Master's thesis

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

Peder Bergebakken Sundt

Single-View 3D Shape Completion for Robotic Grasping of Objects via Deep Neural Fields

Master's thesis in Computer Science
Supervisor: Ekrem Misimi, Sintef OCEAN
Co-supervisor: Theoharis Theoharis, IDI
June 2021

NTNU
Norwegian University of Science and Technology

Peder Bergebakken Sundt

Single-View 3D Shape Completion for Robotic Grasping of Objects via Deep Neural Fields

Master's thesis in Computer Science
Supervisor: Ekrem Misimi, Sintef OCEAN
Co-supervisor: Theoharis Theoharis, IDI
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

In this thesis we investigate 3D shape completion and reconstruction of volumetric objects from a single view, to enable a robot arm controller to make inference of the 3D object's shape during the manipulation stage when equipped with 3D vision. It deals with one of the fundamental problems in robotic object manipulation: perception. Objects may from a single viewpoint be only partially observable by a visual sensor due to various occlusions. As such there are many perception ambiguities to solve before building 3D models of objects, and consequently gripping them, becomes possible.

We investigated a machine learning approach based on implicit surfaces, leveraging the novel study of *neural fields* which has recently become popular. This data-driven learning paradigm handles arbitrary shape topologies and reduce the system requirements by an order of magnitude compared to previous state-of-the-art methods typically based on convolution. Our shape completion method is based on searching for the shape embedded in latent space that best conforms to the single-view observation data, using stochastic gradient decent.

We trained deep neural networks, whose input is a single continuous 3D Cartesian coordinate, to represent implicit surfaces in latent space by approximating their signed distance function (SDF). We experimented with the size of these networks, with LReLU and sinusoidal nonlinearities, and with how to best train the networks on the 3D models of the YCB dataset using various novel regularization techniques and loss functions. We showed that supervising sinusoidal networks with a truncated SDF signal *and* its spatial derivative yield better shape reconstructions, scored with Chamfer distance, earth movers distance, mesh cosine similarity and F-score.

The aim and primary contribution of this thesis was to construct a latent space of not only a wide selection of shapes, but of shapes over a continuous space of orientations, effectively combining shape completion with pose estimation. This had the benefit of promoting learning rotationally invariant shape features. We analyzed how similar shapes cluster and transition between each other in latent spaces learned by auto-decoders. We discovered that including multiple objects in each training batch drastically improved the convergence rate. We additionally proposed a method to sample SDF values from real-world depth sensor data. We showcased the ability of our model to perform shape completion on partial and noisy 3D data in a single-view real-world context. Based on these results, our methodology is a valuable contribution to the robotic based single-view 3D shape completion.

Preface

This master thesis is the result of the work performed over the course of the spring semester 2021 carried out at the Department of Computer and Information Science (IDI), at the Norwegian University of Science and Technology (NTNU). This thesis is also a part of the GentleMAN project at SINTEF Ocean which aims to develop a learning framework using visual and tactile sensing to aid the manipulation of 3D compliant objects with a robot controller by equipping it with 3D RGB-D vision. I want to thank Ekrem Misimi and Theoharis Theoharis for their guidance and proofreading.

Sammendrag

Vi undersøker i denne oppgaven rekonstruksjon av fullstendige volumetriske 3D modeller fra et enkelt synspunkt, for å gi en robotarm utstyrt med 3D syn ferdigheten til å antyde fasongen til objekter og derav håndtere dem. Dette er en av de grunnleggende utfordringene for robotisert manipulasjon: visuell forståelse. Objektene kan være kun delvis synlig fra et enkelt synspunkt, ettersom de kan være tildekket av andre objekter eller fysiske barrierer. Sådan er det uklarheter løse opp i før robotagenter kan bygge fullstendige 3D modeller av objekter og analysere disse for å gripe dem.

Vi tok i bruk den nye maskinlæringsmetoden kjent som *nevrale felt*, og undersøkte implisitte overflater. Denne læringsparadigmen håndterer fasonger av vilkårlig genus og bruker færre systemresurser enn tidligere toppmoderne metoder basert på eksplisitte fasongrepresentasjoner og konvolusjon. Vår metode for å fullføre 3D fasonger er basert på å søke etter fasonger bedt i et latentrom som best anpasser seg til sensordata.

Vi trente dype nevrale nettverk til å representere fasongen til objekter i latentrom ved å approksimere deres fortegnede avstandsfunksjon (SDF): en 3D koordinatfunksjon. Vi eksperimenterte med størrelsen til disse nettverkene, med LReLU og sinus ikke-lineæriterer, og med hvordan å best trene nettverkene på 3D modellene fra YCB datasettet med forskjellige nye regulariseringsmetoder og tapsfunksjoner. Vi oppdaget at det å trene nettverkene med data fra flere forskjellige objekter i hver treningsbunt førte til en økt konvergeringsrate. Vi demonstrerte at det å trene sinusbaserte nettverk med trunkerte avstander *og* deres romslige deriverte fører til de beste fasonsrekonstruksjonene scoret med Camfer avstand, earth movers distance, mesh cosinussimilærhet og F-score.

Vårt hovedbidrag i denne oppgaven var å konstruere et latentrom av ikke bare et bredt utvalg av fasonger, men av kontinuerlig orienterbare fasonger, som effektivt kombinerer fasongrekonstruksjonen med stillingsestimering. Dette motiverte nettverkene til å lære rotasjonsmessig uavhengige fasongtrekk. Vi analyserte hvordan nære fasonger klynget seg sammen og gled mellom hverandre i latentrommene formet av auto-dekodere. Vi foreslo en ny metode for å fordele treningseksemplene gjennom en treningsepoke, for å øke konvergeringsraten. Vi foreslo også en ny metode for å beregne fortegnede avstander fra dybdesensordata tatt fra et enkelt synspunkt i virkeligheten. Vi viste frem hvordan vår modell klarer å fullføre fasonger fra forurenset og bedekket sensordata tatt fra et enkelt synspunkt. Basert på disse resultatene er vår metode et verdifullt bidrag til robotisert visuell forståelse.

Forord

Denne masteroppgaven ble utført over vårsemesteret 2021 på Institutt for datateknologi og informatikk (IDI) på Norges teknisk-naturvitenskaplige universitet (NTNU). Oppgaven er også en del av GentleMAN-prosjektet ved SINTEF Ocean som har som mål å utvikle et maskinlæringsrammeverk som bruker visuelle og taktile sanser til å assistere robotisert håndtering av føyelige objekter ved å utruste dem med 3D syn. Jeg vil takke Ekrem Misimi og Theoharis Theoharis for deres veiledning og korrekturlesing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Humans exhibit an incredible capability for visual understanding. We rapidly learn new tasks in a way that still to this day continues to defy our expectation and understanding. We continually learn new relations by combining our senses with our prior experiences, learning from demonstrations by others and through self-exploration. We aim in this thesis to give machines the visual understanding required to perform complex object manipulation tasks. For this purpose we explore novel techniques to teach machines to see and understand a scene and all of the objects in it. The intent is to enable the manipulation of these objects with a robotic arm.

Grasping unknown objects is a complex operation plagued with ambiguities: Which object do we want to grab? What *is* the object we are about to grab? What do the hidden parts of the object look like? Where are the most affordable spots to place the grippers? How much force can one safely apply to the object to successfully lift it without deforming it? These are central questions in the field of visual servoing, with a plethora of different approaches one might take. In this thesis we look to humans for inspiration:

Humans can infer the occluded parts of objects from just a *single viewpoint*. This incredible inference ability draws from our previous experiences and knowledge of shapes commonly found in a given context. Drawing from these priors we arrive at a good guess about the backside and other occluded parts of the object. This guess equips us with a lot of information to then determine how to grip and manipulate the object.

The aim of this thesis is to investigate a new approach on how to make a machine agent able to infer the full shape of an object from just a single viewpoint. This is an area of research in computer vision commonly referred to as *shape completion*: inferring a full 3D shape from only a partial observation. Single-view perspectives are subject to many occlusion states, such as self object occlusion, inter-object occlusion and background occlusion. We present a motivational scenario riddled with occlusions in figure 1.1.

Recent developments in the field of representation learning have resulted in the novel discovery of neural fields, a deceptively simple and powerful data-driven learning paradigm. These networks learn continuous implicit representations, such as the signed distance field whose zero level set is an implicit surface. They do so in a memory efficient and expressive manner, enabling the reconstruction of shapes with arbitrary mesh resolution, mesh topology and genera. We leverage mathematical properties of these implicit functions to boost feature extraction and the reconstruction quality, then apply these methods to single-view shape completion.

FIGURE 1.1: An example scene showcasing occlusions a robotic arm needs to be able to handle. The mug is subject to self occlusions as its handle occludes a part of its own body. The sugar box is subject to inter-object occlusions, as it is blocked by both the strawberry and the mug. The mug occludes the light cast onto the sugar box, further impeding its classification. The banana is largely cut off by the mug, but with previous knowledge about the typical depth of a mug it is possible to infer the length of the banana.

This master thesis is part of the GentleMAN project at SINTEF Ocean[1], aiming to develop learning frameworks using visual and tactile sensing for manipulation of 3D compliant objects with a robot controller by equipping it with 3D vision. As a part of Work Package 1 - Visual Intelligence - this assignment fits with the Task aiming to develop novel 3D reconstruction methods for robotic applications in the presence of the intra object occlusions but also those including physical occlusions, resulting in partial visual observability of the object to be manipulated.

Most related works have trained on and been evaluated against synthetic data, whereas we target a real-world robotic lab environment. As such we base our work on the YCB object dataset (Calli, Walsman, et al. 2015; Calli, Singh, et al. 2015), which is a benchmark for visual servoing.

[1] https://prosjektbanken.forskningsradet.no/en/project/FORISS/299757?Kilde=FORISS&distribution=Ar&chart=bar&calcType=funding&Sprak=no&sortBy=score&sortOrder=desc&resultCount=30&offset=0&Fritekst=gentleman

1.2 Problem Formulation

Succinctly, our problem formulation is as follows:

> Shape completion of 3D objects only partially observable to the visual sensor due to single-view occlusions, and the generation of a 3D mesh models with an accurate camera space pose. The resulting 3D shape completion technique must be fit for use with real-world depth sensor data.

1.3 Research Goals

Our primary goal is to create a deep learning framework to infer 3D shapes from a single viewpoint, satisfying the requirements stated in our problem formulation. To help achieve our primary goal we define these respective sub goals:

T1 Investigate previous state-of-the-art single-view shape completion approaches.
T2 Define and design a deep learning model for single-view shape completion.
T3 Implement and train this model with the YCB object dataset.
T4 Evaluate and discuss the results for 3D single-view shape completion.
T5 Outline future work.

1.4 Structure of the Thesis

This thesis is structured as follows:

Chapter 1 introduces the topic of this thesis.
Chapter 2 covers relevant theoretical background, with recent novel developments.
Chapter 3 describes technical background information pertinent to our implementation.
Chapter 4 explores related works this thesis builds on.
Chapter 5 outlines our approach and methodology.
Chapter 6 presents and evaluates measured results and findings.
Chapter 7 discusses details our evaluation revealed and how it fared.
Chapter 8 concludes our findings and proposes further work.
Appendix A contains supplementary information and explanations, deemed excessive for the main thesis.

Chapter 2

Theoretical Background

This chapter covers relevant theory to understand our approach and methodology. We encourage the reader to examine referenced sources for further insight. A majority of this chapter is either adapted or reprinted from the preparatory specialization thesis by the same author (Sundt 2020).

Section 2.1 covers ways to represent 3D objects and implication for machine learning.
Section 2.2 explains common 3D transformations and depth sensor processing.
Section 2.3 dives into machine learning, covering many techniques and concepts discussed or used in this thesis.

2.1 3D Shape Representations

There are many ways of representing 3D shapes and objects. Section 2.1.1 briefly goes through terms used to describe different classes of objects. Then section 2.1.2, 2.1.3 and 2.1.4 go over common ways to explicitly represent these 3D shapes (illustrated in fig. 2.1), while comparing their trade-offs in fidelity and efficiency with machine learning in mind. Section 2.1.5 then explores implicit functions and surfaces, only recently leveraged in shape representation learning.

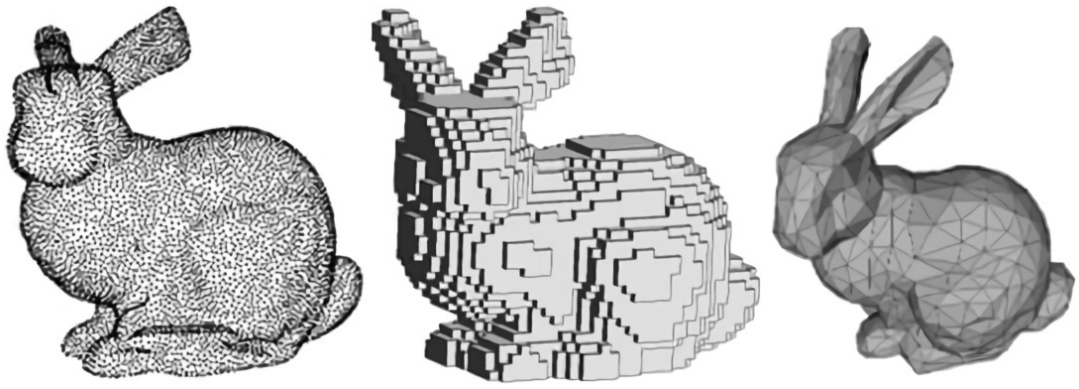


FIGURE 2.1: The Stanford Bunny represented as a surface point cloud, as a occupancy grid voxel model, and as a triangular surface mesh. (From Hoang et al. 2019.)

2.1.1 Object Topology

Topology is a subfield in mathematics concerned with properties of geometric objects. Specifically the properties preserved during deformations that do not cause tears in the surface. Topology

introduces the concept of the object *genus*: the number of "holes" in objects. For example: a sphere has a genus of 0, a torus has a genus of 1, and a mug as a genus of 1. As far as the field of topology is concerned, a torus and a mug are identical.

Pointing outward of surfaces are *normal* vectors: a vector scaled to unit length denoting the orientation of the surface. These vector define a tangental plane along the surface, intersecting the base of the normal vector.

Mesh topology is a somewhat related concept. It describes the layout of vertices on a mesh, and how they connect to each other into *faces*. A normal vector can be derived for each face, assuming the mesh follows a winding direction convention. Each vertex may optionally include their own normal vectors.

2.1.2 Point Cloud Representations

A *point cloud* is a set or collection of 3D Cartesian coordinates, also known as *points* or *vertices*. A 3D object may be represented as a cloud of vertices. The vertices may contain additional data such as color or other physical attributes such as density. For 3D graphic purposes, these vertices are usually sampled along the surface of the object. Point clouds are a good light-weight representation for raw data from sensors such as depth cameras and LiDAR scanners. Information such as object topology is not trivially represented in point clouds and must be inferred. The Stanford Bunny can be seen as a surface point-cloud in fig. 2.1.

2.1.3 Mesh based Representations

A *mesh-based* 3D object representation consists of a list of *vertices* sampled along the surface of the object, along with a list a of *faces* defined as a sequence of at least three or more *edges*. An *edge* is a pair of two connected vertices. The faces, also known as polygons, of a mesh are usually in the form of triangles. A triangular surface mesh of the Stanford Bunny is visualized in fig. 2.1.

One of the greater strengths of mesh-based object representations is that it encodes many topological qualities. As such, meshes are well suited for further analysis.

A problem with meshes is that they do not easily map to a intuitive machine learning architecture, due to their high irregularity. These learning architectures are either not guaranteed to produce non-degenerate watertight[1] meshes, or are limited to a fixed mesh topology. The non-uniformity and irregularity of meshes inhibits efforts using neural networks that combine convolution and pooling operations.

2.1.4 Voxel based Representations

A voxel model is a 3D grid of discrete samples covering a volume. A surface can be extracted from a voxel model by defining a boundary condition. If "density" is sampled, then the surface can simply be defined by a target threshold density. The most common form of voxel 3D models are *occupancy grids*, where the samples are limited to $\{0, 1\}$. Figure 2.1 illustrates the Stanford Bunny as an occupancy voxel model. It is not unusual for voxels models to sample fields such as the signed distance field (SDF). Voxel models sampling a continuous field can be converted into a mesh using the *marching cubes* algorithm.

Voxels are the most straight forward extension from the 2D image domain, as 2D learning techniques such as convolution can be directly applied. Voxels have proved not to be an efficient surface representation however, neither computationally nor with regards to memory use. This is primarily due to the square-cube law. As the surface fidelity scale in a squared manner, the computation and memory requirements scale cubically. As such, current voxel based machine learning methods can only handle smaller resolutions up to 128^3. Some works manage to

[1] Watertight meshes consist of *closed* surfaces, that do not have any holes and have a clearly defined inside.

push the effective resolution up to 512^3 by using octrees[2] to omit areas of lower complexity (Tatarchenko, Dosovitskiy, and Brox 2017).

2.1.5 Implicit Surface based Representations

An *implicit surface* is defined as the 0-level set or isosurface of a 3D function f:

$$f(x, y, z) = 0 \tag{2.1}$$

"Implicit" refers to how f is not solved for x, y or $z \in \mathbb{R}$. Explicit surface representations can depending on the type of implicit function be extracted numerically.

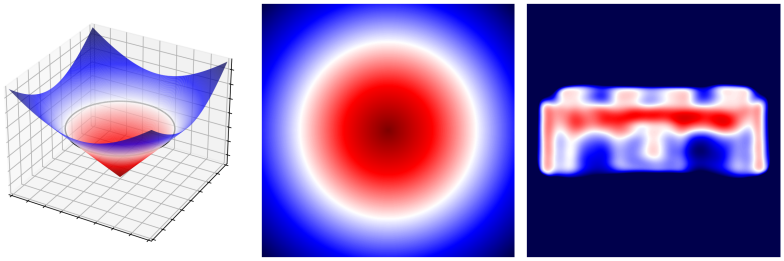
2.1.5.1 Signed Distance Functions and Fields (SDF)

A *signed distance function* is a function that "queries" a *signed distance field*. The SDF abbreviation refers to both. The absolute value of the field describes the distance to the nearest surface. The sign of the field denotes whether the point is on the inside or on the outside of the object, with positive distances being on the outside.

SDFs embed implicit surfaces as their 0-level set or isosurface. We show in equation 2.2 the SDF of a sphere centered in $\mathbf{p} \in \mathbb{R}^3$ with radius $r \in \mathbb{R}$, expressed using both p-norm notation and expanded to a simplified form where x_i is the i'th scalar in the vector $\mathbf{x}$ and likewise for p_i and $\mathbf{p}$.

$$\begin{aligned} \mathrm{SDF}_{\mathrm{sphere}}(\mathbf{x}) &= \|\mathbf{x} - \mathbf{p}\|_2 - r \\ &= \sqrt[2]{(x_1 - p_1)^2 + (x_2 - p_2)^2 + (x_3 - p_3)^2} - r \end{aligned} \tag{2.2}$$

We present a rendered 2D slice of the sphere, or put differently a circle, in figs. 2.2a, 2.2b. We additionally showcase a Lego piece slice in fig. 2.2c.



(A) Circle in 3D (B) Circle (C) Lego

FIGURE 2.2: The surface distance field of a 2D circle and of a Lego cross-section. White is near 0, blue is positive and red is negative.

The SDF of two objects can be combined into a single SDF using the *min* function. This is the basis of the field known as constructive solid geometry (CSG).

If we require $\mathrm{SDF}(\cdot)$ to be continuous, to be over Euclidean space, and to cross the boundary at least once then it is guaranteed that $\mathrm{SDF}(\cdot) = 0$ defines a surface without any holes. The shape is guaranteed to be watertight if we require $\lim_{\|\mathbf{x}\|_2 \to \infty} \mathrm{SDF}(\mathbf{x}) = \infty$. We can render surfaces described by SDFs by either ray-marching them, or by rasterizing a mesh created with *marching cubes*.

[2] Octrees are tree structures where each node has 8 children. Octrees are commonly used to recursively partition or subdivide 3D volumes or cubes into eight octants.

Since neural networks are universal function approximators, they can be trained to predict the signed distance value at any query point. The shapes inferred by these networks are not limited to any kind of mesh topology or genera. The output is just a single scalar, making the size of the network a lot smaller than explicit shape representation networks that compute multiple points, triangles, or samples at a time.

A nice property of SDFs is that the normal vector of the isosurface can be computed analytically as the spatial gradient $\nabla_{\mathbf{x}}\text{SDF}(\mathbf{x}) = \frac{\partial \text{SDF}(\mathbf{x})}{\partial \mathbf{x}}$. If the SDF is over Euclidean space and has a piecewise smooth zero boundary then $\|\nabla_{\mathbf{x}}\text{SDF}(\mathbf{x})\|_2 = 1$. The spatial derivative can be derived via backpropagation if the SDF is approximated by a neural network (Park et al. 2019).

2.2 Transformations and Processing

This section covers basic spatial transformations and their relation to depth maps and machine learning. A *transformation* is a function or operation transforming some input to some output. Transforms are often non-singular, enabling corresponding *inverse* transformations to undo the original transformation. We cover here transformations that can be performed on 3D objects.

2.2.1 Affine Transformations

We can perform linear transformations on points using matrix multiplication. When transforming a 3D coordinate, we first convert it to homogeneous coordinates: a 4D vector where the first 3 scalars come from the original 3D vector, with the last scalar set to 1. Using homogeneous coordinates allows us to perform any affine transformation with matrix multiplication. The added 1 acts as a *bias* term, mainly reserved for the *translation* transform and projection.

All the basic affine 2D transformations (3×3 homogeneous matrices) are showcased in table 2.1. Similar 4×4 matrices exist in 3D space. Other transformations, such as reflection, can be decomposed into a combination of these basic affine transforms. Multiple transformation matrices $\mathbf{M} \in \mathbb{R}^{4\times4}$ can be composed into a single transformation using matrix multiplication:

$$\mathbf{M}_{\text{translate}} \times \mathbf{M}_{\text{rotate}} = \mathbf{M}_{\text{rotate then translate}} \tag{2.3}$$

Matrix multiplication is a noncommutative operation, making the order transformations are applied matter. Combining multiple matrices into a single matrix greatly reduces the amount of computations needed to transform a large batch of 3D vectors.

TABLE 2.1: The basic 2D affine transformation matrices.

Name	Transformation matrix
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$
Scale	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotate	$\begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Shear	$\begin{bmatrix} 1 & c_x & 0 \\ c_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

2.2.2 The Model-View-Projection Matrix

Here we briefly cover projection to better understand unprojection. We perform a sequence of transformations when rendering a 3D object in a 3D scene onto a 2D canvas. These following four terms are common when discussing where along this sequence of transformation we currently are:

MCS - Model Coordinate System: the coordinate system used to define the vertices of a single 3D model.

WCS - World Coordinate System: the shared coordinate system used within a scene, unifying all the models in it.

ECS - Eye Coordinate System: a coordinate system where the camera is centered at the origin.

CSS - Clipspace: the coordinate system of the *viewport*, which is a frustum extruded from the edges of the canvas in ECS, and "squeezed" into a cube in CSS.

One transforms the vertices and normal vectors of a mesh from on system to another using transformation matrices. The matrix transforming from MCS to WCS is known as the *Model* matrix, from WCS to ECS is the *View* matrix, and ECS to CCS is the *Projection* matrix. Composed they form the model-view-projection (MVP) matrix $\mathbf{M}_{\text{MVP}} \in \mathbb{R}^{4\times4}$:

$$\mathbf{M}_{\text{MVP}} = \mathbf{M}_{\text{Projection}} \times \mathbf{M}_{\text{View}} \times \mathbf{M}_{\text{Model}} \tag{2.4}$$

The full *perspective* transform is not an affine transformation, as it requires an additional *perspective divide*:

$$\mathbf{M}_{\text{MVP}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} uw \\ vw \\ dw \\ w \end{bmatrix} \xrightarrow{\substack{\text{perspective} \\ \text{division}}} \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix} \tag{2.5}$$

where x, y, z are MSC coordinates, u, v are canvas coordinates, and d is the viewport depth.

For our purposes we use the term *camera space* for ECS and *object space* for MCS. We use the concept of a *canonical pose* when constructing a normalized coordinate system for learned shapes.

2.2.3 Unprojecting RGB-D Images to Point Clouds

RGB-D images contain both color (RGB) and depth (D) information. Using the depth information it is possible to unproject the pixels of the image into a point cloud (see section 2.1.2) using the inverse of the *intrinsic camera matrix*:

$$\mathbf{K}^{-1} \begin{bmatrix} ud \\ vd \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{2.6}$$

where $\mathbf{K}$ is the intrinsic matrix, u, v are the pixel coordinates, d is the measured depth, and x, y, d are the camera space coordinates.

The *intrinsic camera matrix* is a 4×4 transformation matrix for calibrating ideal pinhole cameras. They embed the sensor resolution, the focal length, and the focal point. If w from equation 2.5 is known then one can instead use the inverse of the projection matrix.

In addition to these linear parameters, there are a couple of nonlinear intrinsic parameters to account for. Real world cameras can not be treated as ideal pinhole cameras, as they

suffer from lens distortions. It is common to model two major kinds of lens distortion: radial distortion (eq. 2.7) and tangential distortion (eq. 2.8). These distortions are defined as infinite series. OpenCV and BigBIRD (Singh et al. 2014) has deemed 3 and 2 terms of the respective distortions sufficient, and calibrate 5 distortion coefficient: $(k_1, k_2, p_1, p_2, k_3)$.

$$\begin{aligned} x_{\text{distorted}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\text{distorted}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.7}$$

$$\begin{aligned} x_{\text{distorted}} &= x + (2p_1 xy + p_2(r^2 + 2x^2)) \\ y_{\text{distorted}} &= y + (p_1(r^2 + 2y^2) + 2p_2 xy) \end{aligned} \tag{2.8}$$

where x and y are coordinates along the image plane with the distortion centered at 0, and $r = \sqrt{x^2 + y^2}$.

More often than not are the RGB and D images captured with different cameras from slightly different perspectives. As such it is common to calibrate an *extrinsic* transformation matrix that transforms from the infrared (D) camera to the RGB camera. It is now possible to find the corresponding color of a depth pixel by first correcting for depth camera distortion, then unproject the point into 3D, transform it to the color camera coordinate system, project it onto the image plane, then finally apply the color camera distortion.

One can in a multi-camera setup (BigBIRD, see sec. 3.1.2) select a common reference point for all the cameras. This allows us to maintain only a single extrinsic transformation matrix from that reference point to each camera. One can then construct a transformation matrix $\mathbf{M}_{A \to B}$ between any two cameras A and B with the following equation:

$$\mathbf{M}_{\text{Ref} \to B} \times \mathbf{M}^{-1}_{\text{Ref} \to A} = \mathbf{M}_{\text{Ref} \to B} \times \mathbf{M}_{A \to \text{Ref}} = \mathbf{M}_{A \to B} \tag{2.9}$$

2.2.4 6D Continuous Representation of Rotation

All possible rotations in 3D space about the origin (i.e. the 3D rotation group $SO(3)$) can be represented using Euler angles: a vector in $\mathbb{R}^3$. Euler angles are however affected by the gimbal lock problem[3], making it tricky for machine learning to properly infer the rotation for a given observation. The gimbal lock problem problem was alleviated in computer graphics by using a different representation rotation in $\mathbb{R}^4$: quaternions.

Zhou et al. (2019) show that neither Euler angles nor quaternions are well suited for ReLU-based machine learning due to their discontinuities. They further go on to prove that all representations of *SO(3)* within four or fewer dimensions must be discontinuous. They demonstrate empirically that the continuous 6D representation $\mathbf{b} = (\mathbf{b_x}, \mathbf{b_y})$ where $\mathbf{b_x}, \mathbf{b_y} \in \mathbb{R}^3$ yield far better results in machine learning applications. This 6D representation can be converted into a 3D rotation matrix $R \in \mathbb{R}^{3\times3}$ given by:

$$R = \begin{bmatrix} | & | & | \\ \mathbf{r_x} & \mathbf{r_y} & \mathbf{r_z} \\ | & | & | \end{bmatrix}, \qquad \begin{aligned} \mathbf{r_x} &= N(\mathbf{b_x}) \\ \mathbf{r_z} &= N(\mathbf{r_x} \times \mathbf{b_y}) \\ \mathbf{r_y} &= \mathbf{r_z} \times \mathbf{r_x} \end{aligned} \tag{2.10}$$

where $\times$ is the vector cross product, and $N(\cdot)$ is the unit vector normalization function.

The two 3D vectors $\mathbf{b_x}$ and $\mathbf{b_y}$ are very resistant to becoming malformed or degenerate. They encode a valid rotation provided they are non-zero and linearly independent, making them ideal targets for inference. Their canonical form has $\|\mathbf{b_x}\|_2 = \|\mathbf{b_y}\|_2 = 1$ and $\mathbf{b_x} \times \mathbf{b_y} = 0$. A

[3]The gimbal lock problem occurs when two of the axes of rotation are driven into a parallel configuration. This causes the loss of a degree of freedom.

6D rotation vector can be normalized into its canonical form cheaply using two unit vector normalization operations and a single Gram–Schmidt orthogonalization.

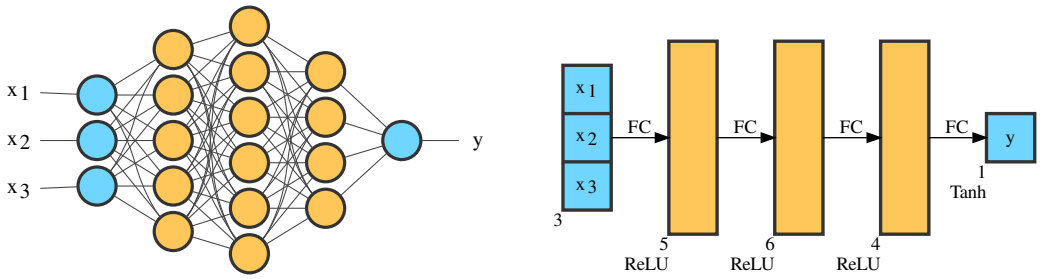
2.3 Machine learning

Machine learning is a field within artificial intelligence with the aim of developing programs that can learn from data and perform tasks not explicitly programmed. Machine learning has close ties with statistics and mathematical optimization.

2.3.1 Artificial Neural Networks

Neural networks are commonly thought of as universal function approximators. These networks consists of neurons with connections between them. The connections may perform transformations on the incoming data passing through. Artificial neural network are often constructed with *layers* of neurons: a input layer, some hidden layers and an output layer. If the network has no hidden layer then it is known as a single-layer perceptron. Networks with at least one hidden layer is known as a multilayer perceptron (MLP).

A *transfer function* is applied between each layer and is responsible for propagating and combining the data between the neuron layers. The neurons may each have an *activation function* introducing nonlinearities into the system. Nonlinearities make the network better able to learn complex relations, and is the reason why we are able to train these network via backpropagation.[4] In figure 2.3 we illustrate a fully connected neural network both as a graph of neurons and as a pipeline of transfer and activation functions.



(A) Nodes and edges (B) Layers and interconnect

FIGURE 2.3: A simple neural network with 3 inputs and a single output; a simple multilayer perceptron. It has 3 hidden fully connected layers of size 5, 6 and 4, respectively. FC is short for Fully Connected. The labels at the bottom of each layer in (B) denote their width and activation function.

2.3.2 Transfer Functions

In this thesis we primarily use linear layers and vector concatenation. Linear layers, also known as fully connected layers, defines a matrix of learnable weights A and vector of biases $\mathbf{b}$, and applies a linear transform on the incoming data with $\mathbf{x}A^T + \mathbf{b}$. Vector concatenation is simply the joining of two vectors, where the scalars in one is followed by the scalars of the other: $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n+m}$. Common notation for vector concatenation is $(\mathbf{x}, \mathbf{y})$ and $\mathbf{x} \oplus \mathbf{y}$.

[4] Optimization of the neural networks is based on the derivative of the learnable parameters with regard to the error of their predictions. The derivative of the network would be constant with no relation to the input if the network is fully linear.

2.3.3 Activation Functions

In artificial neural networks, the *Activation Function* transforms the output from a node/layer before passing it on to the next layer. Different activation functions introduce different mathematical properties to the system. Common properties to consider include: linear vs nonlinear, range, whether it is continuously differentiable and how approximate the function is to the identity near the origin. We list a handful of common activation functions in table 2.2, further graphed in figure 2.4.

TABLE 2.2: A collection of common activation functions used in neural networks, some relevant to computer vision.

Name	Description	Activation $f(x)$	Derivative $f'(x)$
Identity	Linear	x	1
Binary step	Easy to convert to a circuit, but difficult to train	$\begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
ReLU	Rectified linear unit, has a monotonic derivative	$\max\{0, x\}$	$\begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$
LReLU	Leaky ReLU, its derivative is never zero.	$\max\{0.01x, x\}$	$\begin{cases} 0.01 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$
SiLU	Sigmoid linear unit, a smooth approximation of ReLU	$x \cdot \sigma(x) = \frac{e^x x}{e^x + 1}$	$\frac{e^x(1+e^x+x)}{(1+e^x)^2}$
SIREN	Approximates identity near origin, periodic, has infinite derivatives	$\sin(\omega_0 x),\ \omega_0 \in \mathbb{R}_{>0}$	$\omega_0 \cos(\omega_0 x)$
Tanh	Nonlinear, cheap to compute derivative, approximates identity near origin	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$
Sigmoid	Nonlinear, cheap to compute derivative, always positive	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f(x)(1 - f(x))$

Note: What denotes a SIREN network is not the sinusoidal activation function alone. SIRENs use a principled initialization scheme based on the ω_0 hyperparameter to draw the initial weight of the network from a uniform distribution: The first layer is drawn from $\mathcal{U}\left(-\frac{1}{n}, \frac{1}{n}\right)$, and the following layers are drawn from $\mathcal{U}\left(-\frac{1}{\omega_0}\sqrt{\frac{6}{n}}, \frac{1}{\omega_0}\sqrt{\frac{6}{n}}\right)$, where n is the number of input features. (Sitzmann, Martel, et al. 2020)

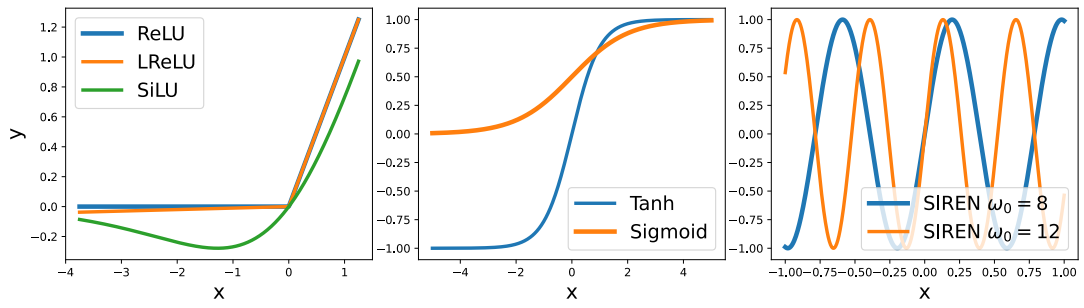


FIGURE 2.4: Plots of ReLU, LReLU, SiLU, Tanh, Sigmoid and SIREN activations.

