound is to be optimized, this means for a sample $y_m$:

\[
\mathcal{L}_{VAE}(w) = \text{KL}(q(z|y_m)||p(z)) - \frac{1}{L} \sum_{l=1}^{L} \ln p(y_m|z_{l,m})
\]

where $z_{l,m} \sim q(z|y)$. The Kullback-Leibler divergence can be computed analytically, however, differentiation through the sampling process $z_{l,m} \sim q(z|y)$ is problematic. Unfortunately, the reparameterization trick used in the Gaussian case is not applicable anymore. At this point, Maddison \textit{et al.} [MMT16] and, concurrently, Jang \textit{et al.} [JGP16] propose an alternative reparameterization trick for general discrete distributions which we define as follows:

**Definition 6.5** Let $z \in \{1, \ldots, K\}$ be a random variable; then $z$ is distributed according to a discrete distribution, \textit{i.e.} $z \sim \text{Dis}(z; \pi)$, with parameters $\pi = (\pi_1, \ldots, \pi_K)$ if

\[
p(z = k) = \pi_k \quad \text{and} \quad \sum_{k=1}^{K} \pi_k = 1.
\]

We represent $z$ using the so-called one-hot encoding, meaning that $z \in \{0, 1\}^K$ such that $z_k = 1$ and $z_{k'} = 0$, $k' \neq k$ if event $k$ occurs.

The reparameterization is additionally based on the Gumbel distribution:

**Definition 6.6** Let $\epsilon \in \mathbb{R}$ be a random variable. Then $\epsilon$ is distributed according to the Gumbel distribution, \textit{i.e.} $\epsilon \sim \text{Gu}(\epsilon; \mu, \beta)$, with parameters $\mu$ and $\beta$ if

\[
p(\epsilon) = \exp \left( - \exp \left( - \frac{\epsilon - \mu}{\beta} \right) \right).
\]

The standard Gumbel distribution is the special case of $\mu = 0$ and $\beta = 1$.

A key insight is that it is very easy to sample from a Gumbel distribution: let $u_k \sim U(0, 1)$, then $\epsilon_k = \mu - \beta \log(-\log(u_k))$ is distributed according to $\text{Gu}((\mu, \beta)$.

The standard Gumbel distribution also helps to sample from a discrete distribution. Let $\epsilon_1, \ldots, \epsilon_K \sim \text{Gu}(\epsilon; 0, 1)$, then set $z_k = 1$ for

\[
k = \arg\max_k \ln \pi_k + \epsilon_k
\]
and $z$ will be distributed according to a discrete distribution, \textit{i.e.} $z \sim \text{Dis}(z; \pi)$; this is called the Gumbel trick. Replacing the argmax with its smooth counterpart, \textit{i.e.} the softmax, yields the final reparameterization trick:

$$
z_k = \frac{\exp(\ln \pi_k + \epsilon_k)}{\sum_{k' = 1}^{d} \exp(\ln \pi_{k'} + \epsilon_{k'})}. \quad (6.10)$$

Both Maddison \textit{et al.} \cite{maddison2016concrete} and Jang \textit{et al.} \cite{jang2016categorical} additionally add a temperature parameter, \textit{i.e.} $\ln \pi_k = \lambda \epsilon_k$, and show that for $\lambda \to 0$, Equation (6.10) tends to the real discrete distribution. However, for simplicity, we neglect the temperature $\lambda$ in our discussion.

The Bernoulli distribution is a special case of the discrete distribution. Letting $z \sim \text{Ber}(z; \theta)$, $z = 1$ means that

$$
\ln \theta + \epsilon_1 > \ln(1 - \theta) + \epsilon_2
$$

where we used Equation (6.9) with $\pi_1 = \theta$ being the probability of $z = 1$ and $\pi_2 = (1 - \theta)$ the probability of $z = 0$ and made the argmax explicit through the inequality. Maddison \textit{et al.} \cite{maddison2016concrete} argue that the difference of two Gumbel random variables $\epsilon_1 - \epsilon_2$ is distributed according to a Logistic distribution (also see \cite[Section 5]{bromley1985applied}) where we can sample from using

$$
\epsilon_1 - \epsilon_2 = \ln u - \ln(1 - u)
$$

with $u \sim U(0, 1)$. Thus,

$$
z = \begin{cases} 
1 & \ln u - \ln(1 - u) + \ln \theta - \ln(1 - \theta) > 0 \\
0 & \text{else}
\end{cases}
$$

which can be made differentiable using a soft thresholding operation such as the sigmoid function. With a change in notation, letting $\epsilon \sim U(0, 1)$ this results in

$$
z_i = g(y, \epsilon) = \sigma(\ln \epsilon - \ln(1 - \epsilon) + \ln \theta_i(y) - \ln(1 - \theta_i(y))) \quad (6.11)
$$

which is the final reparameterization trick used. The remaining framework stays unchanged; the Kullback-Leibler divergence is again computed analytically and the reconstruction error $-\ln p(y|z)$ depends on the exact form of $p(y|z)$ as discussed before.

### 6.3.1 Practical Considerations

The main difference between a Gaussian variational auto-encoder and the Bernoulli counterpart is the reparameterization trick and the corresponding Kullback-Leibler divergence. The latter can, again, be separated over $1 \leq i \leq Q$ and then be calculated analytically:

$$
\text{KL}(q(z_i|y)||p(z_i)) = \text{KL}(\text{Ber}(z_i; \theta_i(y))||\text{Ber}(z_i; 0.5))
\quad = \sum_{k \in \{0, 1\}} \ln \theta_i(y)^k(1 - \theta_i(y))^{1-k} - \ln 0.5^k0.5^{1-k} \quad (6.12)
$$

41
where we use $\text{Ber}(z_i; 0.5)$ as prior, i.e. our standard Bernoulli distribution. The gradient with respect to the parameters $\theta_i(y)$ is, as before, added during error backpropagation. This leads to the Bernoulli Kullback-Leibler divergence layer:

**Definition 6.7** A Bernoulli Kullback-Leibler divergence layer $KLD_{\text{Ber}}$ takes as input a tensor $\theta \in \mathbb{R}^{B \times Q}$ and computes the Kullback-Leibler divergence

$$\sum_{b=1}^{B} \sum_{i=1}^{Q} \text{KL}(\text{Ber}(z_{b,i}; \theta_{b,i}) | \text{Ber}(z_{b,i}; 0.5))$$

before passing the predicted $\theta$ on to the next layer (unchanged):

$$KLD_{\text{Ber}}(\theta) = \theta.$$

Again, this layer just makes the computation of the Kullback-Leibler divergence explicit and takes care of adding the corresponding derivative – which is easily derived from Equation (6.12) – during error backpropagation. The reparameterization trick as outlined in Equation (6.11) is wrapped in the following layer which needs to be followed by a sigmoid non-linearity layer to have the desired effect:

**Definition 6.8** A Bernoulli reparameterization layer $\text{repa}_{\text{Ber}}$ takes as input a tensor $\theta \in \mathbb{R}^{B \times Q}$ and computes

$$\text{repa}_{\text{Ber}}(\theta) = \ln \epsilon - \ln(1 - \epsilon) + \ln \theta - \ln(1 - \theta)$$

where $\epsilon \in \mathbb{R}^{B \times Q}$ with $\epsilon_i \sim U(0, 1)$.

The combination of both layers is analogous to the case of Gaussian variational auto-encoder except that a logistic sigmoid non-linearity layer needs to be added right after the reparameterization layer.

### 6.4 Discussion

Overall, we introduced variational auto-encoders for continuous latent variables as well as binary latent variables. In practice, variational auto-encoders are very similar to standard auto-encoders except that encoder and decoder predict probability distributions, i.e. $q(z|x)$ and $p(y|x)$. Additionally, the Kullback-Leibler divergence makes sure that the encoder, i.e. the recognition model $q(z|y)$, matches a pre-defined prior $p(z)$. Then, the variational auto-encoder has the advantage of providing a generative model, i.e. $p(y|z)$. For shape completion, we use variational auto-encoders to learn a shape prior allowing to easily constrain the space of considered shapes.
Chapter 7

Shape Inference

So far, we are able to use the shape set \( \mathcal{Y} \subseteq \mathbb{R}^{H \times W \times D} \) \( \approx \mathbb{R}^{R} \) from Problem 2.3 in order to learn a prior model \( p(y, z) \) over a possibly low dimensional latent space \( \mathcal{Z} = \mathbb{R}^{Q} \). To this end, we introduced variational auto-encoders where the generative model \( p(y|z) \) and the (approximate) recognition model \( q(z|y) \) are implemented using 3D convolutional neural networks. In our first approach to shape completion, we then intend to learn an inference model

\[
x \mapsto \tilde{z} \approx \arg\max_{z} p(y = x|z)p(z)
\]

using a dataset of observations \( x \in \mathbb{R}^{H \times W \times D} \). We call this approach amortized maximum likelihood because we do not consider the observations \( x \) independently, but understand the maximum likelihood objective as loss allowing to learn the deterministic embedding \( x \mapsto \tilde{z} \) in an unsupervised setting. Given the prior variational auto-encoder, this results in training a new encoder while keeping the pre-trained decoder fixed.

In the proposed extended variational auto-encoder, we instead understand the observation \( x \) as a random variable. For the corresponding joint distribution \( p(x, y, z) \), we are able to derive the evidence lower bound assuming that \( y \) and \( z \) are statistically independent given \( x \). Similar to the shape prior, we learn an approximate recognition model \( q(z|x) \) which can be understood as probabilistic embedding \( x \mapsto z \). In contrast to amortized maximum likelihood, the actual objective tying the observations \( x \) to the shapes \( y \) is hidden in a Kullback-Leibler divergence. Again, the model can be trained in an unsupervised fashion. In addition to training a new encoder, the extended variational auto-encoder also implements an observation model \( p(x|y) \) using a 3D convolutional neural network. Compared to amortized maximum likelihood, this potentially allows to explicitly integrate knowledge about the observation model, however results in longer training time – also because the embedding \( x \mapsto z \) is modeled probabilistically.

In the following, we first discuss a general maximum likelihood approach to shape completion. Originally, we then experimented with different losses to learn the embedding \( x \mapsto y \) in a non-probabilistic framework. This early approach is
discussed in detail in Appendix C. Here, we focus on the proposed amortized maximum likelihood framework which we discuss using both occupancy grids and signed distance functions as shape representations. Finally, we consider the proposed extended variational auto-encoder.

7.1 Maximum Likelihood

The negative log-likelihood corresponding to Equation (7.1) is given by

$$\arg\min_z - \ln p(y|z) - \ln p(z).$$

As $p(y|z)$ is a differentiable model, we can apply gradient descent after decomposing $p(y|z)$ over voxels:

$$- \ln p(y|z) = - \sum_{i=1}^{R} \ln p(y_i|z).$$

where we again flattened the representation of $y \in \mathbb{R}^{H \times W \times D} \approx \mathbb{R}^R$. Considering the actual observations $x_i$, we first discover that we do not necessarily have an observation $x_i$ for every voxel $i$. Formally, we write $x \in \{0, 1, \perp\}^R$ where $\perp$ indicates unknown values; in practice, we simply ignore the corresponding indices. As the probability $p(y_i = x_i|z)$ cannot be determined for $x_i = \perp$, we exclude them from the optimization problem:

$$\arg\min_z - \sum_{i=1, x_i \neq \perp}^{R} p(y_i|z) - \ln p(z).$$

As intuition, we assume that the prior $p(z, y)$ is strong enough that by constraining $p(y|z)$ only for few $y_i$ to specific values (through the observations $x_i \neq \perp$) will result in good shape predictions. This is a practical alternative to considering all $y_i$ with $x_i = \perp$ as hidden variables to optimize in addition to $z$.

7.1.1 Bernoulli Maximum Likelihood

As mentioned above we assume the observations to be binary, per voxel. This results in $p(y_i|z)$ being modeled as Bernoulli distribution. In particular, our generative model predicts Bernoulli probabilities $\theta_i(z)$ such that

$$p(y_i|z) = \text{Ber}(y_i; \theta_i(z)).$$

As observations are binary, as well, the probabilities $p(y_i = x_i|z)$ for $x_i \neq \perp$ can be evaluated directly; the negative log-likelihood objective can then be written as loss with respect to $z$:

$$\mathcal{L}_{\text{ML}}(z) = - \sum_{i=1, x_i \neq \perp}^{R} \ln \text{Ber}(y_i = x_i; \theta_i(z)) - \ln \mathcal{N}(z; 0, I_Q) \quad (7.2)$$
7.2. Amortized Maximum Likelihood

where we also substituted that \( p(z) \) is a unit Gaussian in \( \mathbb{R}^Q \). In practice, this simplifies to

\[
\mathcal{L}_{\text{ML}}(z) = - \sum_{i=1, x_i \neq \perp}^R (x_i \ln \theta_i(z) + (1 - x_i) \ln(1 - \theta_i(z))) + \text{const} + \frac{1}{2} \|z\|^2.
\]

such that the Gaussian prior (\( i.e. - \ln p(z) \|z\|^2 \)) represents a regularizer making sure that the latent code \( z \) does not deviate significantly from the unit Gaussian prior. We note that this is simply a composite function of the binary cross entropy error and the sum-of-squared error as introduced in Section 4.5.1.

7.1.2 Practical Considerations

To apply gradient descent to minimize \( \mathcal{L}_{\text{ML}} \) we need to be able to compute the gradient \( \nabla \mathcal{L}_{\text{ML}} \). If \( \theta_i(z) \) is modeled using a neural network – as in the case of variational auto-encoders – the error backpropagation algorithm can be used. Then, gradient descent with \( z^{(0)} = 0 \) iteratively performs updates

\[
z^{(t+1)} = z^{(t)} - \gamma \nabla \mathcal{L}_{\text{ML}}(z^{(t)})
\]

where, as discussed earlier, we might adapt the learning rate \( \gamma \) over time and also use a momentum parameter. Finally, \( z^{(T)} \) is used to derive the corresponding shape prediction. We also tried randomly initializing \( z^{(0)} \) and optimizing multiple, different \( z^{(0)} \) in parallel but found that this has no significant effect. We additionally experienced that choosing the learning rate \( \gamma \) (as well as the momentum parameter \( \beta \)) and the number of iterations \( T \) is not trivial and might vary significantly across observations.

7.2 Amortized Maximum Likelihood

By amortizing, \( i.e. \) learning, maximum likelihood, we intend to avoid the optimization problem required for inference in the previous section. To this end, we introduce a new, deterministic encoder \( z(x; w) \) intended to represent the mapping

\[
x \mapsto z(x; w) \approx \arg\max_z p(y = x|z)p(z),
\]

\( i.e. \) we intend to learn how to directly predict the maximum likelihood solutions. In practice, the encoder \( z(x; w) \) is also implemented using 3D convolutional neural networks following the architecture of the recognition model \( q(z|y) \) from the shape prior. Using the generative model from the shape-prior, \( i.e. p(y|z) \), the encoder \( z(x; w) \) can be trained by minimizing a loss derived from the maximum likelihood formulation. In contrast to the previous section, however, we also consider the case of signed distance functions as shape representation where \( p(y|z) \) is modeled using Gaussian distributions – this becomes problematic when evaluating \( p(y_i = x_i|z) \) because \( x_i \) is inherently binary (\( i.e. \) occupied or not occupied). Overall, this
Chapter 7. Shape Inference  

7.2. Amortized Maximum Likelihood

Figure 7.1: For implementing the negative log-likelihood \(-\ln p(y_i = x_i | z)\) using Gaussian, \(i.e.\) continuous, predictions \(y_i\), we can derive (signed) distance functions from the given, binary observations \(x_i\) for \(x_i \neq \perp\) using a distance transform. A reasonable loss would then compute the negative log-likelihood for all voxels in free space. However, in the case of noisy observations, even the ground truth shape will have a high negative log-likelihood because distance values are affected significantly by few noisy observations. To illustrate this phenomenon we show the ground truth shape, the noisy observations and the free space in the first three columns for two examples from our synthetic 2D dataset, see Appendix D. The remaining columns show the distance function of the ground truth shape and the noisy observations as well as the the free-space-masked absolute error from Equation (7.4).

leads to two losses, one for occupancy derived by assuming Bernoulli distributions and one for signed distance functions.

7.2.1 Bernoulli Amortized Maximum Likelihood

In the Bernoulli case, the loss reduces to Equation (7.2), except that the loss is with respect to the parameters \(w\) of the new encoder \(z(x; w)\):

\[
\mathcal{L}_{\text{AML,Ber}}(w) = - \sum_{i=1, x_i \neq \perp}^{R} \ln \text{Ber}(y_i = x_i; \theta_i(z)) - \ln \mathcal{N}(z; 0, I_2) \\
= - \sum_{i=1, x_i \neq \perp}^{R} (x_i \ln \theta_i(z) + (1 - x_i) \ln(1 - \theta_i(z))) + \text{const} + \frac{1}{2} \|z\|^2. \quad (7.3)
\]

where the probabilities \(\theta_i(z)\) are predicted by the fixed generative model, \(i.e.\) decoder, of the shape prior. As we are optimizing the parameters \(w\), this is equivalent to a binary cross entropy loss on the observed variables \(x_i \neq \perp\) and a quadratic regularizer on the predicted latent codes \(z\).

