SCALABLE 3D DEEP LEARNING: METHODS AND APPLICATIONS

MAXIM TATARCHENKO

Dissertation zur Erlangung des Doktorgrades der
Technischen Fakultät der
Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan:	Prof. Dr. Rolf Backofen
Prüfungskommission:	Prof. Dr. Thomas Brox
	Prof. Dr. Andreas Geiger
	Prof. Dr. Matthias Teschner
	Prof. Dr. Abhinav Valada
Datum der Disputation:	7.07.2020

Have I kept you waiting, darling? My apologies, Newton was holding me up.

The very same Munchhausen, 1979

ABSTRACT

In recent years, deep convolutional neural networks (CNNs) have achieved remarkable success in most 2D image processing tasks. Still, the direct processing of 3D data, not only their 2D projections, is required in many applications. Naive extensions of 2D deep learning techniques into 3D rely on voxel grids which are inefficient due to their cubic complexity and thus only support prohibitively low resolutions. Therefore, it is crucially important to develop dedicated 3D deep learning algorithms that operate on efficient 3D representations like point clouds and octrees. The algorithm design depends on the task type: in analysis tasks the 3D geometry is given as input and needs to be processed efficiently, while in synthesis tasks the 3D geometry has to be generated from an abstract representation.

The first part of this thesis proposes methods for both efficient analysis and synthesis of 3D data using CNNs. In case of analysis, we focus on the task of semantic segmentation of 3D scenes represented as point clouds. We propose a new construct - tangent convolution - which operates directly on surfaces and can be used to build efficient CNNs. We apply networks that use tangent convolutions to large-scale semantic segmentation of real-world indoor and outdoor datasets. For synthesis, we propose a novel convolutional decoder for generating 3D shapes represented as octrees. Our approach is significantly more efficient than CNNs based on dense voxel grids, both in memory consumption and in computation time. This enables the generation of high-resolution 3D shapes. We validate the proposed method on three different tasks, including shape auto-encoding, generating shapes from high-level information, and single-view 3D reconstruction.

The second part of the thesis is devoted to the problem of single-view 3D reconstruction; in particular, to a realistic analysis of its current state. Many recent CNN-based methods for solving this task focus on developing dedicated decoders for different 3D representations. We systematically analyze these methods comparing them with two custom baselines which internally rely on image classification and retrieval, i.e. which solve a recognition problem. The results of these simple decoder-less baselines are statistically indistinguishable from more sophisticated state-of-the-art networks, indicating that the latter also rely mostly on recognition. We formulate the problems in the widely adopted experimental setup that lead to such behavior and outline possible solutions. Finally, we extensively study how well existing single-view 3D reconstruction methods which are trained on synthetic data, generalize to real-world images. For this, we collected a new dataset of objects of various shapes with fine-grained control over their appearance. We observe that, despite being trained on realistic renderings and a multitude of objects, state-of-the-art methods still struggle to generalize to real-world images. Our analysis also indicates the benefit of introducing more structure into the computational pipeline via making an intermediate geometric representation part of its design.

ZUSAMMENFASSUNG

In den letzten Jahren haben die Deep Convolutional Neural Networks (CNNs) großartige Ergebnisse in den meisten 2D Bildverarbeitungsaufgaben erzielt. Bei vielen Anwendungen ist es wichtig direkt auf 3D Daten zu arbeiten und nicht auf ihren 2D Projektionen. Die einfachsten Erweiterungen von 2D CNNs in den 3D Raum basieren auf Voxel Grids, die wegen ihrer kubischer Komplexität sehr ineffizient sind und daher nur geringe Auflösungen unterstützen. Es ist demzufolge sehr wichtig spezielle 3D Deep Learning Algorithmen zu entwickeln, die auf effizienten 3D Repräsentationen funktionieren, wie zum Beispiel auf Punktwolken und Octrees. Das Design von diesen Algorithmen ist vom Aufgabentyp abhängig: Methoden für die 3D Analyse erhalten die 3D Daten als Eingabe und müssen sie effizient bearbeiten; im Fall von 3D Synthese geht es darum, ein 3D Modell von einer abstrakten Repräsentation zu erzeugen.

Im ersten Teil dieser Arbeit stellen wir neu entwickelte CNN-basierte Methoden für eine effiziente 3D Analyse und 3D Synthese vor. Bei der Analyse ist unsere Aufgabe die semantische Segmentierung von Punktwolken. Wir schlagen Tangent Convolutions vor, eine neue Operation, die auf der Oberfläche vom Objekten definiert ist und als Baustein für effiziente CNNs verwendet werden kann. Wir verwenden Netzwerke, die Tangent Convolutions nutzen für die semantische Segmentierung von weiträumigen 3D Szenen, die im Innenbereich und in der freien Umgebung aufgenommen wurden. Für die Synthese entwickeln wir einen neuen convolutional Decoder. Dieser übersetzt eine abstrakte Repräsentation in eine vom Menschen leicht zu verstehende Darstellung, in unserem Fall ein Octree. Unsere Methode ist wesentlich effizienter in der Rechenzeit und im Speicherverbrauch als Netzwerke, die intern Voxel Grids als Datenstruktur verwenden. Das ermöglicht die Generierung von hochauflösenden 3D Modellen. Wir stellen Experimente für drei verschiedene Aufgabenbereiche vor: Auto-Encoding von 3D Modellen, Modellerzeugung aus einer abstrakten Repräsentation und die 3D Rekonstruktion aus einem einzelnen Bild.

Im zweiten Teil der Arbeit evaluieren wir aktuellste Methoden, die sich mit der 3D Rekonstruktion von Objekten aus einzelnen Bildern befassen. Viele moderne Verfahren legen den Schwerpunkt auf die Entwicklung von speziellen Decodern für verschiedene 3D Repräsentationen. Wir führen eine systematische Analyse dieser bestehenden Methoden durch, indem wir sie mit zwei Referenzmethoden vergleichen, die die Aufgabe mit Hilfe von Bildklassifizierung und Retrieval lösen. Die Ergebnisse dieser Baselines, die ausschließlich auf Wiedererkennung bekannter Objekte basieren, sind von den Ergebnissen der viel komplexeren Encoder-Decoder Netzwerken statistisch ununterscheidbar. Das deutet darauf hin, dass die letztgenannten Netzwerke das Problem intern auch als Erkennung angehen. Wir formulieren die Probleme im konventionellen Versuchsaufbau, die zu solchem Verhalten führen, und schlagen mögliche Lösungen vor.

Als Letztes beschäftigen wir uns mit der Frage, wie gut die existierenden Methoden, die auf synthetischen Daten trainiert wurden, auf echten Bildern generalisieren. Dafür haben wir einen neuen Datensatz gesammelt, der aus einer Vielzahl von 3D Modellen und deren realistischen Renderings besteht. Trotz realistischen Aussehens und der Vielfalt an Daten generalisieren die auf unserem Datensatz trainierten state-of-the-art Methoden immer noch nicht auf echten Daten. Unsere Analyse deutet darauf hin, dass die Einführung von zwischenliegenden geometrischen Repräsentationen als Teil der Pipeline, eine positive Wirkung auf die Ergebnisse haben kann.

PUBLICATIONS

[1] Anton Böhm, Maxim Tatarchenko, and Thorsten Falk. “$ISOO_{DL}^{V2}$ - Semantic Instance Segmentation of Touching and Overlapping Objects.” In: *ISBI*. 2019.

[2] Alexey Dosovitskiy, Jost T. Springenberg, Maxim Tatarchenko, and Thomas Brox. “Learning to Generate Chairs, Tables and Cars with Convolutional Networks.” In: *TPAMI* 39.4 (2017), pp. 692–705.

[3] Barbara Frank, Michael Ruhnke, Maxim Tatarchenko, and Wolfram Burgard. “3D-Reconstruction of Indoor Environments from Human Activity.” In: *ICRA*. 2015.

[4] Oier Mees, Maxim Tatarchenko, Thomas Brox, and Wolfram Burgard. “Self-supervised 3D Shape and Viewpoint Estimation from Single Images for Robotics.” In: *IROS*. 2019.

[5] Sudhanshu Mittal, Maxim Tatarchenko, and Thomas Brox. “Semi-Supervised Semantic Segmentation with High- and Low-level Consistency.” In: *TPAMI* (2020).

[6] Sudhanshu Mittal, Maxim Tatarchenko, Özgün Çiçek, and Thomas Brox. “Parting with Illusions about Deep Active Learning.” In: *arXiv:1912.05361* (2019).

[7] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Multi-view 3D Models from Single Images with a Convolutional Network.” In: *ECCV*. 2016.

[8] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs.” In: *ICCV*. 2017.

[9] Maxim Tatarchenko*, Jaesik Park*, Vladlen Koltun, and Qian-Yi Zhou. “Tangent Convolutions for Dense Prediction in 3D.” In: *CVPR*. 2018.

[10] Maxim Tatarchenko*, Stephan R. Richter*, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. “What Do Single-view 3D Reconstruction Networks Learn?” In: *CVPR*. 2019.

ACKNOWLEDGMENTS

Working on a PhD thesis is a process of constant learning and growing. I consider myself lucky to have met so many inspiring people who taught me a great deal along the way, and I want to express my gratitude to all of them.

First of all, I would like to thank my advisor Thomas Brox. You were the best instructor for all aspects of scientific work. More than that, you were truly a *Doktorvater* providing invaluable support in all situations: from giving me a chance to pursue my teaching inclination to assisting with my career decisions.

I thank Alexey Dosovitskiy for being my mentor during the first part of my PhD program. With your constant streams of amazing ideas, thoughtful insights, and endless optimism you were a scientific role model for me.

I want to thank all of my great LMB colleagues. I can barely imagine a better working environment than we have had in the lab: hot scientific discussions, hard work and late hours balanced out by table football, laser tag and our field trip adventures. A special thank you goes to Jasmin Anders for her help with all bureaucratic questions and for amazing discussions full of vibrant optimism and all-round supportiveness. Finally, I want to thank Stefan Teister, Nikolaus Mayer, Philipp Schröppel, Felix Ratzkowski, and Marc Michel for keeping our computational cluster alive and always lending a helping hand with any technical issues.

I thank all the colleagues from the Intel Intelligent Systems Lab, who I worked with first during my internship and later on, during the subsequent NIS projects. Thank you to Vladlen Koltun for being a motivating supervisor: you taught me a lot about structured scientific thinking and careful attention to detail. Thank you to Stephan Richter, Renè Ranftl, Qian-Yi Zhou, and Jaesik Park for exciting collaborations. It was thrilling to work with you.

I want to express my deepest gratitude to my dear friends Igor, Olga, Marianna, Ira, Oier, Alex, Anton, and Olesya. You guys are my virtuous horcruxes.

I deeply thank my mom and grandmom for their profound love and unconditional support in any imaginable situation. Without it, I would have never ended up where I am now.

Finally, I would love to thank my wife Dasha for sharing this adventure with me. With you, I see how much more there is to life than just science.

CONTENTS

1 INTRODUCTION

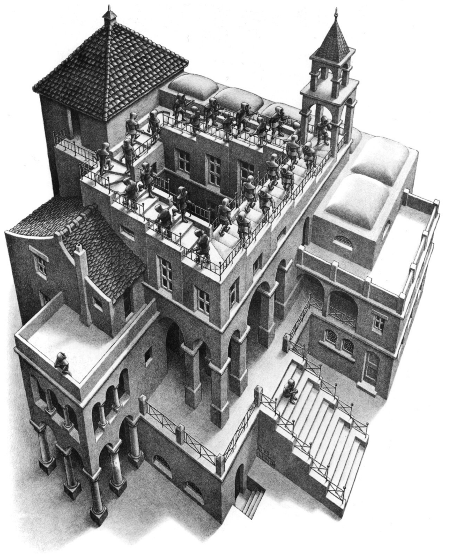


Figure 1.1: "Ascending and Descending" by M. C. Escher depicting an infinite ladder which looks plausible in 2D but is actually impossible in 3D.

We live in three-dimensional space which we perceive primarily visually. Therefore, robust processing of this visual information is crucially important for making decisions and performing actions - both for humans and for AI agents. Computer vision algorithms developed during the last decades have achieved remarkable success in visual understanding. However, most of them operate on 2D images which result from projecting the original 3D information onto a plane. This projection step leads to an inevitable information loss exemplified in Figure 1.1: Escher depicted an infinite ladder which is obviously not possible in 3D but looks plausible in 2D, thanks to the carefully chosen projection angle.

Incorporating knowledge about the structure of the 3D world into machine learning systems is important in various situations. 3D data can be used explicitly by autonomous robots to observe the surrounding environment. It can be semantically analyzed to understand the structure of a scene and act in it. A more subtle use of 3D data stems from the human ability to build implicit cognitive representations (i.e. priors) of the 3D structure of the world. These priors can be used later, even without having access to explicit 3D observations. For example, having seen many cars during the course of their lives, most humans would be able to estimate the most likely side-view of a car given a picture taken from its back. Incorporating a similar kind of prior 3D knowledge into machine learning systems could provide them with deeper understanding of the world and make solving higher-level tasks possible. Naturally, exploiting 3D information in machine learning systems requires the development of dedicated tools that operate on 3D data.

In recent years, the landscape of computer vision has been changed by algorithms based on Convolutional Neural Networks (CNNs). CNNs are particularly capable in two aspects: automatically extracting features useful for solving the task at hand and learning strong high-level priors from data. They have set a new performance bar on classical computer vision tasks including image classification [55, 86, 155],

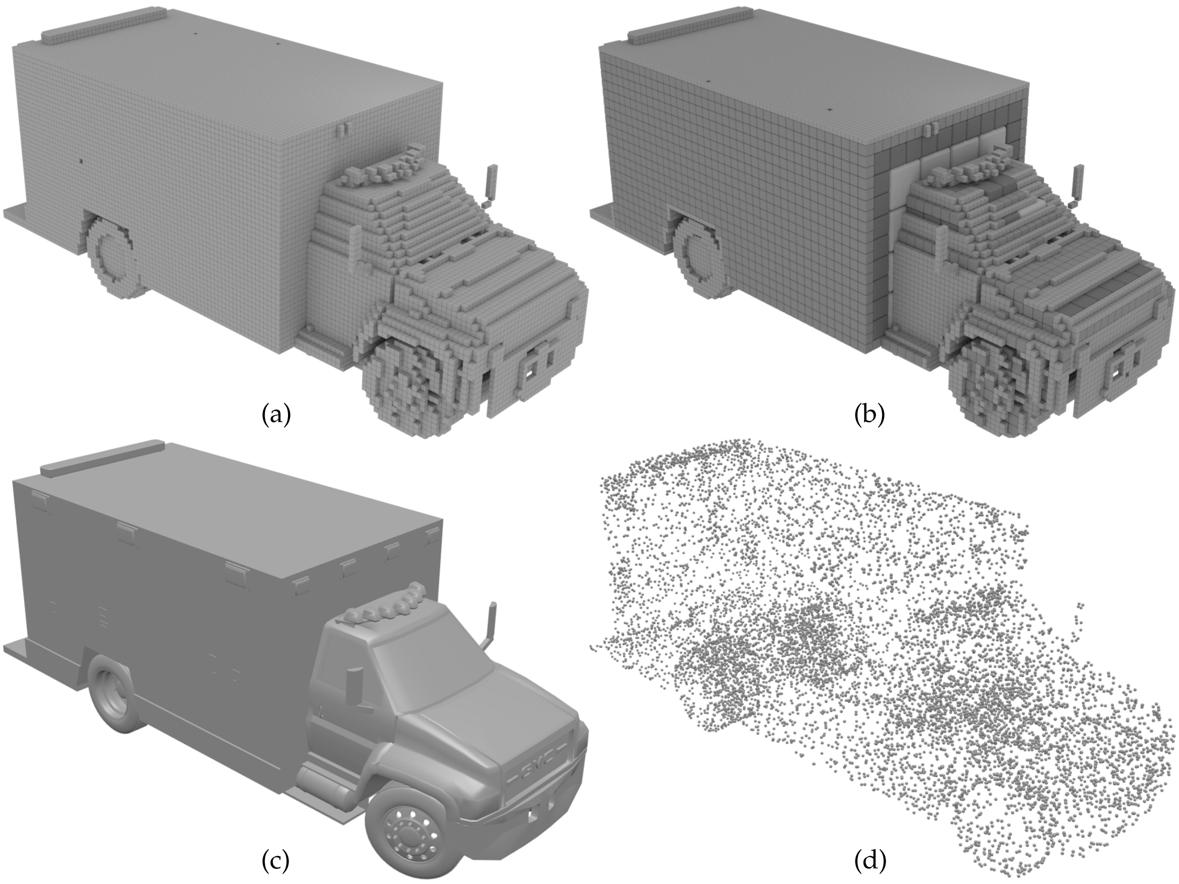


Figure 1.2: Different 3D representations of the same shape: dense voxel grid (a), octree (b), mesh (c), and point cloud (d).

semantic segmentation [102, 143], and optical flow [29, 65]. They have also enabled exciting new applications, such as image and video generation [28, 134], monocular depth estimation [31, 100], and visual question answering [205].

These two properties (automatic feature extraction and data-based priors), as well as the overall success of 2D networks, make CNNs a method of choice for 3D tasks. Technically, the most straightforward way to extend 2D networks to 3D is to extend the internal data representation from a 2D image to a 3D image (a.k.a. voxel grid) and substitute all 2D operations with the corresponding 3D ones. Networks of this kind, while applicable in certain situations [20, 183], turn out to be suboptimal for general-purpose 3D deep learning. A key contributing factor to the success of 2D CNNs is efficient local processing based on the convolution operation. Naively replacing 2D grids with 3D grids and 2D convolutions with 3D convolutions in the processing pipeline changes its scaling from quadratic to cubic, both in computation time and in memory consumption. This makes the resulting networks applicable only to low-resolution voxel grids (approximately up to 64^3) on modern hardware, which is not sufficient for many tasks that require a larger context.

Such a naive technical design, however, does not use a fundamental property of 3D data: its sparsity. Typical 3D sensors (LiDAR, RGB-D camera[1]) measure the distance to an object's surface[2]. This effectively means that only a 2D submanifold of the 3D space needs to be encoded. There are multiple 3D representations that exploit this sparsity: octrees, point clouds, and meshes.

An octree is effectively a voxel grid with locally adaptive resolution. It automatically maintains the appropriate resolution depending on the amount of geometric information in a particular space region: areas close to the surface are usually encoded in high resolution, while large empty spaces are represented in low resolution. A point cloud represents a 3D shape as a collection of points, situated on the object surface. Meshes encode surfaces represented as point sets interconnected with polygons, usually triangles. An example 3D shape represented in four different ways is shown in Figure 1.2.

While all these representations are significantly more efficient than vanilla voxel grids, they are defined on irregular domains. Therefore, developing CNNs that operate on them requires re-designing all the basic computational building blocks: convolution, up-convolution, pooling, etc. In practice there is no single universal representation which would outperform the others – all three have their use in different situations depending on the application. Therefore, designing a CNN for a particular efficient 3D representation requires taking into account both the properties of this representation and of the particular type of task.

All tasks in deep learning, both 2D and 3D, can be broadly subdivided into analysis and synthesis. In case of analysis, a visual input is converted into a class label, a per-pixel segmentation mask, an optical flow field, etc. In case of synthesis, the process is reversed. Given some input representation, the task of the network is to reconstruct the original visual data. When working with 2D images, there is no fundamental technical difference between these two tasks. Networks for semantic segmentation – an analysis task – can be implemented using the same encoder-decoder architecture as networks for image synthesis, because both inputs and outputs of such networks are represented as 2D grids.

There is, however, an important difference between these two types of tasks in 3D, when using one of the previously described efficient representations. In case of analysis, the 3D geometry is given as input, which effectively defines the structure of the domain. The essence of the algorithm design is therefore to enable efficient operations in this domain. One of the most practically relevant classes of analysis tasks is that of processing data from 3D sensors which usually comes in form of point clouds. In Chapter 2, we present an efficient method for dense prediction on unstructured point clouds and other noisy real-world data. The core of our approach is a new convolution operation for 3D data - tangent convolution. Unlike

[1] Strictly speaking, RGB-D cameras are 2.5D sensors. However, we assume that, given the known camera poses, multiple RGB-D images can be fused into a complete 3D scene in a fairly straightforward manner.

[2] One counterexample includes bio-medical imaging, where samples are often recorded layer-by-layer and encode dense volumetric information.

volumetric approaches relying on voxel grids, ours operates on the surface, i.e. precisely where the information from typical 3D sensors is located. Evaluating tangent convolutions is efficient even for point clouds with millions of points. We design a deep fully-convolutional network for semantic segmentation of 3D point clouds using tangent convolution as the main building block. This network is evaluated on three large-scale real-world dataset comprised of indoor and outdoor scenes. Experiments confirm that our method outperforms all prior CNN-based approaches for semantic segmentation in 3D.

Synthesis, on the other hand, requires generating a 3D geometry from a low-dimensional representation. The task of the network is therefore to not only make some predictions about a signal (e.g. occupancy or color) defined in a certain domain, but also to estimate the structure of the domain itself. Octrees are particularly suitable for this class of problems because of their hierarchical design which naturally enables a hierarchical structure of the computational pipeline. In Chapter 3, we introduce a convolutional decoder architecture for generating 3D shapes represented as octrees. The network predicts both the structure of the octree and the signal - in this case occupancy - for individual cells. Our network is compute- and memory-efficient; it enjoys approximately quadratic scaling compared to the cubic scaling of approaches internally operating on dense voxel grids. This allows to generate shapes of significantly higher resolution. Our decoder can be combined with a suitable task-specific encoder. We demonstrate its performance on several tasks: shape auto-encoding, generating objects from high-level information, and single-view 3D reconstruction.

The goal of single-view 3D reconstruction, one of the classical [142] tasks in 3D computer vision, is to predict the 3D shape of an object given its single image. This problem formulation can be useful to learn an implicit representation which contains 3D information. It can also be used to explicitly reason about the 3D geometry, when only little information (i.e. a single camera image) is available. The problem of single-view 3D reconstruction is inherently ill-posed: it is impossible to reliably reconstruct parts of objects which are completely occluded in the input image. The overall task boils down to interpreting the information about the 3D structure present in the input image (lighting, textures, perspective effects, etc.) and complementing it by employing some form of data prior. We refer to solutions using these two sources of information as reconstruction and recognition respectively. A reconstruction-type solution is based on interpreting the local low-level image observations, while a recognition-type solution relies on classifying the object in the input and retrieving the appropriate prior shape. Classical algorithms mostly rely on hand-designed shape priors, which limits their applicability to a narrow range of simplistic objects [61, 142]. CNNs, with their ability to automatically extract priors from data, enable a novel class of learning approaches.

Research in the field has mostly concentrated on developing methods that predict different output representations [20, 33, 48, 138]. All existing techniques, including our method presented in Chapter 3, are united by the idea of having an encoder-decoder network that performs non-trivial reasoning about the 3D structure of the output space. The ability of 3D CNNs to reliably combine local input observations with the data priors has been assumed by default. In Chapter 4, we challenge this assumption and perform extensive analysis of existing CNN-based methods for single-view 3D reconstruction. To this end, we design two alternative approaches for solving the task which rely on image classification and retrieval respectively. Though only relying on object recognition, these simple baselines perform on par with the state-of-the-art methods both qualitatively and quantitatively. The results of our baselines are statistically indistinguishable from those of encoder-decoder CNNs which explicitly reason about the structure of the 3D space. This indicates that state-of-the-art methods largely rely on object recognition. We discuss which flaws of the conventional experimental setup lead to such behavior and outline possible remedies.

In Chapter 5, we continue to analyze the performance of methods for single-view 3D reconstruction, assessing their generalization to real-world data. One of the big issues in machine learning in general – and in single-view 3D reconstruction in particular – is the domain shift which emerges when training methods on synthetic data and testing on real data. Solving this is crucial for making these methods applicable in the real world. We collect a large-scale synthetic dataset and use it to systematically analyze the aforementioned domain shift. Our dataset features shapes of diverse types from three different collections: the most relevant ShapeNet models, lego shapes, and sculptures. We produce high-quality realistic renderings of all shapes while carefully controlling the appearance of objects. In our experiments, we observe that methods which generate shapes directly from RGB images barely generalize to real-world data, even despite being trained on realistically looking objects and diverse shapes. The situation improves when using an intermediate geometric representation, i.e. a depth map, in the pipeline, which indicates the potential general importance of such design choice.

Chapter 6 concludes the thesis and provides an outline of potential future research directions.

CONTRIBUTIONS

The contributions of this thesis are based on the following papers. A detailed description of how exactly the papers were used, as well as an account of other people's contributions, are provided at the beginning of the corresponding chapters.

- Chapter 2: Maxim Tatarchenko*, Jaesik Park*, Vladlen Koltun, and Qian-Yi Zhou (*indicates equal contribution). "Tangent Convolutions for Dense Prediction in 3D." In: *CVPR*, 2018.

  In this work we focus on semantic analysis of 3D scenes with convolutional networks. We propose a novel convolution operation suitable for processing unstructured 3D data - tangent convolution. Tangent convolution operates directly on the object surface and only requires being able to estimate surface normals from the input data. CNNs based on tangent convolutions can be used to solve dense prediction tasks on large-scale point clouds with millions of points. We validate the proposed approach on the task of semantic segmentation of indoor and outdoor scenes from three real-world datasets.

- Chapter 3: Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs." In: *ICCV*, 2017.

  In this work we propose a novel convolutional decoder for generating 3D shapes represented as octrees. Compared to CNNs based on dense voxel grids, our method has much better scaling properties, which allows to produce shapes of significantly higher resolution. Combined with a suitable encoder, our decoder can be used to solve any task which requires generating a 3D shape. We showcase its performance in three settings: shape auto-encoding, generating shapes from high-level information, and singe-view 3D reconstruction.

- Chapter 4: Maxim Tatarchenko*, Stephan R. Richter*, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox (*indicates equal contribution). "What Do Single-view 3D Reconstruction Networks Learn?" In: *CVPR*, 2019.

  In this work we perform extensive analysis of modern encoder-decoder CNNs for single-view 3D reconstruction in order to better understand their mode of operation. We develop two alternative baselines which perform image classification and retrieval respectively. Though both these baselines exclusively rely on recognition and do not include any explicit 3D reasoning, their results turn out to be statistically indistinguishable from those produced by encoder-decoder CNNs. This provides a strong indication that state-of-the-art methods primarily approach the task of single-view shape estimation in the recognition mode. We also formulate a set of problems which cause such behavior and discuss their possible solutions.

- Chapter 5: Maxim Tatarchenko, Stephan R. Richter, Jaesik Park, Vladlen Koltun, and Thomas Brox. "Do Single-view 3D Reconstruction Networks Generalize to Real Data?" In preparation for submission.

  In this work we further analyze the functioning of single-view 3D reconstruction methods focusing on their ability to generalize to real-world input data. We systematically study the domain shift experienced by CNNs trained on synthetic data and tested on real images. For this purpose, we collected a large-scale dataset featuring objects of different shapes and produced their high-quality realistic renderings. We conclude that, despite realistic appearance and diverse training data, existing methods still struggle with generalization to real data. We provide an indication that this effect can be alleviated by using intermediate geometric representations in the pipeline.

2 3D ANALYSIS: TANGENT CONVOLUTIONS

The text of this chapter was largely copied from the following paper.

> Maxim Tatarchenko*, Jaesik Park*, Vladlen Koltun, and Qian-Yi Zhou (*indicates equal contribution). "Tangent Convolutions for Dense Prediction in 3D." In: *CVPR*, 2018.

Jaesik Park contributed by implementing the low-level geometric processing operations in the Open3D framework. He also performed the training of the baseline methods: ScanNet, OctNet, PointNet and SnapNet. All co-authors contributed to the project discussions as well as the final paper text editing.

* * *

2.1 INTRODUCTION

Methods that utilize convolutional networks on 2D images dominate modern computer vision. A key contributing factor to their success is efficient local processing based on the convolution operation. 2D convolution is defined on a regular grid, a domain that supports extremely efficient implementation. This in turn enables using powerful deep architectures for processing large datasets at high resolution.

When it comes to analysis of large-scale 3D scenes, a straightforward extension of this idea is volumetric convolution on a voxel grid [25, 115, 188]. However, voxel-based methods have limitations, including a cubic growth rate of memory consumption and computation time. For this reason, voxel-based ConvNets operate on low-resolution voxel grids that limit their prediction accuracy. The problem can be alleviated by octree-based techniques that define a ConvNet on an octree and enable processing somewhat higher-resolution volumes (e.g., up to 256^3 voxels) [53, 140, 141, 177]. Yet even this may be insufficient for detailed analysis of large-scale scenes.

On a deeper level, both efficient and inefficient voxel-based methods treat 3D data as *volumetric* by exploiting 3D convolutions that integrate over volumes. In reality, data captured by 3D sensors such as RGB-D cameras and LiDAR typically represent *surfaces*: 2D structures embedded in 3D space. (This is in contrast to truly volumetric 3D data, as encountered for example in medical imaging.) Classic features that are used for the analysis of such data are defined in terms that acknowledge the latent surface structure, and do not treat the data as a volume [36, 70, 144].

The drawbacks of voxel-based methods are known in the research community. A number of recent works argue that volumetric data structures are not the natural

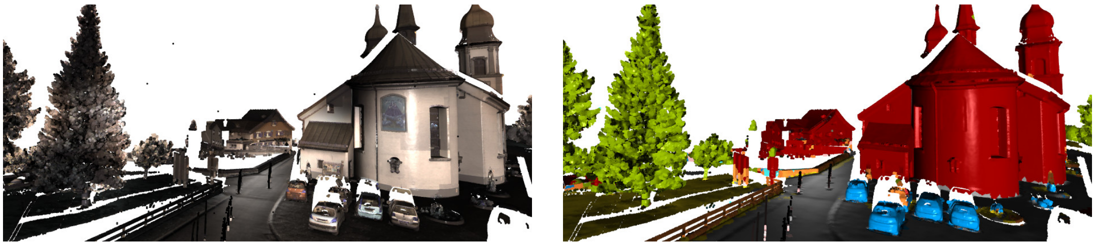


Figure 2.1: Convolutional networks based on tangent convolutions can be applied to semantic analysis of large-scale scenes, such as urban environments. Left: point cloud from the Semantic3D dataset. Right: semantic segmentation produced by the presented approach.

substrate for 3D ConvNets, and propose alternative designs based on unordered point sets [132], graphs [154], and sphere-type surfaces [110].

We develop an alternative construction for convolutional networks on surfaces, based on the notion of *tangent convolution*. This construction assumes that the data is sampled from locally Euclidean surfaces. The latent surfaces need not be known, and the data can be in any form that supports approximate normal vector estimation, including point clouds, meshes, and even polygon soup. (The same assumption concerning normal vector estimation is made by both classic and contemporary geometric feature descriptors [36, 70, 78, 144, 146, 167].) The tangent convolution is based on projecting local surface geometry on a tangent plane around every point. This yields a set of tangent images. Every tangent image is treated as a regular 2D grid that supports planar convolution. The content of all tangent images can be precomputed from the surface geometry, which enables efficient implementation that scales to large datasets, such as urban environments.

Using tangent convolution as the main building block, we design a U-type network for dense semantic segmentation of point clouds. Our proposed architecture is general and can be applied to analysis of large-scale scenes. We demonstrate its performance on three diverse real-world datasets containing indoor and outdoor environments. A semantic segmentation produced by a tangent convolutional network is shown in Figure 2.1.

2.2 RELATED WORK

Dense prediction in 3D, including semantic point cloud segmentation, has a long history in computer vision. Pioneering methods work on aerial LiDAR data and are based on hand-crafted features with complex classifiers on top [12, 13, 44]. Such approaches can also be combined with high-level architectural rules [111]. A popular line of work exploits graphical models, including conditional random fields [2, 35, 57, 88, 120, 174, 182]. Related formulations have also been proposed for interactive 3D segmentation [119, 172].

More recently, the deep learning revolution in computer vision has spread to consume 3D data analysis. A variety of methods that tackle 3D data using deep learning techniques have been proposed. They can be considered in terms of the underlying data representation.

A common representation of 3D data for deep learning is a voxel grid. Deep networks that operate on voxelized data have been applied to shape classification [115, 130, 188], semantic segmentation of indoor scenes [25], and biomedical recordings [15, 22]. Due to the cubic complexity of voxel grids, these methods can only operate at low resolution – typically not more than 64^3 – and have limited accuracy. Attempting to overcome this limitation, researchers have proposed representations based on hierarchical spatial data structures such as octrees and kd-trees [80, 90, 140, 177], which are more memory- and computation-efficient, and can therefore handle higher resolutions. A related family of approaches operates on sparse quantizations of input data [21, 45, 161]. These methods are based on storing and processing information only in relevant spatial regions, that is around object surfaces. An alternative way of increasing the accuracy of voxel-based techniques is to add differentiable post-processing, modeled upon the dense CRF [85, 164].

Several other constructs for convolving point clouds were proposed in the recent years. Hua *et al.* [63] and Li *et al.* [96] rely on the underlying grid for neighborhood computation required to define a convolutional kernel. Xu *et al.* [190] parametrize convolutional filters as a product of a step function for capturing local geodesic information and a Taylor polynomial for representing geometric variations. Groh *et al.* [47] and Wang *et al.* [178] represent convolutional kernels as parametric functions of neighborhood points. Hermozilla *et al.* [58] treat convolution as a Monte Carlo integration problem. Wu *et al.* [186] introduce PointConv where they represent kernel weights as functions of the local point coordinates and point densities parametrized by MLPs. Thomas *et al.* [165] and Atzmon *et al.* [4] associate kernel weights with points and use a correlation function. Zhang *et al.* [199] propose rotation invariant convolutions on point clouds based on low-level geometric features. Mao *et al.* [109] define convolutions on a regular grid and interpolate point features to neighboring kernel weights. Zhang *et al.* [198] use statistics from concentric shells to extract point cloud features and process those with regular convolutions. A similar idea with ring-shaped structures was proposed by Komarichev *et al.* [83].

Qi *et al.* [132] propose a network for analyzing unordered point sets, which is based on independent point processing combined with global context aggregation through max-pooling. In a subsequent work [131] they extend their approach to support local hierarchical processing which makes it more suitable for segmenting large-scale scenes. Rethage *et al.* [136] combine a PointNet-based feature extractor with a regular 3D convolutional network for semantic segmentation. Zhao *et al.* [200] extend the idea of capsule networks [145] to the 3D case and apply the resulting construct to multiple tasks including semantic segmentation. Huang *et al.* [64] process textured meshes parametrized with a consistent 4-way rotationally symmetric field. Hanocka *et al.* [54] implement mesh convolutions by using edges for neighborhood computation. Several approaches adapt graph networks to point cloud analysis tasks [69, 175, 179].

Other applications of CNNs consider RGB-D images, which can be treated with fully-convolutional networks [49, 97, 116] and graph neural networks [133]. These approaches support the use of powerful pre-trained 2D networks, but are not generally applicable to unstructured point clouds with unknown sensor poses. Attempting to address this issue, Boulch *et al.* [8] train a ConvNet on images rendered from point clouds using randomly placed virtual cameras. In a more controlled setting with fixed camera poses, multi-view methods are successfully used for shape segmentation [73] and shape recognition [130, 162]. Chiang *et al.* [18] and Dai *et al.* [24] combine multiple sources of information for segmentation including 2D image features, 3D features and global context features. Our approach can be viewed as an extreme multi-view approach in which a virtual camera is associated with each point in the point cloud. A critical problem that we address is the efficient and scalable implementation of this approach, which enables its application to dense point clouds of large-scale indoor and outdoor environments.

There is a variety of more exotic deep learning formulations for 3D analysis that do not address large-scale semantic segmentation of whole scenes but provide interesting ideas. Yi *et al.* [193] consider shape segmentation in the spectral domain by synchronizing eigenvectors across models. Masci *et al.* [112] and Boscaini *et al.* [7] design ConvNets for Riemannian manifolds and use them to learn shape correspondences. Sinha *et al.* [157] perform shape analysis on geometry images. Simonovsky *et al.* [154] extend the convolution operator from regular grids to arbitrary graphs and use it to design shape classification networks. Li *et al.* [95] introduce Field Probing Neural Networks which respect the underlying sparsity of 3D data and are used for efficient feature extraction. Maron *et al.* [110] design ConvNets on surfaces for sphere-type shapes. Haim *et al.* [51] propose warping an image around a surface and applying regular CNN architectures on this representation.

2.3 APPROACH

In this section we formally introduce tangent convolutions. All derivations are provided for point clouds, but they can easily be applied to any type of 3D data that supports surface normal estimation, such as meshes.

Convolution with a continuous kernel. Let $\mathcal{P} = \{\mathbf{p}\}$ be a point cloud, and let $F(\mathbf{p})$ be a discrete scalar function that represents a signal defined over $\mathcal{P}$. $F(\mathbf{p})$ can encode color, geometry, or abstract features from intermediate network layers. In order to convolve F, we need to extend it to a continuous function. Conceptually, we introduce a virtual orthogonal camera for $\mathbf{p}$. It is configured to observe $\mathbf{p}$ along the normal $\mathbf{n_p}$. The image plane of this virtual camera is the tangent plane $\pi_\mathbf{p}$ of $\mathbf{p}$. It parameterizes a virtual image that can be represented as a continuous signal $S(\mathbf{u})$, where $\mathbf{u} \in \mathbb{R}^2$ is a point in $\pi_\mathbf{p}$. We call S a tangent image.