2.3.4 Supervised Training

Machine learning is based on *learning* behavior and relations. It is commonly through *training* that a network is able to learn. The goal of the network Φ_θ is to map some input $\mathbf{x}$ to some desired output $\mathbf{y}$. The training is *supervised* if it has access to examples of desired output for some given input.

To train a network Φ_θ one start by initializing the learnable *parameters* θ of the network with random values. Xavier initialization[5] is the most common scheme. To train a neural network we:

- Pick a random training *sample* from the training *dataset*: $(\mathbf{x}, \mathbf{y}) \in \Omega$
- Ask the neural network to *predict* the output: $\mathbf{y}_{\text{prediction}} = \Phi_\theta(\mathbf{x})$
- Compute the *loss* (error) by comparing the network output to the known ground truth value, using some loss function: $\mathcal{L}(\mathbf{y}, \mathbf{y}_{\text{prediction}})$
- Compute the derivative of the loss of the network with regard to the learnable network parameters, through *backpropagation*[6]: $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}(\mathbf{y}, \Phi_\theta(\mathbf{x}))}{\partial \theta}$
- Adjust the learnable *weights* θ of the network using the loss gradient $\nabla_\theta \mathcal{L}$, according to some *optimization strategy*: $\theta_{\text{next}} = \text{Optimize}\,(\theta, \nabla_\theta \mathcal{L})$
- Repeat

Non-learnable parameters are commonly known as hyperparameters. These are often set by humans, but can be derived through automatic means such as Bayesian Optimization.

2.3.4.1 Overfitting

A neural network unable to generalize has a low error on the training dataset but a high error on new unseen data. This may be caused by insufficient training data or poor modeling of the network. A certain class of poor generalization is known as *overfitting*: when the network has been too closely fitted to a limited training dataset.

A common approach to observe overfitting is to split the dataset into three parts known as *train*, *validation* and *test*. During training the network should only ever update its weight using the training dataset, while periodically monitoring the network performance across the *validation* data. The network is considered to be overfitting when validation error increase despite the training error remaining low.

The final measurements and benchmarking of a network is done using a separate and unseen *test* dataset, due to hyper-parameters usually being selected based on validation performance.

2.3.4.2 Generalization through Regularization

It is customary to use different regularization techniques to increase the ability of a network to generalize over the training data, in turn making it more robust to outliers and noise.

The simplest regularization scheme is one commonly known as *early stopping*, where you monitor the network and terminate its training when the validation accuracy stagnates.

Another regularization technique is *data augmentation*. The basic idea is to artificially increase the amount of training data. These are domain dependent random transformations applied to the input and target output examples used during training. For 3D point clouds these augmentations could be added stochastic noise, affine transformations or occlusions. Data augmentation can be leveraged to ensure the network is resistant to certain classes of reductions.

[5] Xavier initialization is a principled initialization scheme drawing initial weights from a normal distribution, but may also use a uniform distribution. It is also known as Glorot initialization.

[6] Backpropagation refers to the computation of gradients with respects to some input, often being the learnable network parameters with respect to the loss. All the parameter gradients are computed using the chain rule when "backpropagating" a forward pass. The term also refers loosely to gradient based network optimization.

For image recognition networks, the input images can be augmented with random rotations to ensure that upside-down images do not cause erroneous predictions.

Noise, commonly drawn from a Gaussian distribution, is often injected into the network input to prevent it from overfitting on irrelevant high-frequency patterns. Quantization error from a reduced floating point precision may act as regularization in the same manner other noise distributions would. (Micikevicius et al. 2018)

Dropout is a regularization technique in which you randomly set the output of a neurons to zero, effectively disabling it. This technique is used to fit only a random subset of the neurons at a time. It is usually used to regularize the fully connected layers and ensure the learned relations and patterns are evenly spread throughout the network.

When computing the *loss* of a network during training, L1 and L2 loss (eq. 2.11) are often the ones used. L1 estimates the median of the data while L2 estimates the mean, based on Lasso Regression and Ridge Regression respectably. Both loss functions are subject to omitted-variable bias, making them to not punish certain features which could produce a more accurate prediction.

$$\begin{aligned} \mathcal{L}_{\text{L1}}(y_{\text{ground truth}}, y_{\text{predicted}}) &= |y_{\text{ground truth}} - y_{\text{predicted}}| \\ \mathcal{L}_{\text{L2}}(y_{\text{ground truth}}, y_{\text{predicted}}) &= (y_{\text{ground truth}} - y_{\text{predicted}})^2 \end{aligned} \tag{2.11}$$

The idea behind both L1 and L2 loss is to penalize all features equally. If the features are scaled differently however this assumption may break. As such it is customary to include some form of *normalization* scheme in the network. Network normalization primarily come in the following three flavors:

Input normalization: The network input is normalized to always be represented in some canonical form and amplitude. Quaternions and 6D rotation vectors (sec. 2.2.4) can be normalized with quaternion standardization[7] and Gram–Schmidt orthogonalization respectably. Positional encoding (see sec. 2.3.4.3) can be thought of as input normalization.

Layer normalization: These are dedicated layers in the network that normalize the whole layer (Ba, Kiros, and Hinton 2016) before passing it further down the network. The coefficients used in this linear scaling operation are learnable parameters subject to optimization. Also commonly known as *batch normalization*.

Weight normalization: Here the weights of fully connected layers are decoupled into separate magnitude and a direction components, both subject to optimization as learnable parameters. (Salimans and Kingma 2016) Weight normalization is often faster to compute than batch normalization, as it is not a network feature but instead a reparametrization trick only performed in between optimization steps.

Networks with either layer normalization or batch normalization have been both theorized and shown to learn efficiently within a much wider range of learning rates. They adapt to higher learning rates by decreasing the scale feature and increasing the magnitude of the direction features. (Salimans and Kingma 2016)

2.3.4.3 Positional Encoding

Positional Encoding was first proposed and used in NeRF by Mildenhall et al. (2020). Observing that multilayered perceptrons are biased towards learning low-frequency relations on the input, they mapped the inputs to higher dimensional space using high frequency functions. This enables better fitting of data that contains high frequency patterns and variation. $\gamma_n(\cdot) : \mathbb{R} \to \mathbb{R}^{2n+1}$ elaborated below in eq. 2.12 maps a continuous input p into $2n + 1$ features.

$$\gamma_n(p) = \big(p, \sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{n-1}\pi p), \cos(2^{n-1}\pi p)\big) \tag{2.12}$$

[7] Quaternion standard form: in which the real part is non-negative.

2.3.4.4 Gradient Decent - Adaptive Moment Estimation (Adam)

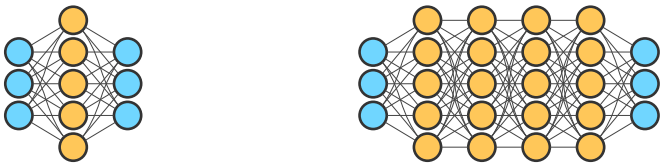
Neural networks are trained through gradient decent: gradient-based iterative methods that satisfy an objective function. We specifically use Adam: adaptive moment estimation, based on adaptive estimation of first and second-order moments. (Kingma and Ba 2017) It is a stochastic approximation of gradient descent using the gradients calculated through network backpropagation. It takes care to roll past smaller local optimums in parameter space to find one off the better solutions within the wider basin of attraction. It is computationally efficient for problems with a large scale of learnable parameters. It handles sparse gradients on noisy problems quite well, and reduce the need for fine-tuning hyperparameters or online adjustments during training. Adam primarily use three hyperparameters: a learning-rate η determining the learning speed, and two decay rates β_1 and β_2.

2.3.5 Deep Learning

Deep learning is a class of machine learning using artificial neural networks, with "deep" meaning the networks have more than one *hidden layer*, as illustrated in fig. 2.5. Stacking more layers allow the network to learn more complex relations thanks to the addition of more nonlinearities and higher number of parameters to embed relations into.

Deep networks have different inductive biases than wide networks. Classically the universal approximation theorem only proves that arbitrarily wide network with a bounded depth can approximate any signal with arbitrary precision. It has since been expanded to include arbitrarily deep network with a bounded width. Deep networks are cheaper to compute than wide networks, as stacking nonlinearities is cheaper than scaling up matrix multiplication. Deeper networks are however more affected by exploding or vanishing gradients, where the gradients either increase or decrease due to the repeated application of nonlinear activations.

The term *deep learning* is also used to describe how the network aims to generalize from a training dataset, as opposed to *reinforcement learning* in which more dynamic agents perform actions and learn through trial and error seeking some reward.



(A) Feed-forward network (B) Deep feed-forward network

FIGURE 2.5: A shallow neural network compared to a deep neural network.

2.3.6 Representation Learning and Latent Spaces

Representation learning or feature learning is the study of extracting features from complex raw data such as images and 3D models. Representation learning use GANs, AEs, VAEs and ADs (covered in the following sections) to learn these features, embedded as structures in latent spaces.

Latent spaces are a central concept in representation learning, simply representing compressed data in a high-dimensional hidden space. One can *encode* data as a latent space coordinate, and *decode* it to reconstruct the original data. These spaces are mathematically and computationally convenient to process, as they are designed to cluster similar data points closer together by their structural similarities, or form manifolds. As such they enable us to better understand complex data and its patterns.

A *manifold* is in data science a high-dimensional space that locally resemble smooth Euclidean

spaces. Manifolds are spaces where similar data points group and cluster in such a way that transitions are smooth, without sharp “spikes”, “edges” or “holes”. A common mental model for manifolds is that a small ant walking along the manifold “plane” believe from its point of view that the plane is flat.

Latent spaces enable many means of data analysis. *T-distributed Stochastic Neighbor Embedding* (t-SNE) is a stochastic dimensionality reduction technique that groups data points by affinity (Maaten and Hinton 2008). It is designed to project high-dimensional data points into low-dimensional spaces we can intuitively visualize and understand. T-SNE projections are commonly used to reveal the inherent structure of latent spaces, unfolding how data points either clusters or form manifolds.

2.3.7 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a class of neural networks using *convolution* or *pooling* layers instead fully connected layers, to reduce the dimensionality of the input. This reduces the amount of network weights to learn and store. It additionally makes the network better able learn relations uniformly across the whole input instead of overfitting on select input features.

2.3.7.1 Convolution

In 2D image processing, *convolution* is the act of processing an image using a *kernel* matrix. The kernel is a $n \times m$ matrix of coefficients, which you slide across the image and multiply the overlapping pixels in the image with the matching coefficient in the kernel. Then you sum the result and store it as a pixel in the output image. A convolution operation is illustrated in fig. 2.6.

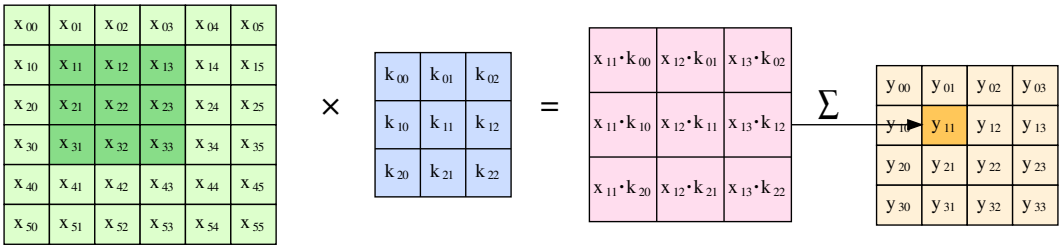


FIGURE 2.6: Image (green) * kernel (blue) = convolution (orange), with an intermediate calculation shown in red.

For a 2D kernel $f(u, v)$ and a 2D image $g(u, v)$, convolution is defined as:

$$(f * g)(u, v) = \sum_i \sum_j f(i, j) g(u - i, v - j) \tag{2.13}$$

Convolution can be generalized to continuous functions over both higher or lower dimensional domains. The mental model equation 2.13 describes is here sufficient to understand how to apply convolution to neural networks. A *convolutional layer* in a neural network is based on the same idea, but can be applied to more dimensions. The input image is the input to the convolution layer, the kernel are the learnable weights in the layer, and the output is the filtered image. Convolution layers use parameter sharing to aid generalization.

Convolution is often used to create image filters such as blurs, sharpening and edge detection filters among many. These kinds of neural network layers are instrumental for tasks such as image classification and segmentation.

2.3.7.2 Pooling

Pooling is a nonlinear down-sampling operation, simpler and cheaper to compute than a convolution layer. The input is subdivided into pools defined by their *dimensions* spaced apart by a *stride*. The pools may overlap if the stride is lower than than the pool dimensions. Each pool is then aggregated using some function, commonly the max function.

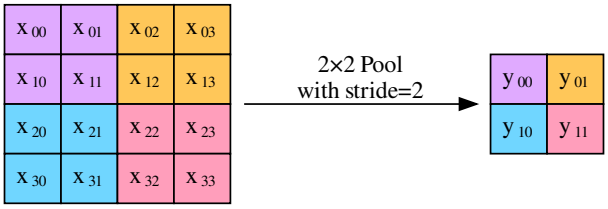


FIGURE 2.7: A 2x2 (max-)pool operation with stride=2, where the separate pools have been visualized with different colors.

2.3.8 Generative Adversarial Networks (GAN)

A Generative Adversarial Network (GAN) consists of a *generator* and a *discriminator* network. The *generator* network generates fake samples similar to the samples found in the training dataset. The *discriminator* network is trained to detect fake samples. The two networks are trained as adversaries: the generator is incentivized to *fool* the discriminator, while the discriminator get more and more skilled at spotting fakes in an arms-race against each other. An overview of the GAN architecture is illustrated in fig. 2.8. GANs are an example of *unsupervised learning*.

The generator network generates samples from random noise passed into it. This random noise vector can be viewed as a coordinate in a latent space representing the training samples, and the generator can be seen as the *decoder* of this latent space.

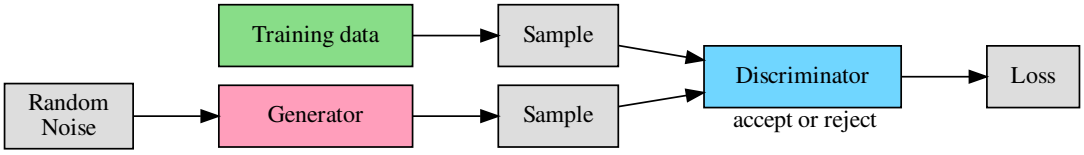


FIGURE 2.8: The architecture of a General Adversarial Network (GAN). It consists of a generator and a discriminator network, each tasked with besting the other. Samples from a training dataset are used to train the Discriminator to tell the fakes generated by the Generator apart from the real ones.

2.3.9 Auto-Encoders (AE)

An auto-encoder is an unsupervised learning technique. Its goal is to reconstruct the original input after first passing it through a bottleneck. The bottleneck lies in between the *encoder* (e) and the *decoder* (d) parts of the network, illustrated in fig. 2.9. The networks train to make $d(e(x)) \approx x$ produce the lowest error possible. The bottleneck becomes a latent vector, also known as a code. The encoder encodes samples into latent vectors, and the decoder reconstruct the samples.

Auto-encoders are well suited for learning relations on the input data without direct supervision with information such as shape orientation or animal species. The decoder part of the network can be further trained in a GAN if it is desirable that all possible latent codes result a believable

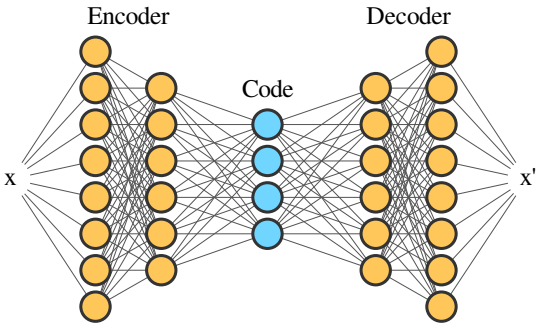


FIGURE 2.9: A simple fully-connected auto-encoder network, with the Encoder and Decoder sections labeled. At the information bottleneck a latent space code emerges.

output, or in other words: that the latent space is highly regular. It is however difficult to ensure that the encoder maps the latent space in such a regular fashion.

Variational auto-encoders (VAE) use reparametrization to perturb the codes with (Gaussian) noise before decoding them again, to encourage smooth and complete latent spaces and avoid overfitting (Kingma and Welling 2019). Instead of modeling the latent code as a single vector within the latent space, it is modeled as a probability distribution over the latent space which is then sampled accordingly. Therefore the code is partitioned into a *mean* and a *variation* feature, each used as parameters for the stochastic sampler. It is possible to chose other distributions than the Normal distribution, but it tends to perform well with when making little to no assumptions about the true underlying distributions.

2.3.10 Neural Fields and Deep Implicit Surfaces

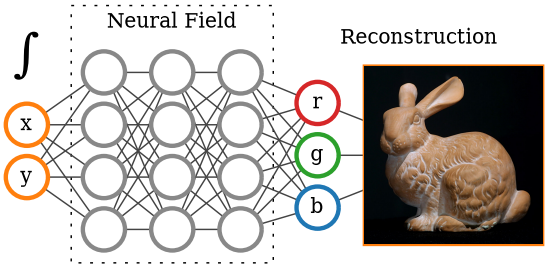


FIGURE 2.10: A 2D RGB neural field and its reconstruction: an image of the Stanford Bunny. One must traverse the two input axes and sample the computed colors to reconstruct the image.

Implicit representation networks approximate functions that query into continuous fields. We illustrate a 2D RGB neural field in figure 2.10. This is a new emerging subfield of representation learning, which has yet to converge on a single generalizing term denoting the concept. We have seen these terms used interchangeably: Implicit representation networks, implicit functions, neural fields, and coordinate-based neural representations.

In this thesis we train neural networks (Φ) to represent implicit 3D surfaces (sec. 2.1.5) of objects, by having them approximate a 3D field that describe some attribute of our shape or surface of interest: the signed distance field (SDF):

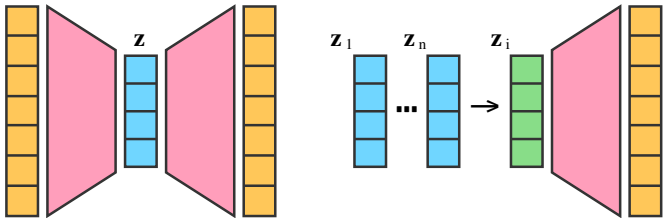
$$\Phi(\mathbf{x}) \approx \text{SDF}(\mathbf{x}) \ : \ \mathbf{x} \in \mathbb{R}^3 \tag{2.14}$$

These coordinate-based neural networks can be viewed as a simplified and continuous variant of convolutional neural networks. Whereas convolution aids generalization by parameter sharing its kernels, coordinate-based networks apply the whole network across the input domain. A weakness of implicit representation networks however are that there is not always an obvious way to supervise encoders for them.

2.3.11 Auto-Decoders (AD)

Auto-encoders are expensive to train, due to having to train both the *encoder* and the *decoder* parts of the network. Often only one of the parts are needed for inference purposes once the training has completed. The unused part are in these cases simply discarded. What if there is no known or suitable input to supervise an encoder with? What if we could omit having to train an encoder at all?

The structural difference between an auto-encoder and an auto-decoder is illustrated in fig. 2.11. In short: omit training any encoder at all and instead learn the optimal latent space vectors directly via backpropagation (Park et al. 2019). This is done by assigning a random latent space code to each unique item in the dataset and optimize these codes as if they were a part of the learnable parameters of the network during training.



(A) Auto-encoder (B) Auto-decoder

FIGURE 2.11: An auto-encoder (AE) compared to an auto-decoder (AD). AE compresses the input down into a latent vector with an *encoder*, then decode it again with its *decoder* trying to match the original input. AD forgoes the encoder and instead maintains a database of n latent vectors (one per item in the dataset), optimizing these vectors along with the rest of the network weights.

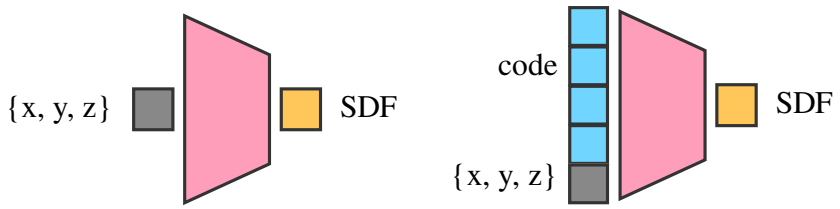
2.3.12 Probabilistic Decoders

A probabilistic decoder is a distribution $p(\mathbf{x} \mid \mathbf{z})$ over each value $\mathbf{x} \in \mathbb{R}^n$ given the code $\mathbf{z} \in \mathbb{R}^m$ (Tran 2016). Here $\mathbf{z}$ can be interpreted as a latent space vector, and $\mathbf{x}$ can be seen as our query into this latent space. These decoders are also commonly referred to as *coded* decoders.

Probabilistic decoders are a great tool for representation learning: Instead of training the weights θ_i for a whole network Φ to represents a single shape i, you could instead train the weights θ_Ω to represent a wider collection of shape priors in Ω. To embed such a large number of shapes into the network you can map each shape i to a point in a latent space, each with their own code $\mathbf{z}_i$, and concatenate this code together with the query location $\mathbf{x}$ before passing it into the network, thus treating it as a coded decoder.

Compared to a simple network where the learned weights θ_i only embeds a single shape i (left in eq. 2.15 and fig. 2.12a), a probabilistic decoder (right eq. 2.15 and fig. 2.12b) is capable of embedding all the shapes $i \in \Omega$ with a single set of learned weights θ_Ω.

$$\Phi_{\theta_i}(\mathbf{x}) \approx \mathrm{SDF}_i(\mathbf{x}) \qquad \Phi_{\theta_\Omega}(\mathbf{x} \mid \mathbf{z}_i) \approx \mathrm{SDF}_i(\mathbf{x}) \tag{2.15}$$



(A) Single shape SDF decoder (B) Coded shape SDF decoder

FIGURE 2.12: A SDF decoder network for a single shape, compared to a coded SDF decoder embedding multiple shapes.

2.3.13 Shape Reconstruction and Completion

Reconstruction is the process decoding a latent vector into the corresponding uncompressed data, such as 2D images or 3D meshes. *Shape* reconstruction is for our purposes the reconstruction of 3D meshes.

Shape *completion* aims to infer the unseen parts of a shape from only a partial observation. An example of shape completion could be the inference of the whole surface mesh, given only a subset of the vertices and faces as the input. *Single-view* shape completion and reconstruction is the task of inferring a full 3D shape from only a single perspective or viewpoint and produce a closed 3D model. 3D shape completion is analogous to 2D image in-painting.

Shape completion is often based on inferring or by other means discovering the latent space vector that best describes the observable parts of the shape, followed by reconstructing the missing parts of the shape from this latent space vector.

2.3.14 Pose Estimation and Registration

Given an observation of a known shape from some point of view, *pose estimation* is the task of finding the spatial transform needed to move that shape into the observed pose.

Registration is the problem of finding the spatial transform that aligns two observations. Point cloud registration is often used to align and merge two point cloud scans taken from different viewpoints.

2.3.15 Classification and Segmentation

Classification is the task of determining the *class*[8] of a given observation. *Classifier* networks can be designed for tasks such as object recognition in images.

Segmentation is the task of segmenting a given observation into multiple segments. *Semantic* segmentation aims to divide the observation data according to some set of classes. *Instance* segmentation aims to divide multiple instances (e.g. separate objects) into separate segments. *Panoptic* segmentation is the combination of semantic and instance segmentation, and aims to segment the observation into single classes *and* instances of these classes.

For example in computer vision, a panoptic object segmentation network aims to segment an image such that every instance of an object is uniquely segmented, and that all the segments are appropriately labeled or classified.

[8] Classes are also known as categories or labels. The scope of what a class means depends on what the author aims to achieve.

Chapter 3

Technical Background

Here we briefly cover technical details such as datasets, frameworks and hardware setup.

Section 3.1 covers a few datasets used or discussed.
Section 3.2 presents AI and computation frameworks.
Section 3.3 discloses the hardware used to run our implementation.

3.1 Data- and Object Sets

To train a shape reconstruction and completion model one needs a large dataset of shape priors to learn from. The following three subsections cover some particular ones in further detail.

(A) Some physical objects of the YCB object set

(B) RGB (C) Semantic segmentation (D) Depth map (E) Bounding boxes

FIGURE 3.1: The YCB object. (A) is a real-world image of the objects in the YCB object set (from Calli, Singh, et al. 2015), and (B-E) are synthetic images from the Falling Things dataset (from Tremblay, To, and Birchfield 2018).

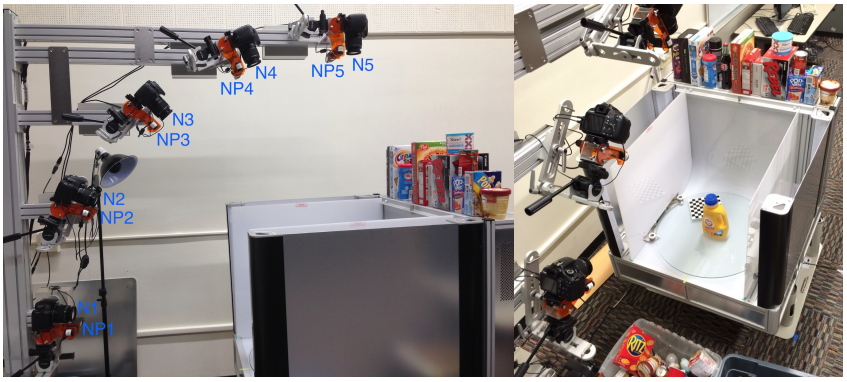
3.1.1 ShapeNet

ShapeNet by Chang et al. (2015) is an effort to establish a large annotated dataset of 3D shapes. The shapes are grouped into different categories, and the shapes within each category are highly aligned. It includes categories such as planes, cars, tables and chairs. These categories

are largely not ones we are interested in for our research purposes. Most of the related 3D completion works described in chapter 4 use the ShapeNet dataset and need to be adapted to other datasets to suit our research goals.

3.1.2 YCB and the BigBIRD Scanner

BigBIRD by Singh et al. (2014) is a 3D scanner at Berkeley, as well as an object dataset addressing the lack of 3D object dataset with corresponding real-world RGB-D images with included pose and camera calibration information. The scanner is constructed of five 3D cameras alongside five high-resolution cameras, situated in a quarter-circular arc over a turntable in a well-lit photo bench. The scanner is pictured in fig. 3.2. The dataset contains for each object reconstructed 3D meshes using both volumetric range image integration (TSDF) and Poisson surface reconstruction, along with 600 registered RGB-D images per object.



(A) 5 Carmine 1.09 (labeled NP1-NP5) and 5 Canon T3s camera (labeled N1-N5) situated in a quarter-circular arc.

(B) Ortery Photobench 260 with a motorized glass turntable.

FIGURE 3.2: The Berkeley BigBIRD 3D scanner. It captures images from 5 polar angles and 120 azimuthal angles equally spaced apart by 3°. (From Singh et al. 2014.)

The YCB Object and Model Set (Calli, Walsman, et al. 2015; Calli, Singh, et al. 2015) is designed to facilitate benchmarking in robotic manipulation. The set consists of both a digital dataset and of a corresponding physical set of objects available for purchase. A selection of these objects are pictured in fig. 3.1a. The digital models have been created using both the Berkeley scanner and the more accurate Google scanner. The Berkeley and Google models are not posed alike however.

The reconstructed meshes in these two datasets vary in quality due to various object properties such as transparency, reflectance and size. As such we handpicked a selection of sound non-degenerate meshes, disclosed in the supplementary table A.2 and rendered in figure A.2. The table includes the object categories we assigned to each object, from here on referred to as their *class*.

3.1.3 Falling Things (FAT)

Falling Things (Tremblay, To, and Birchfield 2018) is a synthetic dataset for object detection and pose estimation. It contains 60 000 annotated images constructed from 21 objects from YCB, vizualised in figure 3.1 (B-E). The object models are combined with complex backgrounds to render photorealistic images. Each image is annotated with 3D object poses, bounding boxes, and per-pixel class segmentation.

3.2 Platforms

3.2.1 PyTorch and CUDA

PyTorch is a machine learning framework for Python by Paszke et al. (2019) at Facebook. It includes facilities to construct neural networks and other kinds of machine learning models. PyTorch provides a tensor math library inspired by Numpy, with a sizeable collection of activation and transfer functions. It constructs a differentiable computation graph automatically during runtime via auto-differentiation. To leverage these it provides many gradient based optimizers. PyTorch can offload computations onto peripheral hardware, such as GPUs[1] via CUDA and TPUs.[2] CUDA let developers to use CUDA-enabled GPUs for general purpose processing, such as machine learning.

3.2.2 PyTorch Lightning and Slurm

PyTorch Lightning (PL) by Falcon (2019) is a module providing something PyTorch does not: a training loop. It aims to reduce the amount of engineering required to implement a self-contained machine learning model, enabling rapid experimentation. It provides tools to distribute and offload the training across across multiple machines, GPUs, and TPUs. PL simplifies common tasks such as early stopping, logging, creation of checkpoints and experiment management. This enabled us to efficiently leverage the SLURM-based[3] compute cluster by Själander et al. (2019).

3.2.3 Intel RealSense

Intel RealSense is a range of depth and tracking products, designed to give machines capable depth perception. They provide the library `librealsense2` for their newer RGB-D cameras, the D4xx series and the SR300, as well as for the T265 tracking camera. Their SDK[4] provides ample calibration data and allows for real-time streaming of color and depth data. Included in the SDK is a small collection of tools, one being the *Intel RealSense Viewer*: a program to view and interact with their cameras in real time. We report in supplementary table A.1 their default post-processing filters.

3.3 System Setup

We trained our neural networks with resources provided by the NTNU IDUN/EPIC computing cluster, equipped with Nvidia Tesla P100 and V100 GPGPUs (Själander et al. 2019).

We evaluated our model on a Intel Core i7-6700 CPU machine with a Nvidia GTX 1080 GPU. The machine runs Ubuntu 18.04 LTS with Nvidia Driver 455.32 on CUDA 11.1.

We used Python 3.7, Pytorch 1.8.1 and Pytorch lightning 1.2.8 on both setups.

[1] GPU: Graphical Processing Unit
[2] TPU: Tensor Processing Unit
[3] SLURM: a job scheduler and workload manager for large compute clusters.
[4] SDK: Software Development Kit

Chapter 4

Related Works

In this chapter we explore related works in computer vision and visual servoing. Computer vision is a scientific field concerned with how computers can gain a high-level understanding of images and shapes. It has a long history with shape analysis, object classification and shape reconstruction. Visual servoing is closely tied, but focuses on real-world robotic application and evaluation. Our research goal is enabling robotic grasping of unknown objects by learning to see and understand the scene from a single-view. We cover here the strengths, approaches and shortcomings of older machine learning approaches, along with the current state-of-the-art.

Section 4.1 explores requirement and affordances in robotic object manipulation.
Section 4.2 cover what object recognition and classification can provide.
Section 4.3 examines historical and the current state of 3D shape reconstruction and shape completion.

4.1 Visual Servoing and Robotic Manipulation

Visual Servoing is a field aiming to control robots with information extracted from a vision sensor. We want our findings to be applicable to robotic manipulation, and perform for this reason a quick investigation into these fields.

Researchers at Berkeley University developed the BigBIRD scanner and the YCB object and model set (Singh et al. 2014; Calli, Walsman, et al. 2015; Calli, Singh, et al. 2015), which enables researchers to easily compare different manipulation approaches.

Within the GentleMAN project, Pedersen, Misimi, and Chaumette (2020) propose a novel approach to transfer knowledge learned in a simulated environment to the real-world. Their approach is based on "tricking" a robot agent into believing it is operating in the synthetic environment by using style transfer techniques on the sensor data.

Yen-Chen et al. (2020) introduce the concept of gripping *affordance*. They use past records of successes and failures in gripping to train a learning model to estimate gripping affordance. This inferred affordance is then used as heuristic to determine on which object and where along the object to place the robot grippers.

From this we have learned we need a method to transform real-world data into a format the model is trained to understand. Furthermore, our method should be controllable by some external heuristic or agent.

4.2 Object Detection and Classification

The ability to tell objects apart from other objects and the background, and to tell *what* exactly an object is, may prove instrumental to single-view shape-completion. For this reason we briefly look into what fields of object detection and classification has to offer.

He et al. (2018) present Mask R-CNN, a framework for efficiently detecting the classes of objects in images while simultaneously generating a high-quality segmentation mask for each object instance. Wu et al. (2019) improve on this work with additional features and reduced the training time, packaged into a PyTorch library called Detectron2. Focusing on point clouds instead of images, Chaton et al. (2020) provide Torch-Points3D, a library for performing object detection, classification, segmentation and registration on point clouds.

From this we have learned that there is an abundance of ready-made solutions for image classification and segmentation. Instance segmentation on images enables filtering RGB-D sensor data such that *single-object* shape completion is possible.

4.3 3D Shape Completion

There have been multiple studies on single-view 3D shape completion. Consequently a plethora different methods have been proposed, each with their own strengths and weaknesses. Standing on the shoulders of such abundant work we seek to design our own method based in the requirements of our targeted robotic environment.

There have been many shape completion approaches that use 2D images to predict 3D voxel models, inspired by how well convolutional techniques and concepts from 2D computer vision translate over to 3D voxels (Yan et al. 2017; Girdhar et al. 2016; Zhu et al. 2017; Wu et al. 2017, 2018; Tulsiani et al. 2017; Yang et al. 2018). These methods have difficulties representing finer surface details, due to how poorly voxel models scale with regards to memory and computational resources. The resulting reconstructions are often not pleasing to look at due to their either cubified look or their poor ability to infer accurate surface normals. Some turned to octrees as a means of increasing the effective resolution near high-detail areas (Häne, Tulsiani, and Malik 2017; Tatarchenko, Dosovitskiy, and Brox 2017; P.-S. Wang et al. 2019), leaving areas of lower complexity less densely subdivided. Some approaches infer voxel grids of signed distances (Dai, Qi, and Nießner 2017; Zeng et al. 2017; Stutz and Geiger 2018). The introduction of the signed distance results in a smoother output with sound normals, but are still subject to aliasing due to the discrete grid.

Methods inferring 3D meshes directly from images often produce non-watertight and degenerate meshes, attributed to their high irregularity (Pontes et al. 2017; Hanocka et al. 2019). Instead of directly inferring meshes, Zou et al. (2017) and Niu, Li, and Xu (2018) perform shape completion into a collection of predetermined shapes primitives. Building on this idea, AtlasNet by Groueix et al. (2018) performs shape completion by producing a collection of parametric surface elements combined with constructive solid geometry. Meanwhile, Wang et al. (2018) introduce Pixel2Mesh, a graph-based network reconstructing 3D manifold meshes. 3DN by W. Wang et al. (2019) introduce the idea of *deforming* the vertices of a source mesh to conform to single-view observations instead of inferring them from the ground up. Wen et al. (2019) build further on this idea in a multi-view environment, proposing Pixel2Mesh++. GEOMetrics by Smith et al. (2019) exploits geometric structures in graph-encoded objects. Liu et al. (2019) likewise took the mesh deformation approach, but with a novel twist: they introduced the concept of a *differentiable renderer*. It models the whole rendering pipeline as a differentiable probabilistic process, enabling backpropagation. As such it can extract textures and simple lighting conditions from images.