7.2.2 Gaussian Amortized Maximum Likelihood

When predicting signed distance functions, these are modeled using voxel-wise Gaussians with fixed variance, \(i.e.\)

\[
p(y_i | z) = \mathcal{N}(y_i; \mu_i(z), \sigma^2).
\]

Ideally, we would derive a signed distance function on our observations in order to directly use the negative log likelihood \(-\ln p(y_i = x_i | z)\) as loss. However, this leads to a problem illustrated in Figure 7.1 where the ground truth shape has
7.2. Amortized Maximum Likelihood

Figure 7.2: Illustration of a Gaussian distribution \( p(y) = \mathcal{N}(y; -0.75, 0.5^2) \) on the left where the marked area corresponds to the probability \( p(y \leq 0) \). In our case this refers to the probability of a voxel being occupied. The corresponding cumulative density function \( p(y' \leq y) \) according to Equation (7.5) is shown on the right.

very high error when defining an absolute error between the distance function \( df(x_p) \) of observed points \( x_p \) and the distance function \( df(y^*) \) of the ground truth shape \( y^* \) when considering voxels in free space only, i.e.

\[
\sum_{i=1}^{R} x_{f,i} \left| df_i(x_p) - df_i(y^*) \right|.
\] (7.4)

Here \( x_p \) defines the occupancy grid corresponding to observed points only, i.e. \( x_{p,i} = 1 \) for \( x_i = 1 \) and \( x_{p,i} = 0 \) otherwise. We also note that \( df(x_p) \) computes, for each voxel, the distance to the nearest occupied pixel \( x_{p,i} = 1 \). Similarly, \( x_f \) is defined as the occupancy grid corresponding to free space voxels, i.e. \( x_{f,i} = 1 \) for \( x_i = 0 \) and \( x_{f,i} = 0 \) otherwise. Of course, this method of deriving distance functions from the observations is not ideal; for example, other authors [SKC13] compute a distance function along the rays from the observed points. However, noisy observations are still problematic.

As of the above discussion we would like to use binary observations while still predicting a shape in signed distance function representation. Then, however, evaluating \( p(y_i = x_i | z) \) is not meaningful. Instead we need to introduce a transformation \( \theta_i(\mu_i) \) quantifying the probability of occupancy after having predicted a mean signed distance function value of \( \mu_i \). To this end, we take a closer look on the univariate Gaussian:

**Example 7.1** As defined in Definition 4.14, the univariate Gaussian distribution takes the form in Figure 7.2. Its cumulative density function is given as

\[
\mathcal{N}(y' \leq y; \mu, \sigma^2) = \int_{-\infty}^{y} \mathcal{N}(y'; \mu, \sigma^2) dy' = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{y - \mu}{\sigma \sqrt{2}} \right) \right).
\] (7.5)
and also shown in Figure 7.2. The error function \( \text{erf} \) is defined as

\[
\text{erf}(y) = \frac{1}{\sqrt{\pi}} \int_{-y}^{y} \exp(-y'^2) dy'.
\]

The cumulative density function quantifies the probability of a variable \( y' \) falling into a specific range – in our case \( y' \) falling into \((-\infty, y]\).

Using the Gaussian cumulative density function we want to determine the probability \( \theta_i(\mu_i) \) of occupancy. This process is illustrated in Figure 7.2; as negative values in a signed distance function refer to occupied voxels, we can use

\[
\theta_i(\mu_i) = \mathcal{N}(y' \leq 0; \mu_i, \sigma^2)
= \frac{1}{2} \left( 1 + \text{erf} \left( \frac{-\mu_i}{\sigma \sqrt{\pi}} \right) \right). \tag{7.6}
\]

Intuitively, we evaluate the cumulative density function at 0 giving the probability \( \mathcal{N}(y \leq 0; \mu_i(\mu), \sigma^2) \) which corresponds to the probability of occupancy. The negative log-likelihood can then be stated as

\[
\mathcal{L}_{\text{AML}}(w) = - \sum_{i=1, x_i \neq \perp}^{R} \ln \text{Ber}(y_i = x_i; \theta_i(\mu_i(\mu))) - \ln \mathcal{N}(z; 0, I_Q).
\]

The second term, the negative log-likelihood of the prior \( p(z) \) is of course only added once if the two losses \( \mathcal{L}_{\text{AML}} \) and \( \mathcal{L}_{\text{AML,Ber}} \) are combined. As before, the negative log-likelihood of the Bernoulli distributions \( \text{Ber}(y_i = x_i; \theta_i(\mu_i(\mu))) \) leads to the binary cross entropy error.

### 7.2.3 Practical Considerations

In practice, we can predict a shape \( y \) in both occupancy or signed distance function representation or use either of them. Compared to Figure 7.2, we generally use a small variance such as \( \ln \sigma^2 = -2 \Leftrightarrow \sigma^2 \approx 0.13533 \). In practice, the fixed variance merely scales the used sum-of-squared loss. As a result, we found the fixed variance to have negligible impact on training as long as it is not chosen too small, \( e.g. \sigma^2 > 0.01 \), or too large, \( e.g. \sigma^2 < 1 \). Then, for predicting a signed distance function, we add a post-processing layer which explicitly implements the transformation \( \theta(\mu(z)) \) from Equation (7.6) and allows to conveniently apply a binary cross entropy loss on top:
Definition 7.1 A Gaussian to Bernoulli layer \(g2b, \sigma\) takes as input a tensor \(\mu \in \mathbb{R}^{B \times C \times H \times W \times D}\) corresponding to an element-wise predicted mean (i.e. \(\mu_i(z)\)) of a signed distance function and computes a tensor of occupancy probabilities as

\[
(g2b, \sigma_i(z)) = \int_{-\infty}^{0} \mathcal{N}(y_i; \mu_i, \sigma_i^2) dy_i = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{-\mu_i}{\sigma_i \sqrt{\pi}} \right) \right).
\]

At this point we note that the derivative of the Gaussian to Bernoulli layer, which essentially corresponds to the derivative of the Gaussian cumulative density function, is a Gaussian distribution:

\[
\frac{\partial}{\partial \mu_i} \left( 1 + \text{erf} \left( \frac{-\mu_i}{\sigma_i \sqrt{\pi}} \right) \right) = \frac{\partial \text{erf}(t)}{\partial t} \bigg|_{t=-\frac{\mu_i}{\sigma_i \sqrt{\pi}}} = \frac{1}{2 \sqrt{2\pi} \sigma} \exp \left( -\frac{1}{2} \frac{(-\mu_i)^2}{\sigma^2} \right) = \mathcal{N}(0; \mu_i, \sigma^2).
\]

The error function \(\text{erf}\) is in practice approximated following [Abr74]\(^1\). If we would additionally predict the variances \(\sigma_i^2\), the derivatives would be more involved, but we did not see any practical benefit of doing so.

For both predicting occupancy and signed distance functions, we found that it is beneficial to weight the prior, e.g.

\[
\mathcal{L}_{AML}(w) = - \sum_{i=1, x_i \neq \perp}^{R} \ln \text{Ber}(y_i = x_i; \theta_i(z)) - \kappa \ln \mathcal{N}(z; 0, I_Q);
\]

especially on datasets with noise, it is important that the encoder \(z(x; w)\) learns to predict latent codes \(z\) that are likely under the unit Gaussian prior \(p(z)\). This prevents the encoder from getting distracted by noise which usually also leads to unlikely codes \(z\).

### 7.3 Extended Variational Auto-Encoder

The approaches discussed above are formulated in a maximum likelihood framework where we consider \(x_i\) to be actual observations of the random variables \(y_i\).

In contrast, we might also directly extend the graphical model in Figure 7.3a to also consider the observation \(x_i\) as random variable. In Figure 7.3b we illustrate a possible extension that we decided to investigate closer. The main idea is to model the observation process \(p(x|y)\) explicitly in order to deduce information about a possible \(y\). There are many possibilities to model \(p(x|y) - e.g.\) by learning it directly from samples \(\{(x_n, y_n^*)\}\) or by modeling the sensor. Here, we leave

\(^1\) We use code provided at [http://hewgill.com/picomath/lua/erf.lua.html](http://hewgill.com/picomath/lua/erf.lua.html).
Chapter 7. Shape Inference  7.3. Extended Variational Auto-Encoder

(a) Illustration of the graphical model of the original variational auto-encoder as discussed in Chapter 6. The generative model is shown on the right and governed by \( p(y, z) = p(y|z)p(z) \), while the recognition model is shown on the left and approximated by \( q(z|y) \).

(b) Illustration of the graphical model of the extended variational auto-encoder explicitly taking into account the observation \( x \). The generative model on the right assumes that first a shape \( y \) is generated through \( p(y|z) \) and the observation \( x \) is then obtained from the observation model \( p(x|y) \). In the inference model on the left the latent code \( z \) is inferred through \( q(z|x) \) – this can be seen as probabilistic embedding of observation \( x \) in the latent shape space. The (dashed) model \( q(y|x) \) is not learned but used to tie the observation \( x \) to the predicted shape \( y \).

Figure 7.3.: Comparison of the regular variational auto-encoder model in Figure 7.3a to learn a shape prior and the extended variational auto-encoder model to learn shape completion in Figure 7.3b.

These options for future work and model \( p(x|y) \) through a neural network without direct supervision by embedding \( p(x|y) \) in the framework of variational inference.

Following the introduction of variational inference in Section 6.1, the evidence lower bound for the model in Figure 7.3b can be written as

\[
-\text{KL}(q(y, z|x)p(y, z)) + \mathbb{E}_{q(y, z|x)}[\ln p(x|y)]
\]

This formulation treats both \( y \) and \( z \) as latent variables regarding the newly introduced random variable \( x \). The exact structure of \( q(y, z|x) \) as well as \( p(y, z) \) is not defined yet. For the latter we again use a pre-trained shape prior \( p(y, z) = p(y|z)p(z) \); for the former we have to make simplifying assumptions. In particular, we assume that \( y \) and \( z \) are statistically independent given \( x \):

\[
q(y, z|x) = q(y|x)q(z|x).
\]
Here, $q(z|x)$ is the mapping we want to learn; $p(y|x)$ decomposes into

$$q(y|x) = \prod_{i=1}^{R} q(y_i|x_i)$$

and can easily be modeled for those voxels $x_i$ where we have observations, i.e. $x_i \neq \perp$. Note that $q(y|x)$ is not learned, i.e. it is not modeled through a neural network – it is not part of the recognition model; however $q(y|x)$ allows us to integrate knowledge about the observations within the Kullback-Leibler divergence:

$$\text{KL}(q(y,z|x)p(y,z)) = \mathbb{E}_{q(y,z|x)} \left[ \ln \frac{q(y,z|x)}{p(y,z)} \right]$$

$$= \mathbb{E}_{q(y,z|x)} \left[ \ln \frac{q(y|x)q(z|x)}{p(y|z)p(y)} \right]$$

$$= \ln \frac{q(y|x)}{p(y|z)} + \mathbb{E}_{q(z|x)} \left[ \ln \frac{q(z|x)}{p(z)} \right]$$

$$= \text{KL}(q(y|x)p(y|z)) + \text{KL}(q(z|x)p(z))$$

Finally, the reconstruction error simplifies to

$$\mathbb{E}_{q(y,z|x)}[\ln p(x|y)] = \mathbb{E}_{q(y|x)}[\ln p(x|y)].$$

Then, the overall objective can then be written as

$$\mathcal{L}_{\text{EVAE}}(w) = \text{KL}(q(z|x)p(z)) + \text{KL}(q(y|x)p(y|z)) - \mathbb{E}_{q(y|x)}[\ln p(x|y)] \quad (7.8)$$

where the Kullback-Leibler divergence $\text{KL}(q(y|x)p(y|z))$ implicitly ties the observation $x$ to the possible shape $y$.

As before, both Kullback-Leibler divergences can be computed analytically. Only the reconstruction error $\mathbb{E}_{q(y|x)}[\ln p(x|y)]$ needs to be approximated using Monte-Carlo samples; for a specific sample $x_n$, this means

$$-\mathbb{E}_{q(y|x)}[\ln p(x|y)] = - \sum_{l=1}^{L} \ln p(x_n|y_{l,n})$$

with

$$y_{l,n} = g_y(z_{l,n}, \epsilon_{l,n}) \quad \text{and} \quad \epsilon_{l,n} \sim U(0, 1)^R$$

$$z_{l,n} = g_z(x_n, \epsilon'_{l,n}) \quad \text{and} \quad \epsilon'_{l,n} \sim \mathcal{N}(\epsilon; 0, I_Q)$$

and usually using $L = 1$ sample. Here, $g_y$ represents the Bernoulli reparameterization trick from Section 6.3 and $g_z$ the Gaussian equivalent from Section 6.2. This also implies that $y_i$ is modeled using a Bernoulli distribution, just like $x_i$, while $z$ is modeled using a Gaussian distribution. To make this explicit, we write

$$p(z) = \mathcal{N}(z; \mu(z), \text{diag}(\sigma^2(z)))$$

$$p(y_i|z) = \text{Ber}(y_i; \theta_i(z))$$

$$p(x_i|y) = \text{Ber}(x_i; \rho_i(y))$$
where the parameters $\mu$, $\sigma^2$ as well as $\theta_i$, $\rho_i$ are modeled using neural networks; in particular, $\theta_i(z)$ is taken from the pre-trained shape prior in order to restrict shape inference to reasonable shapes. Overall, this specifies the generative model of the extended variational auto-encoder as in Figure 7.3b on the right.

In Equation (7.8), only the Kullback-Leibler divergence $\text{KL}(q(y|x)\|p(y|z))$ as well as the recognition model $q(z|x)$ are left for discussion. The approximate posterior $q(z|x)$ is modeled analogously to $q(z|y)$. For the Kullback-Leibler divergence, we follow the maximum likelihood formulation and use:

$$q(y_i|x_i = 1) = \text{Ber}(y_i; 1)$$
$$q(y_i|x_i = 0) = \text{Ber}(y_i; 0)$$

while ignoring unobserved voxels $x_i = \perp$ as we assume a strong enough shape prior that is able to fill in the unobserved voxels. The Kullback-Leibler divergence $\text{KL}(q(y|x)\|p(y|z))$ then implicitly tries to adjust the predicted shape $y$ to the observation $x$.

### 7.3.1 Practical Considerations

We already introduced all the necessary tools for implementing the proposed model in Chapter 6. The latent code $z$ is still modeled as Gaussian; so, for the Kullback-Leibler divergence $\text{KL}(q(z|x)\|p(z))$ Equation (6.4) applies:

$$\text{KL}(q(z|x)\|p(z)) = \sum_{i=1}^{Q} \text{KL}(\mathcal{N}(z_i; \mu_i(x), \sigma_i(x)^2)\|\mathcal{N}(z_i; 0, 1))$$

$$= \sum_{i=1}^{Q} \left( -\frac{1}{2} \ln \sigma_i(x) + \frac{1}{2} \sigma_i(x)^2 + \frac{1}{2} \mu_i(x)^2 - \frac{1}{2} \right).$$

The voxels $y_i$ are modeled using Bernoulli distributions and $p(y|z)$ is taken from the shape prior; it is not fine-tuned but kept fixed. The Kullback-Leibler divergence $\text{KL}(q(y|x)\|p(y|z))$ follows Equation (6.12) and can be written as

$$\text{KL}(q(y|x)\|p(y|z)) = \sum_{i=1}^{R} \text{KL}(q(y_i|x_i)\|p(y_i|z))$$

$$= \sum_{i=1}^{R} \sum_{x_i \neq \perp \in \{0, 1\}} \left( \ln k \theta^0 1^{1-k} \begin{cases} x_i = 1 \\ x_i = 0 \end{cases} - \ln \theta_i(z)^k (1 - \theta_i(z))^{1-k} \right) \tag{7.9}$$

In practice, for numerical reasons, we need to add a small $\epsilon$-term to avoid taking the natural logarithm $\ln 0$, i.e. $\ln \theta^k (1 - \theta)^k$ becomes $\ln(\theta + \epsilon)^k (1 - \theta + \epsilon)^{1-k}$. The pre-trained generative model $p(y|z)$ is then followed by a Bernoulli Kullback-Leibler layer (cf. Definition 6.7) implementing Equation (7.9) and a Bernoulli reparameterization layer (cf. Definition 6.8). The observation model $p(x|y)$ is then learned using a 3D convolutional neural network as illustrated in Figure 7.4.
7.4. Discussion

Overall, we presented two approaches to tackle shape completion in a probabilistic framework based on a pre-trained shape prior and without explicit supervision.
Maximum likelihood, for example, is a natural approach and can be related to works such as [ESL16, ESL17] and [DPRR13] where shapes are modeled using PCA and GP-LVM, respectively, and shape completion is formulated as energy minimization problem. Using amortized maximum likelihood we intend to avoid the explicit optimization problem required for inference and additionally consider both occupancy and signed distance functions. We presented a formulation that allows to use both representations with binary observations — e.g. in the form of observed points and free space. Finally, the extended variational auto-encoder presents an attempt to integrate the observations into the probabilistic framework of variational inference. This model leaves several interesting questions for future work, e.g. how the observation model $p(x|y)$ can be modeled explicitly. We will see in Chapter 9, that both amortized maximum likelihood as well as the extended variational auto-encoder perform well and inherently optimize a similar objective.
Chapter 8

Data

For our formulation of shape completion, i.e. Problem 2.3, we require two sources of data: the shape set $\mathcal{Y}$ to learn the shape prior and the observations $\mathcal{X}$ in order to learn maximum likelihood or an extended variational auto-encoder. For research purposes, however, we created three synthetic datasets also including the ground truth shapes $\mathcal{Y}^*$ corresponding to $\mathcal{X}$ for evaluation: a dataset of rectangles in 2D, a dataset of cuboids in 3D and a dataset of cars from ShapeNet [CFG+15] in 3D. Besides providing both observations $\mathcal{X}$ and ground truth $\mathcal{Y}^*$, these datasets also reflect the progress made in the course of the thesis. For brevity, we will only present experiments in 3D; details on the 2D dataset can be found in Appendix D. Furthermore, we present experiments on the point clouds provided by KITTI [GLU12,GLSU13]. In this case, we use car models from ShapeNet as shape set $\mathcal{Y}$ and the provided 3D bounding boxes to extract the corresponding observations $\mathcal{X}$. However, on KITTI, we are not able to quantitatively evaluate the proposed approaches due to missing ground truth shapes $\mathcal{Y}^*$. In the following, we introduce our process of generating synthetic 3D datasets, i.e. the 3D cuboids dataset and the ShapeNet dataset, and extracting the observations from KITTI.

