

Gernot Erich Riegler

Deep Learning for 2.5D and 3D

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Horst Bischof
Institute for Computer Graphics and Vision
Graz Universtiy of Technology

Dr. Andreas Geiger
Autonomous Vision Group
Max Planck Institute for Intelligent Systems Tübingen

Graz, Austria, Sep. 2017

To Christina, Severin and Adrian

If people never did silly things
nothing intelligent would ever get done.

Ludwig Wittgenstein (1889 - 1951)

Abstract

Deep learning based method are revolutionizing many fields in computer science, especially in computer vision, speech recognition and natural language processing. These methods have in common that they learn a hierarchy of non-linear feature extractors together with a classifier, or regressor in an end-to-end fashion. Although the underlying concepts, especially artificial neural networks and the backpropagation algorithm, are several decades old, they came to new importance mainly due to today's sheer amount of available data and parallel compute power, i.e. GPGPUs. The applications in computer vision mainly focus on recognition tasks on natural images which are 2D projections of the world. In contrast, in this thesis we propose methods that investigate deep learning specifically for 2.5D and 3D data.

2.5D data can be understood as a 2D image that has a depth value associated with each pixel. There exists now a wide range of sensor technologies that enables the fast and cheap recording of such depth maps. However, most of those recordings are influenced by noise and have a low spatial resolution. We propose a novel technique that combines deep convolutional networks with a variational model to tackle this depth super-resolution problem, i.e. a method that increases the spatial resolution and simultaneously reduces the influence of the noise. As it is difficult to obtain high-resolution, high-quality depth maps as training data for this task, we optimize our joint model in an end-to-end fashion on a large corpus of synthetically generated data. In an extensive evaluation on three benchmark datasets, we validate our method that significantly outperforms state-of-the-art methods.

Deep learning on volumetric 3D data faces one crucial problem that is independent of its application: The memory consumption increases cubically with respect to the input resolution, whereas the memory of GPGPUs is limited. We propose in this thesis a drop-in solution based on an efficient space partitioning data structure. Instead of a regular voxel grid, we utilize an octree within the network to focus memory and computation on more relevant regions of the input. With this new technique, we are able to increase the input

resolution by at least a factor of $\times 64$ without losing the representational power of the convolutional network. Further, we show a series of tasks where a detailed representation of the 3D input is absolutely beneficial, i.e. the performance increases simply by increasing the input resolution.

An alleged drawback of this method might be that the space partitioning has to be known in advance. Hence, we also show in this thesis how we can learn the splitting of the octree along with a volumetric 3D reconstruction to circumvent the problem. We subsequently utilize this method for volumetric depth fusion and volumetric depth completion. For the former, our method outperforms established baselines, especially on difficult settings where we have only a low number of input views, or severe input noise. For the latter, i.e. predicting the whole 3D scene from a single depth input, we achieve new state-of-the-art results.

Kurzfassung

Methoden, die auf Deep Learning (auf Deutsch etwa *tiefgehendes Lernen*) basieren, revolutionieren viele Bereiche in der Informatik, vor allem im maschinellen Sehen, der Spracherkennung und der Computerlinguistik. Die Gemeinsamkeit dieser Methoden ist, dass sie eine Hierarchie von nicht-linearen Merkmalsextraktoren zusammen mit einem Klassifikator oder Regressor lernen. Obwohl die zugrunde liegenden Konzepte, allen voran künstliche neuronale Netze und der Backpropagation-Algorithmus, mehrere Jahrzehnte alt sind, kamen sie vor allem aufgrund der heutigen Fülle an verfügbaren Daten und der massiven parallelen Rechenleistung, i.e. GPGPUs, zur neuen Bedeutung. Die Anwendungen im maschinellen Sehen konzentrieren sich vor allem auf Erkennungsaufgaben in natürlichen Bildern, die eine 2D-Projektion der Welt sind. Im Gegensatz dazu präsentieren wir in dieser Arbeit Methoden, die Deep Learning speziell auf 2.5D und 3D Daten anwenden.

2.5D Daten können als 2D Bilder verstanden werden, die jedem Bildpunkt einen Tiefenwert zuordnen. Es gibt bereits jetzt eine breite Palette an Sensoren, die die schnelle und kostengünstige Erfassung solcher Tiefenbilder ermöglichen. Allerdings sind die meisten dieser Aufnahmen von Rauschen beeinflusst und haben eine geringe Bildauflösung. Wir schlagen eine neuartige Methodik vor, die Deep Convolutional Networks (auf Deutsch etwa tiefgehende Faltungsnetzwerke) mit einem Variationsmodell kombiniert, um das Problem der geringen Auflösung und des Rauschens zu verbessern. Da das Aufzeichnen von hochauflösenden, hochqualitativen Tiefendaten ein Hindernis ist, optimieren wir unser kombiniertes Modell auf einem großen Korpus von synthetisch erzeugten Daten. In einer umfangreichen Evaluierung auf drei Standard-Datensätzen bestätigen wir die Genauigkeit unserer Methode, welche den aktuellen Stand der Technik deutlich übertrifft.

Deep Learning auf volumetrischen 3D Daten hat ein entscheidendes Problem, welches unabhängig von der Anwendung ist: Der Speicherverbrauch steigt kubisch in Bezug auf die Auflösung des 3D Volumens an, während der Speicher von GPGPUs begrenzt ist. Wir präsentieren in dieser Arbeit eine Lösung basierend auf einer Datenstruktur, welche das

3D Volumen effizient unterteilt. Anstelle eines dichten Voxelgitters verwenden wir einen Octree in der Netzwerkarchitektur um den Speicher und die Berechnungen auf spezielle Teilbereiche des Volumens zu fokussieren. Mit dieser neuen Methodik sind wir in der Lage, die Auflösung des Eingangsvolumens um mindestens einen Faktor von $\times 64$ zu erhöhen, ohne das die Genauigkeit des Faltungsnetzes abnimmt. Darüber hinaus zeigen wir in einer Reihe von Anwendungen, dass eine detaillierte Darstellung der 3D Daten absolut vorteilhaft ist, i.e. die Genauigkeit steigt einfach durch die Erhöhung der Auflösung des Eingangsvolumens.

Ein vermeintlicher Nachteil dieser Technik ist, dass die Aufteilung des 3D Raums im Voraus bekannt sein muss. Daher zeigen wir in dieser Arbeit auch wie die Octree Datenstruktur zusammen mit einer volumetrischen 3D-Rekonstruktion gelernt werden kann. Wir verwenden diese Technik anschließend für die volumetrische Fusionierung von Tiefenbildern, sowie für die komplette 3D Rekonstruktion einer Szene von einem einzigen Tiefenbild. Unsere Methode übertrifft für beide Anwendungen andere etablierte Verfahren. Vor allem bei schwierigen Szenarien, wie einer geringen Anzahl von Tiefenbildern oder einem hohen Anteil an Rauschen, zeichnet sich unsere Methodik aus.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Acknowledgments

This thesis would not have been imaginable without the great help and backing of so many people, who supported me throughout this whole process called Ph.D. studies.

First of all, I want to thank my supervisor Horst Bischof. Already back in my bachelor and master studies, he aroused my interest in computer vision and machine learning in his lectures that now culminated in this thesis. He gave me the freedom to pursue topics that fitted my research interests. I am forever grateful for this experience. I also want to thank Andreas Geiger for being my second supervisor, but also for the marvelous time I spent in Tübingen. It was an amazing experience to visit the MPI and to have the fruitful ongoing collaborations.

Great thanks are also due to the fascinating people of ICG. It was an honor to work, talk and party with so many interesting people here; memories that shall never be forgotten. Especially, I want to thank the people with whom I had the fortune to collaborate more closely. First, Matthias Rüther, the leader of the Robot Vision group. He took me on board on my first project in collaboration with Infineon Technologies Austria that started my Ph.D. studies and led to my first publications. A token of thanks also to David Ferstl, who was an invaluable support throughout my work here. We did not only share and discuss a lot of fascinating scientific ideas but also developed a friendship that I do not want to miss. I am also very grateful to have worked with René Ranftl and Samuel Schulter, who taught me a great deal about scientific work in the fast-paced world of computer vision. Thanks also to my other co-authors Ali Osman Ulusoy, Christian Reinbacher, Thomas Pock, Markus Oberweger, Paul Wohlhart, Vincent Lepetit, Martin Urschler and Darko Stern for the wonderful collaborations that led to publications.

My office mates throughout this journey deserve also a great round of applause: Christian Reinbacher, David Ferstl, Ludwig Mohr, and Fabian Schenk. You contributed a big deal to making this adventure so funny and enjoyable.

Finally, I want to thank my family. My mum, dad, and grandfather were always eager that I can enjoy a good education and supported me in my decisions. I thank you so much! Last but not least, I want to thank my wife Christina and my sons, Severin and Adrian. Your love makes life better every day!

Contents

Abstract vii

Kurzfassung ix

Affidavit xi

Acknowledgments xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Outline and Contributions . . . 5

2 Convex Optimization in Imaging and Deep Learning 7
- 2.1 Notation and Definitions . . . 8
- 2.2 Convex Optimization . . . 13
  - 2.2.1 Convex Analysis . . . 13
  - 2.2.2 Convex Optimization Algorithms . . . 20
  - 2.2.3 Convex Optimization for Imaging . . . 24
- 2.3 Deep Learning . . . 30
  - 2.3.1 Supervised Learning . . . 33
  - 2.3.2 Feed-Forward Networks . . . 36
  - 2.3.3 Network Optimization . . . 39
  - 2.3.4 Backpropagation . . . 42
  - 2.3.5 Convolutional Networks . . . 44
  - 2.3.6 Losses and Regularization . . . 48
- 2.4 Summary . . . 51

3 Deep Learning for 2.5D **53**
3.1 Introduction . . . 53
3.2 Related Work . . . 56
3.2.1 Super-Resolution . . . 56
3.2.2 Depth Super-Resolution . . . 58
3.2.3 Joint Network Training and Energy Minimization . . . 60
3.2.4 Datasets with Ground-Truth Depth . . . 61
3.3 Deep Learning Meets Variational Methods . . . 61
3.3.1 Implicit Differentiation . . . 63
3.3.2 Unrolling Optimization Scheme . . . 66
3.3.3 Training Data . . . 69
3.4 Evaluation . . . 70
3.4.1 Super-Resolution Network . . . 71
3.4.2 Training Loss . . . 76
3.4.3 Variational Model . . . 76
3.4.4 Variational Network Training . . . 79
3.4.5 Depth Super-Resolution . . . 79
3.4.6 Guided Depth Super-Resolution . . . 96
3.5 Summary & Discussion . . . 99