Approaches based on surface point clouds achieved quite early on impressive 3D shape inference from RGB-D images, as they are not impeded by having to infer sound mesh topologies (Fan, Su, and Guibas 2016; Han et al. 2017). These methods are limited to producing a fixed number

of points however, tied to the number of output neurons. They also spend a great amount of computational resources to ensure the network are invariant to the ordering of the points consumed and produced. PointNet (C. R. Qi, Su, et al. 2017; C. R. Qi, Yi, et al. 2017) address this problem by using max-pool operations to extract global shape features. It has since become a widely used encoder and discriminator for larger generative point cloud networks. Due to the fixed output size, most point cloud networks do not have the ability to selectively generate more points in areas of higher complexity, making them hard to disambiguate. This problem is further amplified by how point clouds often are subject to error-prone post-processing when being converted into meshes (Kazhdan and Hoppe 2013). RL-GAN-Net, a very capable point cloud completion model by Sarmad, Lee, and Kim (2019), address this issue by combining the output of an auto-encoder and a GAN with a special reinforcement learning agent, making it able to (selectively) infer high quality point clouds in real-time.

4.3.1 Implicit Representation Learning

All of the aforementioned methods are based on explicit 3D shape representations. These methods either suffer from a limited spatial resolution, are prone to erroneous post-processing, produce degenerate meshes, or are limited to a fixed mesh topology. Recently a popular alternative to these explicit representations have emerged: *implicit surfaces* approximated with neural fields.

IM-NET (Chen and Zhang 2019) and Occupancy Networks (Mescheder et al. 2019) were the first to explore learning shapes represented as implicit continuous functions. They approximate a binary *occupancy* function of 3D shapes for a given 3D query coordinate and a feature vector. Occupancy Networks consists of 5 ResNet[1] blocks, while IM-NET is a MLP 6-layers deep. Occupancy Networks further use binary search to help disambiguate areas of higher complexity, indicating that the binary occupancy field is not ideal for shape reconstruction. Implicit occupancy functions are further explored by Littwin and Wolf (2019), who adapt the learning of implicit functions to the field of meta-learning[2] by constructing a hypernetwork encoder that infer decoder weights from single-view images.

DeepSDF by Park et al. (2019) regress signed distance fields instead of the binary occupancy field, using a 8 layer MLP with a skip-connection that concatenates the network input to layer 4. This architecture set a trend for following works. Their method predicts the distance to the implicit isosurface, removing the need for an iterative disambiguation procedure like in Occupancy Networks. DeepSDF additionally pioneered the use of the auto-decoder training framework for representation learning.

Multiple works continue exploring deep implicit surfaces and signed distance fields following DeepSDF. DISN (Xu et al. 2019) address the poor shape completion time of DeepSDF by replacing the auto-decoder architecture with a convolutional neural network using RGB images as its input. Their CNN encoder predicts the latent space code of a shape, which is then feed into a SDF decoder network much like DeepSDF. DISN adds additional networks to perform pose estimation and infer further *local* features along the surface of the global shape. Their method requires supervision with a lot of RGB images annotated with corresponding 3D models and poses. As such it proves difficult to adapt to the real-world. Chabra et al. (2020) keep the auto-decoder architecture of DeepSDF and instead focus their attention on local shape completion, applying their network DeepLS on subdivided spatial chunks of point clouds at a time. This makes their model unsuited for single-view shape completion, but results in a surface reconstruction method outperforming TSDF reconstruction. As such it can alleviate the post-processing issue inherent to point cloud based shape completion methods. MetaSDF (Sitzmann, Chan, et al. 2020) applies a meta-learning approach to implicit functions, resulting

[1] ResNet blocks are elaborate constructions applying multiple convolutions and activations before *adding* the activations of the previous layer via a skip connection.

[2] Meta-learning improves generalization in general. Commonly, meta-learning is the application of hypernetworks trained to predict the weights of a separate network. Used to either predict the initial network weights for few-shot optimization, or to fully predict the weights of a decoder network from a latent vector.

in an order of a magnitude faster time to perform the shape completion while making weaker assumptions of the underlying latent space of shapes. They propose using a few-shot approach from a meta-learned initialization, averaging the weights learned from separate short-lived diverging runs. This comes at the cost of being very memory intensive to train. Kleineberg, Fey, and Weichert (2020) combine continuous implicit shape learning with generative adversarial modeling (GAN), applying implicit surfaces to shape synthesis. PIFu (Saito et al. 2019) performs single-view completion by re-projecting 3D points into a pixel-aligned feature representation, learning highly detailed implicit models of humans. PIFu is developed further by Saito et al. (2020) and He et al. (2020).

NeRF (Mildenhall et al. 2020) applies the network architecture of DeepSDF to radiance fields[3] to synthesize novel views of complex scenes. Instead of learning SDF values they regress RGB colors along with a *density*. They propose using *positional encoding* which improves the ability of ReLU-based MLPs to learn higher frequency patterns and relations on low dimensional inputs. Tancik et al. (2020) analyze positional encoding further in greater detail, generalizing it and discovering that there is a balance required to avoid overfitting. Tancik et al. (2021) further explore the application of few-shot meta-learning to neural radiance fields, much like that of MetaSDF.

Sitzmann, Martel, et al. (2020) use neural fields to solve boundary value problems. They call attention to how the gradients of the output of the network with regards to the input can be supervised during training. To make use of this they propose a novel alternative to ReLU-based multilayer perceptrons: SIRENs. These are coordinate-based representation networks whose nonlinearities are periodic, based around sinusoidal functions. SIRENs can accurately learn and represent the first and second order derivatives of the target signal, and its inductive biases can be tuned with a hyperparameter. They show that only supervising certain boundary conditions, such as gradients or the divergence of gradients in limited areas in a 3D field, is sufficient to converge on a full field. They demonstrate this for signed distance fields, generalized as a Eikonal boundary value problem.[4]

4.4 Fall Project by the Author

We wrote a specialization thesis (Sundt 2020) over the leading fall semester to establish the feasibility of this master thesis, and it features some promising preliminary findings: DeepSDF-like network architectures have the capacity to learn not only 3D shapes in one canonical pose, but shapes over a continuous space of orientations. Furthermore they proved able to embed more than one category of objects at a time. We adapted and reprised parts of the fall project here for the sake of making this master thesis more self-contained.

[3] Radiance fields map 3D spatial coordinates along with two sphere coordinates indicating viewing direction to linear RGB colors: $\mathbb{R}^3 \times \mathbb{R}^2 \to \mathbb{R}^3$. It is a simplification of the electromagnetic field where qualities humans can not perceive are omitted, such as details in wavelengths and polarization. NeRF additionally regress a density σ such that their network effectively models a ray tracer for a camera with a fixed roll.

[4] The eikonal equation is a partial differential equation $\|\nabla_x u(\mathbf{x})\|_2 = \frac{1}{f(\mathbf{x})}$ where ∇ is the vector differential operator, $\|\cdot\|_2$ is the Euclidean norm and $u(\mathbf{x}) = 0$ along a well-behaved boundary. Fixing $f = 1$ results in a signed distance function.

Chapter 5

Methodology

We present in this chapter our single-view shape completion approach and methodology. Our approach is based on our investigations in chapter 4, adapted to fit our research goals within the GentleMAN project.

Section 5.1 introduces the task and outlines our overall approach.
Section 5.2 describes how data for network training and evaluation is prepared.
Section 5.3 contains our proposed network architecture.
Section 5.4 presents how we train our networks.
Section 5.5 explains how and why shape completion works.
Section 5.6 covers the nonlinearities, loss functions and parameters we experiment with.
Section 5.7 defines metrics used in our evaluation.

5.1 Overall Approach and Motivation

Our goal is single-view 3D shape completion of objects for use in robotic manipulation. Given a depth image of an object, our aim is to reconstruct a full 3D mesh with correct pose that captures the overall structure of the object including the parts hidden from view. Our proposed processing pipeline to achieve this is rather involved. It deals with: processing depth images, classifying and segmenting the objects in the images, extracting signed distance points, and completing the shape of the object from learned shape priors.

We base our approach on the new and emerging field of implicit representation learning via neural fields, whose popularity was primarily triggered by Park et al. (2019) and Mildenhall et al. (2020). Neural fields (sec. 2.3.10) regress a target signal continuously over the input domain, in our case implicit surfaces in 3D Euclidean space (sec. 2.1.5). This powerful data-driven paradigm enables representation of more complex shapes without discretization errors and uses significantly less memory than previous state-of-the-art methods based on explicit shape representations. Neural fields enables arbitrary 3D shape reconstruction resolution with any mesh topology, of any genera, while guaranteeing the resulting meshes to be non-degenerate. Our field of interest is the *signed distance field* (SDF): a continuous 3D field whose 0-level set describe the surface of an object.

The path to shape completion from learned shape priors starts with shape representation learning and reconstruction. We train neural networks to predict the SDF value when given a spatial query coordinate. One can reconstruct 3D meshes from these neural fields using the marching cubes algorithm. We aim to make our networks learn an extensive latent space of shape priors. The goal is to embed the shapes in the latent space in such a way that one can *search* it for shapes conforming to single-view observations via gradient decent.

To construct this space of implicit surfaces we model a latent space decoder network Φ and have it approximate their signed distance functions:

$$\Phi_\theta(\mathbf{x} \mid \mathbf{z}_i) \approx \mathrm{SDF}_i(\mathbf{x}) \;:\; i \in \Omega,\ \mathbf{x} \in \mathbb{R}^3 \tag{5.1}$$

where Φ is a neural network with θ weights, $\mathbf{x}$ is a spatial query coordinate, Ω is a collection of shape priors, $\mathbf{z}_i$ is a vector into the latent space of shapes pointing at shape i, and $\mathrm{SDF}_i(\cdot)$ is the ground truth signed distance function for shape i.

We train the network as an auto-decoder, optimizing both θ and $\{\mathbf{z}_i\}_{i\in\Omega}$ on fully formed SDF fields. To shape complete we fix θ and optimize a new $\hat{\mathbf{z}}$ on partial single-view data. For this to work we seek to learn meaningful relations between the priors shapes, such that shapes whose near side conform to a single-view observation are also likely to have meaningful far sides.

Accurate surface normals can be computed as the spatial derivative $\nabla_{\mathbf{x}}\Phi$, which we can calculate analytically due to the differentiable nature of neural networks. As such it is possible to supervise this derivative during training by backpropagating its computation graph. We experiment with supervising Φ with a target SDF gradient in addition to the base SDF signal. This demonstrates how powerful of a paradigm neural fields really are.

Related works train and evaluate separate networks for each object category, for example one for chairs, one for airplanes, and one for tables. We by contrast embed a wide variety of 3D shapes into a *single* shared latent space. To this end we explore the construction of these networks in addition to how to best supervise their training. We experiment (sec. 5.6) with both positional encoding and with periodic nonlinearities.

The objects we perform shape completion on do not only come in various shapes and sizes, but also in various orientations. Related works perform pose estimation separately from shape reconstruction, but we experiment with expanding the latent space of shapes to additionally encode their pose. This has the benefit of both regularizing the latent space, and enables searching for conforming shapes of any orientation. As such it bypasses the need to train on separate datasets of single-view observations such as FAT or ShapeNet renderings, which generalize poorly to new scenes with different backgrounds.

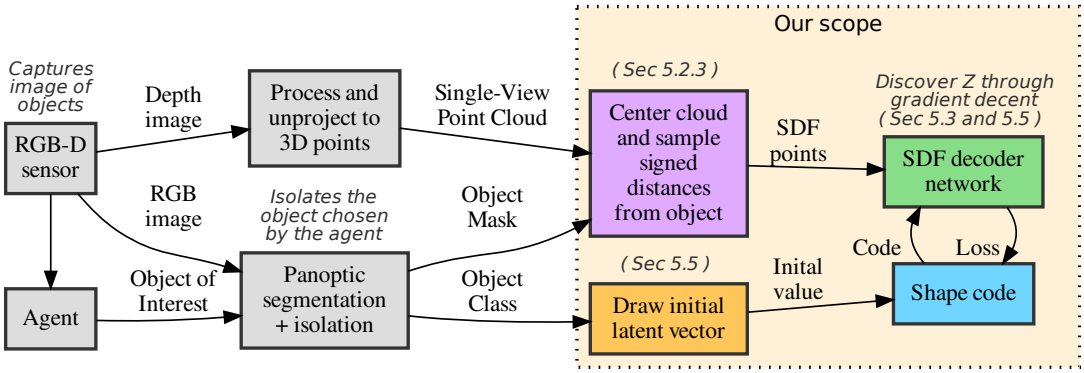


FIGURE 5.1: Our envisioned real-world single-view 3D shape completion pipeline, based on searching through a latent space for the shape that best conforms to the single-view observation data. This graph illustrates the flow of data from a RGB-D camera to the iterative optimization of a shape code (blue), which we use to reconstruct the full shape at the end. We limit our focus to shape completion (orange cluster, dotted border), and assume accurate class and segmentation data of single objects. We need the segmentation mask to extract signed distances from the single-view data (purple), to supervise the decoder network (green). We assume an abstract external "agent" isolates a single object segment for us to shape complete.

We limit our focus in this thesis to shape completion, and assume access to accurately classified and segmented point clouds or RGB-D data of single objects. This information is not trivially accessible in-the-wild however, and must be inferred with either a panoptic segmentation network, or with hand-crafted domain specific algorithms.[1] As such we evaluate our work on the segmented and registered RGB-D images in the YCB dataset. To extract point cloud we implement rudimentary RGB-D processing for this data ourself.[2] We illustrate how our model fits in a real-world inference pipeline in figure 5.1.

5.2 Data Preparation

We train and evaluate our learning model with objects from YCB dataset. It provides 600 RGB-D images for each object, and up to three 3D mesh models. We cover here how we process and extract signed distances (SDF) from this data to feed our learning model. We sample full-view SDF point clouds from the 3D models to train our networks. We perform shape completion on single-view SDF clouds, extracted from either the real-world RGB-D images, or from synthetic scans created by rendering the mesh models.

5.2.1 3D Model Pre-Processing and Normalization

The YCB dataset contains many models that are either largely distorted or corrupt. Some of the models fall outside of our intended scope, either being too thin or consisting of multiple smaller objects. As such we only use a subset of the YCB models, enumerated in supplementary table A.2, to avoid unreliable data and to improve knowledge discovery.

YCB provides up to three meshes per object: two from the BigBIRD scanner and one from the Google scanner. BigBIRD produces low-quality meshes reconstructed with volumetric range image integration (TSDF) and with Poisson reconstruction. The former is more detailed but open, while the latter has less detail but is watertight. The Google scanner provides watertight meshes of much higher quality, but these have not been aligned with the BigBIRD RGB-D images. The Google models do at times not even match the YCB objects at all. These two issues do not pose any problems for shape learning, but can pose a problem for evaluation. We primarily train with the more accurate and well formed Google scanner meshes, falling back to using the BigBIRD Poisson meshes if a Google mesh is not provided.

We scale and translate the 3D meshes of YCB to a common frame of reference. This is to simplify our embedded latent space of shape priors, promoting feature extraction and knowledge discovery. ReLU-based MLPs networks perform better when their target signal is limited to a fixed range (Sitzmann, Martel, et al. 2020). We follow Chen and Zhang (2019) and translate our training shapes such that their axis-aligned bounding box centroid is zero, and scale them such that their most distant vertex touch the unit sphere. This fits the shapes within what we from here on call the *reconstruction volume*: a box with extents $[-1,-1,-1]^T$ to $[1,1,1]^T$. Formally, the vertices V of a mesh are normalized as:

$$\begin{aligned} V_{\text{normalized}} &= \left\{ \frac{\mathbf{v} - \mathbf{v}_{\text{centroid}}}{c_{\text{scale}}} \right\}_{\mathbf{v} \in V} \\ \mathbf{v}_{\text{centroid}} &= \frac{1}{2} \begin{bmatrix} \max_{\mathbf{v} \in V}(x_{\mathbf{v}}) + \min_{\mathbf{v} \in V}(x_{\mathbf{v}}) \\ \max_{\mathbf{v} \in V}(y_{\mathbf{v}}) + \min_{\mathbf{v} \in V}(y_{\mathbf{v}}) \\ \max_{\mathbf{v} \in V}(z_{\mathbf{v}}) + \min_{\mathbf{v} \in V}(z_{\mathbf{v}}) \end{bmatrix} \\ c_{\text{scale}} &= \max_{\mathbf{v} \in V} \|\mathbf{v} - \mathbf{v}_{\text{centroid}}\|_2 \end{aligned} \tag{5.2}$$

where $x_{\mathbf{v}}, y_{\mathbf{v}}, z_{\mathbf{v}}$ are the x y and z coordinates of $\mathbf{v} \in \mathbb{R}^3$, and $\| \cdot \|_2$ is the 2-norm magnitude.

[1] We expect of-the-shelf convolutional solutions such as Mask R-CNN (He et al. 2018; Wu et al. 2019) or Torch-Points3D (Chaton et al. 2020) to perform adequately.

[2] In our targeted robotic lab environment we instead process live depth sensor data from more accurate depth cameras using the Intel RealSense framework, likely to yield far fewer artifacts.

5.2.2 Sampling Full-View SDF Clouds

To train our networks we sample SDF points from 3D models with a strategy based on the Monte Carlo technique proposed by Park et al. (2019). They sample a set of $(\mathbf{x}, y)$ pairs from 3D meshes, where $y = \text{SDF}(\mathbf{x})$. They sample points around the mesh using two strategies:

Uniform points are sampled uniformly within a sphere enclosing the object.
Near-surface points are sampled along the *visible* surface of the object.

These two strategies combined result in a distribution of points that bias surface details near the zero-crossing, while still ensuring the rest of the signed distance field is well behaved. A full-view point cloud of the *visible* surface is prepared as follows:

- Fit the 3D mesh model inside the unit sphere. (sec. 5.2.1)
- Render the surface from 100 virtual cameras into depth and normal buffers.
- Unproject the 100 scans into point clouds using their inverse MVP transforms.
- Combine the 100 registered clouds into a single surface point cloud.

To compute a SDF value (y) for a given $\mathbf{x}$ coordinate, we:

- Find the 11 nearest surface points to $\mathbf{x}$ by searching a KD-Tree.
- Set $|y|$ to the distance of the nearest neighbor.
- Determine the sign of y by a majority vote, using the dot product on the 11 nearest normal vectors and their direction to x $\mathbf{x}$.

We expand their procedure to also compute the spatial SDF gradients, which we denote $\nabla_{\mathbf{x}} y$. If the absolute SDF value is below a threshold value[3], then we use the nearest normal vector in the surface point cloud to determine the direction of the SDF gradient. This is to prevent noisy near-surface SDF gradients. If the point is outside of the this threshold then we instead use the direction to the nearest point in the surface point cloud to determine the SDF gradient direction. The magnitude of the SDF gradient is always 1.

Park et al. (2019) only sample *uniform* points within the unit sphere. We instead sample within a sphere with radius $\sqrt{3}$, to ensure the whole reconstruction volume is valid for marching cubes to traverse regardless how we rotate the shape.

5.2.3 Sampling Single-View SDF Clouds

For shape completion we search via optimization for the shape latent vector that best conforms to the single-view observation data. How do we supervise this optimization? We know two things about the underlying shape from a single-view depth observation: where the visible surface is and where there is free-space.

We propose a novel strategy inspired by Chabra et al. (2020) for sampling SDF clouds from single-view depth scans, segmented into *hits* and *misses*. Scan rays hitting the surface of the object are counted as hits, the other rays are classified as misses. We unproject (sec. 2.2.3) these depth observations into point clouds, keeping track of which points are hits, which are misses and where the camera is in relation. From here on we refer to these as hit+miss clouds.

In figure 5.2 we illustrate an example scene. We first center and scale the single-view hit+miss cloud such that the hit points fit within the unit sphere. *Near-surface* SDF samples can then be sampled directly from the hit points by perturbing them with $\mathcal{N}(0, 0.0025^2)$ along their estimated normal vector. The SDF value becomes the distance displaced. *Uniform* samples can be generated by sampling random points along the scan rays, or put differently: in the visible free-space. The SDF value is set by the distance to the nearest hit point.

[3]The threshold value we use is 3 times the 3D 2-norm standard deviation of the distribution ($\mathcal{N}(0, \sigma^2)$) used to draw the near-surface samples: $3\sqrt{3\sigma^2}$. This captures 99.8% of the *near-surface* samples in addition to stray *uniform* samples near the surface boundary.

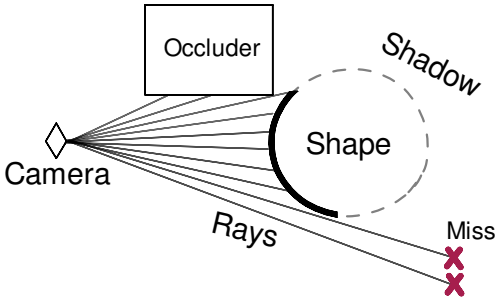


FIGURE 5.2: A diagram of scan rays cast from a camera into a scene with our shape of interest and an occluding object. Scan rays hitting the visible surface of the shape (bold) are counted as *hits*. Rays hitting the either the background or other objects are counted as *misses*. We sample uniform SDF samples within the volume covered by scan rays. Near-surface SDF samples are generated along the bold surface.

We ensure the visible surface is sampled uniformly, as to not inadvertently bias the areas closer to the camera. We set the probability of selecting a specific hit point proportional to the distance to its nearest hit point neighbor, squared. With this strategy the chance of sampling from a given hit point is proportional to the surface area it "covers".

We can not sample along the scan rays uniformly either: they vary in length, and the volumetric cross section "covered" by each ray increases the further they trace into the scene. The probability of picking a particular scan ray is set proportional to its length. We then model the scan ray as a pyramid and use inverse transform sampling to sample a point along it. The probability distribution function is $f(x) = cx^2 \sin(\phi)$, where c ensures $\int_0^1 f dx = 1$ and ϕ is the pyramid slope. We compute the cumulative probability distribution, solve for c, and find the inverse of the cumulative distribution function. Almost all the terms cancel out, assuming $\phi > 0$. This leaves us with the sampler: $F^{-1}(x) = \sqrt[3]{x}$ for $x \in \mathcal{U}(0, 1)$. We reject uniform samples generated outside of a sphere fitted around the hit points with some overhead.

5.2.4 Processing RGB-D Images

We need real-world single-view sensor data to evaluate our shape completion approach. For this purpose we extract hit+miss point clouds (sec. 5.2.3) from the segmented RGB-D depth images provided by the YCB dataset.

Depth sensors tend to yield imprecise measurements near sharp edges. We filter as recommended by Singh et al. (2014) the imprecise depth measurements using *depth discontinuity filtering*. We compute a mask of the all depth pixels whose gradients (computed with a 7×7 kernel) exceed a threshold. This mask is then further dilated[4] and used to filter the unwanted depth pixels.

Using the calibration data provided by YCB we correct the depth maps for lens distortion then unproject the pixels into point clouds in camera space. We then segment the points into hits and misses using the provided RGB segmentation masks. As the mask are taken from the color camera perspective we first perform depth-to-color registration using the provided calibration data. Then we transform the point clouds to the scanner turntable coordinate system, shared with the BigBIRD meshes. Finally we filter hits outside of a cylinder enclosing the turntable, along with hits at or below the ground plane. Specifics are covered in chapter 2.2.3.

[4]Dilation is a morphological operation. Dilation expands the shapes in a binary image using some kernel, in our case a 7×7 square.

5.3 Learning Architecture

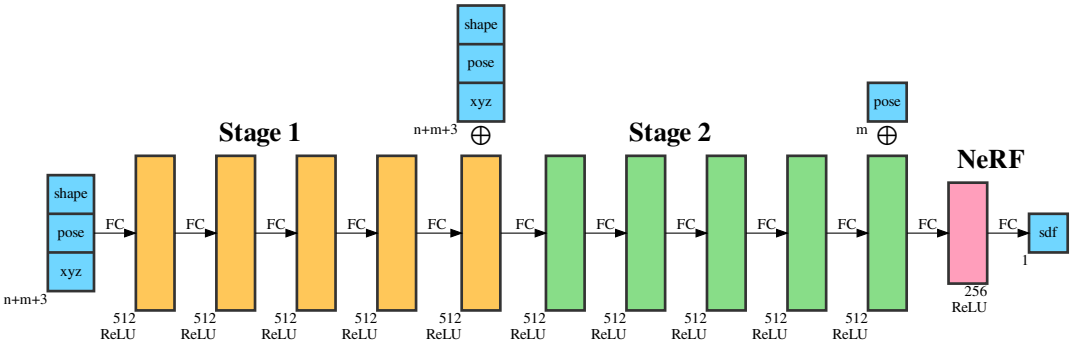


FIGURE 5.3: The structure of our neural signed distance field decoder, inspired by DeepSDF and NeRF. It models a probabilistic decoder over a space of shapes. This variant is 512 neurons wide, use ReLU nonlinearities, and is two stages deep with a final NeRF stage. Our latent vectors consists of a shape and pose component. Skip connections concatenate the network input onto the activations of preceding stages. FC is short for Fully Connected.

We train our artificial neural networks with full-view 3D models, but are limited to partial single-view observations at test time for shape completion. For this reason we went ahead with an auto-decoder architecture, as auto-encoders expect the test input to be similar to the training input. This recently proven framework gives us additional freedom to shape the layout of our latent space and tweak the shape completion process.

We model our network Φ with learnable parameters θ and $\{\mathbf{z}_i\}_{i \in \Omega}$ as a probabilistic decoder (sec. 2.3.12) such that $\Phi_\theta(\mathbf{x} \mid \mathbf{z}_i)$ approximates the signed distance function $\text{SDF}_i(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^3$ for all shapes $i \in \Omega$. The goal is for Φ to embed the shape priors in Ω in such a way that it can be used to inform the shape completion process.

The objects in Ω are normalized (sec. 5.2.1) to fit within the *reconstruction volume* of Φ: a box with extents $[-1, -1, -1]^T$ to $[1, 1, 1]^T$. We refer to the pose of the shapes in Ω as the *canonical pose*.

It is customary in deep learning to include skip connections to enable deeper architectures to make better use of low-level relations between the input and output throughout the whole network. Occupancy Networks (Mescheder et al. 2019) use a residual architecture whose skip connections are based on addition, whereas DeepSDF (Park et al. 2019) use a densely connected architecture with concatenation-based skip connections. We base our work on the latter due to its lower size and complexity yet comparable performance.

We construct Φ as a deep multilayer perceptron (MLP) with nonlinearities subject to experimentation, and concatenate the input (query coordinates and latent code) onto the activation of every fifth layer. These 5-layer blocks are from here on referred to as *stages*. We split the latent vector $\mathbf{z}$ into a *shape* and a *pose* component, to enable pose estimation. As in NeRF (Mildenhall et al. 2020) we additionally concatenate the orientation code alone ($\mathbf{z}_{\text{pose}}$, see sec. 5.4.1) to the last stage and reduce it with two fully-connected layers: the first layer halves the width and final layer reduces the width to one, regressing the final prediction. The final skip connection enables the network to scale its shapes without scaling the magnitude of their SDF gradients. A two-stage ReLU variant of our network architecture 512 neurons wide is illustrated in fig. 5.3.

We cover how we train these networks and the nonlinearities we use in the following sections.

5.4 Training

We train our SDF decoders Φ as auto-decoders (sec. 2.3.11): we maintain a database of latent vectors $\mathbf{z}_i$ for each shape $i \in \Omega$ and optimize these vectors along with the hidden network parameters θ. We train to find the latent vectors and network weights that minimizes the loss computed across the training data:

$$\underset{\theta,\ \{\mathbf{z}_i\}_{i\in\Omega}}{\arg\min} \sum_{i\in\Omega} \sum_{(\mathbf{x}_j, y_j)\in X_i} \mathcal{L}\left(\Phi_\theta(\mathbf{x}_j \mid \mathbf{z}_i),\ y_j\right) \tag{5.3}$$

where X_i is the set of ground truth training examples for shape $i \in \Omega$, and $\mathcal{L}$ is an objective function subject to experimentation.

The training set consists of n pairs of 3D query coordinates $\mathbf{x}$, corresponding ground truth SDF values y, and their spatial gradients $\nabla_{\mathbf{x}} y$:

$$X_i := \{(\mathbf{x}_j, y_j, \nabla_{\mathbf{x}_j} y_j)\}_{j=0}^{n} \ :\ y_j = \mathrm{SDF}_i(\mathbf{x}_j) \tag{5.4}$$

For the sake of brevity we express $\mathcal{L}$ as $\mathcal{L}(y, y_{\mathrm{GT}})$ where y is the predicted SDF value and y_{GT} is the target ground truth. $\mathcal{L}$ also have access to the $\nabla_{\mathbf{x}} y$ and $\nabla_{\mathbf{x}} y_{\mathrm{GT}}$ values, which we express as an operation instead as pre-computed values provided as input.

5.4.1 Augmenting for Pose Estimation

Here we cover our primary contribution, which is to expand the latent space to encode the orientation of a shape in addition to the shape itself. This serves to enable shape completion on single-view observation data that does not conform to the canonical pose. Making the networks learn and perform the pose transformation internally serves to make pose estimation during shape completion draw more influence from the knowledge embedded in the networks

We decompose the latent space vector $\mathbf{z}$ into $(\mathbf{z}_{\mathrm{shape}}, \mathbf{z}_{\mathrm{pose}})$. $\mathbf{z}_{\mathrm{pose}}$ is further composed of a translation $\mathbf{t} \in \mathbb{R}^3$, a scale $s \in \mathbb{R}$ and a 6D rotation $\mathbf{b}_{\mathbf{x}}, \mathbf{b}_{\mathbf{y}} \in \mathbb{R}^3$ such that $\mathbf{z}_{\mathrm{pose}} = (\mathbf{t}, s, \mathbf{b}_{\mathbf{x}}, \mathbf{b}_{\mathbf{y}})$.

During training we augment each example j drawn from X_i (eq. 5.4) with a random spatial transformation based primarily on the normal distribution $\mathcal{N}$, and fix $\mathbf{z}_{\mathrm{pose}}$ to match the augmentation. This forces the network to learn their relation, decomposing the shapes from their orientation. We sample $\mathbf{t}$ from $\mathcal{N}(\mathbf{0}, 0.5^2 I_3)$, sample s from $\mathcal{N}(1, 0.4^2)$, and sample $\mathbf{b}_{\mathbf{x}}, \mathbf{b}_{\mathbf{y}}$ from a uniform distribution on the rotation group $SO(3)$. We the augment the training example j by:

$$\begin{aligned} \mathbf{x}'_j &= sR\mathbf{x}_j + \mathbf{t} \\ y'_j &= s y_j \\ \nabla_{\mathbf{x}_j} y'_j &= R\ \nabla_{\mathbf{x}_j} y_j \end{aligned} \tag{5.5}$$

where $'$ marks the new augmented values, $\mathbf{x}_j$ is the query coordinate, y_j is its corresponding SDF value, R is the rotation matrix corresponding to $(\mathbf{b}_{\mathbf{x}}, \mathbf{b}_{\mathbf{y}})$ as defined in equation 2.10, and ∇ is the vector differential operator.

This formulation results in the canonical pose becoming $[0, 0, 0, 1, 1, 0, 0, 0, 1, 0]^T$, or put differently: the identity $\mathbf{z}_{\mathrm{pose}}$ transform.

We chose the 6D representation of rotation due to it being continuous (Zhou et al. 2019), it being easy to normalize between optimization passes, and it making the orientation transformation into a linear relation to regress. Linear relations are perfect for ReLU-based architectures to learn.

5.4.2 Shaping the Latent Space of Shapes

In this section we constrain and regularize the latent space with various techniques. This is to incentivize the networks to draw meaningful connections between the different shapes, and to make it transition between shapes in a uniform fashion, enabling single-view shape completion via search.

We assume, as Chabra et al. (2020) and Park et al. (2019) did, the prior distribution over $p(\mathbf{z}_{\text{shape}})$ to be a zero-mean multivariate-Gaussian with a isotropic variance $\sigma^2 I$:

$$p(\mathbf{z}_{\text{shape}}) \sim \mathcal{N}(\mathbf{0}_n, \sigma^2 I_n) = \mathcal{N}\left(\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}\sigma^2 & 0 & \cdots & 0\\0 & \sigma^2 & \cdots & 0\\\vdots & \vdots & \ddots & 0\\0 & 0 & 0 & \sigma^2\end{bmatrix}\right) \tag{5.6}$$

where $\mathcal{N}$ is the normal distribution, the vector $\mathbf{0}_n$ is of size n, and the identity matrix I_n is of size $n \times n$. We want to constrain the shape feature standard deviance σ to emerge as small as possible. To this end we use their cost function $\mathcal{L}_{\text{codereg}}$ to constrain the whole auto-decoder database of latent vectors $\{\mathbf{z}_i\}_{i\in\Omega}$:

$$\mathcal{L}_{\text{codereg}} = \frac{1}{|\Omega|}\sum_{i\in\Omega} \|\mathbf{z}_i^{\text{shape}}\|_2 \tag{5.7}$$

This is cost function benefits shape completion for various reasons: (1) It concentrates the embedded shapes near the latent space origin. (2) It helps form a smooth manifold between the origin and each shape, as the latent vectors are punished regardless of whether they are a part of the current training batch or not. This in turn forces the networks to form an opposing "outward" attraction towards the correct location in latent space that counterbalances $\mathcal{L}_{\text{codereg}}$. (3) It punishes latent features with no significant impact on the reconstructed shape, making room for the networks to efficiently allocate each shape feature in a meaningful fashion. (1) and (3) boosts knowledge discovery, while (2) smooths the latent space and makes it more uniform.

The $\mathcal{L}_{\text{codereg}}$ cost must be balanced with a hyperparameter coefficient λ_1 to not overwhelm the SDF reconstruction loss. We additionally schedule it to start at 0% and work its way up to 100% linearly during the first 150 training epochs. This is to not immediately squash down all the codes to $\mathbf{0}$ before the network has even had the chance to learn any meaningful shapes.

At the start of training we initialize the latent vectors with rather small random values sampled near zero. This is to prevent latent codes for similar shapes from diverging. We draw each latent space shape vector from $\mathcal{N}(\mathbf{0}_n, 0.001^2 I_n)$ where n is the width of $\mathbf{z}_{\text{shape}}$, resulting in an average initial 2-norm magnitude of $0.001\sqrt{n}$ for each shape code.

We use the *dropout* regularization technique (sec. 2.3.4.2) to further smooth the latent space. Dropout trades reconstruction quality for a more uniform latent space, and must for this reason be used sparingly. We do not use data augmentation techniques such as noise injection, as it is shown to be detrimental to coordinate-based representation networks such as ours that need to learn high-frequency relations and patterns on low dimensional inputs (Tancik et al. 2020; Mildenhall et al. 2020).