8.1 3D Example and ShapeNet

Our 3D datasets consist of arbitrarily scaled and rotated cuboids or cars; the cuboids are generated on-the-fly, while we use the car models from ShapeNet. The models are provided in the form of triangular meshes such that we need to consider the following processing steps: voxelization and filling of the shapes; rendering, backprojecting and voxelization of the observations; and finally post-processing to compute signed distance functions. This will provide us with both observations and ground truth shapes; to learn the shape prior, and simulate the realistic case, we then voxelize and fill a separate set of shapes. The overall process is also illustrated in Figure 8.1 and discussed in detail in the following.
Figure 8.1: Illustration of the data generation process on an example from the ShapeNet dataset. The original triangular mesh is shown in the top left. In (a), it is first simplified using the approach outlined in Algorithm 2. The simplified mesh is rendered, (b), from a random viewpoint. The simplified mesh is then voxelized using triangle-box intersections, (c), and the rendered depth map is used to generate and voxelize the observations, (d). Finally, in (e), the corresponding signed distance functions are computed. The top two rows show the filling process of the voxelized mesh, the next four rows show the observed points and observed free space and the corresponding distance functions; the last row shows the signed distance function of the filled mesh. Each row shows horizontal slices of the corresponding volumes, i.e. heights $11 + i$ for $0 \leq i < 10$.

8.1.1 Mesh Pre-Processing and Voxelization

We prefer to work with watertight meshes; for the 3D cuboids we have control about this as we automatically generate the meshes. In the case of ShapeNet
Algorithm 8.2 The semi-convex hull algorithm used in [GG15] to obtain watertight, simplified meshes. Details can be found in the text.

**Input:** triangular mesh \( \mathcal{M} 

**Output:** simplified mesh \( \mathcal{M}^{(T)} \)

1. draw sample points \( \mathcal{P} \) from \( \mathcal{M} \)
2. compute convex hull \( \mathcal{M}^{(0)} = (V^{(0)}, F^{(0)}) \) of \( \mathcal{M} \)
3. remesh \( \mathcal{M}^{(0)} \) using [FAKG10]
4. for \( t = 0 \) to \( T - 1 \)
5. \hspace{1em} if \( \mathcal{M} \not\subseteq \text{Vol}(V^{(t)}) \)
6. \hspace{2em} then find smallest \( \alpha > 0 \) such that \( \mathcal{P} \subseteq \text{Vol}((1 + \alpha)V^{(t)}) \)
7. \hspace{2em} \( V^{(t)} := (1 + \alpha)V^{(t)} \)
8. \hspace{2em} remesh \( \mathcal{M}^{(t)} \) using [FAKG10]
9. \hspace{2em} \( V^{(t+1)} = V^{(t)} - \gamma \nabla \mathcal{L}(V^{(t)}) \)
10. return \( \mathcal{M}^{(T)} \)

this is problematic. In addition, ShapeNet may contain very complex models where we need to deal with up to 100000 faces. Often, these models contain a level of detail that we are not interested in as it will be lost during voxelization, especially in low resolution. Therefore, we decided to follow [GG15] and compute a so-called semi-convex hull.

Algorithm 2 first samples a set of points \( \mathcal{P} \) from the given mesh \( \mathcal{M} = (V, F) \). Then, a convex hull is computed – a standard problem in computational geometry, see [CLRS09] or [dBCvKO08]. To reduce the number of initial vertices, the approach of [FAKG10] is used to remesh the initial mesh. The remeshed convex hull, \( \mathcal{M}^{(0)} \), is then iteratively refined by minimizing a loss

\[
\mathcal{L}(V^{(t)}) = \sum_{v \in V^{(t)}} \min_{p \in \mathcal{P}} \|v - p\|_2^2 + \sum_{(i,j) \in E(F^{(t)})} (\|v_i - v_j\|_2^2 - \mu)
\]

using gradient descent. Here, \( \mu \) is the mean edge length of the initial mesh \( \mathcal{M}^{(0)} \). A problem with this formulation is that at any iteration \( t \), the mesh \( \mathcal{M}^{(t)} \) might not contain the point set \( \mathcal{P} \) anymore. In this case, the vertices are rescaled as \( V^{(t)} := (1 + \alpha)V^{(t)} \) such that \( \mathcal{P} \subseteq \text{Vol}(V^{(t)}) \), i.e. the point set \( \mathcal{P} \) is contained in the volume spanned by \( V^{(t)} \) and the mesh \( \mathcal{M}^{(t)} \) is remeshed again. Results of simplification are shown in Figure 8.2.

After obtaining simple, watertight meshes, voxelization is performed using triangle-voxel intersection tests as e.g. proposed in [Ake05]\(^2\). In practice we scale, translate and pad all models to \([0, 1]^3\), which is then subdivided into \( H \times W \times D \) axis-aligned voxels. For the car models, care has to be taken to avoid skewing the models during scaling, translating and padding. Note that all axes are scaled

\(^1\) http://www.cvlibs.net/software/semi_convex_hull/.
\(^2\) We also use the corresponding code from http://fileadmin.cs.lth.se/cs/Personal/To mas_Akenine-Moller/code/.
Chapter 8. Data

8.1. 3D Example and ShapeNet

Figure 8.2.: Illustration of the employed mesh simplification approach, *i.e.* Algorithm 2, on manually selected examples from ShapeNet. These show the variety available in the dataset.

equally and we may consider slight random rotation, translation and scaling for data augmentation.

8.1.2 Mesh Filling

In order to obtain proper occupancy grids, *i.e.* also identify interior voxels, we use a flood filling/connected components algorithm [DST92] [Sze11, Section 3.3]. As we work with watertight meshes, interior and exterior of the shapes are clearly separated. For low resolutions, *e.g.* $32^3$, this approach works very well.

8.1.3 Mesh Rendering and Observation Voxelization

For simplicity, we use an OpenGL based MatLab renderer to obtain depth images from the pre-processed meshes. To this end, we use the code also used in [GW15, GG15]. Using the camera parameters used by OpenGL, the pixels in the depth image can be back-projected into 3D space. By thresholding the depth value, we know which pixels correspond to points on the mesh surface. By controlling the depth image resolution as well as focal length, we indirectly control the sparsity of the obtained point cloud. Using ray tracing we can additionally derive free space. The process is described in detail in the following.

**Definition 8.1** A (simplified) 2D projective camera is a tuple $(\mathcal{K}, \mathcal{R}, t)$ where

$$
\mathcal{K} = \begin{bmatrix}
    f_u & 0 & u \\
    0 & f_v & v \\
    0 & 0 & 1
\end{bmatrix}
$$

is the intrinsic camera matrix, $\mathcal{R} \in \mathbb{R}^{3 \times 3}$ is a rotation matrix and $t \in \mathbb{R}^3$ a translation vector. Here, $f_u$ and $f_v$ are focal lengths along horizontal and vertical direction, respectively, and $(u, v)^T$ defines the principal point of the 2D image.

\footnote{http://www.cvlibs.net/software/librender/}
implicitly also defining its resolution, $2u \times 2v$, assuming that $(u, v)^T$ represents the center pixel.

The overall projection matrix of a camera $(\mathcal{K}, \mathcal{R}, t)$ is given by $\mathcal{P} = \mathcal{K} [\mathcal{R} \ t] \in \mathbb{R}^{3 \times 4}$ and defines the projection of point $p = (p_1, p_2, p_3, 1)^T$ in homogeneous coordinates to the corresponding pixel

$$x = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ 1 \end{bmatrix} \quad \text{with} \quad \hat{x} = \mathcal{P} p.$$  

The inverse projection is defined as its pseudo-inverse [HZ06, Chapter 6]:

$$\mathcal{P}^+ = (\mathcal{P}^T \mathcal{P})^{-1} \mathcal{P}^T \in \mathbb{R}^{4 \times 3}$$

The ray corresponding to a pixel $x$ in homogeneous coordinates, i.e. $x = (x_1, x_2, 1)^T$, can then be obtained as

$$r = \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \hat{r}_3 \\ \hat{r}_4 \end{bmatrix} \quad \text{with} \quad \hat{r} = \mathcal{P}^+ x.$$  

The corresponding 3D point can be derived when knowing the depth $d$ corresponding to the ray $r$:

$$p = \begin{bmatrix} d \cdot \frac{r_1}{r_4} \\ d \cdot \frac{r_2}{r_4} \\ d \cdot \frac{r_3}{r_4} \end{bmatrix}.$$  

In practice, we can also work in the camera’s coordinate system such that $\mathcal{R} = I$ and $t = 0$. As a result, the inverse projection matrix simplifies to $\mathcal{P}^+ = \mathcal{K}^{-1}$. Then, we capture the shapes in different rotations around the (vertical) height axis—corresponding to different viewpoints of the camera. After back-projection, ray-voxel intersection tests [WBMS05] can be used to determine free space voxels and occupied voxels can be determined using point-voxel intersection tests. We rotate and translate the observations to $[0, 1]^3$ and use the exact same subdivision into $H \times W \times D$ axis-aligned voxels as before. This way, the observations are aligned with the corresponding ground truth shapes. To obtain free space voxels, we only consider rays to points on the corresponding shape. We refer to the result as partial free space as we do not consider rays corresponding to points on the background. This is reasonable as rays from background points cannot be used reliably on real data, e.g. on KITTI.

### 8.1.4 Noise

To simulate real conditions, we want to inject artificial noise. As we will see on the KITTI dataset, correctly simulating the Velodyne sensor and the corresponding

Figure 8.3.: Examples from the created datasets corresponding to a model from ShapeNet (top) and a cuboid (bottom). In both cases we show heights $8 + 2i$ for $0 \leq i < 8$, i.e. horizontal slices, illustrating the observed points, the computed partial free space and the ground truth shape. Additionally, we show 3D visualizations of the observed points and the corresponding shape; both in voxelized form.

noise is not trivial. We manually inspected many samples from the KITTI dataset and decided to define two noise parameters, $\lambda_{\text{hit}}$ and $\theta_{\text{ignore}}$. The former defines an exponential distribution [Bis06, Chapter 11]:

**Definition 8.2** Let $\epsilon \in \mathbb{R}$, $\epsilon \geq 0$, be a random variable. Then $\epsilon$ is distributed according to an exponential distribution, i.e. $\epsilon \sim \text{Exp}(\epsilon; \lambda)$, with parameter $\lambda$ if the probability density function is given by

$$p(\epsilon) = \lambda \exp(-\lambda \epsilon).$$

Following inverse transform sampling [Bis06, Chapter 11], we draw samples from this distribution using $u \sim U(0, 1)$ and

$$\epsilon = -\frac{\ln u}{\lambda}.$$

For each pixel in the depth image, we sample an error value $\epsilon \sim \text{Exp}(\epsilon; \lambda_{\text{hit}})$ from an exponential distribution and add the value to the actual depth value. This is reasonable, as $\epsilon$ will always be non-negative and $p(\epsilon)$ decreases exponentially for rising $\epsilon$. The probability $\theta_{\text{ignore}}$ defines how likely an observation is to be ignored. In this case, the depth value of the corresponding pixel is set to the maximum depth.

### 8.1.5 Discussion

Figure 8.2 illustrates the variety of car models covered. In addition, the simplified meshes are shown; we want to note that some meshes, e.g. from the Humvee on the right, are quite complex and this complexity is vastly reduced during simplification. However, this also incurs some loss of detail which we accept as voxelization is performed in low resolution anyway. Figure 8.3 shows the
corresponding voxelizations for both ShapeNet and the 3D cuboids dataset when using $\lambda_{\text{hit}} = 50$, $\theta_{\text{ignore}} = 0.1$, and $2u \times 2v = 24 \times 32$ as resolution for the depth image. The noise can best be observed in the horizontal slices of the sown volumes – especially considering the ignored rays. More examples can be found in Appendix D.

8.2 KITTI

The KITTI dataset is a standard dataset and benchmark for a variety of computer vision tasks. Beneath stereo image pairs, point clouds were captured from a moving vehicle using a 360° Velodyne LiDAR sensor. An example of a captured point cloud is shown in Figure 1.1. Annotations include – among others – 3D bounding boxes for all cars visible on the image plane (i.e. point clouds were not annotated in 360°). As discussed in Chapter 2, we tackle shape completion of an individual object. Therefore, we assume a 3D object detector to be given. Due to the limited scope of this thesis, we use the provided ground truth 3D bounding boxes for our experiments. These are first extracted, scaled and the corresponding points are voxelized. Free space is then computed using ray tracing.

8.2.1 Point Cloud Voxelization

Each ground truth 3D bounding box is provided in the form of its center, i.e. translation from the sensor’s center, the extents in terms of width, height and depth as well as the rotational angle along the (vertical) height axis. For voxelization, the bounding boxes are rotated to be axis aligned and then scaled to the unit cube, i.e. $[0, 1]^3$. Again, we make sure to scale all axes equally such that the observations do not get skewed.

8.2.2 Free Space Voxelization

To voxelize free space, we follow the same approach as before, i.e. ray tracing. Again, we consider partial free space as the Velodyne sensor has difficulties with reflective and transparent surfaces. In particular we found that many rays go through the annotated cars and hit points in the background. Considering partial free space, this effect is reduced by only tracing rays that correspond to points within the 3D bounding box. We then compute ray-box intersections between all rays and all voxels using the same subdivision into voxels as used before to avoid errors. Still, some points are prone to lie within the observed car – e.g. at the height of the windows – causing voxels to erroneously be labeled as free space.
Chapter 8. Data

8.2. KITTI

Figure 8.4.: Two examples as extracted from KITTI. Again, we show horizontal slices of the voxelized points and the corresponding free space together with 3D visualizations of the observed points and the corresponding shape.

8.2.3 Filtering

Using all the voxelized observations to learn shape completion is not realistic within the scope of this thesis – especially as many observations contain only very few observed points and erroneous free space. Therefore, we filtered the voxelized observations to obtain an easier dataset. First, we require that at least \( n_{1, \text{min}} \) points are observed and \( n_{0, \text{min}} \) voxels correspond to free space:

\[
\sum_{i=1}^{R} 1[x_i = 1] > n_{1, \text{min}} \quad \text{and} \quad \sum_{i=1}^{R} 1[x_i = 0] > n_{0, \text{min}};
\]

Second, we require the distance of the 3D bounding box to the Velodyne sensor to be less than \( t_{\text{max}} \). In practice, the three constraints ensure that the extracted dataset is manageable to tackle in the course of this thesis. Higher difficulties can then be obtained by lowering \( n_{1, \text{min}} \) and \( n_{0, \text{min}} \) and increasing \( t_{\text{max}} \), respectively. The exact constraints and some statistics are provided in Chapter 9 when conducting experiments on KITTI.

8.2.4 Discussion

Overall, it is fair to say that KITTI’s Velodyne data exhibits several distinct noise patterns – that we also tried to model in our synthetic datasets. The examples chosen in Figure 8.4 were manually selected to illustrate that some bounding boxes clearly depict cars. Still, even in this example, we notice some artifacts. For example, due to noise and discretization, observed points frequently lie inside the car. This is problematic as the corresponding free space derived by ray tracing is partly invalid. We also notice that this happens more frequently at the height of the car’s windows – sometimes the Velodyne’s rays hit other objects inside the car instead, e.g. seats. In Appendix D, we show further examples.

Chapter 9
Experiments

This section presents experimental results regarding the approaches to shape completion discussed in Chapter 7 on all datasets introduced in Chapter 8. Not all of the discussed formulations perform equally well. Originally, we first conducted experiments on our synthetic 2D rectangle dataset, see Appendix E, and found that maximum likelihood (ML), does not perform well. For clarity, we only present experiments on our synthetic 3D datasets, i.e. cuboids and cars from ShapeNet [CFG+15], as well as on real data, i.e. KITTI [GLU12, GLSU13]. Overall, we consider variational auto encoders (VAEs) for learning shape priors and amortized maximum likelihood (AML) as well as extended variational auto-encoders (EVAEs) for shape completion. Due to space constraints, the presented experiments are complemented by additional results in Appendix E. We first discuss the experimental setup before discussing experiments on 3D cuboids and cars as well as on KITTI.

9.1 Experimental Setup

We implemented all discussed approaches in the Torch1 deep learning framework. The implementations can be understood as prototypes; we did not optimize them with respect to runtime or memory consumption. Data pre-processing and generation as well as evaluation was performed in Python and C++ as described in Chapter 8. For all experiments we use a resolution of $H \times W \times D = 32^3$ and assume a uniform subdivision of $[0, 1]$ into $32^3$ voxels for voxelization. As already stated, we derive signed distance functions from the corresponding occupancy grids using distance transforms. Details on the artificially added noise as well as data augmentation is discussed in the corresponding sections. In the following we briefly describe the used architectures and evaluation metrics.

1 http://torch.ch/.
9.1.1 Architecture and Training

The architectures used for our experiments are kept simple. While we experimented with deeper and more complex architectures including skip connections, residual units \[HZRS16\] and inception-based architectures \[SLJ^*15,SVI^*16\], these changes had no significant influence. We follow the architectures illustrated in Figures 6.2 and 7.4 for VAEs and EVAEs, respectively. Encoder and decoder both consist of four stages of convolutional layers including batch normalization, ReLU non-linearity and max pooling/nearest neighbor upsampling. We follow discussions in the literature \[SVI^*16\] and increase the width of the network (i.e. the number of channels) whenever decreasing the spatial size of the feature maps and use \(3 \times 3 \times 3\) convolution kernels with zero padding and non-overlapping \(2 \times 2 \times 2\) windows for max pooling and nearest neighbor upsampling. For \(32 \times 32 \times 32\) we thus reduce the spatial size to \(2 \times 2 \times 2\) before computing the latent code. For both AML and EVAE, the new encoders, i.e. \(z(x; w)\) and \(q(z|x)\), are trained from scratch but follow the architecture of the shape prior. In this case, the corresponding generative model \(p(y|z)\) is always kept fixed. For AML, we additionally remove the fully connected layer predicting the variance of the latent code to obtain a deterministic encoder. For EVAE, the additional decoder \(p(x|y)\) consists of seven convolutional stages including batch normalization and ReLU non-linearities. When predicting occupancy, we use Sigmoid non-linearities; for predicting signed distance functions we use the identity, i.e. no non-linearity. We refer to Appendix E for training details.