4 Deep Learning for 3D **103**
4.1 Introduction . . . 103
4.2 Related Work . . . 106
4.2.1 Dense Models . . . 107
4.2.2 Sparse Models . . . 109
4.2.3 Space Partitioning Functions . . . 110
4.2.4 Volumetric Fusion . . . 110
4.2.5 Shape Completion . . . 111
4.3 Deep Learning for High-Resolution 3D . . . 113
4.3.1 OctNet . . . 113
4.3.2 Dynamic OctNet . . . 120
4.4 Evaluation . . . 122
4.4.1 3D Shape Classification . . . 123
4.4.2 3D Orientation Estimation . . . 127
4.4.3 3D Point Cloud Segmentation . . . 132
4.4.4 Depth Fusion and Completion . . . 136
4.5 Summary & Discussion . . . 155

5 Conclusion and Outlook **159**
5.1 Conclusion . . . 159
5.1.1 Deep Learning for 2.5D . . . 159

5.1.2 Deep Learning for 3D 161
5.2 Outlook 162

A List of Publications **163**
A.1 2017 163
A.2 2016 164
A.3 2015 166
A.4 2014 168

B Proofs **171**

Bibliography **177**

List of Figures

1.1 Depth super-resolution problem . . . 2
1.2 2.5D vs. 3D . . . 3
1.3 Volumetric depth fusion problem . . . 5

2.1 Unit norm balls for different p-norms . . . 10
2.2 Examples of convex and non-convex sets . . . 14
2.3 The epigraph of a function . . . 14
2.4 Example of a convex function . . . 15
2.5 The convex conjugtate and bi-conjugate of a function . . . 17
2.6 Example of a sub-gradient . . . 17
2.7 Tikhonov denoising example . . . 26
2.8 ROF denoising example . . . 27
2.9 Huber denoising example . . . 29
2.10 TGV denoising example . . . 31
2.11 Influence of regularization terms on depth maps . . . 32
2.12 Under- and over-fitting and bias-variance decomposition . . . 36
2.13 Computational graph of a function with corresponding differential graph . . . 43
2.14 Convolution and pooling operation . . . 45

3.1 Variational methods improve the network's high-resolution estimate . . . 55
3.2 Computation graph of the unrolled primal-dual optimization scheme . . . 66
3.3 Non-local neighborhood for regularization term . . . 68
3.4 Synthethic generated training data . . . 70
3.5 Pixel shuffle . . . 73
3.6 Network architectures . . . 74
3.7 Qualitative results for the Middlebury disparity map *Cones*, $\times 4$ . . . 81
3.8 Qualitative results for the Middlebury disparity map *Tsukuba*, $\times 4$ . . . 82

3.9 Qualitative results for the Middlebury disparity map *Venus*, ×4 83
3.10 Qualitative results for the Noisy Middlebury disparity map *Art*, ×4 87
3.11 Qualitative results for the Noisy Middlebury disparity map *Books*, ×4 . . . 88
3.12 Qualitative results for the Noisy Middlebury disparity map *Moebius*, ×4 . . 89
3.13 Qualitative results for the Noisy Middlebury disparity map *Art*, ×16 90
3.14 Qualitative results for the Noisy Middlebury disparity map *Books*, ×16 . . . 91
3.15 Qualitative results for the Noisy Middlebury disparity map *Moebius*, ×16 . 92
3.16 Qualitative results for the ToFMark image *Books* 93
3.17 Qualitative results for the ToFMark image *Devil* 94
3.18 Qualitative results for the ToFMark image *Shark* 95
3.19 Guided network architectures 98

4.1 Sparse activations of a 3D convolutional network 105
4.2 2D pixel grid vs. 3D voxel grid 107
4.3 Sparsity of 3D data 108
4.4 Hybrid grid-octree data structure 113
4.5 Shallow octree bit representation 115
4.6 Data index computation example 116
4.7 Octree convolution 118
4.8 Efficient octree convolution implementation 119
4.9 Octree pooling 119
4.10 Octree unpooling 121
4.11 OctNet structure module 121
4.12 Octree splitting 122
4.13 OctNet ResNet for different input resolutions 124
4.14 Memory consumption of standard voxel grids compared to hybrid grid-octrees 125
4.15 Memory consumption of a 3D convolutional network on voxel grids and hybrid grid-octrees 125
4.16 Runtime of a 3D convolutional network on voxel grids and hybrid grid-octrees 126
4.17 Classification accuracy for ModelNet10 and ModelNet40 126
4.18 Confusion matrices of the ModelNet10 classification results 127
4.19 ModelNet ambiguities 128
4.20 Quantitative orientation estimation results on ModelNet10 129
4.21 Qualitative orientation estimation results on ModelNet10 130
4.22 Quantitative orientation estiamtion results on the head pose dataset 132
4.23 Qualitative orientation estimation results on the head pose dataset 133
4.24 U-shaped semantic segmentation network 134
4.25 Quantitative results on the VarCity dataset 135
4.26 Qualitative results on the VarCity dataset 137
4.27 OctNetFusion coarse-to-fine network architecture 138
4.28 Encoder-Decoder module 138

4.29 Volumetric depth fusion set-up 141
4.30 Qualitative volumetric depth fusion results for different output resolutions . 149
4.31 Qualitative volumetric depth fusion results wrt. number of input views . . . 150
4.32 Qualitative volumetric depth fusion results wrt. input noise 151
4.33 Qualitative volumetric depth completion results on the Kinect object scans 153
4.34 Qualitative volumetric depth completion results on the tabletop dataset . . 156

List of Tables

3.1 Evaluation of the network architectures . . . 75
3.2 Evaluation of the training losses . . . 77
3.3 Evaluation of the variational models . . . 78
3.5 Evaluation of the variational network training . . . 80
3.6 Evaluation of unguided depth super-resolution on the noise-free Middlebury dataset . . . 84
3.7 Evaluation of unguided depth super-resolution on the noisy Middlebury dataset . . . 86
3.8 Evaluation of unguided depth super-resolution on ToFMark dataset . . . 96
3.9 Evaluation of the guidance networks . . . 97
3.10 Evaluation of guided depth super-resolution on the noisy Middlebury dataset 100
3.11 Evaluation of guided depth super-resolution on ToFMark dataset . . . 101

4.1 Shallow octree trade-off . . . 114
4.2 Quantitative volumetric depth fusion results of different input encodings . . . 143
4.4 Quantitative volumetric depth fusion results wrt. number of input views . . . 144
4.6 Quantitative volumetric depth fusion results wrt. input noise . . . 145
4.8 Quantitative volumetric depth fusion results wrt.. seen vs. unseen categories 146
4.10 Evaluation on Kinect object scans . . . 152
4.12 Qunatitative volumetric depth completion results on the tabletop dataset . 154

CHAPTER 1

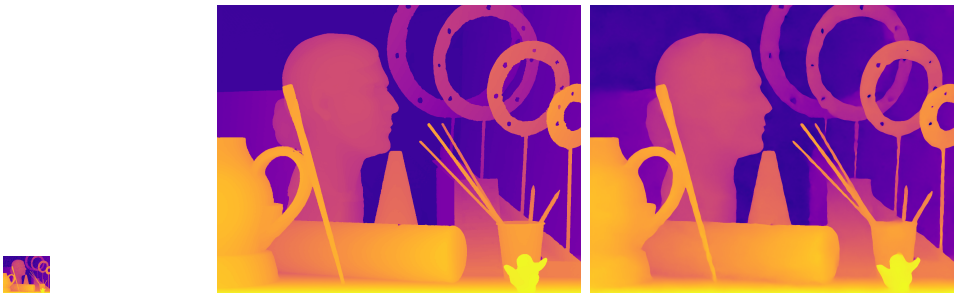
Introduction

1.1 Motivation

We as humans live in a world with three spatial dimensions (3D). Equipped with a pair of eyes we, like most mammals, are able to perceive stereoscopic depth (2.5D), i.e. given the images from two slightly displaced positions, we compute the offset of objects in the two images which is inversely proportional to its depth. However, we know from experiments [241] that this metric perception of depth is actually rather poor in humans. Still, we are able to perceive 3D objects and their relations accurately and interact with them. That is the case because we heavily rely on other clues that we derive from higher level knowledge and reasoning about objects [82] and motion [193, 194].

Most of this knowledge is learned and motivated already the seminal work by Roberts [191] in 1963. In his thesis Roberts developed a system that infers the 3D geometry of a scene from a single image of it. His approach was by today's standards pretty simple: He used a dataset of known objects, computed the edges of the images and fitted the 3D objects to the edges. However, this technique is especially interesting, because it is able to recover the geometry of the non-visible surface, i.e. occluded and back-facing segments.

Despite its visionary work even decades afterwards we have still not solved the problem such that it works on arbitrary images and scenes, but we are getting closer. There is in particular one main ingredient that recently fueled a lot of major progress in the field of computer vision [132], but also in speech recognition [11, 105], natural language processing [227] and many others: deep learning. If we talk about deep learning we usually refer to artificial neural networks whose foundations already date back several decades [74, 104, 156, 196, 199]. What changed is the availability of massive parallel computational



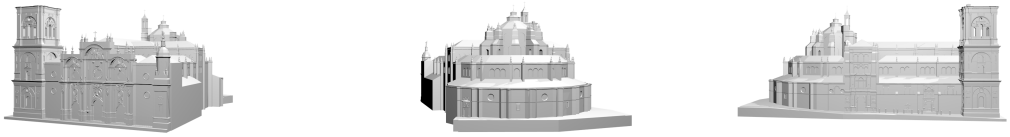
(a) Input, 174 × 139 **(b)** Ground-Truth, 1392 × 1112 **(c)** Estimate, 1392 × 1112

Figure 1.1: The depth super-resolution problem. Given a noisy, low-resolution depth map as input (a), the goal is to recover a plausible high-resolution estimate (b)-(c). The problem is ill-posed because infinitely many high-resolution depth maps can map the noisy, low-resolution input.

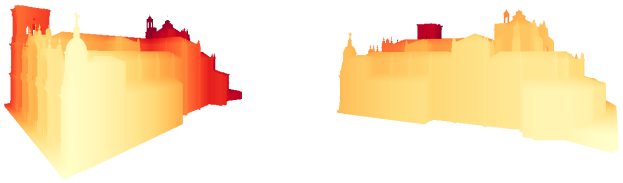
power provided by modern general-purpose graphics processing units (GPGPUs) and large annotated datasets like ImageNet [51], which combined with recent advances in network training [117, 162] enabled the training of ever deeper and hence more expressive networks. Nowadays deep learning based methods are state-of-the-art in all image based recognition tasks, like image classification [102, 132, 220, 228], object detection [85, 86, 147, 178, 179, 180], and semantic segmentation [6, 34, 35, 81, 148, 267].