The tangent convolution at $\mathbf{p}$ is defined as

$$X(\mathbf{p}) = \int_{\pi_\mathbf{p}} c(\mathbf{u}) S(\mathbf{u}) \, d\mathbf{u}, \qquad (2.1)$$

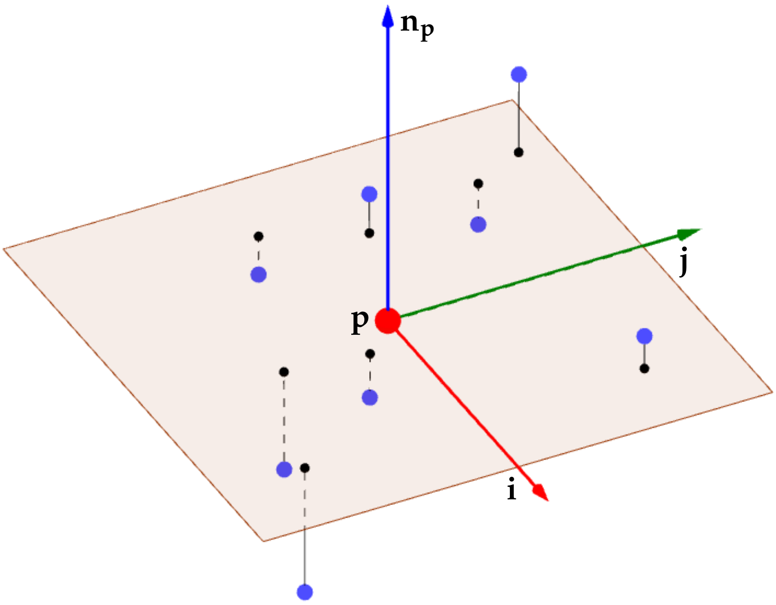


Figure 2.2: Points $\mathbf{q}$ (blue) from the local neighborhood of a point $\mathbf{p}$ (red) are projected onto the tangent image.

where $c(\mathbf{u})$ is the convolution kernel. We now describe how S is computed from F.

Tangent plane estimation. For each point $\mathbf{p}$ we estimate the orientation of its camera image using local covariance analysis. This is a standard procedure [146] but we summarize it here for completeness. Consider a set of points $\mathbf{q}$ from a spherical neighborhood of $\mathbf{p}$, such that $\|\mathbf{p} - \mathbf{q}\| < R$. The orientation of the tangent plane is determined by the eigenvectors of the covariance matrix $\mathbf{C} = \sum_{\mathbf{q}} \mathbf{r}\mathbf{r}^\top$, where $\mathbf{r} = \mathbf{q} - \mathbf{p}$. The eigenvector of the smallest eigenvalue defines the estimated surface normal $\mathbf{n_p}$, and the other two eigenvectors $\mathbf{i}$ and $\mathbf{j}$ define the 2D image axes that parameterize the tangent image.

Signal interpolation. Now our goal is to estimate image signals $S(\mathbf{u})$ from point signals $F(\mathbf{q})$. We begin by projecting the neighbors $\mathbf{q}$ of $\mathbf{p}$ onto the tangent image, which yields a set of projected points $\mathbf{v} = (\mathbf{r}^\top \mathbf{i}, \mathbf{r}^\top \mathbf{j})$. This is illustrated in Figure 2.2. We define

$$S(\mathbf{v}) = F(\mathbf{q}). \tag{2.2}$$

As shown in Figure 2.2 and Figure 2.3(a), points $\mathbf{v}$ are scattered on the image plane. We thus need to interpolate their signals in order to estimate the full function $S(\mathbf{u})$ over the tangent image:

$$S(\mathbf{u}) = \sum_{\mathbf{v}} \left(w(\mathbf{u}, \mathbf{v}) \cdot S(\mathbf{v}) \right), \tag{2.3}$$

where $w(\mathbf{u}, \mathbf{v})$ is a kernel weight that satisfies

$$\sum_{\mathbf{v}} w = 1. \tag{2.4}$$

We consider two schemes for signal interpolation: nearest neighbor and Gaussian kernel mixture. These schemes are illustrated in Figure 2.3.

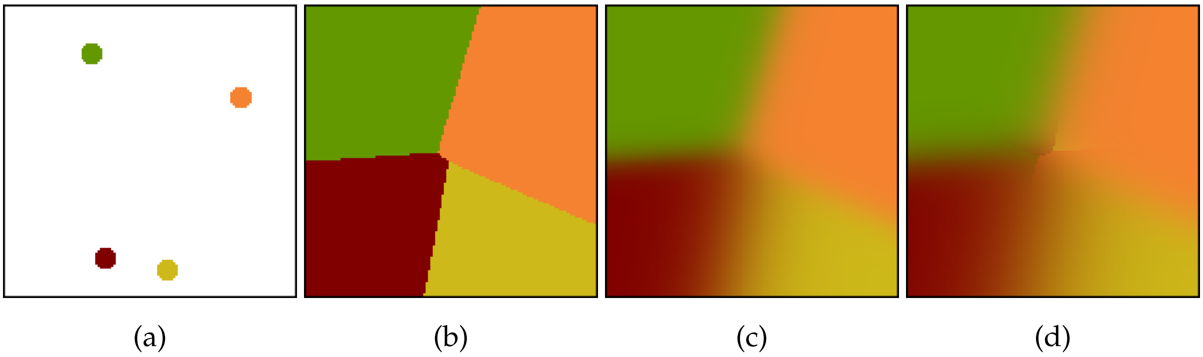


Figure 2.3: Signals from projected points (a) can be interpolated using one of the following schemes: nearest neighbor (b), full Gaussian mixture (c), and Gaussian mixture with top-3 neighbors (d).

In the nearest neighbor (NN) case,

$$w(\mathbf{u},\mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v} \text{ is } \mathbf{u}\text{'s NN,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

In the Gaussian kernel mixture case,

$$w(\mathbf{u},\mathbf{v}) = \frac{1}{A} \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\sigma^2} \right), \tag{2.6}$$

where A normalizes the weights such that $\sum_{\mathbf{v}} w = 1$. More sophisticated signal interpolation schemes can be considered, but we have not observed a significant effect of the interpolation scheme on empirical performance and will mostly use simple nearest-neighbor estimation.

Finally, if we rewrite Equation (2.1) using the definitions from Equations (2.2) and (2.3), we get the formula for the tangent convolution:

$$X(\mathbf{p}) = \int_{\pi_{\mathbf{p}}} c(\mathbf{u}) \cdot \sum_{\mathbf{v}} \left(w(\mathbf{u},\mathbf{v}) \cdot F(\mathbf{q}) \right) d\mathbf{u}. \tag{2.7}$$

Note that the role of the tangent image is increasingly implicit: it provides the domain for $\mathbf{u}$ and figures in the evaluation of the weights w, but otherwise it need not be explicitly maintained. We will build on this observation in the next section to show that tangent convolutions can be evaluated efficiently at scale, and can support the construction of deep networks on point clouds with millions of points.

2.4 EFFICIENCY

In this section we describe how the tangent convolution defined in Section 2.3 can be computed efficiently. In practice, the tangent image is treated as a discrete function on a regular $l \times l$ grid. Elements $\mathbf{u}$ are pixels in this virtual image. The convolution kernel c is a discrete kernel applied onto this image. Let us first consider the nearest-neighbor signal interpolation scheme introduced in Equation (2.5). We can rewrite Equation (2.7) as

$$X(\mathbf{p}) = \sum_{\mathbf{u}} \Big(c(\mathbf{u}) \cdot F\big(g(\mathbf{u})\big) \Big), \tag{2.8}$$

where $g(\mathbf{u})$ is a selection function that returns a point which projects to the nearest neighbor of $\mathbf{u}$ on the image plane. Note that g only depends on the point cloud geometry and does not depend on the signal F. This allows us to precompute g for all points.

From here on, we employ standard ConvNet terminology and proceed to show how to implement a convolutional layer using tangent convolutions. Our goal is to convolve an input feature map $\mathbf{F}_{in}$ of size $N \times C_{in}$ with a set of weights W to produce an output feature map $\mathbf{F}_{out}$ of size $N \times C_{out}$, where N is the number of points in the point cloud, while C_{in} and C_{out} denote the number of input and output channels respectively. For implementation, we unroll 2D tangent images and convolutional filters of size $l \times l$ into 1D vectors of size $1 \times L$, where $L = l^2$. From then on, we compute 1D convolutions. Note that such representation of a 2D tangent convolution as a 1D convolution is not an approximation: the results of the two operations are identical.

We start by precomputing the function g, which is represented as an $N \times L$ index matrix $\mathbf{I}$. Elements of $\mathbf{I}$ are indices of the corresponding tangent-plane nearest-neighbors in the point cloud. Using $\mathbf{I}$, we gather input signals (features) into an intermediate tensor $\mathbf{M}$ of size $N \times L \times C_{in}$. This tensor is convolved with a flattened set of kernels W of size $1 \times L$, which yields the output feature map $\mathbf{F}_{out}$. This process is illustrated in Figure 2.4.

Consider now the case of signal interpolation using Gaussian kernel mixtures. For efficiency, we only consider the set of top-k neighbors for each point, denoted NN_k. An example image produced using the Gaussian kernel mixture scheme with top-3 neighbors is shown in Figure 2.3(d). Equation (2.6) turns into

$$w(\mathbf{u}, \mathbf{v}) = \begin{cases} \frac{1}{A} \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\sigma^2} \right) & \text{if } \mathbf{v} \in NN_k \\ 0 & \text{otherwise,} \end{cases} \tag{2.9}$$

where A normalizes weights such that $\sum_{\mathbf{v}} w = 1$. With this approximation, each pixel $\mathbf{u}$ has at most k non-zero weights, denoted by $w_{1..k}(\mathbf{u})$. Their corresponding selection functions are denoted by $g_{1..k}(\mathbf{u})$. Both the weights and the selection functions are independent of the signal F, and are thus precomputed. Equation (2.7) becomes

$$\begin{aligned} X(\mathbf{p}) &= \sum_{\mathbf{u}} \Big(c(\mathbf{u}) \cdot \sum_{i=1}^{k} \big(w_i(\mathbf{u}) \cdot F(g_i(\mathbf{u})) \big) \Big) && (2.10) \\ &= \sum_{i=1}^{k} \sum_{\mathbf{u}} \Big(w_i(\mathbf{u}) \cdot c(\mathbf{u}) \cdot F(g_i(\mathbf{u})) \Big). && (2.11) \end{aligned}$$

As with the nearest-neighbor signal interpolation scheme, we represent the precomputed selection functions g_i as k index matrices $\mathbf{I}_i$ of size $N \times L$. These index matrices are used to assemble k intermediate signal tensors $\mathbf{M}_i$ of size $N \times L \times C_{in}$. Additionally, we collate the precomputed weights into k weight matrices $\mathbf{H}_i$ of size $N \times L$. They are used to compute the weighted sum $\mathbf{M} = \sum_i \mathbf{H}_i \odot \mathbf{M}_i$, which is finally convolved with the kernel W.

We implemented the presented construction in TensorFlow [1]. It consists entirely of differentiable atomic operations, thus backpropagation is done seamlessly using the automatic differentiation functionality of the framework.

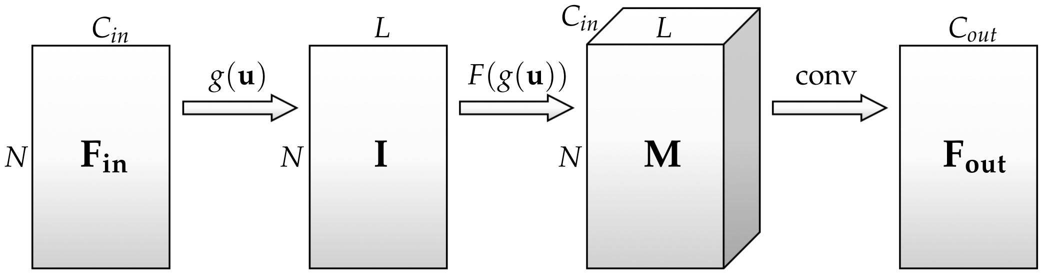


Figure 2.4: Efficient evaluation of a convolutional layer built on tangent convolutions.

2.5 ADDITIONAL INGREDIENTS

In this section we introduce additional ingredients that are required to construct a convolutional network for point cloud analysis.

2.5.1 *Multi-scale analysis*

Pooling. Convolutional networks commonly use pooling to aggregate signals over larger spatial regions. We implement pooling in our framework via hashing onto a regular 3D grid. Points that are hashed onto the same grid point pool their signals. The spacing of the grid determines the pooling resolution. Consider points $\mathcal{P} = \{\mathbf{p}\}$ and corresponding signal values $\{F(\mathbf{p})\}$. Let $\mathbf{g}$ be a grid point and let $\mathcal{V}_\mathbf{g}$ be the set of points in $\mathcal{P}$ that hash to $\mathbf{g}$. (The hash function can be assumed to be simple quantization onto the grid in each dimension.) Assume that $\mathcal{V}_\mathbf{g}$ is not empty and consider average pooling. All points in $\mathcal{V}_\mathbf{g}$ and their signals are pooled onto a single point:

$$\mathbf{p}'_\mathbf{g} = \frac{1}{|\mathcal{V}_\mathbf{g}|} \sum_{\mathbf{p} \in \mathcal{V}_\mathbf{g}} \mathbf{p} \quad \text{and} \quad F'(\mathbf{p}'_\mathbf{g}) = \frac{1}{|\mathcal{V}_\mathbf{g}|} \sum_{\mathbf{p} \in \mathcal{V}_\mathbf{g}} F(\mathbf{p}). \tag{2.12}$$

In a convolutional network based on tangent convolutions, we pool using progressively coarser grids. Starting with some initial grid resolution (5cm in each dimension, say), each successive pooling layer increases the step of the grid by a factor of two (to 10cm, then 20cm, etc.). Such hashing also alleviates the problem of non-uniform point density. As a result, we can select the neighborhood radius for the convolution operation globally for the entire dataset.

After each pooling layer, the radius r that is used to estimate the tangent plane and the pixel size of the virtual tangent image are doubled accordingly. Thus the resolution of all tangent images decreases in step with the resolution of the point cloud. Note that the downsampled point clouds produced by pooling layers are independent of the signals defined over them. The downsampled point clouds, the associated tangent planes, and the corresponding index and weight functions can thus all be precomputed for all layers in the convolutional network: they need only be computed once per pooling layer.

The implementation of a pooling layer is similar in spirit to that of a convolutional layer described in Section 2.4. Consider an input feature map $\mathbf{F}_{in}$ of size $N_{in} \times C$. Using grid hashing, we assemble an index matrix $\mathbf{I}$ of size $N_{out} \times 8$, which contains indices of points that hash to the same grid point. Assuming that we decrease the grid resolution by a factor of 2 in each dimension in each pooling layer, the number of points that hash to the same grid point will be at most 8 in general. (For initialization, we quantize the points to some base resolution.) Using $\mathbf{I}$, we assemble an intermediate tensor of size $N_{out} \times 8 \times C$. We pool this tensor along the second dimension according to the pooling operator (max, average, etc.), and thus obtain an output feature map $\mathbf{F}_{out}$ of size $N_{out} \times C$.

Note that all stages in this process have linear complexity in the number of points. Although points are hashed onto regular grids, the grids themselves are never constructed or represented. Hashing is performed via modular arithmetic on individual point coordinates, and all data structures have linear complexity in the number of points, independent of the extent of the point set or the resolution of the grid.

Unpooling has an opposite effect to pooling: it distributes signals from points in a low-resolution feature map $\mathbf{F}_{in}$ onto points in a higher-resolution feature map $\mathbf{F}_{out}$. Unpooling reuses the index matrix from the corresponding pooling operation. We copy features from a single point in a low-resolution point cloud to multiple points from which the information was aggregated during pooling.

2.5.2 *Local distance feature*

So far, we have considered signals that could be expressed in terms of a scalar function $F(\mathbf{q})$ with a well-defined value for each point $\mathbf{q}$. This holds for color, intensity, and abstract ConvNet features. There is, however, a signal that cannot be expressed in such terms and needs special treatment. This signal is distance to the tangent plane $\pi_{\mathbf{p}}$. This local signal is calculated by taking the distance from each neighbor $\mathbf{q}$ to the tangent plane of $\mathbf{p}$: $d = (\mathbf{q} - \mathbf{p})^\top \mathbf{n}_{\mathbf{p}}$.

This signal is defined in relation to the point $\mathbf{p}$, therefore it cannot be directly plugged into the pipeline shown in Figure 2.4. Instead, we precompute the distance images for every point. Scattered signal interpolation is done in the same way as for scalar signals (Equation (2.3)). After assembling the intermediate tensor **M** for the first convolutional layer, we simply concatenate these distance images as an additional channel in **M**. The first convolutional layer generates a set of abstract features $\mathbf{F}_{out}$ that can be treated as scalar signals from here on.

All precomputations are implemented using Open3D [202].

2.6 ARCHITECTURE

Using the ingredients introduced in the previous sections, we design an encoder-decoder network inspired by the U-net [143]. The network architecture is illustrated in Figure 2.5. It is a fully-convolutional network over a point cloud, where the convolutions are tangent convolutions. The encoder contains two pooling layers. The decoder contains two corresponding unpooling layers. Encoder features are propagated to corresponding decoder blocks via skip-connections. All layers except the last one use 3×3 filters and are followed by Leaky ReLU with negative slope 0.2 [106]. The last layer uses 1×1 convolutions to produce final class predictions. The network is trained by optimizing the cross-entropy objective using the Adam optimizer with initial learning rate 10^{-4} [79].

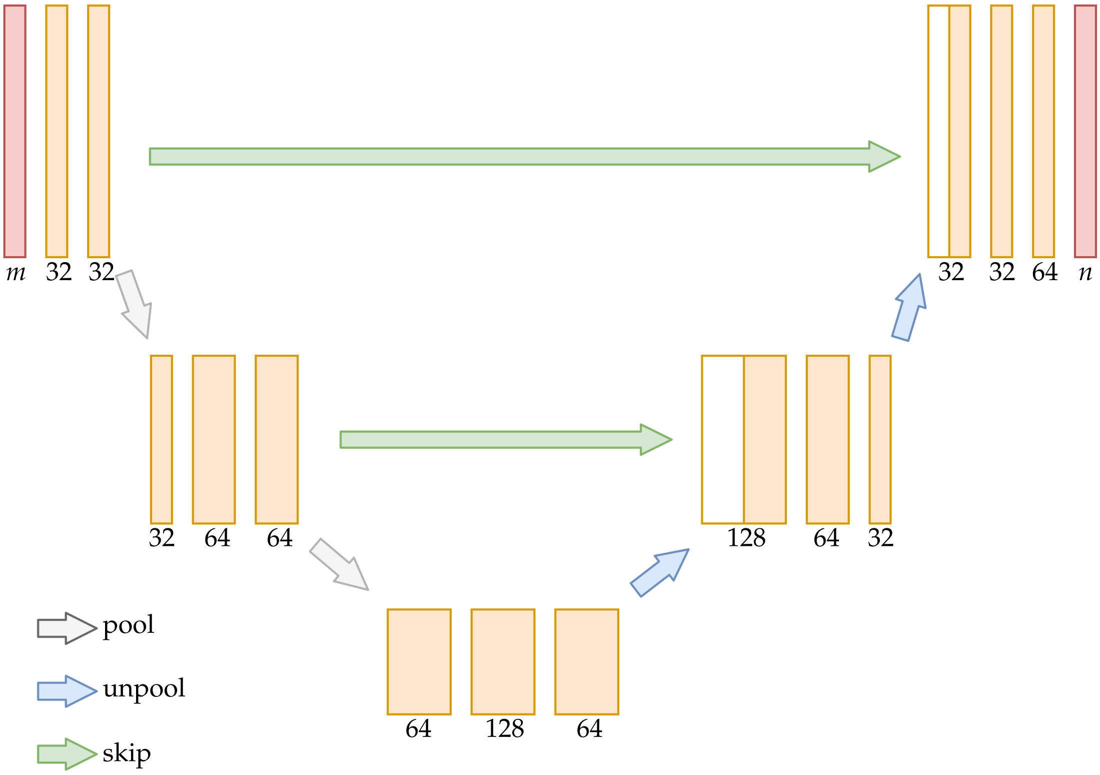


Figure 2.5: We use a fully-convolutional U-shaped network with skip connections. The network receives m-dimensional features as input and produces prediction scores for n classes.

Receptive field. The receptive field size of one convolutional layer is determined by the pixel size r of the tangent image and the radius R that is used to collect the neighbors of each point $\mathbf{p}$. We set $R = 2r$, therefore the receptive field size of one layer is R. After each pooling layer, r is doubled. The receptive field of an element in the network can be calculated by tracing the receptive fields of preceding layers. With initial $r = 5\text{cm}$, the receptive field size of elements in the final layer of the presented architecture is $4 \cdot 10 + 4 \cdot 20 + 2 \cdot 40 = 200\text{cm}$.

2.7 EXPERIMENTS

We evaluate the performance of the presented approach on the task of semantic 3D scene segmentation. Our approach is compared to several deep networks for 3D data on three different datasets.

2.7.1 *Datasets and measures*

We conduct experiments on three large-scale datasets that contain real-world 3D scans of indoor and outdoor environments.

Semantic3D [50] is a dataset of scanned outdoor scenes with over 3 billion points. It contains 15 training and 15 test scenes annotated with 8 class labels. Being unable to evaluate the baseline results on the official test server, we use our own train/test split: Bildstein 1-3-5 are used for testing, the rest for training.

Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [3] contains 6 large-scale indoor areas from 3 different buildings, with 13 object classes. We use Area 5 for testing and the rest for training.

ScanNet [25] is a dataset with more than 1,500 scans of indoor scenes with 20 object classes collected using an RGB-D capture system. We follow the standard train/test split provided by the authors.

Measures. We report three measures: mean accuracy over classes (mA), mean intersection over union (mIoU), and overall accuracy (oA). We build a full confusion matrix based on the entire test set, and derive the final scores from it. Measures are evaluated over the original point clouds. For approaches that produce labels over downsampled or voxelized representations, we map these predictions to the original point clouds via nearest-neighbor assignment.

Although we report oA for completeness, it is not a good measure for semantic segmentation. If there are dominant classes in the data (e.g., walls, floor, and ceiling in indoor scenes), making correct predictions for these but poor predictions over the other classes will yield misleadingly high oA scores. Therefore, we do not recommend using this metric in the future for assessing the performance of semantic segmentation methods. Instead, IoU should be used as a more suitable quantitative metric for the task.

2.7.2 *Baselines*

We compare our approach to three recent deep learning methods that operate on different underlying representations. We have chosen reasonably general methods that have the potential to be applied to general scene analysis and have open-source implementations. Our baselines are PointNet [132], which operates on points, ScanNet [25], which operates on low-resolution voxel grids, and OctNet [140], which operates on higher-resolution octrees. We used the source code provided by the authors. Due to the design of these methods, the data preparation routines and the input signals are different for each dataset, and sometimes deviate from the guidelines provided in the papers.

PointNet. For indoor datasets, we used the data sampling strategy suggested in the original paper with global *xyz*, locally normalized *xyz*, and RGB as inputs. For Semantic3D, we observed global *xyz* to be harmful, thus we only use local *xyz* and color. Training data is generated by randomly sampling $(3\text{m})^3$ cubes from the training scenes. Evaluation is performed by applying a sliding window over the entire scan.

ScanNet. The original network used 2 input channels: occupancy and a visibility mask computed using known camera trajectories. Since scenes in general are not accompanied by known camera trajectories, we only use occupancy in the input signal. Following the original setup, we use $1.5 \times 1.5 \times 3$m volumes voxelized into $31 \times 31 \times 62$ grids and augmented by 8 rotations. Each such cube yields a prediction for one $1 \times 1 \times 62$ column. (I.e., the ScanNet network outputs a prediction for the central column only.) We use random sampling for training, and exhaustive sliding window for testing.

OctNet. We use an architecture that operates on 256^3 octrees. Inputs to the network are color, occupancy, and a height-based feature that assigns each point to the top or bottom part of the scan. Based on correspondence with the authors regarding the best way to set up OctNet on different datasets, we used $(45\text{m})^3$ volumes for Semantic3D and $(11\text{m})^3$ volumes for the indoor datasets.

2.7.3 *Setup of the presented approach*

The architecture described in Section 2.6 is used in all experiments. We evaluate four variants that use different input signals: distance from tangent plane (D), height above ground (H), normals (N), and color (RGB). All input signals are normalized between 0 and 1. The initial resolution r of the tangent image is 5cm for the indoor datasets and 10cm for Semantic3D. It is doubled after each pooling layer. In addition to providing the distance from tangent plane as input to the first convolutional layer, we concatenate the local distance features from all scales of the point cloud to the feature maps of the corresponding resolution produced by pooling layers.

For ScanNet and S3DIS, we used whole rooms as individual training batches. For Semantic3D, each batch was a random sphere with a radius of 6m. For indoor scans, we augment each scan by 8 rotations around the vertical axis. To correct for imbalance between different classes, we weigh the loss with the negative log of the training data histogram.

2.7.4 *Signal interpolation*

We compare the effectiveness of two different signal interpolation schemes: nearest neighbor and Gaussian mixture. Both networks were trained on S3DIS with D and H as the input signals; see Table. 2.1. The two networks produce similar results. We conclude that the nearest neighbor signal estimation scheme is sufficient, and use it in all other experiments.

Signal	mIoU	mA	oA
NN	50.0	60.0	81.2
Gaussian	50.7	59.6	81.3

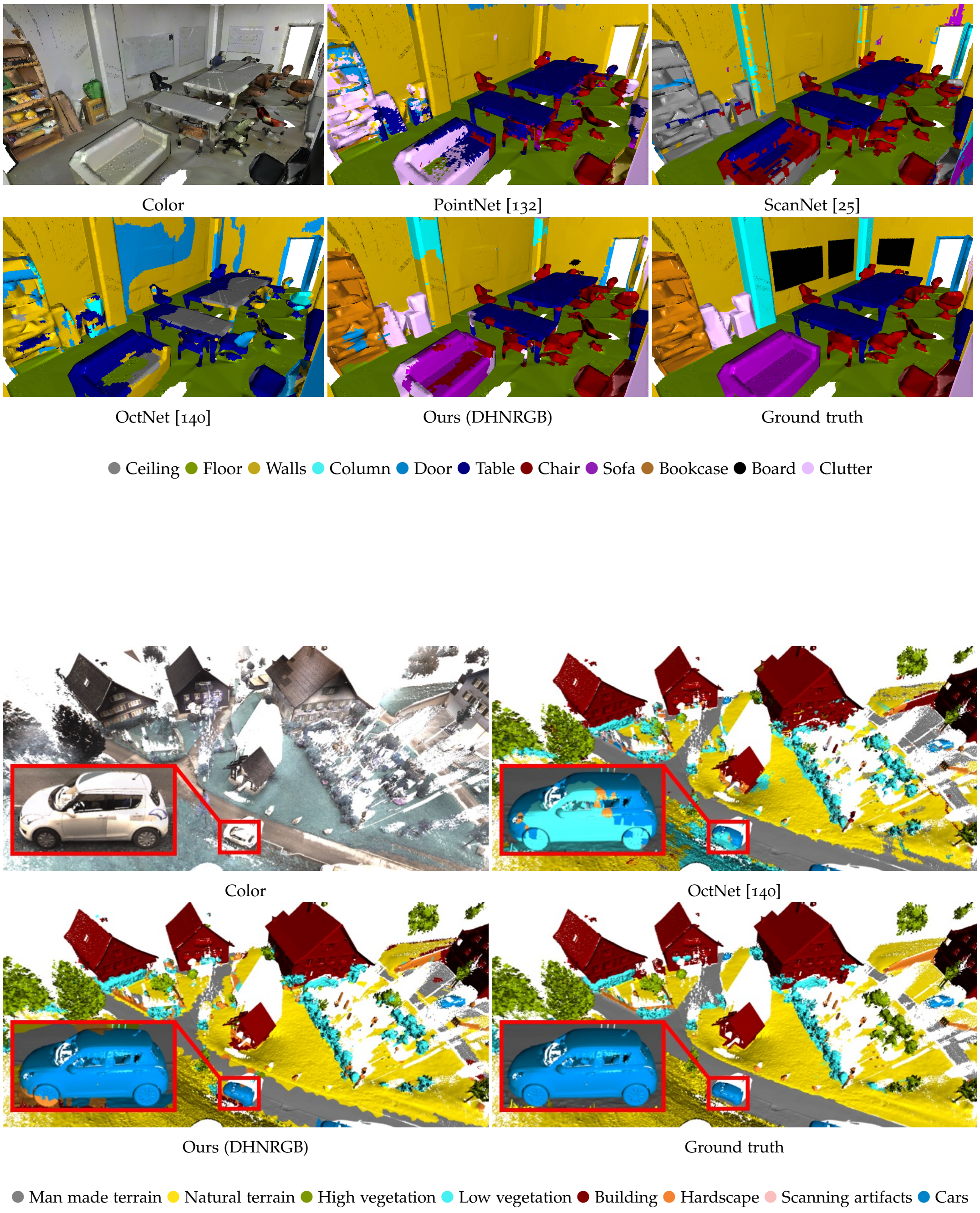
Table 2.1: Signal interpolation using the nearest neighbor scheme and the Gaussian mixture scheme produce similar results.

2.7.5 *Main results*

Quantitative results for all methods are summarized in Table 2.2. Overall, our method produces high scores on all datasets and consistently outperforms the baselines. Qualitative comparisons are shown in Figure 2.6.

	Semantic3D [50]			ScanNet [25]			S3DIS [3]		
	mIoU	mA	oA	mIoU	mA	oA	mIoU	mA	oA
PointNet [132]	3.76	16.9	16.3	12.2	17.9	68.1	41.3	49.5	78.8
OctNet [140]	50.7	71.3	80.7	18.1	26.4	76.6	26.3	39.0	68.9
ScanNet [25]	n/a	n/a	n/a	13.5	19.2 (*50.8*)	69.4 (*73.0*)	24.6	35.0	64.2
Ours (D)	58.1	78.9	84.8	40.9	52.5	**80.9**	49.8	60.3	80.2
Ours (DH)	58.0	75.8	83.3	40.3	52.2	80.6	50.0	60.0	81.2
Ours (DHN)	52.5	79.3	79.5	40.7	**55.3**	80.3	51.7	61.0	82.2
Ours (DHNRGB)	**66.4**	**80.7**	**89.3**	**40.9**	55.1	80.1	**52.8**	**62.2**	**82.5**

Table 2.2: Semantic segmentation accuracy for all methods across the three datasets. We report mean intersection over union (mIoU), mean class accuracy (mA), and overall accuracy (oA). Note that oA is a bad measure and we recommend against using it in the future. We tested different configurations of our method by combining four types of input signals: depth (D), height (H), normals (N), and color (RGB).


Color

PointNet [132]

ScanNet [25]

OctNet [140]

Ours (DHNRGB)

Ground truth

● Ceiling ● Floor ● Walls ● Column ● Door ● Table ● Chair ● Sofa ● Bookcase ● Board ● Clutter

Color

OctNet [140]

Ours (DHNRGB)

Ground truth

● Man made terrain ● Natural terrain ● High vegetation ● Low vegetation ● Building ● Hardscape ● Scanning artifacts ● Cars

Figure 2.6: Qualitative comparisons on S3DIS [3] (top) and Semantic3D [50] (bottom). Labels are coded by color.

Comparing the configurations of our networks that use different input signals, we can see that geometry is much more important than color on the indoor datasets. Adding RGB information only slightly improves the scores on S3DIS and is actually harmful for mean and overall accuracy on the ScanNet dataset. The situation is different for the Semantic3D dataset: the network trained with color significantly outperforms all other configurations. Due to the fact that H is normalized between 0 and 1 for every scan separately, this information turns out to be harmful when the global height of different scans is significantly different. Therefore, the network trained only with the distance signal performs better than the other two geometric configurations.

In setting up and operating the baseline methods, we found that all of them are quite hard to apply across datasets: some non-trivial decisions had to be made for each new dataset during the data preparation stage. None of the baselines showed consistent performance across the different types of scenes.

PointNet reaches high oA scores on both indoor datasets. However, the oA measure is strongly dominated by large classes such as walls, floor, and ceiling. S3DIS has a fairly regular layout because of the global room alignment procedure, which is very beneficial for PointNet and allows it to reach reasonable mA and mIoU scores on this dataset. However, PointNet performs poorly on the ScanNet dataset, which has more classes and noisy data. All but the most prominent classes (i.e., walls and floor) are misclassified. PointNet completely fails to produce meaningful predictions on the even more challenging Semantic3D dataset.

Our configuration of the ScanNet method produces reasonable oA scores on both indoor datasets, but does much worse in the other two measures. For reference, on the ScanNet dataset we additionally report the number from the original paper where a binary visibility-from-camera mask was used as an additional input channel. This number is much higher than our occupancy-only results, which do not assume a known camera trajectory. Due to the fact that the network only outputs predictions for the central column of the voxel grid, evaluation is extremely time-consuming for the large scenes in the Semantic3D dataset. Because of this scalability issue, we did not succeed in evaluating ScanNet on this dataset.

OctNet reaches good performance on the Semantic3D dataset. However, the same network configuration yields bad results when applied to the indoor datasets. A possible explanation for this may be poor generalization due to overfitting to the structure of training octrees. Another reason for such performance decrease may be sensitivity to the hyperparameter setting.

Additional qualitative results showcasing the performance of our method on different datasets are shown in Figure 2.7 (S3DIS), Figure 2.8 (ScanNet) and Figure 2.9 (Semantic3D).

2.7.6 *Efficiency*

We compared the efficiency of different methods on a scan from S3DIS containing 125K points after grid hashing. The results are reported in Table 2.3. Since ScanNet

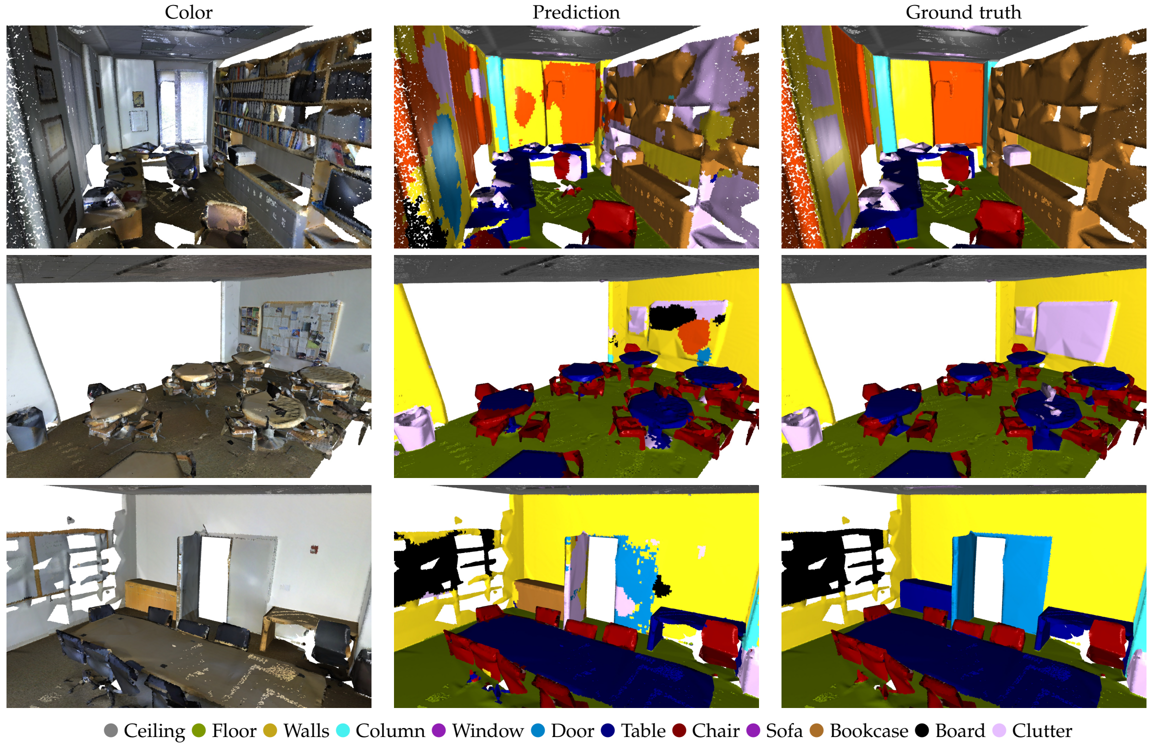

Color
Prediction
Ground truth
● Ceiling ● Floor ● Walls ● Column ● Window ● Door ● Table ● Chair ● Sofa ● Bookcase ● Board ● Clutter

Figure 2.7: Qualitative results on S3DIS [3].