9.1.2 Evaluation

For evaluation we resort to the absolute error between prediction and ground truth for both representations, i.e. occupancy and signed distance functions. For a prediction \(y \in \mathbb{R}^{H \times W \times D}\) and ground truth \(y^* \in \mathbb{R}^{H \times W \times D}\) we average over all spatial dimensions:

\[
\text{Abs}(y, y^*) = \frac{1}{HWD} \sum_{i_1=1}^{H} \sum_{i_2=1}^{W} \sum_{i_3=1}^{D} |y_{i_1i_2i_3} - y^*_{i_1i_2i_3}|
\]

We always report the average on the validation set (or on batches during training). We additionally use the absolute error after thresholding the predicted shapes, i.e. after obtaining proper occupancy grids. For occupancy grids, we threshold the predicted occupancy probabilities at 0.5 and for signed distance functions we threshold at 0 (here, negative values correspond to occupied voxels). These thresholds are also used for our 3D visualizations. We refer to the absolute error after thresholding as \(\text{Abs}_{\text{thresh}}\). Overall, the absolute error is easy to interpret, e.g. it provides a clear lower bound (which is \(\text{Abs} \geq 0\)), and comparable across datasets and methods.

Because the discussed approaches try to maximize the likelihood – or the corresponding evidence lower bound – during training, we would also like to con-
Table 9.1.: Overview of the generated datasets. We created datasets of three difficulties, easy, moderate and hard, which refer to increased noise and less observations. Details on the parameters are discussed in Section 8.1. Additionally, we report statistics such as the percentage of observed voxels, free space voxels and occupied voxels (of the ground truth shapes) over the training set used for shape inference.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>3D easy</th>
<th>3D moderate</th>
<th>3D hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Size (Prior/Inference)</td>
<td>10000/10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation Size</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution $H \times W \times D$</td>
<td>$32 \times 32 \times 32$</td>
<td>$24 \times 32$</td>
<td></td>
</tr>
<tr>
<td>Resolution $2u \times 2v$</td>
<td>$48 \times 64$</td>
<td>$24 \times 32$</td>
<td></td>
</tr>
<tr>
<td>Noise $\lambda_{hit}$</td>
<td>0</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Noise $\theta_{ignore}$</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Observed Voxels (Inference Training Set)</td>
<td>1.43%</td>
<td>0.53%</td>
<td>0.48%</td>
</tr>
<tr>
<td>Free Space Voxels (Inference Training Set)</td>
<td>10.73%</td>
<td>7.97%</td>
<td>8.32%</td>
</tr>
<tr>
<td>Occupied Voxels (Inference Training Set)</td>
<td>16.24%</td>
<td>16.17%</td>
<td>16.21%</td>
</tr>
</tbody>
</table>

9.2 3D Example

We start with experiments on our synthetic 3D cuboids dataset. Here, we are able to present both quantitative and qualitative results on a controlled, simple dataset. To this end, we created datasets of three difficulties following the procedure in Chapter 8: easy, hard and moderate with details in Table 9.1. With rising difficulty, less observations are provided and the underlying noise increases. The hard case is supposed to represent real conditions as found on KITTI. Examples for all three difficulties can be found in Appendix 8. In Table 9.1 we additionally report some basic statistics, e.g. the percentage of occupied voxels considered the negative log-likelihood as measure. However, we found the negative log-likelihood to be unsuited for evaluation. First, the negative log-likelihood is harder to interpret as it strongly depends on the model (e.g. VAE or EVAE). In particular, besides the reconstruction loss, all models also include (possibly weighted) prior terms – e.g. in form of Kullback-Leibler divergences or the negative log-likelihood corresponding to the prior $p(z)$. Furthermore, for EVAE, the reconstruction loss does not represent the objective we are actually trying to optimize for shape completion. Second, across datasets, the negative log-likelihood depends on the number of observed voxels. Thus, performance of the proposed shape completion approaches cannot be compared to the reconstruction performance of the shape prior – which would be a natural baseline. Third, the negative log-likelihood on Bernoulli observations is inherently skewed towards “unsure” predictions; meaning that the negative log-likelihood prefers unsure predictions over very certain predictions with few mistakes (see Appendix E for an example). This is reasonable during training where we explicitly want to model uncertainty, but hinders fair evaluation. Overall, we decided not to report any negative log-likelihoods.
Figure 9.1.: Training curves for a VAE with $Q = 15$ trained on the 3D cuboids dataset. We show the training loss, i.e. $\mathcal{L}_{\text{BCE}} + \text{KL}$, on the training (train) and validation set (val) as well as the corresponding absolute error Abs on the left. Statistics corresponding to the latent space, particularly, the average $\mu$ of the predicted means and the corresponding standard deviation $\sqrt{\text{Var}[\mu]}$ as well as the average of the predicted standard deviations $\exp(\frac{1}{2})$ are shown on the right. For the latter we also refer to Equations (6.6), (6.7) and (6.8) for details.

to give an impression of how trivial predictions would perform or the percentage of observed and free space voxels to indicate what level of supervision is available. We start with discussing the VAE shape prior before proceeding to the problem of shape completion. As we found ML to perform poorly in the 2D case, we exclude experiments on 3D data.

### 9.2.1 Shape Prior

For the shape prior, the size $Q$ of the latent space is crucial. We found that in practice a suitable size can be determined by monitoring the training progress of the VAE for different sizes $Q$. For the following discussion, we determined $Q = 15$ to be suitable and the corresponding training curves are shown in Figure 9.1. An important cue to judge $Q$ is the latent space, i.e., whether learning the latent space converged. This is usually be indicated by low predicted log-variances and the statistics of the predicted means slowly resembling a unit Gaussian (i.e. zero mean and unit variance). Additionally, the obtained reconstruction error can be used as indicator. For $Q = 5$, for example, the absolute error does not fall below $\text{Abs} \approx 0.04$. Considering that, only 16.2% of the voxels are occupied (cf. Table 9.1) this error is still very large. For $Q = 15$ and above, in contrast, the error reduces to $\text{Abs} \approx 0.0054$ or lower. Finally, we also consider random samples utilizing the generative model. As can be seen in Figures 9.2 and 9.3, the random samples look appropriate and mostly resemble cuboids. Note that due to rotations, the cuboids might not appear rectangular when showing horizontal slices of the corresponding volumes. Overall, we did not exploit all possibilities regarding hyper parameters and training time, but are satisfied with the obtained performance using $Q = 15$.

On 2D examples, we realized that predicting both occupancy and signed distance functions is beneficial compared to predicting signed distance functions only. On 3D, we make a similar observation and show qualitative results in Figure 9.2. Again, we use $Q = 15$, and achieve an absolute error of $\text{Abs} \approx 0.0064$
Figure 9.2.: Qualitative results for the trained VAE shape prior with $Q = 15$ on the 3D cuboids dataset. We consider two models; one trained on occupancy only and one trained on both occupancy and signed distance functions. In the first case, we show reconstruction results on the left and random samples on the right. For the latter case, we show only random samples for both modalities. In all cases we show horizontal slices, i.e. heights $8 + 2i$ for $0 \leq i < 8$. For random samples in the occupancy only case, we complement the results with 3D visualizations in Figure 9.3.

Figure 9.3.: Three random example of the VAE shape prior with $Q = 15$ on the 3D cuboids dataset trained on occupancy only. In all three cases, we show two different viewpoints. We find that all three examples resemble cuboids. We also note that the VAE has no difficulties predicting sharp corners and edges.

for occupancy and $\text{Abs} \approx 0.071$ for signed distance functions. After thresholding the predicted representations to obtain occupancy grids (at 0.5 for occupancy probabilities and at 0 for signed distance functions), the absolute error drops to
Chapter 9. Experiments

9.2. 3D Example

Figure 9.4.: Training curves for AML using occupancy only and a VAE prior with $Q = 15$ on the 3D cuboids dataset, specifically the moderate case. Again, we show the quantities as in Figure 9.1. We want to highlight that training takes place in the first few iterations; afterwards, training stagnates mostly.

Figure 9.5.: Absolute error Abs and its thresholded variant $\text{Abs}_{\text{thresh}}$, i.e. the absolute error on thresholded predictions, for a comparison between the VAE prior, AML and EVAE on occupancy only and AML predicting both occupancy and signed distance functions (indicated as occ+sdf). In each case, the left bar represents results on occupancy; the right bar corresponds to results on signed distance functions (if applicable). The comparison to the VAE prior provides a possible lower bound on the performance.

Abs$_{\text{thresh}} \approx 0.0045$ and Abs$_{\text{thresh}} \approx 0.0053$, respectively. This means that both representations can be used to derive low-error occupancy grids. Random samples also look suitable and clearly depict cuboids in most cases; however, we find the samples to be slightly less sharp compared to predicting occupancy only.

9.2.2 Amortized Maximum Likelihood

For AML, we mainly present experiments on moderate and hard difficulties as we found that AML performs very well on moderate difficulty. First of all, we note that the weight $\kappa$ on the negative log-likelihood of the prior, i.e. $-\ln p(z)$, is crucial for successful training. Figure 9.4 shows training curves for the moderate case when using $\kappa = 15$. Overall, the network achieves an absolute error of Abs $\approx 0.036$. If the weight $\kappa$ would not be large enough, the network would quickly deviate from the unit Gaussian prior. This can be observed when monitoring the latent space, i.e. the observed statistics $\sqrt{\text{Var}[\mu]}$ and $\overline{\mu}$ deviate significantly from the unit Gaussian prior. If $\kappa$ is chosen too large, the prior “collapses”, i.e. $\sqrt{\text{Var}[\mu]}$ approaches zero. If weighted correctly, the prior takes care of enforcing the unit
9.2. 3D Example

Figure 9.6: Qualitative results for AML on the 3D cuboids dataset with a VAE prior and $Q = 15$. We show results on moderate and hard difficulties in comparison with EVAE. In all cases we show horizontal slices of the volumes, i.e., heights $8 + 2i$ for $0 \leq i < 8$, for two samples samples, each showing the observed points, the partial free space, the target shape as well as the predicted shape and the corresponding error. For AML and the hard case we additionally illustrate the weights $\rho_i$ that were used for both AML and EVAE.

Gaussian in the first few iterations. We found that matching the prior becomes important as a big portion of learning takes place in the early iterations. This can also be seen in Figure 9.4; it seems that AML needs only few epochs to “learn” inference. Experimentally, we found that $\kappa = 15$ performs well for the moderate case, however, $\kappa = 30$ is necessary for the hard case. Unfortunately, we did not find any rule of thumb for setting $\kappa$ but needed to resort to trial and error. Overall, we find that training AML gets is tricky in 3D; experimenting with hyper-parameters becomes more important.

Figure 9.5 compares prior performance with the obtained absolute errors on easy, moderate and hard difficulties. Qualitative results for the moderate case
Chapter 9. Experiments

9.2. 3D Example

Figure 9.7: Qualitative results for AML using a VAE prior with $Q = 15$ trained on both occupancy and signed distance functions. Results correspond to the hard case. We show two samples and both modalities. In all cases we show horizontal slices as in Figure 9.6 corresponding to the observed points, the partial free space, the target shape as well as the predicted shape and its error. We selected two examples illustrating that the model resorts to blob-like “standard” shapes.

can be found in Figure 9.6. AML performs reasonably well on moderate difficulty. However, we found performance to be strongly influenced by the noisy free space observations in the hard case. Especially the ignored rays cause problems. After closer investigation we resorted to a simple approach to avoid these difficulties: we weight observations $x_i = 0$ corresponding to free space by

$$
\rho_i = 1 - \frac{\sum_{m=1}^{M} y_{m,i}}{M}, \quad Y = \{y_m\}_{m=1}^{M} \subseteq \{0, 1\}^R.
$$

(9.1)

The weight $\rho_i$ can be interpreted as free space statistics, i.e. the likelihood that voxel $i$ is not occupied over the prior training set. This concept is also illustrated in Figure 9.6. We additionally experimented with using $\rho_i^\lambda$, $\lambda \in (0, 1)$ and determined $\lambda = 0.5$ to work well. As a result, we are able to obtain nearly equal performance on moderate and hard difficulties. Overall, this discussion also shows the influence of individually weighting voxels in order to cope noisy observations.

Figure 9.5 also shows results obtained when predicting both occupancy and signed distance functions. The corresponding qualitative results for the hard can be found in Figure 9.7. We found that in both the moderate and the hard case, the predicted shapes are slightly larger than the target shapes. Specifically, the model appears to resort to “standard”, blob-like shapes which be close to a mean shape. We already now that learning and predicting signed distance functions
is harder compared to occupancy only. The blob-like predictions could also be explained by choosing the weight $\kappa$ too large, implicitly constraining the model to shapes close to the mean shape. Unfortunately, an extensive investigation and hyper-parameter tuning was not possible within the limited time-frame of this thesis.

9.2.3 Extended Variational Auto-Encoder

For the EVAE, we obtain similar results as presented above, see Figure 9.5. For example, in the easy case, an absolute error Abs of $\sim 0.034$ is achieved – this is only slightly worse than AML. For the moderate and hard cases, errors of $\sim 0.042$ and $\sim 0.057$ are achieved. This is worse than AML; however, we also note that we did not spend as much time tuning hyper parameters and we might have underestimating required training time which is more relevant for the EVAE as it includes significantly more parameters. We also note that the observed performance is in contrast to the 2D case, see Appendix E, where EVAE slightly outperformed AML which also indicates that training time is relevant – in the 2D case, it seems, we allocated enough training time. For the hard case, examples can be found in Figure 9.6. We can also see that AML and EVAE give very similar results, apart from the fact that EVAE slightly under-estimates the true cuboids’ size. However, this is not surprising as both approaches optimize a similar objective, only that it is wrapped in a Kullback-Leibler divergence in the case of EVAE. We intend to perform further experiments regarding EVAE in future work; however, given the provided evidence, we prefer AML due to lower training times and slightly better performance.

9.2.4 Discussion

First of all, we find that both the shape prior, i.e. a VAE, as well as the shape inference models are hard to train on 3D data. Tuning hyper-parameters is important and made difficult by the long training times, even in low resolutions such as $32^3$. Specifically, we found that enforcing the shape prior is crucial. For AML, for example, we increased the weight on the negative log-likelihood $-\ln p(z)$ in order to obtain reasonable results. Additionally, to cope with the hard case, we weighted free space voxels individually by their likelihood to actually correspond to free space on the prior training set. Overall, we demonstrated that VAEs are able to learn appropriate shape priors using both occupancy and signed distance functions. Regarding shape completion, we showed that both AML and EVAE give reasonable performance while AML performs slightly better than EVAE. We also found that AML and EVAE give very similar results – which is reasonable considering the theoretic background. In the end, we are satisfied by the presented experiments and proceed to the more complicated ShapeNet dataset.
Chapter 9. Experiments

9.3. ShapeNet

<table>
<thead>
<tr>
<th></th>
<th>3D easy</th>
<th>3D moderate</th>
<th>3D hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Size (Prior/Inference)</td>
<td>20496/20496</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation Size</td>
<td>4550</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution $H \times W \times D$</td>
<td>$32 \times 32 \times 32$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution $2u \times 2v$</td>
<td>$48 \times 64$</td>
<td>$24 \times 32$</td>
<td></td>
</tr>
<tr>
<td>Noise $\lambda_{hit}$</td>
<td>0</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Noise $\theta_{ignore}$</td>
<td>0</td>
<td>0</td>
<td>0.075</td>
</tr>
<tr>
<td>Observed Voxels</td>
<td>0.62%</td>
<td>0.31%</td>
<td>0.304%</td>
</tr>
<tr>
<td>Free Space Voxels</td>
<td>3.91%</td>
<td>3.75%</td>
<td>4.37%</td>
</tr>
<tr>
<td>Observed Voxels</td>
<td>5.54%</td>
<td>5.51%</td>
<td>5.52%</td>
</tr>
</tbody>
</table>

Table 9.2.: Overview of the generated datasets. We created datasets of three difficulties, easy, moderate and hard, which refer to increased noise and less observations. Details on the parameters are discussed in Section 8.1.

9.3 ShapeNet

ShapeNet [CFG+15] is our first dataset comprising realistic objects, in particular cars. Later, ShapeNet will also be used to train the shape prior for shape completion on KITTI [GLU12, GLSU13]. We first manually discarded 262 of the 3514 simplified meshes that could not be automatically scale and rotated to $[0,1]^3$ for further processing. We split the remaining models into two training sets – for prior and inference – and a validation set corresponding to the fractions 0.45 : 0.45 : 0.1. On each model, we apply seven random transformations including slight scaling, rotation and translation to the meshes; additionally, we flip every variant. Overall, we obtain the datasets outlined in Table 9.2; examples can be found in Appendix D.

9.3.1 Shape Prior

For the VAE prior, we again use $Q = 15$. From training, we might conclude that the ShapeNet dataset is slightly easier to learn. In particular, considering

![Training curves for a VAE with Q = 15 trained on occupancy only on the ShapeNet dataset. In comparison with Figure 9.1, we plot training (train) and validation (val) loss, i.e. $\mathcal{L}_{BCE} + \text{KL}$, the corresponding absolute error Abs as well as latent space statistics $\mu$, $\sqrt{\text{Var}[\mu]}$ and $\exp\left(\frac{1}{2}I\right)$ corresponding to the average of the predicted means, the corresponding standard deviation and the average of the predicted standard deviations.](image)
9.3. ShapeNet

Figure 9.9.: Qualitative results considering reconstruction and random samples for a VAE prior, $Q = 15$, learned on ShapeNet using occupancy only (top) and both occupancy and signed distance functions. In the latter case we only show random samples in both modalities. For reconstructions we show the target shape, the prediction as well as the corresponding error. In all cases we resort to showing horizontal slices as done before. 3D visualizations of the random samples can be found in Figures 9.10 and 9.11.

Figure 9.10.: 3D visualizations of random samples obtained from a VAE prior trained with $Q = 15$ and occupancy only on the ShapeNet dataset. We show three distinct samples using two viewpoints each. The samples can easily be recognized as cars, although details seem to be missing. However, this is also due to the low resolution of $32^3$ used.

Figure 9.8 we notice that both reconstruction and the latent space are learned faster. However, it is hard to say whether ShapeNet is inherently easier than the 3D cuboids dataset. On the one hand, we only consider slight rotations around
all axes; on the other hand, the cars show more intra-class variation compared to cuboids. Overall, the model achieves an absolute error of $\text{Abs} \approx 0.0073$ and $\text{Abs}_{\text{thresh}} \approx 0.0048$ after thresholding. However, we note that only $\sim 5.54\%$ of the voxels are occupied in the first place (cf. Table 9.2). We also notice that the model is less certain considering the random samples in Figures 9.9 and 9.10, i.e. predictions appear less sharp. Especially regarding the wheels and the roofs, the model has difficulties. Overall, we are satisfied by the obtained performance – for experiments on KITTI, we will also have more training data.