In this thesis, we focus on domains that are not yet fully exploited by the success of deep learning, i.e. 2.5D and 3D. 2.5D data generally describes 2D projections like images, but instead of color values per pixel, each location encodes the depth to the first surface along the viewing ray. Formally, a depth map can be described as a function $d(u, v) : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ that maps each pixel location u, v of the image domain Ω to a depth value. This value is the distance to the first surface along the viewing direction. By applying the inverse camera matrix K to compute the viewing rays and multiplying those by the distance $(x, y, z)^T = d(u, v)K^{-1}(u, v, 1)^T$ we can observe that a depth map is equivalent to a 3D point cloud of an object, or scene viewed from a single viewpoint [100]. An important difference to a full 3D representation is that information about the structure behind this first surface is lost in the projection. In contrast, 3D commonly refers to a mathematical representation of the whole 3D surface of an object, or scene. There exist many equivalent mathematical representations for this. One used in this thesis is the signed distance function $s(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$ that assigns each 3D point (x, y, z) the distance to the closest surface boundary. The distance is negative, if the point lies inside an object, and positive otherwise. In addition, the surface is the zero level-set $\{(x, y, z) : s(x, y, z) = 0\}$ of the signed distance function. See Figure 1.2 for a visualization.

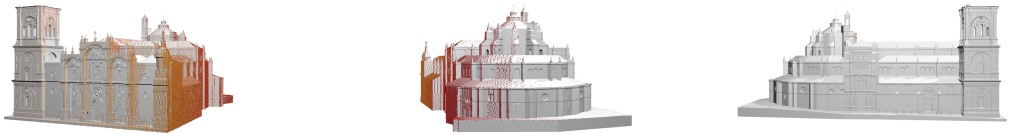
Naturally, 2.5D and 3D are closely related and with the advent of low-cost depth sensors like Microsoft Kinect V2, Intel DepthSense, Creative Senz3D, or PMD Flexx a new range of exciting applications was enabled: For example in human pose estimation [87, 243], head pose estimation [64, 183, 208, 229], hand pose estimation [167, 168, 182, 230, 244], object detection [95], or real-time 3D reconstruction [165]. Those sensors can be produced



(a) 3D object



(b) 2.5D depth maps



(c) Points from depth map projected onto 3D object

Figure 1.2: 2.5D vs. 3D. The 3D object in (a) is defined by a mathematical representation of its 3D surface. In contrast, 2.5D depth maps as in (b) contain only a subset of points that lie on this 3D surface, see (c). A depth map only encodes the first surface it hits along the viewing ray and the remaining information is lost due to the projection to 2D, i.e. everything behind the first surface.[1]

in a small package size and have a small energy footprint which makes them also applicable in mobile applications. However, the depth maps are also affected by degenerations due to noise, quantization and missing values and moreover, they typically have a low lateral resolution. The estimation of the high-resolution depth map that corresponds to the noisy low-resolution input is an ill-posed problem as there exists no unique solution. This makes the depth super-resolution problem a challenging task as depicted in Figure 1.1

The problem is related to the single image super-resolution problem on natural images, with the difference that depth maps do not exhibit any texture, lighting, or shading and mostly contain piece-wise affine surface with sharp depth discontinuities. This knowledge

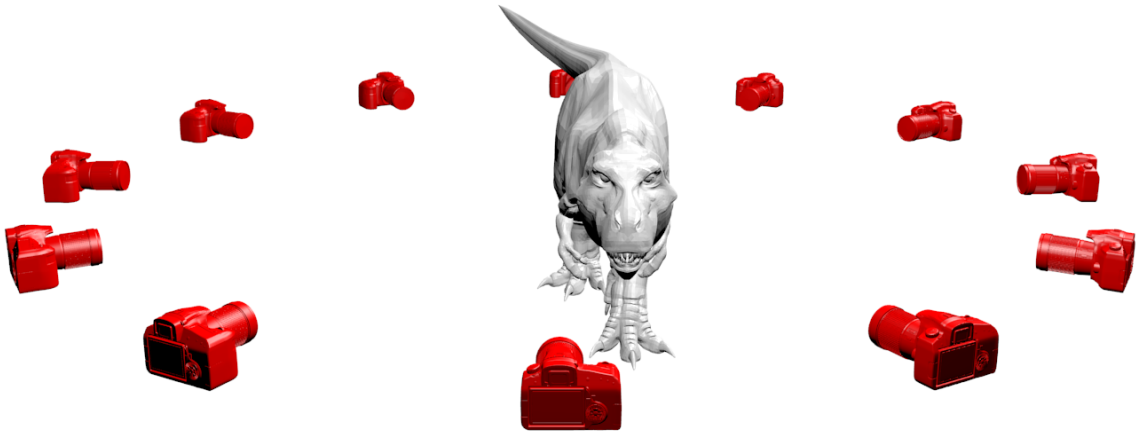
[1] 3D model taken from `https://www.turbosquid.com/FullPreview/Index.cfm/ID/878305`, last accessed on October 26, 2017.

can be incorporated into priors of global energy minimization models to tackle this problem [52, 67]. But we have also observed that on natural images that deep learning based methods [55, 126, 152, 215] outperform other methods by a large margin. Hence, motivated by the success of those methods on single image super-resolution for color images we propose in this thesis a method to combine best of both worlds. Unlike as for color images, we can not collect the necessary training data from the web in large quantities. However, we demonstrate that we can obtain state-of-the-art results by training a deep convolutional network on carefully generated synthetic depth maps. Interestingly, we observe that a post-processing step with a variational model further improves the accuracy of the high-resolution estimate. Therefore, we additionally propose the tight integration of such a variational model on top of the deep network to train both models end-to-end which further pushes the accuracy of our model.

As long as we process 2.5D data we can apply deep networks on the common 2D pixel grid. The situation changes with 3D data, where an additional dimension is needed to represent the input and depending on the application for the output, too. The typical approach is to elevate the input representation from a 2D pixel grid to a 3D voxel grid [41, 154, 176, 259]. While 3D data can be represented in various ways, the 3D voxel grid is the natural choice for convolutional networks: First, we can simply define all the useful network operations on a 2D pixel grid in a similar way on a 3D voxel grid, for example, convolution, pooling, and normalization. Second, this further enables the reuse of established 2D network architectures for the corresponding problems in 3D. Finally, we can not only represent the object surface with signed distance functions in the 3D voxel grid, but we can also incorporate supplementary information in additional feature channels of the grid, e.g. surface normals, or color. One major drawback of this approach, however, is the cubic increase of memory with respect to the input resolution. This limits the model complexity that fits on a single GPGPU.

Based on the assumption that 3D data is usually sparse, we propose in this thesis the utilization of an efficient space-partitioning data structure within the network to tackle this memory problem. More specifically, we use a hybrid grid-octree representation where we focus memory and computation on interesting segments of the 3D volume, i.e. near the surface, and summarize the other parts of the volume in larger octree cells. We define the most common network operations on this special data structure and therefore, our method can be used as a drop-in replacement for 3D networks on regular voxel grids, allowing the training on higher-resolution inputs. In fact, we show that our method can handle an input resolution that is at least larger by a factor of $\times 64$ without changing the architecture, or training setting.

A limitation of this method is that the space-partitioning has to be known in advance. This is no problem for tasks like shape classification and orientation estimation, or semantic segmentation, where the output is either a 1D vector or has the same structure as the input, respectively. For 3D reconstruction tasks, e.g. volumetric depth fusion, or volumetric depth fusion, as visualized in Figure 1.3 the input structure is different to the output structure



(a) 3D Volumetric Depth Fusion Set-Up



(b) 2.5D depth maps

Figure 1.3: Volumetric depth fusion problem. Each camera in (a) records a depth map of the 3D object. Given those depth maps as depicted in (b), the goal of volumetric depth fusion is to recover the 3D object.[2]

and not known a priori. Therefore, we further extend our method such that it can output a reconstruction and jointly computes the supporting octree structure of the output. We enable this by introducing a structure split module that derives an octree split mask given an intermediate reconstruction. Embedded in a coarse-to-fine architecture we demonstrate superior 3D volumetric depth fusion and volumetric depth fusion results.

1.2 Outline and Contributions

The thesis is organized as follows: In Chapter 2 we first introduce the mathematical notation and definitions used throughout this work. Additionally, we overview the basic concepts of convex analysis, e.g. convex sets, convex functions and how they can be

[2] 3D models are taken from `https://www.turbosquid.com/FullPreview/Index.cfm/ID/933905` and `https://www.turbosquid.com/FullPreview/Index.cfm/ID/1138743`, last accessed on October 26, 2017.

efficiently optimized. These concepts are then applied to solve variational models for imaging problems, a core building block of our depth super-resolution method that we will encounter later. Finally, the chapter introduces the basic concepts of supervised machine learning and then goes into detail on deep learning and convolutional networks, the main ingredient of this thesis.

Chapter 3 is dedicated to deep learning on 2.5D data. More specifically, we propose a novel method to tackle the depth super-resolution problem by combining a deep convolutional network and a variational method in an end-to-end framework. The network gets a single low-resolution depth map as input along with an optional high-resolution guidance image and outputs a highly accurate high-resolution estimate. We demonstrate that training the joint model on a large corpus of synthetic data we are able to outperform current state-of-the-art methods by a large margin.

In Chapter 4 we address deep learning on 3D data. The major problem with 3D data in the context of deep learning is the quadratic increase of memory that quickly exceeds the available GPGPU memory. We propose a technique based on an efficient space partitioning data structure to focus memory and computation on interesting parts of the input volume, i.e. the surface of a 3D shape, and share them on other parts of the volume. Our method drastically decreases the memory footprint and enables faster learning, but is at the same time as expressive as convolutional networks operating on an equivalent regular voxel grid. Further, we show how this method can then be used for volumetric depth fusion and volumetric depth completion in an end-to-end fashion. The clue is that we can learn the space partitioning for this task along with the reconstruction.

The thesis is based on the research that was presented in the publications [181, 184, 185, 187, 188]. A full list of (co-)authored publications along with the abstracts is listed in Appendix A.