It is common to have fewer dimensions in the latent space than the number of items in the training set, which in our case is $|\Omega| = 83$ objects. This is to create an information bottleneck forcing the network to compress their representation using their structural similarities. Otherwise the network could learn an orthogonal set of codes for each object, or put differently: a one-hot encoding. We experiment with a smaller number of dimensions in our latent space than most other related works have. This also serves to make our latent space even more compact, such that shape completion via latent vector optimization becomes less likely to "fall off the manifold" as Hao et al. (2020) put it.

5.4.3 Training Order

Related works use a simple training scheme where they train on *one* shape at a time throughout an epoch, going through the dataset in a serial fashion. We instead draw batches from *multiple* shapes chosen at random. This is to prevent the latent vectors optimized earlier in an epoch from "drifting" by the end of the epoch, becoming "outdated" with regards to the network weights.

We split the $|X_i| = 250\ 000$ SDF samples for each shape $i \in \Omega$ (eq. 5.4) into a *train* and *validation* part, with 70% and 30% of the samples respectably. Our novel contribution is to further divide these into *sub-batches* of 12500 samples. We construct an index listing all the sub-batches for all shapes along with which shape the sub-batch came from, and shuffle it at the start of every training epoch. This randomized index is then used to construct larger batches consisting of n sub-batches, with n depending on the amount of memory available. The total batch size thus equal $12500n$. Our method enables us train on up to n different objects per batch along with the correct latent vectors per training example.

5.5 Shape Completion Method

At shape completion time, we apply the network in reverse. We fix the hidden network parameters θ and then search for the latent vector $\hat{\mathbf{z}}$ that best conforms to the single-view observations data. We optimize $\hat{\mathbf{z}}$ with stochastic gradient decent to minimize a loss function $\mathcal{L}$:

$$\underset{\hat{\mathbf{z}}}{\operatorname{argmin}} \sum_{(\mathbf{x},y)\in X} \mathcal{L}(\Phi_\theta(\mathbf{x} \mid \hat{\mathbf{z}}), y) \tag{5.8}$$

where Φ is our SDF decoder network, and X is the set of observed single-view 3D points $\mathbf{x}$ and corresponding SDF measurements y we complete the shape of.

We assume the same prior distribution over $\hat{\mathbf{z}}_{\text{shape}}$ as in equation 5.6. This formulation (proposed by Park et al. 2019) applies to any number of SDF samples $(\mathbf{x}, y) \in X$ of arbitrary distributions, as we optimize $\hat{\mathbf{z}}$ separately for each SDF sample. As such it can handle any form of partial observation, and is not limited to single-views. This makes this shape completion method extendable to registered multi-view and online[5] shape completion.

We initialize $\hat{\mathbf{z}} = (\hat{\mathbf{z}}_{\text{shape}}, \hat{\mathbf{z}}_{\text{pose}})$ such that $\hat{\mathbf{z}}_{\text{pose}}$ has zero translation, a scale of 1, and a random rotation. $\hat{\mathbf{z}}_{\text{shape}}$ can be drawn near zero, but if the class or category of the object is known then one can initialize $\hat{\mathbf{z}}_{\text{shape}}$ near where that class clusters in latent space. We include the cost function $\mathcal{L}_{\text{codereg}}$ (eq. 5.7) in our loss, scaled with a hyperparameter, to ensure that $\hat{\mathbf{z}}_{\text{shape}}$ stays within the bounds of our compact latent space.

The 6D representation of rotation we use in $\hat{\mathbf{z}}_{\text{pose}}$ is very resistant to becoming malformed. This makes it fit not only as an inference target, its intended use case, but also as an optimization target. We normalize the rotation vector between every optimization step to not break any "assumptions" learned by the network, having only seen their canonical form.

To get the correct pose we first transform the samples in X to fit within the reconstruction volume, then apply the inverse transformation on the reconstructed shape.

We cover the different search strategies we experiment with during our evaluation.

[5] An online algorithm can process input data in the order the input is fed to the algorithm, without having the entire input available when starting. Often used to process live streaming sensor data.

5.6 Experimental Setup

We explore in this thesis with different network sizes, nonlinearities, and objective functions.

We experiment with the novel SIREN architecture (Sitzmann, Martel, et al. 2020), alongside a baseline multilayer perceptron (MLP) with ReLU-based nonlinearities. SIRENs are fully connected MLPs with sinusoidal nonlinearities whose harmonic and inductive characteristics are tuned by a hyperparameter ω_0. We construct both of these networks with varying number of *stages*, where each stage feed into the next stage concatenated with the input query coordinates and the latent vector via a skip connection.

Following Mildenhall et al. (2020) we experiment with using positional encoding (PE, defined in sec. 2.3.4.3). Tancik et al. (2020) show that using a Fourier mapping (a generalization of positional encoding) on the network inputs results in an increased bias to learn high-frequency components of the target signal.[6] We apply the positional encoding γ_n on the query coordinates $\mathbf{x}$ and on the orientation part of the latent space vector $\mathbf{z}_{\text{pose}}$ (defined in sec. 5.4.1). We apply γ_8 separately on each of the three coordinate values in $\mathbf{x}$ and the translation in $\mathbf{z}_{\text{pose}}$, and apply γ_4 on each of the 6D rotation coordinate values in $\mathbf{z}_{\text{pose}}$.

We investigate using different loss functions for training our networks and performing shape completion. We experiment with both L1 and L2 variants of the truncated SDF loss function $\mathcal{L}_{\text{TSDF}}$ proposed by Park et al. (2019) and the weighted loss function $\mathcal{L}_{\text{DISN}}$ proposed by Xu et al. (2019), defined below in equation 5.9 and visualized in figure 5.4. These loss functions penalize the network when its SDF prediction deviates from the actual SDF value. $\mathcal{L}_{\text{TSDF}}$ allows the network to "cheat" further away from the surface boundary by only accurately approximating a reduced SDF range, whereas $\mathcal{L}_{\text{DISN}}$ biases the zero-crossing with a weight. We additionally experiment with objective functions that constrain the analytical SDF gradient of the network: the loss function $\mathcal{L}_{\text{sim}}$ and the cost function $\mathcal{L}_{\text{norm}}$ proposed by Sitzmann, Martel, et al. (2020). $\mathcal{L}_{\text{sim}}$ constrains the orientation of the spatial SDF derivative while $\mathcal{L}_{\text{norm}}$ constrains the magnitude. $\mathcal{L}_{\text{norm}}$ constrains a property of the signed distance field that is independent of the shape, acting as regularization, and limits our shape completion search to valid fields only.

$$
\begin{aligned}
\mathcal{L}_{\text{TSDF}}(y, y_{\text{GT}}) &= \mathcal{L}_{\text{inner}}(\text{clamp}(y, -\delta_1, \delta_1), \text{clamp}(y_{\text{GT}}, -\delta_1, \delta_1)) \\
\mathcal{L}_{\text{DISN}}(y, y_{\text{GT}}) &= \begin{cases} m_1 \mathcal{L}_{\text{inner}}(y, y_{\text{GT}}) & \text{if } \; y_{\text{GT}} < \delta_2 \\ m_2 \mathcal{L}_{\text{inner}}(y, y_{\text{GT}}) & \text{otherwise} \end{cases} \\
\mathcal{L}_{\text{L1}}(y, y_{\text{GT}}) &= |y - y_{\text{GT}}| \\
\mathcal{L}_{\text{L2}}(y, y_{\text{GT}}) &= (y - y_{\text{GT}})^2 \\
\mathcal{L}_{\text{sim}}(y, y_{\text{GT}}) &= 1 - \langle \nabla_{\mathbf{x}} y, \nabla_{\mathbf{x}} y_{\text{GT}} \rangle \\
&= 1 - \frac{\nabla_{\mathbf{x}} y \cdot \nabla_{\mathbf{x}} y_{\text{GT}}}{\max(\|\nabla_{\mathbf{x}} y\|_2 \cdot \|\nabla_{\mathbf{x}} y_{\text{GT}}\|_2, \epsilon)} \\
&= 1 - \cos \phi_{y y_{\text{GT}}} \\
\mathcal{L}_{\text{norm}}(y) &= \|\nabla_{\mathbf{x}} y\|_2 - 1
\end{aligned}
\tag{5.9}
$$

where GT is short for "ground truth", $\mathcal{L}_{\text{inner}}$ is a nested loss function subject to experimentation, $\delta_1, \delta_2, m_1, m_2$ are hyperparameters with $m_1 > m_2$, $\text{clamp}(x, a, b)$ clamps x between a and b inclusive, $|\cdot|$ is the absolute value, $\langle \cdot, \cdot \rangle$ is the cosine similarity between two vectors (expanded in the following equality), $\epsilon = 10^{-8}$ prevents divisions by zero, $\|\cdot\|_2$ is the Euclidean norm, $\nabla_{\mathbf{x}}$ is the vector differential operator with respect to $\mathbf{x}$, and $\cdot$ is the inner product. The third equality of $\mathcal{L}_{\text{sim}}$ follows from the Euclidean dot product formula, where ϕ_{ab} is the angle between the gradients of a and b.

[6] The same mapping is already inherent to SIRENs, but across the whole network and not just on the inputs.

$\mathcal{L}_{\text{TSDF}}$ and $\mathcal{L}_{\text{DISN}}$ were in their original papers based on L1 regression, due to L2 loss "amounting to assuming Gaussian noise on the SDF values" (Park et al. 2019). Put differently: $\mathcal{L}_{\text{inner}} = \mathcal{L}_{\text{L1}}$. We investigate their L2 counterparts to see if they aid with shape completion on noisy data.

We combine all the loss functions defined in equation 5.9 into a combined loss function, which computes the mean loss over the training batch. Combined with the previously covered code cost ($\mathcal{L}_{\text{codereg}}$ in eq. 5.7) we derive the following loss function where each term is scaled with their own λ hyperparameter:

$$\begin{aligned}\mathcal{L}_{\text{combined}} = \lambda_1 \mathcal{L}_{\text{codereg}} + \frac{1}{|Y|} \sum_{(y, y_{\text{GT}}) \in Y} \Big[& \lambda_2 \mathcal{L}_{\text{TSDF}}(y, y_{\text{GT}}) \\ & + \lambda_3 \mathcal{L}_{\text{DISN}}(y, y_{\text{GT}}) \\ & + \lambda_4 \mathcal{L}_{\text{DISN,TSDF}}(y, y_{\text{GT}}) \\ & + \lambda_5 \mathcal{L}_{\text{sim}}(y, y_{\text{GT}}) \\ & + \lambda_6 \mathcal{L}_{\text{norm}}(y) \Big]\end{aligned} \tag{5.10}$$

where Y contains the predicted and ground truth (GT) SDF value pairs for a single training batch, and $\mathcal{L}_{a,b}$ denotes that a encloses the nested loss b. λ_2, λ_3 and λ_4 are mutually exclusive.

The composition of $\mathcal{L}_{\text{TSDF}}$ and $\mathcal{L}_{\text{DISN}}$ into $\mathcal{L}_{\text{DISN,TSDF}}$ is one of our novel contributions. Also novel is training SIRENs with $\mathcal{L}_{\text{TSDF}}$ and $\mathcal{L}_{\text{DISN}}$. Lastly, training networks with positional encoding over a continuous space of orientations has not been done before.

If $\mathcal{L}_{\text{TSDF}}$ is active then we can not have any expectations of what the networks do outside of the truncation range $\pm\delta_1$. As such for each prediction y we fix $\mathcal{L}_{\text{sim}}$ and $\mathcal{L}_{\text{norm}}$ to 0 when $(|y_{\text{GT}}| > \delta_1) \wedge (\lambda_2 \neq 0 \vee \lambda_4 \neq 0)$.

We use a **shorthand notation** when discussing the combined loss in equation 5.10 going forward. As an example: "$\mathcal{L}_{\text{DISN,L1+sim}}$" means that λ_3 and λ_5 are non-zero, and that $\mathcal{L}_{\text{DISN}}$ encloses $\mathcal{L}_{\text{L1}}$. λ_1 is independent of these shorthands. If $\mathcal{L}_{\text{sim}}$ is mentioned then it is scaled with $\lambda_5 = 0.04$. If $\mathcal{L}_{\text{norm}}$ is mentioned then it is scaled with $\lambda_6 = 0.015$. $\lambda_2, \lambda_3, \lambda_4$ are set to 1 if mentioned. When discussing network architectures, "ReLU-PE ∇" means that the network is constructed with ReLU nonlinearities, that it uses positional encoding (PE), and is supervised with $\mathcal{L}_{\text{sim}}$ (∇). We train SIRENs with both 3D (Euler) and 6D rotation, and denote them as such.

We expect the SIRENs to deal quite well with direct gradient supervision. The derivative of a SIREN is yet another SIREN after all. On the other hand, SIRENs have yet to be used to directly decode latent vectors. As SIREN can't "turn off" neurons as ReLU does, we expect SIRENs to either fail entirely or to draw even more connections between the shapes than ReLU.

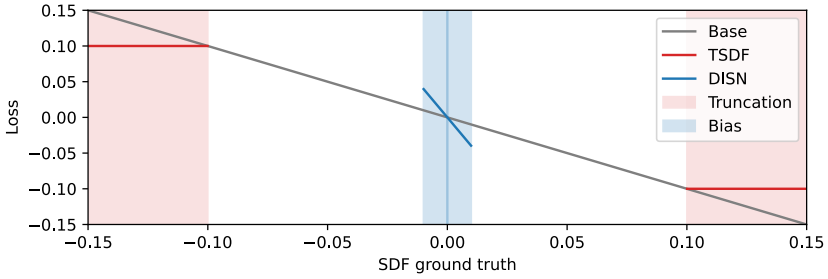


FIGURE 5.4: How the truncated (TSDF) and weighted (DISN) loss functions deviate from a baseline linear loss when we fix the prediction to zero. Truncating the signed distance reduces the range which needs to be accurately approximated. Biasing the zero-crossing with a large weight promotes learning more intricate surface details.

5.7 Evaluation Metrics

Following prior work we measure the accuracy of the SDF predictions against a separate *validation* dataset while training our networks, using the following two metrics:

Mean Squares Error (MSE), which measures of the quality of estimators. It is the mean of all the prediction errors, squared. MSE scores are always positive, with lower scores being better. The MSE of n predictions is given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i^{\text{GT}}\right)^2 \tag{5.11}$$

where y_i is the i'th predicted SDF value, and y_i^{GT} is the i'th ground truth.

Peak signal-to-noise ratio (PSNR), which is a widely adopted metric measuring the quality of signals. It computes the ratio between the peak signal power and the typical error. Higher ratios are better. Expressed in the logarithmic decibel scale it is given by:

$$\text{PSNR} = 10 * \log_{10}\left(\frac{\max_{i=0}^{n} y_i^2}{\text{MSE}}\right) \tag{5.12}$$

where y_i is the i'th predicted SDF value, and MSE is defined in equation 5.11.

We track both the SDF and TSDF MSE and PSNR. The TSDF variants are computed with truncated SDF values. We also compute for each SDF prediction the corresponding spatial gradient using backpropagation, and score these as well. We track their 2-norm magnitude (their Euclidean length) and their cosine similarity:

Cosine similarity is a metric for comparing the orientation of two vectors independent of their magnitude. We use it to both constrain ($\mathcal{L}_{\text{sim}}$) and score SDF gradients during training. Scores range in $[-1, 1]$, higher being better. Zero indicates the vectors are orthogonal, positive scores means the vectors have a similar orientation while negative scores indicate they are opposed. We compute the cosine similarity as:

$$\langle \nabla_{\mathbf{x}} y, \nabla_{\mathbf{x}} y \rangle = \frac{\nabla_{\mathbf{x}} y \cdot \nabla_{\mathbf{x}} y_{\text{GT}}}{\max(\|\nabla_{\mathbf{x}} y\|_2 \cdot \|\nabla_{\mathbf{x}} y_{\text{GT}}\|_2, \epsilon)} \tag{5.13}$$

where $\nabla_{\mathbf{x}} y$ is the predicted SDF gradient, and $\nabla_{\mathbf{x}} y_{\text{GT}}$ is the ground truth gradient.

We track both the SDF and the TSDF 2-norm magnitude and cosine similarity. The TSDF variants are computed using only the gradients whose ground truth signed distance lies within the truncation threshold ($\pm\delta_1$). These validation metrics is during training our "window" into the generalizable performance of our networks.

After the training, we score the shapes reconstructed from known latent vectors. For each shape $i \in \Omega$ we reconstruct a mesh M_i with marching cubes, then uniformly sample points along its surface and store them in U_i. Corresponding points are sampled from the ground truth mesh in V_i. We then compute, following prior work, these reconstruction metrics:

Chamfer Distance (CD) is a metric for scoring the difference between two point clouds, known as the *average* case metric. Lower scores are better. It is given by:

$$\begin{aligned}\text{CD} = &\frac{1}{|U_i|}\sum_{\mathbf{u}\in U_i}\min_{\mathbf{v}\in V_i}\|\mathbf{u}-\mathbf{v}\|_2^2 \\ &+ \frac{1}{|V_i|}\sum_{\mathbf{v}\in V_i}\min_{\mathbf{u}\in U_i}\|\mathbf{v}-\mathbf{u}\|_2^2\end{aligned} \tag{5.14}$$

where $|\cdot|$ is the number of items in a set.

Earth Mover's Distance (EMD), also known as the Wasserstein distance, is an *optimal transport* metric for scoring the difference between two distributions. Lower scores are better. EMD is commonly known as the *best case* metric when applied to point clouds. It favors the inferred points to be distributed in a similar fashion to the ground truth points. As such it can not be "cheated" as one can with the Chamfer Distance, as a one-to-one correspondence between the points is constructed. The earth mover's distance is given by:

$$\text{EMD} = \min_{\phi: U_i \to V_i} \sum_{\mathbf{u} \in U_i} \|\mathbf{u} - \phi(\mathbf{u})\|_2 \tag{5.15}$$

where $\phi : U_i \to V_i$ is a bijection (a one-to-one matching) from U_i to V_i.

Xu et al. (2019) claim the CD and EMD metrics mostly penalize the overall shape, leaving out smaller details. To address this we include the following two metrics:

Mesh Cosine Similarity (COS) is a metric for scoring the orientation of the normal vectors in a generated mesh. Higher scores are better. For each ground truth surface point in V: find the closest face F in the inferred mesh M_i and compute the cosine similarity between their normal vectors. The average mesh cosine similarity is thus given by:

$$\text{COS} = \frac{1}{|V_i|} \sum_{\mathbf{v} \in V_i} \hat{\mathbf{n}}_{\mathbf{v}} \cdot \hat{\mathbf{n}}_{\arg\min_{F \in M_i} \|\mathbf{v} - F\|_2} \tag{5.16}$$

where $\hat{\mathbf{n}}_x$ is the normal vector of x.

F-Score is a metric for scoring binary classifiers, or put differently: scoring positive and negative measurements. It is concerned with detecting false positives and false negatives, i.e. the *correctness* of the classification. We compute the *precision*[7] and *recall*[8] over all the surface points in U_i and V_i. Tatarchenko et al. (2019) demonstrates that the F-score is a comprehensive evaluation of the amount of correctly reconstructed surface area, when we define a point in U_i or V_i to be *positive* if its distance to the nearest point in the other set is less than a threshold value $c \in \mathbb{R}_{>0}$. We compute the precision and recall as defined in equation 5.17 below, and combine them into a F_β measure, where $\beta \in \mathbb{R}_{\geq 0}$ puts more emphasis on one or the other. $\beta = 1$ results in their harmonic mean. The score ranges from 0 to 1, with 1 indicating a perfect precision and recall.

$$\begin{aligned} F_\beta &= (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \\ \text{precision} &= \frac{\left|{}_{\text{Positive}}^{\text{True}}\right|}{\left|{}_{\text{Positive}}^{\text{True}}\right| + \left|{}_{\text{Positive}}^{\text{False}}\right|} = \frac{|\{\mathbf{u} \mid \mathbf{u} \in U_i, \min_{\mathbf{v} \in V_i} \|\mathbf{u} - \mathbf{v}\|_2^2 < c\}|}{|U_i|} \\ \text{recall} &= \frac{\left|{}_{\text{Positive}}^{\text{True}}\right|}{\left|{}_{\text{Positive}}^{\text{True}}\right| + \left|{}_{\text{Negative}}^{\text{False}}\right|} = \frac{|\{\mathbf{v} \mid \mathbf{v} \in V_i, \min_{\mathbf{u} \in U_i} \|\mathbf{u} - \mathbf{v}\|_2^2 < c\}|}{|V_i|} \end{aligned} \tag{5.17}$$

[7] Precision is the proportion of positive identifications that is actually correct, also referred to as the true positive rate or *sensitivity*.

[8] Recall is the proportion of actual positives identified correctly, also referred to as positive predictive value.

Chapter 6

Evaluation

We present in this chapter our evaluation of major experiments. We only provide for the sake of brevity a broad description of the minor experiments, due to the sheer size of our experimental setup. We start of with implementation details of our model described in chapter 5 pertinent to our evaluation, and move on to our experiments training them. We then inspect and compare 3D meshes reconstructed from known latent vectors between competing models, then evaluate how their learned knowledge transfer over to shape completion in a single-view context. We encourage digital readers to use a dual-page view[1], as this chapter feature many large floating figures that cover entire pages.

Section 6.1 starts of with data preparation results.
Section 6.2 broadly covers the training of our models.
Section 6.3 evaluates the quality of 3D meshes reconstructed from known shape codes.
Section 6.4 explores the characteristics of the latent spaces formed by competing models.
Section 6.5 evaluates the shape completion ability of our model from a single-view.

6.1 Data Preparation

This section covers the preparation of training and evaluation data as outlined in chapter 5.2. We sampled full-view SDF clouds with from 3D meshes for use in training, and sampled single-view SDF clouds from both synthetic scans and RGB-D images for use in single-view evaluation. We based our processing pipeline on the publicly available `mesh_to_sdf` library made available by Kleineberg, Fey, and Weichert (2020).

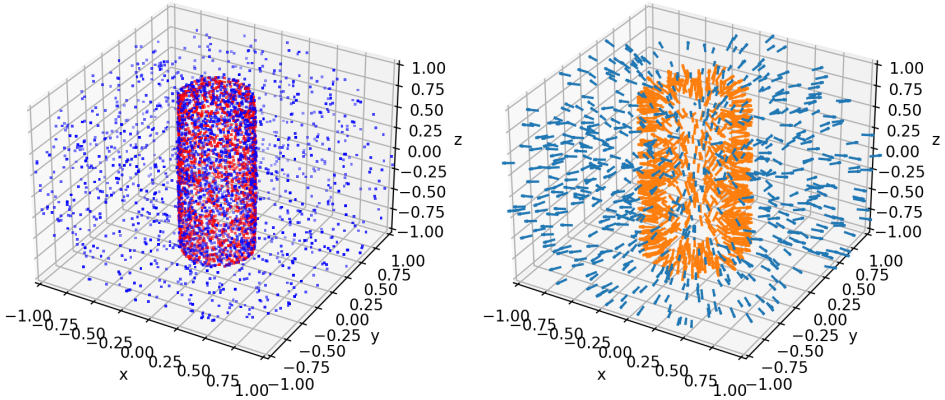
Some of the BigBIRD meshes featured incorrect face normals and windings. We tried fixing these automatically[2] but it did not work for all objects. `001_chips_can` is particularly affected, having inverted normals on its head. As such we omit back-face culling when scanning meshes, but the illumination in some of the figures presented in this thesis are still affected.

6.1.1 Sampling SDF Gradients

The `mesh_to_sdf` library implements the full-view SDF sampling strategy proposed by Park et al. (2019). We use it to sample 250 000 points from each object in the YCB dataset, 92% near-surface and 8% uniformly. We extended `mesh_to_sdf` to additionally compute the spatial gradients of the SDF values, and merged these contributions upstream for others to benefit from.

[1] Dual-page view with this page on the right (odd pages left).
[2] Using https://trimsh.org/trimesh.repair.html#trimesh.repair.fix_normals

We computed the spatial gradients for uniformly sampled SDF points using the direction to the nearest surface point scanned by the 3D virtual cameras. The spatial gradients for *near-surface* samples however could not use this strategy as it proved too noisy. For this reason we instead fix the near-surface SDF gradients to the normal vector of corresponding mesh face of the nearest scanned surface point. We showcase a resulting SDF cloud and its spatial gradients in figure 6.1.



(A) SDF cloud. Blue is positive, red is negative.

(B) Gradient cloud. Blue vectors are uniform, orange are near-surface.

FIGURE 6.1: A full-view SDF cloud of the `001_chips_can` YCB object. Our training dataset consists of clouds like these, where 92% of the points are sampled near-surface and 8% are sampled uniformly within a sphere with radius $\sqrt{3}$. Here we show a coarse cloud with radius $\sqrt{2}$: (A) has 1200 uniform and 3500 near-surface SDF samples. (B) has 600 uniform and 600 near-surface vectors.

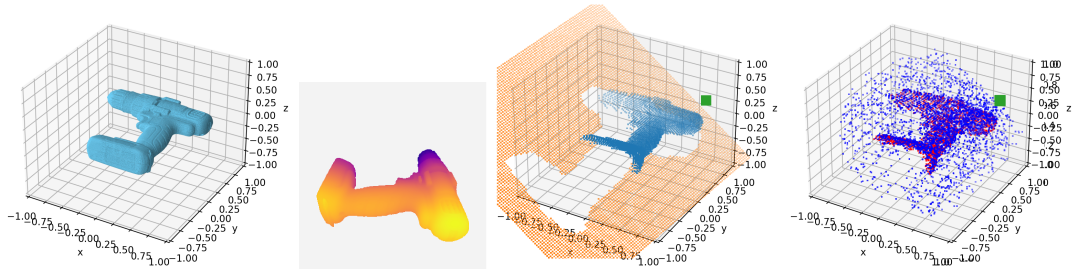
6.1.2 Single-View Point Clouds

We create single-view SDF clouds from both synthetic scans and from real-world RGB-D images provided by the YCB dataset.

The synthetic scans are created by rendering the high quality Google scanner models provided by YCB to 600×600 depth buffers from random viewpoints. These depth maps are then unprojected back into point clouds in model space using the inverse MVP transform. We compute both *hit* and *miss* points form these depth maps, with the depth of the missing points fixed to the far clipping plane. We showcase the steps of producing a synthetic single-view hit+miss cloud in figure 6.2a-c. We implemented the single-view SDF sampling procedure outlined in section 5.2.3 and apply it to a synthetic hit+miss cloud in figure 6.2d.

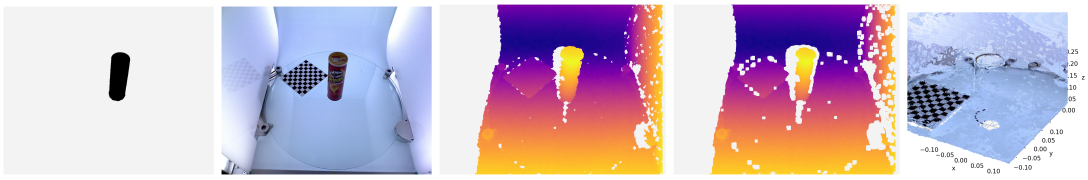
We process[3] the real-world single-view RGB-D depth observations to hit+miss point clouds as described in section 5.2.4. Figure 6.3 showcases raw RGB-D data and resulting hit+miss clouds and SDF clouds for five equidistant turntable angles. We can not post-process the single-view point clouds using the same techniques as the BigBIRD scanner use, as its techniques only work on full-view clouds combined from many single-view clouds. Due to this we observe some erroneous points around the fringes of the objects, even after the application of depth discontinuity filtering.

[3]We implemented rudimentary image processing ourselves, including lens distortion correction, depth map filtering and the unprojection into 3D point clouds. This is normally handled by the Intel RealSense framework in our targeted lab environment, but it is not trivial to use on YCB data.

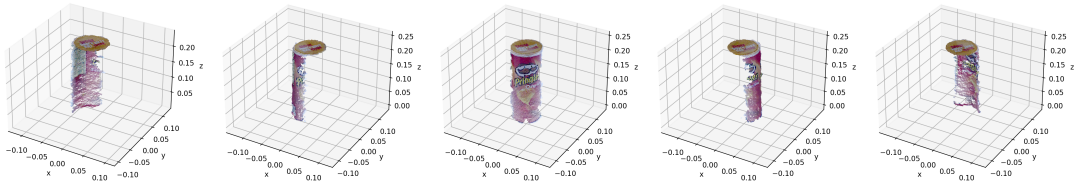


(A) Google mesh (B) Depth buffer (C) Hit+miss cloud (D) SDF cloud

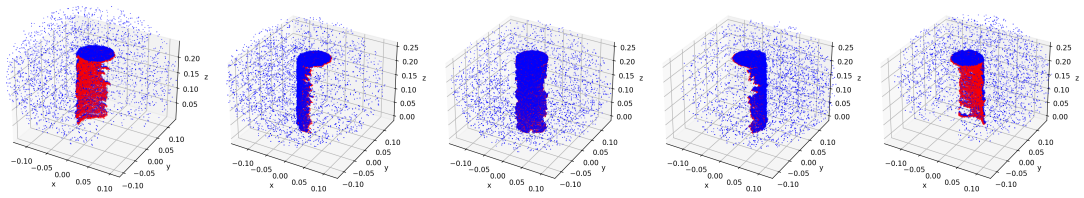
FIGURE 6.2: The process of generating a synthetic single-view point cloud from a 3D mesh. The `035_power_drill` mesh (A) is here rasterized to a depth buffer (B) where orange is near the camera. The buffer is unprojected into model space as a hit+miss point cloud (C) where blue points are hits, orange are misses, and green is the camera position. The hit+miss cloud is used to sample a SDF cloud (D) where blue points are positive and red are negative. Note how the near-surface samples in (D) are distributed more uniformly than the hit points in (C).



(A) Mask (B) Color (C) Depth (D) Filtered depth (E) Miss



(F) 213° Hit (G) 285° Hit (H) 357° Hit (I) 69° Hit (J) 141° Hit



(K) 213° SDF (L) 285° SDF (M) 357° SDF (N) 69° SDF (O) 141° SDF

FIGURE 6.3: A segmentation mask (A), color image (B) and depth image (C) of the YCB object `001_chips_can` taken from the NP3 BigBIRD perspective. Note how the color and depth images have slightly different camera perspectives. (D) shows the results of applying discontinuity filtering to (C). (F-J) show *hit* point clouds produced from these images for various turntable rotations, aligned to the checkerboard. (E) visualize all the *miss* points merged into a single cloud. In (K-O) we showcase single-view SDF clouds sampled from the corresponding hit+miss point clouds, where blue points are positive and red negative.

6.2 Training

This section covers how we trained our networks and how certain findings influenced our design decisions along the way. Finally we present the SDF MSE and PSNR validation metrics measured after the last training epoch, for the most promising architectures.

We implemented our neural network in PyTorch and trained using the PyTorch-Lightning framework. The network and its trainer can be configured to use the various nonlinearities and loss functions covered in section 5.6.

We used Adam optimization with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ decay rates. We varied the learning rate η a lot throughout our experimentation, which we cover in the following subsection. We landed on a final learning rate of $\eta_{\text{L1}} = 5 \times 10^{-5}$ for the L1-based loss functions and $\eta_{\text{L2}} = 5 \times 10^{-4}$ for L2. We fixed the hyperparameters defined in eq. 5.9 to the values used in their original papers: $\delta_1 = 0.1$, $\delta_2 = 0.01$, $m_1 = 4$, and $m_2 = 1$.

We trained ReLU-based MLPs with a 10% dropout rate, and used 16-bit floating point precision for both ReLU and SIREN. The reduced precision enabled us to fit larger batches of training data in memory, optimizing over larger subset of objects for each step. We briefly experimented with a stochastic weight averaging training schedule[4], but its implementation in PyTorch Lightning promptly broke upstream while we were experimenting with it, making us abandon it for this thesis.

6.2.1 Discoveries, Optimization and Re-Design

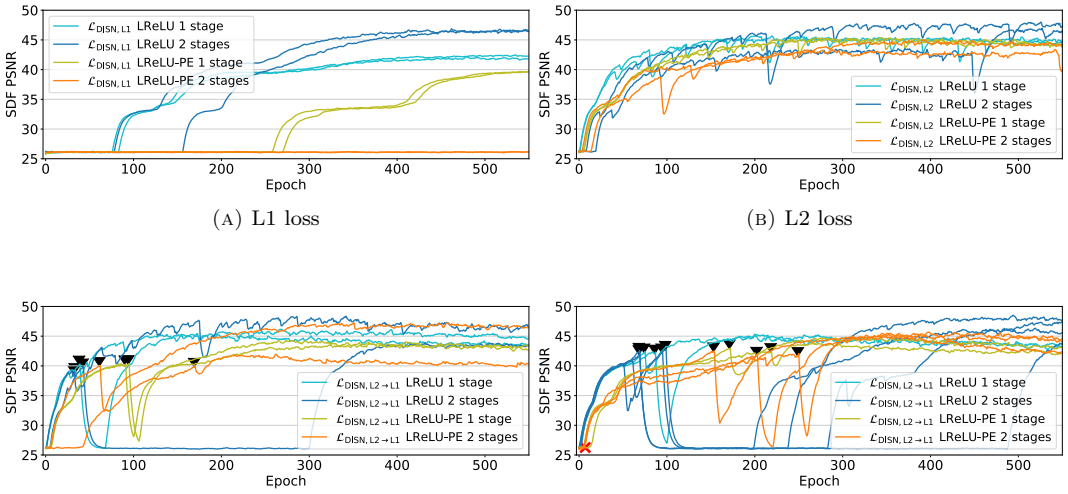
We discovered early on that L2-based reconstruction loss started converging much earlier than L1 did. We showcase validation data logged during training, smoothed with exponential moving average (EMA), in figure 6.4. Seeing this we based much of our earlier experimentation on L2. L2 had trouble figuring out the relation between $\mathbf{z}_{\text{pose}}$ and the augmented SDF samples. The training shapes were approximated correctly near the origin, but got warped near the peripheries. This resulted in boxes and cylinders becoming noticeably concave or convex. The networks trained with L2 loss did not become any more or less robust to sensory noise. Seeing this we switched to primarily using L1 loss. We briefly experimented speeding up initial convergence with a L2→L1 schedule triggered by reaching a target validation MSE threshold. This midway change of loss proved unstable, causing the networks to immediately diverge once hitting the trigger. We instead got L1 to converge earlier by using weight normalization, a slower learning rate, and by reducing the initial magnitude of the latent vectors.

We saw on multiple occasions the SDF MSE and PSNR scores worsening while the mean SDF gradient cosine similarity scores continued to rise, even when training *without* SDF gradient supervision. This prevents us from using a early-stopping strategy, as no single validation metric alone conclusively indicates overfitting. Finding better metrics to observe the validation performance of SDF decoders could greatly benefit future research.

We initially trained on one object at a time in a serial fashion like Park et al. (2019). We saw an increased rate of convergence when we switched to our own training order covered in section 5.4.3.

We trained with and without 6D rotation encoding, otherwise falling back to Euler angles. We found that 6D rotation was a major improvement for ReLU-based networks, whereas SIRENs performed measurably worse with it. We tried additionally feeding in the cross product between the two halves of the 6D rotation vectors, effectively resulting in the full 3×3 rotation matrix. Both ReLU and SIREN converged faster with the cross products, now not having to regress them themselves.