On ShapeNet, too, we trained a prior model on both occupancy and signed distance functions. For the first time, we also demonstrate why we would prefer to work with signed distance functions: it is possible to derive triangular meshes at sub-voxel accuracy, e.g. using the marching cubes algorithm [LC87]\textsuperscript{2}. Again, we use $Q = 15$ to achieve a thresholded absolute error $\text{Abs}_{\text{thresh}} \approx 0.0048$ (occupancy) and $\text{Abs}_{\text{thresh}} \approx 0.0053$ (signed distance function), respectively. We show meshes corresponding to random samples in Figure 9.11. In addition, in Figure 9.9, we show both modalities, also to illustrate the involved uncertainty. Overall, we find that the random samples are appropriate. While some random samples appear rather weird, it is not hard to imagine that they depict cars. We also want to stress that the meshes look rather smooth although the signed distance functions the model was trained on were originally derived from the occupancy grids. We assume this to be the result of the probabilistic formulation, i.e. of training a VAE. In particular, the encoder predicts a Gaussian distribution where all samples need to result in a proper reconstruction. This forces the model the smoothly interpolate between shapes.

9.3.2 Amortized Maximum Likelihood

For AML, we follow the procedure of the 3D cuboids dataset; in Figure 9.12 we plot quantitative results for the easy, moderate and hard cases. Again, it is important to keep in mind that only roughly 5.54\% of the voxels are occupied. Thus, results for the moderate and hard cases seem quite poor: $\text{Abs} \approx 0.023$ and $\text{Abs} \approx 0.027$, respectively. This can also be observed when considering qualitative results in Figures 9.13 and 9.14 showing results for the hard case. We found that in many case, the predictions are very uncertain. When considering

\textsuperscript{2} We use the implementation by Pablo Márquez Neila available at https://github.com/pmn eila/PyMCubes.
9.3. ShapeNet

Figure 9.12.: Absolute error Abs and its thresholded variant Abs\text{\textsubscript{thresh}}, \textit{i.e.} the absolute error on thresholded predictions, comparing AML and EVAE with the supervised baseline and the reconstruction performance of the shape prior. For the VAE shape prior, we report the reconstruction performance for training on occupancy only as well as on both occupancy and signed distance functions. For AML, we also consider both modalities. For EVAE and the supervised baseline, we present results on occupancy only. For the latter, we refer to Section 9.3.4 for details. In all cases, the left bar represents results on occupancy; the right bar corresponds to results on signed distance functions (if applicable).

Figure 9.13.: Qualitative results of AML for hard difficulty of the ShapeNet dataset. We show results for occupancy only in comparison with the supervised baseline. On top, we show horizontal slices of the volumes corresponding to the observed points, the partial free space, the target shape, the predicted shape and the corresponding error. On the bottom, \textit{i.e.} for the baseline, we only show the target shape, the predicted shape and the corresponding error. Again, we show heights $8 + 2i$ for $0 \leq i < 8$. 3D visualizations of these results can be found in Figure 9.14.

3D visualizations this is stressed even more due to the thresholding. For the moderate case, in contrast, results look more plausible in many cases. We are not sure whether these observations can be attributed to the prior model – which could \textit{e.g.} not be trained long enough – or the variation in the dataset that allows these models. It might also be beneficial to enforce the negative log-likelihood on the prior $p(z)$ more stringent – this assumes that the latent space is “more reliable” close to 0 than towards the tail of the Gaussian. This could prevent
shape inference from learning unlikely shapes and potentially reduce the influence of weight initialization and stochastic training in the first few epochs. Overall, we see potential for improvement by tuning hyper-parameters and longer training.

When predicting both occupancy and signed distance functions, we are not able to achieve a performance comparable to the occupancy only case. Especially in the moderate and hard cases, performance drops significantly from Abs $\approx 0.026$ to Abs $\approx 0.032$ or higher (on occupancy). We cannot say whether this is a drawback of signed distance functions as modality, due to the shape prior or because of the training procedure. In contrast to the occupancy only case, we did not spend as much time tuning parameters. An exact comparison can be found in Figure 9.12 while we show qualitative results in Figures 9.15 and 9.16. Considering the qualitative results we can, however, appreciate the smoothing effect on the prior; the meshed predictions look very appealing – in contrast to the meshed targets. Here it gets apparent that our signed distance functions are derived from the occupancy grids. We can also see the drop in performance; some predictions do not match the targets as well as before; especially as AML consistently underestimates the size of the cars. The problems with predicting signed distance functions seem to be consistent across datasets and models; overall, we believe that an alternative representation might be easier to learn, e.g. normalized, discretized or just replacing the logarithm.

### 9.3.3 Extended Variational Auto Encoder

Although EVAE performed slightly worse than AML on the 3D cuboids dataset, we still include quantitative results in Figure 9.12. On ShapeNet, the performance difference between AML and EVAE becomes more pronounced. However, we also

---

**Figure 9.14:** 3D visualizations for comparing AML and the supervised baseline on hard difficulty of the ShapeNet dataset. Results were obtained using occupancy only. As can be seen, AML has difficulties with the roofs; additionally, the first example occurs to the predicted with flipped orientation as our ShapeNet dataset also includes flipped variants.
9.3. ShapeNet

Chapter 9. Experiments

Figure 9.15.: Qualitative results for AML using both occupancy and signed distance functions. We show two examples from the hard dataset for both modalities. In both cases we show slices of the volumes corresponding to the observed points, the partial free space, the target shape as well as the predicted shape and its error. Again, we show heights $8 + 2i$ for $0 \leq i < 8$. Triangular meshes corresponding the the predicted signed distance functions can be found in Figure 9.16.

Figure 9.16.: Qualitative results for AML after using marching cubes to derive meshes from the predictions and the targets. For the targets, we can clearly see that the used signed distance functions are derived from the occupancy grids. The predictions, in contrast are more smooth, however, consistently underestimate the size of the car.

want to note that, again, we did not put as much effort into tuning parameters and training compared to AML. Specifically, we suspect that the weight on the Kullback-Leibler divergences may make a significant difference. Unfortunately, we were unable to investigate this problem further. Still, we showed, that the framework is also applicable to real-world objects – even if only the easy case results in appropriate performance.

9.3.4 Supervised Baseline

For ShapeNet, we also prepared a supervised baseline. For a fair comparison, we used the shape prior architecture with $Q = 15$, to learn the mapping $x_n \rightarrow y_n^*$ directly from the synthetic data. We did not notice a significant difference be-
between training the exact same architecture versus removing the Kullback-Leibler divergence and the corresponding reparameterization layer; for fairness, we followed the former approach. We considered the hard case only. Later, we will also evaluate how well the learned model generalizes to KITTI. Figure 9.12 shows that supervision is able to get closer to the reconstruction performance of the shape prior, with an absolute error of roughly $\text{Abs} \approx 0.014$ thereby outperforming all other presented approaches for shape completion. Qualitative results are shown in Figures 9.13 and 9.14 in comparison with AML. Overall, the supervised baseline could potentially also outperform AML on KITTI, given that the used observation model resembles KITTI’s Velodyne sensor closely enough. Overall, it is not surprising that the supervised baseline outperforms AML considering that AML only uses a fraction, in particular 4.06% in the moderate case, of the information during training.

9.3.5 Discussion

Overall, the results obtained on ShapeNet are not convincing in all cases. Especially on moderate and hard difficulty, shape completion appears to be significantly more challenging than on the 3D cuboids dataset. Unfortunately, limited time prevented us from conducting more experiments regarding both prior and shape completion, e.g. to investigate the influence of training time, architectural changes and hyper-parameters. Still, the presented experiments show that the proposed approach, especially AML but also EVAE in a limited setting, are able to learn shape completion under difficult conditions. Although we were not able to reach supervised performance, it is still surprising what is possible under weak supervision when relying on strong shape priors. We discuss possible future experiments based on the above observations in detail in Section 10.1.

9.4 KITTI

Using ShapeNet [CFG+15] we are able to learn a shape prior enabling us to perform shape completion on KITTI [GLU12, GLSU13]. On KITTI, however, we do not have access to ground truth shapes. We found that manual annotation (e.g. following [MG15]) would go far beyond the time frame of this thesis. Still, we intend to provide insightful experiments thereby, again, focusing on AML using both modalities, i.e. occupancy and signed distance functions. Our goal is to demonstrate that AML is able to predict reasonable shapes given the noisy observations from KITTI’s Velodyne sensor.

Following the discussion in Section 8.2, we filtered the provided ground truth 3D bounding boxes using $n_{1,\text{min}} = 150$, $n_{0,\text{min}} = 1500$ and $t_{\text{max}} = 30$. This means, that we require at least 150 voxels to be observed as occupied and 1500 observed voxels to correspond to free space. Additionally, we only consider bounding boxes within $30m$ of the sensor. Overall, we obtained 1928 distinct car observations.
which we split into a training set with 1714 and a validation with 214 samples. On average, \( \sim 0.634\% \) of voxels are observed points; \( \sim 5.98\% \) of voxels are free space. In this regard, the dataset might be slightly easier than the hard ShapeNet dataset, where only \( 0.304\% \) of voxels are observed as being occupied. In spite of these strict requirements regarding the number of observed voxels, we find the extracted dataset to be particularly challenging due to the corresponding noise patterns. We show additional examples of the dataset in Appendix D.

### 9.4.1 Amortized Maximum Likelihood

Using AML, we are able to obtain reasonable shape completions from the noisy observations. As shape prior, we re-trained the model used for ShapeNet on all 40992 samples from both training sets, cf. Table 9.2. In Figure 9.17 we first show the observations, i.e. observed points and free space, together with the shape predictions for several examples from the validation set. We also show the influence of using the weights from Equation (9.1) as well as training the prior on the combined ShapeNet training sets. As expected, using the weighted loss has significant influence on the quality of the predicted shapes. In addition, we also notice the benefit of having more training data for the prior. For the remaining discussion we always use the weights as well as the stronger prior. In Figure 9.18 we show 3D visualizations corresponding to the voxelized observations and the predicted shapes. We find that the predicted shapes match the observed...
Figure 9.18: Comparison of AML and the supervised baseline on KITTI; here, AML uses occupancy only and we show the observed points and the predicted shapes in voxelized form. For both, we show 2 distinct viewpoints. For AML we show two examples. For the latter, we also show the predicted shape using the supervised baseline to illustrate that AML still misses details, e.g. along the root.

points rather well and clearly depict cars. However, the predictions still miss a considerable level of detail around the wheels and the roof. This can be seen as the trade-off between noise robustness, integrated through a strong prior, and detail-orientation. Surprisingly, the predicted shapes look better than on the hard ShapeNet-based dataset, i.e. Figure 9.14. This confirms our intuition that the observations extracted from KITTI are slightly easier. Overall, AML is able to
correctly predict orientation and rough shape of the observed cars but also leaves room for improvement regarding details.

We also used a ShapeNet prior trained on both occupancy and signed distance functions. However, we found the results to be slightly worse compared to ShapeNet. As before, we suspect that longer training times and fine-tuning the hyper parameters would be necessary to improve results. Nevertheless, Figure 9.19 shows some examples of the predictions both in the form of occupancy grids and meshes. As in the hard case on ShapeNet, the prior seems to favor small and thin cars. This tendency is emphasized by the meshes derived using marching cubes and might be explained by the uncertainty involved when reconstructing details such as wheels and roof. Again, we can conclude that signed distance functions, while being advantageous for deriving meshes, are more difficult to learn in a weakly-supervised setting.

### 9.4.2 Supervised Baseline

We also investigated to which extent the supervised baseline from Section 9.3.4 is able to generalize to KITTI. Figure 9.18 shows qualitative results when applying the learned model to KITTI observations. As can be seen, the model predicts reasonable cars. In contrast to AML, the supervised model has no troubles predicting details along or the roof or wheels. For some samples, however, the predictions look very similar. Overall, the supervised baseline appears to predict slightly more coherent shapes; but an exact evaluation and comparison is not possible without ground truth annotations. Overall, we find that there does not seem to be a significant gap between AML and the supervised baseline in terms of the visual quality of the corresponding predictions.
9.4.3 Discussion

In the end, we can conclude that AML is able to perform shape completion even under real conditions as illustrated on KITTI. We also found that the supervised baseline generalizes surprisingly well. However, compared to the supervised baseline, AML requires significantly less supervision. We find that using occupancy grids only, we are able to recover slightly more detail compared to signed distance functions as shape representation. The latter, however, allows to derive comparably smooth meshes. As discussed before, we were not able to explore the full design space regarding architectures, hyper-parameters and training. We expect AML using signed distance functions to show improved performance with increased training time and optimized hyper parameters; however, we can only leave these experiments for future work. Additionally, we could only judge results qualitatively. While quantitative results are provided for ShapeNet, we do not believe that these can be transferred one-to-one to KITTI. In particular, we think the observation model used for ShapeNet does not completely match the true observation model underlying KITTI. In conclusion, we are satisfied by the demonstrated shape completion performance on KITTI and are looking forward to future experiments in order to improve the proposed approach.
Chapter 10

Conclusion

In the introduction, we presented shape completion as the problem of reconstructing an individual object given a partial observation, e.g. from a single view. We argued that shape priors play a crucial role in shape perception in general [Piz07, Piz10] and shape completion in specific. We also outlined the recent success in learning shape models using deep generative models, e.g. in [GFRG16, BLRW16, WSK+15, WZX+16], and the use of shape priors for data-driven shape completion [DPRR13, ESL16, ESL17]. Learning-based approaches to shape completion, in contrast, directly learn shape completion in a supervised setting [SM17, DQN16, SGF16, FSG16, REM+16, RUBG17], most of them using deep neural networks. While these approaches allow efficient inference, i.e. shape completion is “just a forward pass” in the trained network, their applicability is limited by the requirement of annotated training data. Data-driven approaches are directly applicable to real data by posing shape completion – and thereby inference – as energy minimization. We hypothesized that using generative shape models enables us to learn shape completion in an unsupervised setting. As result, shape completion would still be efficiently performed by a trained network while avoiding the need of annotated datasets.

With the goal to learn shape completion of cars on KITTI [GLU12, GLSU13], we proposed and implemented two probabilistic frameworks in order to experimentally test our hypothesis. In particular, based on a variational auto-encoder as shape prior, we posed shape completion as maximum likelihood problem over the learned latent space – in the spirit of e.g. [ESL16]. Following the idea of amortized inference, we then trained an encoder to directly predict maximum likelihood solutions, i.e. learning an embedding of observations within the latent shape space, by interpreting the maximum likelihood objective as unsupervised loss. As alternative approach, we also considered directly integrating the observations as random variable within the latent variable model, i.e. within the variational auto-encoder. Considering the corresponding joint distribution allowed us to derive an evidence lower bound and train an extended variational auto-encoder implicitly also learning shape completion. Both approaches can
be trained on KITTI using cars from ShapeNet [CFG+15] for the shape prior. Furthermore, shape completion, i.e. inference, is efficiently done by the trained network. Experimentally, we showed that both approaches optimize a similar objective. On a synthetically generated, ShapeNet-based dataset we showed that both are able to compete with a fully-supervised baseline, both quantitatively and qualitatively. On KITTI, we demonstrated that the proposed approaches are, indeed, applicable to real data and can learn shape completion in an unsupervised setting.

In conclusion, the presented experiments are in favor of our hypothesis that strong shape priors allow to learn shape completion without supervision. We believe this to be an important first step in reasoning about shapes as well as scenes in a weakly-supervised manner by explicitly using strong shape priors that can be learned on synthetic data. Finally, the presented results also pose new questions going beyond the scope of this work – some of which we will discuss in the following.

10.1 Future Work

We believe that the presented work poses many interesting new research questions. Within the proposed frameworks we are e.g. interested in replacing the used ground truth 3D bounding boxes with 3D detections from a state-of-the-art object detector or extending the presented experiments to consider true signed distance functions allowing to derive more accurate and detailed meshes. Considering the extended variational auto-encoder, we can imagine directly learning the observation model from synthetic data or explicitly modeling real sensors, e.g. KITTI’s Velodyne sensor. It might also be worthwhile to investigate the limitations of the proposed approaches in more details. For example, how many observations we need to learn proper shape completion and whether the prior is able to express ambiguity in terms of uncertainty within the predictions. In comparison to the baseline it is also interesting to consider additional advantages of learning shape completion on real data, e.g. by additionally learning color (given enough observations) or a figure-ground segmentation of the provided observations. To further improve the ability to predict more detailed shapes, we might also consider related work for learning in higher resolutions, e.g. octree-based approaches [RUG16, RUBG17, TDB17] or PointNet-based architectures [QSMG16, FSG16, QYSG17]. As we are currently also limited to one object category, we would like to extend the proposed frameworks to consider multiple object categories at ones, e.g. on other real datasets such as SUNRGBD [SLX15] and thereby also experiment with different modalities (i.e. RGB-D images from Kinect-like sensors compared to KITTI’s Velodyne sensor).
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Conference/Journal</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>[IAMB17]</td>
<td>D. J. Im, S. Ahn, R. Memisevic, and Y. Bengio</td>
<td>Denoising criterion for variational auto-encoding framework</td>
<td>AAAI Conference on Artificial Intelligence</td>
<td>2017</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>


Appendix A

Shape Representation

To complement the rough notion of watertight meshes as closed surfaces with clearly defined interior and exterior, we provide the corresponding mathematical background for clarification.

A.1 Watertight Meshes

We follow [BKP+10, Section 1.3], [Ede03] and [Gib10, Chapter 3] to first introduce the necessary terminology.

Definition A.1 (i) A self intersection is an intersection of two faces of the same mesh.

(ii) A non-manifold edge has more than two incident faces.

(iii) The star of a vertex is the union of all its incident faces.

(iv) A non-manifold vertex is a vertex where the corresponding star is not connected when removing the vertex.

(v) A mesh is 2-manifold if it does contain neither self intersections, nor non-manifold edges, nor non-manifold vertices.