CHAPTER 2

Convex Optimization in Imaging and Deep Learning

Contents

2.1 Notation and Definitions . . . 8
2.2 Convex Optimization . . . 13
2.2.1 Convex Analysis . . . 13
2.2.2 Convex Optimization Algorithms . . . 20
2.2.3 Convex Optimization for Imaging . . . 24
2.3 Deep Learning . . . 30
2.3.1 Supervised Learning . . . 33
2.3.2 Feed-Forward Networks . . . 36
2.3.3 Network Optimization . . . 39
2.3.4 Backpropagation . . . 42
2.3.5 Convolutional Networks . . . 44
2.3.6 Losses and Regularization . . . 48
2.4 Summary . . . 51

In this thesis, we present novel deep learning methods for 2.5 and 3D data. This chapter first introduces the basic notation and definitions used throughout the thesis. Then, we give an overview on the broad topic of convex optimization in computer vision. Finally, we introduce the basics of supervised learning and more specifically deep learning with convolutional networks. Those methods and algorithms build the foundation of our work.

2.1 Notation and Definitions

This section introduces the notation and definitions which are used throughout this thesis. We mostly stick to a notation that is common in modern scientific literature. Scalar values are written in lower-case Latin or Greek letters, e.g. x, λ. For vectors, we use lower-case and bold letters $\mathbf{x} = (x_1, \dots, x_N)^T$. Further, if not otherwise stated we assume that a vector $\mathbf{x}$ is an element of a real vector space $\mathbb{R}^N$, i.e. $\mathbf{x} \in \mathbb{R}^N$. To denote matrices we use an upper-case bold letter like $\mathbf{A}$. Again, we assume that the entries a_{ij} of the matrix are real numbers, hence $\mathbf{A} \in \mathbb{R}^{M \times N}$ with $i \in \{1, \dots M\}$ and $j \in \{1, \dots N\}$ denoting the row and column indices, respectively.

First, we need to define a space on which we can further define operations.

Definition 2.1 (Vector Space)**.** Let X be a set. We call X a vector space iff the following 8 conditions for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in X$ and $\lambda, \mu \in \mathbb{R}$ hold.

$$\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x} \qquad \text{(commutativity)} \tag{2.1}$$
$$(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z}) \qquad \text{(associativity)} \tag{2.2}$$
$$0 + \mathbf{x} = \mathbf{x} \qquad \text{(additive identity)} \tag{2.3}$$
$$\mathbf{x} + (-\mathbf{x}) = 0 \qquad \text{(additive inverse)} \tag{2.4}$$
$$(\lambda + \mu)\mathbf{x} = \lambda\mathbf{x} + \mu\mathbf{x} \qquad \text{(scalar distributivity)} \tag{2.5}$$
$$\lambda(\mathbf{x} + \mathbf{y}) = \lambda\mathbf{y} + \lambda\mathbf{x} \qquad \text{(vector distributivity)} \tag{2.6}$$
$$1\mathbf{x} = \mathbf{x} \qquad \text{(scalar multiplication identity)} \tag{2.7}$$

Simply put, a vector space is a mathematical structure in which linear combinations of elements make sense.

The vector space lacks the notation of a scalar quantity that can be geometrically interpreted as an angle between two vectors. Therefore, an inner product space is a vector space with an extra structure, the inner product, such that the notion of angles to make sense.

Definition 2.2 (Inner Product)**.** Let X be a vector space, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in X$. An inner product is any function $\langle \cdot, \cdot \rangle : X \times X \to \mathbb{R}$ that satisfies the following conditions.

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle \qquad \text{(symmetry)} \tag{2.8}$$
$$\langle \lambda\mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \lambda \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle \qquad \text{(linearity in the first argument)} \tag{2.9}$$
$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0 \text{ with } \langle \mathbf{x}, \mathbf{x} \rangle = 0 \Leftrightarrow \mathbf{x} = 0 \qquad \text{(positive-definiteness)} \tag{2.10}$$

There are many contexts where we want to measure how close two mathematical objects are. What is a reasonable distance? It should be always greater or equal than 0 and the distance should from a point $\mathbf{x}$ to another point $\mathbf{y}$ be the same as from $\mathbf{y}$ to $\mathbf{x}$.

Lastly, we want to have that the direct distance between two points is less or equal than the distance via an intermediate point. This yields the following definition.

Definition 2.3 (Metric)**.** Let X be a vector space and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in X$. A function $d : X \times X \to \mathbb{R}$ is a metric iff

$$d(\mathbf{x}, \mathbf{y}) \geq 0 \text{ with } d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y} \qquad \text{(non-negativity)} \tag{2.11}$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \qquad \text{(symmetry)} \tag{2.12}$$

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \qquad \text{(triangle inequality)} \tag{2.13}$$

If a set X is combined with a metric d it is called a metric space (X, d).

If a metric is translation invariant, i.e. $d(\mathbf{x} + \mathbf{u}, \mathbf{y} + \mathbf{u}) = d(\mathbf{x}, \mathbf{y})$, then by setting $\mathbf{x} = -\mathbf{u}$ it follows that $d(\mathbf{x}, \mathbf{y}) = d(0, \mathbf{y} - \mathbf{x})$. This implies if we know the distance from 0, we know all distances. If we further assume that $d(0, \lambda\mathbf{x}) = |\lambda|\, d(0, \mathbf{x})$, then we can write $\|\mathbf{x}\| = d(0, \mathbf{x})$ and call $\|\cdot\|$ a norm.

Definition 2.4 (Norm)**.** Let X be a vector space and $\mathbf{x}, \mathbf{y} \in X$. We call $\|\cdot\|$ a norm iff

$$\|\mathbf{x}\| \geq 0 \text{ with } \|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = 0 \qquad \text{(non-negativity)} \tag{2.14}$$

$$\|\lambda\mathbf{x}\| = |\lambda|\, \|\mathbf{x}\| \qquad \text{(absolute scalability)} \tag{2.15}$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \qquad \text{(triangle inequality)} \tag{2.16}$$

A vector space equipped with a norm defined on it is called a normed space.

Any function that fulfills the properties above is called a norm. A fundamental class of norms is the ℓ_p-norms defined as

$$\ell_p(\mathbf{x}) = \left(\sum_{i=1}^{N} |x_i|^p \right)^{\frac{1}{p}} \text{ with } p \geq 1\,. \tag{2.17}$$

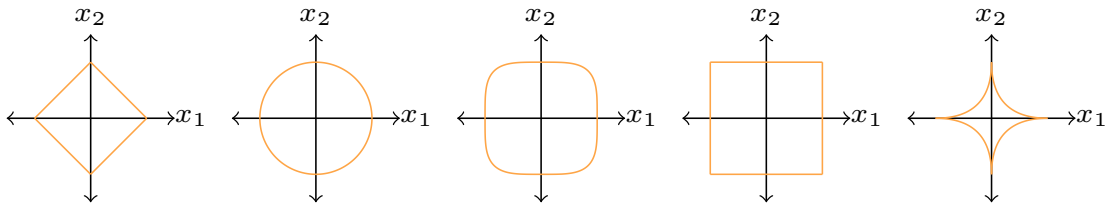
The ℓ_p-norm includes the Euclidean norm (ℓ_2-norm) and Manhatten norm (ℓ_1-norm) as special cases, whereas the Euclidean norm corresponds to our natural intuition about distances. An additional special case is the maximum norm

$$\ell_\infty(\mathbf{x}) = \lim_{p \to \infty} \ell_p(\mathbf{x}) = \max\left\{ |x_1|, \cdots, |x_N| \right\}\,, \tag{2.18}$$

which is obtained for the limit $p \to \infty$. Further, the p-norms satisfy the following relationship

$$\ell_{p_2}(\mathbf{x}) \leq \ell_{p_1}(\mathbf{x}) \text{ for } p_1 < p_2\,, \tag{2.19}$$

which means that for example the values of the ℓ_2 norm are bounded by the ℓ_1 norm. In Figure 2.1 we show the unit norm balls of different p-norms in $\mathbb{R}^2$.



(a) $\ell_1(\mathbf{x}) = 1$ **(b)** $\ell_2(\mathbf{x}) = 1$ **(c)** $\ell_4(\mathbf{x}) = 1$ **(d)** $\ell_\infty(\mathbf{x}) = 1$ **(e)** $\ell_{0.5}(\mathbf{x}) = 1$

Figure 2.1: The unit norm ball for different p-norms in $\mathbb{R}^2$. Note that $\ell_{0.5}(\mathbf{x})$ depicted in (e) defines no valid norm.

Another important norm from the field of robust statistics is the so-called Huber norm [115]. It is defined as the combination of an ℓ_1 and a squared ℓ_2 norm and denoted as $\|\cdot\|_\epsilon$

$$\|\mathbf{x}\|_\epsilon = \sum_{i=1}^{N} \begin{cases} \frac{x_i^2}{2\epsilon} & \text{if } |x_i| \leq \epsilon \\ |x_i| - \frac{\epsilon}{2} & \text{else} \end{cases} . \tag{2.20}$$

The parameter $\epsilon \in \mathbb{R}_{\geq 0}$ controls the approximation accuracy to the ℓ_1-norm and the smoothness of the norm. We observe that $|x| - \frac{\epsilon}{2} \leq \|x\|_\epsilon \leq |x|$. Hence, we derive the approximation accuracy $\lim_{\epsilon \leftarrow 0} \|x\|_\epsilon = \|x\|_1$. To show the smoothness we can simply bound the second derivative, i.e. the curvature, with $\frac{\mathrm{d}^2}{\mathrm{d}x^2} \|x\|_\epsilon \leq \frac{1}{\epsilon}$.

Later in this chapter, we will often encounter quantities which are given as vectors per element, e.g. a vector per image pixel. For those quantities, we will use an inner and an outer norm. The former is applied to each component vector and the latter to the resulting vector. For example, let $\mathbf{x} \in \mathbb{R}^{C \times N}$ with C being the vector length per component then

$$\|\mathbf{x}\|_{p,q} = \left\| \begin{pmatrix} \|\mathbf{x}_1\|_p \\ \vdots \\ \|\mathbf{x}_C\|_p \end{pmatrix} \right\|_q , \tag{2.21}$$

where $\|\cdot\|_p$ is the inner norm, and $\|\cdot\|_q$ is the outer norm. If not otherwise stated, we will always assume an ℓ_2 norm as the inner norm. Then, for the example above, the ℓ_1 norm is given by

$$\|\mathbf{x}\|_{2,1} = \sum_{i=1}^{N} \sqrt{\sum_{c=1}^{C} |x_{c,i}|^2} . \tag{2.22}$$

A last important definition regarding norms is the dual norm which arises frequently in convex analysis and optimization.