[4]Stochastic Weight Averaging approximates fast geometric ensembling at a fraction of the computational cost. It averages learned parameters between multiple short runs with a cyclical learning rate, making it adjacent to meta-learning.



(A) L1 loss

(B) L2 loss

(C) L2 changed to L1 when SDF MSE=0.001

(D) L2 changed to L1 when SDF MSE=0.0001

FIGURE 6.4: ReLU-based networks with one and two stages, trained with both L1 and L2 variants of $\mathcal{L}_{\text{DISN}}$. We plot the SDF PSNR measured across the validation dataset, smoothed with α=0.8 EMA. L2 (B) loss began converging earlier than L1 (A) did. A L2→L1 schedule (C-D) proved unstable and difficult to tune. ▼ denotes when the loss changed. Note how (A) is smoother than (B-D).

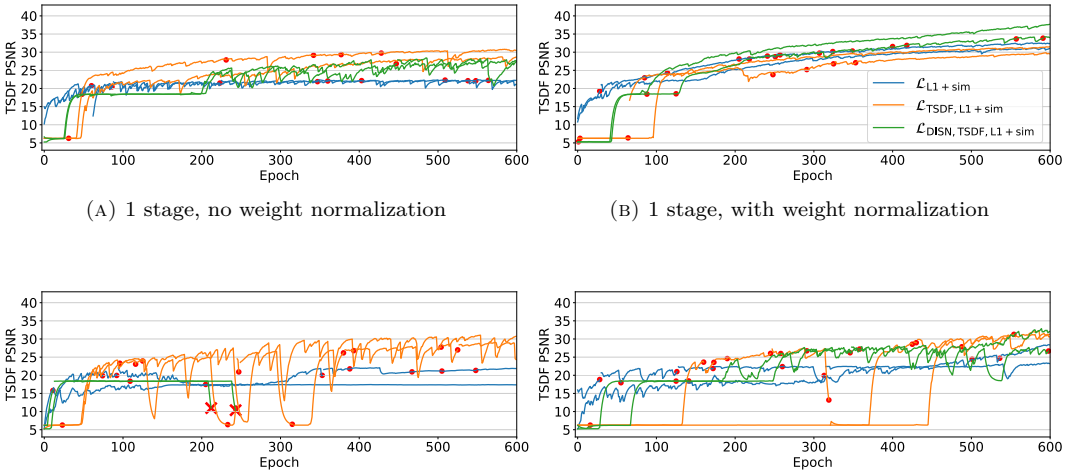
16-bit floating point precision initially proved unstable on the deeper ReLU networks. We mitigated this with weight normalization, improving the performance of ReLU-based networks constructed with two or more stages. Weight normalization improving the performance of SIRENs however is not a "given". SIRENs expect the weights of the fully connected layers to be drawn from a uniform distribution parametrized with ω_0, an assumption weight normalization may break. Deep SIRENs have been shown to not suffer from either vanishing nor exploding gradients, so we thought the lack of weight normalization would not pose a problem. Yet we observed that SIRENs got more unstable the more stages we added, and thus we briefly experimented with adding weight normalization. We show in figure 6.5 how weight normalization drastically improved the performance of SIRENs with one and two stages. Exactly why that is has yet to our knowledge to be formally proved, and may become the subject of a separate paper.

We initially trained with 64 features wide latent vectors, fewer features than training shapes. (Supplementary table A.2 list the 83 shapes.) 64 features proved insufficient for ReLU-based MLPs to embed all the shapes. Most of the shapes got squashed into lathed[5] approximations, for instance simplifying boxes into cylinders. A dataset with more similar shapes may not have had the same problem. Seeing this we worked our way up using larger latent vectors, stopping at vectors 128 features wide. Consequently we later must verify that the learned codes are not all orthogonal.

Initially we did not properly scale the signed distance values when augmenting with random orientations as defined in equation 5.5. The original DeepSDF architecture proved well able to learn these ill-formed fields. These fields not being valid however would have cause problems further down the line. Once we started scaling the SDF properly the networks would suddenly not converge anymore. This is when we introduced the final NeRF stage into our architecture as visualized in figure 5.3, solving the problem.

All experiment using $\mathcal{L}_{\text{norm}}$ diverged immediately. As we always paired $\mathcal{L}_{\text{sim}}$ and $\mathcal{L}_{\text{norm}}$ together

[5]Lathed shapes are symmetrical about an axis of rotation. These are the shapes a real lathe produce. Lathing may be partial.



(A) 1 stage, no weight normalization

(B) 1 stage, with weight normalization

(C) 2 stages, no weight normalization

(D) 2 stages, with weight normalization

FIGURE 6.5: SIRENs (ω_0=25) with one and two stages, trained both with and without weight normalization. We plot the TSDF PSNR measured across the validation dataset, smoothed with α=0.8 EMA. Observe how (B) and (D) trained *with* weight normalization converged more steadily, and had yet to plateau after 600 epochs. Red dots and crosses show NaNs, the latter indicating the network never recovered. SIRENs seem to produce and recover from regressing NaNs quite often, a characteristic not observed with ReLU.

we figured both were broken. A great while later did we discover what the problem was. All our network converged on a mean SDF gradient magnitude around 0.1. (1 is expected.) $\mathcal{L}_{\text{sim}}$ thankfully worked quite well when used alone. We checked for mistakes in how we generated SDF training samples or how we calculated the SDF gradients, but found none. For this reason we had to stop using $\mathcal{L}_{\text{norm}}$ entirely. This is a shame, as it would have been a great tool to regularize the latent space in areas where the shape is not known, as it constrains an invariant property of signed distance field that is independent of the shape.

We trained initially with a $\mathcal{L}_{\text{codereg}}$ cost of $\lambda_1 = 0.01^2$ as proposed by Park et al. (2019). The resulting latent spaces were uneven and proved difficult to search efficiently. Observing this we bumped λ_1 up to 0.04^2, which yielded a more compact shape manifold. With this change the SIRENs saw an increase in shape completion performance, the reason for which is visible in figure 6.8c.

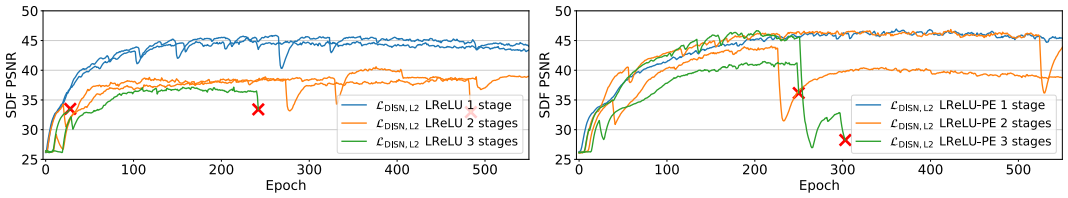
6.2.2 Finding the Best Combination

We trained in total 821 networks on the GPGPU compute cluster by Själander et al. (2019). The majority of our efforts was here spent tweaking hyperparameters and ensuring the networks produced sound looking output. We started of small, incrementally training larger and larger networks. We use a final hidden width of 512 neurons, same as most other related works. We compare the 8 following objective functions: $\mathcal{L}_{\text{L1}}$, $\mathcal{L}_{\text{L1+sim}}$, $\mathcal{L}_{\text{TSDF,L1}}$, $\mathcal{L}_{\text{TSDF,L1+sim}}$, $\mathcal{L}_{\text{DISN,L1}}$, $\mathcal{L}_{\text{DISN,L1+sim}}$, $\mathcal{L}_{\text{DISN,TSDF,L1}}$, and $\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$. The building blocks for these loss functions are defined in equation 5.9.

The measured SDF MSE and PSNR plateaued early when we trained LReLU MLPs with only 64 shape features with $\mathcal{L}_{\text{DISN,L1}}$ loss, a problem $\mathcal{L}_{\text{TSDF,L1}}$ did not exhibit. $\mathcal{L}_{\text{DISN,L1+sim}}$ resulted in more accurate gradients, at the cost of an increased signed distance error. We believe $\mathcal{L}_{\text{DISN}}$ allocated more latent features to describe the full SDF range instead of focusing on details near the zero boundary. Interestingly $\mathcal{L}_{\text{DISN,L1}}$ outperformed $\mathcal{L}_{\text{TSDF,L1}}$ when we used 128 latent features. Combining the two $\mathcal{L}_{\text{DISN,TSDF,L1}}$ quickly learned the finer surface details, achieving

a higher PSNR despite a higher MSE. $\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$ performed better than $\mathcal{L}_{\text{DISN,TSDF,L1}}$, being improved by SDF gradient supervision. $\mathcal{L}_{\text{DISN,L1+sim}}$ resulted in better SDF gradients than $\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$, but performed worse when compared against $\mathcal{L}_{\text{DISN,L1}}$ in terms of MSE and PSNR.

Positional encoding (PE) helped deeper networks convergence initially, at the cost of a lower final accuracy. We show this behavior in figure 6.6. The deeper networks responded better to PE, likely as a result of their higher capacity for rote memorization. We experimented with multiple values of n in γ_n ranging from 4 to 12, but could do not determine any n as being more optimal. PE did not respond well to SDF gradient supervision, likely not utilizing the positionally encoded features at all.



(A) No encoding (B) Positional encoding

FIGURE 6.6: ReLU-based networks without weight normalization, trained both with and without positional encoding (PE). We plot the SDF PSNR measured across the validation dataset, smoothed with α=0.8 EMA. PE seems to aid the deeper networks learn, while slowing down the more shallow ones. The red crosses are NaNs, showcasing how networks may suddenly diverge without weight normalization.

SIRENs proved highly volatile to train and required SDF gradient supervision to perform well at all. Without gradient supervision they spent on average 300 epochs before starting to converge. With even basic $\mathcal{L}_{\text{L1+sim}}$ loss however they suddenly converged right away. We observed that SIRENs regressed well oriented SDF gradients almost immediately, then spent around 30 epochs before the TSDF PSNR started to climb. This indicates that the SIRENs found the curvature of the distance field early, but needed time to move the isosurface towards 0 and even out the field. Curiously, the SIRENs we trained with $\mathcal{L}_{\text{DISN,L1+sim}}$ would not converge until after at least 350 epochs had passed. Pressed for time we did not investigate why. $\mathcal{L}_{\text{TSDF,L1+sim}}$ converged as normal and is not affected this problem. $\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$ again achieved the highest SDF PSNR measurements despite a worse MSE. $\mathcal{L}_{\text{L1+sim}}$ and $\mathcal{L}_{\text{TSDF,L1+sim}}$ had the better MSE scores and SDF gradients orientations.

We experimented with various values of ω_0 in SIREN networks, ranging from 15 up to 45. (Sitzmann, Martel, et al. 2020 recommend $\omega_0 = 30$.) We found no major differences within the range 25 ± 5, and thus we fixed it to 25.

We switched from ReLU to LReLU along the way, to avoid the "dying ReLU" problem. We did not have time to extensively measure the difference in performance between ReLU and LReLU however, nor any of the other related nonlinearities such as SiLU.

Overall we observed the following trends: Including $\mathcal{L}_{\text{sim}}$ made the networks converge earlier and at a more steady pace. With it the variation between training runs was greatly reduced. It improved the mean SDF cosine similarity by about 13% after reaching 600 epochs, diminishing to a 6% lead after 1500 epochs. On LReLU, $\mathcal{L}_{\text{sim}}$ outperformed not using gradient supervision until about 350 epochs, after which $\mathcal{L}_{\text{TSDF}}$ and $\mathcal{L}_{\text{DISN}}$ overtook their $\mathcal{L}_{\text{sim}}$ counterparts in terms of SDF MSE and PSNR. The inclusion of $\mathcal{L}_{\text{sim}}$ increased the total magnitude of the loss, making the $\mathcal{L}_{\text{codereg}}$ cost less effective.

We trained LReLU MLPs and SIRENs of various depths, comparing networks constructed with one, two, and three stages, capped of with a final NeRF stage. We concluded that LReLU

MLPs perform best with 2 stages, likely suffering from vanishing gradients when constructed with more. Deeper networks likely have a greater capacity to learn complex shapes, but could not do so within the 1500 epochs we trained for. SIRENs performed best when constructed with only a single stage, getting more unstable the more stages we added. This we attribute to SIRENs possibly not responding well to the skip connections between each stage. Future works should explore different SIREN architectures than ones based on DeepSDF and NeRF.

6.2.3 The Final Training Batch

Here we train our final batch of networks for 1500 epochs, using the settings and hyperparameters we established as being optimal in the previous two sections. We present in table 6.1 below the final SDF MSE, PSNR and mean field gradient cosine similarities measured across the validation dataset. The LReLU MLPs were reaching their plateau, while the SIRENs were converging at a steady rate when we terminated the training. Network #4 diverged and predicted NaNs[6] midway in. We tried in vain to train it multiple times.

TABLE 6.1: The final SDF MSE($\times 10^7$), PSNR and mean $\langle \nabla_{\mathbf{x}} \rangle$ (gradient cosine similarity) measurements for a batch of networks trained for 1500 epochs. **Bold** highlights the best scores in each group. The networks were trained with weight normalization, $0.04^2 \mathcal{L}_{\text{codereg}}$, and $\mathbf{z}_{\text{shape}}$ vectors 128 features wide. PE is positional encoding, n is the number of network stages not counting the final NeRF stage. These metrics are defined in chapter 5.7, and graphed over time in supplementary figure A.3.

#	Type	n	Training loss	SDF MSE↓	SDF PSNR↑	TSDF MSE↓	TSDF PSNR↑	TSDF $\langle \nabla_{\mathbf{x}} \rangle$↑
1	LReLU-PE	2	$\mathcal{L}_{\text{TSDF,L1}}$	-	-	824	27.5	0.603
2	LReLU-PE	2	$\mathcal{L}_{\text{DISN,L1}}$	4025	43.4	**354**	30.8	**0.734**
3	LReLU-PE	2	$\mathcal{L}_{\text{DISN,TSDF,L1}}$	-	-	3578	**30.9**	0.600
4	LReLU-PE	2	$\mathcal{L}_{\text{TSDF,L1+sim}}$	-	-	*NaN*	*NaN*	*NaN*
5	LReLU-PE	2	$\mathcal{L}_{\text{DISN,L1+sim}}$	10932	39.1	**431**	29.9	**0.868**
6	LReLU-PE	2	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	-	-	1400	**35.3**	0.846
7	LReLU	2	$\mathcal{L}_{\text{TSDF,L1}}$	-	-	267	32.4	0.817
8	LReLU	2	$\mathcal{L}_{\text{DISN,L1}}$	5810	41.8	**263**	32.1	0.821
9	LReLU	2	$\mathcal{L}_{\text{DISN,TSDF,L1}}$	-	-	432	**39.6**	**0.839**
10	LReLU	2	$\mathcal{L}_{\text{TSDF,L1+sim}}$	-	-	377	30.7	0.855
11	LReLU	2	$\mathcal{L}_{\text{DISN,L1+sim}}$	7457	40.8	**358**	30.9	0.880
12	LReLU	2	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	-	-	601	**37.9**	**0.897**
13	SIREN 3D	1	$\mathcal{L}_{\text{L1+sim}}$	906	50.4	**249**	32.7	0.944
14	SIREN 3D	1	$\mathcal{L}_{\text{TSDF,L1+sim}}$	-	-	317	31.5	**0.961**
15	SIREN 3D	1	$\mathcal{L}_{\text{DISN,L1+sim}}$	904	50.3	277	32.1	0.943
16	SIREN 3D	1	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	-	-	357	**40.8**	0.956
17	SIREN 6D	1	$\mathcal{L}_{\text{L1+sim}}$	544	52.6	**171**	34.3	0.962
18	SIREN 6D	1	$\mathcal{L}_{\text{TSDF,L1+sim}}$	-	-	287	32.1	**0.965**
19	SIREN 6D	1	$\mathcal{L}_{\text{DISN,L1+sim}}$	1044	49.6	302	31.7	0.945
20	SIREN 6D	1	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	-	-	544	**39.4**	0.958

This table indicates that the SIRENs performed better in general than the LReLU MLPs, but the validation metrics disagree on which loss function was the best.

We could not at this stage determine whether the SDF MSE or PSNR was the better metric,

[6]Not a Number (NaN): Invalid floating point operations, such as divisions by zero, may in certain systems produce NaN values instead of halting the computation.

not having measured the quality of reconstructed meshes. We believe however none of them are a good metric alone: $\mathcal{L}_{\text{DISN,TSDF}}$ produced by far the best PSNR scores, but this came with the cost of an increased MSE. We saw this behavior in both LReLU and SIREN. An increased MSE should in theory lower the PSNR score, meaning that something has to compensate. This hints at there being some sort of trade-off in play, making the PSNR not conclusively indicate higher reconstruction quality or generalization.

LReLU: The LReLU MLPs did not for our applications benefit from positional encoding (PE). PE have worked wonders for related works featuring canonical-only poses, but it did not work over a continuum of different orientations. We attribute their subpar performance to how transforming positionally encoded query coordinates to a different orientation is a very hard problem to solve, even for machines. Applying the rotation of a 6D vector to raw coordinates on the other hand is just a simple linear relation, perfect for non-PE LReLU MLPs to learn. The LReLU-PE MLPs likely resorted to a great deal of rote memorization, generalizing poorly to new unseen shapes.

For non-PE LReLU we observe that $\mathcal{L}_{\text{DISN}}$ outperformed $\mathcal{L}_{\text{TSDF}}$ on nearly every validation metric, even achieving the lowest MSE within the TSDF range. We expected $\mathcal{L}_{\text{TSDF}}$ to produce more accurate field gradients than $\mathcal{L}_{\text{DISN}}$, though this proved not to be the case. $\mathcal{L}_{\text{DISN,TSDF}}$ produced even more accurate field gradients than $\mathcal{L}_{\text{DISN}}$ did however, indicating both are benefitial.

$\mathcal{L}_{\text{sim}}$ trades an increased SDF error for more accurate field gradients. Whether this is beneficial for the quality of the reconstructed shapes or not determines whether $\mathcal{L}_{\text{sim}}$ should be used. We note that $\mathcal{L}_{\text{sim}}$ made the networks converge at a much faster rate initially, and in a more steady manner with less variation between training runs. This makes $\mathcal{L}_{\text{sim}}$ an attractive loss function for finding meta-learned initializations for few-shot optimization, applicable to the work of Tancik et al. (2021).

SIREN: The SIRENs produced the best measurements across the board, exhibiting by far the most accurate field gradients. This is despite being half the size of the LReLU MLPs. Seeing this we expect SIRENs to produce the best shape reconstructions.

SIRENs initially converged at a faster rate when trained with 6D rotation input (and cross products) instead of traditional Euler angles, but their performance evened out as the training progressed. 6D rotation working this well is beneficial for shape completion, as Euler angles are susceptible to gimbal lock. The SIRENs seemed in terms of MSE and PSNR to perform better with basic $\mathcal{L}_{\text{L1,sim}}$ loss compared to $\mathcal{L}_{\text{TSDF,L1+sim}}$ and $\mathcal{L}_{\text{DISN,L1+sim}}$. The two SIRENs trained with $\mathcal{L}_{\text{DISN,L1+sim}}$ spent around 400 epochs idle before converging at all, but then suddenly caught up to the truncated SIRENs over the following 900 epochs. The fact that network #15 surpassed #14 at all in terms of TSDF MSE is so far an anomaly, as we have not seen this happen in any of the earlier experiments.

6.2.4 Training Time

The two-stage LReLU MLPs without SDF gradient supervision each spent on average six-and-a-half hours training 1500 epochs on a Nvidia V100. With SDF gradient supervision, that time jumped up to 11 hours. LReLU MLPs with positional encoding each spent around ten hours without gradient supervision, and 17 hours with it. Single-stage SIRENs training with SDF gradient supervision each spent on average 8 hours.

6.3 Evaluation of Reconstructed Shapes

Here we evaluate the quality of 3D meshes reconstructed from known latent vectors. We report in table 6.2 various metrics scoring the distance between reconstructed shapes and their corresponding ground truth meshes:

TABLE 6.2: The $\overline{\text{mean}}$ and $\widetilde{\text{median}}$ CD($\times 10^4$), EMD($\times 10^7$) and COS, defined in chapter 5.7, for each network in table 6.1. CD and EMD measure distances inside the unit-scale reconstruction volume. **Bold** highlights the best scores in each group, and the three best performing networks. Network #7 is comparable to DeepSDF. We further explore the those marked * from here on, chosen by their median performance.

#	Type	Training loss	$\overline{\text{CD}}\downarrow$	$\widetilde{\text{CD}}\downarrow$	$\overline{\text{EMD}}\downarrow$	$\widetilde{\text{EMD}}\downarrow$	$\overline{\text{COS}}\uparrow$	$\widetilde{\text{COS}}\uparrow$
1	LReLU-PE	$\mathcal{L}_{\text{TSDF,L1}}$	9.816	6.084	3.279	1.828	0.484	0.447
2	LReLU-PE*	$\mathcal{L}_{\text{DISN,L1}}$	16.846	**1.571**	4.860	**0.574**	0.534	0.630
3	LReLU-PE	$\mathcal{L}_{\text{DISN,TSDF,L1}}$	**9.194**	5.151	**2.758**	1.840	0.566	0.634
4	LReLU-PE	$\mathcal{L}_{\text{TSDF,L1+sim}}$	-	-	-	-	-	-
5	LReLU-PE*	$\mathcal{L}_{\text{DISN,L1+sim}}$	21.892	1.949	6.331	0.851	**0.575**	**0.743**
6	LReLU-PE	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	14.582	3.552	4.619	1.526	0.571	0.679
7	LReLU	$\mathcal{L}_{\text{TSDF,L1}}$	9.369	0.967	2.491	0.534	0.623	0.690
8	LReLU	$\mathcal{L}_{\text{DISN,L1}}$	8.364	0.747	2.466	0.245	0.608	0.665
9	LReLU*	$\mathcal{L}_{\text{DISN,TSDF,L1}}$	9.844	0.574	2.594	0.351	0.623	0.702
10	LReLU	$\mathcal{L}_{\text{TSDF,L1+sim}}$	8.856	3.959	2.453	0.977	**0.711**	0.763
11	LReLU	$\mathcal{L}_{\text{DISN,L1+sim}}$	11.744	0.473	3.943	0.213	0.623	0.772
12	**LReLU***	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	**4.638**	**0.378**	**1.881**	**0.127**	0.670	**0.791**
13	SIREN 3D	$\mathcal{L}_{\text{L1+sim}}$	0.138	0.067	0.126	0.015	0.779	0.838
14	**SIREN 3D***	$\mathcal{L}_{\text{TSDF,L1+sim}}$	3.296	**0.058**	1.404	**0.014**	0.771	0.846
15	SIREN 3D	$\mathcal{L}_{\text{DISN,L1+sim}}$	1.470	0.224	0.615	0.117	0.739	0.821
16	SIREN 3D	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	0.256	0.087	0.245	0.026	0.774	0.842
17	SIREN 6D	$\mathcal{L}_{\text{L1+sim}}$	3.997	0.312	1.693	0.208	0.676	0.809
18	**SIREN 6D***	$\mathcal{L}_{\text{TSDF,L1+sim}}$	**0.103**	0.065	**0.064**	0.016	**0.790**	**0.851**
19	SIREN 6D	$\mathcal{L}_{\text{DISN,L1+sim}}$	0.891	0.103	0.616	0.030	0.755	0.827
20	SIREN 6D	$\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$	2.519	0.121	1.439	0.064	0.728	0.812

The LReLU MLPs without positional encoding outperformed the ones with it. The SIRENs performed better than all of the LReLU MLPs. Many of the SIRENs (#14, #17 and #20) have mean scores that suffers from outliers. We attribute this to how volatile the SIRENs behaved during training. If we had terminated the training just an epoch earlier then we would likely see radically different mean scores. It is likely that SIRENs may benefit from a scheduled learning rate that decrease towards the end of training. Between the 3D and 6D SIRENs there were no significant difference in performance, aside from the aforementioned random outliers. This shows how well the periodic activation of SIRENs deal with Euler angles discontinuities. Even so, the properties of 6D rotation vectors are more beneficial for shape completion.

The SDF MSE and PSNR scores indicated that LReLU MLPs trained with SDF gradient supervision would perform worse than those trained without. This has now been flipped on it head, as we find just the opposite to be true! As such we conclude that SDF MSE and TSDF scores do not accurately reflect the final reconstruction quality. The training metric that best hints at the final shape reconstruction performance appear to be the mean SDF gradient similarity. This metric alone however does not account for where the 0-level set is.

Going by median performance, we determine that the LReLU MLPs performed best with $\mathcal{L}_{\text{DISN,TSDF,L1+sim}}$, and that the SIRENs performed best when trained with $\mathcal{L}_{\text{TSDF,L1+sim}}$.

SIRENs clearly came out on top between the two. We mark the median winner networks with a * and examine primarily these networks going forward, for the sake of brevity. We further report in table 6.3 F-scores for the median winners, along with the best SDF MSE and PSNR scoring SIRENs.

TABLE 6.3: The mean F_1-score defined in chapter 5.7 (higher is better), for varying thresholds as a % of reconstruction volume side length. We include networks from table 6.2 marked *, as well as the best TSDF MSE and PSNR scoring SIRENs. **Bold** highlights the best scores in each group. ∇ indicate the network was supervised with $\mathcal{L}_{\text{sim}}$.

#	Type	0.1%	0.2%	0.3%	0.5%	0.7%	1%	3%	5%	10%	20%
2	LReLU-PE*	0.016	0.077	0.166	0.331	0.437	0.53	0.74	0.81	0.90	0.98
5	LReLU-PE ∇*	0.013	0.058	0.126	0.265	0.373	0.50	0.78	0.83	0.89	0.96
9	LReLU*	0.026	0.135	0.292	0.534	0.647	0.73	0.87	0.91	0.96	**1.00**
12	LReLU ∇*	**0.028**	**0.138**	**0.295**	**0.564**	**0.708**	**0.81**	**0.93**	**0.95**	**0.98**	**1.00**
13	SIREN 3D ∇	0.047	0.246	0.500	0.821	0.922	0.97	**1.00**	**1.00**	**1.00**	**1.00**
14	SIREN 3D ∇*	**0.063**	**0.290**	**0.561**	**0.872**	0.939	0.96	0.97	0.98	0.99	**1.00**
16	SIREN 3D ∇	0.050	0.218	0.432	0.777	0.918	0.96	0.99	**1.00**	**1.00**	**1.00**
17	SIREN 6D ∇	0.018	0.107	0.258	0.572	0.732	0.82	0.94	0.97	0.99	**1.00**
18	SIREN 6D ∇*	0.047	0.241	0.501	0.848	**0.952**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**
20	SIREN 6D ∇	0.032	0.161	0.357	0.711	0.849	0.91	0.96	0.98	0.99	**1.00**

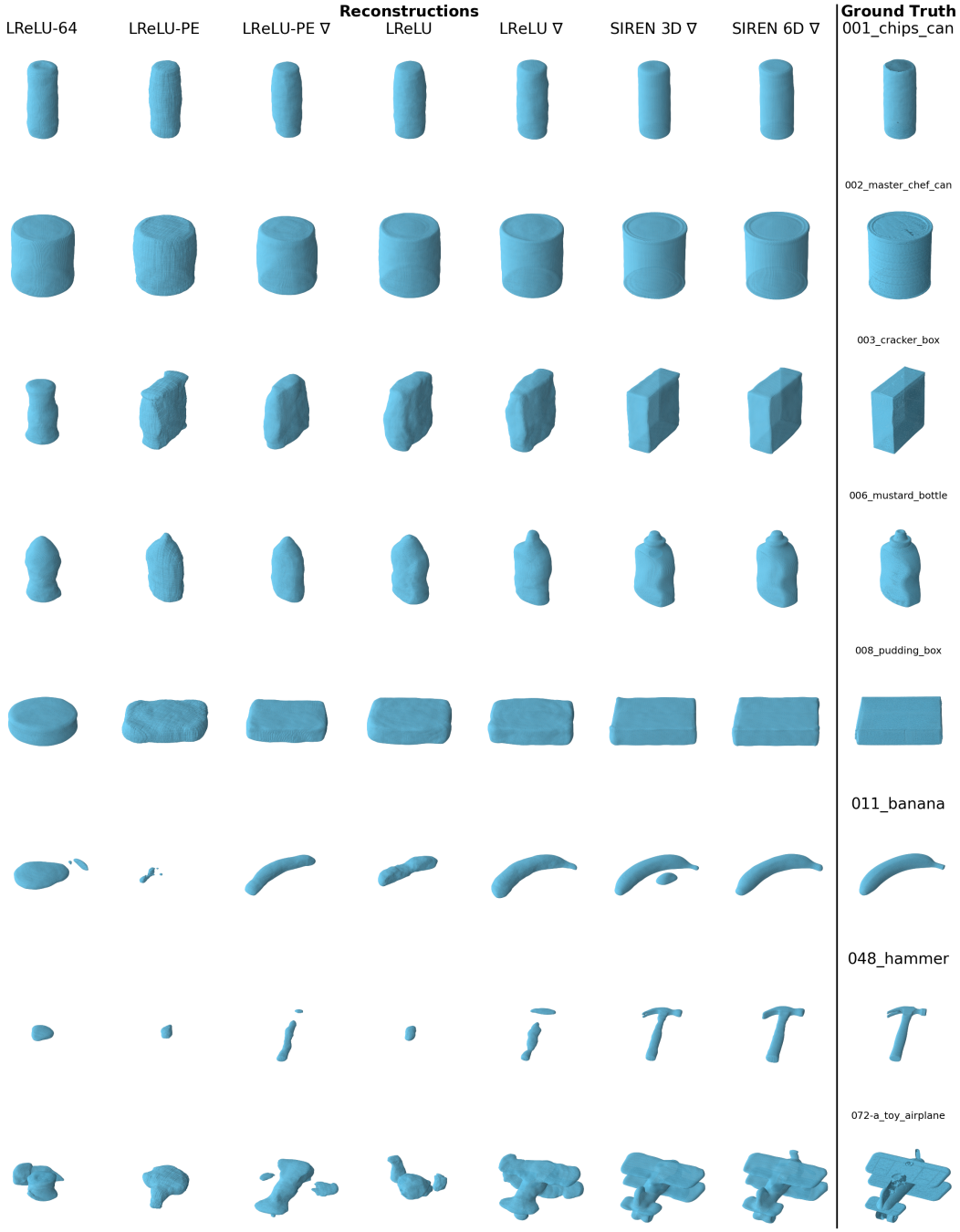
These F-scores show how $\mathcal{L}_{\text{sim}}$ improves the reconstruction quality for LReLU, while making LReLU-PE perform worse. Xu et al. (2019) measure a 0.943 F-score at 5% with their dual-network ReLU architecture. This score is comparable to network #12, which is smaller than theirs and embeds orientation in addition to the shape. Network #14 and #18 meanwhile reach the same score at 0.7%. To put the 0.7% threshold in perspective: it is on average a bit over 2 millimeter for the YCB objects. 5% is near 15 millimeters.

We render in figure 6.7 a handful of shapes reconstructed by the 6 networks marked with a * in table 6.2. The simpler shapes look good in general. The LReLU MLPs seem to struggle with the more intricate shapes, and fails to reconstruct most of the tools and Lego pieces. The SIREN reconstructions on the other hand are all accurate and look great.

LReLU-PE: The positional encoding (PE) reproduced noisy surfaces with substantial undulations or ripples, likely resulting from the high-frequency encoding. The ripples disappeared when supervised with $\mathcal{L}_{\text{sim}}$, but the accuracy suffered as a result. Due to their subpar performance on all metrics we set LReLU-PE aside and focus primarily on LReLU and SIREN going forward.

LReLU: Notice how the well the lip around the lid of the master chefs can was reproduced. We find that the networks learned rotationally invariant relations and patterns on the shapes, which we attribute to the orientation augmentations. These augmentations proved however to have been a bit harsh, as most of the reconstructions have a rounded quality to them. The LReLU with only 64 shape features showcased further emphasize this effect, primarily producing lathed approximations. SDF gradient supervision ($\mathcal{L}_{\text{sim}}$) alleviated this issue, properly constraining surface normals near corners and edges. $\mathcal{L}_{\text{sim}}$ resulted in a well-formed banana and a recognizable airplane. None of the LReLUs managed to to reconstruct the hammer or any of the other hand tools.

SIREN: One of the outlier errors tanking the mean scores for network #14 in table 6.2 can be seen on its banana reconstruction. It was able to reconstruct the banana, but the surrounding volume seem to not have been sufficiently constrained. These random outliers issues are not limited to those trained with $\mathcal{L}_{\text{TSDF}}$, indicating that a wider truncation range (larger δ_1) would not solve this issue.



(A) 3D reconstructions from known latent vectors

FIGURE 6.7: A handful of YCB objects reconstructed by the 6 networks in table 6.2 and 6.3 tagged with a *, along with a smaller LReLU network with only 64 shape dimensions. We showcase the ground truth mesh alongside reconstructions from the learned latent vectors. PE denotes positional encoding, while ∇ indicates supervision with SDF gradients. The meshes were constructed with marching cubes in a 123^3 voxel grid. We note that the SIRENs are only half as the size of the LReLU MLPs, showcasing their superior efficiency. This figure does not showcase single-view completions.

6.3.1 Evaluation Metric Details

To compute the metrics we reconstructed a mesh with marching cubes for each object, then we sampled 32768 points uniformly along both the reconstructed and ground truth surface meshes. We marched in a 256^3 grid spanning [-1.05, 1.05] along the x, y and z axes. We opted to sample that many points, as the modest number of 2048 used by Xu et al. (2019) and Kleineberg, Fey, and Weichert (2020) resulted in relatively large distances between even ground truth meshes compared against themselves. The SIREN reconstructions strayed well into this noise floor, prompting for a larger sample population. EMD got too expensive to compute with this many points however, as the search for the optimal bijection did not scale well. Luckily this metric is in deep learning commonly approximated with various distributed approximation schemes. We opted to use Sinkhorn Divergence with a blur of 0, implemented in CUDA by Fey and Lenssen (2019).