Illustrations of these somewhat abstract definitions can be found in [BKP+10, Figure 1.6]. In general, 2-manifold meshes are preferable to arbitrary meshes as many algorithms and applications are not applicable to non-manifold meshes [BKP+10]. In our case, however, the definition of 2-manifold meshes is only motivated by the need to formally define watertight meshes (which are sometimes also referred to as closed meshes). Intuitively, the only constraint missing from 2-manifold meshes is a notion of “closedness”, i.e. a clear interior and exterior. This becomes apparent when considering the definition of a non-manifold edge – which also allows edges with only one incident faces, so-called boundary edges.

Definition A.2 A 2-manifold mesh is called watertight if each edge has exactly two incident faces, i.e. no boundary edges exist.
Appendix A. Shape Representation

A.1. Watertight Meshes

The above definitions, while appearing abstract, are also useful in practice. In software, e.g. in MeshLab\textsuperscript{1}, it is easy to identify and label non-manifold vertices, edges as well as boundary edges to help design and work with triangular meshes.

\textsuperscript{1} http://www.meshlab.net/.
Appendix B

Shape Prior

We complement the discussion of using variational auto-encoders as shape prior with a simple, linear alternative: probabilistic principal component analysis (PCA). Originally, we used probabilistic PCA to perform experiments on our 2D dataset as presented in Appendix E.

B.1 Probabilistic Principal Component Analysis

Again, we assume a flattened version $y \in \mathbb{R}^R \simeq \mathbb{R}^{H \times W \times D}$ for simplicity. We first remind the reader of general, non-probabilistic PCA following [Bis06, Section 12.1]:

**Example B.1** The goal of PCA is to find a linear mapping from $y \in \mathbb{R}^R$ to a lower-dimensional latent code $z \in \mathbb{R}^Q$ that captures as much variance as possible. Considering a one-dimensional latent space $z \in \mathbb{R}$ ($Q = 1$), we are looking for a vector $u \in \mathbb{R}^R$ that maximizes the variance captured in $z$:

$$\text{Var}[z] = \text{Var}[u^T y].$$

Defining the mean $\mu$ as

$$\mu = \frac{1}{M} \sum_{m=1}^{M} y_m,$$

(B.1)

this can be written as

$$\mathbb{E}[u^T (y - \mu)(y - \mu)^T u] = u^T \Sigma u$$

(B.2)

with $\Sigma$ being the corresponding covariance matrix. As $\Sigma$ is positive semi-definite, its eigenvalues are all real and positive [MN99, Section 1.13, Theorem 4 and Theorem 8]. Because the length of $u$ does not matter, we require $\|u\|_2 = 1$ and can choose it as the eigenvector corresponding to the largest eigenvalue $\lambda$. Then

$$u^T \Sigma u = \lambda u^T u = \lambda,$$

and $\text{Var}[z]$ is maximal.
The above idea can be generalized to \( Q \)-dimensional latent spaces. Then the linear mapping \( U \in \mathbb{R}^{R \times Q} \) is found by computing the eigenvalue decomposition \([\text{MN99, Section 1.14, Theorem 13}]\) of \( \Sigma \):

\[
\Sigma = V \Lambda V^T \tag{B.3}
\]

where \( \Lambda \in \mathbb{R}^{R \times R} \) is a diagonal matrix containing the eigenvalues – sorted from largest to smallest eigenvalue – and \( V \in \mathbb{R}^{R \times R} \) is an orthogonal matrix containing the corresponding eigenvectors. Taking \( U \) as the first \( Q \) eigenvectors (corresponding to the \( Q \) largest eigenvalues), i.e.

\[
U = V_Q := \begin{bmatrix} v_1 & \ldots & v_Q \end{bmatrix},
\]

yields the linear mapping maximizing the variance and thereby also minimizing the reconstruction error \([\text{Bis06, Section 12.1}]\).

The outline from the above example can be stated more precise in terms of an encoding transformation and a decoding transformation – which will correspond to the recognition model and generative model in probabilistic PCA:

**Definition B.1** Given data \( \mathcal{Y} = \{y_1, \ldots, y_M\} \subseteq \mathbb{R}^R \) with mean \( \mu \) and covariance matrix \( \Sigma \) as defined according to Equations (B.1) and (B.2), we define

\[
U = V_Q := \begin{bmatrix} v_1 & \ldots & v_Q \end{bmatrix} \in \mathbb{R}^{R \times Q}
\]

using the first \( Q \) eigenvectors \( v_1, \ldots, v_Q \) obtained from the eigenvalue decomposition \( \Sigma = V \Lambda V^T \). Then, PCA defines an encoding transformation

\[
z = U^T(y - \mu)
\]

and a decoding transformation

\[
\tilde{y} = U z + \mu.
\]

The problem with general PCA is that no generative model is included. Probabilistic PCA wraps a probabilistic interpretation around the linear encoding and decoding transformations. Following \([\text{Bis06, Section 12.2}]\) and \([\text{TB99}]\), we still assume a linear model

\[
y = U z + \mu + \epsilon
\]

with \( \epsilon \sim \mathcal{N}(\epsilon; 0, \sigma^2 I_R) \) and \( U \in \mathbb{R}^{R \times Q} \). With a unit Gaussian prior \( p(z) = \mathcal{N}(z; 0, I_Q) \), this implicitly defines a generative model: sample \( z \sim p(z) \) and

\[
y \sim p(y|z) = \mathcal{N}(y; \mu, U z + \mu, \sigma^2 I_R). \tag{B.4}
\]

Given the generative model, we also need the recognition model, i.e. the posterior \( p(z|y) \), representing the encoding transformation. This can be easily derived using the following result \([\text{Bis06, Section 2.3}]\):
Lemma B.1 Given Gaussian distributions \( p(z) \) and \( p(y|z) \) with parameters
\[
p(z) = \mathcal{N}(z; 0, I) \\
p(y|z) = \mathcal{N}(y; Uz + \mu, \sigma^2 I)
\]
then the posterior \( p(z|y) \) is given as
\[
p(z|y) = \mathcal{N}(z; S^{-1}U^T(y - \mu), \sigma^{-2}S^{-1})
\]
with \( S = U^TU + \sigma^2 I_Q \).

Proof: See [Bis06, Section 2.3].

Determining the parameters \( U, \mu \) and \( \sigma^2 \) would involve maximizing the likelihood
\[
p(y) = \int p(y|z)p(z)dz.
\]
As all involved distributions are Gaussians, the marginalization is again a Gaussian [Bis06, Section 2.3] with mean
\[
\mathbb{E}[y] = \mathbb{E}[Uz + \mu + \epsilon] = \mu
\]
and covariance matrix
\[
\text{Cov}[y] = \text{Cov}[Uz + \mu + \epsilon, Uz + \mu + \epsilon] \\
= \text{Cov}[Uz + \epsilon, Uz + \epsilon] \\
= \mathbb{E}[(Uz + \epsilon)(Uz + \epsilon)^T] \\
= \mathbb{E}[Uz^TU^T] + \mathbb{E}[\epsilon\epsilon^T] = UU^T + \sigma^2 I_R =: S
\]
Here, we used that \( p(z) \) is a standard Gaussian with zero mean and unit variance and \( p(y|z) \) takes the form in Equation (B.4). Maximizing the likelihood is equivalent to minimizing the negative log-likelihood:
\[
\mathcal{L}(U, \mu, \sigma^2) = - \sum_{m=1}^{M} \ln \mathcal{N}(y_m | \mu, UU^T + \sigma^2 I_R) \\
= \text{const} + \frac{M}{2} \ln |S| - \frac{1}{2} \sum_{m=1}^{M} (y_m - \mu)^T S^{-1} (y_m - \mu).
\]
Considering the gradient with respect to \( \mu \) and solving for \( \nabla_\mu \mathcal{L} = 0 \) yields:
\[
\nabla_\mu \mathcal{L} = \sum_{m=1}^{M} (y_m - \mu)S^{-1} = 0 \iff \mu = \frac{1}{N} \sum_{m=1}^{M} y_m.
\]
Regarding $\Sigma$, it is easier to first rewrite the log-likelihood
\[
\mathcal{L}(U, \mu, \sigma^2) = \text{const} + \frac{M}{2} \ln |S| + \frac{1}{2} \text{Tr} \left( \sum_{m=1}^{M} (y_m - \mu)(y_m - \mu)^T S^{-1} \right)
\]
\[
= \text{const} + \frac{M}{2} \left( \ln |S| + \text{Tr}(\Sigma S^{-1}) \right)
\]
where $\Sigma$ is the data covariance matrix:
\[
\Sigma := \frac{1}{M} \sum_{m=1}^{M} (y_m - \mu)(y_m - \mu)^T.
\]
Taking the derivative with respect to $\Sigma$ is more involved and requires some well-known matrix derivative identities [MN99, Section 8]:

**Lemma B.2** Let $S, \Sigma$ be square, symmetric matrices; then it holds
\[
\frac{\partial \ln |S|}{\partial S} = S^{-T} = S^{-1}
\]
\[
\frac{\partial \text{Tr}(\Sigma S^{-1})}{\partial S} = -(S^{-1} \Sigma S^{-1})^T = -S^{-1} \Sigma S^{-1}
\]

**Proof:** A proof of the first identity can be found in [MN99, Section 8.3, Theorem 1]; the second identity follows from [MN99, Section 8.2; Section 8.4, Theorem 3].

With the above lemma we can now use the chain rule to derive
\[
\nabla_S \mathcal{L} = M \left[ S^{-1} U - S^{-1} \Sigma S^{-1} U \right] \overset{!}{=} 0
\]
which leads to
\[
U = \Sigma S^{-1} U. \tag{B.5}
\]
At this point, there are three different cases. The first, $U = 0$ is trivial and not informative. The second, $S = \Sigma$, implies that the observed covariance is exact which is undesirable in the presence of additive noise [TB99]. Therefore, the third case is the most interesting one: $U \neq 0$ and $S \neq \Sigma$. Considering the singular value decomposition
\[
U = V \Lambda V^T
\]
(note that $\Lambda$ holds the singular values here, in contrast to Equation (B.3)), using $S = U U^T + \sigma^2 I_R$ and substituting into Equation (B.5):
\[
V \Lambda V^T = \Sigma (V \Lambda V^T (V \Lambda V^T)^T + \sigma^2 I)^{-1} V \Lambda V^T
\]
\[
V \Lambda = \Sigma (V \Lambda^2 V^T + \sigma^2 I)^{-1} V \Lambda
\]
\[
V (\Lambda^2 + \sigma^2 I) = \Sigma.
\]
From the last identity, it follows that with $\Lambda = \text{diag}(\lambda_i)$ and for a specific $\lambda_i \neq 0$ it needs to hold
\[
\Sigma v_i = (\sigma^2 + \lambda_i^2)v_i
\]
meaning that $v_i$ is an eigenvector of $\Sigma$, i.e. the data covariance matrix, with corresponding eigenvalue $(\sigma^2 + \lambda_i^2)^{\frac{1}{2}}$. A solution for $U$ might therefore be
\[
U = V_Q(\Lambda_Q - \sigma^2 I_Q)^{\frac{1}{2}}.
\]
Note that this solution is not unique; for any orthogonal matrix $U'$, $UU'$ is a solution, as well. Now only $\sigma^2$ is left to be determined. However, as we merely approximate $\sigma^2$ in practice and it is less relevant to understand the general idea of probabilistic PCA, we refer to [TB99]. Overall, the derivation leads to:

**Definition B.2** Given data $Y = \{y_1, \ldots, y_M\} \subseteq \mathbb{R}^R$ with mean $\mu$ and covariance matrix $\Sigma$, we define $U = V_Q(\Lambda_Q - \sigma^2 I_Q)^{\frac{1}{2}}$ with $\Sigma = V \Lambda V^T$ being the eigenvalue decomposition of $\Sigma$, and
\[
\sigma^2 = \frac{1}{R-Q} \sum_{i=Q+1}^{R} \lambda_i
\]
where $\lambda_1, \ldots, \lambda_R$ are the eigenvalues of $\Sigma$ in decreasing order. Then, probabilistic PCA defines as recognition model
\[
p(z|y) = \mathcal{N}(z; S^{-1}U^T(y - \mu), \sigma^{-2}S^{-1})
\]
with $S = UU^T + \sigma^2 I_R$ and as generative model
\[
p(z) = \mathcal{N}(z; 0, I_Q)
\]
\[
p(y|z) = \mathcal{N}(y; Uz + \mu, S).
\]

**B.1.1 Practical Considerations**

Due to a simple result from linear algebra non-probabilistic PCA as introduced in Example B.1 can be implemented in tow different ways; following [MN99, Section 1.16]:

**Lemma B.3** Let
\[
Y = \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} \in \mathbb{R}^{R \times M}
\]
be the data matrix, $\mu$ the corresponding mean and $\Sigma$ the covariance matrix. Then, $\Sigma = YY^T$ where $Y$ is the centered data matrix
\[
\bar{Y} = \begin{bmatrix} y_1 - \mu & \cdots & y_M - \mu \end{bmatrix}
\]
Then, the singular value decomposition of $\bar{Y} = V \Lambda V^T$ leads to the eigenvalue decomposition
\[
\Sigma = YY^T = V \Lambda^2 V^T,
\]
meaning that the singular values of $\overline{Y}$ are the square roots of the eigenvalues of $\Sigma$.

Proof: The result follows directly from

$$\Sigma = \overline{Y}\overline{Y}^T = V\Lambda V^T (V\Lambda V^T) = V\Lambda^2 V^T.$$ 

More details can be found in [MN99, Section 1.16].

The lemma implies that we can use either the eigenvalue decomposition of the covariance matrix $\Sigma = V\Lambda V^T$, or the singular value decomposition of the centered data matrix $\overline{Y} = V\Lambda^2 V^T$. Usually, the latter approach is faster and more memory efficient as it avoids explicitly computing the covariance matrix. For non-probabilistic PCA, only the first $Q$ eigenvalues and eigenvectors are required; efficient algorithms for this case are available, e.g. see [GL13]. For probabilistic PCA, all eigenvalues are required in order to compute $\sigma^2$. In practice, for large data matrices (e.g. $R$ and $M$ in the order of ten thousands), we compute the $Q' > Q$ largest eigenvalues. Pick the $Q$ largest eigenvalues and eigenvectors to form $U$ and the remaining to approximate $\sigma^2$. 

106
Appendix C

Shape Inference

Here, we briefly describe an intermediate approach that was used for various experiments in the course of this thesis. Specifically it builds upon the idea of avoiding the optimization problem of general maximum likelihood which is require for shape inference by learning, i.e. amortizing, the optimization problem. However, this approach does not fit our probabilistic framework and performs poorly in practice. Therefore, we only give a rough overview. Some of the experiments can additionally be found in Appendix E.

C.1 Non-Probabilistic Approach

The maximum likelihood approach presented in Chapter 7 has two significant disadvantages. First, we do not learn anything from performing inference, i.e. shape inference is performed for each observation \( x \) independently. And second, shape inference involves minimizing a complex, non-linear objective, in particular the negative log-likelihood (specifically when using variational auto-encoders as shape priors). In order to avoid the explicit optimization problem required for shape inference, we intend to directly learn a deterministic mapping \( x \rightarrow z \), i.e. a new encoder \( z(x; w) \), which can be trained by defining losses on the shape \( y \) corresponding to the estimated latent code, e.g. \( y = \mathbb{E}_{p(y|x)}[y] \). We define two losses intended to express how close the observed points, i.e. \( x_i = 1 \), are to the predicted shape and how far the predicted shape reaches into free space, i.e. where \( x_i = 0 \). Additionally, we use the negative log-likelihood \( -\ln p(z) \) to force the encoder to predict likely shapes. The overall loss can then be written as

\[
\mathcal{L}_{DL}(w) = \mathcal{L}_{DL,0}(w) + \mathcal{L}_{DL,1}(w) - \ln p(z)
\]

where \( w \) refers to the encoder’s parameters. We note that the decoder is kept fixed (the parameters are not updated). The loss \( \mathcal{L}_{DL,0} \) corresponds to observations \( x_i = 0 \), i.e. free space, and \( \mathcal{L}_{DL,1} \) corresponds to observations \( x_i = 1 \), i.e. the observed points. Both are introduced in the follow sections.
Appendix C. Shape Inference

C.1. Non-Probabilistic Approach

Figure C.1.: Illustration of the distance function approximation as discussed in Section C.1.2.1 for $T = 3, 5, 7$ iterations (from left to right). In each case, two samples are shown, including the original binary shape, the true distance function, the approximation and the error.

C.1.1 Free Space Loss $\mathcal{L}_{DL,0}$

For the free space loss, we want to penalize voxels of the predicted shape $y$ that lie inside free space. We intend to penalize voxels stronger if they reach farther into free space. We consider the tensor $x_f \in \mathbb{R}^R$ with $x_{f,i} = 1$ if $x_i = 0$ and $x_{f,i} = 0$ otherwise – i.e. $x_f$ is an occupancy representation of which voxels are free space. Then we define

$$\mathcal{L}_{DL,0}(w) = \sum_{i,x_{f,i}=1} y_i \text{df}_i(1 - x_f). \quad \text{(C.1)}$$

Here, df is the distance function operation from Remark 5.1. In practice, the loss can easily be implemented by pre-computing the distance function $\text{df}(1 - x_{n,f})$ for every sample $x_n$.

C.1.2 Point Loss $\mathcal{L}_{DL,1}$

The point loss is similar to the free space loss in that we want to encourage observed points lying close to the predicted shape. Therefore, we would like to use

$$\mathcal{L}_{DL,1}(w) = \sum_{i,x_{p,i}=1} x_{p,i} \text{df}_i(y). \quad \text{(C.2)}$$

where $x_p \in \mathbb{R}^R$ is defined by $x_{p,i} = 1$ for $x_i = 1$ and $x_{p,i} = 0$ otherwise. However, the distance function $\text{df}(y)$ is not well defined for $y_i \in [0, 1]$ which is why we need to threshold $y_i$ in practice (e.g. at 0.5). Additionally, we need to compute the gradient $\nabla_y \mathcal{L}_{DL,1}$ which – by the chain rule – involves the differentiation of the distance function operation $\text{df}(y)$ with respect to $y$. 