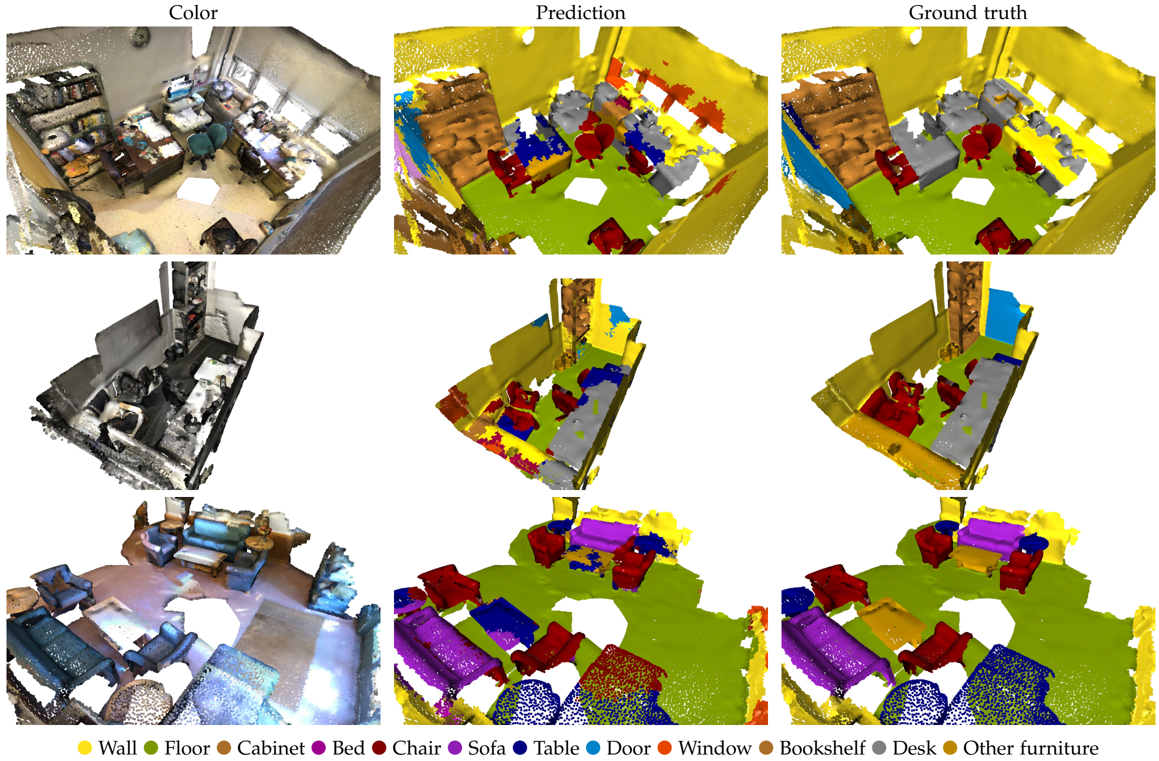

Color
Prediction
Ground truth
● Wall ● Floor ● Cabinet ● Bed ● Chair ● Sofa ● Table ● Door ● Window ● Bookshelf ● Desk ● Other furniture

Figure 2.8: Qualitative results on ScanNet [25].

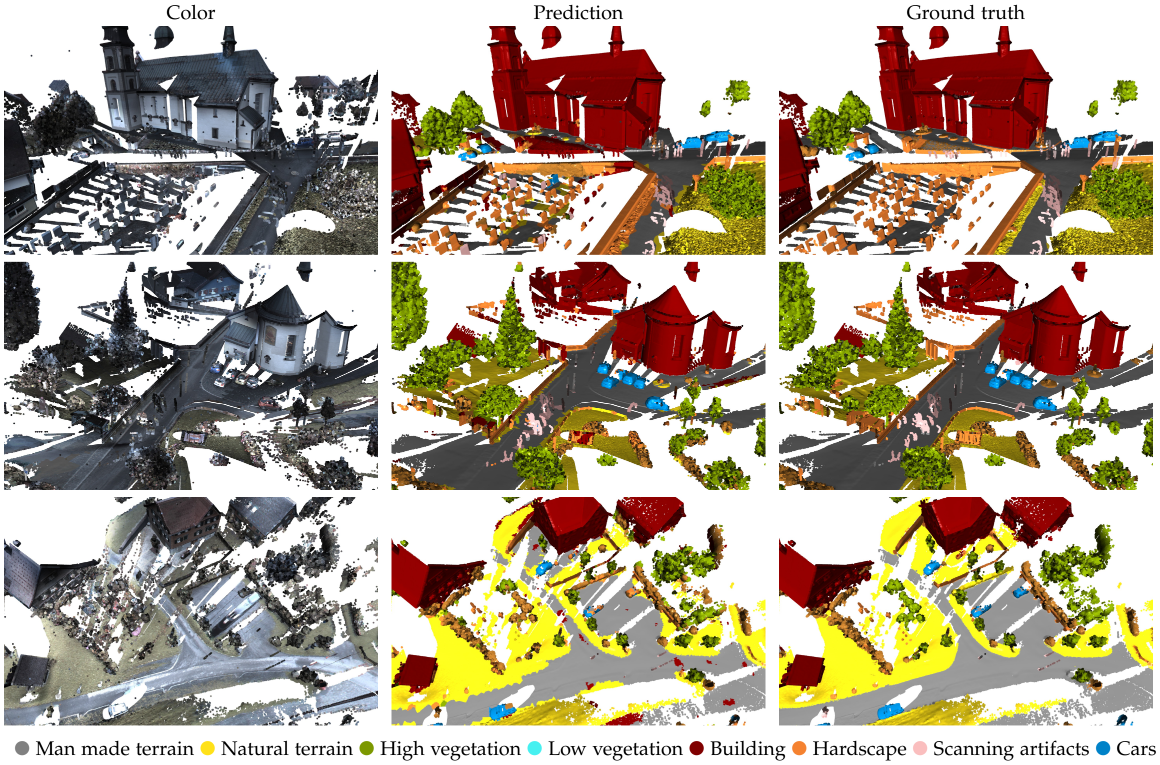


Figure 2.9: Qualitative results on Semantic3D [50].

and PointNet require multiple iterations for labeling a single scan, we report both the time of a single forward pass and the time for processing a full scan. OctNet and our method process a full scan in one forward pass, which also explains their higher memory consumption compared to ScanNet and PointNet. ScanNet does not provide code for data preprocessing, so we report the runtime of our Python implementation needed for generating 38K sliding windows during inference. Our method exhibits the best runtime for both precomputation and inference. The implementation of our approach is publicly available[1].

	Prep (s)	FP (s)	Full (s)	Mem (GB)
PointNet	16.5	0.01	0.65	0.39
OctNet	15.5	0.61	0.61	3.33
ScanNet	867.8	0.002	6.34	0.97
Ours	1.59	0.52	0.52	2.35

Table 2.3: Efficiency of different methods. We report preprocessing time (Prep), time for a single forward pass (FP), time for processing a full scan (Full), and memory consumption (Mem).

[1] https://github.com/tatarchm/tangent_conv

2.7.7 *Robustness to noise*

We evaluated the robustness of our approach to noise. Several instances of our network were trained on the S3DIS dataset perturbed with different amounts of additive Gaussian noise with standard deviation σ. The results are reported in Table 2.4. We selected small subsets of the data for training and testing (Area 1 for training and Area 5 for testing), which is why the final performance numbers are not compatible with those reported in Table 2.2.

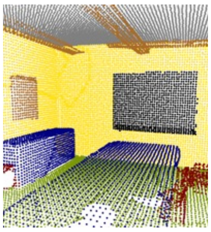
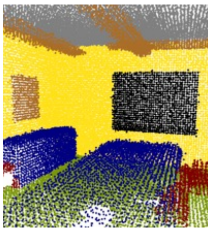
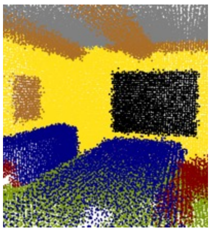
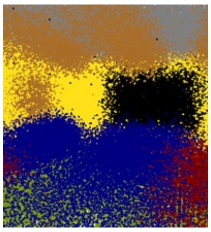
Training data					
σ, m	0.00	0.02	0.04	0.08	0.16
OA	0.59	0.63	0.63	0.68	0.17

Table 2.4: Performance evaluation with different levels of noise.

Interestingly, reasonable amounts of noise improve overall accuracy. We hypothesize that this observation can be explained by the regularizing effect of adding noise to the original point cloud geometry. The method only suffers if the noise severely damages semantic structure in the point cloud ($\sigma = 0.16$ in Table 2.4). We did not tune any parameters in the pipeline for these experiments.

2.7.8 *Comparison with SnapNet*

We also compared our approach with the SnapNet by Boulch *et al.* [8]. Their method is based on projecting a 3D scene onto a set of 2D images. Those images are then segmented with a regular 2D ConvNet. The main strength of this approach is the possibility to combine it with transfer learning and use the weights of a network pre-trained on ImageNet for initialization. Applying this strategy yields the mIoU score of 67.7 on the Semantic3D dataset, compared to 66.4 produced by our approach. However, the limitation of their approach is the need to sample camera poses for training and testing. The original sampling scheme provided by the authors was optimized for an outdoor dataset. Modifying it to support indoor data was not trivial, therefore we could not obtain results there.

2.8 CONCLUSION

We have presented tangent convolutions – a new construction for convolutional networks on 3D data. The key idea is to evaluate convolutions on virtual tangent planes at every point. Crucially, tangent planes can be precomputed and deep convolutional networks based on tangent convolutions can be evaluated efficiently on large point clouds. We have applied tangent convolutions to semantic segmentation of large indoor and outdoor scenes. The presented ideas may also be applicable to other problems in analysis of 3D data.

3 3D SYNTHESIS: OCTREE GENERATING NETWORKS

The text of this chapter was largely copied from the following paper.

> Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs." In: *ICCV*, 2017.

All co-authors contributed to the project discussions as well as the final paper text editing.

* * *

3.1 INTRODUCTION

Up-convolutional[1] architectures have become a standard tool for image synthesis tasks [71, 134]. They consist of a series of convolutional and up-convolutional (upsampling+convolution) layers operating on regular grids, with resolution gradually increasing towards the output of the network. The architecture is trivially generalized to volumetric data. However, similar to the case of 3D data analysis discussed in Chapter 2, because of cubic scaling of computational and memory requirements, training up-convolutional decoders becomes infeasible for high-resolution three-dimensional outputs.

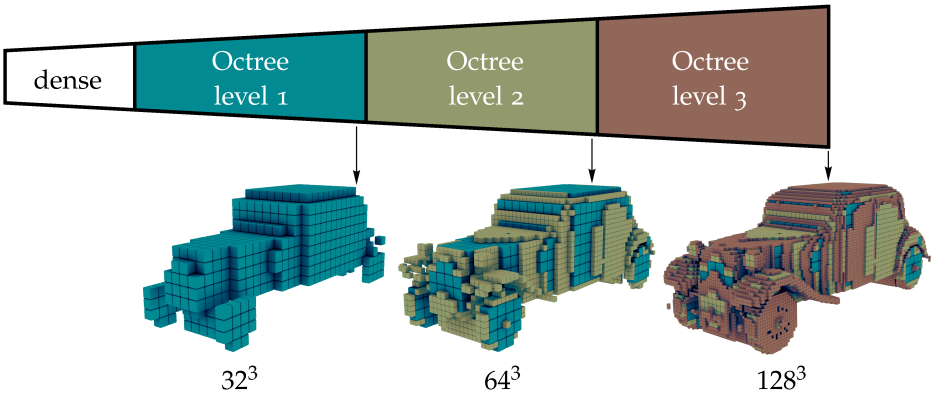


Figure 3.1: The proposed OGN represents its volumetric output as an octree. Initially estimated rough low-resolution structure is gradually refined to a desired high resolution. At each level only a sparse set of spatial locations is predicted. This representation is significantly more efficient than a dense voxel grid and allows generating volumes as large as 512^3 voxels on a modern GPU in a single forward pass.

[1] Also known as deconvolutional

Poor scaling can be resolved by exploiting structure in the data. In many learning tasks, neighboring voxels on a voxel grid share the same state — for instance, if the voxel grid represents a binary occupancy map or a multi-class labeling of a three-dimensional object or a scene. In this case, data can be efficiently represented with octrees — data structures with adaptive cell size. Large regions of space sharing the same value can be represented with a single large cell of an octree, resulting in savings in computation and memory compared to a fine regular grid. At the same time, fine details are not lost and can still be represented by small cells of the octree.

We present an octree generating network (OGN) - a convolutional decoder operating on octrees. The coarse structure of the network is illustrated in Figure 3.1. Similar to a usual up-convolutional decoder, the representation is gradually convolved with learned filters and up-sampled. The difference is that, starting from a certain layer in the network, dense regular grids are replaced by octrees. Therefore, the OGN predicts large uniform regions of the output space already at early decoding stages, saving the computation for the subsequent high-resolution layers. Only regions containing fine details are processed by these more computationally demanding layers.

Here we focus on generating shapes represented as binary occupancy maps. We thoroughly compare OGNs to standard dense nets on three tasks: auto-encoding shapes, generating shapes from a high-level description, and reconstructing 3D objects from single images. OGNs yield the same accuracy as conventional dense decoders while consuming significantly less memory and being much faster at high resolutions. For the first time, we can generate shapes of resolution as large as 512^3 voxels in a single forward pass. Our OGN implementation is publicly available[2].

3.2 RELATED WORK

The majority of deep learning approaches generate volumetric data based on convolutional networks with feature maps and outputs represented as voxel grids. Applications include single- and multi-view 3D object reconstruction trained in supervised [20, 27, 42, 46] and unsupervised [38, 137, 191] ways, probabilistic generative modeling of 3D shapes [150, 183, 188], and shape deformation [194]. A fundamental limitation of these approaches is the low resolution of the output. Memory and computational requirements of approaches based on the voxel grid representation scale cubically with the output size. Thus, training networks with resolutions higher than 64^3 comes with memory issues on the GPU or requires other measures to save memory, such as reducing the batch size or generating the volume part-by-part. Moreover, with growing resolution, training times become prohibitively slow.

Computational limitations of the voxel grid representation led to research on alternative representations of volumetric data in deep learning. Sinha *et al.* [157] convert shapes into two-dimensional geometry images and process those with conventional CNNs – an approach only applicable to certain classes of topologies. Net-

[2] `https://github.com/lmb-freiburg/ogn`

works producing point clouds have been applied to object generation [33, 108]. Multiple approaches [41, 91, 121, 125, 169] aimed for structural 3D understanding, approximating 3D shapes with a pre-defined set of primitives. Gao *et al.* [39] and Wu *et al.* [187] make the next step in this direction and propose systems for shape reconstruction based on recombining parts of existing objects. Several concurrent works [53, 140] performed hierarchical partitioning of the output space to achieve computational and memory efficiency, which allows predicting higher-resolution 3D shapes. Johnston *et al.* [72] reconstructed high-resolution 3D shapes with an inverse discrete cosine transform decoder. Wang *et al.* [176] and Wen *et al.* [180] generated meshes by deforming a sphere into a desired shape, assuming a fixed distance between camera and objects. A similar idea based on interpolating between multiple deformed meshes was proposed by Pontes *et al.* [128]. Groueix *et al.* [48] assembled surfaces from small patches. Multiple methods [92, 99, 105, 158] produced multi-view depth maps that are fused together into an output point cloud. Richter *et al.* [138] extended this with nested shapes fused into a single voxel grid. Wu *et al.* [184] learned the mapping from input images to 2.5D sketches in a fully-supervised fashion, and then trained a network to map these intermediate representations to the final 3D shapes. Kong *et al.* [84] use 2D landmark locations together with silhouettes to retrieve and deform CAD models. Pontes *et al.* [129] improved upon this work by using a free-form deformation parametrization to model shape variation. Several works proposed using implicit surface methods [17, 118, 124].

3.3 octrees

An octree [117] is a 3D grid structure with adaptive cell size, which allows for lossless reduction of memory consumption compared to a regular voxel grid. Octrees have a long history in classical 3D reconstruction and depth map fusion [9, 23, 37, 77, 159, 171]. A function defined on a voxel grid can be converted into a function defined on an octree. This can be done by starting from a single cell representing the entire space and recursively partitioning cells into eight octants. If every voxel within a cell has the same function value, this cell is not subdivided and becomes a leaf of the tree. The set of cells at a certain resolution is referred to as an octree level. The recursive subdivision process can also be started not from the whole volume, but from some initial coarse resolution. Then the maximal octree cell size is given by this initial resolution. The most straightforward way of implementing an octree is to store in each cell pointers to its children. In this case, the time to access an element scales linearly with the tree's depth, which can become costly at high resolutions. We use a more efficient implementation that exploits hash tables. An octree cell with spatial coordinates $\mathbf{x} = (x, y, z)$ at level l is represented as an index-value pair (m, v), where v can be any kind of discrete or continuous signal. m is calculated from $(\mathbf{x}, l)$ using Z-order curves [40]

$$m = \mathcal{Z}(\mathbf{x}, l), \tag{3.1}$$

which is a computationally cheap transformation implemented using bit shifts. An octree O is, hence, a set of all pairs

$$O = \{(m, v)\}. \tag{3.2}$$

Storing this set as a hash table allows for constant-time element access.

When training networks, we will need to compare two different octrees O^1 and O^2, i.e. for each cell $(\mathbf{x}, l)$ from O^1, query the corresponding signal value v in O^2. Since different octrees have different structure, two situations are possible. If $\mathcal{Z}(\mathbf{x}, k)$ is stored at a level k in O^2, which is the same or lower than l, the signal value of this cell can be uniquely determined. If $\mathcal{Z}(\mathbf{x}, k)$ is stored at one of the later levels, the cell is subdivided in O^2, and the value of the whole cell is not defined. To formalize this, we introduce a function f for querying the signal value of an arbitrary cell with index $m = \mathcal{Z}(\mathbf{x}, l)$ from octree O:

$$f(m, O) = \begin{cases} v, \text{ if } \exists k \leq l : (\mathcal{Z}(\mathbf{x}, k), v) \in O \\ \varnothing, \text{ otherwise} \end{cases}, \tag{3.3}$$

where $\varnothing$ denotes an unavailable value.

3.4 OCTREE GENERATING NETWORKS

An Octree Generating Network (OGN) is a convolutional decoder that yields an octree as output: both the structure, i.e. which cells should be subdivided, and the signal value of each cell. In this work we concentrate on binary occupancy values $v \in \{0, 1\}$, but the proposed framework can be easily extended to support arbitrary signals. As shown in Figure 3.1, an OGN consists of a block operating on dense regular grids, followed by an arbitrary number of hash-table-based octree blocks.

The dense block is a set of conventional 3D convolutional and up-convolutional layers producing a feature map of size $d_1 \times d_2 \times d_3 \times c$ as output, where $\{d_i\}$ are the spatial dimension and c is the number of channels.

From here on, the representation is processed by our custom layers operating on octrees. The regular-grid-based feature map produced by the dense block is converted to a set of index-value pairs stored as a hash table (with values being feature vectors), and is further processed in this format. We organize octree-based layers in blocks, each responsible for predicting the structure and the content of a single level of the generated octree.

Figure 3.2 illustrates the functioning of a single such block that predicts level l of an octree. For the sake of illustration, we replaced three-dimensional octrees by two-dimensional quadtrees. Feature maps in Figure 3.2 are shown as dense arrays only for simplicity; in fact the green cells are stored in hash maps, and the white cells are not stored at all. We now give a high-level overview of the block and then describe its components in more detail.

Input to the block is a sparse hash-table-based convolutional feature map F_{l-1} of resolution $(d_1 \cdot 2^{l-1}, d_2 \cdot 2^{l-1}, d_3 \cdot 2^{l-1})$ produced by the previous block. First this

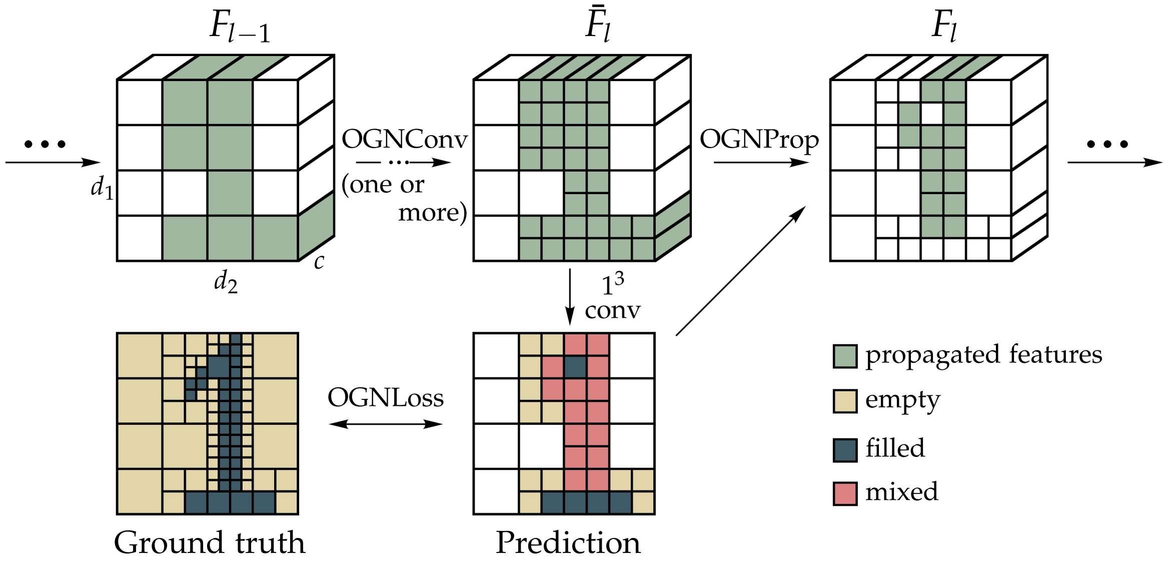


Figure 3.2: Single block of an OGN illustrated as 2D quadtree for simplicity. After convolving features F_{l-1} of the previous block with weight filters, we directly predict the occupancy values of cells at level l using 1^3 convolutions. Features corresponding to "filled" and "empty" cells are no longer needed and thus not propagated, which yields F_l as the final output of this block.

feature map is processed with a series of custom convolutional layers and one up-convolutional layer with stride 2, all followed by non-linearities.

This yields a new feature map $\bar{F}_l$ of resolution $(d_1 \cdot 2^l, d_2 \cdot 2^l, d_3 \cdot 2^l)$. Based on this feature map, we directly predict the content of level l. For each cell, there is a two-fold decision to be made: should it be kept at level l, and if yes, what should be the signal value in this cell? In our case making this decision can be formulated as classifying the cell as being in one of three states: "empty", "filled" or "mixed". These states correspond to the outputs of state-querying function f from Equation (3.3), with "empty" and "filled" being the signal values v, and "mixed" being the state where the value is not determined. We make this prediction using a convolutional layer with 1^3 filters followed by a three-way softmax. This classifier is trained in a supervised manner with targets provided by the ground truth octree.

Finally, in case the output resolution has not been reached, features from $\bar{F}_l$ that correspond to "mixed" cells are propagated to the next layer[3] and serve as an input feature map F_l to the next block.

In the following subsections, we describe the components of a single octree block in more detail: the octree-based convolution, the loss function, and the feature propagation mechanism.

[3] Additional neighboring cells may have to be propagated if needed by subsequent convolutional layers. This is described in section 3.4.3.

3.4.1 *Convolution*

We implemented a custom convolutional layer *OGNConv*, which operates on feature maps represented as hash tables instead of usual dense arrays. Our implementation supports strided convolutions and up-convolutions with arbitrary filter sizes. It is based on representing convolution as a single matrix multiplication, similar to standard *Caffe* [68] code for dense convolutions.

In the dense case, the feature tensor is converted to a matrix with the *im2col* operation, then multiplied with the weight matrix of the layer, and the result is converted back into a dense feature tensor using the *col2im* operation. In OGN, instead of storing full dense feature tensors, only a sparse set of relevant features is stored at each layer. These features are stored in a hash table, and we implemented custom operations to convert a hash table to a feature matrix and back. The resulting matrices are much smaller than those in the dense case. Convolution then amounts to multiplying the feature matrix by the weight matrix. Matrix multiplication is executed on GPU with standard optimized functions, and our conversion routines currently run on CPU. Even with this suboptimal CPU implementation, computation times are comparable to those of usual dense convolutions at 32^3 voxel resolution. At higher resolutions, *OGNConv* is much faster than dense convolutions (see Section 3.5.2).

The convolution operation on a voxel grid is perfectly shift invariant by design. This is no longer true for convolutions on octrees: a shift by a single pixel in the original voxel grid can change the structure of the octree significantly. We study the effect of this in the Experiments section.

3.4.2 *Loss*

The classifier at level l of the octree outputs the probabilities of each cell from this level being "empty", "filled" or "mixed", that is, a three-component prediction vector $\mathbf{p_m} = (p^0_m, p^1_m, p^2_m)$ for cell with index m. We minimize the cross-entropy between the network predictions and the cell states of the ground truth octree O_{GT}, averaged over the set M_l of cells predicted at layer l:

$$\mathcal{L}_l = \frac{1}{|M_l|} \sum_{m \in M_l} \left[\sum_{i=0}^{2} h^i(f(m, O_{GT})) \log p^i_m \right], \tag{3.4}$$

where function $\mathbf{h}$ yields a one-hot encoding (h^0, h^1, h^2) of the cell state value returned by f from Equation (3.3). Loss computations are encapsulated in our custom *OGNLoss* layer.

The final OGN objective is calculated as a sum of loss values from all octree levels

$$\mathcal{L} = \sum_{l=1}^{L} \mathcal{L}_l. \tag{3.5}$$

3.4.3 *Feature propagation*

At the end of each octree block there is an *OGNProp* layer that propagates to the next octree block features from cells in the "mixed" state, as well as from neighboring cells if needed to compute subsequent convolutions. Information about the cell state can either be taken from the ground truth octree, or from the network prediction. This spawns two possible propagation modes: using the known tree structure (*Prop-known*) and using the predicted tree structure (*Prop-pred*). Section 3.4.4 describes use cases for these two modes.

The set of features to be propagated depends on the kernel size in subsequent *OGNConv* layers and is computed based on the network architecture, before the training starts. The example in Figure 3.2 only holds for 2^3 up-convolutions which do not require any neighboring elements to be computed. This situation is illustrated in Figure 3.3-A in a one-dimensional case. Circles correspond to cells of an octree. The green cell in the input is the only one for which the value was predicted to be "mixed". Links between the circles indicate which features of the input are required to compute the result of the operation (convolution or up-convolution) for the corresponding output cell. In this case, we can see that the output cells in the next level are only affected by their parent cell from the previous level.

To use larger convolutional filters or multiple convolutional layers, we must propagate not only the features of the "mixed" cells, but also the features of the neighboring cells required for computing the convolution at the locations of the "mixed" cells. This more general situation is shown in Figure 3.3-B. The input is processed with an up-convolutional layer with 4^3 filters and stride 2, which is followed by a convolutional layer with 3^3 filters and stride 1. Again, only one cell was predicted to be "mixed", but in order to perform convolutions and up-convolutions in subsequent layers, we additionally must propagate some of its neighbors (marked red). Therefore, with this particular filter configuration, two cells in the output are affected by four cells in the input.

Generally, the number of features that should be propagated by each *OGNProp* layer is automatically calculated based on the network architecture before starting the training.

3.4.4 *Training and testing*

The OGN decoder is end-to-end trainable using standard backpropagation. The only subtlety is in selecting the feature propagation modes during training and testing. At training time the octree structure of the training samples is always available, and therefore the *Prop-known* mode can be used. At test time, the octree structure may or may not be available. We have developed two training regimes for these two cases.

If the tree structure is available at test time, we simply train the network with *Prop-known* and test it the same way. This regime is applicable for tasks like seman-

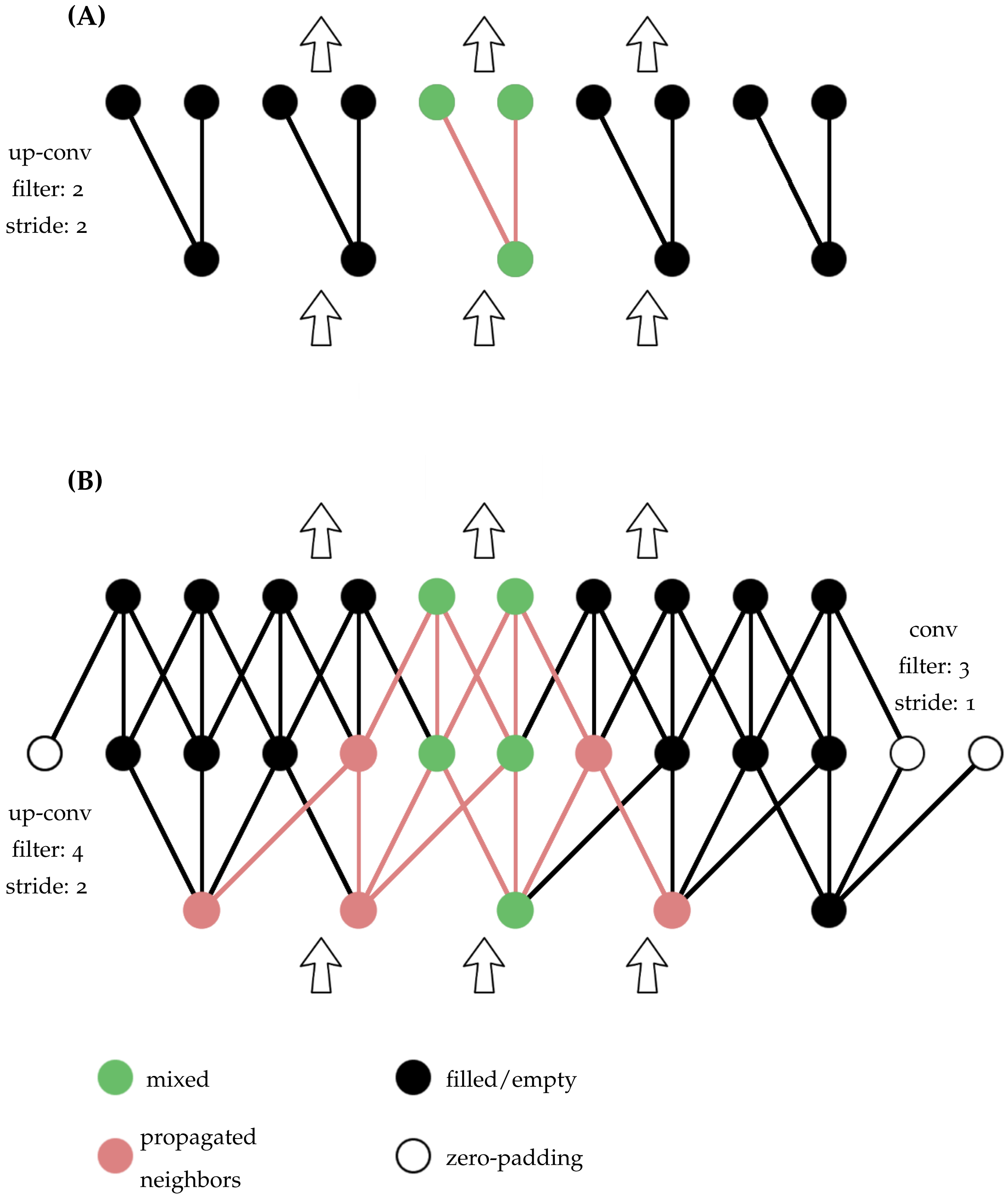


Figure 3.3: The *OGNProp* layer propagates the features of "mixed" cells together with the features of the neighboring cells required for computations in subsequent layers. We show the number of neighbors that need to be propagated in two cases: 2^3 up-convolutions (A), and 4^3 up-convolutions followed by 3^3 convolutions (B). Visualized in 1D for simplicity.

tic segmentation, or, more generally, per-voxel prediction tasks, where the structure of the output is exactly the same as the structure of the input.

If the tree structure is not available at test time, we start by training the network with *Prop-known*, and then fine-tune it with *Prop-pred*. This regime is applicable to any task with volumetric output.

We have also tested other regimes of combining *Prop-pred* and *Prop-known* and found those to perform worse than the two described variants. This is discussed in more detail in the Experiments section.

3.5 EXPERIMENTS

In our experiments we verified that the OGN architecture performs on par with the standard dense voxel grid representation, while requiring significantly less memory and computation, particularly at high resolutions. The focus of the experiments is on showcasing the capabilities of the proposed architecture. How to fully exploit the new architecture in practical applications is a separate problem that is left to future work.

3.5.1 *Experimental setup*

For all OGN decoders used in our evaluations, we followed the same design pattern: 1 or 2 up-convolutional layers interleaved with a convolutional layer in the dense block, followed by multiple octree blocks depending on the output resolution. In the octree blocks we used 2^3 up-convolutions. We also evaluated two other architecture variants, presented in section 3.5.3.1. ReLU non-linearities were applied after each (up-)convolutional layer. The number of channels in the up-convolutional layers of the octree blocks was set to 32 in the outermost layer, and was increased by 16 in each preceding octree block. The exact network architectures used in individual experiments are shown in Tables 3.1, 3.2 and 3.3.

The networks were trained using ADAM [79] with initial learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The learning rate was decreased by a factor of 10 after 30K and 70K iterations. We did not apply any additional regularization.

For quantitative evaluations, we converted the resulting octrees back to regular voxel grids, and computed the Intersection over Union (IoU) measure between the ground truth model and the predicted model. To quantify the importance of high-resolution representations, in some experiments we upsampled low-resolution network predictions to high-resolution ground truth using trilinear interpolation, and later binarization with a threshold of 0.5. We explicitly specify the ground truth resolution in all experiments where this was done.

If not indicated otherwise, the results were obtained in the *Prop-pred* mode.

3.5.1.1 *Datasets*

In our evaluations we used three datasets:

ShapeNet-all

Approximately 50.000 CAD models from 13 main categories of the ShapeNet dataset [11], used by Choy *et al.* [20]. We also used the renderings provided by Choy *et al.* [20].

ShapeNet-cars

A subset of ShapeNet-all consisting of 7497 car models.

BlendSwap

A dataset of 4 scenes we manually collected from `blendswap.com`, a website containing a large collection of Blender models.

All datasets were voxelized in multiple resolutions from 32^3 to 512^3 using the *binvox*[4] tool, and were converted into octrees. We set the interior parts of individual objects to be filled, and the exterior to be empty. However, the method also works for data where both interior and exterior voxels are empty, and only voxels on the object surface are filled.

3.5.1.2 *Network architectures*

In this section, we provide the exact network architectures used in the experimental evaluations.

Autoencoders

The architectures of OGN autoencoders are summarized in Table 3.1. For the dense baselines, we used the same layer configurations with usual convolutions instead of *OGNConv*, and predictions being made only after the last layer of the network. All networks were trained with batch size 16.

3D shape from high-level information

OGN decoders used on the Shapenet-cars dataset are shown in Table 3.2. Encoders consisted of three fully-connected layers, with output size of the last encoder layer being identical to the input size of the corresponding decoder.

For FAUST and BlendSwap the 256^3 output octrees had four levels, not five like those in Table 3.2. Thus, the dense block had an additional deconvolution-convolution layer pair instead of one octree block. The 512^3 decoder on BlendSwap had one extra octree block with 32 output channels.

All 64^3 and 128^3 networks were trained with batch size 16, 256^3 — with batch size 4, 512^3 — with batch size 1.

Single-image 3D reconstruction

In this experiment we again used decoder architectures shown in Table 3.2. The architecture of the convolutional encoder is shown in Table 3.3. The number of channels in the last encoder layer was set identical to the number of input channels of the corresponding decoder.

[4] `http://www.patrickmin.com/binvox`

<table>
<tr><th>$\mathbf{32^3}$</th><th>$\mathbf{64^3}$ (2^3 filters)</th><th>$\mathbf{64^3}$ (4^3 filters)</th><th>$\mathbf{64^3}$ (InvConv)</th></tr>
<tr><td></td><td colspan="3">[$64^3 \times 1$]</td></tr>
<tr><td>[$32^3 \times 1$]</td><td colspan="3">Conv (3^3)
[$32^3 \times 32$]</td></tr>
<tr><td>Conv (3^3)
[$16^3 \times 32$]</td><td colspan="3">Conv (3^3)
[$16^3 \times 48$]</td></tr>
<tr><td>Conv (3^3)
[$8^3 \times 48$]</td><td colspan="3">Conv (3^3)
[$8^3 \times 64$]</td></tr>
<tr><td>Conv (3^3)
[$4^3 \times 64$]</td><td colspan="3">Conv (3^3)
[$4^3 \times 80$]</td></tr>
<tr><td>FC
[1024]</td><td colspan="3">FC
[1024]</td></tr>
<tr><td>FC
[1024]</td><td colspan="3">FC
[1024]</td></tr>
<tr><td>FC
[$4^3 \times 80$]</td><td colspan="3">FC
[$4^3 \times 96$]</td></tr>
<tr><td>Deconv (2^3)
[$8^3 \times 64$]</td><td colspan="3">Deconv (2^3)
[$8^3 \times 80$]</td></tr>
<tr><td>Conv (3^3)→ **l1**
[$8^3 \times 64$]</td><td colspan="3">Conv (3^3)
[$8^3 \times 80$]</td></tr>
<tr><td>*OGNProp*
OGNConv(2^3) → **l2**
[$16^3 \times 48$]</td><td colspan="3">Deconv (2^3)
[$16^3 \times 64$]</td></tr>
<tr><td>*OGNProp*
OGNConv(2^3) → **l3**
[$32^3 \times 32$]</td><td colspan="3">Conv (3^3)→ **l1**
[$16^3 \times 64$]</td></tr>
<tr><td></td><td>*OGNProp*
OGNConv(2^3) → **l2**
[$32^3 \times 48$]</td><td>*OGNProp*
OGNConv(4^3) → **l2**
[$32^3 \times 48$]</td><td>*OGNProp*
OGNConv(2^3)
[$32^3 \times 48$]
*OGNConv**(3^3) → **l2**
[$32^3 \times 48$]</td></tr>
<tr><td></td><td>*OGNProp*
OGNConv(2^3) → **l3**
[$64^3 \times 32$]</td><td>*OGNProp*
OGNConv(4^3) → **l3**
[$64^3 \times 32$]</td><td>*OGNProp*
OGNConv(2^3)
[$64^3 \times 32$]
*OGNConv**(3^3) → **l3**
[$64^3 \times 32$]</td></tr>
</table>

Table 3.1: OGN architectures used in our experiments with autoencoders. *OGNConv* denotes up-convolution, *OGNConv** — convolution. Layer name followed by '→ **lk**' indicates that level *k* of an octree is predicted by a classifier attached to this layer.