Definition 2.5. Let $\|\cdot\|$ be a norm, then its dual norm $\|x\|_*$ is defined as

$$\|\mathbf{x}\|_* = \sup_{\|\mathbf{z}\|\leq 1} \mathbf{z}^T\mathbf{x}\,. \tag{2.23}$$

For p-norms one can show that the following relationship holds

$$\ell_{p*}(\mathbf{x}) = \ell_q(\mathbf{x}) \text{ with } \frac{1}{p} + \frac{1}{q} = 1\,. \tag{2.24}$$

This implies that the dual norm of the ℓ_2-norm is the ℓ_2-norm itself and the dual norm of the ℓ_1-norm is the maximum norm ℓ_∞, and vice versa.

One important concept that is heavily used throughout this thesis is the one of differentiation. Given a function $f : \mathbb{R} \to \mathbb{R}$ and a point x, one wants to find the best linear approximation of f at a in a certain range h. Let $g(x) = f(a) + m(x-a)$ be the linear approximation. By using the substitute $x = a+h$, we can rewrite this as $g(a+h) = f(a)+mh$. Then, the goal is to derive an m such that the error $\epsilon(a) = f(a+h) - f(a) - mh$ is small compared to h. This gives rise to the following definition of the derivative operator.

Definition 2.6 (Differential Operator)**.** Let f be a function $f : \mathbb{R} \to \mathbb{R}$. We call f differentiable at a if the following limit is satisfied

$$\lim_{h\to 0} \frac{f(a+h) - f(a) - Df(a)h}{h} = 0\,. \tag{2.25}$$

We call D the differential operator. This is equivalent to

$$Df(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h}\,. \tag{2.26}$$

Sometimes it is also convenient to write the differential operator in Leibniz's notation $Df(x) \equiv \frac{\mathrm{d}f}{\mathrm{d}x}$. The definition of the differential operator can also be extended to multivariate functions $f : \mathbb{R}^N \to \mathbb{R}$ by introducing the gradient $\nabla f(\mathbf{x}) : \mathbb{R} \to \mathbb{R}^N$ operator

$$Df(\mathbf{x}) = \nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_N}\right)^T\,. \tag{2.27}$$

We further define the divergence operator $\operatorname{div}\mathbf{g}(\mathbf{x}) : \mathbb{R}^N \to \mathbb{R}$ as adjoint to the gradient operator. In particular, we have $-\operatorname{div} = \nabla^T$ defined by the following equation

$$\langle \nabla f(\mathbf{x}), \mathbf{p}\rangle = -\langle f(\mathbf{x}), \operatorname{div}\mathbf{p}\rangle\,, \tag{2.28}$$

with $\mathbf{p} \in \mathbb{R}^N$. Next, given a general function $\mathbf{f} : \mathbb{R}^N \to \mathbb{R}^M$, we can define the Jacobian

matrix

$$D\mathbf{f}(\mathbf{x}) = \mathbf{J}(\mathbf{f}(\mathbf{x})) = \begin{pmatrix} (\nabla f_1(\mathbf{x}))^T \\ \vdots \\ (\nabla f_M(\mathbf{x}))^T \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{pmatrix} . \tag{2.29}$$

Finally, the differential operator can be applied multiple times, especially, given a function $f : \mathbb{R}^N \to \mathbb{R}$, the result of applying the differential operator twice is called the Hessian $\mathbf{H} \in \mathbb{R}^{N \times N}$ and defined as

$$D^2 f(\mathbf{x}) = \mathbf{H}(f(\mathbf{x})) = \nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{pmatrix} . \tag{2.30}$$

The last fundamental definition is the one of an image itself and operations on it. An image is usually defined on a regular Cartesian pixel grid Ω of size $H \times W$

$$\Omega = \left\{ (x, y) \in \mathbb{N}^2 | 1 \leq x \leq W, 1 \leq y \leq H \right\} . \tag{2.31}$$

Then we can define the image $I((h, w))$ as a two-dimensional scalar field on Ω, i.e. the following map

$$I((x, y)) : \Omega \to \mathbb{R} . \tag{2.32}$$

Using the gradient definition from above, the gradient operator on the image space becomes

$$\nabla I : \Omega \to \mathbb{R}^{2 \times W \times H} \tag{2.33}$$

$$\nabla I((x, y)) = \begin{pmatrix} \frac{\partial}{\partial x} I((x, y)) \\ \frac{\partial}{\partial y} I((x, y)) \end{pmatrix} = \begin{pmatrix} \nabla_x I((x, y)) \\ \nabla_y I((x, y)) \end{pmatrix} , \tag{2.34}$$

which defines the gradient in x- and y-direction of the Cartesian pixel grid. In the discrete setting of the image space, the gradient operators ∇_x and ∇_y are approximated by forward-differences with Neumann boundary conditions as follows

$$\nabla_x I((x, y)) = \begin{cases} I((x+1, y)) - I(x, y) & \text{if } x < W \\ 0 & \text{if } x = W \end{cases} \tag{2.35}$$

$$\nabla_y I((x, y)) = \begin{cases} I((x, y+1)) - I(x, y) & \text{if } y < H \\ 0 & \text{if } y = H \end{cases} . \tag{2.36}$$

Similarly, the discrete adjoint gradient operator ∇^T can be approximated with nega-

tive backward differences using Dirichlet boundary conditions. Given that $\mathbf{P}((x,y)) = \nabla I((x,y)) \in \mathbb{R}^{2\times W\times H}$, then we have

$$\nabla^T \mathbf{P}((x,y)) = \nabla_x^T \mathbf{P}_x((x,y)) + \nabla_y^T \mathbf{P}_y((x,y)) \tag{2.37}$$

$$\nabla_x^T \mathbf{P}_x((x,y)) = \begin{cases} \mathbf{P}_x(x,y) & \text{if } x = 1 \\ \mathbf{P}_x(x,y) - \mathbf{P}_x(x-1,y) & \text{if } 1 < x < W \\ -\mathbf{P}_x(x-1,y) & \text{if } x = W \end{cases} \tag{2.38}$$

$$\nabla_y^T \mathbf{P}_y((x,y)) = \begin{cases} \mathbf{P}_y(x,y) & \text{if } y = 1 \\ \mathbf{P}_y(x,y) - \mathbf{P}_y(x,y-1) & \text{if } 1 < y < H \\ -\mathbf{P}_y(x,y-1) & \text{if } y = H \end{cases} . \tag{2.39}$$

2.2 Convex Optimization

This section is intended to give a rough overview of the broad field of convex analysis and optimization. It builds the foundation to efficiently solve optimization problems that arise in variational methods for image processing, a topic that we will present in the next section, but convex optimization is also widely adopted in other areas. It provides strong bounds on the number of iterations, the quality of the solutions, and they are very efficient in praxis. The more interested reader is referred to the excellent works dedicated to this topic [17, 164, 192].

2.2.1 Convex Analysis

Convex analysis is a specific branch of mathematics that deals with convex sets and convex functions. It forms the theoretical basis of convex optimization problems and is therefore of significant importance to introduce those basic concepts.

2.2.1.1 Convex Sets

Given two points $\mathbf{x}_1, \mathbf{x}_2 \in X$ in a vector space, e.g. $X = \mathbb{R}^N$, a line is simply defined by a linear combination of those two points

$$\mathbf{y} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \,, \tag{2.40}$$

with $\lambda \in \mathbb{R}$. Further, the line segment is the subset of points $\mathbf{y}$ for which $\lambda \in [0,1]$.

With the notion of a line segment, we can define the convex set.

Definition 2.7 (Convex Set)**.** Let X be a vector space. X is called a convex set, iff for any two vectors $\mathbf{x}_1, \mathbf{x}_2 \in X$ and $\lambda \in [0,1]$ the following relation holds

$$\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in X \,. \tag{2.41}$$

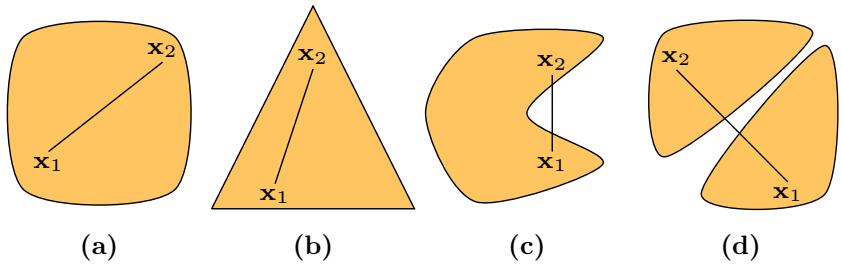


Figure 2.2: Examples of convex sets (a), (b) and non-convex sets (c), (d), respectively.

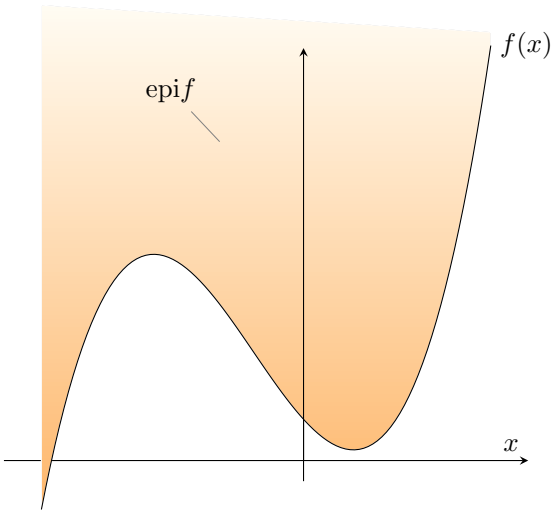


Figure 2.3: The epigraph of a function $f(x)$ is the set of points that lie on, or above its graph.

The definition has a nice geometrically interpretation: X is convex, if and only if all points on the line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$ are also in X, or in other words, if you can see from one vector $\mathbf{x}_1$ all other vectors of the set. See Figure 2.2 for some visual examples. Further, we note that the empty set $\varnothing$, as well as the real numbers of any dimension $\mathbb{R}^N$ are convex sets, as is any given line segment in $\mathbb{R}^N$. Another important subclass of convex sets are cones. X is called a convex cone, iff it is a convex set and for any $\mathbf{x}_1, \mathbf{x}_2 \in X$, $\lambda_1, \lambda_2 \in \mathbb{R}_{\geq 0}$ the following relation holds

$$\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 \in X \,. \tag{2.42}$$

There exists also several operators that preserve the convexity of sets. One such operation is the intersection of two convex sets. If X_1 and X_2 are convex sets, then also the intersection of both $X_1 \cap X_2$ is a convex set. Proof in Appendix B.1. Another operation that preserves the convexity of a set is an affine transformation $\mathbf{A}\mathbf{x}+\mathbf{b}$, where $\mathbf{A}$ is a linear transformation in X and $\mathbf{x}, \mathbf{b} \in X$. Proof in Appendix B.2. From this, it is easy to show that the Minkowski sum of two convex sets $X_1 + X_2 = \{\mathbf{x}_1 + \mathbf{x}_2 | \mathbf{x}_1 \in X_1, \mathbf{x}_2 \in X_2\}$ is also a convex set by using the affine transformation $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right), \mathbf{b} = 0$.