108
In the ideal case, we would predict both occupancy and a signed distance functions, avoiding the problem of having to differentiate through $\text{df}(y)$. For early experiments, however, we decided to approximate $\text{df}(y)$ by a succession of convolutional layers allowing to compute $\nabla_y \mathcal{L}_{DL,1}$ using error backpropagation.

### C.1.2.1 Approximate Distance Function

The approximation relies on a simple observation: given an occupancy grid, successive convolution with a fixed kernel $w = 1^{K \times K}$ will lead to larger values for occupied voxels and lower values for non-occupied voxels further away. Depending on how many iterations are performed and on the size $K$ of the kernel, far away voxels might still be 0. Adding 1 to every voxel, and taking the multiplicative inverse leads to a distance function with values in $[0, 1]$ that are closer to 1 for further away voxels and closer to 0 for occupied voxels. If we require that the distance function is 0 for occupied voxels, we can finally multiply by $1 - y$.

In practice, we found that thresholding the predicted shape is not necessary. An illustration of the result is given in Figure C.2 in terms of:

**Definition C.1** Let $\times$ be an element-wise tensor operation, $x$ an arbitrary input tensor and $s$ be a fixed tensor, then the corresponding layers $\cdot \times s$ or $s \times \cdot$ compute

$$(\cdot \times s)(x)_i = x_i \times s_i$$

$$(s \times \cdot)(x)_i = s_i \times x_i.$$      

These layers can analogously be extended to cases taking two tensors as input.

### C.1.3 Practical Considerations

When using a variational auto-encoder prior, the decoder, i.e. $p(y|z)$ is retained and only the encoder is trained. In the case of a probabilistic PCA prior, we use

$$\theta(z) = g(Uz + \mu)$$
as decoder, where $g$ is a clipping function or a scaled sigmoid. This can be implemented as fully-connected layer with bias and non-linearity.
Appendix D

Data

Complementing the discussion of the introduced datasets in Chapter 8, we additionally discuss the 2D rectangle dataset originally used to prototype the proposed shape completion approaches. Additionally, we provide further examples for the remaining datasets.

D.1 2D Example

Our synthetic 2D dataset consists of arbitrarily scaled and rotated rectangles. Because we work with either occupancy grids or signed distance functions, the rectangles are directly generated in discretized forms, i.e., as binary images. This way we can directly generate the shape set $\mathcal{Y} \subseteq \mathbb{R}^{H \times W}$ used to learn the shape prior as well as the ground truth shapes $\mathcal{Y}^* \subseteq \mathbb{R}^{H \times W}$. We note that $\mathcal{Y} \cap \mathcal{Y}^* = \emptyset$, i.e., we assume that the shape prior is not trained on the same shapes also encountered during shape completion. It remains to synthesize the observations $\mathcal{X} \subseteq \mathbb{R}^{H \times W}$ from the ground truth shapes $\mathcal{Y}^*$. These are obtained through a ray tracing process given a 1D projective camera. The overall process is also illustrated in Figure D.1a and the ray tracing part is described in detail in the following.

D.1.1 Ray Tracing

Given the discretized shapes $\mathcal{Y}^* \subseteq \mathbb{R}^{H \times W}$, we place a 1D projective camera in the scene in order to record a 1D projection of the shapes which can subsequently be back-projected to form the observations $\mathcal{X}$:

**Definition D.1** A 1D projective camera is described by a tuple $(\mathcal{K}, \mathcal{R}, t)$ where $\mathcal{K} \in \mathbb{R}^{2 \times 2}$ is the intrinsic camera matrix and $\mathcal{R} \in \mathbb{R}^{2 \times 2}$, $t \in \mathbb{R}^2$ define a similarity in 2D. The intrinsic camera matrix has the form

$$
\mathcal{K} = \begin{bmatrix}
    f & u \\
    0 & 1
\end{bmatrix}
$$
(a) Illustration of the data generation process for our 2D rectangle dataset. We sample random rectangles \( y \in \mathbb{R}^{H \times W} \) with different size and rotation. Then, indicated with (a), we set up a 1D camera given the corresponding center \( t_{\text{cam}} \) and viewing direction \( r_{\text{cam}} \), cf. Figure D.1b. Subsequently, marked by (b), we perform ray tracing to compute the observation \( x \) in the form of observed points, full free space and partial free space. In (c), we post-process the shapes \( y^* \) as well as the observations \( x \) to obtain the corresponding (signed) distance functions.

(b) Illustration of the 1D camera from Definition D.1. The camera is set up by specifying the camera center \( t_{\text{cam}} \) and the corresponding viewing direction \( r_{\text{cam}} \) which is used to derive the rotation matrix \( R_{\text{cam}} \).

Figure D.1.: Illustration of the data generation process in Figure D.1a and the used camera model in Figure D.1b.
D.1. 2D Example

and $R$ is orthogonal. Here, $f$ is the focal length, i.e. the distance between camera center and virtual image line; $u$ is the principal point, i.e. center pixel, in 1D image coordinates – we always assume $u$ to be the middle pixel of the 1D image, thereby also defining the resolution as $2u$.

The above definition is illustrated in Figure D.1b. Given a camera $(K, R, t)$, the overall projection matrix is

$$\mathcal{P} = K[R \ t] \in \mathbb{R}^{2 \times 3}.$$  \hspace{1cm} (D.1)

However, for ray tracing, we are interested in the inverse transformation [HZ06, Chapter 6]

$$\mathcal{P}^+ = (\mathcal{P}^T \mathcal{P})^{-1} \mathcal{P}^T \in \mathbb{R}^{3 \times 2}$$  \hspace{1cm} (D.1)

which gives the ray going through the camera center and pixel $x = (x_1, 1)^T$:

$$r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ 1 \end{bmatrix} \quad \text{with} \quad \tilde{r} = \mathcal{P}^+ x.$$  

To set up $\mathcal{P}^+$, we randomly choose a camera position $t_{\text{cam}}$ within appropriate ranges and compute the corresponding viewing direction $r_{\text{cam}}$ letting the camera look to the center of our shapes. Then, following [HZ06, Chapter 6]:

$$R = R_{\text{cam}}^T \quad \text{and} \quad t = R t_{\text{cam}}.$$  

where the matrix $R_{\text{cam}}$ corresponding to the camera’s viewing direction $r_{\text{cam}}$ can be set up given the angle $\alpha = \angle(r_{\text{cam}}, (1, 0)^T)$ between viewing direction and $x$-axis:

$$R_{\text{cam}} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}.$$  

. Using $R$, $t$ as well as the intrinsic camera matrix $K$, which we fix in advance, $\mathcal{P}^+$ is computed following Equation (D.1).

Ray tracing is then performed by taking each pixel in homogeneous coordinates, computing the corresponding ray and following it to determine all intersected pixels of the binary image until it hits an occupied pixel. All pixels up to the occupied pixel are marked as free space. If we also consider rays that leave the image without hitting an occupied pixel, we obtain the full free space; if we only consider rays to occupied pixels, we obtain the partial free space, see Figure D.2 for examples regarding this distinction. The distinction becomes important on real data, *e.g.* on KITTI [GLU12, GLSU13], where full free space cannot be computed reliably.

D.1.2 Noise

So far, the ray tracing approach is fully deterministic. Although we can reduce the resolution of the 1D image thereby reducing the number of observed pixels, we also
Appendix D. Data

D.1. 2D Example

Figure D.2: Examples from our synthetic 2D dataset with different resolutions of the 1D image, i.e. 64 and 32, and difference noise parameters, i.e. $\theta_{\text{hit}} \in \{1, 0.7\}$ and $\theta_{\text{ignore}} \in \{0, 0.2\}$. For each example, we show the full shape $y$ and the observation $x$ in the form of observed points, free space from all rays (i.e. full free space) and free space from rays corresponding to observed points (i.e. partial free space).

want to inject some randomness into ray tracing to simulate real conditions. To this end, we introduce two probabilities $\theta_{\text{hit}}$ and $\theta_{\text{ignore}}$ describing the probability that a ray actually hits an occupied pixel and the probability that all occupied pixels along a ray are ignored, respectively. Examples are provided in Figure D.2.

D.1.3 Discussion

Overall, our synthetic 2D dataset is kept very simple – if not to say easy. We will see that even predicting a mean shape performs reasonable well. However, the 2D dataset had significant influence on the problem formulation and always allowed us to backtrack from 3D to a simple 2D case. In Section E.3, the 2D dataset allows us to present comprehensive experiments which would not have been possible in 3D due to the higher running times. The dataset also enabled us to experiment with different noise models.
D.2 3D Example

In Figure D.3 we show additional examples of the generated 3D cuboids datasets used for experiments, i.e. corresponding to easy, moderate and hard difficulties.

Figure D.3.: More examples from the generated 3D cuboid datasets; we show the easy, moderate and hard cases as outlined in Table 9.1, two examples each. Again, we show heights $8 + 2i$ for $0 \leq i < 8$, illustrating the observed points, the computed partial free space and the ground truth shape. Additionally, we show 3D visualizations of the observed points and the corresponding shape.
D.3 ShapeNet

In Figure D.4 we show additional examples of the generated ShapeNet [CFG+15] datasets as used for experiments, i.e. corresponding to easy, moderate and hard difficulties.

Figure D.4.: More examples from the generated ShapeNet datasets; we show the easy, moderate and hard cases as outlined in Table 9.2, two examples each. Again, we show heights $8 + 2i$ for $0 \leq i < 8$, illustrating the observed points, the computed partial free space and the ground truth shape. Additionally, we show 3D visualizations of the observed points and the corresponding shape.
D.4 KITTI

In Figure D.5, we show additional examples extracted from KITTI [GLU12, GLSU13]. We particularly want to highlight that noisy observations, especially observations lying clearly within the car, occur frequently at window height of the cars. Here, it becomes apparent that the Velodyne sensor has difficulties with reflective or transparent surfaces.

Figure D.5.: More example from the extracted KITTI dataset as used for the experiments presented in Chapter 9. We show heights $8 + 2i$ for $0 \leq i < 8$, illustrating the observed points and the corresponding free space. We also show 3D visualizations of the observed points from two viewpoints.
Appendix  E

Experiments

In this chapter, we present additional experiments, further qualitative examples as well as details regarding training and evaluation.

E.1 Training Details

For training, we use stochastic gradient descent with momentum and weight decay. The initial learning rate was experimentally determined to be $\gamma^{(0)} = 10^{-6}$ for most experiments; for experiments involving signed distance functions the initial learning rate might be slightly slower, e.g. $\gamma^{(0)} = 5 \cdot 10^{-7}$ – on the 2D dataset we even reduced it to $\gamma^{(0)} = 5 \cdot 10^{-8}$ when training signed distance functions only. It is then reduced every $T_\gamma = 500$ iterations using $\alpha_\gamma = 0.95$. The minimum learning rate $\gamma_{\text{min}} = 10^{-15}$ is usually only reached for training the shape prior, which involves significantly more epochs than training the inference models. We usually use a momentum of $\beta^{(0)} = 0.5$ and increase it every $T_\beta = T_\alpha = 500$ iterations with a factor $\alpha_\beta = 1.025$ until $\beta_{\text{max}} = 0.9$ is reached. In 2D, because training is easier, we increase the momentum term using $\alpha_\beta = 1.05$ instead. For weight decay, we usually use a weight of $\kappa = 10^{-6}$. We trained the shape prior to up to 250 epochs; and inference models usually up to 25 epochs – except on KITTI [GLU12, GLSU13] where significantly more epochs are used as the dataset is very small. We always present results corresponding to a model trained for the optimal number of iterations determined using early stopping. We found that weight decay has negligible influence on the obtained performance, but slightly helps training. Additionally, the initial learning rate has a significant effect when chosen too large – in the extreme case, training diverges in the first few iterations. Using a momentum term helps in all cases and gradually reducing the learning rate is necessary to obtain good results. We always use Xavier initialization [GB10]. We also experimented with alternative optimization algorithms, e.g. ADAM [KB14], but found standard stochastic gradient descent to work most reliably.
Appendix E. Experiments

Table E.1.: Overview of the generated datasets. We created datasets of three difficulties, easy, moderate and hard, which refer to increased noise and less observations. Additionally, we report statistics such as the percentage of observed voxels, free space voxels and occupied voxels (of the ground truth shapes) over the training set used for shape inference.

<table>
<thead>
<tr>
<th></th>
<th>2D easy</th>
<th>2D moderate</th>
<th>2D hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Size (Prior/Inference)</td>
<td>10000/10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation Size</td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>Resolution $H \times W$</td>
<td>32 x 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution 2u</td>
<td>64</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Noise $\theta_{hit}$</td>
<td>1</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>Noise $\theta_{ignore}$</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Observed Voxel (Inference Training Set)</td>
<td>1.79%</td>
<td>1.09%</td>
<td>0.97%</td>
</tr>
<tr>
<td>Free Space Voxel (Inference Training Set)</td>
<td>13.63%</td>
<td>10.99%</td>
<td>16.59%</td>
</tr>
<tr>
<td>Occupied Voxel (Inference Training Set)</td>
<td>29.6%</td>
<td>29.57%</td>
<td>29.31%</td>
</tr>
</tbody>
</table>

**E.2 Evaluation**

We claimed that the negative log-likelihood is unsuited for evaluation partly because it favors “unsure” predictions, i.e. occupancy probabilities close to 0.5 over very certain predictions with few mistakes. The following example is intended to illustrate this:

**Example E.1** We consider a $R$-dimensional binary volume representing a shape. Let $y \in [0, 1]^R$ the predicted shape; let $y^* \in \{0, 1\}^R$ be the true shape. Assuming $y$ fits $y^*$ perfectly, the negative log-likelihood is zero. Flipping the prediction of one $y_i = 1$ to $y_i \approx 0.0001$ (due to numerical stability) changes the negative log likelihood to $\sim 9.21$. Then, we can set $-\ln 0.0001 \approx 9.21 \approx 9.966$ of $y^*$’s occupied voxels to 0.5 and the negative log-likelihood will not exceed 9.21. In practice, this means that an “unsure” model predicting an occupancy probability of 0.5 for many pixels will have a lower negative log-likelihood than a nearly perfect prediction with few but very certain mistakes.

**E.3 2D Example**

We originally developed the 2D dataset to understand how probabilistic PCA (PPCA) and variational auto-encoders (VAEs) can learn shape priors; and to provide a proof of concept that the methods for shape inference introduced in Chapter 7 as well as in Appendix C are applicable. We also intend to present a short quantitative comparison to identify well-performing approaches in order to proceed to the 3D datasets as well as KITTI.

For all experiments regarding shape completion, we constructed three datasets of varying difficulties: easy, hard and moderate with details in Table E.1. In all three cases, we always use partial free space as this will also be the case on KITTI. With rising difficulty, less observations are provided and the underlying
E.3. 2D Example

Appendix E. Experiments

E.3. 2D Example Appendix E. Experiments

E.3. 2D Example Appendix E. Experiments

Figure E.1.: Absolute error Abs and its thresholded pendant Abs\(_{\text{thresh}}\), i.e. the absolute error after thresholding the predicted shapes, for PPCA and VAE with different size \(Q \in \{2, 5, 10, 25\}\) of the latent space. For VAE we additionally compare the different representations, i.e. occupancy (occ) and signed distance functions (sdf). For PPCA we only report results on occupancy. Note that for VAEs and occupancy, Abs and Abs\(_{\text{thresh}}\) are usually very close together. In each case, the bars on the left represent results on occupancy (if applicable); while the bars on the right correspond to results on signed distance functions (if applicable).

noise increases. The hard case is supposed to represent real conditions as found on KITTI.

The experimental setup follows Chapter 9. Additionally, PPCA was implemented using SciPy’s sparse SVD implementation\(^1\) and \(\sigma^2\) from Definition B.2 was approximated using the first \(\rho \cdot HW\) eigenvalues of the data matrix; here, \(\rho\) is the fraction of pixels occupied over the whole training set. The \(Q\) largest eigenvalues are chosen to determine the projection \(U\) and the remaining ones are used to compute \(\sigma^2\). We found this heuristic to work well in both 2D and 3D. In the 2D case, \(\rho \approx 0.297\) and for \(H \times W = 32 \times 32\), we have \(\rho HW \approx 302\).

E.3.1 Shape Prior

First, we want to understand how well PPCA and VAEs can learn a shape prior – especially regarding the latent space, e.g. its dimensionality and the quality of random samples. For VAEs we are additionally interested in the two introduced representations – occupancy and signed distance functions.

E.3.1.1 Probabilistic PCA

In Figure E.1, we first consider PPCA to learn shape priors to understand the influence of the size of the latent space, i.e. \(Q\). Obviously, increasing \(Q\) leads to better reconstruction errors. As comparison, roughly 29.7 percentage of pixels are occupied, i.e. predicting all zero would result in an error of approximately Abs = Abs\(_{\text{thresh}}\) \(\approx 0.297\). The corresponding qualitative results in Figure E.2, for \(Q = 2\), show what PPCA actually does: it uses principal directions to approximate its input. Increasing \(Q\) allows PPCA to capture more variation, leading to better reconstructions but also allowing more variation in random samples. For shape inference, we require that the latent space holds only “reasonable” shapes. Taking this into account \(Q = 5\) seems to be a reasonable trade-off between reconstruction error and appropriate samples.

\(^1\)https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svd.html.
Appendix E. Experiments E.3. 2D Example

Figure E.2: Qualitative results for PPCA with different sizes $Q \in \{2, 5, 10, 25\}$ of the latent space. From top to down, we show the input shape, the reconstruction, the corresponding error and the thresholded reconstruction. The last two rows show random samples. We clipped the visualization to $[0, 1]$ for convenience.

Figure E.3: Absolute error $\text{Abs}$, average $\mu$ of the predicted means, the corresponding standard deviation $\sqrt{\text{Var}[\mu]}$ and the average $\exp\left(\frac{1}{2}\right)$ of the predicted standard deviations on the validation set for VAEs trained with $Q = 5$ (left) and $Q = 25$ (right). For the actually observed standard deviation, we plot its deviation from 0 for convenience, i.e. $|1 - \sqrt{\text{Var}[\mu]}|$. We refer to Equations (6.6), (6.7) and (6.8) for details. Note that for $Q = 25$, the actual standard deviation collapses to roughly 0.55 such that $|1 - \sqrt{\text{Var}[\mu]}|$ approaches $\sim 0.45$ meaning that the unit Gaussian prior is not met.