$\mathbf{32^3}$	$\mathbf{64^3}$	$\mathbf{128^3}$	$\mathbf{256^3}$
$[4^3 \times 80]$	$[4^3 \times 96]$	$[4^3 \times 112]$	$[4^3 \times 112]$
Deconv (2^3) $[8^3 \times 64]$	Deconv (2^3) $[8^3 \times 80]$	Deconv (2^3) $[8^3 \times 96]$	Deconv (2^3) $[8^3 \times 96]$
Conv (3^3) → **l1** $[8^3 \times 64]$	Conv (3^3) $[8^3 \times 80]$	Conv (3^3) $[8^3 \times 96]$	Conv (3^3) $[8^3 \times 96]$
OGNProp *OGNConv* (2^3) → **l2** $[16^3 \times 48]$	Deconv (2^3) $[16^3 \times 64]$	Deconv (2^3) $[16^3 \times 80]$	Deconv (2^3) $[16^3 \times 80]$
OGNProp *OGNConv* (2^3) → **l3** $[32^3 \times 32]$	Conv (3^3) → **l1** $[16^3 \times 64]$	Conv (3^3) → **l1** $[16^3 \times 80]$	Conv (3^3) → **l1** $[16^3 \times 80]$
	OGNProp *OGNConv* (2^3) → **l2** $[32^3 \times 48]$	*OGNProp* *OGNConv* (2^3) → **l2** $[32^3 \times 64]$	*OGNProp* *OGNConv* (2^3) → **l2** $[32^3 \times 64]$
	OGNProp *OGNConv* (2^3) → **l3** $[64^3 \times 32]$	*OGNProp* *OGNConv* (2^3) → **l3** $[64^3 \times 48]$	*OGNProp* *OGNConv* (2^3) → **l3** $[64^3 \times 48]$
		OGNProp *OGNConv* (2^3) → **l4** $[128^3 \times 32]$	*OGNProp* *OGNConv* (2^3) → **l4** $[128^3 \times 32]$
			OGNProp *OGNConv* (2^3) → **l5** $[256^3 \times 32]$

Table 3.2: OGN decoder architectures used in shape from ID, and single-image 3D reconstruction experiments.

[$137 \times 137 \times 3$]
Conv (7×7) [$69 \times 69 \times 32$]
Conv (3×3) [$35 \times 35 \times 32$]
Conv (3×3) [$18 \times 18 \times 64$]
Conv (3×3) [$9 \times 9 \times 64$]
Conv (3×3) [$5 \times 5 \times 128$]
FC [1024]
FC [1024]
FC [$4^3 \times c$]

Table 3.3: Convolutional encoder used in the single-image 3D reconstruction experiment.

3.5.2 *Computational efficiency*

We start by empirically demonstrating that OGNs can be used at high resolutions when the voxel grid representation becomes impractical both because of the memory requirements and the runtime.

The number of elements in a voxel grid is uniquely determined by its resolution, and scales cubically as the latter increases. The number of elements in an octree depends on the data, leading to variable scaling rates: from constant for cubic objects aligned with the grid, to cubic for pathological shapes such as a three-dimensional checkerboard. In practice, octrees corresponding to real-world objects and scenes scale approximately quadratically, since they represent smooth two-dimensional surfaces in a three-dimensional space.

We empirically compare the runtime and memory consumption values for a dense network and OGN, for varying output resolution. Architectures of the networks are the same as used in Section 3.5.4 – three fully connected layers followed by an up-convolutional decoder. We performed the measurements on an NVidia TitanX Maxwell GPU, with 12Gb of memory. To provide actual measurements for dense networks at the largest possible resolution, we performed the comparison with batch size 1. The 512^3 dense network does not fit into memory even with batch size 1, so we extrapolated the numbers by fitting cubic curves.

Figure 3.4 and Table 3.4 show the results of the comparison. The OGN is roughly as efficient as its dense counterpart for low resolutions, but as the resolution grows, it gets drastically faster and consumes far less memory. At 512^3 voxel resolution, the OGN consumes almost two orders of magnitude less memory and runs 20 times faster.

	Memory, GB		Iteration time, s	
Resolution	Dense	OGN	Dense	OGN
32^3	0.33	**0.29**	**0.015**	0.016
64^3	0.50	**0.36**	0.19	**0.06**
128^3	1.62	**0.43**	0.56	**0.18**
256^3	9.98	**0.54**	3.89	**0.64**
512^3	74.28	**0.88**	41.3	**2.06**

Table 3.4: Memory consumption and iteration time of OGN and a dense network at different output resolutions. Batch size 1.

In order to further study this matter, we have designed a set of slim decoder networks that fit on a GPU in every resolution, including 512^3, both with an OGN and a dense representation. The architectures of those networks are similar to those from Table 3.2, but with only 1 channel in every convolutional layer, and a single fully-connected layer with 64 units in the encoder. The resulting measurements are shown in Figure 3.5 for memory consumption and runtime. To precisely quantify the scaling, we subtracted the constant amount of memory reserved on a GPU by *Caffe* (190 MB) from all numbers.

Both plots are displayed in the log-log scale, i.e., functions from the family $y = ax^k$ are straight lines. The slope of this line is determined by the exponent k, and the vertical shift by the coefficient a. In this experiment we are mainly interested in the slope, that is, how do the approaches scale with increasing output resolution. As a reference, we show dashed lines corresponding to perfect cubic and perfect quadratic scaling.

Starting from 64^3 voxel resolution both the runtime and the memory consumption scale almost cubically in case of dense networks. For this particular example, OGN scales even better than quadratically, but in general scaling of the octree-based representation depends on the specific data it is applied to.

To put these numbers into perspective, training OGN at 256^3 voxel output resolution takes approximately 5 days. Estimated training time of its dense counterpart would be almost a month. Even if the 512^3 voxel dense network would fit into memory, it would take many months to train.

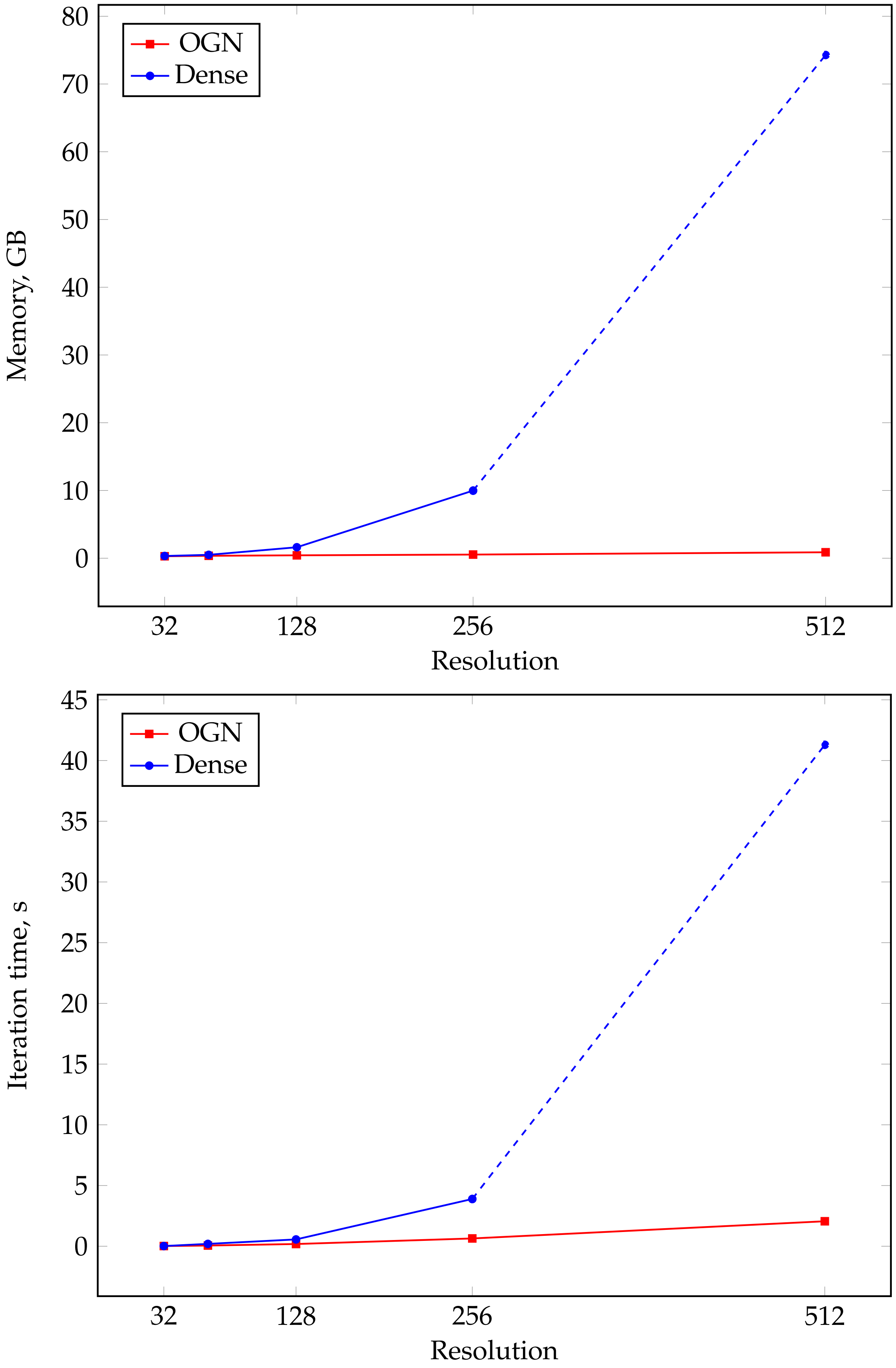
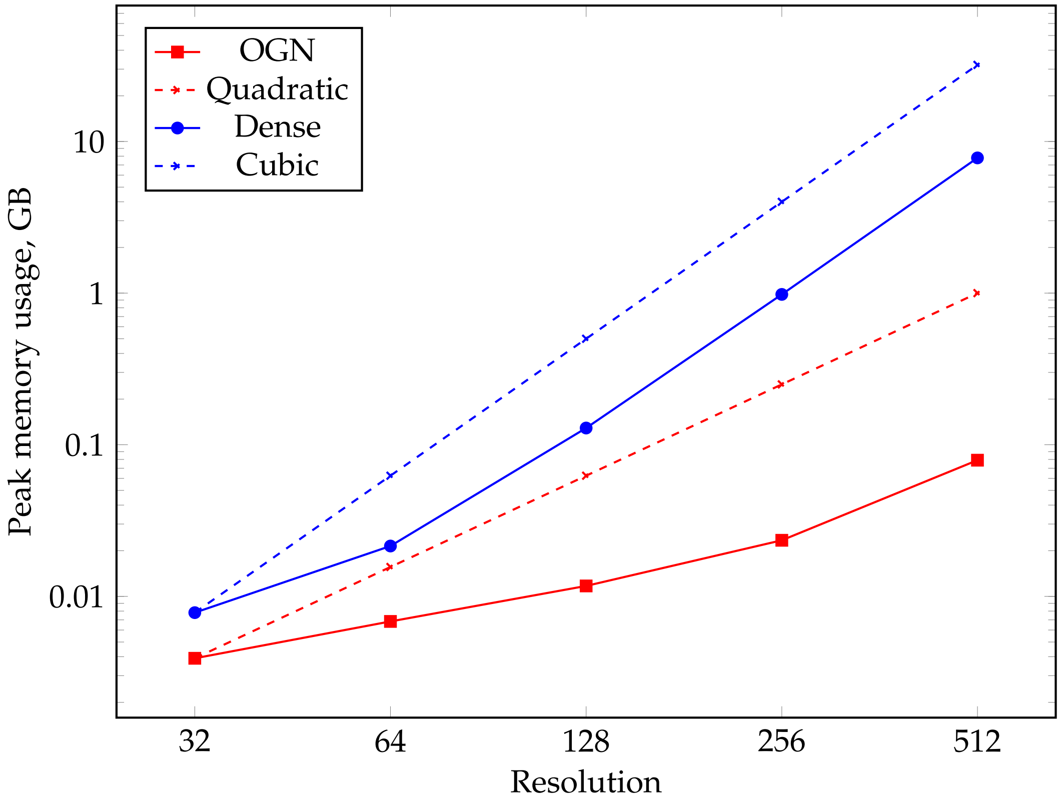


Figure 3.4: Memory consumption (left) and iteration time (right) of OGN and a dense network at different output resolutions. Forward and backward pass, batch size 1.



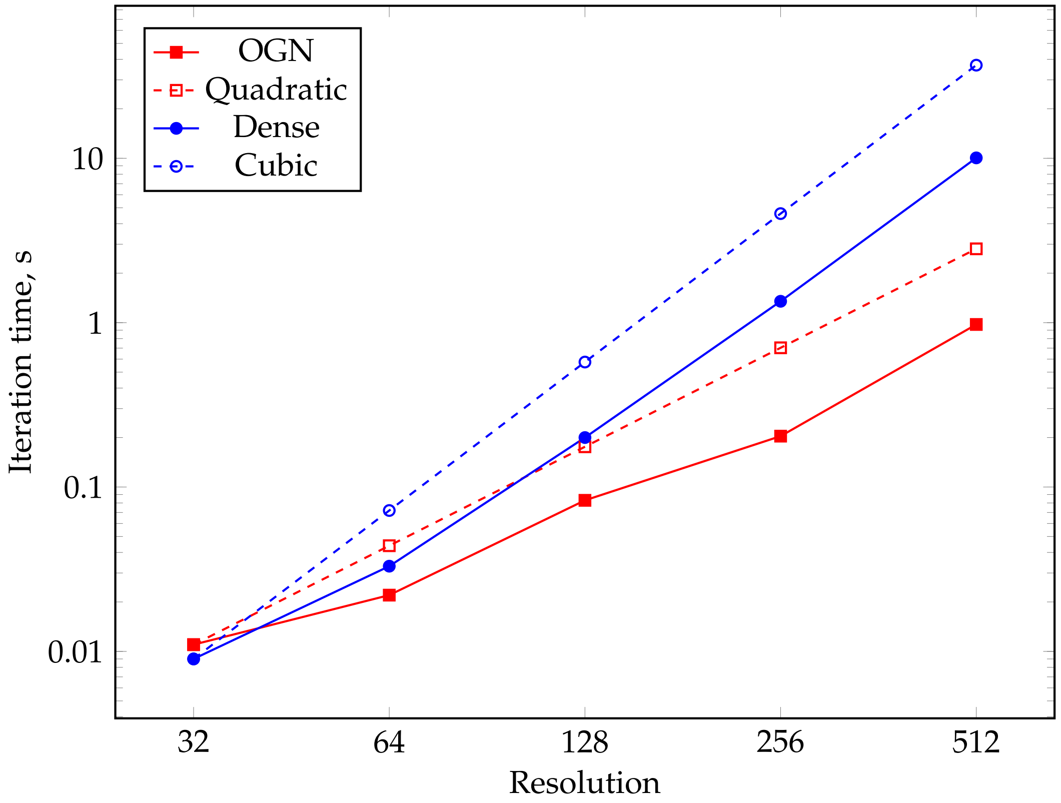


Figure 3.5: Memory consumption and iteration time for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling.

3.5.3 *Autoencoders*

Autoencoders and their variants are commonly used for representation learning from volumetric data [42, 150]. Therefore, we start by comparing the representational power of the OGN to that of dense voxel grid networks on the task of auto-encoding volumetric shapes.

We used the decoder architecture described in section 3.5.1 both for the OGN and the dense baseline. The architecture of the encoder is symmetric to the decoder. Both encoders operate on a dense voxel grid representation[5].

We trained the autoencoders on the ShapeNet-cars dataset in two resolutions: 32^3 and 64^3. We used 80% of the data for training, and 20% for testing. Quantitative results are summarized in Table 3.5. With predicted octree structure, there is no significant difference in performance between the OGN and the dense baseline.

Network	32^3	64^3
Dense	0.924	0.890
OGN+*Prop-known*	0.939	0.904
OGN+*Prop-pred*	0.924	0.884

Table 3.5: Quantitative results for OGN and dense autoencoders. Predictions were compared with the ground truth at the corresponding resolution, without upsampling.

3.5.3.1 *Flexibility of architecture choice*

To show that OGNs are not limited to up-convolutional layers with 2^3 filters, we evaluated two alternative 64^3 OGN auto-encoders: one with 4^3 up-convolutions and one with 2^3 up-convolutions interleaved with 3^3 convolutions. The results are summarized in Table 3.13. There is no significant difference between the architectures for this task. With larger filters, the network is roughly twice slower in our current implementation, so we used 2^3 filters in all further experiments.

Mode	2x2 filters	4x4 filters	IntConv
OGN+*Prop-known*	0.904	0.907	0.907
OGN+*Prop-pred*	0.884	0.885	0.885

Table 3.6: Using more complex architectures in 64^3 OGN autoencoders does not lead to significant performance improvements.

[5] Here, we focus on generating 3D shapes. Thus, we have not implemented an octree-based convolutional encoder. This could be done along the lines of Riegler *et al.* [141]

3.5.3.2 *Using known structure*

Interestingly, OGN with known tree structure outperforms the network based on a dense voxel grid, both qualitatively and quantitatively. An example of this effect can be seen in Figure 3.6: the dense autoencoder and our autoencoder with predicted propagation struggle with properly reconstructing the spoiler of the car. Intuitively, the known tree structure provides additional shape information to the decoder, thus simplifying the learning problem. In the autoencoder scenario, however, this may be undesirable if one aims to encode all information about a shape in a latent vector. In tasks like semantic segmentation, the input octree structure could help introduce shape features implicitly in the learning task.

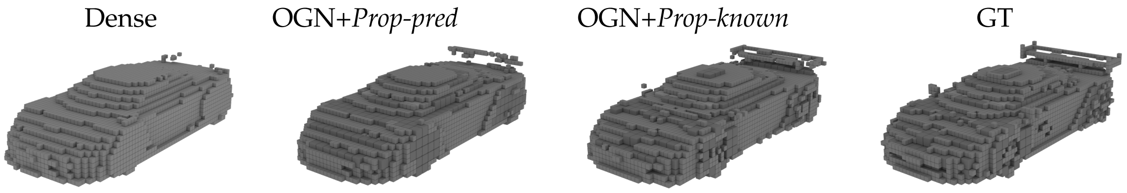


Figure 3.6: Using the known tree structure at test time leads to improved performance.

3.5.4 *3D shape from high-level information*

We trained multiple OGNs for generating shapes from high-level parameters similar to Dosovitskiy *et al.* [28]. In all cases the input of the network is a one-hot encoded object ID, and the output is an octree with the object shape.

	64	128	256	512
ShapeNet-cars	0.856	**0.901**	0.865	-
BlendSwap	0.535	0.649	0.880	**0.969**

Table 3.7: Evaluation of 3D shapes generated from high-level information. Lower-resolution predictions from ShapeNet-cars were upsampled to 256^3 ground truth, scenes from BlendSwap – to 512^3.

3.5.4.1 *ShapeNet-cars*

First, we trained on the whole ShapeNet-cars dataset in three resolutions: 64^3, 128^3 and 256^3. Example outputs are shown in Figure 3.7 and quantitative results are presented in Table 3.7. Similar to the two-dimensional case [28], the outputs are accurate in the overall shape, but lack some fine details. This is not due to the missing resolution, but due to general limitations of the training data and the learning task. Table 3.7 reveals that a resolution of 128^3 allows the reconstruction of a more accurate shape with more details than a resolution of 64^3. At an even higher resolution of 256^3, the overall performance decreased again. Even though the higher-resolution network is architecturally capable of performing better, it is not guaranteed to train better. Noisy gradients from outer high-resolution layers may hamper learning of deeper layers, resulting in an overall decline in performance. This problem is orthogonal to the issue of designing computationally efficient architectures, which we aim to solve here. More results for this experiment are shown in Section 3.5.4.4.

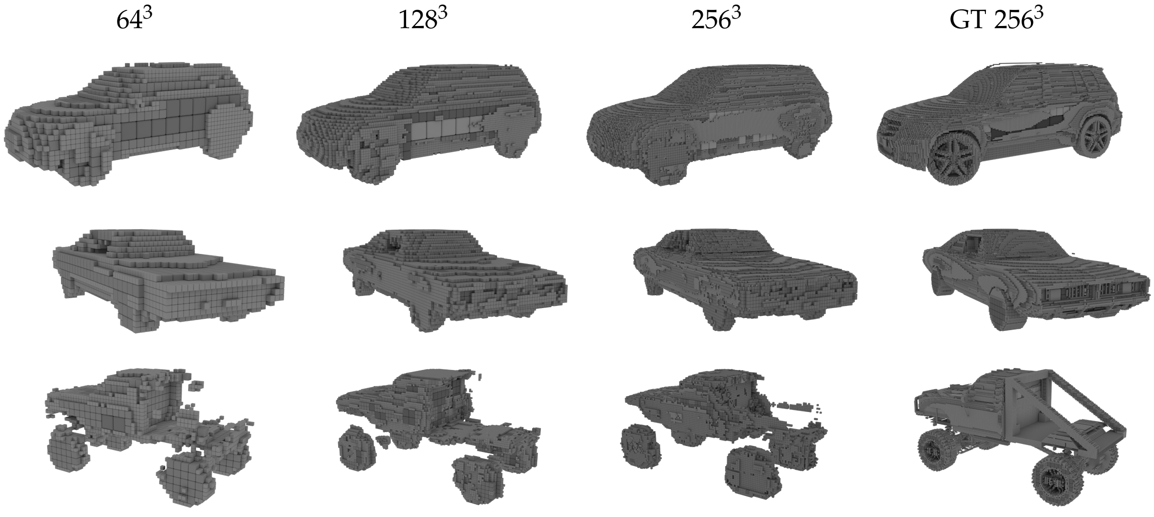


Figure 3.7: Training samples from the ShapeNet-cars dataset generated by our networks. Cells at different octree levels vary in size and are displayed in different shades of gray.

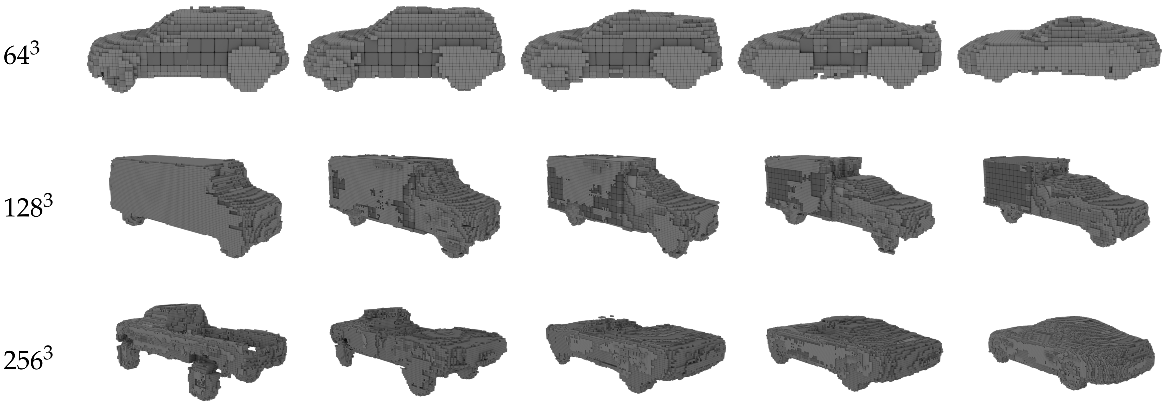


Figure 3.8: Our networks can generate previously unseen cars by interpolating between the dataset points, which demonstrates their generalization capabilities.

Notably, the network does not only learn to generate objects from the training dataset, but it can also generalize to unseen models. We demonstrate this by interpolating between pairs of one-hot input ID vectors. Figure 3.8 shows that for all intermediate input values the network produces consistent output cars, with the style being smoothly changed between the two training points.

3.5.4.2 *BlendSwap*

To additionally showcase the benefit of using higher resolutions, we trained OGNs to fit the BlendSwap dataset containing 4 whole scenes. In contrast to the ShapeNet-cars dataset, such amount of training data does not allow for any generalization. The experiment aims to show that OGNs provide sufficient resolution to represent such high-fidelity shape data.

Figure 3.9 shows the generated scenes. In both examples, 64^3 and 128^3 resolutions are inadequate for representing the details. For the bottom scene, even the 256^3 resolution still struggles with fine-grained details. This example demonstrates that tasks like end-to-end learning of scene reconstruction requires high-resolution representations, and the OGN is an architecture that can provide such resolutions.

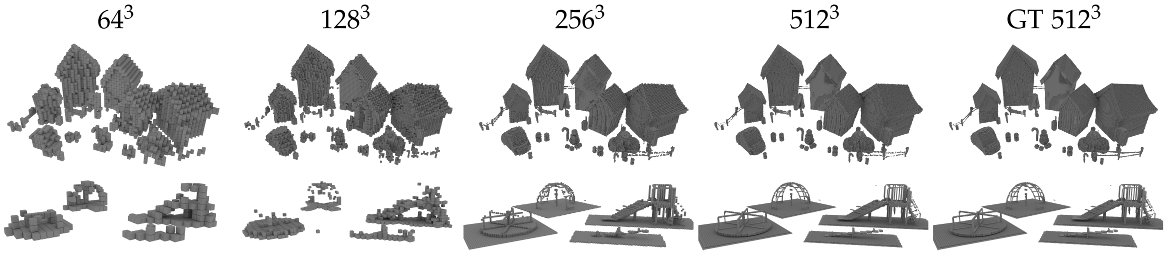


Figure 3.9: OGN is used to reproduce large-scale scenes from the dataset, where high resolution is crucial to generate fine-grained structures.

These qualitative observations are confirmed quantitatively in Table 3.7. Higher output resolutions allow for more accurate reconstruction of the samples in the dataset.

3.5.4.3 *MPI-FAUST*

To additionally showcase the benefit of using higher resolutions, we trained OGNs to fit the MPI-FAUST dataset [6]. It contains 300 high-resolution scans of human bodies of 10 different people in 30 different poses. Same as with the BlendSwap, the trained networks cannot generalize to new samples due to the low amount of training data.

64	128	256
0.890	0.933	**0.969**

Table 3.8: 3D shape from high-level information on the FAUST dataset. Lower-resolution predictions were upsampled to 256^3 ground truth.

Figure 3.10 and Table 3.8 demonstrate qualitative and quantitative results respectively. Human models from MPI-FAUST include finer details than cars from ShapeNet, and therefore benefit from the higher resolution.

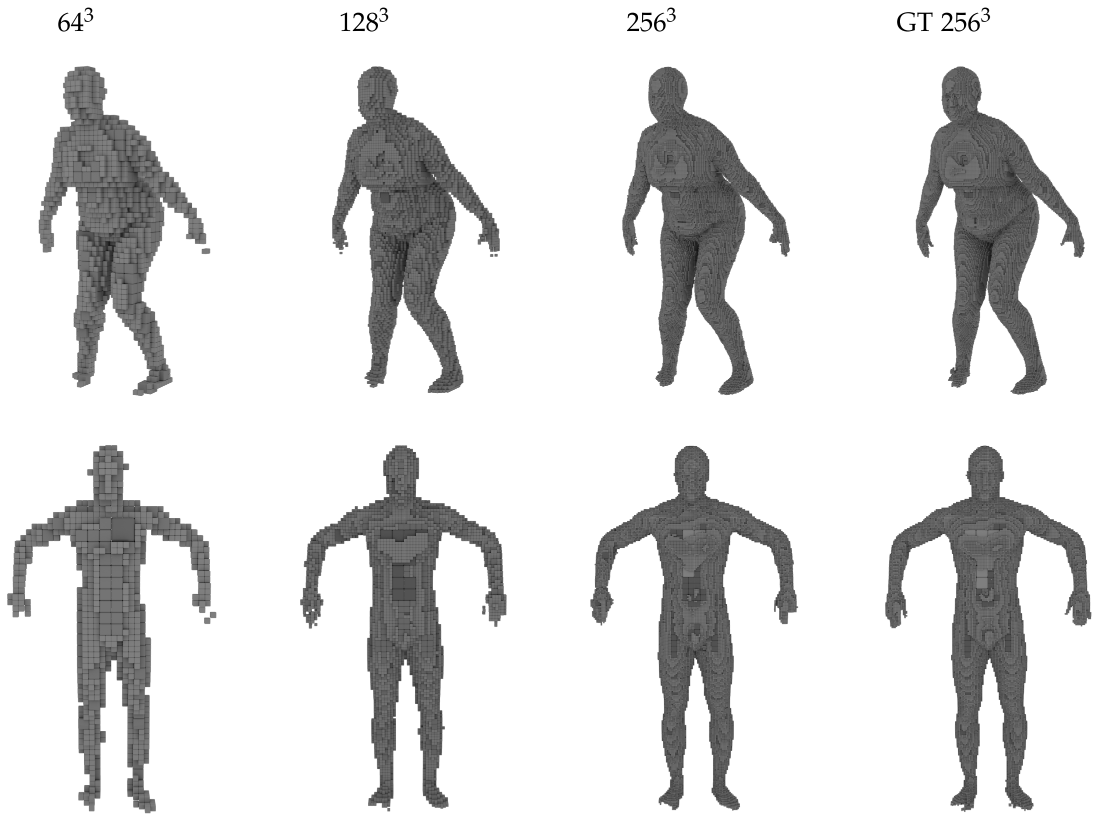


Figure 3.10: Training samples from the FAUST dataset reconstructed by OGN.

Dataset	128^3	256^3
Shapenet-cars (full)	0.901	0.865
Shapenet-cars (subset)	0.922	0.931

Table 3.9: There is no drop in performance in higher resolution, when training on a subset of the Shapenet-cars dataset.

3.5.4.4 *Fitting reduced ShapeNet-cars*

To better understand the performance drop at 256^3 resolution observed in Table 3.7, we performed an additional experiment on the ShapeNet-Cars dataset. We trained an OGN for generating car shapes from their IDs on a reduced version of ShapeNet-Cars, including just 500 first models from the dataset. Quantitative results for different resolutions, along with the results for the full dataset, are shown in Table 3.9. Interestingly, when training on the reduced dataset, high resolution is beneficial. This is further supported by examples shown in Figure 3.11 – when training on the reduced dataset, the higher-resolution model contain more fine details. Overall, these results support our hypothesis that the performance drop at higher resolution is not due to the OGN architecture, but due to the difficulty of fitting a large dataset at high resolution.

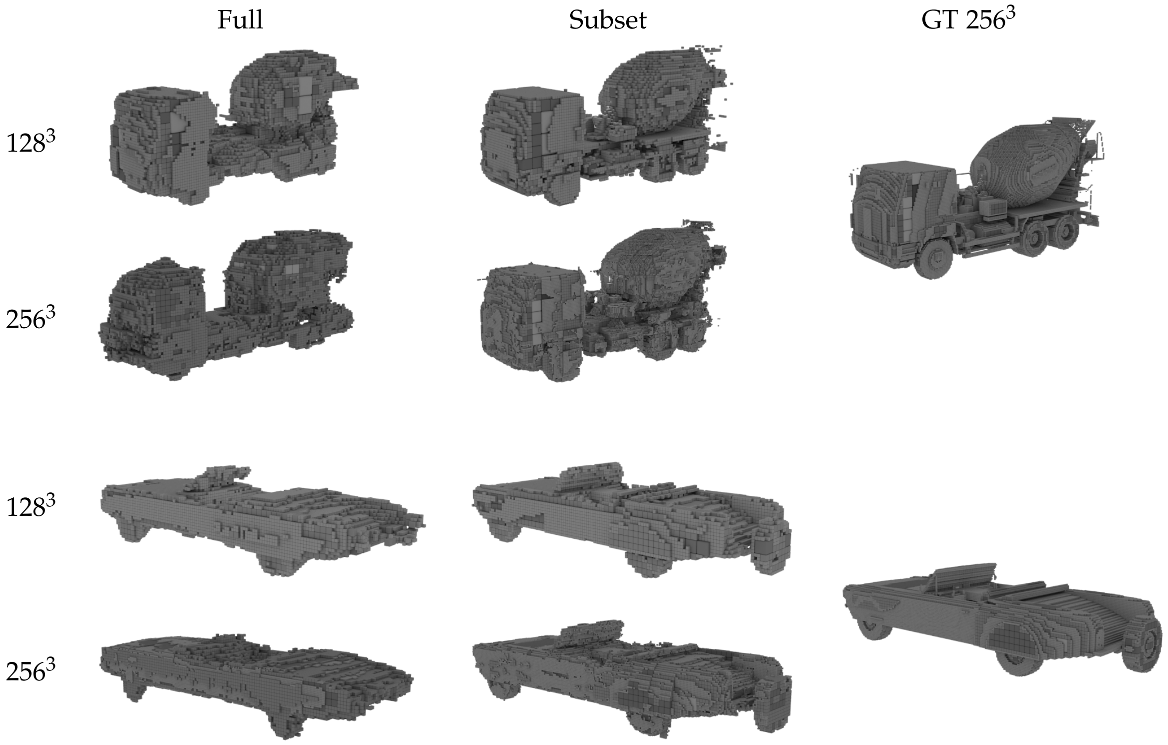


Figure 3.11: When training on a subset of the Shapenet-cars datset, higher resolution models contain more details.

3.5.5 *Single-image 3D reconstruction*

In this experiment we trained networks with our OGN decoder on the task of single-view 3D reconstruction. To demonstrate that our dense voxel grid baseline, as already used in the autoencoder experiment, is a strong baseline, we compare to the approach by Choy *et al.* [20]. This approach operates on 32^3 voxel grids, and we adopt this resolution for our first experiment.

To ensure a fair comparison, we trained networks on ShapeNet-all, the exact dataset used by Choy *et al.* [20]. Following the same dataset splitting strategy, we used 80% of the data for training, and 20% for testing. As a baseline, we trained a network with a dense decoder which had the same configuration as our OGN decoder. Table 3.10 shows that compared to single-view reconstructions from [20], both the OGN and the baseline dense network compare favorably for most of the classes. In conclusion, the OGN is competitive with voxel-grid-based networks on the complex task of single-image class-specific 3D reconstruction.

We also evaluated the effect of resolution on the ShapeNet-cars dataset. Figure 3.12 shows that OGNs learned to infer the 3D shapes of cars in all cases, and that high-resolution predictions are clearly better than the 32^3 models commonly used so far. This is backed up by quantitative results shown in Table 3.11: 32^3 results are significantly worse than the rest. At 256^3 performance drops again for the same reasons as in the decoder experiment in section 3.5.4.1.

Category	R2N2 [20]	OGN	Dense
Plane	0.513	**0.587**	0.570
Bench	0.421	**0.481**	0.481
Cabinet	0.716	0.729	**0.747**
Car	0.798	0.816	**0.828**
Chair	0.466	**0.483**	0.481
Monitor	0.468	0.502	**0.509**
Lamp	0.381	**0.398**	0.371
Speaker	**0.662**	0.637	0.650
Firearm	0.544	**0.593**	0.576
Couch	0.628	0.646	**0.668**
Table	0.513	0.536	**0.545**
Cellphone	0.661	**0.702**	0.698
Watercraft	0.513	**0.632**	0.550
Mean	0.560	**0.596**	0.590

Table 3.10: Single-view 3D reconstruction results on the 32^3 version of ShapeNet-all from Choy *et al.* [20] compared to OGN and a dense baseline. OGN is competitive with voxel-grid-based networks.

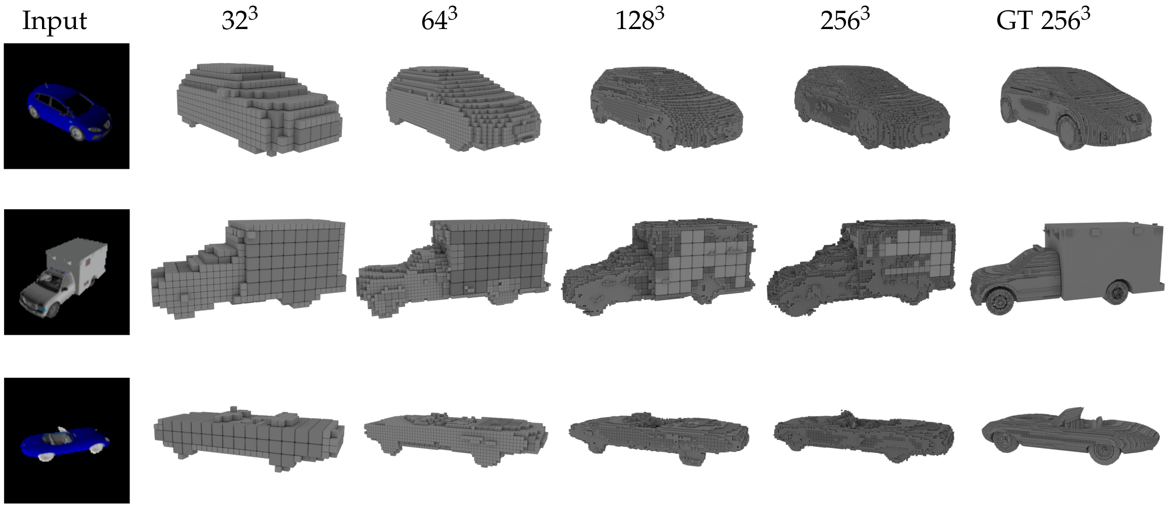


Figure 3.12: Single-image 3D reconstruction on the ShapeNet-cars dataset using OGN in different resolutions.