The mesh cosine similarity metric (COS) proved too expensive to compute for the fine-grained meshes we marched, as there simply were too many polygons to search though. We tried speeding up the search by sectioning the faces by volume into R-trees, but this quickly consumed all the available memory. We instead simplified the metric by reusing the surface points sampled earlier for the CD, EMD and F_1 metrics: for each ground truth point we located the nearest predicted point and compared those. This works because as we kept track of the face indices used to sample every surface point. To derive the normal vector $\hat{\mathbf{n}}$ of a sampled point we converted the point to three barycentric coordinates b_1, b_2, b_3 between the three face vertices, and used them to interpolate between the corresponding three vertex normals:

$$\hat{\mathbf{n}}_{\text{point}} = \frac{\hat{\mathbf{n}_1} b_1 + \hat{\mathbf{n}_2} b_2 + \hat{\mathbf{n}_3} b_3}{\|\hat{\mathbf{n}_1} b_1 + \hat{\mathbf{n}_2} b_2 + \hat{\mathbf{n}_3} b_3\|_2} \tag{6.1}$$

6.4 Examination of the Latent Space of Shapes

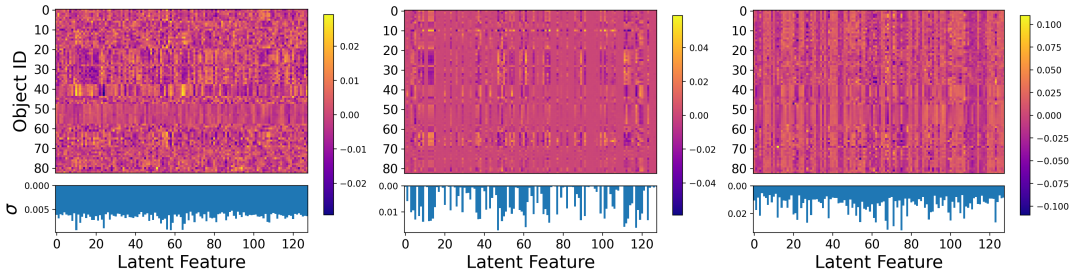
Here we examine how the different networks constructed their latent spaces for the training shapes enumerated in supplementary table A.2. We largely observed the LReLU MLPs converge on similar looking latent spaces. The inclusion of SDF gradient supervision or positional encoding made here little to no difference. The SIRENs also converged on similar latent spaces, but distinct from LReLU.

6.4.1 Latent Space Saturation

We present in figure 6.8 a height map of all the $\mathbf{z}_{\text{shape}}$ codes learned by three networks: #12 and #18 in table 6.2, along with a SIREN trained with a low $\mathcal{L}_{\text{codereg}}$ presented here to illustrate a trait. We additionally report the per-feature standard deviation. These visualizations show us whether the latent spaces are saturated or not: columns with low variation are features the networks have not assigned any meaning to, and have as a result been clamped down to 0 by $\mathcal{L}_{\text{codereg}}$.

LReLU MLPs with 128 feature wide latent vectors got packed with information spread in a uniform fashion (fig. 6.8a). This is the distribution we were aiming for in equation 5.6, as shown in supplementary figure A.1a. When we tried training LReLU MLPs with 256 shape features (as proposed by Park et al. (2019)) we found that most of the features went unused. Any number lower than 128 resulted in an increased shape reconstruction error. This indicates that 128 features is a sweet-spot for LReLU over our dataset, and that they are unable to embed more shapes.

SIRENs (network #18) left many of the features unused (fig. 6.8b), and show signs of having the capacity to embed many more shapes. This hints at their ability to compress the training shapes down into fewer features than the LReLU MLPs were able to. We attribute this to how ReLU-based MLPs are more biased towards learning low-frequency components, whereas SIRENs work with a wider frequency spectrum as tuned by ω_0. The second SIREN we showcased



(A) LReLU (#12) (B) SIREN (#18) (C) Relaxed SIREN

FIGURE 6.8: Raw known $\mathbf{z}_{\text{shape}}$ codes along with the standard deviation of each feature, learned by network #12 and #18 in table 6.2, and a third SIREN. The two first networks are representative for most LReLU MLPs and SIRENs. (A) and (B) trained with $0.04^2\mathcal{L}_{\text{codereg}}$, while (C) only used 0.01^2. Observe how all features in (A) vary uniformly, while a sizeable number in (B) go unused. SIRENs produce at times stray features not seen in LReLU, visible here as bright or dark spots. We attribute these to the periodicity of SIRENs, believing they have nudged themselves in a neighboring phase. This appears to have happened to a whole object (row) in (B): the banana. Figure 6.9 explore these latent vectors in further detail. Supplementary table A.2 map the objects IDs.

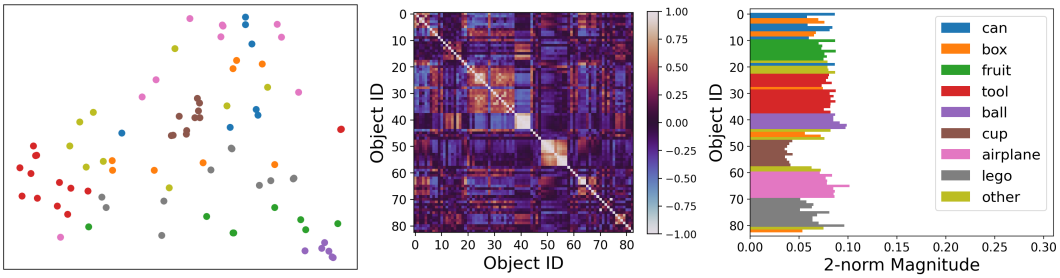
(fig. 6.8c) used more shape features, making each of them alone have a smaller impact on the shape. Its height map reveals prominent vertical lines, indicating that its latent space is not zero-centered. These characteristics came as a result of it not being constrained by $\mathcal{L}_{\text{codereg}}$ to the same degree.

6.4.2 Knowledge Discovery

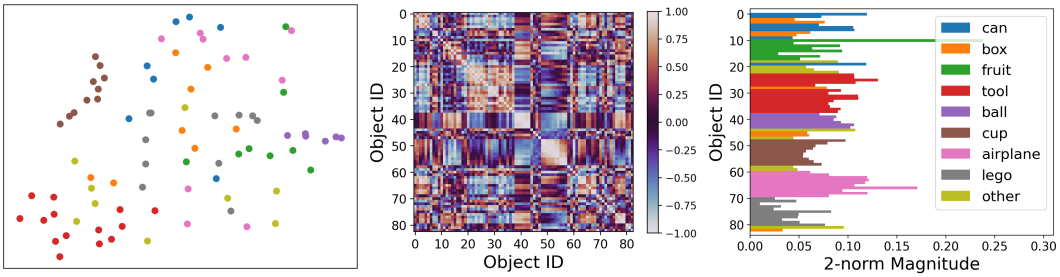
We explore these three sets of learned shape vectors in greater detail in figure 6.9. In it we project the high-dimensional latent vectors down onto a 2D plane using T-distributed Stochastic Neighbor Embedding (t-SNE). An inherent structure may emerge if we color the projected vectors by their class or category. We additionally compute a cosine similarity matrix between all vector pairs to see how they correlate with each other and whether the network learned an orthogonal set of codes or not. If the matrix resembles the identity then the codes are all orthogonal, indicating that the network likely do not generalize well to unseen shapes.

Starting with LReLU, we observe that the objects in the `ball` and `cup` classes have each been grouped together closely. This is apparent on both the t-SNE projection and in the similarity matrix. Both graphs show that some of the `fruits` have been found similar to the `ball` class as well. The `cans` and `boxes` are somewhat related to each other, but have become spread far apart across the latent space. We attribute this to how poorly aligned the training shapes are to one another, and how few instances of each class there are. The `tool` class, a class exhibiting a lot of variation due to the specialized nature of tools, has created a plateau in the similarity matrix. This indicates that the network has found shared features in their design, likely their lower volume or perhaps their slim and elongated shape.

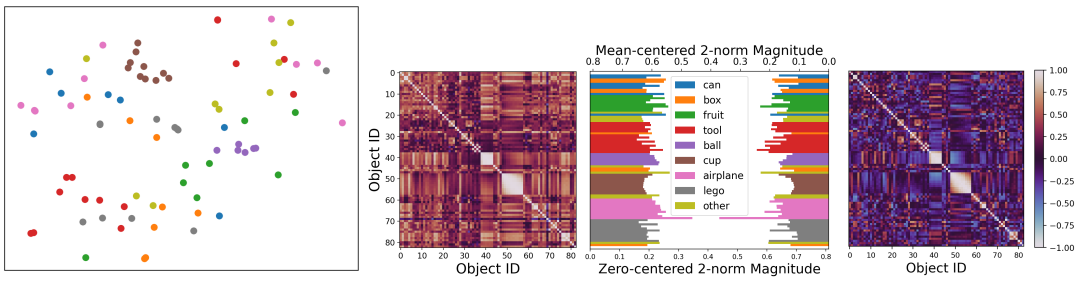
The SIRENs too managed to cluster and draw meaningful connections between similar objects. This is apparent in the t-SNE projection which indicates that the SIREN managed to group the object classes much like how the LReLU MLPs were able to. The similarity matrix feature many of the same clusters as for LReLU, but the amount of correlation is here a lot more prominent. This indicates that SIRENs have different inductive biases during shape completion. The second SIREN was also able to group the objects quite well. If however we assume its distribution is instead centered around the geometric mean as opposed to the origin, then it produces a similarity matrix closely resembling the identity (this is not the case for sufficiently



(A) LReLU (#12) - t-SNE (B) Cosine similarity (C) Vector magnitudes



(D) SIREN (#18) - t-SNE (E) Cosine similarity (F) Vector magnitudes



(G) "Relaxed SIREN" - t-SNE (H) Zero-centered (I) Magnitudes (J) Mean-centered

FIGURE 6.9: Three visualizations for the raw latent codes shown in figure 6.8. Each row explores a separate network. The t-SNE scatter plots illustrate the layout of and relation between the classes and how they cluster, distribute, and interlink in latent space. The similarity matrices show how similar each latent vector pair are, assuming a zero-centered spherical distribution: 0 indicates orthogonality while non-zero values are correlated: positive scores are similar while negative are dissimilar. As the "relaxed" SIREN is likely not zero-centered, we additionally report a similarity matrix centered around the geometric mean vector, revealing a near-orthogonal set. However, (G) still indicates the object classes cluster as in (A) and (D). Finally we present the Euclidean magnitude of each known shape vector, colored by object class. These magnitudes proved instrumental in tuning single-view vector optimization. Supplementary table A.2 map the object IDs.

constrained SIRENs). The reduced $\mathcal{L}_{\text{codereg}}$ seem to in this case to have allowed that SIREN to overfit on the training shapes, making it likely not able to generalize to new unseen shapes. This is thankfully not the case for any of the SIRENs showcased in table 6.2 which trained with sufficient $\mathcal{L}_{\text{codereg}}$.

6.4.3 Latent Space Smoothness

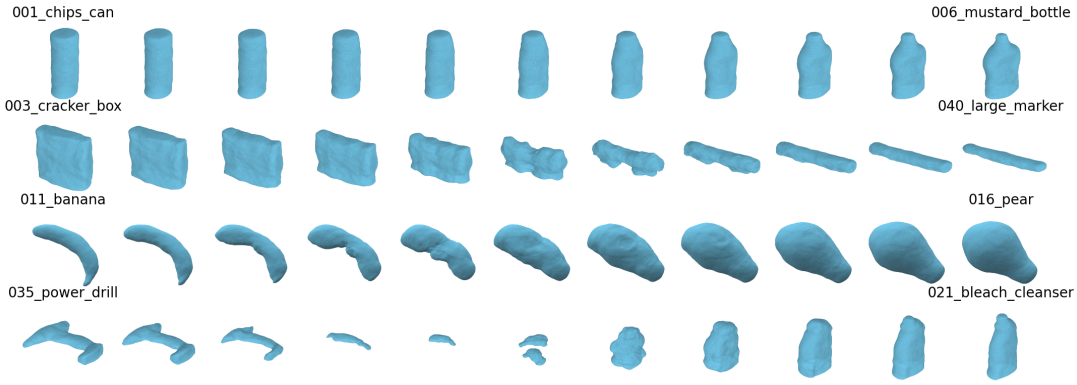
To be able to traverse the latent space via optimization towards a target shape, it is important that the latent space is continuous, that the shapes transition smoothly between one another, and that most latent vectors produce meaningful shapes. Otherwise it is less likely that a search for conforming shapes can perform well.

The SIREN latent space distribution is not spherical like the prior assumed in section 5.4.2. The shape features extracted by SIRENs are a lot more dependent with one another than the ones regressed by the LReLU MLPs. This can be seen in supplementary figure A.1, not included here for brevity. The figure shows that there is a considerable amount of correlation between many of the shape features, a lot more than for LReLU MLPs. Luckily the Bayesian inference analysis by Park et al. (2019) still hold water under these circumstances, indicating that shape completion should work. However, as the SIREN latent spaces are not spherical they are more likely to feature holes, sharp edges, or cluster shapes into separate neighborhoods. Noisy data may bridge or "short-circuit" parts of the manifold that would otherwise in t-SNE projections be well-separated. As such we can not conclude from the t-SNE plots alone whether the SIREN latent spaces are uniform.

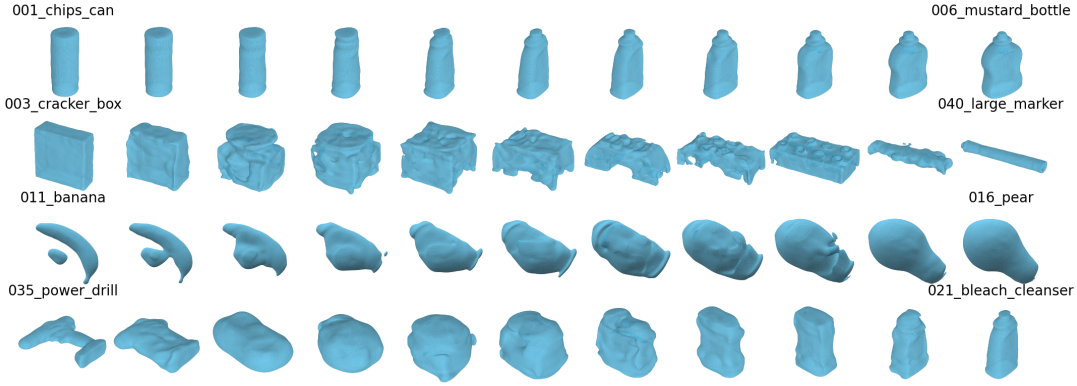
We evaluate uniformity by traversing through latent space. In figure 6.10 we linearly interpolate between known shape vectors and render in-between reconstructions. Observe how uniform the spherical LReLU space is. It only has trouble when transitioning from a drill lying on its side to an upright bleach bottle, a transition even humans find difficult to imagine. The shapes blend smoothly from one to the other, indicating that the reconstructions are very pliable. SIRENs on the other hand pack their shapes closer together. When SIREN #14 traversed from the cracker box to the marker it passed by the colored wood block, then the foam brick, then the `073-e` Lego piece, then the `073-c` piece, before reaching the marker. Some of the more distant shapes seem to be separated into distinct neighborhoods, causing the interpolation to leave the shape manifold and reconstruct abnormal shapes. This is apparent in the interpolation from the drill to the bleach cleanser by SIREN #18, indicating that there are holes and edges in SIREN spaces. However, SIRENs interpolate smoothly when staying inside of well formed neighborhoods, arguable better than what LReLU does. This is demonstrated by the SIRENs showcased in figure 6.10, which both passed by the bleach bottle when interpolating from the chips can to the mustard bottle, staying more rigid and "true" to the training shapes. Linear interpolation alone can not tell the full tale however, as it performs better on convex manifolds than for concave ones.

We conclude that the LReLU MLPs produced the most *uniform* latent spaces. However, LReLU seem to reconstruct shapes as the sum of many simple and independent features, whereas SIREN regress complex spaces of less pliable shapes that stay more true to the YCB objects. We attribute this to how ReLU-based MLPs have a bias to learn low-frequency patterns, while SIRENs learn more complex patterns as tuned by ω_0.

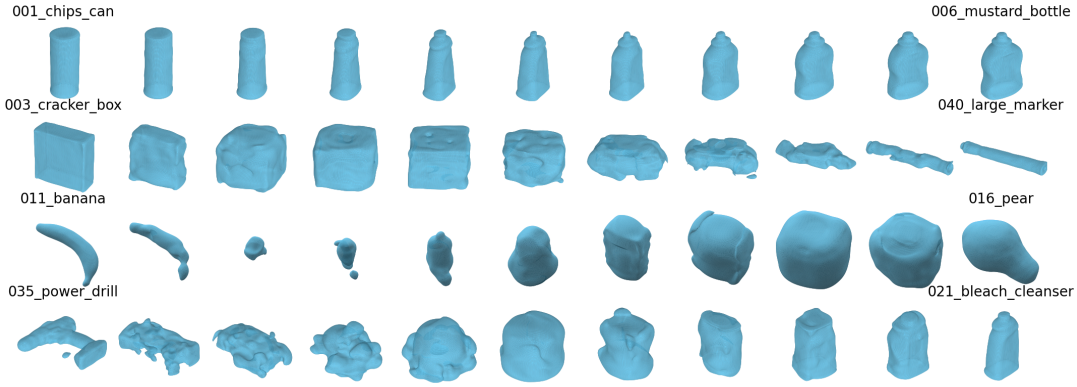
Altogether we find that both LReLU and SIREN is able to generalize structural commonalities between the training shapes. The LReLU MLPs appear to have become saturated with even the few shapes provided by YCB, indicating they won't scale well to richer datasets. The SIRENs showed early signs of overfitting when we interpolated through latent space, indicating a lack of training data. Still the SIRENs feature an impressive ability to extract and compress high-level relations from the shapes, which we attribute to the impressive ability of auto-decoders to generalize.



(A) LReLU (network #12)



(B) SIREN (network #14)



(C) SIREN (network #18)

FIGURE 6.10: Linear interpolations in latent space between pairs of known shapes, by network #12, #14 and #18 in table 6.2. Inspecting the appearance of in-between reconstructions may aid our understanding of the latent space distribution. LReLU latent spaces appears highly uniform, although it seems to struggle with poorly aligned shapes. The SIRENs behave well between closely related shapes, but do at time leave the manifold. We note that the banana in network #18 is a major outlier, apparent in fig. 6.9f. The meshes were marched in a 128^3 grid in the spatial range $[-1.1, 1.1]$ with marching cubes. For each (a, b) pair of objects we mix the codes $\mathbf{z}$ from left to right as $(1 - c)\mathbf{z}_a + c\mathbf{z}_b$ for $c \in \{\frac{i}{10}\}_{i=0}^{10}$.

6.5 Single-View Shape Completion

We perform shape completion by searching through latent space with stochastic gradient decent. With fixed network weights θ we search for a latent vector $\hat{\mathbf{z}} = (\hat{\mathbf{z}}_{\text{shape}}, \hat{\mathbf{z}}_{\text{pose}})$ that minimize the loss across the single-view observation. We again used Adam optimization, this time with a scheduled learning rate η as a function of the number of optimization steps taken:

$$\eta = \max\left(\eta_{\text{start}} \cdot c^{\frac{\text{current step}}{\text{total steps}}}, \eta_{\text{min}}\right) \tag{6.2}$$

making it so the learning rate start at η_{start} and work its way down to $c \cdot \eta_{\text{start}}$, optionally clamped up to η_{min}.

This works for $\hat{\mathbf{z}}_{\text{shape}}$, but this learning rate is insufficient for $\hat{\mathbf{z}}_{\text{pose}}$ due to its much larger magnitude. To address this we decompose $\hat{\mathbf{z}}$ into its shape and three orientation components $(\hat{\mathbf{z}}_{\text{shape}}, \hat{\mathbf{z}}_{\mathbf{t}}, \hat{\mathbf{z}}_s, \hat{\mathbf{z}}_{\text{rot}})$, and optimize each with separate learning rates. Between each optimization step we normalize the 6D rotation vector $\hat{\mathbf{z}}_{\text{rot}}$.

Tuning the learning rates proved quite tricky, as there were non-trivial amounts of variation in how the different networks responded. Consequently we can not compare all the networks in table 6.2 in a fair quantitative manner. We investigate different optimization strategies using synthetic scans in the two following subsections, followed by a subsection where we perform shape completion on real-world data. To this end, we introduce the following terms:

Global centroid: A centroid is the geometric mean between points, i.e. a axis-wise mean. When discussing LReLU MLPs we fix the global centroid to the origin.[7] For SIRENs we compute the global centroid from the known codes for all objects in Ω as:

$$\bar{\mathbf{z}}_{\text{shape}} = \frac{1}{|\Omega|} \sum_{i \in \Omega} \mathbf{z}_i^{\text{shape}} \tag{6.3}$$

Mean magnitude: While optimizing a latent vector we track how far away from the global centroid it is, as there is an expected magnitude we aim for. (This is apparent by the low variance featured in fig. 6.9c.) We found that $\hat{\mathbf{z}}_{\text{shape}}$ breaks loose from the shape manifold and "shoots away" from the global centroid if the learning rate is too aggressive. As such we aim for a mean magnitude defined as:

$$\mu_{\|\mathbf{z}\|} = \frac{1}{|\Omega|} \sum_{i \in \Omega} \|\mathbf{z}_i^{\text{shape}} - \bar{\mathbf{z}}_{\text{shape}}\|_2 \tag{6.4}$$

Class centroid: The centroid between all the known latent vectors for a class of object. The classes and its members are enumerated in supplementary table A.2. We compute the centroid of a `class` as:

$$\bar{\mathbf{z}}_{\text{class}}^{\text{shape}} = \frac{1}{|\Omega_{\text{class}}|} \sum_{i \in \Omega_{\text{class}}} \mathbf{z}_i^{\text{shape}} \tag{6.5}$$

Class variance/deviation: We define the deviation σ_{class} for a `class` of objects as the standard deviation of the Euclidean distance from the class centroid to each of class member:

$$\sigma_{\text{class}} = \sqrt{\frac{1}{|\Omega_{\text{class}}| - 1} \sum_{i \in \Omega_{\text{class}}} \|\mathbf{z}_i^{\text{shape}} - \bar{\mathbf{z}}_{\text{class}}^{\text{shape}}\|_2^2} \tag{6.6}$$

The class deviation loosely indicates the volume in latent space that the class members cover. The variance is the class deviation squared.

[7] We verified on all trained LReLU MLPs that the geometric mean is near-zero.

6.5.1 A Naive Approach

In this section we search the latent space to find shapes that simply conform to the single-view data. The data is scaled and centered such that near-surface samples fit within the reconstruction volume. We initialize $\hat{\mathbf{z}}_{\text{shape}}$ near the global centroid, set its translation $\hat{\mathbf{z}}_{\mathbf{t}}$ to $[0,0,0]^T$, its scale $\hat{\mathbf{z}}_s$ to 1, and draw a random rotation uniformly within $SO(3)$ for $\hat{\mathbf{z}}_{\text{rot}}$.

Gentle search: From this starting position we search with a "gentle" learning rate in the ballpark of $\eta_{\text{start}} = 3{\times}10^{-4}$, $c = 2 \times 10^{-1}$, and $\eta_{\text{min}} = 1{\times}10^{-4}$ for 600 optimization steps. We determined these rates by observing how close $\|\hat{\mathbf{z}}_{\text{shape}} - \bar{\mathbf{z}}_{\text{shape}}\|_2$ got to the mean magnitude $\mu_{\|\mathbf{z}\|}$. The optimal learning rates varied from network to network.

To determine the orientation learning rates we rendered animations of the path traversed while searching, marching a mesh between every optimization step. Armed with these animations we could spot how different learning rates were either insufficient or overshot their target. We found it best to optimize the translation and scaling with a 50η learning rate, 6D rotation vectors with 500η, and Euler angles with 50η. To keep the translation from "shooting off" we constrained it by adding $0.001\|\hat{\mathbf{z}}_{\mathbf{t}}\|_2^2$ to the loss, still allowing it to move somewhat freely within the bounds of the reconstruction volume.

We discovered that shape completion required $\mathcal{L}_{\text{codereg}}$, as $\hat{\mathbf{z}}_{\text{shape}}$ otherwise expands past the manifold. This is a property of how we trained the networks, as they had to learn an outward attraction to counterbalance the code magnitude cost.

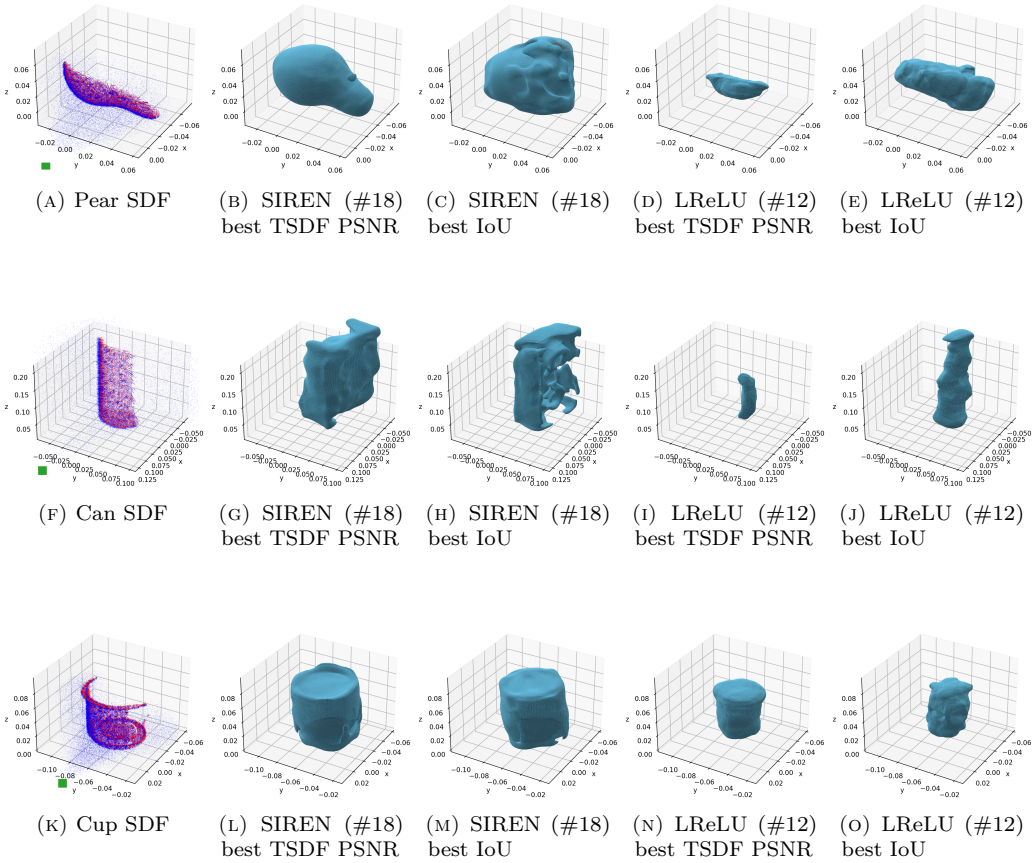
The randomly drawn initial orientation caused at times the shape completion to fail. To combat this we optimized a batch of multiple vectors, each initialized with different orientations. We then select a winner from this batch scored by some metric. Although we have established that neither the TSDF MSE nor the PSNR metrics are optimal, we still determine the batch winner with TSDF PSNR. This worked great for SIREN, but not for LReLU as it resulted in picking "shrunken" shapes. We experimented briefly with alternative metrics, and found one that worked better: Intersection over Union (IoU), which scores the similarity between two sets as $\frac{|A \cap B|}{|A \cup B|}$. We defined A as the set of points with negative ground truth SDF values, and B as the points predicted negative. With this IoU only scores the amount of correctly conforming near-surface points instead of the quality of the whole field.

We showcase some naive single-view shape completions via "gentle" search in figure 6.11. The SIRENs were able to draw from its shape embeddings and conform to the observation data quite well, whereas the LReLU MLPs had trouble searching efficiently. We frequently observed that many of the shape features barely moved during optimization, while the SIRENs optimized all features with ease. We attribute this to how ReLU-based neurons become inactive, a problem we hoped to alleviate when switching to leaky ReLU over standard ReLU.

The LReLU MLPs do not seem to guarantee that all latent codes produce well-formed fields (where $\|\nabla_{\mathbf{x}}\text{SDF}\|_2$ is constant). They construct their fields as the sum of many simple latent parameters, which makes it possible to combine them ways that form into uneven distance fields. We believe this is the reason the TSDF PSNR metric performs poorly for LReLU. We had intended to constrain our shape completion with $\mathcal{L}_{\text{norm}}$ to fix this, but could not do so as it was broken. The SIRENs construct more well-formed distance fields from its "high-level" shape embeddings, and were for this reason not as affected this issue.

Aggressive search: We investigated, in an attempt to make LReLU MLPs perform better, a more "aggressive" search strategy. We limited the search volume to the surface of a hypersphere whose radius equal the mean magnitude, paired with a *much* higher learning rate. This we achieved by normalizing the latent vector $\hat{\mathbf{z}}$ into $\hat{\mathbf{z}}'$ between each optimization step:

$$\hat{\mathbf{z}}'_{\text{shape}} = \hat{\mathbf{z}}_{\text{shape}} \frac{\mu_{\|\mathbf{z}\|}}{\|\hat{\mathbf{z}}_{\text{shape}}\|_2} = \hat{\mathbf{z}}_{\text{shape}} \frac{1}{\|\hat{\mathbf{z}}_{\text{shape}}\|_2} \frac{\sum_{i \in \Omega} \|\mathbf{z}_i^{\text{shape}}\|_2}{|\Omega|} \tag{6.7}$$

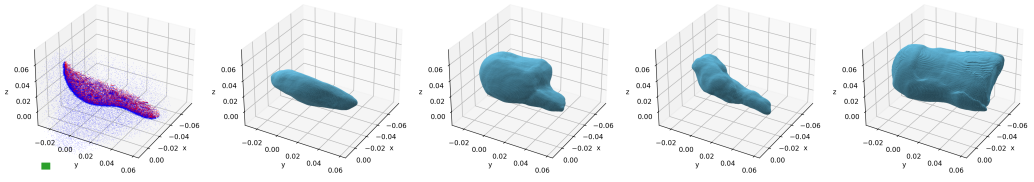


(A) Pear SDF (B) SIREN (#18) best TSDF PSNR (C) SIREN (#18) best IoU (D) LReLU (#12) best TSDF PSNR (E) LReLU (#12) best IoU

(F) Can SDF (G) SIREN (#18) best TSDF PSNR (H) SIREN (#18) best IoU (I) LReLU (#12) best TSDF PSNR (J) LReLU (#12) best IoU

(K) Cup SDF (L) SIREN (#18) best TSDF PSNR (M) SIREN (#18) best IoU (N) LReLU (#12) best TSDF PSNR (O) LReLU (#12) best IoU

FIGURE 6.11: Naive single-view shape completions with "gentle" search from the global centroid on synthetic data. Leftmost column is the single-view SDF target, with blue and red being positive and negative, and green being the camera position. LReLU spent five minutes searching 25 codes for 600 steps and scoring the winner, while SIREN spent one minute searching ten. We present here winning shapes determined with both TSDF PSNR and IoU. LReLU struggled to conform to the single-view data, while SIREN performed well once it found a nearby shape: It found the pear correctly. For the chips can it matched the side of a lego piece. For the cup it likely matched the master chefs can. All ground truth shapes are rendered in supplementary figure A.2. Network numbers refer to rows in table 6.2.



(A) Pear SDF (B) LReLU (#9) best TSDF PSNR (C) LReLU (#9) best IoU (D) LReLU (#12) best TSDF PSNR (E) LReLU (#12) best IoU

FIGURE 6.12: Single-view shape completions with "aggressive" search, constrained by equation 6.7. Here we show the two best completions out of 25 optimization attempts, determined by TSDF PSNR and IoU. It took five minutes to search 25 codes for 600 steps. Network numbers refer to rows in table 6.2.

We optimized with a learning rate of $\eta_{\text{start}} = 1{\times}10^{0}$, $c = 2.5 \times 10^{-3}$, and $\eta_{\text{min}} = 2{\times}10^{-4}$ for 600 optimization steps, only normalizing $\hat{\mathbf{z}}$ for the first 300. Although intense, it does not leave the manifold. With such an intense base learning rate we now optimized the translation, scale and Euler angles with a learning rate of 0.1η, and 6D rotations with 10η.

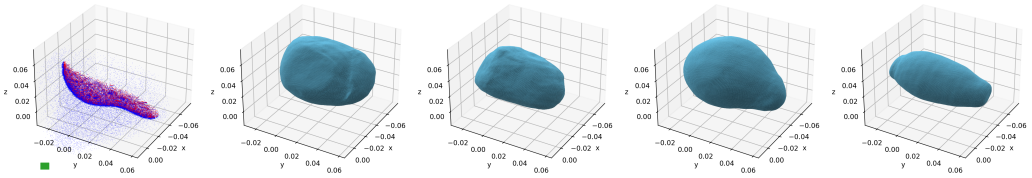
This "aggressive" search approach helped cover more of the LReLU latent space, but it still only conforms to the single-view observations to a limited degree. We present some LReLU MLP shape completion using this search approach on a pear in figure 6.12. It was able to find shapes that conform to the single-view observation, but these shapes can not be described as anything meaningful.

Figure 6.9c indicates that the latent spaces are not perfectly spherical like we assumed the prior distribution to be. There is a non-trivial amount of variation in magnitude between the different object classes. Seeing this we tried normalizing the shape codes to a *weighted* mean magnitude, weighted by how similar each known code $\{\mathbf{z}_i\}_{i\in\Omega}$ are to $\hat{\mathbf{z}}$:

$$\begin{aligned} \hat{\mathbf{z}}'_{\text{shape}} &= \hat{\mathbf{z}}_{\text{shape}} \frac{1}{\|\hat{\mathbf{z}}_{\text{shape}}\|_2} \frac{\sum_{i\in\Omega} \|\mathbf{z}_i^{\text{shape}}\|_2 \phi_i^n}{\sum_{i\in\Omega} \phi_i^n} \\ \phi_i &= \max(\langle \hat{\mathbf{z}}, \mathbf{z}_i \rangle, \epsilon) \end{aligned} \tag{6.8}$$

where $\langle \cdot, \cdot \rangle$ is the cosine similarity, $\| \cdot \|_2$ is the Euclidean norm, n is a "sharpness" factor, and $\epsilon = 1 \times 10^{-8}$ make the equation fall back to eq. 6.7 if no known code share any similarities. In supplementary figure A.5 we illustrate in 2D how the hypersurface covered by this equation conforms to the magnitude of known codes.

We present a few LReLU shape completions using this weighted clamping scheme in figure 6.13. Now the TSDF PSNR score again become the better metric, but it still performs worse than what the SIRENs did via simple "gentle" search. The "aggressive" search approach is not applicable to SIREN latent spaces.

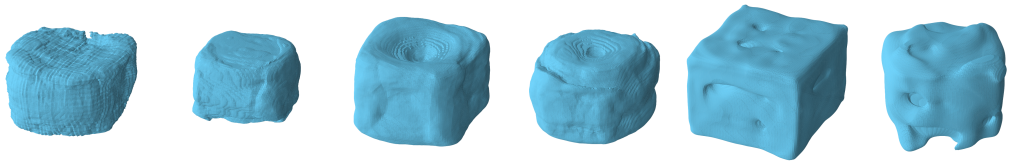


(A) Pear SDF (B) LReLU (#9) best TSDF PSNR (C) LReLU (#9) best IoU (D) LReLU (#12) best TSDF PSNR (E) LReLU (#12) best IoU

FIGURE 6.13: Single-view shape completions with "aggressive" search, constrained by equation 6.8 with $n = 10$. The shapes do not conform as well as in figure 6.12, but the reconstructed fields are now valid SDF fields, making PSNR the better judge. It took five minutes to search 25 codes for 600 steps. Network numbers refer to rows in table 6.2.