E.3.1.2 Variational Auto Encoder

Next, we consider the VAE equivalent using occupancy as shape representation. First of all, we discuss training; Figure E.3 shows absolute error $\text{Abs}$ on the validation set for $Q = 5$. We additionally monitor the predicted means through $\text{Var}[\mu]$ and $\mu$ as well as the predicted standard deviations, i.e. $\exp\left(\frac{1}{2}\right)$. As can be seen, network training seems to stagnate after around $100k$ iterations according to the reconstruction error. However, the latent space is still being learned, especially regarding the actually observed standard deviation $\text{Var}[\mu]$ which still
Figure E.4: A comparison of random samples for VAE shape rpiors using $Q = 5$ and $Q = 25$ for occupancy only (top) and $Q = 5$ for occupancy and signed distance functions (bottom). In the former case, we show reconstructions of five samples for $Q = 5$ and $Q = 25$, respectively, illustrating target shape in the first row, followed by the prediction and its error. Below we show 25 random samples. For the latter case, we show the target shapes in the first row, the reconstruction and the corresponding error in the next two rows and, finally, two rows of random samples.

deviates significantly from 1, i.e. $|1 - \sqrt{\text{Var}[\mu]}|$ deviates from 0. Overall, VAEs usually need to be trained longer than the reconstruction loss suggests.

Monitoring the latent space also helps to identify an appropriate size $Q$ of the latent space. To this end, Figure E.3 also shows training of the same network
for $Q = 25$. First, we notice that the predicted standard deviations do not approach zero anymore. The latent space is larger for $Q = 25$, however, the size of the dataset did not increase. Thus, the network needs to “fill” a larger latent space with the same number of samples; thus, the predicted standard deviations are higher. Second, the actually measured standard deviation does collapse to around $\sim 0.55$; this means that the unit Gaussian prior is not met exactly. Both phenomena indicate that the latent space is too large. Similar observations can be made for smaller latent spaces. Using $Q = 5$ for the following experiments is also supported by the random samples in Figure E.4. For $Q \in \{2, 5, 10, 25\}$ we compare the reconstruction performance with PPCA in Figure E.1.

The above observations can be confirmed when training a VAE on a signed distance function representation. Again, we use $Q = 5$ and observe that signed distance functions are harder to learn. Therefore, we focus on predicting both occupancy and signed distance functions instead. In Figure E.5, we show the corresponding training curves. We observe that longer training is necessary. This has, of course, also influence on the learned latent space. The corresponding qualitative results in Figure E.4 show that the reconstructions exhibit very low error while random samples appear to be of lower quality. In particular, the shown random samples do not appear as sharp as the reconstructions; similarly, some of them deviate more strongly from the rectangles seen during training.

Overall, we are satisfied by the performance of VAEs with respect to reconstruction performance. Random samples, as well, appear reasonable, especially in the occupancy only case. However, judging the quality of the latent space is still difficult – in our case this conclusion is based on manually inspecting many random samples and reconstructions.

### E.3.2 Maximum Likelihood

We followed Section 7.1 and performed maximum likelihood (ML) using PPCA and VAE priors with $Q = 5$. For optimization we used gradient descent with initial learning rate $\gamma^{(0)} = 0.01$, which is decreased every $T_\gamma = 100$ iterations according to $\alpha_\gamma = 0.95$; similarly we use a momentum parameter of $\beta^{(0)} = 0.3$ which is
E.3. 2D Example

**Figure E.6.** Comparison of all proposed approaches, ML, AML and EVAE on the 2D dataset for easy, moderate and hard difficulties with respect to absolute error $Abs$ and thresholded absolute error $Abs_{\text{thresh}}$. We only consider occupancy, for signed distance functions as shape representation, we refer to Figure E.9. For AML and EVAE we additionally compare the use of the weighted maximum likelihood loss, marked by *.

**Figure E.7.** Qualitative results for ML and DL using both PPCA and VAE priors. We show results for the easy and the hard cases in the respective columns. For each approach, we show five shape predictions including the corresponding error. On top we additionally show the observed points, the free space as well as the target shape.
Appendix E. Experiments

E.3. 2D Example

Figure E.8.: Comparison of ML and DL using both PPCA and VAE priors. We plot the absolute error Abs and its thresholded version Abs\textsubscript{thresh} for all three difficulties, i.e. easy, moderate and hard. The results can also be compared to AML and EVAE in Figure E.8.

increased every $T_\beta = T_\gamma = 100$ iterations using $\alpha_\beta = 1.05$. We usually run 2$k$ iterations starting with $z^{(0)} = 0$. Figure E.8 shows results for the three difficulty levels. Surprisingly, regarding Abs\textsubscript{thresh}, the PPCA prior outperforms than the VAE prior. This can be explained when considering Figure E.7 showing qualitative results. In particular, we notice that the VAE prior predicts very clear rectangles. These however, do not fit the targets perfectly. Still, the results from the VAE prior look more applicable. It is also surprising, that the influence of the difficulty is negligible. Overall, ML performs rather poorly; considering that only roughly 29.6% of the pixels are occupied, the absolute errors of $> 0.1$ are still very high. For the hard difficulty, the predicted shapes do not even match original rotation and size.

E.3.3 Non-Probabilistic Approach

In the non-probabilistic approach, referred to as DL, we interpret the pre-trained generative model $p(y | z)$ as general decoder $y(z; w)$; we then train a new encoder with the same architecture as the recognition model $q(z | y)$ and the decoder is kept fixed during training. The used losses then tie the predicted shapes to the observations (cf. Equations (C.1) and (C.2)). In addition, a negative log-likelihood loss on the prior $p(z)$ ensures that the encoder only selects shapes from the learned latent space. Again, we investigate both PPCA and VAEs as priors.

The results are shown in Figure E.8 in comparison with ML. In this formulation, the VAE prior performs slightly better; however, an absolute error Abs of $\sim 0.095$ for the easy case is still pretty poor. It shows, however, that amortized inference, i.e. learning the shape completion task, may be beneficial. Considering the qualitative results in Figure E.7, we also see that PPCA merely predicts a mean shape which happens to perform well on the 2D dataset by construction. The VAE prior still predicts reasonable shapes in the easy case; but it also resorts to “standard” shapes in the hard case. Overall, the non-probabilistic approach does not perform very well. In the following, we resort to the maximum likelihood loss $\mathcal{L}_{ML}$ instead, leading to amortized maximum likelihood.
E.3.2D Example

Figure E.9.: Absolute error Abs and thresholded absolute error Abs\text{thresh} for AML on both occupancy and signed distance functions and in comparison to the corresponding VAE shape prior. Results for all three difficulties easy, moderate and hard are shown. For hard, we additionally show results for the weighted loss; these results are marked with a star, *.

E.3.4 Amortized Maximum Likelihood

Amortized maximum likelihood, referred to as AML, follows the idea of the non-probabilistic approach but uses the original maximum likelihood loss (cf. Equation (7.3)) instead. In the following, we present experiments regarding both occupancy and signed distance functions.

Figure E.9 shows quantitative results of AML on occupancy grids considering all three difficulty levels. Especially the hard difficulty causes problems; then again, the VAE prior performs very well on easy and moderate difficulties. This is also stressed in Figure E.10. We interpret these results as a proof of concept that amortized maximum likelihood is beneficial; still, improvements for the hard case are necessary. Here, PPCA performs better, which might be explained by the very strong (as linear) prior. The main problem in the hard case is invalid free space, especially the ignored rays going through the target shape. This also leads to predictions which should actually not lie in the learned latent space. To this end, we experimented with a weighted variant of the maximum likelihood loss (cf. Equation (7.3)). In particular, we weight the maximum likelihood loss on free space pixels, i.e. $x_i = 0$, by

$$
\rho_i = 1 - \frac{\sum_{m=1}^{M} y_{m,i}}{M}, \quad Y = \{y_m\}_{m=1}^{M} \subseteq \{0, 1\}^R.
$$

Intuitively, $\rho_i$ is intended to express the likelihood that the observation $x_i = 0$ can be trusted. For the 2D dataset, this is illustrated in Figure E.10. Additionally, following Equation (7.7), we weight the negative log-likelihood $-\ln p(z)$ using $\kappa = 15$ (experimentally determined). The results of both changes can also be seen in Figures E.9 and E.10, marked by *.

Next, we discuss the case of learning both occupancy and signed distance functions. In Figure E.9 we show quantitative results in comparison with predicting occupancy only. Especially for the easy case, the thresholded absolute error Abs\text{thresh} degrades in both cases; Surprisingly, predicting both modalities only
improves performance in the hard case; here also predicting occupancy seems to be beneficial. Figure E.11 shows qualitative results for the easy and hard cases; it can be seen that even in the easy case, the network is not always able to learn rectangles of correct rotation and size. For the hard case, we see why occupancy can be beneficial: the invalid and noisy free space seems to be particular hard when predicting signed distance functions. Again, the weighted loss is helpful in these cases. Overall, predicting both modalities seems to be a reasonable choice.
Figure E.11.: Qualitative results for AML predicting both modalities, i.e. occupancy and signed distance functions. We show results for both the easy and the hard case. For the easy case, we show the observed points, the corresponding free space and the target shape; followed by the prediction and the corresponding error. For the hard case we follow the same procedure, however, we also show results using the weighted loss.

considering that signed distance functions are harder to learn and performance, thus, decreases slightly, but we still want to benefit from subvoxel accuracy potentially provided by signed distance functions.

### E.3.5 Extended Variational Auto-Encoder

For the extended variational auto-encoder, EVAE in short, we only present experiments for occupancy predictions. In Figure E.9 we compare the achieved shape completion performance with AML. Surprisingly, EVAE marginally outperforms AML. However, this can also be due to differences in training and initialization which cannot be avoided completely. In the end, AML and EVAE optimize the same
maximum likelihood objective – in the case of EVAE, this objective is, however, hidden within a Kullback-Leibler divergence. To complement the discussion, we present qualitative results in Figure E.11; we can see that EVAE and AML indeed seem to optimize the same objective and obtain very similar results.

### E.3.6 Discussion

Considering the experiments presented so far, and the additional material in Section E.3, we understand that the shape completion problem is hard. Especially for the – artificially – harder versions of our dataset, *i.e.* **moderate**, where on average only 12% of the $32 \times 32 = 1024$ pixels are observed – 1% observed points and 11% free space. In the case of **hard**, even less points are observed and 2% of the observed free space is invalid, *i.e.* goes through the target shapes. While a VAE shape prior is able to learn a model with roughly 0.005 absolute reconstruction error (with $Q = 5$ dimensional latent space), the proposed shape completion approaches such as AML or EVAE achieve an absolute error of $\sim 0.037$ in the **easy** case. The performance degrades to roughly 0.073 on the **hard** dataset. We also want to remind the reader that the considered dataset – even tough artificially made harder – is still very simple. The rectangles are convex shapes and the network only has to consider rotation, scaling as well as minor translations. Still the completed shapes do not always look convincing, especially in the **hard** case. Here it is surprising that predicting a mean shape, as done by PPCA, may outperform a more complex VAE prior, but only in the case of **ML**. For AML and EVAE, the VAE prior performs better and also predicts qualitatively sharper and more reasonable shapes. For proceeding to the 3D case, we do not consider PPCA as prior anymore. For shape completion, we focus on AML as well as EVAE.
E.4 3D Example

We complement the discussion of experimental results on our synthetic 3D dataset in Chapter 9 with additional results regarding the VAE shape prior and shape completion using AML and EVAE.

In Figure E.12 we illustrate the training of two VAEs on occupancy only with $Q = 5$ and $Q = 30$. In practice, these plots may give valuable clues on the correct size $Q$ of the latent space. As can be seen, using $Q = 5$, the network is not able to obtain reasonably low reconstruction error (in terms of Abs) and the learned latent space deviates significantly from the unit Gaussian prior. For $Q = 30$, in contrast, a significantly lower absolute error is achieved. The latent space additionally converges roughly to a unit Gaussian prior. From these results, we should clearly prefer $Q = 30$, even compared to the used $Q = 15$. However, Figure E.13 demonstrates that large $Q$ can lead to worse random samples as the VAE has problems “filling” the latent space with reasonable samples. Overall, this supports our choice of $Q = 15$.

For AML and EVAE we present additional qualitative results in Figure E.14 when considering occupancy only. We consider the easy and moderate cases as they have not been discussed before. Again, we can see that both approaches provide reasonable predictions; additionally, it shows that the easy case provides significant more observation compared to moderate and hard difficulties and, thus, is indeed a very simple problem for AML and EVAE. In Figure E.15 we show the corresponding results for AML using both occupancy and signed distance functions. In this case, AML struggles to predict matching shapes on moderate difficulties and predictions occur less sharp. We suspect that tuning the shape prior (i.e. longer training, adapting $Q$ etc.) might mitigate these problems.

![Figure E.12.](image-url) Training curves for VAE shape priors on the 3D cuboids dataset with $Q = 5$ (top) and $Q = 30$ (bottom) showing loss $L_{BCE} + KL$ and absolute error Abs on training (train) and validation set (val) as well as latent space statistics, i.e. the average $\mu$ of the predicted means, the corresponding standard deviation $\sqrt{\text{Var}[\mu]}$ and the average of the predicted standard deviations $\exp(\frac{1}{2} l)$. 

For Figure E.12, we observe that when $Q = 5$, the network fails to achieve a low absolute error. In contrast, when $Q = 30$, the network converges to a lower absolute error and the latent space more closely resembles the unit Gaussian prior. This supports the choice of $Q = 15$ over $Q = 5$. For AML and EVAE, we present additional qualitative results in Figure E.14 when considering occupancy only. We consider the easy and moderate cases, which have not been discussed before. Both approaches provide reasonable predictions, and we observe that the easy case provides significantly more observation compared to the moderate and hard cases. This is indicative of a simpler problem for AML and EVAE. In Figure E.15, we show the corresponding results for AML using both occupancy and signed distance functions. However, AML struggles to predict matching shapes on moderate difficulties, and predictions are less sharp. We speculate that tuning the shape prior (e.g., through longer training or adapting $Q$) might mitigate these issues.
Figure E.13.: Random samples for VAEs trained on the 3D cuboids dataset using $Q = 15$ and $Q = 30$. Although random samples are to be judged with caution, we find the model with $Q = 15$ generates more reasonable random samples. Again, we show horizontal slices of the volumes, in particular heights $8 + 2i$ for $0 \leq i < 8$.

Figure E.14.: Qualitative results for AML and EVAE on occupancy only considering the easy and hard cases of the 3D cuboids dataset. As can be seen, both approaches perform reasonably well in these cases. As before, we show horizontal slices of the volumes, in particular heights $8 + 2i$ for $0 \leq i < 8$. 
Figure E.15: Qualitative results for AML using both occupancy and signed distance functions on easy and moderate difficulties of the 3D cuboids dataset. Here, AML has considerable difficulties on moderate difficulty. We show the corresponding horizontal slices of the volumes, i.e. height levels $8 + 2i$ for $0 \leq i < 8$. 
Appendix E. Experiments

E.5 ShapeNet

Regarding experiments on our synthetic ShapeNet [CFG+15] dataset, we provide additional 3D visualizations corresponding to the VAE shape prior as well as shape completion using AML in comparison with the supervised baseline. For the VAE shape prior we show qualitative results in Figure E.16 when training on occupancy only or on both modalities, i.e. occupancy and signed distance functions. Results for AML, again using either occupancy only or both modalities, can be found in Figures E.17 and E.18, respectively. Here, we consider the moderate case in more detail showing that AML is able to make significantly better predictions with less noisy observations. In comparison to the supervised baseline, AML is able to predict shapes that are – on the low resolution of $32^3$ – harder to distinguish from the target or the supervised prediction. This also holds when using both modalities, i.e. occupancy and signed distance functions, where AML performed particularly poor on the hard case. On moderate difficulty, in contrast, we are able to derive smooth meshes closely resembling the target mesh. Additionally, we are able to recover more detail, e.g. the roof or the wheels.

![3D visualizations](image)

(a) 3D visualizations of the reconstruction capabilities using occupancy only. We show the occupancy grids corresponding to the target shape, the reconstructed shape and its error.

![3D visualizations](image)

(b) 3D visualizations of random samples generated in the occupancy only case. Each sample is shown from two distinct viewpoints.

![3D visualizations](image)

(c) Random samples generated using both occupancy and signed distance functions. We show meshes obtained by running marching cubes [LC87] on the generated signed distance functions.

**Figure E.16.:** Additional qualitative results for a VAE shape prior using $Q = 15$ trained on occupancy only, Figures E.16a and E.16b, and on both modalities, Figure E.16c, using ShapeNet.
Figure E.17.: Qualitative results for AML and the supervised baseline for the moderate and hard cases of our ShapeNet-based dataset. We show the occupancy grids corresponding to the observed points, the AML prediction, the prediction from the supervised baseline and the target shape.
Figure E.18.: 3D visualizations of AML using both occupancy and signed distance functions on the ShapeNet dataset. Here we show meshes obtained from marching cubes applied on the predicted signed distance functions in comparison to the corresponding targets. We note that the signed distance functions are derived from the corresponding occupancy grids; this explains the “voxelized” appearance of the ground truth meshes.

Figure E.19.: Qualitative results for AML predicting both occupancy and signed distance functions on KITTI. We show the occupancy grids corresponding to the observed points, the predicted shape and the mesh derived from the predicted signed distance functions using marching cubes. On KITTI, we do not have access to ground truth shapes.
E.6 KITTI

On KITTI [GLU12, GLSU13], we also present additional qualitative results in the form of 3D visualizations. In particular, in Figure E.20 we provide additionally qualitative results comparing AML against the supervised baseline using occupancy only. In Figure E.19 we provide additional qualitative results for AML using both occupancy and signed distance functions.

**Figure E.20.** Additional qualitative results of AML and the supervised baseline on KITTI. Here, we do not have access to ground truth shapes. Therefore, we can only show the occupancy grids corresponding to the observed points, the AML predictions and the predictions from the supervised baseline.