Resolution	32	64	128	256
Single-view 3D	0.641	0.771	**0.782**	0.766

Table 3.11: Single-image 3D reconstruction results on ShapeNet-cars. Low-resolution predictions are upsampled to 256^3. Commonly used 32^3 models are significantly worse than the rest.

3.5.6 *Shift invariance*

To study the effect of shifts, we trained two fully convolutional autoencoders - one with an OGN decoder, and one with a dense decoder - on 64^3 models, with lowest feature map resolution 4^3 (so the networks should be perfectly invariant to shifts of 16 voxels). Both were trained on non-shifted Shapenet-Cars, and tested in the *Prop-pred* mode on models shifted by a different number of voxels along the z-axis. The results are summarized in Table 3.12.

Shift (voxels)	OGN	Dense
0	0.935	0.932
1	0.933	0.93
2	0.929	0.925
4	0.917	0.915
8	0.906	0.904

Table 3.12: Fully-convolutional networks tested on shifted data. Even though not shift invariant by design, OGN shows robust performance.

There is no significant difference between OGN and the dense network. A likely reason is that different training models have different octree structures, which acts as an implicit regularizer. The network learns the shape, but remains robust to the exact octree structure.

3.5.7 *Train/test modes*

In section 3.4.4, we described how we use the two propagation modes (*Prop-known* and *Prop-pred*) during training and testing. Here we motivate the proposed regimes, and show additional results with other combinations of propagation modes.

When the structure of the output tree is not known at test time, we train the networks until convergence with *Prop-known*, and then additionally fine-tune with *Prop-pred* - line 4 in Table 3.6. Without this fine-tuning step (line 2), there is a decrease in performance, which is more significant when using larger convolutional filters. Intuitively, this happens because the network has never seen erroneous propagations during training, and does not now how to deal with them at test time.

When the structure of the output is known at test time, the best strategy is to simply train in *Prop-known*, and test the same way (line 1). Additional fine-tuning in the *Prop-pred* mode slightly hurts performance in this case (line 3). The overall conclusion is not surprising: the best results are obtained when training networks in the same propagation modes, in which they are later tested.

Training	Testing	2^3 filters	4^3 filters	IntConv
Known	*Known*	0.904	0.907	0.907
Known	*Pred*	0.862	0.804	0.823
Pred	*Known*	0.898	0.896	0.897
Pred	*Pred*	0.884	0.885	0.885

Table 3.13: Reconstruction quality for autoencoders with different decoder architectures: 2^3 up-convolutions, 4^3 up-convolutions, and 2^3 up-convolutions interleaved with 3^3 convolutions, using different configurations of *Prop-known* and *Prop-pred* propagation modes.

3.6 CONCLUSIONS

We presented a convolutional decoder architecture for generating high-resolution 3D outputs represented as octrees. We have demonstrated that this architecture is flexible in terms of the exact layer configuration, and that it provides the same accuracy as dense voxel grids in low resolution. At the same time, it scales much better to higher resolutions, both in terms of memory and runtime.

This architecture enables end-to-end deep learning to be applied to tasks that appeared unfeasible before. In particular, learning tasks that involve 3D shapes, such as 3D object and scene reconstruction, are likely to benefit from it.

4 WHAT DO SINGLE-VIEW 3D RECONSTRUCTION NETWORKS LEARN?

The text of this chapter was largely copied from the following paper.

> Maxim Tatarchenko*, Stephan R. Richter*, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox (*indicates equal contribution). "What Do Single-view 3D Reconstruction Networks Learn?" In: *CVPR*, 2019.

Stephan R. Richter contributed by training the Matryoshka Networks and the AtlasNet baselines. He also implemented and trained the retrieval approach and rendered the qualitative reconstruction examples. Zhuwen Li contributed by evaluating the Pixel2Mesh baseline. All co-authors contributed to the project discussions as well as the final paper text editing.

* * *

4.1 INTRODUCTION

Object-based single-view 3D reconstruction calls for generating the 3D model of an object given a single image. Consider the motorcycle in Figure 4.1. Inferring its 3D structure requires a complex process that combines low-level image cues, knowledge about structural arrangement of parts, and high-level semantic information. We refer to the extremes of this spectrum as *reconstruction* and *recognition*. *Reconstruction* implies reasoning about the 3D structure of the input image using cues such as texture, shading, and perspective effects. *Recognition* amounts to classifying the input image and retrieving the most suitable 3D model from a database, in our example finding a pre-existing 3D model of a motorcycle based on the input image.

While various architectures and 3D representations have been proposed in the literature, existing methods for single-view 3D understanding all use an encoder-decoder structure, where the encoder maps the input image to a latent representation and the decoder is supposed to perform non-trivial reasoning about the 3D structure of the output space. To solve the task, the overall network is expected to incorporate low-level as well as high-level information.

In this work, we analyze the results of state-of-the-art encoder-decoder methods (AtlasNet [48], Matryoshka Networks [138], Octree Generating Networks described in Chapter 3) and find that they rely primarily on recognition to address the single-view 3D reconstruction task, while showing only limited reconstruction abilities. To support this claim, we design two pure recognition baselines: one that combines 3D shape clustering and image classification and one that performs image-based 3D shape retrieval. Based on these, we demonstrate that the performance of modern

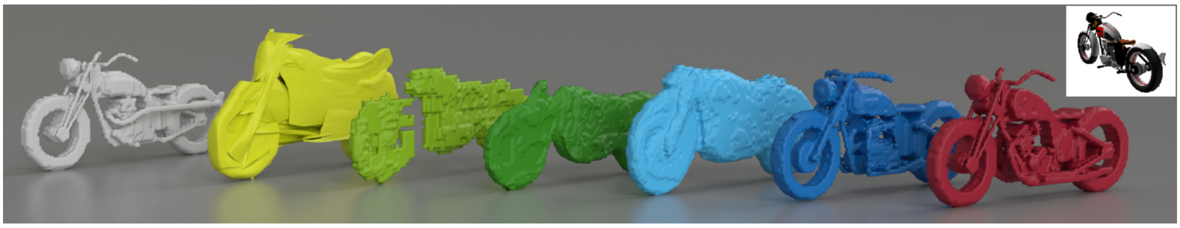


Figure 4.1: We provide evidence that state-of-the-art single-view 3D reconstruction methods (AtlasNet (light green, 0.38 IoU) [48], OGN (green, 0.46 IoU), Matryoshka Networks (dark green, 0.47 IoU) [138]) do not actually perform reconstruction but image classification. We explicitly design pure recognition baselines (Clustering (light blue, 0.46 IoU) and Retrieval (dark blue, 0.57 IoU)) and show that they produce similar or better results both qualitatively and quantitatively. For reference, we show the ground truth (white) and a nearest neighbor from the training set (red, 0.76 IoU). The inset shows the input image.

convolutional networks for single-view 3D reconstruction can be surpassed even without explicitly inferring the 3D structure of objects. In many cases the predictions of the recognition baselines are not only better quantitatively, but also appear visually more appealing, as demonstrated in Figure 4.1.

We argue that the dominance of recognition in convolutional networks for single-view 3D reconstruction is a consequence of certain aspects of popular experimental procedures, including dataset composition and evaluation protocols. These allow the network to find a shortcut solution, which happens to be image recognition.

4.2 RELATED WORK

Historically, single-image 3D reconstruction has been approached via shape-from-shading [30, 62, 195]. More exotic cues for reconstruction are texture [101] and defocus [34]. These techniques only reason about visible parts of a surface using a single depth cue. More general approaches for depth estimation from a single monocular image use multiple cues as well as structural knowledge to infer an estimate of the depth of visible surfaces. Saxena *et al*. [148] estimated depth from a single image by training an MRF on local and global image features. Oswald *et al*. [122] solved the same problem with interactive user input. Hoiem *et al*. [61] used recognition together with simple geometric assumptions to construct 3D models from a single image. Karsch *et al*. [75] proposed a non-parametric framework that uses part- and object-level recognition to assemble an estimate from a database of images and corresponding depth maps. More recently, significant advances have been made in monocular depth estimation by employing convolutional networks [16, 32, 43, 98, 189].

This chapter focuses on methods that not only reason about the 3D structure of object parts visible in the input image, but also hallucinate the invisible parts using priors learned from data. Tulsiani *et al*. [168] approached this task with deformable models for specific object categories. Most of the recent methods trained convolutional networks that map 2D images to 3D shapes using direct 3D supervision. These methods were covered in the Related Work section of Chapter 3.

Recently, there has been a trend towards using weaker forms of supervision for single-view 3D shape prediction with convolutional networks. Multiple approaches [66, 76, 137, 169, 191, 204] trained shape regressors by comparing projections of ground-truth and predicted shapes. Kanazawa *et al.* [74] predicted deformations from mean shapes trained from multiple learning signals.

4.3 RECONSTRUCTION VS. RECOGNITION

Single-view 3D understanding is a complex task that requires interpreting visual data both geometrically and semantically. In fact, these two modes are not disjoint, but span a spectrum from pure geometric reconstruction to pure semantic recognition.

Reconstruction implies per-pixel reasoning about the 3D structure of the object shown in the input image, which can be achieved by using low-level image cues such as color, texture, shading, perspective, shadows, and defocus. This mode does not require semantic understanding of the image content.

Recognition is an extreme case of using semantic priors: it operates on the level of whole objects and amounts to classifying the object in the input image and retrieving a corresponding 3D shape from a database. While it provides a robust prior for reasoning about the invisible parts of objects, this kind of purely semantic solution is only valid if the new object can be explained by an object in the database.

As reconstruction and recognition represent opposing ends of a spectrum, resorting exclusively to either is unlikely to produce the most accurate 3D shapes, since both ignore valuable information present in the input image. It is thus commonly hypothesized that a successful approach to single-view 3D reconstruction needs to combine low-level image cues, structural knowledge, and high-level object understanding [149].

In the following sections, we argue that current methods tackle the problem predominantly using recognition.

4.4 CONVENTIONAL SETUP

In this section, we analyze current methods for single-view 3D reconstruction and their relation to reconstruction and recognition. We employ a standard setup for single-view 3D shape estimation. We use the ShapeNet dataset [11]. Unlike several recent approaches, which evaluated only on the 13 largest classes, we deliberately use all 55 classes, as was done in [192]. This allows us to investigate how the number of samples within a class influences shape estimation performance. Within each class, the shapes are randomly split into training, validation, and test sets, containing 70%, 10%, and 20% of the samples respectively. Every shape was rendered using the ShapeNet-Viewer from five uniformly sampled viewpoints

$(\theta_{azimuth} \in [0^\circ, 360^\circ), \theta_{elevation} \in [0^\circ, 50^\circ))$. The distance to the camera was set such that each rendered shape roughly fits the frame. We rendered RGB images of size 224×224, which were downsampled to the input resolution that is required by each method.

All 3D shapes have a consistent canonical orientation and are represented as 128^3 voxel grids. Using high-resolution ground truth (compared to the often used 32^3 voxel grids) is crucial for evaluating a method's ability to reconstruct fine detail. Evaluating on a higher resolution than 128^3 does not offer additional benefits, since the performance of state-of-the-art methods saturates at this level [138], while training and evaluation become much more costly. We follow standard procedure and measure shape similarity with the mean Intersection over Union (mIoU) metric, aggregating predictions within semantic classes [20, 33, 53, 138, 152, 191].

4.4.1 *Existing approaches*

We base our experiments on modern convolutional networks that predict high-resolution 3D models from a single image. A taxonomy of approaches arises by categorizing them based on their output representation: voxel grids, meshes, point clouds, and depth maps. To this end, we chose state-of-the-art methods that cover the dominant output representations or have clearly shown to outperform other related representations for our evaluation.

We use Octree Generating Networks (OGN) introduced in Chapter 3 as the representative method that predicts the output directly on an (efficient) voxel grid. Compared to earlier works [20] that operate on this output representation, OGN allows predicting higher-resolution shapes by using octrees to represent the occupied space efficiently. We evaluate AtlasNet [48] as the representative approach for surface-based methods. AtlasNet predicts a collection of parametric surfaces and constitutes the state-of-the-art among methods that operate on this output representation. It was shown to outperform the only approach that directly produces point clouds as output [33], as well as another octree-based approach [53]. Finally, we evaluate the current state-of-the-art among depth-based methods, Matryoshka Networks [138]. Matryoshka Networks use a shape representation that is composed of multiple, nested depth maps, which are fused into a single output object.

For IoU-based evaluation of the surface predictions from AtlasNet, we project them to depth maps, which we further fuse to a volumetric representation. In our experiments, this approach reliably closed holes in the reconstructed surfaces while retaining fine details. For surface-based evaluation metrics, we use the marching cubes algorithm [104] to extract meshes from volumetric representations.

4.4.2 *Recognition baselines*

We implemented two straightforward baselines that approach the problem purely in terms of recognition. The first is based on clustering of the training shapes in conjunction with an image classifier; the second performs database retrieval.

Clustering. In this baseline, we cluster the training shapes into K sub-categories using the K-means algorithm [107]. Since using 128^3 voxelizations as feature vectors for clustering is too costly, we run the algorithm on 32^3 voxelizations flattened into a vector. Once the cluster assignments are determined, we switch back to working with high-resolution models.

Within each of the K clusters, we calculate the mean shape as

$$\hat{m}_k = \frac{1}{N_k} \sum_{n=0}^{N_k} v_n, \tag{4.1}$$

where v_n is one of the N_k shapes belonging to the k-th cluster. We threshold the mean shapes at τ_k, where the optimal τ_k value is determined by maximizing the average IoU over the models belonging to the k-th cluster:

$$\tau_k = \arg\max_{\tau} \frac{1}{N_k} \sum_{n=0}^{N_k} \mathrm{IoU}(\hat{m}_k > \tau, v_n), \tag{4.2}$$

where the thresholding operation is applied per voxel. We enumerate τ in the interval $[0.05, 0.5]$ with a step size of 0.05 to find the optimal threshold. We set $K = 500$.

Since correspondences between images and 3D shapes are known for the training set, images can be readily matched with the respective cluster k. Subsequently, we train a 1-of-K classifier that assigns images to cluster labels. At test time, we set the mean shape of the predicted cluster as the inferred solution. For classification, we use the ResNet-50 architecture [55], pre-trained on the ImageNet dataset [26], and fine-tuned for 30 epochs on our data.

Retrieval. Our retrieval baseline is inspired by the work of Li *et al.* [94], which learns to embed images and shapes in a joint space. The embedding space is constructed from the pairwise similarity matrix of all 3D shapes in the training set by compressing each row of the matrix to a low-dimensional descriptor via Multi-Dimensional Scaling [87] with Sammon mapping [147]. To compute the similarity of two arbitrary shapes, Li *et al.* employ the lightfield descriptor [14]. To embed images in the space spanned by the shape descriptors, a convolutional network [86] is trained to map images to the descriptor given by the corresponding shape in the training set. During training, the network optimizes the Euclidean distance between predicted and ground-truth descriptors.

We adapt the work of Li *et al.* in several ways. As with our clustering baseline, we determine the similarity between two shapes via the IoU of their 32^3 voxel grid representation. We then compute a low-dimensional descriptor via principal component analysis. We further use a larger descriptor (512 vs. 128) and a network with

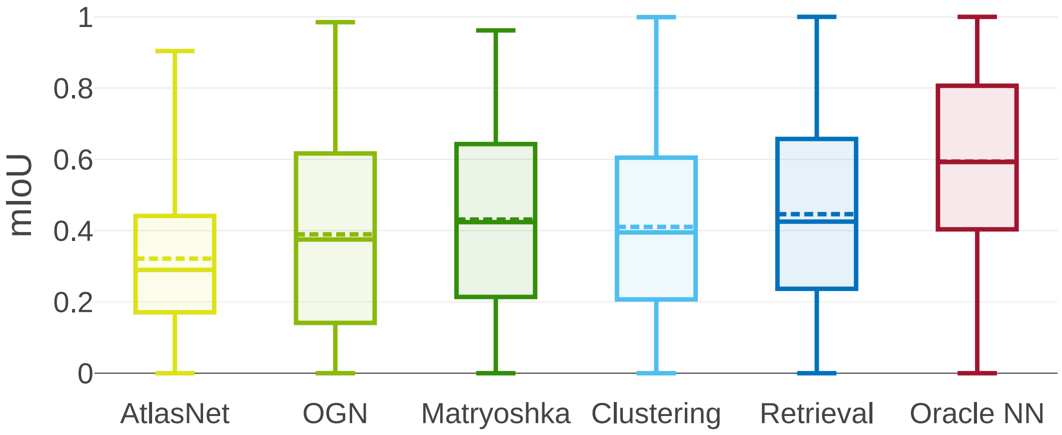


Figure 4.2: Comparison by mean IoU over the dataset. The box corresponds to the second and third quartile. The solid line in the box depicts the median; the dashed line the mean. Whiskers mark the minimum and maximum values, respectively.

larger capacity (ResNet-50 [55], pre-trained on ImageNet [26], without fixing any layers during fine-tuning). Finally, instead of minimizing the Euclidean distance, we maximize the cosine similarity between descriptors during training.

Oracle nearest neighbor. To gain more insight into the characteristics of the dataset, we evaluate an Oracle Nearest Neighbor (Oracle NN) baseline. For each of the test 3D shapes, we find the closest shape from the training set in terms of IoU. This method cannot be applied in practice, but gives an upper bound on how well a retrieval method can solve the task.

4.4.3 *Analysis*

We start by conducting a standard comparison of all methods in terms of their mean IoU scores. The results are summarized in Figure 4.2. We find that state-of-the-art methods, despite being backed by different architectures, perform at a remarkably similar level. Interestingly, the retrieval baseline, a pure recognition method, outperforms all other approaches both in terms of mean and median IoU. The simple clustering baseline is competitive and outperforms both AtlasNet and OGN. We further observe that a perfect retrieval method (Oracle NN) performs significantly better than all other methods. Strikingly, the variance in the results is extremely high (between 35% and 50%) for all methods. This implies that quantitative comparisons that rely solely on the mean IoU do not provide a full picture at this level of performance. To shed more light on the behavior of the methods, we proceed with a more detailed analysis.

Per-class analysis. The similarity in average accuracy cannot be attributed to methods specializing in different subsets of classes. In Figure 4.3 we observe consistent relative performance between methods across different classes. The retrieval baseline achieves the best results for 30 out of 55 classes. The classes are sorted from left

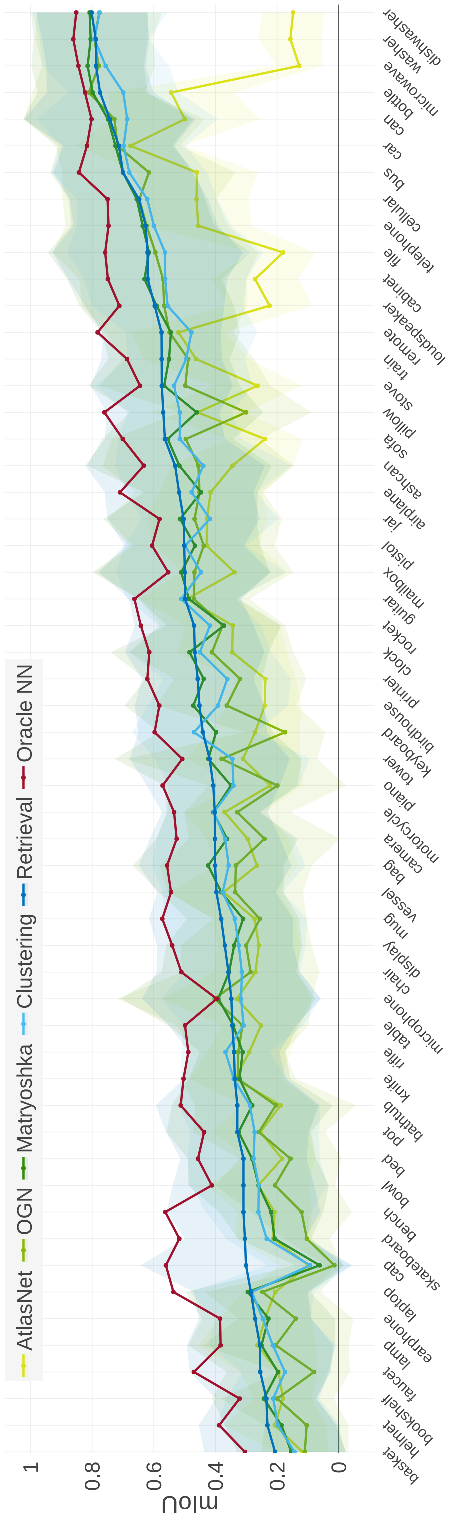


Figure 4.3: Comparison by mIoU per class. Overall, the methods exhibit consistent relative performance across different classes. The retrieval baseline produces the best reconstructions for the majority of classes. The variance is high for all classes and methods.

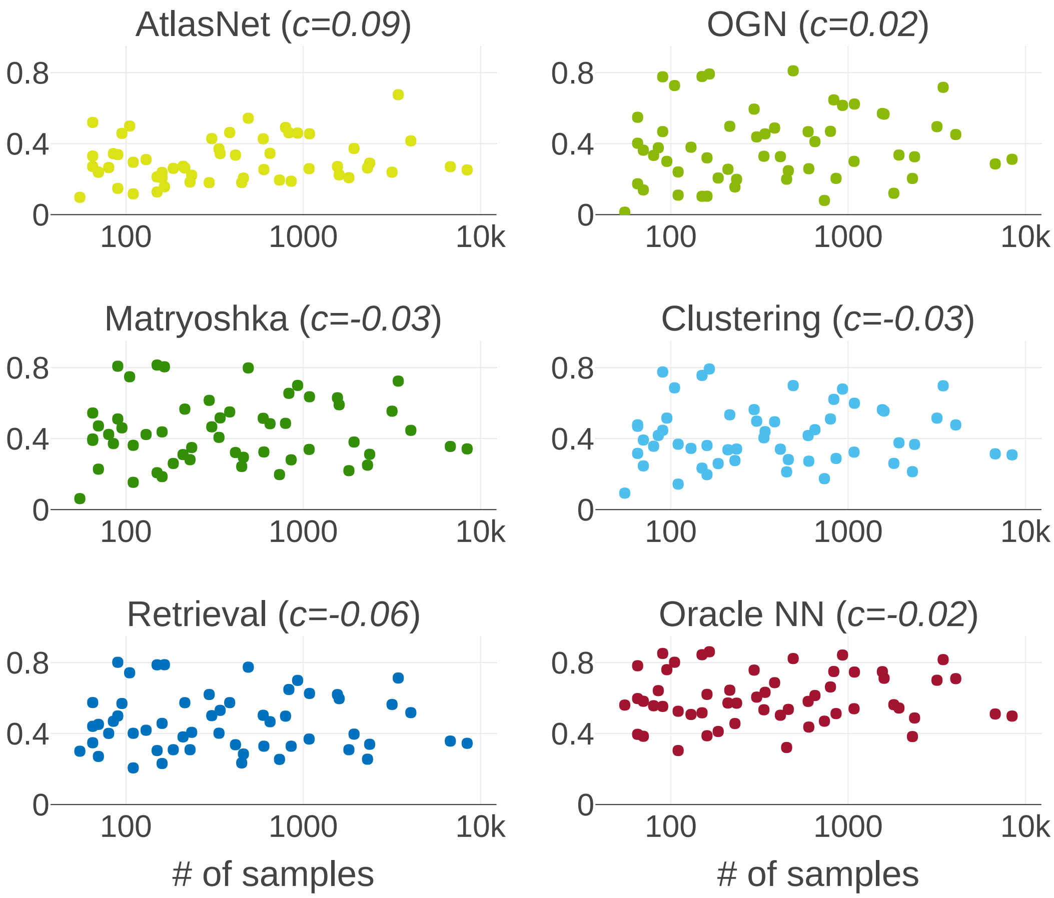


Figure 4.4: mIoU versus number of training samples per class. We find no correlation between the number of samples within a class and the mIoU score for this class. The correlation coefficient c is close to zero for all methods.

to right in ascending order according to the performance of the retrieval baseline. The variance is high for all classes and all methods.

One might assume that the per-class performance depends on the number of training samples that are available for a class. However, we find no correlation between the number of samples in a class and its mean IoU score; see Figure 4.4. The correlation coefficient between the two quantities is close to zero for all methods. This implies that there is no justification for only using 13 out of the 55 classes, as was done in many prior works [20, 33, 48, 138, 191].

The quantitative results are backed by qualitative results shown in Figure 4.5. For most classes, there is no significant visual difference between the predictions of the decoder-based methods and our clustering baseline. Clustering fails when the sample is far from the mean shape of the cluster, or when the cluster itself cannot be described well by the mean shape (this is often the case for chairs or tables because of thin structures that get averaged out in the mean shape). The predictions of the retrieval baseline look more appealing in most cases due to the presence of fine details, even though these details are not necessarily correct.

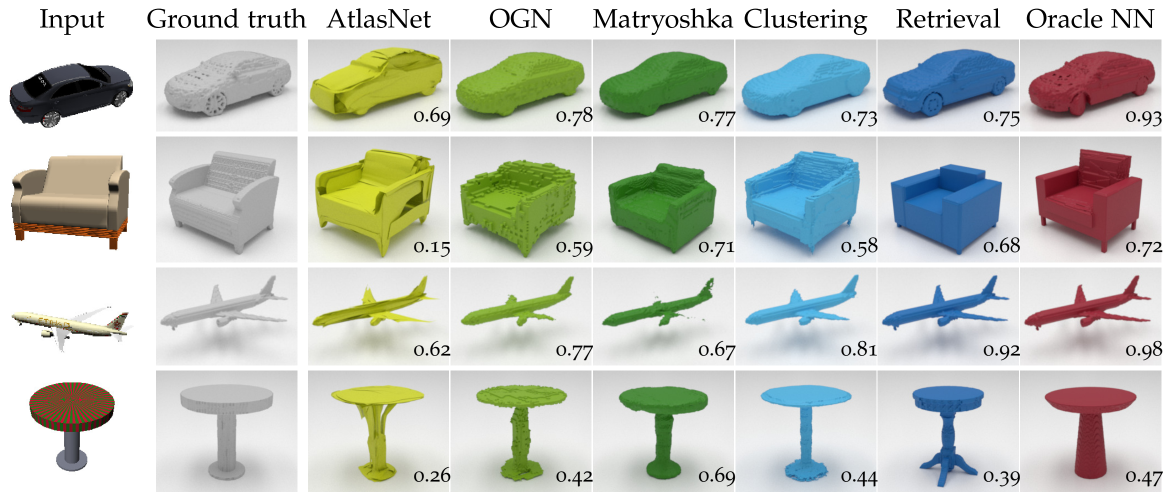


Figure 4.5: Qualitative results. Our clustering baseline produces shapes at a quality comparable to state-of-the-art approaches. Our retrieval baseline returns high-fidelity shapes by design, although details may not be correct. Numbers in the bottom right corner of each sample indicate the IoU.

Statistical evaluation. To further investigate the hypothesis that convolutional networks bypass reconstruction via image recognition, we visualize the histograms of IoU scores for all object classes in Figure 4.6. Although the distributions differ between classes, the within-class distributions of decoder-based methods and recognition baselines are surprisingly similar. For reference, we also plot the results of the Oracle NN baseline, which, for many classes, differs substantially. To verify this observation rigorously, we perform the Kolmogorov-Smirnov statistical test [113] on the 50-binned versions of the histograms for all classes and all pairs of methods. The null hypothesis assumes that two distributions exhibit no statistically significant difference. We visualize the results of the test in Figure 4.7. Each cell in the displayed table represents the p-value for a specific object class and a specific pair of methods. Green cells correspond to classes for which the statistical test does not allow to reject the null hypothesis, *i.e.*, where the p-value is larger than 0.05. All other cells are colored orange. We find that for decoder-based methods and recognition baselines the null hypothesis cannot be rejected for the vast majority of classes.

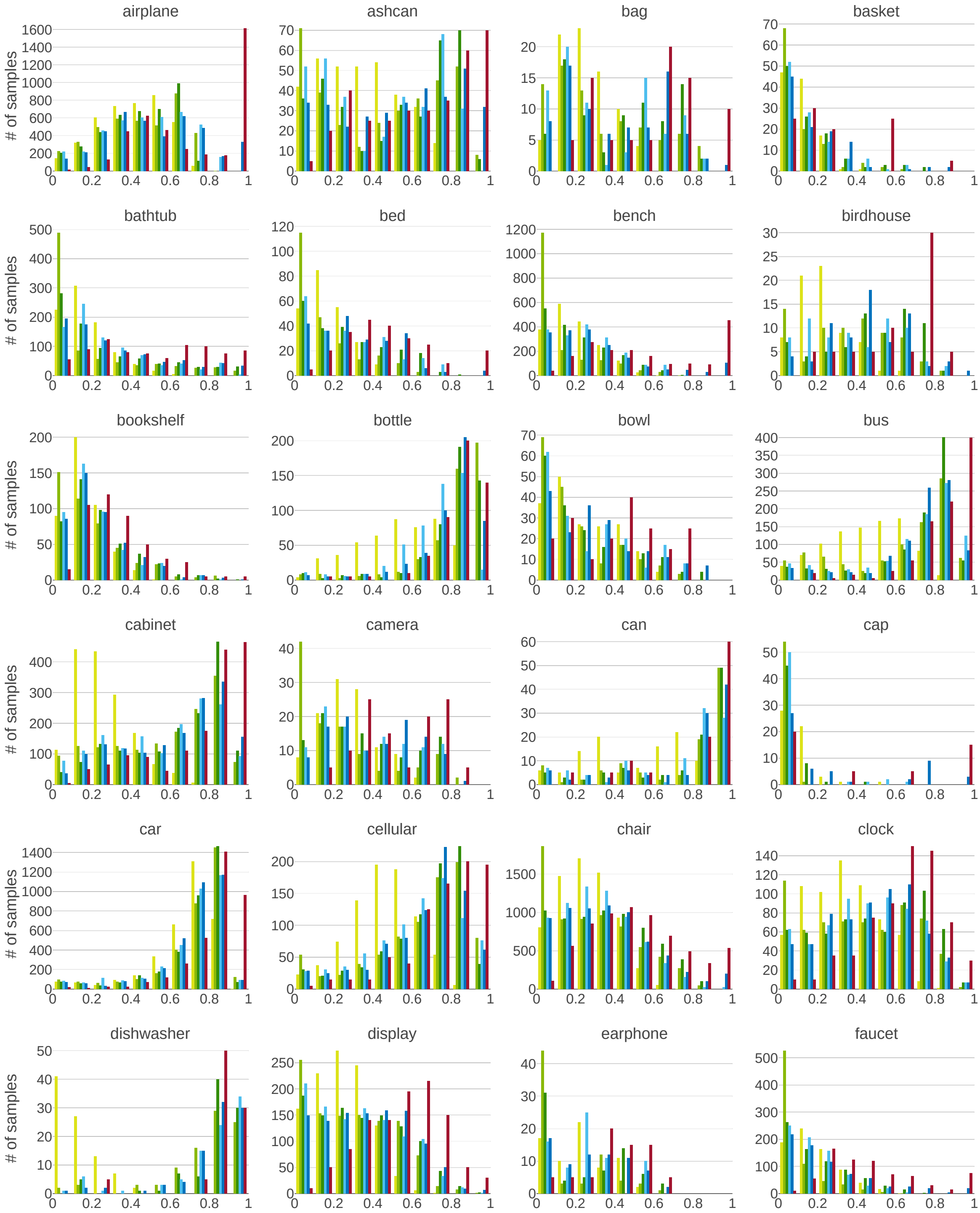


Figure 4.6: Distribution of within-class reconstruction performance for all ShapeNet classes, measured by IoU.

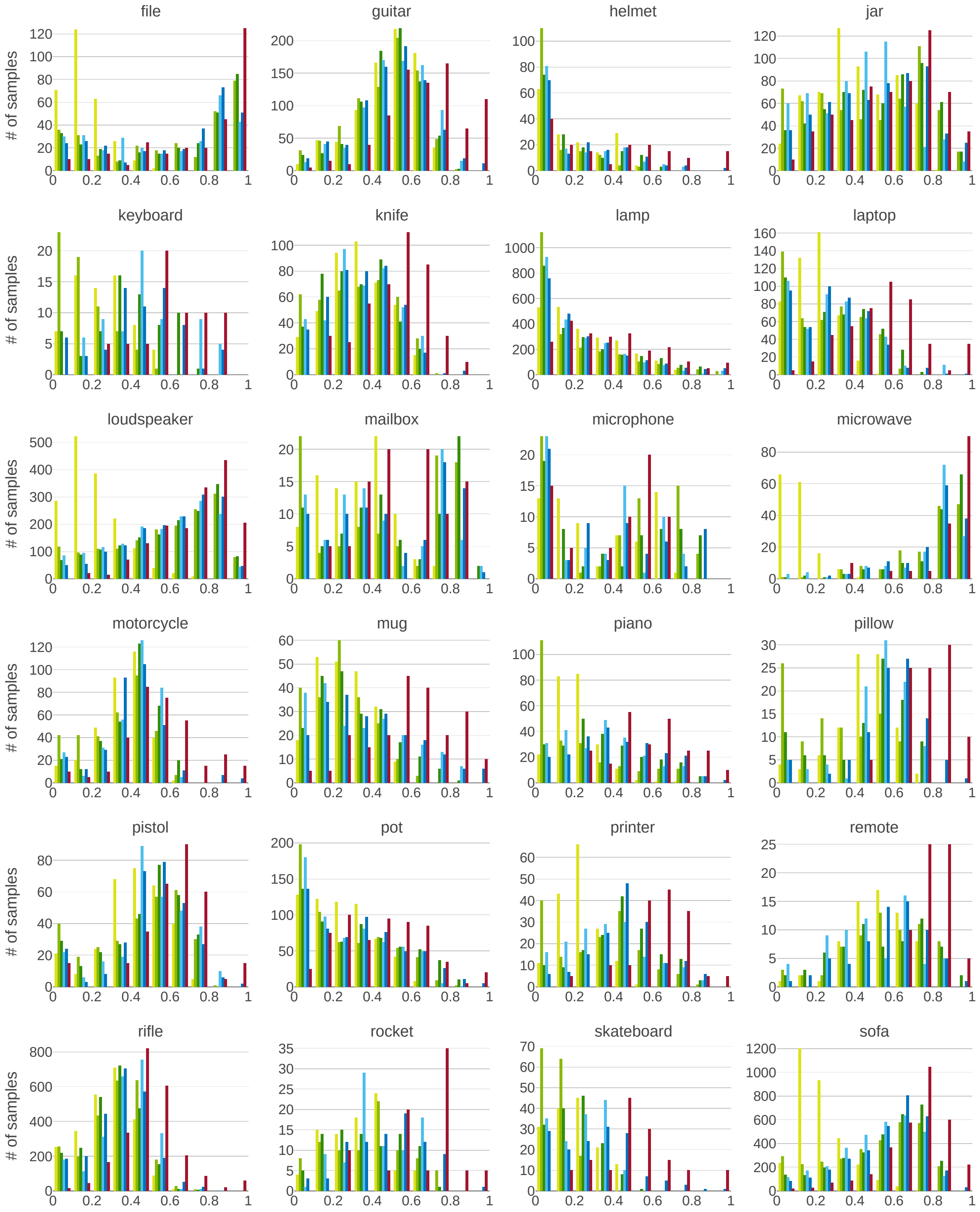


Figure 4.6: Distribution of within-class reconstruction performance for all ShapeNet classes, measured by IoU (continued).