These two naive search strategies were able to find *somewhat* conforming shapes, but whether the far side is correct or not is a long shot. Lathed shapes showcased thus far are simple targets, but more complex shapes fail with this approach. We see two failure states: (1) Failure to find a conforming shape. (2) Failure to infer the unseen parts of the shape. LReLU MLPs struggles with both when searching naively, while SIRENs primarily struggle with the latter. To address the latter shortcoming we propose a class-aware approach in the following section.

6.5.2 A Class-Aware Approach

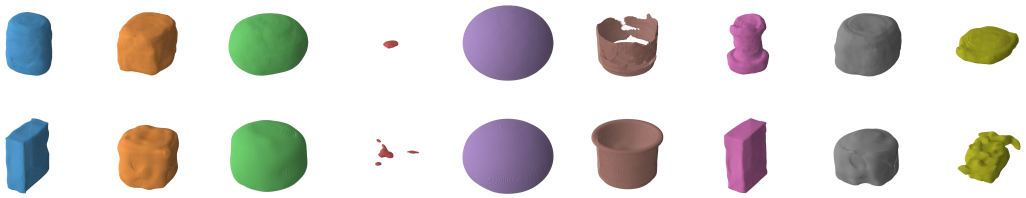


(A) #2 (B) #5 (C) #9 (D) #12 (E) #14 (F) #18

FIGURE 6.14: Shapes at the global centroid reconstructed by the networks marked * and numbered in table 6.2. These are the "starting shapes" for a naive search approach. They're different for each network, as they have not been constrained to a shape beneficial for single-view shape completion, other than what $\mathcal{L}_{\text{codereg}}$ managed to carve.

Naive search starts near the global centroid, with no knowledge about the underlying object other than what parts are visible. We show what the reconstructed global centroid looks like for 6 networks in figure 6.14. Some networks created a global centroid shape resembling a cross between a cube and a cup, while others produce no particularly meaningful shape at all. Training with $\mathcal{L}_{\text{codereg}}$ did make it so the path from the global centroid to each shape is a somewhat smooth one, but what path to take at all is still up in the air. How can a naive search tell the difference between a can and a mug, if the handle is not visible? For example, had the network in figure 6.11l been guided with information indicating that the target ought to have been hollow, then it could have made it so.

An object classifier can help. Our approach so far make no use of the single-view color data: we have only used the depth information to constrain the SDF field. A classifier network can from the color data infer which category of object we are shape completing. With this information it is possible to "shortcut" the search and make it less holistic. In figure 6.15 we render what the shape at each class centroid looks like, reconstructed with both a LReLU MLP and a SIREN. Not only are these starting points closer to their target *shape*, but also considerably closer in latent space.

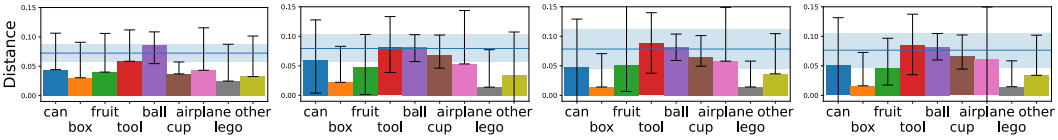


(A) can (B) box (C) fruit (D) tool (E) ball (F) cup (G) plane (H) lego (I) other

FIGURE 6.15: The shape at each class centroid. First row is a LReLU (#12) and the second is a SIREN (#18) from table 6.2. These are the "starting shapes" for class-aware search approaches. It differs for each network, but these are a lot more guided by the embedded shapes than the global centroids shown in figure 6.14 are. Note: the `airplane` class is in figure 6.16 shown to have a large error.

Our class-aware search strategy is based on the "gentle" search approach outlined in the previous section, but now with an *even* more "gentle" learning rate. We now draw the initial $\hat{\mathbf{z}}_{\text{shape}}$ near the class centroid instead of the global centroid. The initial translation $\hat{\mathbf{z}}_{\mathbf{t}}$ and scale $\hat{\mathbf{z}}_s$ are still set to $[0, 0, 0]^T$ and 1, and the rotation $\hat{\mathbf{z}}_{\text{rot}}$ is drawn at random.

To measure at how well such a class-aware approach might work, we present the magnitude of each class centroid along with the class deviations in figure 6.16. The magnitude of each class centroid indicate how much the class prediction bias the search away from the global centroid. Classes with a low variance is expected to traverse less through latent space than those with high variance. A good dataset should seek to maximize the magnitude of each class centroid and minimize the class variances. This can be achieved by properly aligning each object to minimize the distance between them, and by dividing them into more descriptive categories.



(A) LReLU (#12) (B) SIREN (#14) (C) SIREN (#18) (D) SIREN (#18), sans banana

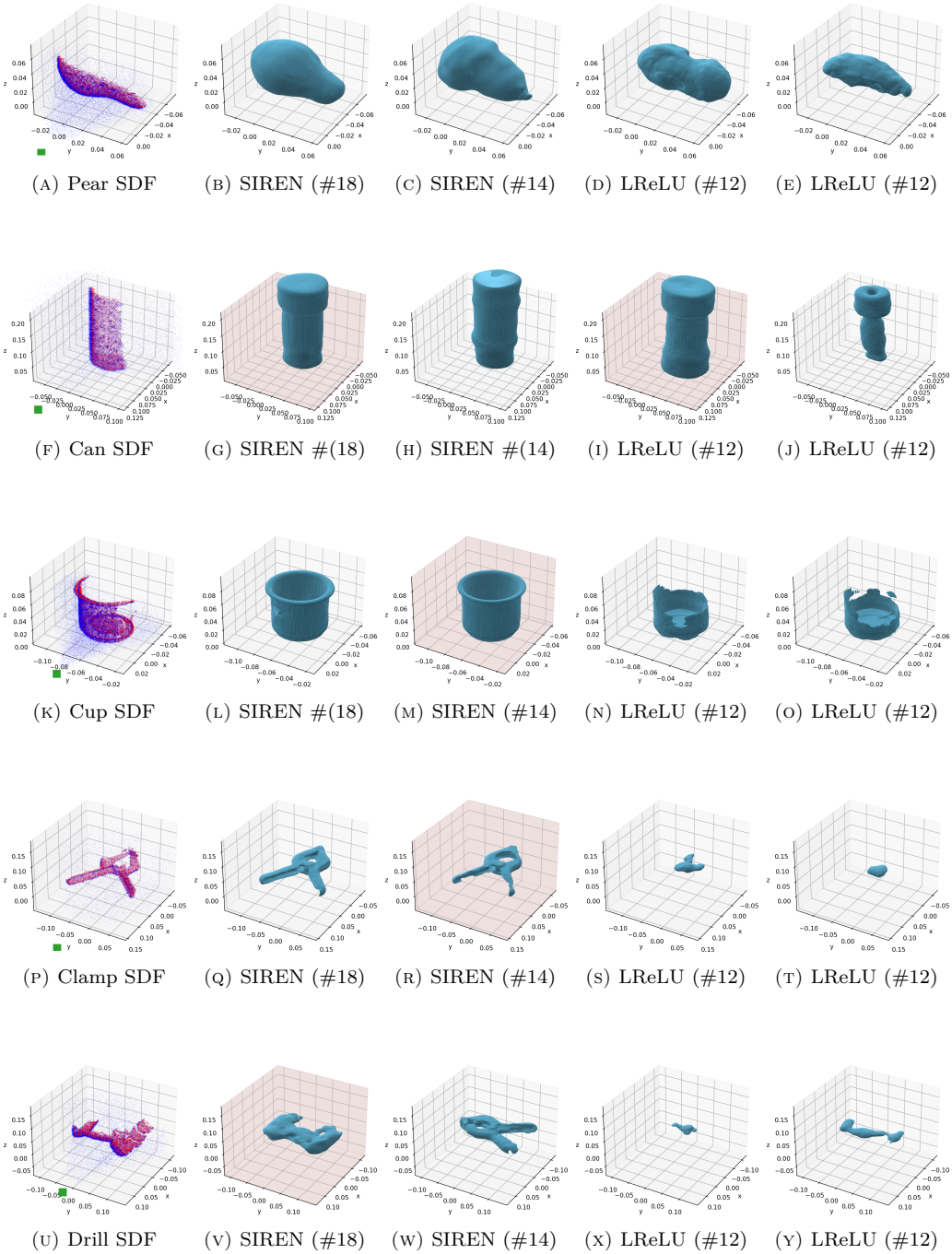
FIGURE 6.16: Bar plots of the Euclidean distance between the global centroid and the class centroids, for network #12, #14 and #18 in table 6.2. These plots hint at how well a classifier may aid shape completion for each object class. The error bars measure the class deviation, with the upper bound calculated from the codes further away from the global centroid than the class centroid, and the lower bound by those closer. The blue line and span measure the mean code magnitude and its standard deviation. In (D) we show the same network as in (C), with the outlier code for `011_banana` removed. Supplementary figure A.2 render all the training shapes in matching category colors.

We showcase some class-aware single-view shape completion in figure 6.17. In it we selected the best TSDF PSNR scoring shape our of 25 attempts for LReLU MLPs, and out of three attempts with SIRENs. Apparent is how much better the SIRENs performed, succeeding on many of the objects. In fig. 6.17w it failed to find the drill from the `tool` centroid, getting stuck in a local minimum where it fits a clamp to the observation data. If the SIREN had more attempts then it may have found a better scoring solution. We found SIRENs in general to succeed on many of the shapes, aside the more complex ones such as the cutlery seen in supplementary figure A.2. It performs well in general on the more "volumetric" objects.

We optimized the shapes using the same loss function the networks were trained with, excluding $\mathcal{L}_{\text{sim}}$. This proves that networks trained *with* SDF gradient supervision do not not require it at test time. We did not find $\mathcal{L}_{\text{sim}}$ to particularity benefit the shape completions at this stage, and chose to leave it out as its exclusion drastically reduces the computation time.

We visualize in figure 6.18 "animations" from searching through latent space for five of the shape completions shown in figure 6.17, with a play-by-play description of what we see. The LReLU MLPs in general seem to have too many simple shape parameters, and are as a consequence less able to optimize on a "higher" more meaningful level. They prefer to morph the current shape instead of reorienting it. SIRENs on the other hand are likely not able to perform as well on new unseen shapes. This issue may be remedied with a larger dataset of training shapes, with more variations of each object class, giving the SIRENs more priors to draw from.

A problem both networks have in common, is that the initial guess of a zero translation, a scale of 1 and a random rotation is more often than not wrong. Our method could be further aided by not only a classifier network inferring the object category, but also by a pose estimation network that predicts an initial pose.



(A) Pear SDF (B) SIREN (#18) (C) SIREN (#14) (D) LReLU (#12) (E) LReLU (#12)

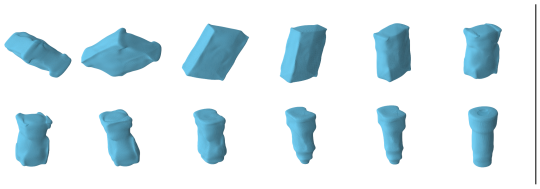
(F) Can SDF (G) SIREN #(18) (H) SIREN #(14) (I) LReLU (#12) (J) LReLU (#12)

(K) Cup SDF (L) SIREN #(18) (M) SIREN (#14) (N) LReLU (#12) (O) LReLU (#12)

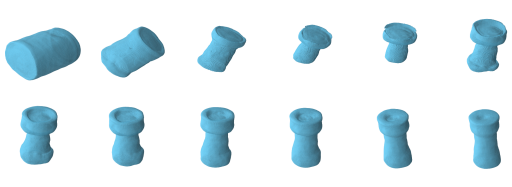
(P) Clamp SDF (Q) SIREN (#18) (R) SIREN (#14) (S) LReLU (#12) (T) LReLU (#12)

(U) Drill SDF (V) SIREN (#18) (W) SIREN (#14) (X) LReLU (#12) (Y) LReLU (#12)

FIGURE 6.17: Class-aware single-view shape completions with "gentle" search from the class centroid on synthetic data. Leftmost column is the single-view SDF target, with blue and red being positive and negative. We tested the LReLU network twice for each shape. LReLU spent five minutes searching 25 codes for 600 steps and scoring the winner with TSDF PSNR, while the SIRENs spent 18 seconds searching three. SIREN completed most of the shapes accurately. We explore in figure 6.18 the shape completions shown with red grids (G, I, M, R and V) in further detail. All ground truth shapes are rendered in supplementary figure A.2. Network numbers refer to rows in table 6.2.

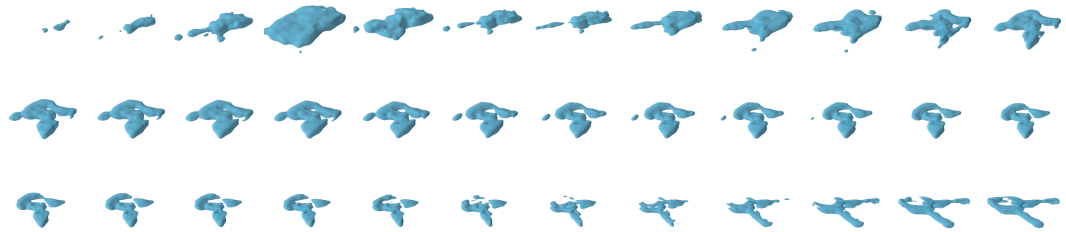


(A) SIREN (#18) chips can from can centroid

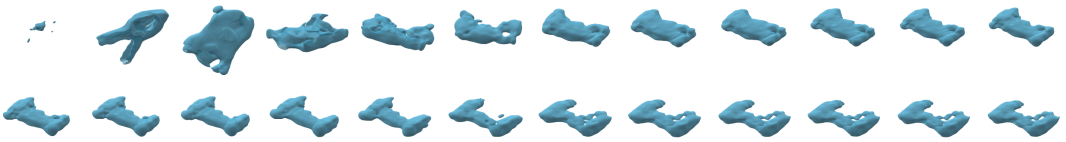


(B) LReLU (#12) chips can from can centroid

(C) SIREN (#14) cup from cup centroid



(D) SIREN (#14) clamp from tool centroid



(E) SIREN (#18) drill from tool centroid

FIGURE 6.18: "Animations" of intermediate shapes while searching through latent space for the shape completions in figs. 6.17g, 6.17i, 6.17m, 6.17r, 6.17v. (A-D) is from a different camera perspective, while (E) has a matching camera. These animations cover the first 200 out of 600 optimizations steps. Only changes to the shape and rotation are obvious, as the change in scale and translation is difficult to convey in a grid. (A) started at a box-like class centroid, but still managed to reorient and adapt its shape. (B) initially moved away from the `can` class towards one of the `airplane` parts, backtracking once it was oriented correctly. (C) started at a class centroid with a very low shape error, then drifted away while reorienting itself. Once the pose matched it solved the shape again. (D) initially matched with a clamp in the wrong orientation, but elected to change its shape to a different clamp instead of rotating the one it had already found. (There are clamps in two different orientations in our dataset, see fig A.2.) (E) started of initially matching a clamp. While reorienting the clamp it matched with the drill, but upside-down. From here it tried to "morph" that upside-down drill to the best of its ability.

6.5.3 Real-World Data and Occlusions

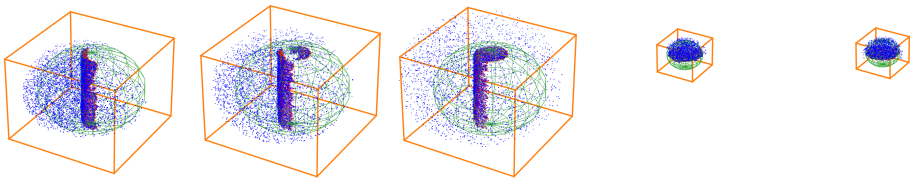
How well do our method perform on real-world sensor data? How well can it handle different occlusion states? To answer these questions we consider the three occlusion states Jalal and Singh (2012) find common in real-world data: (1) Self occlusion, where one part of the object occludes itself. (2) Inter-object occlusion, where two objects occlude each other. (3) Background occlusion, where a structure in the scene occludes the object. Assuming access to accurate object segmentation masks we can reduce (2) and (3) to simple *occlusion*.

To illustrate a shortcoming with our setup we explore two occlusion scenarios. One scenario occludes the observed surface data without drastically changing how it is gets centered and scaled to fit inside the reconstruction volume, while the other scenario do affect it. We denote each:

Slice: Occludes a slice of the depth map, leaving the bonding box unscathed.
Barrier: Occludes half of the depth map, reducing the bonding box.

We present in figure 6.20A-D shape completions from real-world YCB data augmented with both occlusion scenarios, by a 6D SIREN using $\mathcal{L}_{\text{TSDF,L1}}$. In E we perturb the depth map with values drawn from a Gaussian distribution to see how resilient our approach is to high-frequency noise.

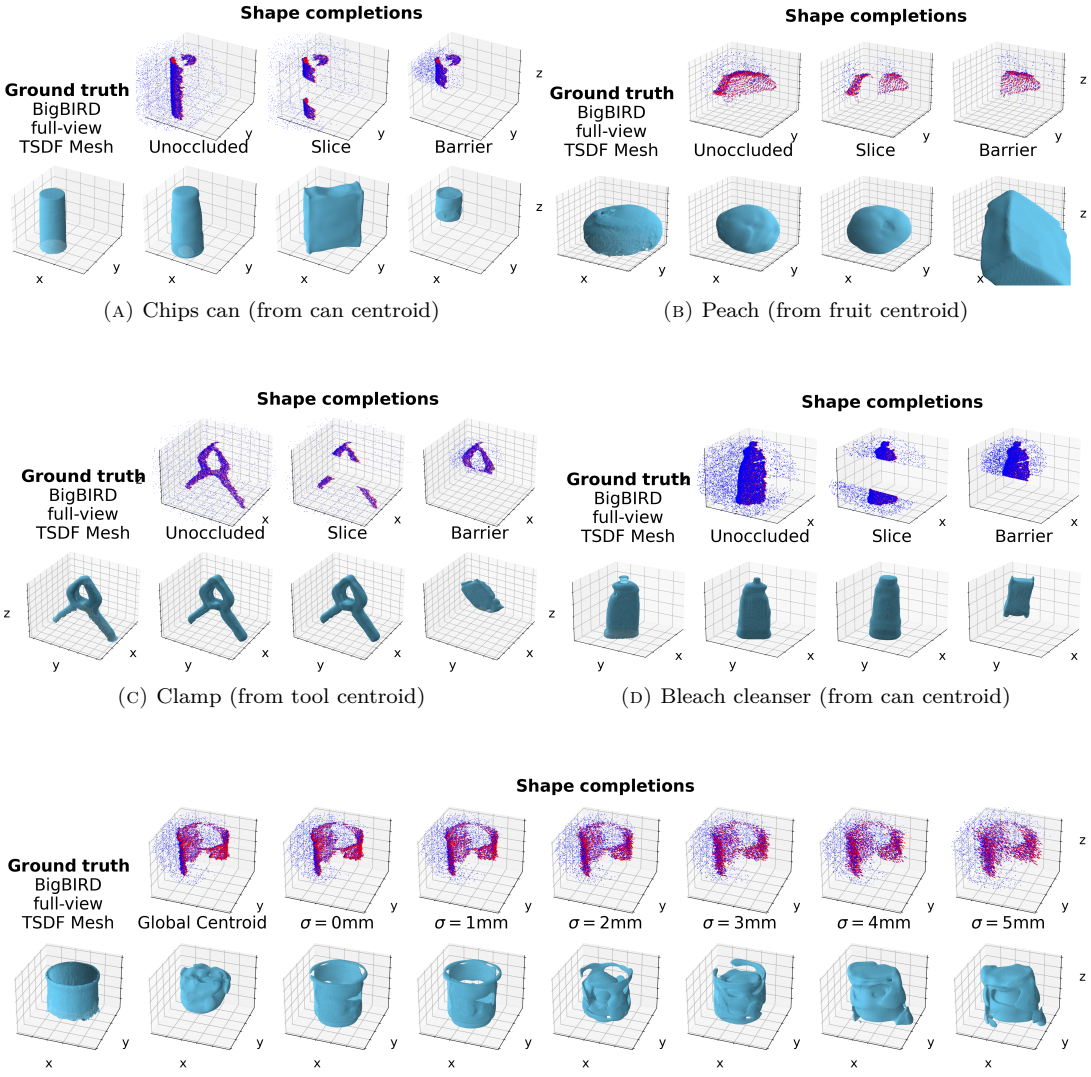
Our model deals quite well with missing data, and is for the most part resilient to the "slice" scenario. The SIREN thought however that the surface of the chips can was flat after we sliced it, showcasing one issue with real-world data: depth discontinuity filtering tends to remove data around the edges, which results in a shrunken silhouette. As a result the chips can was completed as a box (the centroid, see fig. 6.15a) instead of as a cylinder, due to the rounded front side largely being hidden. The "slice" augmentation simply passed the breaking point. We note that shape completing the same data with $\mathcal{L}_{\text{sim}}$ fixed this instance, but this may not apply to all cases. The bleach bottle, which started at the same class centroid, was not completed as a box when we sliced it. This is likely due to how the tip of the bottle conveys more information about the underlying shape than the top of the chips can does.



(A) NP1 (B) NP2 (C) NP3 (D) NP4 (E) NP5

FIGURE 6.19: The YCB `001_chips_can` at 285° for all BigBIRD depth camera perspectives NP1 - NP5 shown in figure 3.2a. We fit the near-surface SDF samples (blue and red) within the green sphere. The orange axis-aligned cube is the reconstruction volume we traverse with marching cubes. Note how the walls of the chips can disappear as the 3D camera moves towards to the zenith: depth cameras often fail to measure steep surfaces. Shape completion on NP4 and NP5 scans fail for this reason, as they lack any indication of how long the can ought be.

The "barrier" scenario consequently trips up our model. We illustrate in figure 6.19 why: in our setup, the placement of the reconstruction volume is what conveys most "contextual" information. Our model is not aware of any ground plane, nor the direction of gravity, nor any other information one can extract from the scene and use to infer a shape. The model *is* able to see the top of the chips can from the more elevated camera perspectives (D-E), but has no indication of how far down it is to the surface which the can stands on. The NP4 perspective exemplifies a problem with real-world depth sensor data: depth sensor often fail to register



(A) Chips can (from can centroid)

(B) Peach (from fruit centroid)

(C) Clamp (from tool centroid)

(D) Bleach cleanser (from can centroid)

(E) Cup subject to noise (from cup centroid)

FIGURE 6.20: Shape completion on real-world SDF clouds extracted from YCB RGB-D images, by network #18 (a 6D SIREN) in table 6.2. The first row in each subfigure display the single-view SDF clouds used to supervise the search. We display the best TSDF PSNR scoring shape out of ten, which took one minute to optimize for 600 steps and score. In (A-D) we augment the depth maps with occlusions: in one we *slice* of the mid-section and in the other we cut away half with a *barrier*. The latter augmentation affects how the SDF cloud is fitted within the reconstruction volume, further illustrated in figure 6.19. In (E) we inject into each depth pixel noise on drawn from $\mathcal{N}(0, \sigma^2)$. The "Global Centroid" completion in (E) started searching from the global centroid, while all the other completions started at their respective class centroid. All RGB-D images are taken from the NP2 camera angle, shown in figure 3.2a.

steep surfaces, in this case the walls of the chips can when viewed from above.

Our models is quite resilient to high-frequency noise. It was largely able to infer the orientation of the cup, despite a sample uncertainty with a standard deviation of 5mm. The defining shape of the cup were however lost with noise at this scale. We expect SIRENs trained on richer datasets to fare much better in a test like this. In reality, high-frequency noise such as this is not to be expected. Supplementary table A.1 enumerates common pre-processing techniques applied to depth maps, which are all designed to mitigate high-frequency noise. Low-frequency artifacts such as dents are still a major issue, and must be dealt with while pre-processing the depth data.

6.5.4 Non-Truncated Single-View Shape Completion

So far we have only discussed single-view shape completion where we search through the latent space with the truncated loss function $\mathcal{L}_{\text{TSDF,L1}}$. This is because $\mathcal{L}_{\text{L1}}$ and $\mathcal{L}_{\text{DISN,L1}}$ are affected by an issue with how we extract signed distances from single-views as defined in section 5.2.3. We illustrate and describe the problem in figure 6.21:

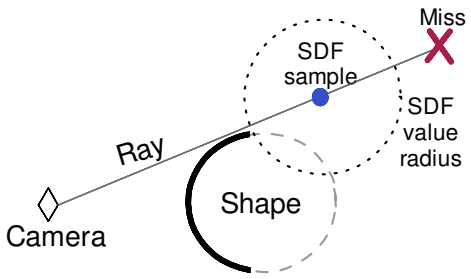


FIGURE 6.21: A diagram based on figure 5.2 of a scan ray cast from a camera into a scene with our shape of interest. The *uniform* single-view SDF points are sampled within the free-space covered by scan rays traced from the camera into the scene. We compute the SDF value (radius of dotted sphere) for each of these uniform points (blue dot) as the distance to the nearest hit point, that is, the visible surface of our object of interest (bold). Uniform points sampled far behind the shape end up with rather large SDF values, which end up "carving out" hidden parts of the object obscured by its own shadow. Non-truncated loss functions (like $\mathcal{L}_{\text{DISN,L1}}$ and $\mathcal{L}_{\text{L1}}$ from eq. 5.9) struggle with single-view shape completion for this reason. Their L2 variants are even more affected. $\mathcal{L}_{\text{TSDF}}$ mitigates this issue by clipping how much uniform points contribute to the loss, making it nearly unaffected.

SIRENs originally trained with $\mathcal{L}_{\text{L1}}$ and $\mathcal{L}_{\text{DISN,L1}}$ were able to perform shape completion just as well as what we have showcased thus far when we replaced their loss function with $\mathcal{L}_{\text{TSDF,L1}}$ or $\mathcal{L}_{\text{DISN,TSDF,L1}}$.

Chapter 7

Discussion

All in all our approach to single-view shape completion is promising. Here we briefly discuss some aspects our evaluation revealed, and how it fared.

7.1 Pose Estimation and Local Minima

Our approach to pose estimation is based on the same premise as the works of Liu et al. (2019) and Mildenhall et al. (2020), where pose estimation is achieved by iteratively optimizing the pose until the reconstruction match the target pose. This pose estimation method is prone to getting stuck in local minima. This happened for us in fig. 6.17v where the drill was completed upside down, and in figure 6.20d which completed the bleach bottle faced the wrong way around.

Fortunately we optimize both the pose and the shape at the same time. The shape completion animation in figure 6.18d shows us something interesting: The network matched with a clamp in the wrong orientation at first, but elected changed its shape to a different clamp instead of rotating the first one. This indicates that our method can avoid getting stuck if the shape space is enriched with copies of the same shape in multiple poses. This is trivial to do with data augmentation.

7.2 Learning Shapes by Learning to Pose Estimate

To account for observations not conforming to the canonical pose we initially planned to just use a transformation matrix to transform the input coordinates being fed into the SDF decoder. It is possible to optimize the parameters of such a matrix along with the shape code during shape completion. We instead chose to make the networks learn to perform this orientation transformation internally, by optimizing shapes over a continuous space of orientations (sec. 5.4.1). This was to to make shape completion draw more influence from the priors embedded in the network when optimizing the pose.

This novel augmentation strategy seem to have had a regularizing side effect. It encouraged the networks to extract shape features that are rotationally invariant. These features are easier for the networks to learn in a rotating context, as they have more degrees of freedom.

The LReLU MLP with 64 shape features in figure 6.7 showcase the effects of this, as it learned primarily lathed approximations of the training shapes. LReLU MLPs with sufficient learning capacity fared quite well with this intensive regularization. It did not reduce the rate of convergence all that much while training. Thanks to $\mathcal{L}_{\text{sim}}$ it not degrade the reconstruction quality either, making it remain competitive to related works, as we note in the paragraph below table 6.3.

This makes our novel orientation augmentation scheme (sec. 5.4.1) a good contribution to the study of 3D shape representation learning and feature extraction, even when pose estimation is not needed.

7.3 Transfer of Knowledge

The auto-decoder architecture we used exhibited an impressive capacity for knowledge discovery. Both the LReLU and SIREN decoders managed to embed a wide selection of objects in a shared latent space, grouping them by their structural similarities in a reproducible fashion. But how did this knowledge of prior shapes that the networks learned during training transfer over to a single-view context?

The LReLU networks learned very uniform latent spaces. It did so by extracting many simple shape features from the training data, which it adds back up again to reconstruct the shapes. This makes LReLU shape reconstructions very pliable. That however is a problem for shape completion which relies on inductive biases to account for missing data. The rotationally invariant shape features did help however. The LReLU MLPs were largely for this reason only able to shape complete lathed shapes, such as the chips can.

The SIRENs converged on latent spaces less regular than what LReLU did, but its reconstructions stay more true to the training data, leading to stronger inductive biases. The SIRENs used their shape embeddings efficiently to inform the shape completion process, showing that what it learned during training can transfer over to a single-view context.

7.4 Setbacks

Our approach shows much promise, but we did not quite manage to make it reach its full potential. This we primarily attribute to two issues that came up during our evaluation:

Issue 1: The small size of the YCB object dataset, and how its 3D models are not aligned with one another. This led to less uniform latent spaces as it prevented our networks from extracting shape features shared between the different objects. This is apparent in figure 6.15 by how most of the class centroid shapes were pretty nondescript. None of the tool objects (see supplementary figure A.2) are aligned, preventing the networks from learning that their structure are in fact are similar. The boxes and Lego pieces are also misaligned, hindering the networks from learning a shared understanding of what a box is. As such they mixed the boxes and the cylinders, apparent in figs. 6.9a, 6.9d. Although YCB is a good benchmark for shape completion and visual servoing, it proved inadequate as a shape learning curriculum.

Issue 2: How all our networks learned a signed distance field (SDF) whose *mean* gradient magnitude equaled 0.1, instead of the expected constant value of 1. The magnitudes seem to diminish the closer we move to the surface, implying that the issue might lie with how the near-surface SDF training examples are generated. This issue caused one of the latent space regularization schemes we intended to use ($\mathcal{L}_{\text{norm}}$ in eq. 5.9) to not work at all. This is a shame, as we had hoped to use it to constrain a property of the signed distance field which is independent of the underlying shape, enabling us to smooth the space in areas not covered by the database of known latent vectors maintained by the auto-decoder trainer. Tracking down this issue cost us a lot of time, and we yet to uncover exactly what the root cause is.

7.5 Meeting our Research Goals

Here we revisit our research goals **T1** through **T5** stated in section 1.3:

T1 Investigate previous state-of-the-art single-view shape completion approaches.

Our investigation spanned the related works covered in chapter 4. We reviewed many different single-view shape completion techniques, highlighting their strengths and weaknesses.

T2 Define and design a deep learning model for single-view shape completion.

We defined our learning model in chapter 5, and elaborated on our single-view search strategies while evaluating them in in section 6.5.

T3 Implement and train this model with the YCB object dataset.

We implemented our learning model in PyTorch and trained on the NTNU IDUN/EPIC GPGPU computing cluster by Själander et al. (2019). We trained in total 821 networks on the YCB object dataset while tuning its performance against the validation metrics, the journey of which is broadly covered in section 6.2. We presented the final 20 networks trained in table 6.1, which we then proceeded to evaluate.

T4 Evaluate and discuss the results for 3D single-view shape completion.

We evaluated the quality of shapes reconstructed from known latent vectors in section 6.3, and examined how the shapes are represented in latent space in section 6.4. We evaluated the ability of our networks to perform single-view shape completion using both synthetic and real-world depth sensor data in section 6.5. Our best model completed credible shapes with accurate camera space poses. We discussed aspects of our learning model revealed during our evaluation here in chapter 7.

T5 Outline future work.

We indicate future directions for improving on our approach in section 8.2.

Chapter 8

Conclusion & Future Work

8.1 Conclusion

We trained both LReLU and SIREN based signed distance field (SDF) auto-decoders to represent the YCB dataset in latent space. Supervising the networks with SDF gradients reduced the variation between training runs, and produced more accurate shape reconstructions. Including multiple objects in each training batch improved the rate of convergence. LReLU performed better with a weighted and truncated loss, while SIREN performed better with a just truncated loss. Both LReLU and SIREN grouped the shapes in latent space by their structural similarities in a reproducible fashion, drawing meaningful connections between them. The reconstitution quality of SIREN outperformed LReLU in every metric, despite SIREN being half their size.

We showed that these SDF decoders can embed a wide variety of shapes over a continuous space of orientations. We augmented the networks to learn to reconstruct the shapes in any orientation, to enable pose estimation influenced by the embedded shapes. The networks also extracted a few rotationally invariant shape features as a result. This transferred over to single-view shape completion quite well.

We showed that a single-view shape completion approach based on searching through latent space for shapes conforming to depth observations is viable in a real-world setting. LReLU struggled to aid the search, but SIREN inferred hidden parts of the shapes from its shape embeddings to an impressive degree, making it resistant to missing data. When we used the object class to inform the search, it was able infer shapes with credible far sides.

Based on the findings in this thesis, we conclude that our approach presented here contains multiple novel contributions to the study of 3D implicit shape representation learning, and is promising for robotic single-view 3D shape completion of objects during the manipulation phase.

8.2 Future work

Here we propose recommendations and indicate future directions to bring higher robustness to the approach we presented.

First of all, regarding the datasets used in this work: although the YCB object dataset is a good benchmark, it is an inadequate learning curriculum. We propose training with richer datasets containing more geometric primitives such as boxes, cones, tori, and cylinders, and with more members in each category. The dataset should also be be augmented to with multiple poses, to avoid the networks getting stuck in local minima. The training shapes should also be aligned, either done so automatically or by hand. One could explore minimizing the optimal transport distance between uniformly sampled surface points, or aligning the principal inertia vectors of the shapes. Instead of centering shapes by their axis-aligned bounding boxes (which is easily dominated by outlier protrusions) we instead propose using a more meaningful center such as the mass center, or the Wasserstein barycenter for the surface points.

Searching the latent space could be further enhanced. Additional networks could be used to predict the most probable pose, center and scale to start searching from. The search itself could be further tuned to not be as gentle as ours, thus requiring fewer optimization steps. The scheduled learning rate for the pose and for the shape do not have to be tied together. Instead of manually tuning the shape completion hyperparameters for each network like we did, they could be automatically discovered using Bayesian optimization.

Shape completion does not have to be done on a static RGB-D image. Humans continually adjust their prediction based on visual feedback when manipulating shapes. As such we propose expanding the shape completion search to work on live-streaming sensor data as an online algorithm.

Instead of supervising the single-view shape completion search with explicit SDF targets, which we showed was problematic in section 6.5.4, one could use a cost function inspired by the boundary value problem formulations of Sitzmann, Martel, et al. (2020): constrain the near-surface points to explicit SDF targets like we do, but constrain the uniform points sampled within the visible free-space to simply be positive.