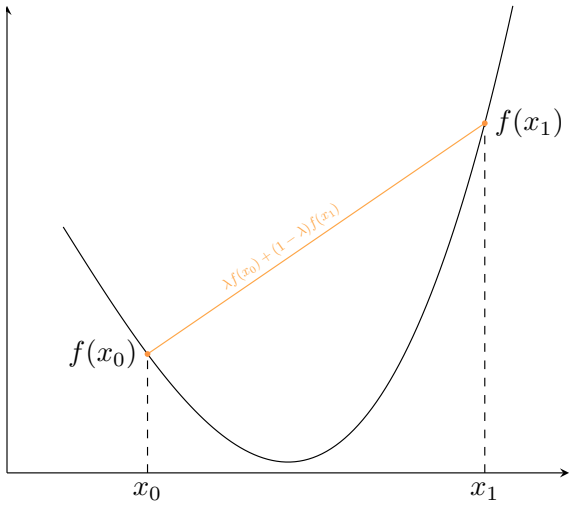


Figure 2.4: For every convex function any line segment from $(x_0, f(x_0))$ to $(x_1, f(x_1))$ has to be on, or above the graph of $f(x)$.

2.2.1.2 Convex Functions

As a primer to this section, we will first introduce the epigraph of a function. A visual example is depicted in Figure 2.3.

Definition 2.8 (Epigraph)**.** Given a function $f : \mathbb{R}^N \to \mathbb{R}$, we call the set of points that lie on, or above its graph the epigraph of f

$$\operatorname{epi} f = \left\{(\mathbf{x}, x_{N+1}) | \mathbf{x} \in \mathbb{R}^N, x_{N+1} \in \mathbb{R}, x_{N+1} \geq f(\mathbf{x})\right\} . \tag{2.43}$$

This brings us to the most central definition of the section.

Definition 2.9 (Convex Function)**.** Given a convex set X, we call a function $f : X \to \mathbb{R}$ convex, if the following relation holds for every $\mathbf{x}_1, \mathbf{x}_2 \in X$ and $\lambda \in [0, 1]$

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) . \tag{2.44}$$

This inequality extends to arbitrary sums with $\lambda_1, \cdots, \lambda_N \geq 0$ and $\sum_{i=1}^N \lambda_i = 1$ and is also called Jensen's inequality in other contexts. We call f strictly convex iff

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) , \tag{2.45}$$

and a function f is called (strictly) concave, iff $-f$ is (strictly) convex.

One of the fundamental properties of convex functions is that every minimizer is a global minimizer. Proof in Appendix B.3.

The definition of convex functions has also a nice geometrical interpretation. It implies that every line segment from $(\mathbf{x}_1, f(\mathbf{x}_1))$ to $(\mathbf{x}_2, f(\mathbf{x}_2))$ lies on, or above the graph of f. See

Figure 2.4 for a visual example of this interpretation. The epigraph of f establishes the link between convex functions and convex sets. A function f is only convex, iff epi f is a convex set, and vice versa. Note, that it is common in convex optimization to extend convex functions to cover all of $\mathbb{R}^N$, if the dom $f \subset \mathbb{R}^N$ by setting the function value to ∞ outside of dom f. Hence, we usually consider the extended-value extension $\tilde{f} : \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ of f if not otherwise noted

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \in \text{dom } f \\ \infty & \text{else} \end{cases} . \tag{2.46}$$

One interesting property of convex functions is, that the first-order approximation of f is a global under-estimator of the function. We can prove, that for differentiable convex functions the following relation is satisfied

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle . \tag{2.47}$$

Proof in Appendix B.4.

From this equivalent definition of convex functions, we can conclude that the gradient of convex functions is monotone, i.e. for scalar functions is non-decreasing. Proof in Appendix B.5. Further, we can show from this definition of convexity that $\nabla f(\mathbf{x}^*) = 0$ is a necessary and sufficient condition for $\mathbf{x}$ to be a global optimizer. Proof in Appendix B.6.

There exists also a second-order condition for convex function. It can be shown by using the Taylor expansion of a twice-differentiable function $f : X \to \mathbb{R}$ that f is convex and only if

$$\nabla^2 f(\mathbf{x}) \succeq 0 , \tag{2.48}$$

which means that the Hessian matrix $\mathbf{H}$ of $f(\mathbf{x})$ has to be positive semi-definite, i.e. $\forall \mathbf{x} : \mathbf{x}^T \mathbf{H} \mathbf{x} \geq 0$.

Equivalently to convex sets, there also exist certain operations for convex functions that preserve convexity. A trivial operation is the weighted sum of convex functions. If f_1, and f_2 are convex functions and $w_1, w_2 > 0$, then $f = w_1 f + w_2 f$ is also convex. Proof in Appendix B.7. The affine transformation of the argument $g(\mathbf{x}) = f(\mathbf{A}\mathbf{x} + \mathbf{b})$ is another operation under which convexity is kept, if f is convex. This is simply given by the fact that the affine transformation of dom f is again a convex set, cf. previous section on convex sets. Similarly to the weighted sum of convex functions, also the pointwise maximum of convex functions is again convex. Hence, if f_1 and f_2 are convex, then $f(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$ is also a convex function. Proof in Appendix B.8.

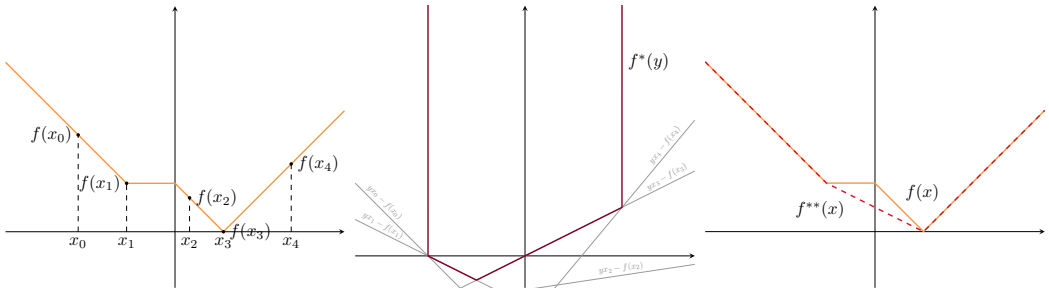


Figure 2.5: In (a) we show a non-convex function $f(x)$ and (b) shows the corresponding convex conjugate $f^*(y)$. We show some affine functions $xy - f(x)$ to illustrate the point-wise supremum. In (c) we show the function $f(x)$ with its bi-conjugate $f^{**}(x)$ which is the convex envelope of the function.

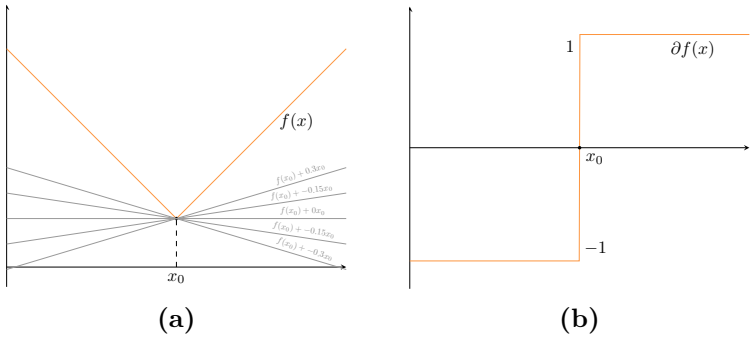


Figure 2.6: In (a) we visualize the function $f(x) = |x|$ and some subgradients at $x_0 = 0$. In (b) we depict the subgradient $\partial f(x)$ over the full domain.

2.2.1.3 Convex Conjugate

The definition of the convex conjugate, also called Legendre-Fenchel transform for differentiable functions, allows characterizing functions by their slopes and intersections.

Definition 2.10 (Convex Conjugate)**.** For a function $f : X \to \mathbb{R}$ the convex conjugate is defined as

$$f^*(\mathbf{y}) = \sup_{\mathbf{x} \in \operatorname{dom} f} \{\langle \mathbf{x}, \mathbf{y} \rangle - f(\mathbf{x})\} \ . \tag{2.49}$$

As the name suggests, the convex conjugate of an arbitrary function is convex. This can be easily observed, as it is the point-wise supremum of a set of affine, hence convex functions in $\mathbf{y}$. We can further define the bi-conjugate of a function.

Definition 2.11 (Bi-Conjugate)**.** For a function $f : X \to \mathbb{R}$ the bi-conjugate is defined as

$$f^{**}(\mathbf{x}) = \sup_{\mathbf{y} \in \operatorname{dom} f^*} \{\langle \mathbf{y}, \mathbf{x} \rangle - f^*(\mathbf{y})\} \ . \tag{2.50}$$

Note, that the bi-conjugate $f^{**}(\mathbf{x})$ is the convex envelope of $f(\mathbf{x})$. Hence, we have $f^{**}(\mathbf{x}) \leq f(\mathbf{x})$ for any function $f(\mathbf{x})$. Further, if $f(\mathbf{x})$ is convex and closed (epi f is a closed set), then the identity $f(\mathbf{x}) = f^{**}(\mathbf{x})$ holds, i.e. the bi-conjugate is the function itself again. A visual example of a convex conjugate and bi-conjugate is given in Figure 2.5. From the definition of the convex conjugate, we can further deduce the Fenchel inequality (also called Fenchel-Young inequality for differentiable functions)

$$f(\mathbf{x}) + f^*(\mathbf{y}) \geq \langle \mathbf{x}, \mathbf{y} \rangle \ . \tag{2.51}$$

An important example is the conjugate of a norm $f(\mathbf{x}) = \|\mathbf{x}\|$, which is given by the indicator function of the dual norm unit ball

$$f^*(\mathbf{y}) = \begin{cases} 0 & \text{if } \|\mathbf{y}\|_* \leq 1 \\ +\infty & \text{else} \end{cases} \ . \tag{2.52}$$

Proof in Appendix B.9.

2.2.1.4 Subgradient and Subdifferential

We have shown earlier in this section in Equation (B.9) that the first-order approximation of a convex function f is a global under-estimator of the function. This is a very useful property if we want to optimize a convex function, but it requires the function to be differentiable on the whole domain. However, we can generalize the notation of the differential operator using Equation (B.9) to handle non-smooth functions.