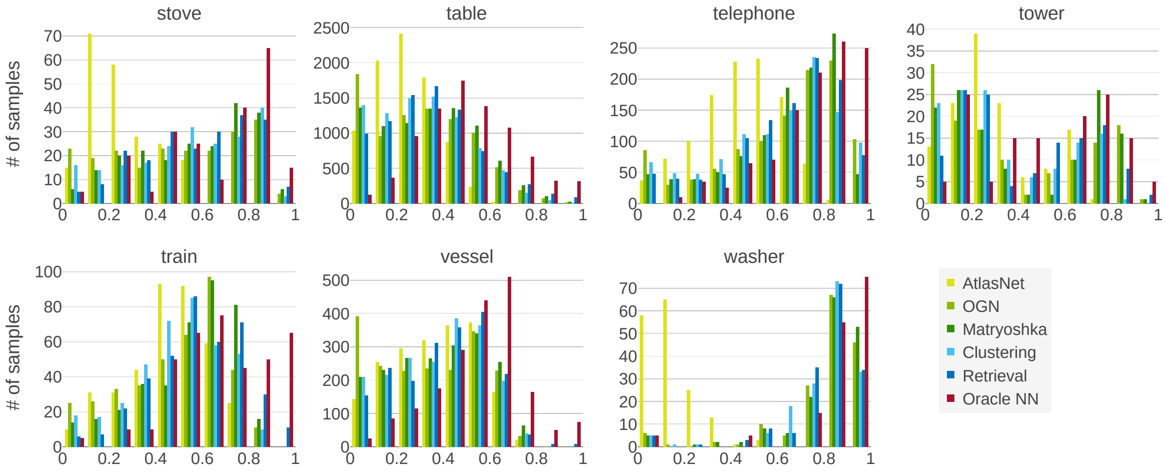


Figure 4.6: Distribution of within-class reconstruction performance for all ShapeNet classes, measured by IoU (continued).

	airplane	ashcan	bag	basket	bathtub	bed	bench	birdhouse	bookshelf	bottle	bowl	bus	cabinet	camera	can	cap	car	cellular	chair
ATL-OGN	0.1	0.15	0.15	0.68	0.0	0.68	0.15	0.1	0.02	0.0	0.36	0.03	0.0	0.95	0.0	0.51	0.95	0.51	0.01
ATL-MN	0.68	0.15	0.36	0.15	0.0	0.02	0.51	0.1	0.02	0.0	0.68	0.03	0.03	0.51	0.03	1.0	1.0	0.68	0.01
ATL-CL	0.06	0.68	1.0	1.0	0.0	0.24	0.51	0.95	0.51	0.01	1.0	0.15	0.0	1.0	0.0	0.95	0.68	0.36	0.1
ATL-RET	0.02	0.15	0.1	0.51	0.0	0.06	0.0	0.51	0.03	0.03	0.95	0.06	0.0	0.84	0.03	1.0	0.68	0.36	0.0
ATL-ORA	0.02	0.06	0.15	0.95	0.0	0.36	0.0	0.15	0.15	0.0	0.51	0.0	0.15	0.03	0.0	0.84	0.68	0.51	0.0
OGN-MN	0.51	0.95	1.0	0.95	0.06	0.36	0.03	1.0	1.0	1.0	0.68	0.03	0.36	0.24	1.0	0.95	0.95	0.84	0.36
OGN-CL	0.84	0.68	0.03	1.0	0.06	0.68	0.02	0.68	0.68	0.36	0.51	0.84	0.95	0.95	0.1	1.0	0.95	0.84	0.03
OGN-RET	0.24	0.51	1.0	0.84	0.0	0.24	0.0	1.0	0.95	0.84	0.95	0.15	1.0	0.51	1.0	0.1	0.95	1.0	0.02
OGN-ORA	0.02	0.1	0.0	0.36	0.0	0.06	0.0	0.0	0.36	0.01	0.06	0.0	0.0	0.0	0.0	0.68	0.1	0.1	0.51
MN-CL	0.68	0.36	0.1	0.68	1.0	0.95	0.95	0.68	0.68	0.51	0.68	0.24	0.51	0.84	0.24	1.0	0.84	0.36	0.03
MN-RET	0.06	0.68	0.68	1.0	0.68	1.0	0.03	1.0	1.0	0.68	1.0	0.95	0.36	1.0	1.0	0.68	0.95	1.0	0.03
MN-ORA	0.03	0.24	0.0	0.06	0.03	0.15	0.03	0.0	0.36	0.03	0.03	0.01	0.15	0.0	0.0	1.0	0.24	0.15	0.15
CL-RET	0.68	0.15	0.02	1.0	0.68	0.84	0.02	0.95	0.84	0.95	0.84	0.68	0.95	0.84	0.24	0.36	0.95	0.68	0.15
CL-ORA	0.01	0.68	0.51	0.84	0.02	0.51	0.03	0.01	0.84	0.06	0.68	0.0	0.0	0.01	0.15	1.0	0.1	0.1	0.24
RET-ORA	0.02	0.02	0.0	0.24	0.36	0.24	0.24	0.0	0.51	0.02	0.06	0.0	0.0	0.0	0.0	0.24	0.1	0.06	0.36

	clock	dishwasher	display	earphone	faucet	file	guitar	helmet	jar	keyboard	knife	lamp	laptop	loudspeaker	mailbox	microphone	microwave	motorcycle	mug
ATL-OGN	0.1	0.51	0.06	0.68	0.24	0.0	0.36	0.51	0.36	1.0	0.84	0.03	0.24	0.0	1.0	1.0	0.1	1.0	0.84
ATL-MN	0.1	1.0	0.06	1.0	0.68	0.0	0.95	1.0	0.36	0.84	0.95	0.36	0.06	0.0	0.84	1.0	0.1	0.68	0.15
ATL-CL	0.36	1.0	0.1	0.51	1.0	0.0	0.68	0.95	0.68	0.95	1.0	0.68	0.1	0.0	1.0	0.03	0.02	1.0	0.51
ATL-RET	0.24	0.95	0.03	0.95	0.06	0.0	0.24	0.95	0.15	0.1	0.84	0.36	0.02	0.0	0.51	1.0	0.1	0.84	0.06
ATL-ORA	0.36	0.24	0.03	0.36	0.03	0.68	0.95	0.36	0.24	0.24	0.84	0.1	0.24	0.15	0.0	0.01	0.84	0.24	0.84
OGN-MN	0.84	0.36	0.95	0.51	0.1	0.95	0.84	0.36	0.51	0.84	0.51	0.36	0.95	0.84	1.0	1.0	0.51	0.95	0.51
OGN-CL	0.68	0.84	0.95	0.95	0.36	0.95	1.0	0.68	0.68	0.84	0.51	0.51	0.84	0.84	0.95	0.24	0.84	0.95	0.95
OGN-RET	1.0	1.0	0.68	0.1	0.02	0.84	0.68	0.36	1.0	0.15	0.84	0.36	0.84	0.36	1.0	1.0	1.0	0.84	0.24
OGN-ORA	0.02	0.0	0.68	0.24	0.0	0.0	0.36	0.0	0.24	0.15	0.68	0.02	0.84	0.01	0.0	0.1	0.0	0.68	0.68
MN-CL	0.36	1.0	1.0	0.15	0.68	1.0	0.95	1.0	0.51	0.15	0.68	0.36	0.95	0.84	0.68	0.06	0.51	0.36	1.0
MN-RET	0.84	0.84	0.84	0.68	0.36	0.95	0.84	1.0	0.84	0.84	0.95	0.51	0.95	0.51	1.0	1.0	0.51	1.0	0.84
MN-ORA	0.0	0.36	0.68	0.1	0.15	0.0	0.84	0.06	0.06	0.01	0.51	0.03	0.95	0.03	0.0	0.02	0.0	0.1	0.02
CL-RET	0.84	1.0	0.95	0.06	0.03	0.95	0.68	0.68	0.68	0.0	0.95	0.36	0.95	0.68	0.36	0.1	1.0	0.68	0.68
CL-ORA	0.1	0.1	0.51	0.68	0.02	0.0	0.84	0.06	0.68	0.95	0.68	0.06	0.68	0.02	0.01	0.36	0.0	0.1	0.1
RET-ORA	0.06	0.02	0.51	0.03	0.68	0.0	0.36	0.1	0.15	0.0	0.84	0.03	0.51	0.24	0.0	0.03	0.0	0.06	0.0

	piano	pillow	pistol	pot	printer	remote	rifle	rocket	skateboard	sofa	stove	table	telephone	tower	train	vessel	washer
ATL-OGN	0.24	0.84	0.15	0.36	0.02	1.0	1.0	0.68	0.1	0.0	0.02	0.02	0.36	0.51	0.68	0.15	0.84
ATL-MN	0.01	0.84	0.51	0.1	0.01	0.51	1.0	1.0	0.95	0.0	0.06	0.01	0.68	1.0	0.84	0.84	1.0
ATL-CL	0.06	0.51	1.0	0.68	0.03	0.68	0.95	0.36	0.95	0.01	0.01	0.02	0.36	0.95	1.0	0.68	1.0
ATL-RET	0.0	1.0	0.68	0.1	0.0	1.0	0.68	0.68	0.15	0.0	0.03	0.0	0.36	0.68	0.51	0.95	1.0
ATL-ORA	0.03	0.0	0.51	0.15	0.84	0.0	0.03	0.0	0.68	0.06	0.84	0.0	0.36	0.03	0.51	0.36	0.24
OGN-MN	0.1	1.0	1.0	0.84	1.0	0.95	1.0	1.0	0.15	0.15	0.84	0.84	0.84	1.0	0.24	0.24	1.0
OGN-CL	0.36	0.24	0.1	0.36	1.0	0.24	0.84	0.01	0.68	0.1	1.0	0.24	0.84	0.51	0.36	0.68	0.51
OGN-RET	0.36	0.68	0.15	0.84	1.0	0.84	0.95	1.0	0.0	0.03	0.95	0.15	1.0	0.68	0.68	0.51	1.0
OGN-ORA	0.06	0.0	0.1	0.51	0.0	0.0	0.15	0.0	0.36	0.0	0.0	0.24	0.03	0.0	0.1	0.03	0.01
MN-CL	0.95	0.24	0.15	0.36	1.0	0.02	0.95	0.1	0.95	0.84	1.0	0.36	0.15	0.51	1.0	0.84	0.95
MN-RET	1.0	0.68	0.51	1.0	1.0	0.68	0.84	0.95	0.24	0.24	0.68	0.24	0.95	0.95	0.15	0.84	1.0
MN-ORA	0.06	0.0	0.15	0.68	0.0	0.0	0.06	0.0	0.68	0.03	0.0	0.68	0.01	0.0	0.03	0.06	0.06
CL-RET	0.95	0.51	0.24	0.36	1.0	0.51	0.51	0.01	0.02	0.68	0.84	0.51	0.36	0.51	0.68	0.84	0.84
CL-ORA	0.36	0.24	0.84	0.51	0.0	0.15	0.24	0.51	0.95	0.1	0.0	0.1	0.0	0.06	0.15	0.1	0.51
RET-ORA	0.03	0.0	0.15	0.68	0.0	0.0	0.68	0.0	0.03	0.15	0.0	0.95	0.0	0.0	0.01	0.36	0.03

Figure 4.7: P-values of the pairwise Kolmogorov-Smirnov test on per-class IoU performance histograms. The null-hypotheses of two distributions being the same can be rejected for $p < 0.05$ (orange) and cannot be rejected for $p > 0.05$ (green).

4.5 problems

In the preceding section we provided evidence that current methods for single-view 3D object reconstruction predominantly rely on recognition. Here we discuss aspects of popular experimental procedures that may need to be reconsidered to elicit more detailed reconstruction behavior from the models.

4.5.1 *Choice of coordinate system*

The vast majority of existing methods predict output shapes in an object-centered coordinate system, which aligns objects of the same semantic category to a common orientation. Aligning objects this way makes it particularly easy to find spatial regularities. It encourages learning-based approaches to recognize the object category first, and refine the shape later if at all.

Shin *et al*. [152] studied how the choice of coordinate frames affects reconstruction performance and generalization abilities of learning-based methods, comparing object-centered and viewer-centered coordinate frames. They found that a viewer-centered frame leads to significantly better generalization to object classes that were not seen during training, a result that can only be achieved when a method operates in a geometric reconstruction regime.

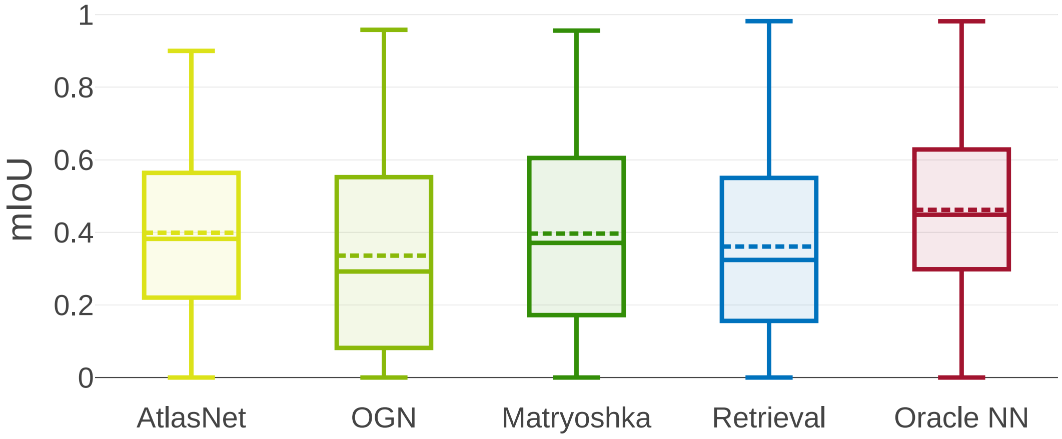


Figure 4.8: Mean IoU in viewer-centered mode. The retrieval baseline does not perform as well in this mode.

To validate these conclusions, we repeated the experimental evaluation (Section 4.4) in a viewer-centered coordinate frame. We attempted to extend the clustering baseline with a viewpoint prediction network which would regress the azimuth and elevation angles of the camera w.r.t. the canonical frame. This naive approach failed because the canonical frame has a different meaning for each object class, implying that the viewpoint network needs to use class information in order to solve the task. For the retrieval baseline, we retrained the method, treating each training view as a separate sample. To avoid artifacts from rotating voxelized shapes, we synthesized ground-truth shapes by rotating and then voxelizing the original meshes, resulting in a distinct target shape for each view of each object. Results are shown in Figure 4.8, where we observe a mild decrease in performance for OGN and Matryoshka networks, and a larger drop for the retrieval baseline. For the retrieval setting, the viewer-centered setup is computationally more demanding, as different views of the same object now refer to different shapes to be retrieved. Consequently, less learning capacity is available for each individual object.

4.5.2 *Evaluation metric*

Intersection over union. The mean IoU is commonly used as the primary quantitative measure for benchmarking single-view reconstruction approaches. The IoU between two shapes $\mathcal{G}$ and $\mathcal{R}$, represented as binary occupancy maps, is commonly defined as

$$\mathrm{IoU}(\mathcal{G}, \mathcal{R}) = \frac{|\mathcal{G} \cap \mathcal{R}|}{|\mathcal{G} \cup \mathcal{R}|}. \tag{4.3}$$

In our evaluation protocol, we compare shapes A, B at a resolution of 128^3 binary cells (voxels). This can be problematic if it is used as the sole metric to argue for the merits of an approach, since it is only indicative of the quality of a predicted shape if it reaches sufficiently high values. Low to mid-range scores indicate a significant discrepancy between two shapes.

An example is shown in Figure 4.9, which compares a car model to different shapes in the dataset and illustrates their similarity in terms of IoU scores. As shown in the figure, even an IoU of 0.59 allows for considerable deviation from the ground-truth shape. For reference, note that 75% of the predictions by the best performing approach, our retrieval baseline, have an IoU below 0.66; 50% are below 0.43 (*c.f.* Figure 4.2).

All information about an object's shape is situated on its surface. However, for voxel-based representations with a solid interior, the IoU is dominated by the interior parts of objects. As a consequence, even seemingly high IoU values may poorly reflect the actual surface similarity.

Moreover, while IoU can easily be evaluated for a volumetric representation, there is no easy way to evaluate it for point clouds. A good measure should allow comparing different 3D representations within the same unified framework. Point-based measures are most suitable for this, because a point cloud can be obtained from any other 3D representation via (a) surface point sampling for meshes, (b) per-pixel reprojection for depth maps, or (c) running the marching cubes algorithm followed by point sampling for voxel grids.

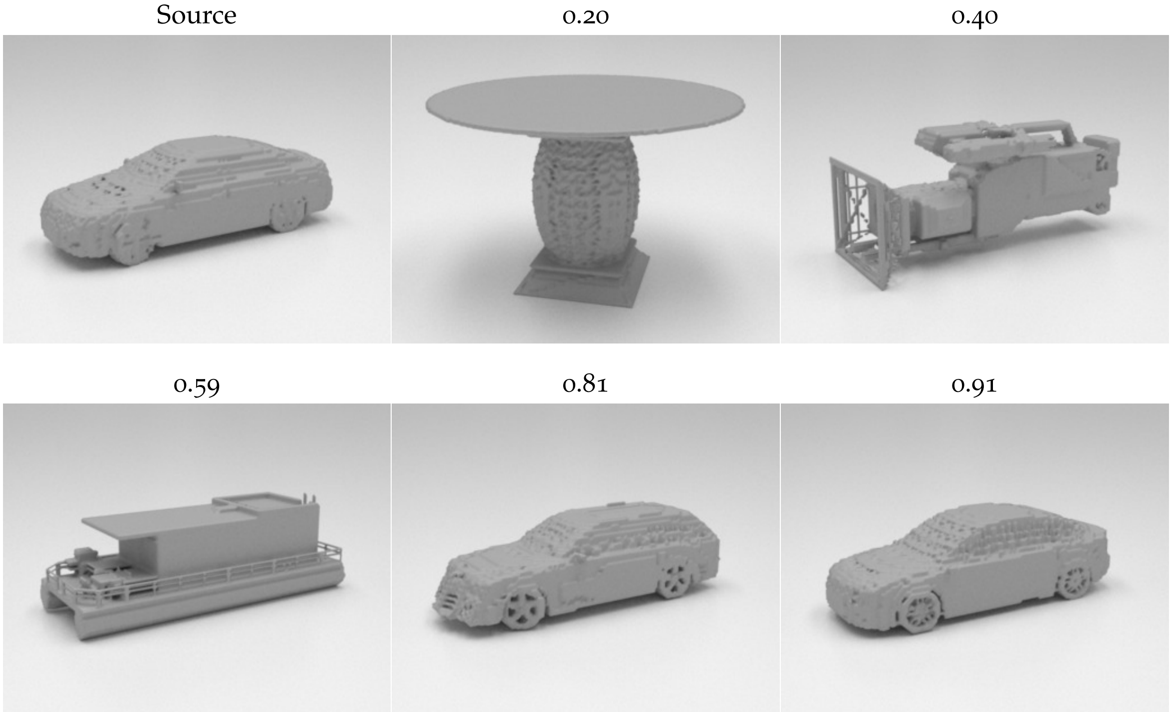


Figure 4.9: IoU between a source shape and various target shapes. Low to mid-range IoU values are a poor indicator of shape similarity.

Chamfer distance. Some recent methods use the Chamfer Distance (CD) for evaluation [33, 48, 163]. CD between the ground truth shape $\mathcal{G}$ and the reconstructed shape $\mathcal{R}$ (both represented as point clouds) is defined as

$$\mathrm{CD}(\mathcal{G},\mathcal{R}) = \frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}}\min_{g\in\mathcal{G}}\|r-g\|_2 + \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}\min_{r\in\mathcal{R}}\|g-r\|_2\,. \tag{4.4}$$

Although it is defined on point clouds and by design satisfies the requirement of being applicable (after conversion) to different 3D representations, it is a problematic measure because of its sensitivity to outliers. Consider the example in Figure 4.10. Both target chairs perfectly match the source chair in the lower part and are completely wrong in the upper part. However, according to the CD score, the second target is much better than the first. As this example shows, the CD measure can be significantly perturbed by the geometric layout of outliers. It is affected by how far the outliers are from the reference shape. We argue that in order to reliably reflect real reconstruction performance, a good quantitative measure should be robust to the detailed geometry of outliers.

F-score. Motivated by the insight that both IoU and CD can be misleading, we propose to use the F-score [81], an established and easily interpretable metric that is actively used in the multi-view 3D reconstruction community. The F-score explicitly evaluates the distance between object surfaces and is defined as the harmonic mean between precision and recall. Here we provide a full definition of the F-score

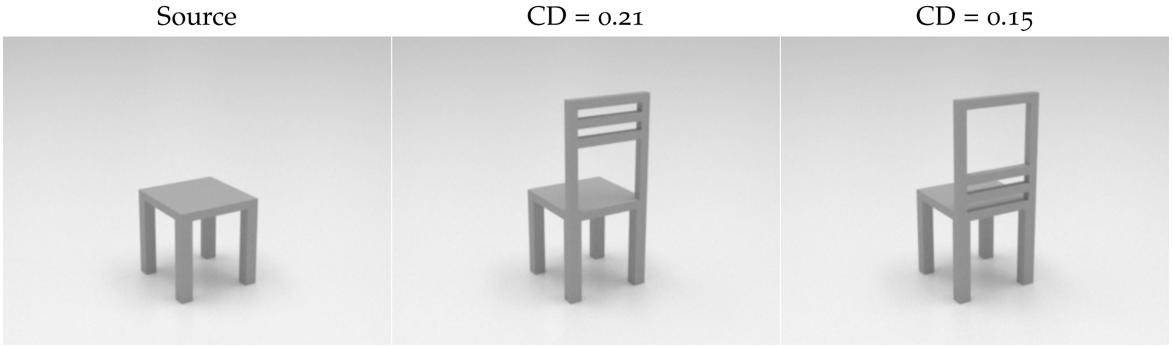


Figure 4.10: The Chamfer distance is sensitive to outliers. Compared to the source, both target shapes exhibit non-matching parts that are equally wrong. While the F@1% is 0.56 for both shapes, the Chamfer distance differs significantly.

measure. Consider a ground truth shape $\mathcal{G}$ and a reconstructed shape $\mathcal{R}$ both represented as point clouds. For every point $r \in \mathcal{R}$ its distance to $\mathcal{G}$ is calculated as

$$e_r = \min_{g \in \mathcal{G}} \|r - g\|_2 .$$

Subsequently, we calculate the percentage of points reconstructed better than a certain threshold d which results in the *precision* value

$$P(d) = \frac{100}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} [e_r < d].$$

The same procedure is repeated in the opposite direction to produce the *recall* value

$$e_g = \min_{r \in \mathcal{R}} \|g - r\|_2 , \qquad R(d) = \frac{100}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} [e_g < d].$$

The final F-score is given by the harmonic mean of the precision and recall values

$$F(d) = \frac{2P(d)R(d)}{P(d) + R(d)}. \tag{4.5}$$

In practice, we set d as a fraction of the side length of the reconstructed volume (*e.g.*, 1%). The metric has an intuitive interpretation: the percentage of points (or surface area) that was reconstructed correctly.

We plot the F-score of viewer-centered reconstructions for different distance thresholds d in Figure 4.11 (left). At $d = 2\%$ of the side length of the reconstructed volume, the absolute F-score values are in the same range as the current mIoU scores, which, as we argued before, is not indicative of the prediction quality. We therefore suggest evaluating the F-score at distance thresholds of 1% and below. In Tab. 4.1 we provide the exact F-score values at 1% threshold in the viewer-centered mode.

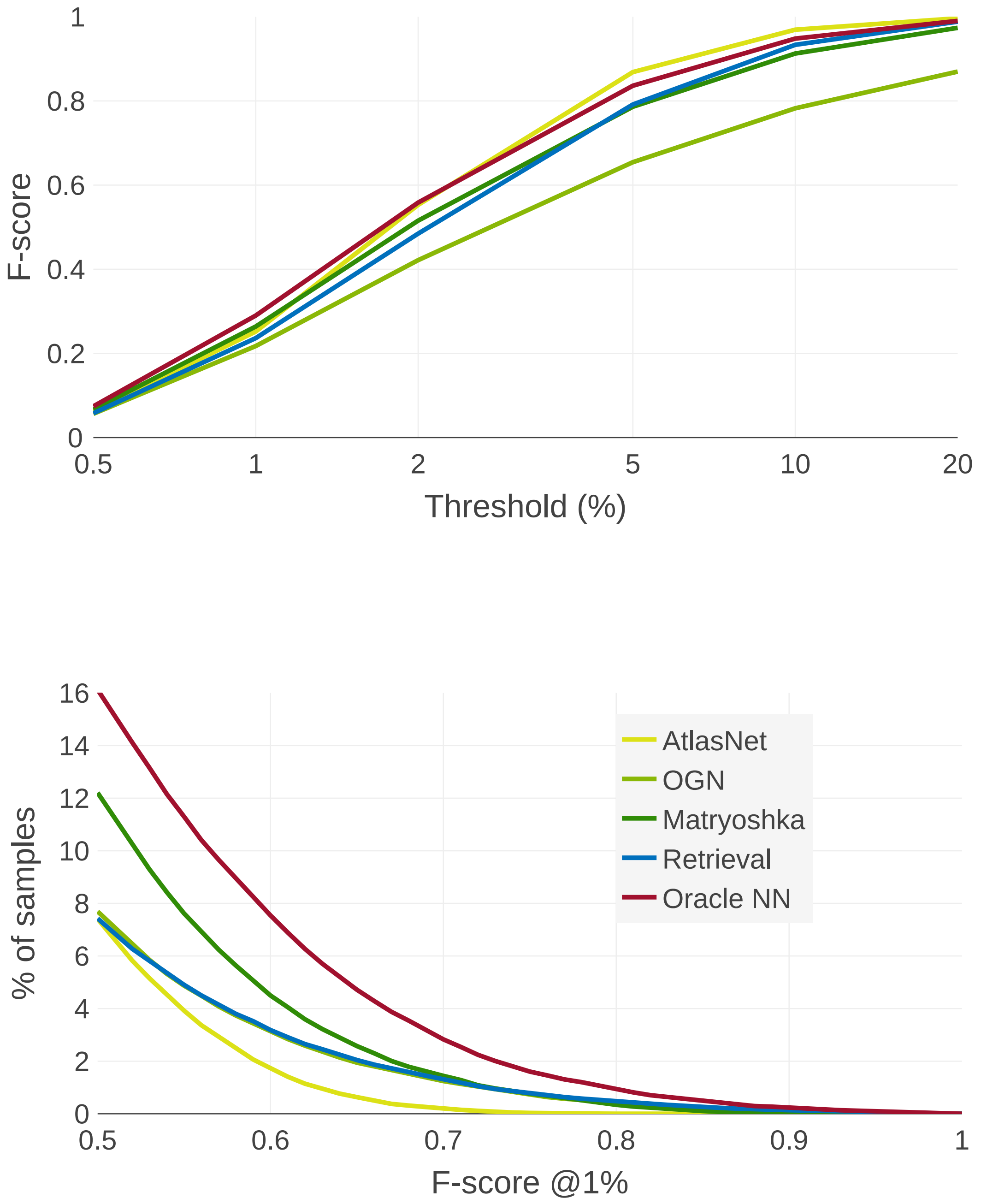


Figure 4.11: F-score statistics in viewer-centered mode. Left: F-score for varying distance thresholds. Right: percentage of reconstructions with F-score above a value specified on the horizontal axis, with a distance threshold $d = 1\%$.

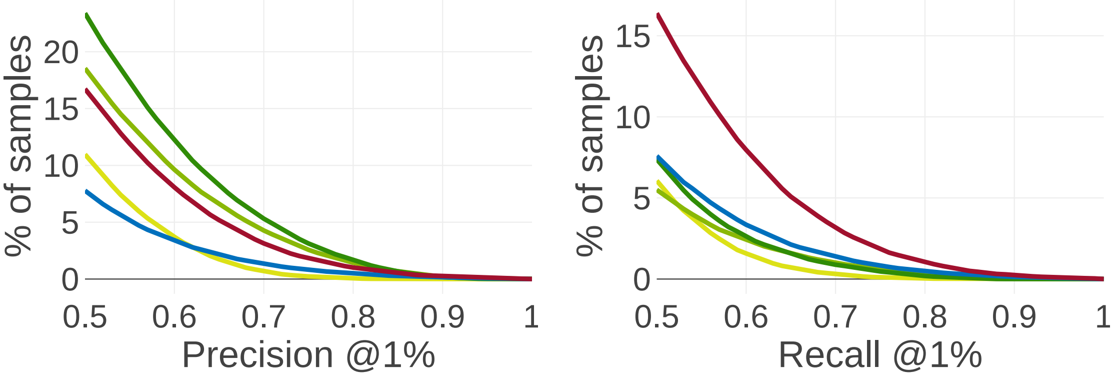


Figure 4.12: Percentage of samples with precision (left) and recall (right) of 0.5 or higher. Existing CNN-based methods show good precision but miss parts of objects, which results in lower recall.

In Figure 4.11 (bottom), we show the percentage of models with an F-score of 0.5 or higher at a threshold $d = 1\%$. Only a small number of shapes is reconstructed accurately, indicating that the task is still far from solved. Our retrieval baseline is no longer a clear winner, further showing that a reasonable solution in viewer-centered mode is harder to get using a pure recognition method.

We observe that AtlasNet often produces qualitatively good surfaces. It even outperforms the Oracle NN baseline on more liberal (above 2%) thresholds, as shown in Figure 4.11 (top). Perceptually, humans tend to judge quality by global and semi-global features and tolerate if parts are slightly wrong in position or shape. We observe that AtlasNet, which was trained to optimize surface correspondence, rarely completely misses parts of the model, but tends to produce poorly localized parts. This is reflected in the high-performance range analysis, shown in Figure 4.11 (bottom), where AtlasNet trails all other approaches.

Analyzing precision and recall separately provides additional insights into each method's behavior. In Figure 4.12 we see that OGN and Matryoshka Networks outperform Oracle NN in terms of precision. However, both Oracle NN and the retrieval baseline show higher recall. This is supported by qualitative observations that OGN and Matryoshka Networks tend to produce incomplete models.

Both recall and precision can be easily visualized to gain further insights, as illustrated in Figure 4.13.

4.5.3 *Dataset*

The problem of networks finding a semantic shortcut solution is closely related to the choice of training data. The ShapeNet dataset has been used extensively because of its size. However, its particular composition – single objects of representative types, aligned to a canonical reference frame – enables recognition models to masquerade as reconstruction. In Figure 4.2, we demonstrate that a retrieval solution (Oracle NN) outperforms all other methods on this dataset, *i.e.*, the test data

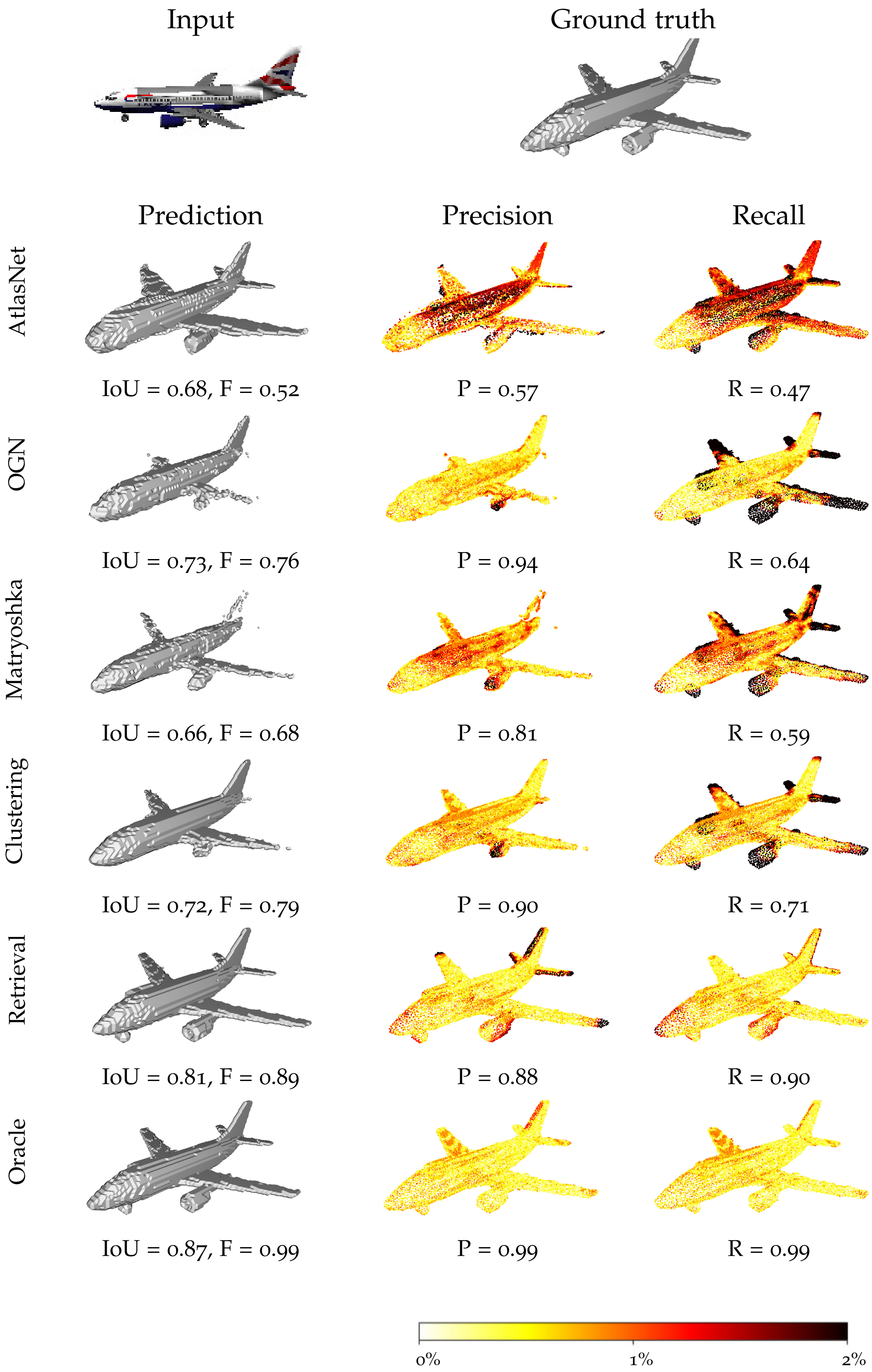


Figure 4.13: Exemplary predictions of different methods compared by visual quality, precision and recall. Colors encode the point-to-surface distance between shapes, normalized by the side length of the reconstructed volume.

	AtlasNet	OGN	Matryoshka	Retrieval	Oracle NN
airplane	0.39	0.26	0.33	0.37	0.45
ashcan	0.18	0.23	0.26	0.21	0.24
bag	0.16	0.14	0.18	0.13	0.15
basket	0.19	0.16	0.21	0.15	0.15
bathtub	0.25	0.13	0.26	0.22	0.26
bed	0.19	0.12	0.18	0.15	0.17
bench	0.34	0.09	0.32	0.3	0.34
birdhouse	0.17	0.13	0.18	0.15	0.15
bookshelf	0.24	0.18	0.25	0.2	0.2
bottle	0.34	0.54	0.45	0.46	0.55
bowl	0.22	0.18	0.24	0.2	0.25
bus	0.35	0.38	0.41	0.36	0.44
cabinet	0.25	0.29	0.33	0.23	0.27
camera	0.13	0.08	0.12	0.11	0.12
can	0.23	0.46	0.44	0.36	0.44
cap	0.18	0.02	0.15	0.19	0.25
car	0.3	0.37	0.38	0.33	0.39
cellular	0.34	0.45	0.47	0.41	0.5
chair	0.25	0.15	0.27	0.2	0.23
clock	0.24	0.21	0.25	0.22	0.27
dishwasher	0.2	0.29	0.31	0.22	0.26
display	0.22	0.15	0.23	0.19	0.24
earphone	0.14	0.07	0.11	0.11	0.13
faucet	0.19	0.06	0.13	0.14	0.2
file	0.22	0.33	0.36	0.24	0.25
guitar	0.45	0.35	0.36	0.41	0.58
helmet	0.1	0.06	0.09	0.08	0.12
jar	0.21	0.22	0.25	0.19	0.22
keyboard	0.36	0.25	0.37	0.35	0.49
knife	0.46	0.26	0.21	0.37	0.54
lamp	0.26	0.13	0.2	0.21	0.27
laptop	0.29	0.21	0.33	0.26	0.33
loudspeaker	0.2	0.26	0.27	0.19	0.23

Table 4.1: F-score evaluation (@1%) in the viewer-centered mode.

	AtlasNet	OGN	Matryoshka	Retrieval	Oracle NN
mailbox	0.21	0.2	0.23	0.2	0.19
microphone	0.23	0.22	0.19	0.18	0.21
microwave	0.23	0.36	0.35	0.22	0.25
motorcycle	0.27	0.12	0.22	0.24	0.28
mug	0.13	0.11	0.15	0.11	0.17
piano	0.17	0.11	0.16	0.14	0.17
pillow	0.19	0.14	0.17	0.18	0.3
pistol	0.29	0.22	0.23	0.25	0.3
pot	0.19	0.15	0.19	0.14	0.16
printer	0.13	0.11	0.13	0.11	0.14
remote	0.3	0.33	0.31	0.31	0.37
rifle	0.43	0.28	0.3	0.36	0.48
rocket	0.34	0.2	0.23	0.26	0.32
skateboard	0.39	0.11	0.39	0.35	0.47
sofa	0.24	0.23	0.27	0.21	0.27
stove	0.2	0.19	0.24	0.18	0.19
table	0.31	0.24	0.34	0.26	0.34
telephone	0.33	0.42	0.45	0.4	0.5
tower	0.24	0.2	0.25	0.25	0.25
train	0.34	0.29	0.3	0.32	0.38
vessel	0.28	0.19	0.22	0.23	0.29
washer	0.2	0.31	0.31	0.21	0.25

Table 4.1: F-score evaluation (@1%) in the viewer-centered mode (continued).

can be explained by simply retrieving models from the training set. This indicates a critical problem in using ShapeNet to evaluate 3D reconstruction: for a typical shape in the test set, there is a very similar shape in the training set. In effect, the train/test split is contaminated, because so many shapes within a class are similar. A reconstruction model evaluated on ShapeNet does not need to actually perform reconstruction: it merely needs to retrieve a similar shape from the training set.

4.6 conclusion

In this chapter, we reasoned about the spectrum of approaches to single-view 3D reconstruction, spanned by reconstruction and recognition. We introduced two baselines, classification and retrieval, which leverage only recognition. We showed that the simple retrieval baseline outperforms recent state-of-the-art methods. Our analysis indicates that state-of-the-art approaches to single-view 3D reconstruction primarily perform recognition rather than reconstruction. We identify aspects of common experimental procedures that elicit this behavior and make a number of recommendations, including the use of a viewer-centered coordinate frame and a robust and informative evaluation measure (the F-score). Another critical problem, the dataset composition, is deeper studied in the next chapter.

5 DO SINGLE-VIEW 3D RECONSTRUCTION NETWORKS GENERALIZE TO REAL DATA?

The text of this chapter was largely copied from the following paper.

> Maxim Tatarchenko, Stephan R. Richter, Jaesik Park, Vladlen Koltun, and Thomas Brox. "Do Single-view 3D Reconstruction Networks Generalize to Real Data?" In preparation for submission.

Stephan R. Richter contributed by providing the initial scene sampling code for dataset rendering. He also provided the code for collecting and converting Lego shapes. Jaesik Park contributed by implementing the voxel grid processing operations in the Open3D framework. All co-authors contributed to the project discussions as well as the final paper text editing.