The latent spaces could be better formed and constrained during training. We propose to: (1) Constrain the global centroid to a shape more beneficial to shape completion. This may enable class-unaware search. (2) Perturb the latent vectors during training with noise. This may widen and join the shape neighborhoods in latent space, and make it more uniform. (3) Experiment with constraining invariant field properties on random latent vectors. One such property is the spatial SDF gradient magnitude. We did not manage to do this due to a setback discussed in section 7.4. (4) Experiment with alternative network architectures. Several papers pushing the field forward have been released in 2021 alone, concurrent with this thesis. Most promising is the work of Mehta et al. (2021), who propose a way to condition SIRENs to generalize better to never-before seen targets, by first feeding the latent vectors through a separate modulator network. Each hidden modulator layer is then multiplied onto a corresponding SIREN layer. We found their paper only 16 days after submission, but still too late to incorporate their findings into our learning model.

We suggest experimenting with convolutional techniques for enhancing the RGB-D sensor data, to clean up discontinuities and holes, and superscale it. This should enable shape completion on more challenging objects from less reliable data. We suggest looking into works such as MSG-Net by Hui, Loy, and Xiaoou Tang (2016) and ClearGrasp by Sajjan et al. (2019). The latter enables shape completion on objects whose visual properties classically elude depth sensors, such as transparency.

Bibliography

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. "Layer Normalization." http://arxiv.org/abs/1607.06450.

Calli, B., A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. 2015. "The Ycb Object and Model Set: Towards Common Benchmarks for Manipulation Research." In *2015 International Conference on Advanced Robotics (Icar)*, 510–17. https://doi.org/10.1109/ICAR.2015.7251504.

Calli, B., A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. 2015. "Benchmarking in Manipulation Research: Using the Yale-Cmu-Berkeley Object and Model Set." *IEEE Robotics Automation Magazine* 22 (3): 36–52. https://doi.org/10.1109/MRA.2015.2448951.

Chabra, Rohan, Jan Eric Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. 2020. "Deep Local Shapes: Learning Local Sdf Priors for Detailed 3D Reconstruction." http://arxiv.org/abs/2003.10983.

Chang, Angel X., Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, et al. 2015. "ShapeNet: An Information-Rich 3D Model Repository." arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Chaton, Thomas, Chaulet Nicolas, Sofiane Horache, and Loic Landrieu. 2020. "Torch-Points3d: A Modular Multi-Task Frameworkfor Reproducible Deep Learning on 3D Point Clouds." In *2020 International Conference on 3D Vision (3DV)*. IEEE. https://github.com/nicolas-chaulet/torch-points3d.

Chen, Zhiqin, and Hao Zhang. 2019. "Learning Implicit Fields for Generative Shape Modeling." *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dai, Angela, Charles Ruizhongtai Qi, and Matthias Nießner. 2017. "Shape Completion Using 3D-Encoder-Predictor Cnns and Shape Synthesis." http://arxiv.org/abs/1612.00101.

Falcon, et al., WA. 2019. "PyTorch Lightning." *GitHub*. https://github.com/PyTorchLightning/pytorch-lightning.

Fan, Haoqiang, Hao Su, and Leonidas Guibas. 2016. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." http://arxiv.org/abs/1612.00603.

Fey, Matthias, and Jan E. Lenssen. 2019. "Fast Graph Representation Learning with PyTorch Geometric." In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Girdhar, Rohit, David F. Fouhey, Mikel Rodriguez, and Abhinav Gupta. 2016. "Learning a Predictable and Generative Vector Representation for Objects." http://arxiv.org/abs/1603.08637.

Groueix, Thibault, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 2018. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation." http://arxiv.org/abs/1802.05384.

Han, Xiaoguang, Zhen Li, Haibin Huang, Evangelos Kalogerakis, and Yizhou Yu. 2017. "High-Resolution Shape Completion Using Deep Neural Networks for Global Structure and Local Geometry Inference." http://arxiv.org/abs/1709.07599.

Häne, Christian, Shubham Tulsiani, and Jitendra Malik. 2017. "Hierarchical Surface Prediction for 3D Object Reconstruction." http://arxiv.org/abs/1704.00710.

Hanocka, Rana, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. "MeshCNN: A Network with an Edge." *ACM Transactions on Graphics (TOG)* 38 (4): 90:1–90:12.

Hao, Zekun, Hadar Averbuch-Elor, Noah Snavely, and Serge Belongie. 2020. "DualSDF: Semantic Shape Manipulation Using a Two-Level Representation," 7631–41.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2018. "Mask R-Cnn." http://arxiv.org/abs/1703.06870.

He, Tong, John Collomosse, Hailin Jin, and Stefano Soatto. 2020. "Geo-Pifu: Geometry and Pixel Aligned Implicit Functions for Single-View Human Reconstruction." http://arxiv.org/abs/2006.08072.

Hoang, Long, Suk-Hwan Lee, Oh-Heum Kwon, and Ki-Ryong Kwon. 2019. "A Deep Learning Method for 3D Object Classification Using the Wave Kernel Signature and a Center Point of the 3D-Triangle Mesh." *Electronics* 8 (10). https://doi.org/10.3390/electronics8101196.

Hui, Tak-Wai, Chen Change Loy, and and Xiaoou Tang. 2016. "Depth Map Super-Resolution by Deep Multi-Scale Guidance." In *Proceedings of European Conference on Computer Vision (Eccv)*, 353–69. http://mmlab.ie.cuhk.edu.hk/projects/guidance_SR_depth.html.

Jalal, Anand, and Vrijendra Singh. 2012. "The State-of-the-Art in Visual Object Tracking." *Informatica (Slovenia)* 36 (January): 227–48.

Kazhdan, Michael, and Hugues Hoppe. 2013. "Screened Poisson Surface Reconstruction." *ACM Trans. Graph.* 32 (3). https://doi.org/10.1145/2487228.2487237.

Kingma, Diederik P., and Jimmy Ba. 2017. "Adam: A Method for Stochastic Optimization." http://arxiv.org/abs/1412.6980.

Kingma, Diederik P., and Max Welling. 2019. "An Introduction to Variational Autoencoders." *Foundations and Trends® in Machine Learning* 12 (4): 307–92. https://doi.org/10.1561/2200000056.

Kleineberg, Marian, Matthias Fey, and Frank Weichert. 2020. "Adversarial Generation of Continuous Implicit Shape Representations." http://arxiv.org/abs/2002.00349.

Littwin, Gidi, and Lior Wolf. 2019. "Deep Meta Functionals for Shape Representation." http://arxiv.org/abs/1908.06277.

Liu, Shichen, Tianye Li, Weikai Chen, and Hao Li. 2019. "Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning." *The IEEE International Conference on Computer Vision (ICCV)*, October.

Maaten, Laurens van der, and Geoffrey Hinton. 2008. "Visualizing Data Using T-Sne." *Journal of Machine Learning Research* 9 (86): 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html.

Mehta, Ishit, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. 2021. "Modulated Periodic Activations for Generalizable Local Functional Representations." http://arxiv.org/abs/2104.03960.

Mescheder, Lars, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. "Occupancy Networks: Learning 3D Reconstruction in Function Space." In *Proceedings Ieee Conf. On Computer Vision and Pattern Recognition (Cvpr)*.

Micikevicius, Paulius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, et al. 2018. "Mixed Precision Training." http://arxiv.org/abs/1710.03740.

Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." http://arxiv.org/abs/2003.08934.

Niu, Chengjie, Jun Li, and Kai Xu. 2018. "Im2Struct: Recovering 3D Shape Structure from a Single Rgb Image." http://arxiv.org/abs/1804.05469.

Park, Jeong Joon, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation." In *The Ieee Conference on Computer Vision and Pattern Recognition (Cvpr)*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, 8024–35. Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedersen, Ole-Magnus, Ekrem Misimi, and François Chaumette. 2020. "Grasping Unknown Objects by Coupling Deep Reinforcement Learning, Generative Adversarial Networks, and Visual Servoing." In *ICRA 2020 - IEEE International Conference on Robotics and Automation*, 1–8. Paris, France: IEEE. https://hal.inria.fr/hal-02495837.

Pontes, Jhony K., Chen Kong, Sridha Sridharan, Simon Lucey, Anders Eriksson, and Clinton Fookes. 2017. "Image2Mesh: A Learning Framework for Single Image 3D Reconstruction." *arXiv:1711.10669*.

Qi, Charles R., Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." http://arxiv.org/abs/1612.00593.

Qi, Charles R., Li Yi, Hao Su, and Leonidas J. Guibas. 2017. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." http://arxiv.org/abs/1706.02413.

Saito, Shunsuke, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. 2019. "PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization." http://arxiv.org/abs/1905.05172.

Saito, Shunsuke, Tomas Simon, Jason Saragih, and Hanbyul Joo. 2020. "PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization." http://arxiv.org/abs/2004.00452.

Sajjan, Shreeyak S., Matthew Moore, Mike Pan, Ganesh Nagaraja, Johnny Lee, Andy Zeng, and Shuran Song. 2019. "ClearGrasp: 3D Shape Estimation of Transparent Objects for Manipulation." http://arxiv.org/abs/1910.02550.

Salimans, Tim, and Diederik P. Kingma. 2016. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks." http://arxiv.org/abs/1602.07868.

Sarmad, Muhammad, Hyunjoo Jenny Lee, and Young Min Kim. 2019. "RL-Gan-Net: A Reinforcement Learning Agent Controlled Gan Network for Real-Time Point Cloud Shape Completion." http://arxiv.org/abs/1904.12304.

Singh, Arjun, James Sha, Karthik Narayan, Tudor Achim, and Pieter Abbeel. 2014. "BigBIRD: A Large-Scale 3D Database of Object Instances." In, 509–16. https://doi.org/10.1109/ICRA.2014.6906903.

Sitzmann, Vincent, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. 2020. "MetaSDF: Meta-Learning Signed Distance Functions." In *ArXiv*.

Sitzmann, Vincent, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. “Implicit Neural Representations with Periodic Activation Functions.” In *Proc. NeurIPS*.

Själander, Magnus, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. 2019. “EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure.” *arXiv:1912.05848 [Cs]*, December. http://arxiv.org/abs/1912.05848.

Smith, Edward, Scott Fujimoto, Adriana Romero, and David Meger. 2019. “GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects.” In *Proceedings of the 36th International Conference on Machine Learning*, edited by Kamalika Chaudhuri and Ruslan Salakhutdinov, 97:5866–76. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR. http://proceedings.mlr.press/v97/smith19a.html.

Stutz, David, and Andreas Geiger. 2018. “Learning 3D Shape Completion from Laser Scan Data with Weak Supervision.” In *IEEE Conference on Computer Vision and Pattern Recognition (Cvpr)*. IEEE Computer Society.

Sundt, Peder Bergebakken. 2020. “Single View 3D Reconstruction for Robotic Grasping of 3D Objects.” NTNU, Specialization Thesis.

Tancik, Matthew, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. 2021. “Learned Initializations for Optimizing Coordinate-Based Neural Representations.” In *CVPR*.

Tancik, Matthew, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains.” http://arxiv.org/abs/2006.10739.

Tatarchenko, Maxim, Alexey Dosovitskiy, and Thomas Brox. 2017. “Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs.” http://arxiv.org/abs/1703.09438.

Tatarchenko, Maxim, Stephan R. Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. 2019. “What Do Single-View 3D Reconstruction Networks Learn?” http://arxiv.org/abs/1905.03678.

Tran, Dustin. 2016. “Probabilistic Decoder.” Edwardlib. 2016. http://edwardlib.org/tutorials/decoder.

Tremblay, Jonathan, Thang To, and Stan Birchfield. 2018. “Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation.” http://arxiv.org/abs/1804.06534.

Tulsiani, Shubham, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. 2017. “Multi-View Supervision for Single-View Reconstruction via Differentiable Ray Consistency.” http://arxiv.org/abs/1704.06254.

Wang, Nanyang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. “Pixel2Mesh: Generating 3D Mesh Models from Single Rgb Images.” http://arxiv.org/abs/1804.01654.

Wang, Peng-Shuai, Chun-Yu Sun, Yang Liu, and Xin Tong. 2019. “Adaptive O-Cnn.” *ACM Transactions on Graphics* 37 (6): 1–11. https://doi.org/10.1145/3272127.3275050.

Wang, Weiyue, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. “3DN: 3D Deformation Network.” In *CVPR*.

Wen, Chao, Yinda Zhang, Zhuwen Li, and Yanwei Fu. 2019. “Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation.” http://arxiv.org/abs/1908.01491.

Wu, Jiajun, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. 2017. "MarrNet: 3D Shape Reconstruction via 2.5D Sketches." http://arxiv.org/abs/1711.03129.

Wu, Jiajun, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T. Freeman, and Joshua B. Tenenbaum. 2018. "Learning Shape Priors for Single-View 3D Completion and Reconstruction." http://arxiv.org/abs/1809.05068.

Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. "Detectron2." https://github.com/facebookresearch/detectron2.

Xu, Qiangeng, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. "DISN: Deep Implicit Surface Network for High-Quality Single-View 3D Reconstruction." In *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, 492–502. Curran Associates, Inc. http://papers.nips.cc/paper/8340-disn-deep-implicit-surface-network-for-high-quality-single-view-3d-reconstruction.pdf.

Yan, Xinchen, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2017. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction Without 3D Supervision." http://arxiv.org/abs/1612.00814.

Yang, Guandao, Yin Cui, Serge Belongie, and Bharath Hariharan. 2018. "Learning Single-View 3D Reconstruction with Limited Pose Supervision." In *The European Conference on Computer Vision (Eccv)*.

Yen-Chen, Lin, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. 2020. "Learning to See Before Learning to Act: Visual Pre-Training for Manipulation." In *IEEE International Conference on Robotics and Automation (Icra)*. https://yenchenlin.me/vision2action/.

Zeng, Andy, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 2017. "3DMatch: Learning Local Geometric Descriptors from Rgb-d Reconstructions." http://arxiv.org/abs/1603.08182.

Zhou, Yi, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. 2019. "On the Continuity of Rotation Representations in Neural Networks." In *The Ieee Conference on Computer Vision and Pattern Recognition (Cvpr)*.

Zhu, Rui, Hamed Kiani Galoogahi, Chaoyang Wang, and Simon Lucey. 2017. "Rethinking Reprojection: Closing the Loop for Pose-Aware Shapereconstruction from a Single Image." http://arxiv.org/abs/1707.04682.

Zou, Chuhang, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 2017. "3D-Prnn: Generating Shape Primitives with Recurrent Neural Networks." http://arxiv.org/abs/1708.01648.

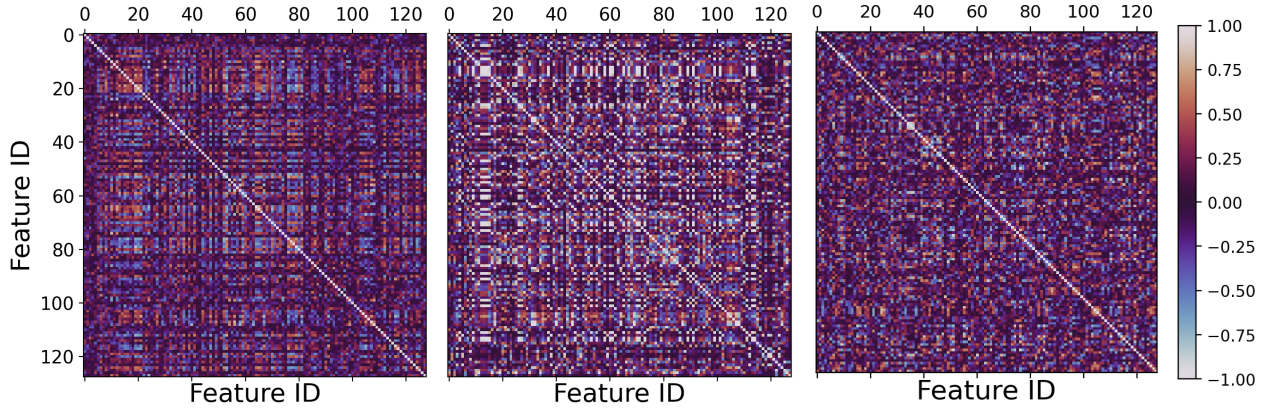
Appendix A

Supplementary

Here we report information and provide explanations deemed excessive for the main thesis.

TABLE A.1: The post-processing filters applied by default to the depth image stream in Intel RealSense Viewer in order. In general they filter out high-frequency noise and increase the dynamic sensor range.

#	Filter name	On by default?	Description
1	Decimation Filter	✓	2×2 to 8×8 median-pooling.
2	HDR Merge	✓	Fuses two consecutive depth images captured at different exposures.
3	Filter by Sequence ID	-	Only use every N frame.
4	Threshold Filter	✓	Clips data outside a min and max distance.
5	Depth to Disparity	✓	Transforms to disparity space ($1/d_{\text{Distance}}$)
6	Spatial Filter	✓	Edge-Preserving 1D filter applied both horizontally and vertically.
7	Temporal Filter	✓	Manipulation based on previous frames. Effectively increases exposure time.
8	Hole Filling Filter	-	Rectifies missing data based on neighboring data.
9	Disparity To Depth	✓	Transforms back to distance space ($1/d_{\text{Disparity}}$)



(A) LReLU (#12) (B) SIREN (#18) (C) "Relaxed SIREN"

FIGURE A.1: Pearson product-moment correlation matrices for the three sets of learned shape features exhibited in figure 6.8. It measures the linear dependence between features. The # numbers refer to the rows in table 6.2.

TABLE A.2: Our whitelist of objects in the YCB object and dataset used to train our shape completion network, along with the class labels we assigned to them. We present a render of each object in figure A.2. We filtered many of the objects due to either distortions or poor alignment.

#	Name	Class	#	Name	Class
1	001_chips_can	can	43	057_racquetball	ball
2	002_master_chef_can	can	44	058_golf_ball	ball
3	003_cracker_box	box	45	059_chain	other
4	004_sugar_box	box	46	061_foam_brick	box
5	005_tomato_soup_can	can	47	062_dice	box
6	006_mustard_bottle	can	48	065-a_cups	cup
7	007_tuna_fish_can	can	49	065-b_cups	cup
8	008_pudding_box	box	50	065-c_cups	cup
9	009_gelatin_box	box	51	065-d_cups	cup
10	010_potted_meat_can	can	52	065-e_cups	cup
11	011_banana	fruit	53	065-f_cups	cup
12	012_strawberry	fruit	54	065-g_cups	cup
13	013_apple	fruit	55	065-h_cups	cup
14	014_lemon	fruit	56	065-i_cups	cup
15	015_peach	fruit	57	065-j_cups	cup
16	016_pear	fruit	58	070-a_colored_wood_blocks	other
17	017_orange	fruit	59	071_nine_hole_peg_test	other
18	018_plum	fruit	60	072-a_toy_airplane	airplane
19	019_pitcher_base	other	61	072-b_toy_airplane	airplane
20	021_bleach_cleanser	can	62	072-c_toy_airplane	airplane
21	024_bowl	other	63	072-d_toy_airplane	airplane
22	026_sponge	other	64	072-e_toy_airplane	airplane
23	029_plate	other	65	072-f_toy_airplane	airplane
24	030_fork	tool	66	072-h_toy_airplane	airplane
25	031_spoon	tool	67	072-i_toy_airplane	airplane
26	032_knife	tool	68	072-j_toy_airplane	airplane
27	033_spatula	tool	69	072-k_toy_airplane	airplane
28	035_power_drill	tool	70	073-b_lego_duplo	lego
29	036_wood_block	box	71	073-c_lego_duplo	lego
30	037_scissors	tool	72	073-d_lego_duplo	lego
31	040_large_marker	tool	73	073-e_lego_duplo	lego
32	042_adjustable_wrench	tool	74	073-f_lego_duplo	lego
33	043_phillips_screwdriver	tool	75	073-g_lego_duplo	lego
34	044_flat_screwdriver	tool	76	073-h_lego_duplo	lego
35	048_hammer	tool	77	073-i_lego_duplo	lego
36	050_medium_clamp	tool	78	073-j_lego_duplo	lego
37	051_large_clamp	tool	79	073-k_lego_duplo	lego
38	052_extra_large_clamp	tool	80	073-l_lego_duplo	lego
39	053_mini_soccer_ball	ball	81	073-m_lego_duplo	lego
40	054_softball	ball	82	076_timer	other
41	055_baseball	ball	83	077_rubiks_cube	box
42	056_tennis_ball	ball			

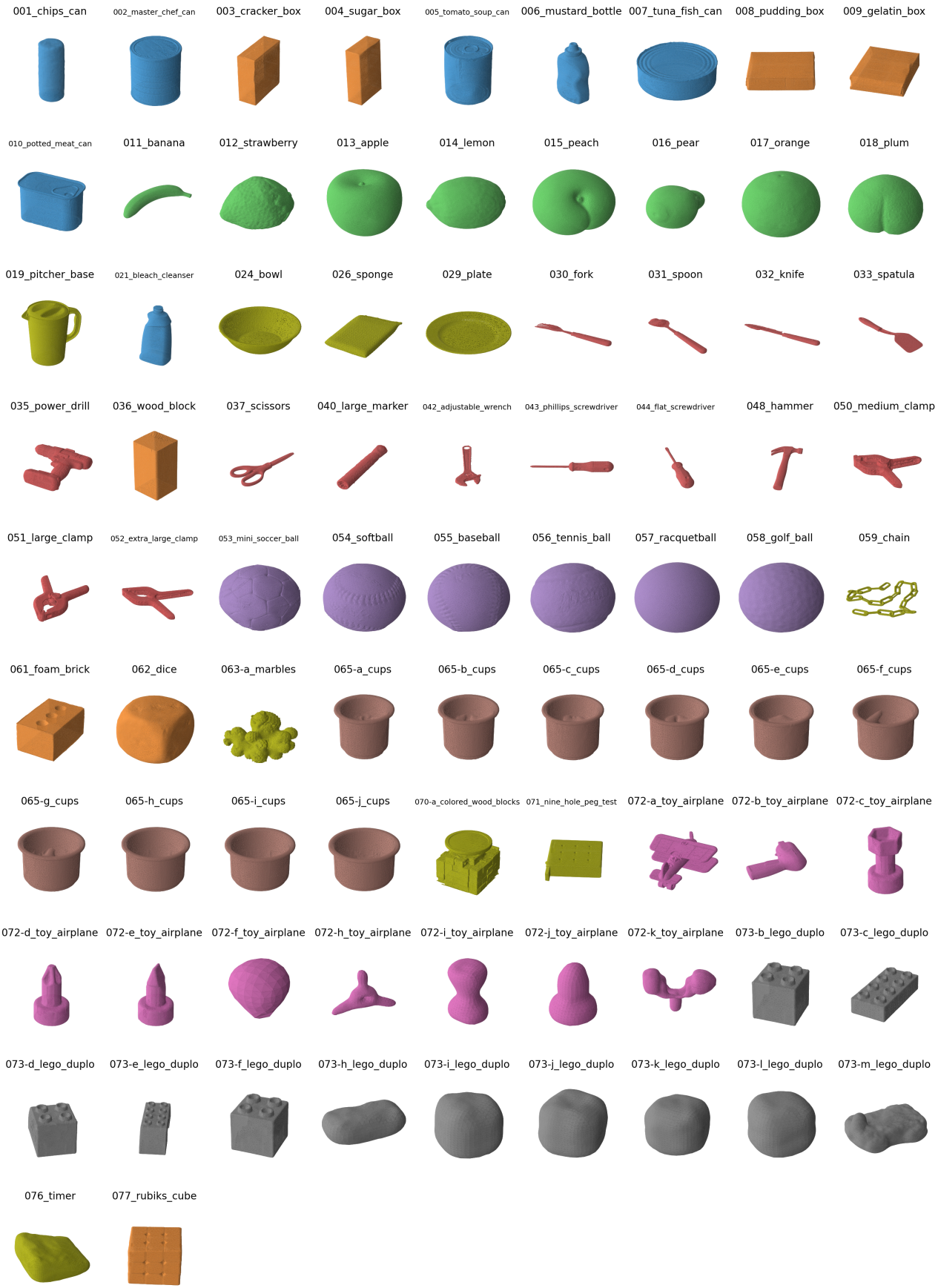
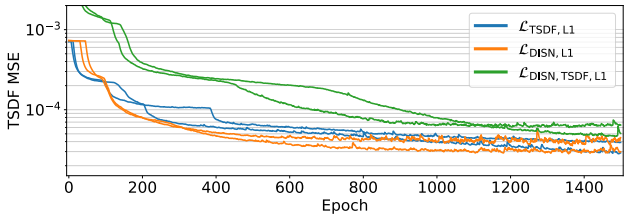
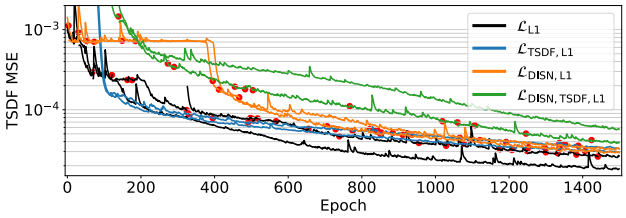


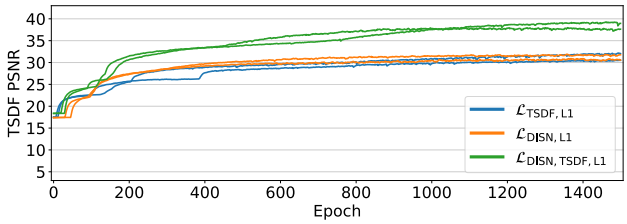
FIGURE A.2: The 3D YCB meshes we trained our networks with, rendered in their canonical pose. We use the Google scanner meshes if available, falling back to BigBIRD Poisson reconstructions otherwise. The meshes are colored according to our assigned classes, using the same colors as other figures. Apparent here is how few of the objects have been aligned to one another, leading to poor knowledge discovery.



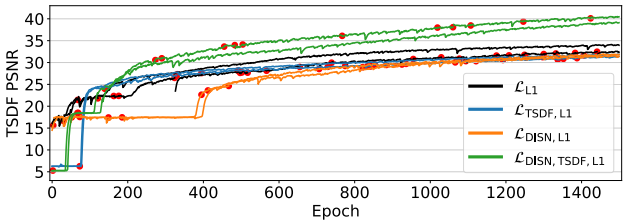
(A) LReLU: TSDF MSE



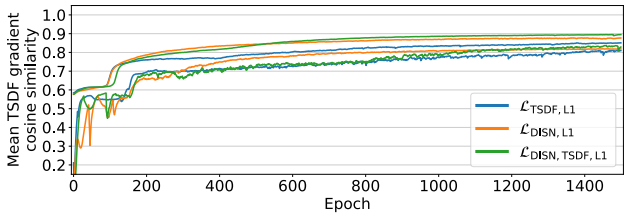
(B) SIREN: TSDF MSE



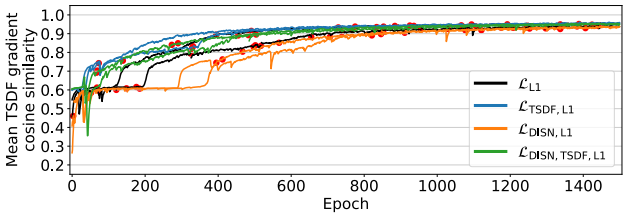
(C) LReLU: TSDF PSNR



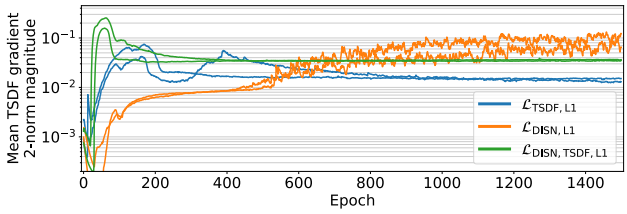
(D) SIREN: TSDF PSNR



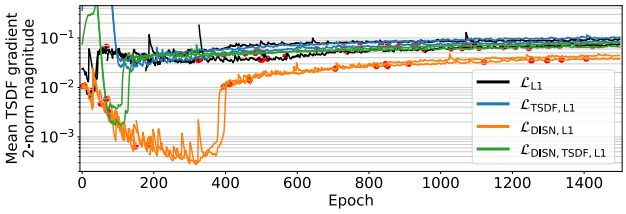
(E) LReLU: TSDF gradient cosine similarity



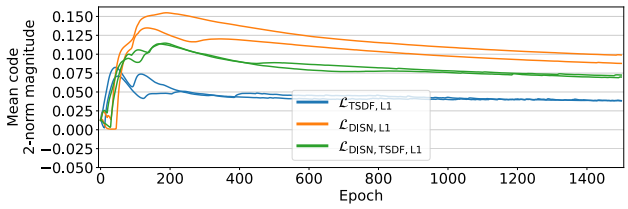
(F) SIREN: TSDF gradient cosine similarity



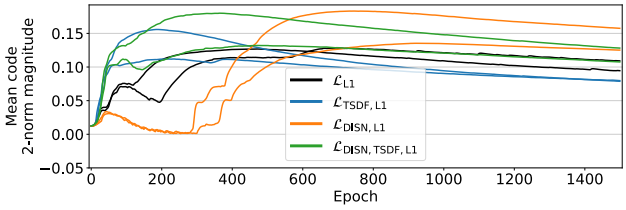
(G) LReLU: TSDF gradient magnitude



(H) SIREN: TSDF gradient magnitude



(I) LReLU: code magnitude



(J) SIREN: code magnitude

FIGURE A.3: All training metrics measured across the validation dataset during training, smoothed with α=0.8 EMA. We show the networks without positional encoding in table 6.1. There are two runs for every loss function: LReLUs trained with and without gradient supervision, while the SIRENs trained with both 3D (Euler) and 6D rotation vectors (expanded with a cross product). Red dots and crosses are NaNs, the latter indicating the network never recovered. SIRENs seem to produce and recover from regressing NaNs quite often, a characteristic not observed with LReLU.

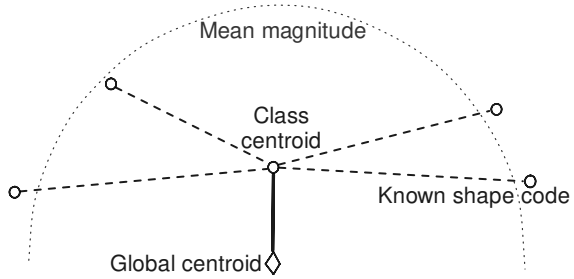


FIGURE A.4: A visual explanation in 2D of how some of some of class centroids reported in figure 6.16 may have such a low magnitude despite how all its members each have a magnitude near the global mean. This issue is more pronounced in higher dimensions.

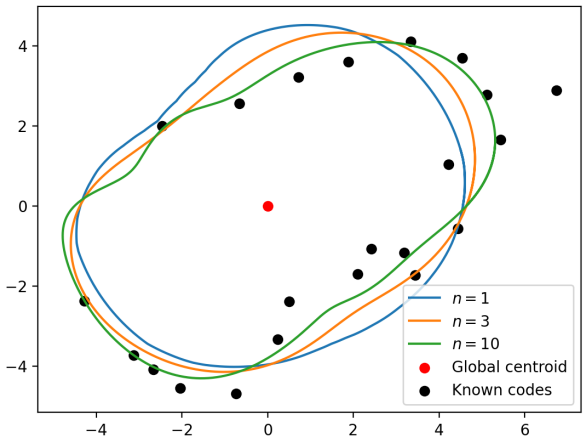
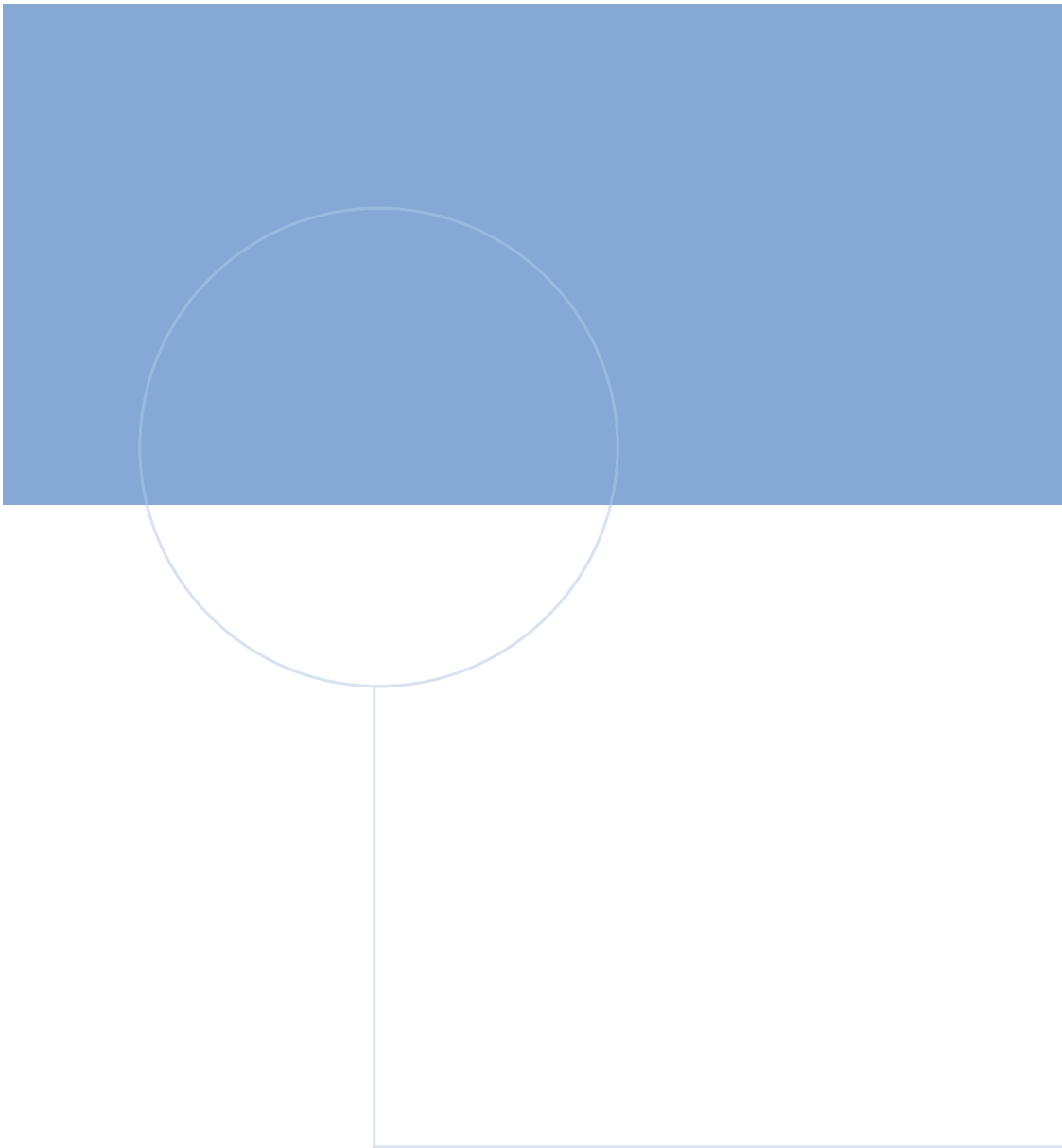


FIGURE A.5: A 2D visualization of the hull we tried normalizing the LReLU shape codes to during "aggressive" shape completion search. The distance from the global centroid to the hypersurface is determined by the magnitude of similar known latent vectors.



NTNU

Norwegian University of
Science and Technology