Definition 2.12 (Subgradient and Subdifferential)**.** Let f be a convex function. We call a vector $\mathbf{g}$ the subgradient of f at $\mathbf{x}_0$ if

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \langle \mathbf{g}, \mathbf{x} - \mathbf{x}_0 \rangle \ , \tag{2.53}$$

holds for any $\mathbf{x} \in \text{dom}\, f$. Further, the set of all subgradients of f at $\mathbf{x}_0$, $\partial f(\mathbf{x}_0)$ is called the subdifferential of the function f at $\mathbf{x}_0$. Note that if f is a smooth function, then the subgradient of f contains only one element, which is the gradient of f itself, i.e. $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$. From the definition it also directly follows that $\mathbf{x}^*$ is a minimum of f, iff $0 \in \partial f(\mathbf{x})$.

The canonical example to show the subdifferential is the absolute value function $f(x) = |x|$. It is trivial to show that f is convex and the subdifferential is given by

$$\partial f(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x > 0 \\ [-1, 1] & \text{if } x = 0 \end{cases} \ . \tag{2.54}$$

A visual illustration is given in Figure 2.6. It also shows the geometrical interpretation of the subgradient: The graph of the affine function $h(\mathbf{g}) = f(\mathbf{x}_0)+\langle \mathbf{g}, \mathbf{x}-\mathbf{x}_0\rangle$ is a supporting hyperplane to the convex set epi f at $(\mathbf{x}_0, f(\mathbf{x}_0))$.

There exists a close relationship between conjugate functions and the subdifferential. Given that $\mathbf{g} \in \partial f(\mathbf{x}_0)$ we have

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \langle \mathbf{g}, \mathbf{x}-\mathbf{x}_0\rangle \, f(\mathbf{x}_0) + \langle \mathbf{g}, \mathbf{x}\rangle \leq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{x}_0\rangle \ . \tag{2.55}$$

Hence, we can derive the convex conjugate as

$$f^*(\mathbf{g}) = \sup_{\mathbf{x}} \{\langle \mathbf{g}, \mathbf{x}\rangle - f(\mathbf{x})\} = \langle \mathbf{g}, \mathbf{x}_0\rangle - f(\mathbf{x}_0)\,. \tag{2.56}$$

Finally, by a simple transformation it follows that $f^*(\mathbf{g}) + f(\mathbf{x}_0) = \langle \mathbf{g}, \mathbf{x}_0\rangle$.

2.2.1.5 Proximal Mapping

A final important concept in convex optimization is the proximal mapping, also known as resolvent, or proximity operator. It is a main building block in proximal algorithms to efficiently solve convex optimization problems as shown in the next section.

Definition 2.13 (Proximal Mapping)**.** Let $f : \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ be a closed proper convex function. Hence, epi f is a closed convex set. The proximal operator $\text{prox}_{\lambda f} : \mathbb{R}^N \to \mathbb{R}^N$ of f is given by

$$\text{prox}_{\lambda f}(\mathbf{y}) = \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}) + \frac{1}{2\lambda} \|\mathbf{x}-\mathbf{y}\|_2^2 \right\} . \tag{2.57}$$

Note that $f(\mathbf{x}) + \frac{1}{2\lambda}\|\mathbf{x}-\mathbf{y}\|_2^2$ is a strongly convex function and further, not everywhere infinite. This implies that it has a unique minimizer $\mathbf{x}^*$ for every $\mathbf{y}$. We can express the proximal mapping also in terms of the optimality condition

$$0 \in \partial f(\mathbf{x}) + \mathbf{x} - \mathbf{y} \tag{2.58}$$

$$\mathbf{y} \in (\mathbf{I} + \partial f)\mathbf{x} \tag{2.59}$$

$$\mathbf{x} = (\mathbf{I} + \partial f)^{-1}(\mathbf{y}) = \text{prox}_f(\mathbf{y})\,. \tag{2.60}$$

An important property of the proximal mapping is that $\mathbf{x}^*$ is the minimizer of f, iff $\mathbf{x}$ is a fixed point of prox_f

$$\mathbf{x}^* = \text{prox}_f(\mathbf{x}^*)\,. \tag{2.61}$$

Proof in Appendix B.10. Another property that is heavily used in convex optimization for images is the separability of the proximal mapping. If $f : \mathbb{R}^N \to \mathbb{R}$ is a separable function,

i.e. $f(\mathbf{x}) = \sum_i f_i(x_i)$, then we have that

$$\operatorname{prox}_f(\mathbf{y})_i = \operatorname{prox}_{f_i}(y_i)\,. \tag{2.62}$$

The proximal mapping is also related to the convex conjugate by the Moreau decomposition that states

$$\mathbf{x} = \operatorname{prox}_f(\mathbf{x}) + \operatorname{prox}_f^*(\mathbf{x})\,. \tag{2.63}$$

Proof in Appendix B.11.

Finally, we give an overview of some relevant proximal operators

- If f is an indicator I_C function of a convex set X

$$I_C(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in X \\ \infty & \text{else} \end{cases}\,, \tag{2.64}$$

  then the proximal mapping is given by the orthogonal projection $\operatorname{proj}_X(\mathbf{y})$ onto X

$$\operatorname{prox}_f(\mathbf{y}) = \operatorname{proj}_X(\mathbf{y}) = \underset{\mathbf{x}\in X}{\arg\min} \left\{ \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|_2^2 \right\}\,. \tag{2.65}$$

- If f is the indicator function of the maximum-norm unit ball $I_{\|\cdot\|_\infty \leq 1}$, then the proximal mapping is given by the following point-wise projection

$$\operatorname{prox}_f(x)_i = \frac{x_i}{\max(1, |x_i|)}\,. \tag{2.66}$$

- If f is the ℓ_1-norm $\|\mathbf{x}\|_1$, then the proximal operator is given by the soft-shrinkage function

$$\operatorname{prox}_{\lambda f}(\mathbf{y})_i = \operatorname{sgn}(x_i) \max(0, (|x_i| - \lambda))\,. \tag{2.67}$$

2.2.2 Convex Optimization Algorithms

In the last few sections, we introduced the basics of convex analysis. Those definitions will be used in this section, where we are concerned with the problem of the following canonical form

$$\min_{\mathbf{x}\in \operatorname{dom} f} f(\mathbf{x})\,, \tag{2.68}$$

where f is a convex function, i.e. the goal is to minimize convex functions. As discussed previously, convex functions have several attractive properties. Most importantly, every local minimum is a global minimum. This property can be exploited to built highly

Algorithm 1 Gradient Descent

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$
1: **while** not converged **do**
2: $\quad \mathbf{x}^{(\mathrm{i}+1)} = \mathbf{x}^{(\mathrm{i})} - \eta^{(\mathrm{i})} \nabla f(\mathbf{x}^{(\mathrm{i})})$
3: **end while**

efficient optimization algorithms. In fact, for the class of optimization problems that we consider in this thesis, we can state bounds for the convergence rates and demonstrate optimal algorithms. Assume that f is a convex function, $\mathbf{x}^*$ is the optimum for f, and i is the iteration index, then we define the convergence rate $\mathcal{O}\left(\frac{1}{g(i)}\right)$ as

$$f(\mathbf{x}_i) - f(\mathbf{x}^*) \leq \frac{c}{g(i)} \in \mathcal{O}\left(\frac{1}{g(i)}\right) , \tag{2.69}$$

where g is a monotonic increasing function on i, and c is a positive constant. Hence, the convergence rate defines how fast the error decreases with respect to the number of iterations.

Another important definition to derive optimal optimization algorithms is the notion of Lipschitz-continuous functions. A function f is said to be Lipschitz continuous with parameter $L > 0$, iff

$$\forall \mathbf{x}, \mathbf{y} \in \operatorname{dom} f : \|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq L \|\mathbf{x} - \mathbf{y}\|_2 . \tag{2.70}$$

The interpretation is that the distance between two points is at maximum only L-times larger than the difference of the function values at those points. This notation generalizes to gradients. Hence, the gradient of f is Lipschitz continuous with parameter $L > 0$, iff

$$\forall \mathbf{x}, \mathbf{y} \in \operatorname{dom} f : \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq L \|\mathbf{x} - \mathbf{y}\|_2 . \tag{2.71}$$

In the following, we will show first-order optimization methods for two classes of convex functions. First, for smooth convex functions, i.e. functions for which $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$ for all $\mathbf{x} \in \operatorname{dom} f$. In the second part we will discuss efficient optimization algorithms for non-smooth convex functions.

2.2.2.1 Smooth Convex Optimization

In this section we present two standard optimization algorithms for the problem in Equation (2.68) with f being a smooth convex function. It can be shown that if f has an L-Lipschitz gradient, then the optimal convergence rate that can be obtained by any optimization algorithm is $\mathcal{O}\left(\frac{1}{i^2}\right)$ [164].

The simplest algorithm for this class of functions is gradient decent as stated in Algorithm 1. Gradient descent minimizes the function f strictly to the optimality condition

Algorithm 2 Nesterov's Accelerated Gradient Descent

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$

1: $t^{(0)} = 1, \mathbf{y}^{(1)} = \mathbf{x}^{(0)}$
2: **while** not converged **do**
3: $\quad \mathbf{x}^{(\mathrm{i})} = \mathbf{y}^{(\mathrm{i})} - \frac{1}{L}\nabla f(\mathbf{y}^{(\mathrm{i})})$
4: $\quad t^{(\mathrm{i}+1)} = \frac{1+\sqrt{1+4t^{(\mathrm{i})2}}}{2}$
5: $\quad \mathbf{y}^{(\mathrm{i}+1)} = \mathbf{x}^{(\mathrm{i}+1)} + \frac{t^{(\mathrm{i})}-1}{t^{(\mathrm{i}+1)}}(\mathbf{x}^{(\mathrm{i})} - \mathbf{x}^{(\mathrm{i}-1)})$
6: **end while**

$\nabla f(\mathbf{x}) = 0$. An important choice for this algorithm is the step size $\eta^{(\mathrm{i})}$. It can be shown that gradient descent converges for a constant step size $\eta^{(\mathrm{i})} \in (0, \frac{2}{L})$ for all i with $\mathcal{O}\left(\frac{1}{i}\right)$. This is obviously worse than the theoretical optimal convergence rate.

An optimal algorithm in terms of convergence rate has been proposed by Nesterov [163]. See Algorithm 2 for the scheme. The algorithm stores a moving sum of all previous gradients and uses it for an extrapolation step. This doubles the memory requirements of the algorithm, which is in practice seldom problematic. However, the convergence rate can be shown to be optimal. An important note is that the algorithm does not monotonically decrease the function value f in contrast to gradient descent. Hence, it shows periodic phases of increase and decrease in f.