* * *

5.1 INTRODUCTION

The ultimate benchmark for many tasks in computer vision is the deployment on real-world data. To perform well in this challenging setup, learning-based approaches are commonly trained on large-scale datasets, preferably collected under similar conditions as the evaluation set. In the case of single-view 3D reconstruction, this collection is prohibitively expensive as it requires time-consuming scanning of thousands of diverse 3D objects [163]. To mitigate the lack of real-world training data, it is common to train instead on large-scale synthetic datasets [20, 33, 48, 118, 124, 138]. This introduces the problem of *domain shift*, *i.e.*, a decrease in performance due to the different characteristics of synthetic training data and real testing data. The domain shift poses a considerable challenge to the application of single-view 3D reconstruction methods in the real world: following the commonly employed setup to train on synthetic images and to evaluate on real images, we find that recent approaches hardly generalize at all. To investigate why networks fail to generalize, we decompose the domain shift into two factors: a discrepancy in the 2D appearance and a discrepancy in the shape distribution. The discrepancy in the 2D appearance stems from the different modalities of recording images: rendering an image *vs.* taking a picture with a camera. The discrepancy in the shape distributions stems from collecting CAD models that are different from real-world objects.

For studying the two components of domain shift and their effect on generalization in isolation, we collected a new synthetic dataset, which allows explicit control over the two factors. The dataset consists of three data sources with successively decreasing shape regularity: a practically relevant subset of ShapeNet shapes (Figure

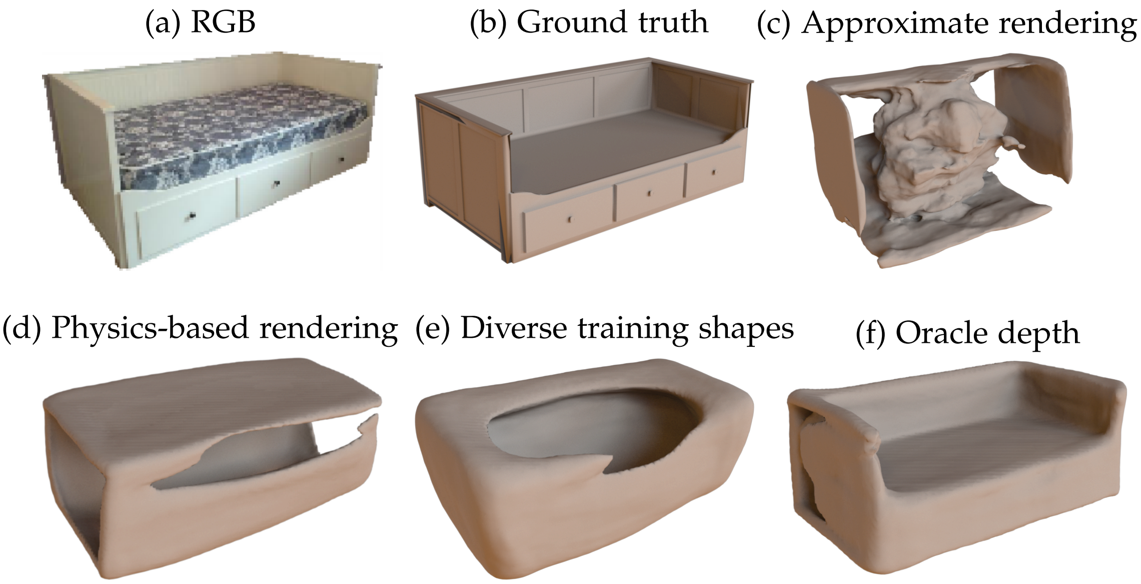

(a) RGB
(b) Ground truth
(c) Approximate rendering
(d) Physics-based rendering
(e) Diverse training shapes
(f) Oracle depth

Figure 5.1: We study how training data affects methods' real-world generalization. The figure shows reconstructions of a Pix3D object (b) from a single image (a) produced with ONet [118] trained under different circumstances: on low-quality approximate ShapeNet renderings (c), on high-quality realistic ShapeNet renderings (d), on realistic renderings of a larger collection of diverse shapes (e) and on oracle depth maps of the same shapes (f).

5.4), high-quality scans of 3D art objects (Figure 5.2), and a collection of diversely arranged Lego blocks (Figure 5.3). We produced realistic renderings of these shapes using a physics-based renderer with careful randomization of different aspects of object appearance.

Equipped with the new dataset, we tested the generalization performance of single-view 3D reconstruction methods aiming to answer the following questions: Does rendering quality affect generalization properties? Is physics-based rendering sufficient to alleviate the appearance domain gap? How important is it to have more diverse shapes in the training set?

Surprisingly, we find that neither high-quality rendering nor its combination with lots of diverse shapes are sufficient for reasonable generalization to real-world data — methods trained under these conditions still strongly overfit to synthetic training samples; see Figure 5.1.

We further investigated the issue by looking into the effect of using different input data modalities. For a number of computer vision tasks, it has been observed that depth maps as input representation can reduce the discrepancy between synthetic and real data [127, 151, 153]. It has also been hypothesized that depth maps as an intermediate representation could be beneficial to the reconstruction of complete 3D shapes [184, 185, 196].

Our analysis of using an oracle depth map as input provides a more systematic view on this hypothesis. It indicates that a decomposition of the single-view 3D reconstruction task into monocular depth estimation and shape completion given a depth map is a promising direction for generalization to real-world scenarios.

5.2 RELATED WORK

Generalization from synthetic visual data. The question of how to make synthetically generated data maximally useful for real-world applications has been addressed in many works. Su *et al.* [160] discuss important rendering techniques for generating synthetic training data. Zhang *et al.* [197] produce physically-based renderings of synthetic indoor scenes, use them for solving multiple computer vision tasks (semantic segmentation, normal estimation, object boundary prediction) and conclude that realistic appearance is important for generalization. Richter *et al.* [139] show that supplementing real-world data with photo-realistic synthetic images enhances the performance of segmentation networks. Hoffman *et al.* [60] use GANs to make synthetic images look more realistic. Tobin *et al.* [166] propose domain randomization for sim2real transfer.

Using synthetic depth maps to better address the domain shift was proposed for multiple tasks. Systems for human pose estimation and hand pose estimation using the Kinect camera are famous success stories [151, 153]. Other examples include semantic segmentation [52], object recognition [127], pose estimation [123] and grasp planning [114].

Datasets. Datasets for object-based single-view 3D reconstruction are scarce. Choi *et al.* [19] scanned real-world objects with commodity sensors. Wiles & Zisserman collected scans from sculptures [181]. Sun *et al.* aligned CAD models to real-world images [163]. Several datasets for robotics tasks were collected in controlled environments [10, 59, 156]. However, since the collection of real-world scans is laborious, the number of images and objects in these real-world datasets is too small for training high-capacity deep networks. Hence, training is usually performed on synthetic datasets, which comprise CAD models from various sources; ShapeNet contains CAD models of everyday objects and furniture [11], Thingi10K [203] and ABC [82] contain models for 3D printing, Blobby Objects features procedurally generated blobs [181].

5.3 DATASET

Reconstructing 3D shapes from a single image is a challenging and ill-posed problem. In order to examine a broad variety of cues and priors and their effect on generalization, we specifically designed a new dataset for analysis. There are two major components of the domain shift: object appearance and the distribution of shapes. The design is motivated by these two factors. The appearance shift is addressed through different modes of rendering while the shape shift is addressed through distinct subsets featuring unique distributions of shapes.

5.3.1 *Data sources*

ShapeNetR2. To cover shapes with highly regular structure, we include a subset of the ShapeNet [11] dataset which is the dominant dataset actively used in the community at the moment. We selected classes that are most likely to appear in everyday natural images and are thus most useful for analyzing the real-world generalization of trained models. These include chairs, tables, cars, motorcycles, sofas, beds, bookshelves, lamps, and airplanes. We refer to this as ShapeNetR2 (ShapeNet Realistically Rendered). Most object instances in this subset are regular in the sense that they have a standard spatial arrangement of parts (e.g. a typical chair has four legs, a seat and a back). We used 29,227 shapes, each rendered from 10 different viewpoints, resulting in a total of 292,270 samples.

Sculptures. Our second subset includes shapes that are less regular and more diverse than ShapeNetR2 samples. It is comprised of 9,129 art objects initially intended for 3D printing, collected from `myminifactory.com`. The collection consists of high-quality 3D scans captured in art galleries around the world and represented as triangular meshes. Most of the objects are organic statues (humans, animals), which implies that they have specific structural priors. However, the immense variety of poses and object combinations make resulting object part arrangements less uniform than in the case of ShapeNetR2. Each shape was rendered from 10 different viewpoints resulting in a total of 91,290 samples.

Lego. This subset was designed to cover highly irregular shapes with no notion of typical object structure. We achieved this by extracting random bundles of blocks out of lego shapes. We used a collection of 83 full Lego Technic shapes. For each shape, we centered a bounding box at a random block and extracted out all blocks fully or partially falling within it. This results in shapes consisting of approximately 5 to 20 blocks (see Figure 5.3). We collected 8,300 such shapes, each rendered from 5 different viewpoints, resulting in a total of 41,500 samples.

5.3.2 *Rendering*

One of the core aspects influencing the appearance domain shift between synthetic and real data is rendering quality. To minimize the discrepancy between rendered images in our dataset and real-world images, we rendered all shapes realistically with a physics-based renderer [67]. Based on the insights about generating synthetic data for training CNNs [160], we randomized most parameters of the rendering procedure. While this does not ensure that synthetic images are indistinguishable from real ones, it alleviates the possibility of CNNs not generalizing to real data because of overfitting to specific unrealistic appearance cues.

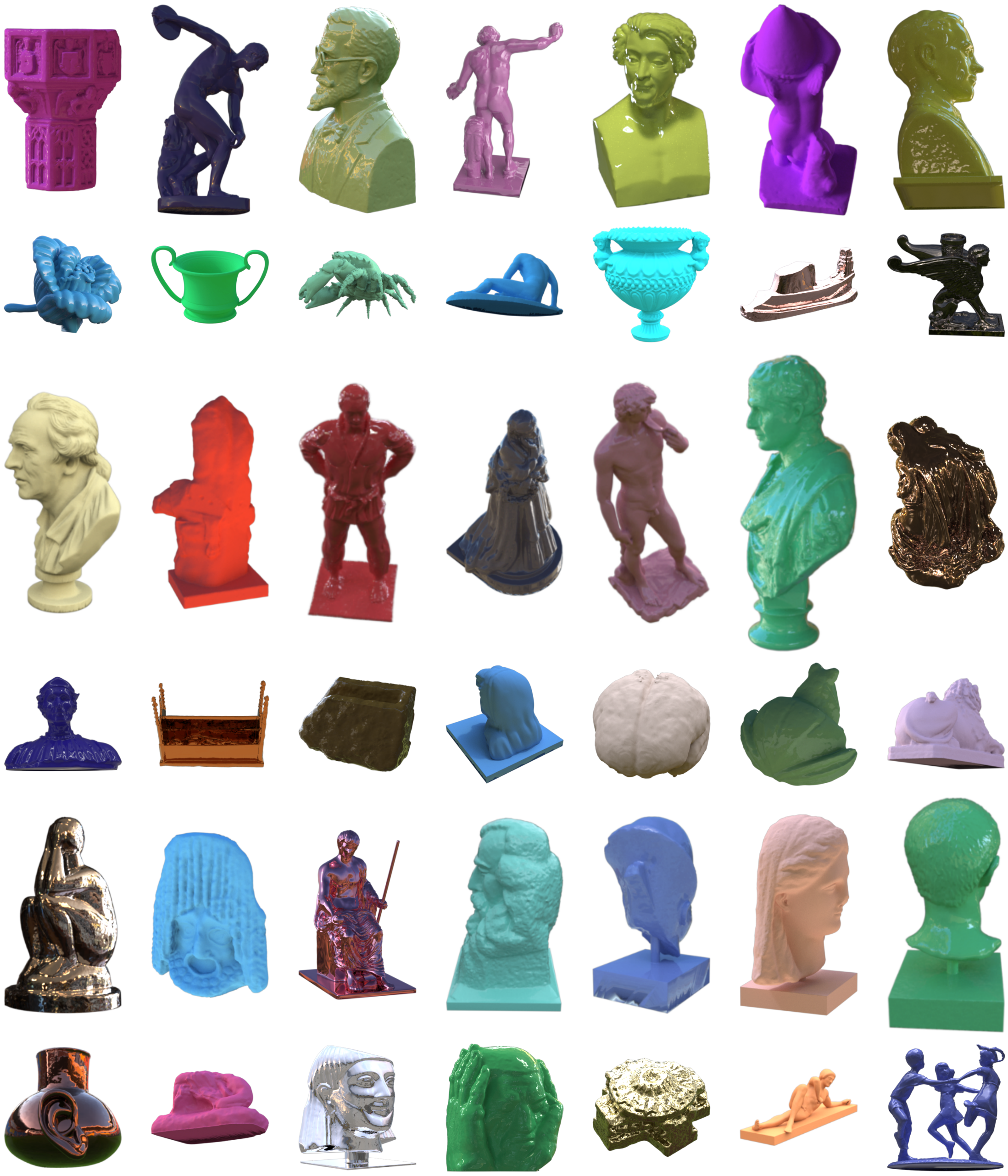


Figure 5.2: Qualitative examples of our high-quality realistic renderings of shapes from Sculptures.

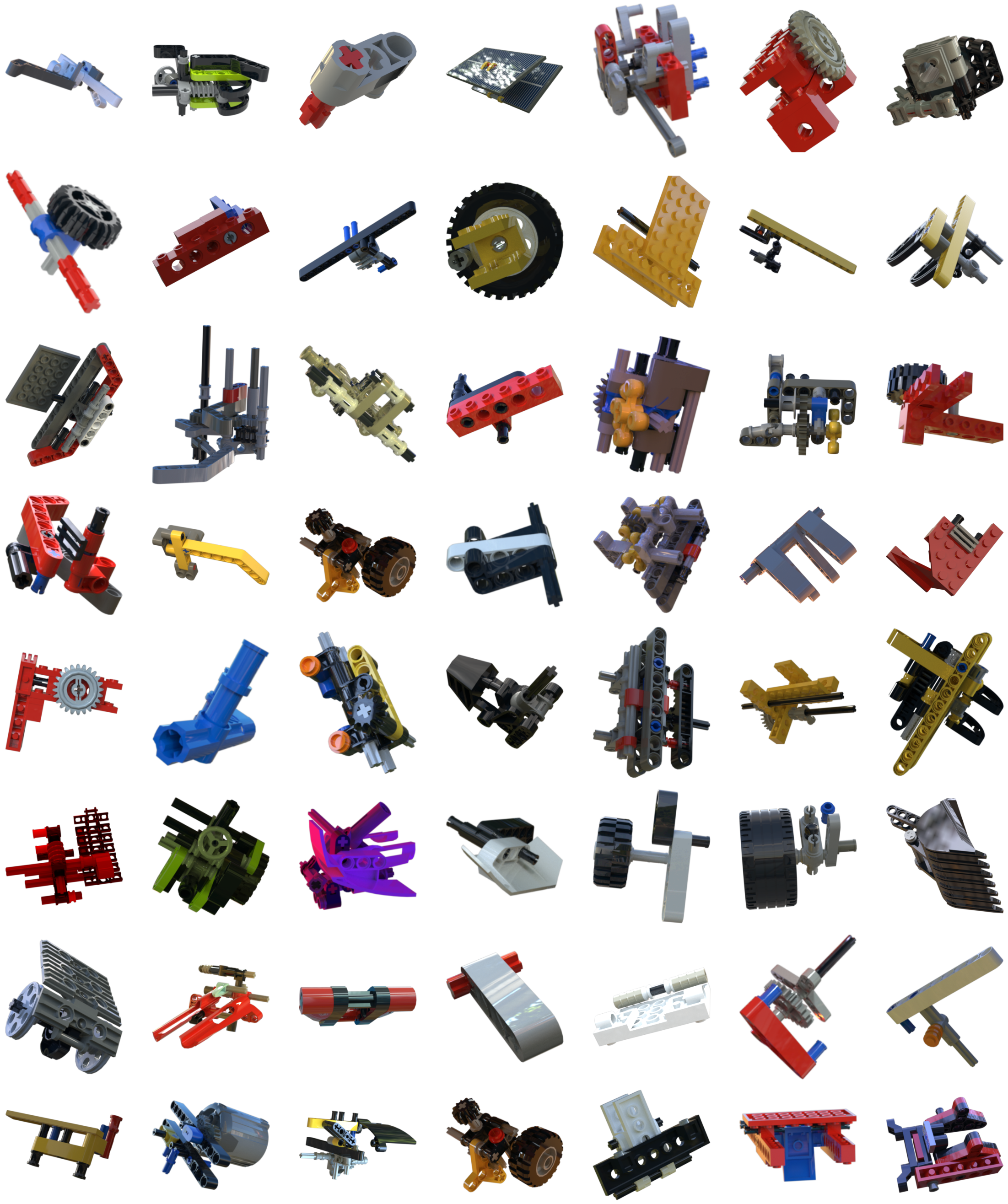


Figure 5.3: Qualitative examples of our high-quality realistic renderings of shapes from Lego.

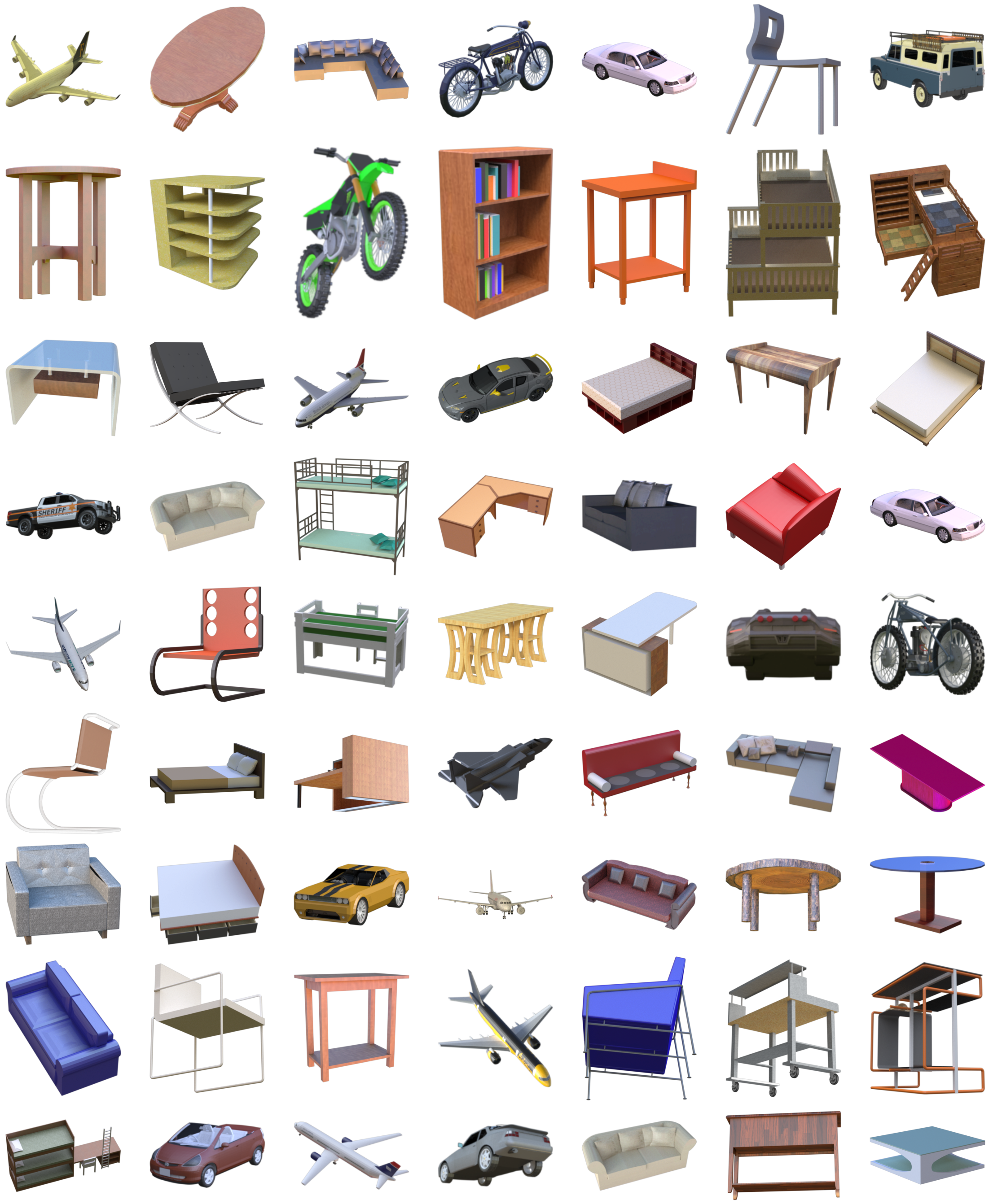


Figure 5.4: Qualitative examples of our high-quality realistic renderings of shapes from ShapeNetR2.

Materials and textures. For Sculptures, we randomly sampled object materials provided in Mitsuba [67]. For Lego, we used a polycarbonate material and the original colors from the shape specifications. For ShapeNetR2, we used the original materials and textures from the dataset. We discuss the importance of textures for generalization in Section 5.4.2.1.

Lighting. To avoid unrealistic light source patterns, we used complex environment maps to specify the lighting conditions[1][2][3]. For every sample, we randomly picked one out of 597 environment maps and randomly rotated it around the vertical axis.

5.3.3 *Evaluation*

5.3.3.1 *Real-world datasets*

In order to test generalization of single-view 3D reconstruction methods to the real world, we evaluate on three real-world datasets: Pix3D [163], YCB [10] and T-LESS [59].

Pix3D comprises 10,068 real-world images of indoor objects from 9 classes accurately aligned with their corresponding 3D CAD models. The classes of shapes in Pix3D mostly match those in the ShapeNetR2 dataset.

YCB is a dataset of household objects captured in a lab setup and initially intended for robotics manipulation. We curated the dataset by manually removing objects with low-quality segmentation masks. Since the viewpoints for every object are sampled densely in the original dataset, many images are similar to each other. Thus, we sub-sampled the set of available views keeping four input image orientations per shape. This resulted in a total of 272 evaluation samples.

T-LESS consists of 30 industry-relevant objects for 6-DoF pose estimation captured in a controlled environment. With the same motivation as in the YCB case, we sub-sampled the original set of available images keeping 6-12 views per object, which results in a total of 348 samples. The objects are effectively textureless, making the setup similar to the Sculptures and Lego training sets. Example images from all of these test sets can be found in Figure 5.5.

5.3.3.2 *Synthetic train/test splits*

To investigate the influence of shape priors, we also define test sets for the Sculptures and Lego subsets of our dataset. To ensure that the training and test sets are not too similar, we clustered all shapes into 100 clusters and performed a random search over the cluster subsets, such that the nearest neighbor distance between the test and training set is maximized. We used the agglomerative clustering algorithm with an F-score-based pairwise similarity matrix. We do not include ShapeNetR2 as a test dataset because the same shape priors are already represented in Pix3D.

[1] http://www.hdrlabs.com/sibl/archive.html

[2] https://hdrihaven.com

[3] https://www.openfootage.net

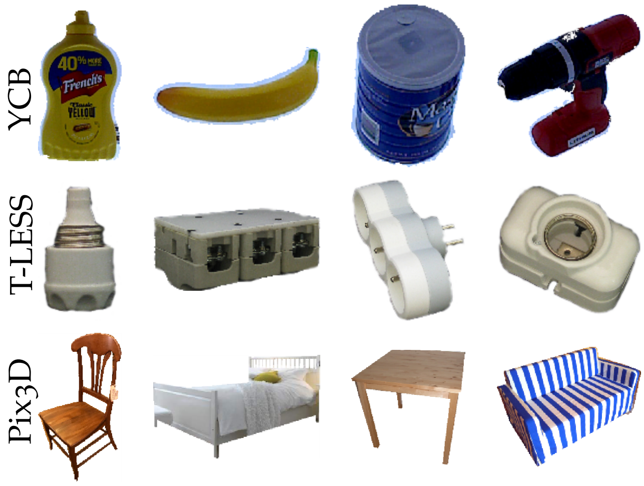


Figure 5.5: Examples from the three real-world test sets we used: YCB (top row), T-LESS (middle row), Pix3D (bottom row).

5.3.3.3 *Metric*

For each predicted shape, we sample 10K points from the surface to generate a point cloud. We then run ICP with scaling [5, 170] to better align predictions with the ground truth and evaluate the F-score with a 1%, same as in Chapter 4. The resulting F-score values range between 0 and 100%.

5.3.4 *Efficient toolbox*

With ∼400K images and ∼50K shapes, processing and evaluation on our dataset becomes challenging, as high-quality assets require much storage, methods require pre- and postprocessing of data and conversions between different 3D representations. Thus, we developed an efficient toolkit based on Open3D [202], which supports seamless on-the-fly conversion between different 3D representations. This simplifies the interaction with the data. For example, a standard procedure for training voxel-based networks includes converting meshes into voxel grids and storing them on the disk. Our on-the-fly conversion works orders of magnitude faster and does not require storing redundant data.

5.4 EXPERIMENTS

5.4.1 *Methods*

For our experiments, we selected two state-of-the-art methods that are representative of different output shape representations commonly used in single-view 3D reconstruction; AtlasNet [48] predicts meshes and Occupancy Networks (ONet) [118] predicts implicit surfaces. We further included a diagnostic method (Oracle Nearest

Neighbor – Oracle NN) that uses privileged information to analyze the properties of the dataset.

AtlasNet. We picked AtlasNet based on the experiments presented in Chapter 4, where it yielded the best results among the compared methods in the viewer-centered mode. It predicts a set of parametric surface patches and arranges those to provide the best coverage of the ground truth shape.

Occupancy Networks. This method represents a modern family of approaches [118, 124] based on implicit functions. The basic idea of the approach is to perform conditional prediction of per-point occupancy given the 3D coordinates and the input RGB image. Sampling many points within the volume of interest and estimating occupancy for each of them allows reconstructing full 3D shapes. The network is a combination of a regular 2D convolutional encoder and a set of fully-connected layers for occupancy prediction.

Oracle NN. This method defines the upper bound of retrieval approaches. Using the privileged information of knowing the ground-truth test shape, it selects the training shape which is closest to the ground truth. It measures how well every test shape can be explained by one of the training shapes. Performing exhaustive search over all views of all shapes is prohibitively expensive. Instead, for every shape represented as a point cloud, we run the PCA decomposition in the point space. We then rotate the shapes to align the resulting principal components with the coordinate axes. Since this manipulation gives the same result for all views of one shape, we do not need to treat every view of the shape as a separate sample in the nearest neighbor search.

5.4.2 *Analysis*

5.4.2.1 *Does realistic appearance matter?*

Realistic rendering. We started by testing the hypothesis that training on high-quality renderings is important for model generalization. Most methods in the literature are trained on renderings used in Choy *et al.* [20]. We reproduced these by generating images using the ShapeNet Viewer[4], a tool for approximate rendering of ShapeNet shapes. We compared the network trained on these low-quality renderings with the model trained on our realistic images.

The results shown in Table 5.1 clearly illustrate that networks trained on low-quality renderings fail to generalize to real-world images, whereas the ones trained on high-quality renderings perform considerably better. This suggests that training on high-quality renderings should be the default mode of operation for single-view 3D reconstruction methods.

Meaningful textures. Another aspect influencing the appearance of objects is the set of textures used. In this experiment, we study how important it is for generalization performance to have semantically meaningful textures. We rendered two

[4] `https://github.com/ShapeNet/shapenet-viewer`

	ShapeNet	ShapeNetR2
Pix3D	4.6	15.5

Table 5.1: Using high-quality renderings for training significantly increases performance on real data (Results using AtlasNet, reported as F-Score@1%).

diagnostic versions of the ShapeNetR2 dataset: one where all textures are removed and the object appearance is instead modulated by sampling random materials (same as for rendering the Sculptures); in the other one we replaced the original semantically meaningful textures with randomly sampled ornaments.

The results are summarized in Table 5.2. Since the textures in ShapeNetR2 to a large extent reflect the real-world object appearance, training with the original ShapeNetR2 textures has a positive influence on Pix3D test performance. This is different for other real-world test sets: ShapeNetR2 textures are useless for YCB and T-LESS, and replacing them with either random textures or random materials does not affect performance in a significant way.

Texture	Pix3D	YCB	T-LESS
Original	**15.5**	4.4	5.7
None	12.9	4.5	5.8
Random	12.9	4.4	5.9

Table 5.2: Training on ShapeNetR2 shapes with semantically meaningful textures increases performance on Pix3D but is irrelevant for other datasets.

	Oracle NN	AtlasNet	ONet
Pix3D	**34.8**	15.5	10.6

Table 5.3: Realistic rendering is not enough to ensure generalization, ConvNets trained on ShapeNetR2 still perform much worse than picking the closest training sample.

5.4.2.2 *Is realistic rendering sufficient?*

While realistic rendering provides substantial improvement over more simplistic approximate rendering, it is still not sufficient to ensure generalization to real data. In Table 5.3 we can see a comparison of all three baselines evaluated on Pix3D when using ShapeNetR2 for training. Even though ShapeNetR2 and Pix3D consist of the same shape types, reconstruction networks do not come even remotely close to the performance of the Oracle NN method. This demonstrates that realistic rendering is not the ultimate solution for the problem of domain shift. In the following section we analyze how the training shape distribution influences the situation.

	Sculptures	Pix3D	Lego	YCB	T-LESS
AtlasNet (Sculptures)	18.8	7.4	4.4	3.8	5.3
ONet (Sculptures)	13.7	4.9	2.8	7.2	6.8
AtlasNet (ShapeNetR2)	12.5	15.5	5.2	4.4	5.7
ONet (ShapeNetR2)	5.0	10.6	1.7	5.9	6.7
AtlasNet (Lego)	8.9	7.1	5.3	4.0	5.2
ONet (Lego)	7.0	5.5	3.8	5.1	5.6
AtlasNet (Combined)	15.1	12.1	5.5	4.5	6.1
ONet (Combined)	13.0	8.3	2.6	6.7	6.9

Table 5.4: Shape prior analysis results. When training end-to-end based on RGB input images, generalization can be mostly attributed to reusing the priors from the training set. Performance is poor when the priors in the test shapes do not match those of the training shapes.

5.4.2.3 *Do diverse shapes help?*

Our dataset is designed to evaluate how well a method generalizes with respect to the difference between the types of shapes in the training and test sets. Pix3D by design contains shapes of the same classes as ShapeNetR2. YCB and T-LESS, on the other hand, do not have a counterpart with suitable shape priors in the training set.

We started off by evaluating all possible combinations of training and testing sets; see Table 5.4. The results clearly show that the best performance is achieved when the priors in the training set match those in the test set. The best performance on Pix3D is obtained with AtlasNet trained on ShapeNetR2. AtlasNet trained and tested on sculptures also demonstrates consistent results. However, when there is effectively no prior to learn – like in the in-distribution case of Lego – the methods' performance is drastically reduced. ONet shows the same trend with lower absolute numbers, as the predictions are often significantly misplaced w.r.t. the ground truth shapes. Both AtlasNet and ONet perform poorly on YCB and T-LESS because there are no suitable shape priors in any of the training sets.

Training set	Oracle NN	AtlasNet	# samples
Full	34.8	15.5	293K
Cars	17.2	6.9	34K
Chairs	27.1	11.3	67K

Table 5.5: Effect of using different ShapeNetR2 subsets for training. Matching shape priors are crucial for achieving reasonable performance on Pix3D.

To further investigate the effect of shape distribution on generalization, we generated diagnostic subsets of ShapeNetR2 and checked how the baselines trained on those modified datasets score on Pix3D. We consider subsets which only contain samples from the chair and the car class respectively. Table 5.5 shows that the Oracle NN method using chairs for NN search – the class with the highest population in Pix3D – retains a large portion of the performance of the full ShapeNetR2 training set. Using cars, on the other hand, decreases the performance by a factor of two. The same trend appears in the AtlasNet results.

Method	Sculptures	Pix3D	Lego	YCB	T-LESS
AtlasNet (ShapeNetR2)	17.6	21.3	8.5	13.6	11.6
Oracle NN (Sculptures)	20.9	21	15.1	18.7	14.1
AtlasNet (Sculptures)	26.2	13.8	6.9	15.2	10.9
AtlasNet (Lego)	16	14.7	9.7	11.4	10.5
AtlasNet (Combined)	24.2	25.4	9.3	17.1	13.0
ONet (Combined)	22.3	21.9	6.7	16.1	12.3

Table 5.6: Results when using depth maps as input. The methods' performance significantly improves over RGB as input. Adding more training data leads to models that generalize better.

Finally, we tested the assumption that generalization can be improved by simply training on a larger dataset with more diverse shapes in it, and found that this is not true. Models trained on a combined version of our dataset (ShapeNetR2 + Lego + Sculptures) and evaluated on Pix3D are inferior to the ones trained only on ShapeNetR2.

All these results confirm that the methods still heavily overfit to the particular priors present in the synthetic training data. In the next section we look deeper into this overfitting by switching the modality of the input images given to the network and analyzing the resulting performance.

5.4.2.4 *Input modality*

Wu *et al.* [184, 185] and Zhang *et al.* [196] proposed a line of single-view 3D reconstruction methods based on estimating intermediate geometric representations. The core idea is to split the single-view 3D reconstruction task into two modules. The first estimates a depth map (and potentially other modalities like surface normals and silhouettes) from the input image. The second performs 3D shape completion based on the outputs of the first module. The underlying hypothesis is that the domain shift between synthetic and real depth maps is less severe than the one for RGB images, hence the shape completion module may generalize better.

We could not directly use the full framework of Zhang *et al.* [196] due to their assumption of having a fixed camera-to-object distance and fixed internal camera parameters. Instead, we set up a controlled experiment for evaluating the upper bound on the generalization of depth-based shape completion. For all our training and testing datasets, we rendered orthographic depth maps from uniformly scaled objects. We used these oracle depth maps both for training and testing AtlasNet and ONet. In the case of AtlasNet, we started from the same pre-trained autoencoder as in the corresponding RGB experiment, thus ensuring that only the input modality and the weights of the encoder change. This allows to perfectly distill the influence of the input representation, which was not possible in the experiments of Zhang *et al.* [196].

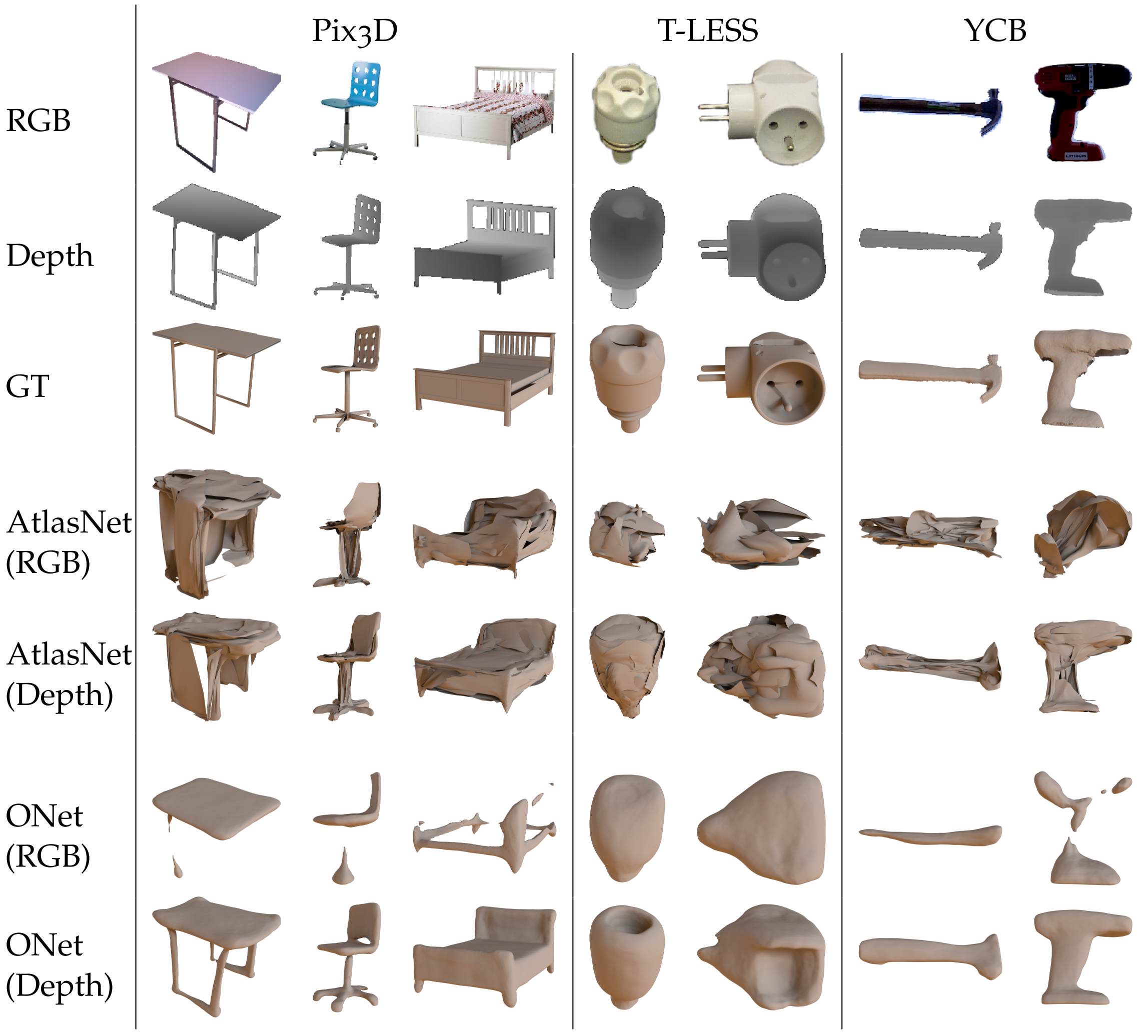


Figure 5.6: Qualitative results on real-world test sets when training on the combined version of our dataset (ShapeNetR2 + Lego + Sculptures). Depth-based predictions are significantly better than the RGB-based counterparts both for AtlasNet and ONet. When using depth as input, the methods generalize much better to shapes which are completely different from the ones seen during training.

The results are summarized in Table 5.6. The performance on all test sets has significantly improved compared to the RGB experiments. For the first time, the in-distribution test, where the network was both trained and tested on Sculptures, shows better results than the Oracle NN baseline. Results on Pix3D when training on depth maps from Lego and Sculptures, which have entirely different shape priors, are nearly the same as when training on RGB images from ShapeNetR2, which has very similar shape priors.

Networks starting from input depth maps can also better exploit large amounts of diverse data. Unlike in the RGB experiments, training on a combination of different data sources improves the results. Figure 5.6 shows the results produced by networks trained on the combined version of our dataset (ShapeNetR2 + Lego + Sculptures). Depth-based predictions are substantially better than their RGB-based counterparts.

Note that the decoder weights stayed fixed in all of the AtlasNet experiments. This indicates that the decoder is capable of producing a more comprehensive range of shapes than those it has seen during training. Another experiment further confirms this point: we keep a fixed decoder trained on Lego and Sculptures respectively and train the encoder on ShapeNetR2, see Table 5.7. Despite the fixed decoder, both networks trained on ShapeNetR2 perform better on Pix3D than the counterparts trained on Lego and Sculptures.

Decoder	Matching encoder	ShapeNetR2 encoder
ShapeNetR2	15.5	15.5
Lego	7.1	12.6
Sculptures	7.4	11.2

Table 5.7: AtlasNet with decoder weights fixed and the encoder weights trained on different subsets of our dataset. Training the encoder on a set of images which is more similar to the test set always increases performance, even if the decoder was trained to produce a different set of shapes.

This observation also explains the seeming discrepancy between the fact that single-view 3D reconstruction networks mostly perform recognition formulated in Chapter 4 and that linear interpolation between the latent codes of different shapes results in the smooth morphing of shapes in the output space [183]. It is the encoder that only maps to a small subset of the hidden space. When a more substantial subset is explored by interpolating between the latent codes, a more extensive set of 3D shapes can be produced.

The conclusion from the above experiment is similar to the one from Zhou *et al.* [201]: explicit intermediate representations help solve the final task, in our case 3D reconstruction.

5.5 conclusion

The analysis presented in this chapter explicitly demonstrates that the problem of synthetic-to-real generalization in the case of single-view 3D reconstruction cannot be trivially solved by carefully designing the training dataset. Highly diverse shapes and realistic rendering are necessary, but not sufficient for developing models that work reliably on real images. What turns out to be helpful is introducing more structure into the model design and using depth as an intermediate representation in the pipeline. This is not a particularly novel observation – the positive influence of making depth-based predictions has been actively exploited in other fields like segmentation [52] or robotics [114]. However, in modern single-view 3D reconstruction, this design choice was only made in one line of work [184, 185, 196], and so far, it was not apparent that this should be the default mode of operation.

Our results show that single-view 3D reconstruction networks generalize much better when provided with oracle depth maps. This bridges the gap to existing monocular depth estimation works and suggests that single-view 3D reconstruction would benefit from improved single-view depth estimation.

6 CONCLUSION

In the first part of the thesis, we presented deep learning methods for analyzing and synthesizing 3D geometry. Our main focus was on the scalability of approaches, i.e. their ability to efficiently work with high-resolution data. For geometry analysis we proposed tangent convolution - a novel operation for convolving point clouds. Convolutional networks based on tangent convolutions are suitable for processing large-scale indoor and outdoor scenes with millions of points. For synthesis, we presented Octree Generating Networks, which enable the reconstruction of 3D shapes in high resolution given arbitrary inputs. Together, these two approaches form a solid toolkit for solving the most common tasks in 3D deep learning.

In the second part of the thesis, we focused on the problem of single-view 3D reconstruction. We performed a detailed analysis of a group of state-of-the-art methods comparing them with two approaches explicitly solving the task in the recognition mode. The fact that the results of these two groups of approaches were statistically indistinguishable led us to the conclusion that current methods for single-view 3D reconstruction largely solve the task through recognition. It is important to note that this observation does not undermine the technical contributions of existing methods. It rather reveals some poorly planned design choices in the conventional experimental setup which led to such behavior. Fixing those is a necessary but not sufficient step on the way to developing methods that perform fine-grained single-view 3D reconstruction based on local image cues. Furthermore, a method approaching 3D prediction as a recognition task is not a bad thing per se - it may actually be desirable in many situations. However, the decision to work with such a method should be made consciously and not come as a byproduct of a misconception.

Finally, we analyzed modern single-view 3D reconstruction approaches in terms of their generalization to real-world images. For this, we gathered a large collection of diverse shapes and produced their realistic renderings. We concluded that, at the moment, neither increasing the amount and diversity of synthetic training data nor making its appearance more realistic is sufficient to ensure generalization of current approaches to real data. In other words, data is not the ultimate solution, and more work has to go into the algorithm design. This conclusion is, of course, limited by the quality of our generated data. Even though we put considerable effort into carefully designing the dataset, renderings from it can still be distinguished from real-world images. There is a chance that training on synthetic data of even better quality will be sufficient to generally solve the task. Nevertheless, our findings indicate that at the moment one important design choice is to have intermediate geometric representations in the processing pipeline. Though not providing any formal information gain, such pipeline structure can simplify the task.

For example, in case of single-view 3D reconstruction, the first part of the pipeline (the one estimating depth) still operates in the image space. It thus can rely on the input image structure when predicting partial 3D information. The task of the second part is only to complement the prediction of the first part, this can be much easier than a completely unguided RGB-to-3D transition. A similar observation was formulated in Zhang *et al*. [196]. Note that this is an indication, not a proof: we can only hypothesize, why such a method design makes the task easier. An explicit evaluation of modern mono-depth estimation methods as part of a single-view 3D reconstruction pipeline is required in order to draw stronger conclusions.

FUTURE WORK

Despite the recent rapid progress in 3D deep learning, many challenges still remain open. We provide an outline of what we believe to be the most important directions of future research.

Standardized implementations. Many methods for 3D deep learning developed in the recent years are shipped as independent incompatible frameworks. While sufficient for initial proof-of-concept experiments, this strategy is barely suitable for using these methods as parts of larger computational pipelines. Standardized and highly optimized implementations [1, 68, 126] proved to be critically important for scaling up 2D deep learning algorithms. Similar 3D frameworks [135, 173] are starting to emerge. However, they still lack universal support of important data structures and algorithms; therefore, more engineering effort is required on this side.

Differentiable rendering for single-view 3D reconstruction. The best-performing deep learning methods for 3D synthesis rely on full 3D supervision during training. This requires collecting large amounts of 3D data, either synthetic or real. While working with synthetic data, one needs to address the domain shift when applying the trained models to real-world data - an issue thoroughly discussed in Chapter 5. Training directly on real data, on the other hand, requires expensive and complicated data collection. An alternative solution is to use a weaker form of supervision during training, namely 2D projections. Some first steps have been made in this direction [66, 76, 137, 169, 191, 204], but the current methods are still mostly using silhouette images which provide a limited amount of information. Existing differentiable rendering algorithms [56, 93, 103] have not been widely adopted for 3D prediction tasks with natural images used for supervision during training. We believe that such design choice may be important to develop data-efficient single-view 3D reconstruction algorithms which generalize to real-world images.

Intermediate geometric representations. In Chapter 5 we have shown that single-view 3D reconstruction algorithms enjoy much better generalization when applied to depth inputs instead of RGB inputs. This indicates the potential importance of making monocular depth estimation an explicit part of the computation pipeline, thus bridging the gap to existing works in this field [32, 89]. At the same time, many questions remain unanswered. How accurate does a depth estimate need to be in order to be useful for shape completion? Do separate modules (e.g. mono-depth estimation and shape completion) need to be trained together? Is the problem of estimating depth from a single image practically easier than that of estimating a full 3D shape from an RGB image directly? All these questions span exciting new research possibilities.

Single-view 3D reconstruction beyond individual objects. Current single-view 3D reconstruction methods almost exclusively solve the task for individual segmented objects. A variant of the task which involves explaining full scenes observed from a single viewpoint is no less attractive, although more challenging, both in terms of method design and available data.

Compositional reasoning. Most objects can be naturally decomposed into meaningful parts. Combinatorially, the number of unique parts is significantly smaller than the number of combinations. Therefore, compositional reasoning about the object structure, that is, identifying the constituting parts and arranging them the right way, could be an attractive approach to the problem of 3D reconstruction. Initial steps in this direction, when using existing object parts to reconstruct a new object, were made in several recent works [39, 187]. However, they still require a database of annotated object parts in order to be trained. A more interesting variant is the one which can be trained without ground truth part annotations, and even, preferably, without the assumption that individual parts have fixed semantic meaning. The Lego dataset introduced in Chapter 5 could provide suitable data for developing this sort of algorithms.

Real applications of single-view 3D reconstruction. Another underexplored aspect of single-view 3D reconstruction is its use in practical tasks. There are two modes for this: explicit and implicit. In the explicit mode, generated shapes can be used for 3D reasoning, e.g. for solving robotics tasks. In the implicit mode, learned compact representations containing 3D information can be combined with other useful features to solve the tasks at hand. We believe, there are many promising applications for both of these modes.

BIBLIOGRAPHY

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, et al. "TensorFlow: A system for large-scale machine learning." In: *OSDI*. 2016.

[2] Abhishek Anand, Hema Swetha Koppula, Thorsten Joachims, and Ashutosh Saxena. "Contextually guided semantic labeling and search for three dimensional point clouds." In: *IJRR* 32.1 (2013).

[3] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces." In: *CVPR*. 2016.

[4] Matan Atzmon, Haggai Maron, and Yaron Lipman. "Point convolutional neural networks by extension operators." In: *ACM Trans. Graph.* 37.4 (2018), 71:1–71:12.

[5] Paul J. Besl and Neil D. McKay. "A Method for Registration of 3-D Shapes." In: *TPAMI* 14.2 (1992), pp. 239–256.

[6] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. "FAUST: Dataset and evaluation for 3D mesh registration." In: *CVPR*. 2014.

[7] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael M. Bronstein. "Learning Shape Correspondence with Anisotropic Convolutional Neural Networks." In: *NIPS*. 2016.

[8] Alexandre Boulch, Bertrand Le Saux, and Nicolas Audebert. "Unstructured Point Cloud Semantic Labeling Using Deep Segmentation Networks." In: *Eurographics Workshop on 3D Object Retrieval*. 2017.

[9] Fatih Calakli and Gabriel Taubin. "SSD: Smooth Signed Distance Surface Reconstruction." In: *Computer Graphics Forum* (2011).

[10] Berk Çalli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. "Yale-CMU-Berkeley dataset for robotic manipulation research." In: *IJRR* 36.3 (2017), pp. 261–268.

[11] Angel X. Chang et al. "ShapeNet: An Information-Rich 3D Model Repository." In: *arXiv:1512.03012* (2015).

[12] Amin P. Charaniya, Roberto Manduchi, and Suresh K. Lodha. "Supervised Parametric Classification of Aerial LiDAR Data." In: *CVPR Workshops*. 2004.

[13] Nesrine Chehata, Li Guo, and Clément Mallet. "Contribution of airborne full-waveform lidar and image data for urban scene classification." In: *ICIP*. 2009.

[14] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. "On Visual Similarity Based 3D Model Retrieval." In: *Comput. Graph. Forum* 22.3 (2003), pp. 223–232.

[15] Hao Chen, Qi Dou, Lequan Yu, Jing Qin, and Pheng-Ann Heng. "VoxResNet: Deep voxelwise residual networks for brain segmentation from 3D MR images." In: *NeuroImage* 170 (2018), pp. 446–455.

[16] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. "Single-Image Depth Perception in the Wild." In: *NIPS*. 2016.

[17] Zhiqin Chen and Hao Zhang. "Learning Implicit Fields for Generative Shape Modeling." In: *CVPR*. 2019.

[18] HungYueh Chiang, Yen-Liang Lin, Yueh-Cheng Liu, and Winston H. Hsu. "A Unified Point-Based Framework for 3D Segmentation." In: *3DV*. 2019.

[19] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. "A Large Dataset of Object Scans." In: *arXiv:1602.02481* (2016).

[20] Christopher Bongsoo Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction." In: *ECCV*. 2016.

[21] Christopher Choy, Jun Y. Gwak, and Silvio Savarese. "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks." In: *CVPR*. 2019.

[22] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." In: *MICCAI*. 2016.

[23] C. Ian Connolly. "Cumulative generation of octree models from range data." In: *ICRA*. 1984.

[24] Angela Dai and Matthias Nießner. "3DMV: Joint 3D-Multi-view Prediction for 3D Semantic Scene Segmentation." In: *ECCV*. 2018.

[25] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes." In: *CVPR*. 2017.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. "ImageNet: A large-scale hierarchical image database." In: *CVPR*. 2009.

[27] Xinhan Di, Rozenn Dahyot, and Mukta Prasad. "Deep Shape from a Low Number of Silhouettes." In: *ECCV Workshops*. 2016.

[28] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. "Learning to generate chairs with convolutional neural networks." In: *CVPR*. 2015.

[29] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. "FlowNet: Learning Optical Flow with Convolutional Networks." In: *ICCV*. 2015.

[30] Jean-Denis Durou, Maurizio Falcone, and Manuela Sagona. "Numerical methods for shape-from-shading: A new survey with benchmarks." In: *CVIU* 109.1 (2008), pp. 22–43.

[31] David Eigen and Rob Fergus. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture." In: *ICCV*. 2015.

[32] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network." In: *NIPS*. 2014.

[33] Haoqiang Fan, Hao Su, and Leonidas J. Guibas. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." In: *CVPR*. 2017.

[34] Paolo Favaro and Stefano Soatto. "A geometric approach to shape from defocus." In: *TPAMI* 27.3 (2005), pp. 406–417.

[35] Georgios Floros and Bastian Leibe. "Joint 2D-3D temporally consistent semantic segmentation of street scenes." In: *CVPR*. 2012.

[36] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. "Recognizing objects in range data using regional point descriptors." In: *ECCV*. 2004.

[37] Simon Fuhrmann and Michael Goesele. "Fusion of Depth Maps with Multiple Scales." In: *SIGGRAPH Asia*. 2011.

[38] Matheus Gadelha, Subhransu Maji, and Rui Wang. "3D Shape Induction from 2D Views of Multiple Objects." In: *arXiv:1612.05872* (2016).

[39] Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. "SDM-NET: deep generative network for structured deformable mesh." In: *ACM Trans. Graph.* 38.6 (2019), 243:1–243:15.

[40] Irene Gargantini. "Linear octtrees for fast processing of three-dimensional objects." In: *Computer Graphics and Image Processing* 20.4 (1982), pp. 365 –374.

[41] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas A. Funkhouser. "Learning Shape Templates with Structured Implicit Functions." In: *ICCV*. 2019.

[42] Rohit Girdhar, David Fouhey, Mikel Rodriguez, and Abhinav Gupta. "Learning a Predictable and Generative Vector Representation for Objects." In: *ECCV*. 2016.

[43] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. "Unsupervised Monocular Depth Estimation with Left-Right Consistency." In: *CVPR*. 2017.

[44] Aleksey Golovinskiy, Vladimir G. Kim, and Thomas A. Funkhouser. "Shape-based recognition of 3D point clouds in urban environments." In: *ICCV*. 2009.

[45] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. "3D Semantic Segmentation with Submanifold Sparse Convolutional Networks." In: *CVPR* (2018).

[46] Edward Grant, Pushmeet Kohli, and Marcel van Gerven. "Deep disentangled representations for volumetric reconstruction." In: *ECCV Workshops*. 2016.

[47] Fabian Groh, Patrick Wieschollek, and Hendrik Lensch. "Flex-Convolution - Million-Scale Point-Cloud Learning Beyond Grid-Worlds." In: *ACCV*. 2018.

[48] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation." In: *CVPR*. 2018.

[49] Saurabh Gupta, Pablo Andrés Arbeláez, Ross B. Girshick, and Jitendra Malik. "Indoor Scene Understanding with RGB-D Images: Bottom-up Segmentation, Object Detection and Semantic Segmentation." In: *IJCV* 112.2 (2015).

[50] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan Dirk Wegner, Konrad Schindler, and Marc Pollefeys. "Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2017.

[51] Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. "Surface Networks via General Covers." In: *ICCV*. 2019.

[52] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. "SceneNet: Understanding Real World Indoor Scenes With Synthetic Data." In: *CVPR*. 2016.

[53] Christian Häne, Shubham Tulsiani, and Jitendra Malik. "Hierarchical Surface Prediction for 3D Object Reconstruction." In: *3DV*. 2017.

[54] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. "MeshCNN: a network with an edge." In: *ACM Trans. Graph.* 38.4 (2019), 90:1–90:12.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In: *CVPR*. 2016.

[56] Paul Henderson and Vittorio Ferrari. "Learning Single-Image 3D Reconstruction by Generative Modelling of Shape, Pose and Shading." In: *IJCV* (2019).

[57] Alexander Hermans, Georgios Floros, and Bastian Leibe. "Dense 3D semantic mapping of indoor scenes from RGB-D images." In: *ICRA*. 2014.

[58] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Alvar Vinacua, and Timo Ropinski. "Monte Carlo convolution for learning on non-uniformly sampled point clouds." In: *ACM Trans. Graph.* 37.6 (2018), 235:1–235:12.

[59] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects." In: *WACV* (2017).

[60] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. "CyCADA: Cycle-Consistent Adversarial Domain Adaptation." In: *ICML*. 2018.

[61] Derek Hoiem, Alexei A. Efros, and Martial Hebert. "Automatic photo pop-up." In: *ACM Trans. Graph.* 24.3 (2005), pp. 577–584.

[62] Berthold K.P. Horn. "Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View." PhD thesis. Cambridge, MA, USA: Massachusetts Institute of Technology, 1970.

[63] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. "Pointwise Convolutional Neural Networks." In: *CVPR*. 2018.

[64] Jingwei Huang, Haotian Zhang, Li Yi, Thomas A. Funkhouser, Matthias Nießner, and Leonidas J. Guibas. "TextureNet: Consistent Local Parametrizations for Learning From High-Resolution Signals on Meshes." In: *CVPR*. 2019.

[65] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks." In: *CVPR*. 2017.

[66] Eldar Insafutdinov and Alexey Dosovitskiy. "Unsupervised Learning of Shape and Pose with Differentiable Point Clouds." In: *NeurIPS*. 2018.

[67] Wenzel Jakob. *Mitsuba renderer*. http://www.mitsuba-renderer.org. 2010.

[68] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding." In: *arXiv:1408.5093* (2014).

[69] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. "Hierarchical Point-Edge Interaction Network for Point Cloud Semantic Segmentation." In: *ICCV*. 2019.

[70] Andrew Edie Johnson and Martial Hebert. "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes." In: *PAMI* 21.5 (1999).

[71] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution." In: *ECCV*. 2016.

[72] Adrian Johnston, Ravi Garg, Gustavo Carneiro, and Ian D. Reid. "Scaling CNNs for High Resolution Volumetric Reconstruction from a Single Image." In: *ICCV Workshops*. 2017.

[73] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. "3D Shape Segmentation with Projective Convolutional Networks." In: *CVPR*. 2017.

[74] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. "Learning Category-Specific Mesh Reconstruction from Image Collections." In: *ECCV*. 2018.

[75] Kevin Karsch, Ce Liu, and Sing Bing Kang. "Depth Transfer: Depth Extraction from Video Using Non-Parametric Sampling." In: *TPAMI* 36.11 (2014), pp. 2144–2158.

[76] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3D Mesh Renderer." In: *CVPR*. 2018.

[77] Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction." In: *SGP*. 2006.

[78] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. "Learning Compact Geometric Features." In: *ICCV*. 2017.

[79] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *ICLR*. 2014.

[80] Roman Klokov and Victor Lempitsky. "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models." In: *ICCV*. 2017.

[81] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. "Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction." In: *ACM Trans. Graph.* 36.4 (2017), 78:1–78:13.

[82] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. "ABC: A Big CAD Model Dataset For Geometric Deep Learning." In: *CVPR*. 2019.

[83] Artem Komarichev, Zichun Zhong, and Jing Hua. "A-CNN: Annularly Convolutional Neural Networks on Point Clouds." In: *CVPR*. 2019.

[84] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. "Using Locally Corresponding CAD Models for Dense 3D Reconstructions from a Single Image." In: *CVPR*. 2017.

[85] Philipp Krähenbühl and Vladlen Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials." In: *NIPS*. 2011.

[86] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *NIPS*. 2012.

[87] J. B. Kruskal. "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis." In: *Psychometrika* 29.1 (1964), 1—27.

[88] Abhijit Kundu, Yin Li, Frank Dellaert, Fuxin Li, and James M. Rehg. "Joint Semantic Segmentation and 3D Reconstruction from Monocular Video." In: *ECCV*. 2014.

[89] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer." In: *arXiv:1907.01341* (2019).

[90] Huan Lei, Naveed Akhtar, and Ajmal Mian. "Octree Guided CNN With Spherical Kernels for 3D Point Clouds." In: *CVPR*. 2019.

[91] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao (Richard) Zhang, and Leonidas J. Guibas. "GRASS: generative recursive autoencoders for shape structures." In: *ACM Trans. Graph.* 36.4 (2017), 52:1–52:14.

[92] Kejie Li, Trung Pham, Huangying Zhan, and Ian D. Reid. "Efficient Dense Point Cloud Object Reconstruction Using Deformation Vector Fields." In: *ECCV*. 2018.

[93] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. "Differentiable Monte Carlo ray tracing through edge sampling." In: *ACM Trans. Graph.* 37.6 (2018), 222:1–222:11.

[94] Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J. Guibas. "Joint embeddings of shapes and images via CNN image purification." In: *ACM Trans. Graph.* 34.6 (2015), 234:1–234:12.

[95] Yangyan Li, Sören Pirk, Hao Su, Charles Ruizhongtai Qi, and Leonidas J. Guibas. "FPNN: Field Probing Neural Networks for 3D Data." In: *NIPS*. 2016.

[96] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. "PointCNN: Convolution On X-Transformed Points." In: *NeurIPS*. 2018.

[97] Zhen Li, Yukang Gan, Xiaodan Liang, Yizhou Yu, Hui Cheng, and Liang Lin. "LSTM-CF: Unifying Context Modeling and Fusion with LSTMs for RGB-D Scene Labeling." In: *ECCV*. 2016.

[98] Zhengqi Li and Noah Snavely. "MegaDepth: Learning Single-View Depth Prediction from Internet Photos." In: *CVPR*. 2018.

[99] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. "Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction." In: *AAAI*. 2018.

[100] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. "Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields." In: *TPAMI* (2016).

[101] Angeline M. Loh. "The recovery of 3-D structure using visual texture patterns." PhD thesis. University of Western Australia, 2006.

[102] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In: *CVPR*. 2015.

[103] Matthew M. Loper and Michael J. Black. "OpenDR: An Approximate Differentiable Renderer." In: *ECCV*. 2014.

[104] William E. Lorensen and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm." In: *SIGGRAPH*. 1987.

[105] Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. "3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks." In: *CVPR*. 2017.

[106] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." In: *ICML Workshops*. 2013.

[107] James MacQueen. "Some methods for classification and analysis of multivariate observations." In: *Berkeley Symposium on Mathematical Statistics and Probability*. 1967.

[108] Priyanka Mandikal and Venkatesh Babu Radhakrishnan. "Dense 3D Point Cloud Reconstruction Using a Deep Pyramid Network." In: *WACV*. 2019.

[109] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. "Interpolated Convolutional Networks for 3D Point Cloud Understanding." In: *ICCV*. 2019.

[110] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. "Convolutional neural networks on surfaces via seamless toric covers." In: *ACM Trans. Graph.* 36.4 (2017).

[111] Andelo Martinovic, Jan Knopp, Hayko Riemenschneider, and Luc J. Van Gool. "3D all the way: Semantic segmentation of urban scenes from start to end in 3D." In: *CVPR*. 2015.

[112] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. "Geodesic convolutional neural networks on Riemannian manifolds." In: *ICCV Workshops*. 2015.

[113] F. J. Massey. "The Kolmogorov-Smirnov test for goodness of fit." In: *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78.

[114] Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. "Learning ambidextrous robot grasping policies." In: *Science Robotics* 4.26 (2019).

[115] Daniel Maturana and Sebastian A. Scherer. "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition." In: *IROS*. 2015.

[116] John McCormac, Ankur Handa, Andrew J. Davison, and Stefan Leutenegger. "SemanticFusion: Dense 3D semantic mapping with convolutional neural networks." In: *ICRA*. 2017.

[117] Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Tech. rep. IPL-TR-80-111. 1980.

[118] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy Networks: Learning 3D Reconstruction in Function Space." In: *CVPR*. 2019.

[119] Ondrej Miksik, Vibhav Vineet, Morten Lidegaard, Ram Prasaath, Matthias Nießner, Stuart Golodetz, Stephen L. Hicks, Patrick Pérez, Shahram Izadi, and Philip H. S. Torr. "The Semantic Paintbrush: Interactive 3D Mapping and Recognition in Large Outdoor Spaces." In: *CHI*. 2015.

[120] Daniel Munoz, Nicolas Vandapel, and Martial Hebert. "Onboard contextual classification of 3-D point clouds with learned high-order Markov Random Fields." In: *ICRA*. 2009.

[121] Chengjie Niu, Jun Li, and Kai Xu. "Im2Struct: Recovering 3D Shape Structure from a Single RGB Image." In: *CVPR*. 2018.

[122] Martin R. Oswald, Eno Töppe, and Daniel Cremers. "Fast and globally optimal single view reconstruction of curved objects." In: *CVPR*. 2012.

[123] Jeremie Papon and Markus Schoeler. "Semantic Pose Using Deep Networks Trained on Synthetic RGB-D." In: *ICCV*. 2015.

[124] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation." In: *CVPR*. 2019.

[125] Despoina Paschalidou, Ali O. Ulusoy, and Andreas Geiger. "Superquadrics Revisited: Learning 3D Shape Parsing Beyond Cuboids." In: *CVPR*. 2019.

[126] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *NeurIPS*. 2019.

[127] Benjamin Planche, Ziyan Wu, Kai Ma, Shanhui Sun, Stefan Kluckner, Terrence Chen, Andreas Hutter, Sergey I. Zakharov, Harald Kosch, and Jan Ernst. "DepthSynth: Real-Time Realistic Synthetic Data Generation from CAD Models for 2.5D Recognition." In: *3DV* (2017).

[128] Jhony K. Pontes, Chen Kong, Sridha Sridharan, Simon Lucey, Anders P. Eriksson, and Clinton Fookes. "Image2Mesh: A Learning Framework for Single Image 3D Reconstruction." In: *ACCV*. 2018.

[129] Jhony Kaesemodel Pontes, Chen Kong, Anders Eriksson, Clinton Fookes, Sridha Sridharan, and Simon Lucey. "Compact Model Representation for 3D Reconstruction." In: *3DV*. 2017.

[130] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. "Volumetric and Multi-view CNNs for Object Classification on 3D Data." In: *CVPR*. 2016.

[131] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: *NIPS*. 2017.

[132] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In: *CVPR*. 2017.

[133] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. "3D Graph Neural Networks for RGBD Semantic Segmentation." In: *ICCV*. 2017.

[134] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *ICLR*. 2016.

[135] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. *PyTorch3D*. https://github.com/facebookresearch/pytorch3d. 2020.

[136] Dario Rethage, Johanna Wald, Jürgen Sturm, Nassir Navab, and Federico Tombari. "Fully-Convolutional Point Networks for Large-Scale Point Clouds." In: *ECCV*. 2018.

[137] Danilo Jimenez Rezende, S. M. Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. "Unsupervised Learning of 3D Structure from Images." In: *NIPS*. 2016.

[138] Stephan R. Richter and Stefan Roth. "Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers." In: *CVPR*. 2018.

[139] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for Data: Ground Truth from Computer Games." In: *ECCV*. 2016.

[140] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. "OctNet: Learning Deep 3D Representations at High Resolutions." In: *CVPR*. 2017.

[141] Gernot Riegler, Ali Osman Ulusoy, Horst Bischof, and Andreas Geiger. "OctNetFusion: Learning Depth Fusion from Data." In: *3DV*. 2017.

[142] Lawrence G. Roberts. *Machine Perception of Three-Dimensional Solids*. Outstanding Dissertations in the Computer Sciences. 1963. ISBN: 0-8240-4427-4.

[143] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *MICCAI*. 2015.

[144] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast Point Feature Histograms (FPFH) for 3D registration." In: *ICRA*. 2009.

[145] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic Routing Between Capsules." In: *NIPS*. 2017.

[146] Samuele Salti, Federico Tombari, and Luigi di Stefano. "SHOT: Unique signatures of histograms for surface and texture description." In: *Computer Vision and Image Understanding* 125 (2014).

[147] J. W. Sammon. "A nonlinear mapping for data structure analysis." In: *IEEE Trans. Comp.* 18.5 (1969), 401—409.

[148] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. "Learning Depth from Single Monocular Images." In: *NIPS*. 2005.

[149] Ashutosh Saxena, Min Sun, and Andrew Y. Ng. "Learning 3-D Scene Structure from a Single Still Image." In: *ICCV*. 2007.

[150] Abhishek Sharma, Oliver Grau, and Mario Fritz. "VConv-DAE: Deep Volumetric Shape Learning Without Object Labels." In: *ECCV Workshops*. 2016.

[151] Toby Sharp et al. "Accurate, Robust, and Flexible Real-time Hand Tracking." In: *CHI*. 2015.

[152] Daeyun Shin, Charless C. Fowlkes, and Derek Hoiem. "Pixels, voxels, and views: A study of shape representations for single view 3D object shape prediction." In: *CVPR*. 2018.

[153] Jamie Shotton et al. "Efficient Human Pose Estimation from Single Depth Images." In: *TPAMI* 35 (2013).

[154] Martin Simonovsky and Nikos Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs." In: *CVPR*. 2017.

[155] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *ICLR*. 2015.

[156] Arjun Singh, James Sha, Karthik S. Narayan, Tudor Achim, and Pieter Abbeel. "BigBIRD: A large-scale 3D database of object instances." In: *ICRA*. 2014.

[157] Ayan Sinha, Jing Bai, and Karthik Ramani. "Deep Learning 3D Shape Surfaces Using Geometry Images." In: *ECCV*. 2016.

[158] Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D. Kulkarni, and Joshua B. Tenenbaum. "Synthesizing 3D Shapes via Modeling Multi-View Depth Maps and Silhouettes with Deep Generative Networks." In: *CVPR*. 2017.

[159] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. "Volumetric 3D mapping in real-time on a CPU." In: *ICRA*. 2014.

[160] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. "Multi-view Convolutional Neural Networks for 3D Shape Recognition." In: *ICCV*. 2015.

[161] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. "SPLATNet: Sparse Lattice Networks for Point Cloud Processing." In: *CVPR*. 2018.

[162] Hao Su, Charles Ruizhongtai Qi, Yangyan Li, and Leonidas J. Guibas. "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views." In: *ICCV*. 2015.

[163] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. "Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling." In: *CVPR*. 2018.

[164] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. "SEGCloud: Semantic Segmentation of 3D Point Clouds." In: *3DV*. 2017.

[165] Hugues Thomas, Charles Ruizhongtai Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. "KPConv: Flexible and Deformable Convolution for Point Clouds." In: *ICCV*. 2019.

[166] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world." In: *IROS*. 2017.

[167] Federico Tombari, Samuele Salti, and Luigi di Stefano. "Unique shape context for 3D data description." In: *ACM Workshop on 3D Object Retrieval*. 2010.

[168] Shubham Tulsiani, Abhishek Kar, João Carreira, and Jitendra Malik. "Learning Category-Specific Deformable 3D Models for Object Reconstruction." In: *TPAMI* 39.4 (2016), pp. 719–731.

[169] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. "Learning Shape Abstractions by Assembling Volumetric Primitives." In: *CVPR*. 2017.

[170] Shinji Umeyama. "Least-Squares Estimation of Transformation Parameters Between Two Point Patterns." In: *TPAMI* 13.4 (1991), pp. 376–380.

[171] Benjamin Ummenhofer and Thomas Brox. "Global, Dense Multiscale Reconstruction for a Billion Points." In: *ICCV*. 2015.

[172] Julien P. C. Valentin, Vibhav Vineet, Ming-Ming Cheng, David Kim, Jamie Shotton, Pushmeet Kohli, Matthias Nießner, Antonio Criminisi, Shahram Izadi, and Philip H. S. Torr. "SemanticPaint: Interactive 3D Labeling and Learning at your Fingertips." In: *ACM Trans. Graph.* 34.5 (2015).

[173] Julien Valentin, Cem Keskin, Pavel Pidlypenskyi, Ameesh Makadia, Avneesh Sud, and Sofien Bouaziz. *TensorFlow Graphics: Computer Graphics Meets Deep Learning*. 2019.

[174] Vibhav Vineet et al. "Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction." In: *ICRA*. 2015.

[175] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. "Graph Attention Convolution for Point Cloud Semantic Segmentation." In: *CVPR*. 2019.

[176] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images." In: *ECCV*. 2018.

[177] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. "O-CNN: Octree-based convolutional neural networks for 3D shape analysis." In: *ACM Trans. Graph.* 36.4 (2017).

[178] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. "Deep Parametric Continuous Convolutional Neural Networks." In: *CVPR*. 2018.

[179] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. "Dynamic Graph CNN for Learning on Point Clouds." In: *ACM Trans. Graph.* 38.5 (2019), 146:1–146:12.

[180] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. "Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation." In: *ICCV*. 2019.

[181] Olivia Wiles and Andrew Zisserman. "SilNet: Single- and Multi-View Reconstruction by Learning from Silhouettes." In: *BMVC*. 2017.

[182] Chenxia Wu, Ian Lenz, and Ashutosh Saxena. "Hierarchical Semantic Labeling for Task-Relevant RGB-D Perception." In: *RSS*. 2014.

[183] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. "Learning a probabilistic latent space of object shapes via 3D generative adversarial modeling." In: *NIPS*. 2016.

[184] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. "MarrNet: 3D Shape Reconstruction via 2.5D Sketches." In: *NIPS*. 2017.

[185] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. "Learning 3D Shape Priors for Shape Completion and Reconstruction." In: *ECCV*. 2018.

[186] Wenxuan Wu, Zhongang Qi, and Fuxin Li. "PointConv: Deep Convolutional Networks on 3D Point Clouds." In: *CVPR*. 2019.

[187] Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. "SAGNet: structure-aware generative network for 3D-shape modeling." In: *ACM Trans. Graph.* 38.4 (2019), 91:1–91:14.

[188] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: A deep representation for volumetric shapes." In: *CVPR*. 2015.

[189] Ke Xian, Chunhua Shen, Zhiguo Cao, Hao Lu, Yang Xiao, Ruibo Li, and Zhenbo Luo. "Monocular Relative Depth Perception with Web Stereo Data Supervision." In: *CVPR*. 2018.

[190] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. "SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters." In: *ECCV*. 2018.

[191] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision." In: *NIPS*. 2016.

[192] Li Yi, Lin Shao, Manolis Savva, et al. "Large-Scale 3D Shape Reconstruction and Segmentation from ShapeNet Core55." In: *arXiv:1710.06104* (2017).

[193] Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. "SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation." In: *CVPR*. 2017.

[194] Mehmet Ersin Yümer and Niloy J. Mitra. "Learning Semantic Deformation Flows with 3D Convolutional Networks." In: *ECCV*. 2016.

[195] Ruo Zhang, Ping sing Tsai, James Edwin Cryer, and Mubarak Shah. "Shape from Shading: A survey." In: *TPAMI* 21 (1999), pp. 690–706.

[196] Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Joshua B Tenenbaum, William T Freeman, and Jiajun Wu. "Learning to Reconstruct Shapes from Unseen Classes." In: *NeurIPS*. 2018.

[197] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas A. Funkhouser. "Physically-Based Rendering for Indoor Scene Understanding Using Convolutional Neural Networks." In: *CVPR*. 2017.

[198] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. "ShellNet: Efficient Point Cloud Convolutional Neural Networks using Concentric Shells Statistics." In: *ICCV*. 2019.

[199] Zhiyuan Zhang, Binh-Son Hua, David W. Rosen, and Sai-Kit Yeung. "Rotation Invariant Convolutions for 3D Point Clouds Deep Learning." In: *3DV*. 2019.

[200] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. "3D Point Capsule Networks." In: *CVPR*. 2019.

[201] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. "Does computer vision matter for action?" In: *Science Robotics* 4.30 (2019).

[202] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing." In: *arXiv:1801.09847* (2018).

[203] Qingnan Zhou and Alec Jacobson. "Thingi10K: A Dataset of 10,000 3D-Printing Models." In: *arXiv:1605.04797* (2016).

[204] Rui Zhu, Hamed Kiani Galoogahi, Chaoyang Wang, and Simon Lucey. "Rethinking Reprojection: Closing the Loop for Pose-Aware Shape Reconstruction from a Single Image." In: *ICCV*. 2017.

[205] Yuke Zhu, Oliver Groth, Michael S. Bernstein, and Li Fei-Fei. "Visual7W: Grounded Question Answering in Images." In: *CVPR*. 2016.