2.2.2.2 Non-Smooth Convex Optimization

We can even consider a broader class of convex functions of the minimization problem in Equation (2.68) if we allow f to be non-smooth, i.e. there exists at least one point $\mathbf{x} \in \operatorname{dom} f$ for which $|\partial f(\mathbf{x})| > 1$. The best general convergence rate that can be obtained on this class of functions is $\mathcal{O}(\frac{1}{\sqrt{i}})$ [164]. A simple modification of gradient descent yields the sub-gradient descent scheme as listed in Algorithm 3, where the gradient computation is replaced with the computation of the sub-gradient. Interestingly, this method can be shown to be already optimal in terms of the convergence rate [164].

The question is then if we can do better if we assume any additional structure of the problem. In fact, many problems in computer vision can be modeled as a sum of two convex functions $f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$, such that the optimization problem of Equation (2.68) becomes

$$\min_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x}) \equiv g(\mathbf{x}) + h(\mathbf{x})\,. \tag{2.72}$$

If we now assume that h has an L-Lipschitz continuous gradient and h is possible non-smooth. Further, if we can smooth h by utilizing the proximal operator, then this gives rise to the proximal gradient descent algorithm as listed in Algorithm 4. Using the same time steps as argued for gradient descent, it can be shown to obtain also the same convergence rate, i.e. $\mathcal{O}\left(\frac{1}{i}\right)$, if the proximal operator of h is inexpensive to compute.

Algorithm 3 Sub-Gradient Descent

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$

while not converged **do**
  $\mathbf{g}^{(\mathrm{i})} \in \partial f(\mathbf{x}^{(\mathrm{i})})$
  $\mathbf{x}^{(\mathrm{i}+1)} = \mathbf{x}^{(\mathrm{i})} - \eta^{(\mathrm{i})}\mathbf{g}^{(\mathrm{i})}$
end while

Algorithm 4 Proximal Gradient Decent

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$

while not converged **do**
  $\mathbf{x}^{(\mathrm{i}+1)} = \operatorname{prox}_{\eta^{(\mathrm{i})}h}(\mathbf{x}^{(\mathrm{i})} - \eta^{(\mathrm{i})}\nabla g(\mathbf{x}^{(\mathrm{i})}))$
end while

Algorithm 5 Fast Iterative Shrinkage Thresholding Algorithm (FISTA)

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$

$t^{(0)} = 1, \mathbf{y}^{(1)} = \mathbf{x}^{(0)}$
while not converged **do**
  $\mathbf{x}^{(\mathrm{i})} = \operatorname{prox}_{\frac{1}{L}g}(\mathbf{y}^{(\mathrm{i})} - \frac{1}{L}\nabla h(\mathbf{y}^{(\mathrm{i})}))$
  $t^{(\mathrm{i}+1)} = \frac{1+\sqrt{1+4t^{(\mathrm{i})2}}}{2}$
  $\mathbf{y}^{(\mathrm{i}+1)} = \mathbf{x}^{(\mathrm{i})} + \frac{t^{(\mathrm{i})}-1}{t^{(\mathrm{i}+1)}}(\mathbf{x}^{(\mathrm{i})} - \mathbf{x}^{(\mathrm{i}-1)})$
end while

Similarly, the method can be accelerated by keeping a sum of update directions. The resulting method is the fast iterative shrinkage thresholding algorithm (FISTA) [10] and is listed in Algorithm 5. The authors show that the convergence rate of this algorithm is $\mathcal{O}(\frac{1}{i^2})$. This is the same rate as the optimal rate for smooth convex optimization problems and therefore also optimal for this class of problems.

To motivate the last optimization algorithm let us consider a more general optimization problem, which fits many imaging applications

$$\min_{\mathbf{x}\in\operatorname{dom} f} g(\mathbf{K}\mathbf{x}) + h(\mathbf{x})\,, \tag{2.73}$$

where g and h are proper, convex and lower-semicontinuous and $\mathbf{K}$ is a linear operator, e.g. $\mathbf{K} \in \mathbb{R}^{M\times N}$. We explicitly do not assume that g, or h are smooth. If we now replace $g(\mathbf{K}\mathbf{x})$ with its bi-conjugate, which is as shown before equivalent for convex function, we yield the following convex-concave saddle-point problem

$$\min_{\mathbf{x}\in\operatorname{dom} h}\max_{\mathbf{y}\in\operatorname{dom} g^*} \langle \mathbf{K}\mathbf{x}, \mathbf{y}\rangle + h(\mathbf{x}) - g^*(\mathbf{y})\,. \tag{2.74}$$

The above problem is also called the primal-dual formulation. The primal problem of

Algorithm 6 Primal-Dual Algorithm

Given an initial point $\mathbf{x}^{(0)} \in \operatorname{dom} f$
1: $\bar{\mathbf{x}}^{(0)} = \mathbf{x}^{(0)}, \mathbf{y}^{(0)} \in \operatorname{dom} f^*$
2: **while** not converged **do**
3: $\quad \mathbf{y}^{(\mathrm{i}+1)} = \operatorname{prox}_{\sigma g^*}(\mathbf{y}^{(\mathrm{i})} + \sigma \mathbf{K} \bar{\mathbf{x}}^{(\mathrm{i})})$
4: $\quad \mathbf{x}^{(\mathrm{i}+1)} = \operatorname{prox}_{\tau h}(\mathbf{x}^{(\mathrm{i})} - \tau \mathbf{K}^* \mathbf{y}^{(\mathrm{i}+1)})$
5: $\quad \bar{\mathbf{x}}^{(\mathrm{i}+1)} = \mathbf{x}^{(\mathrm{i}+1)} + \theta(\mathbf{x}^{(\mathrm{i}+1)} - \mathbf{x}^{(\mathrm{i})})$
6: **end while**

Equation (2.73) has also the equivalent dual problem

$$\max_{\mathbf{y} \in \operatorname{dom} g^*} -(h^*(-\mathbf{K}^* \mathbf{y}) + g^*(\mathbf{y}))\,, \tag{2.75}$$

where $\mathbf{K}^*$ is the adjoint operator to $\mathbf{K}$ such that $\langle \mathbf{K}\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{K}^* \mathbf{y} \rangle$ is fulfilled, e.g. for $\mathbf{K} \in \mathbb{R}^{M \times N}$ it is simply $\mathbf{K}^T$. The primal-dual algorithm [27] operates directly on the primal-dual formulation of Equation (2.74) and is listed in Algorithm 6. It consists of a proximal gradient ascent step in the dual variable $\mathbf{y}$ and a proximal gradient descent step in the primal variable $\mathbf{x}$. Additionally, it adds an over-relaxation step. The algorithm is guaranteed to converge with $\mathcal{O}\left(\frac{1}{i}\right)$ if the parameters step sizes σ and τ fulfill $\sigma\tau \left\|\mathbf{K}\right\|_2^2 < 1$ and over-relaxation parameter $\theta = 1$.

The authors of [27] further show that the primal-dual algorithm can be accelerated if either g^* or h have an L-Lipschitz continuous gradient. Let us assume that this is the case for h, and set $\gamma = \frac{1}{L}$. Then we can proof an $\mathcal{O}\left(\frac{1}{i^2}\right)$ convergence rate, if we set $\sigma^{(0)}\tau^{(0)} \left\|\mathbf{K}\right\|_2^2 < 1$ and update the step sizes as well as the over-relaxation parameter in each iteration according to $\theta^{(\mathrm{i})} = \frac{1}{\sqrt{1+2\gamma\tau^{(\mathrm{i})}}}$, $\tau^{(\mathrm{i}+1)} = \theta^{(\mathrm{i})}\tau^{(\mathrm{i})}$, and $\sigma^{(\mathrm{i}+1)} = \frac{\sigma^{(\mathrm{i})}}{\theta^{(\mathrm{i})}}$.

2.2.3 Convex Optimization for Imaging

In the last two sections we have introduced the basic mathematical preliminaries, concepts of convex analysis and optimization. This last section on convex optimization is devoted to its application in imaging. In computer vision we generally deal with inverse problems: We are given an observation f, e.g. an image, or a sequence of images, and are interested in what caused this observation f. In general, we can write that f is the result of a forward process, $f = G(u)$. Hence, the objective of the inverse problem is to find the model parameter u that best describe our observations. A classical example in imaging is denoising, where we observe a noisy image f that is the result of adding some noise ϵ to a clean image u, i.e. $f = u + \epsilon$. In this case, the inverse problem is to recover the clean image u from the noisy observation f. However, this is an ill-posed problem, because infinitely many u can produce f as we do not know ϵ. A remedy to this dilemma is to utilize prior information to regularize the problem. In the case of image denoising, a reasonable assumption is that natural images are smooth.

We can formulate this high-level description with the following optimization problem

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \Gamma(\mathbf{u}) + \lambda \Psi(\mathbf{u}, \mathbf{f}) \,, \tag{2.76}$$

where $\mathbf{f}$ is our observation and $\mathbf{u}$ is the minimizer of our model. $\Psi(\cdot, \cdot)$ is the data fidelity term and ensures that the minimizer is the best explanation for our observation $\mathbf{f}$. Back to our denoising example, $\mathbf{u}^*$ should not deviate too much from the observation $\mathbf{f}$ with respect to a certain norm. Hence, a reasonable choice for the data fidelity term could be $\Psi(\mathbf{u}, \mathbf{f}) = \frac{1}{2} \|\mathbf{u} - \mathbf{f}\|_2^2$. The second term in the optimization problem above is the regularization term $\Gamma(\cdot)$ which encodes prior information about the domain. For problems in computer vision, where we want to recover images, this regularization term usually enforces smooth results. Finally, the parameter $\lambda > 0$ controls the trade-off between the deviation from the observation $\mathbf{f}$ and the regularization of the minimizer $\mathbf{u}^*$.

To use an uncluttered notation within this section, all quantities, i.e. images, are stacked into vectors. For example, an image I of size $H \times W$ is stacked into a vector $\mathbf{u} = (I(1,1), \cdots, I(1,W), I(2,1), \cdots, I(H,W))^T \in \mathbb{R}^{HW}$. Hence, the gradient operator $\nabla \in \mathbb{R}^{2HW \times HW}$ is a sparse matrix that computes the forward differences according to Equation (2.36). In the remainder of this section, we will discuss various choices of the regularization term $\Gamma(\cdot)$ and how to efficiently find the minimizer $\mathbf{u}^*$ of the resulting optimization problems.

2.2.3.1 Tikhonov Regularization

A reasonable assumption for natural images [245], but also for depth maps, is that they are mostly smooth with large gradients only occurring near edges. That means that a regularizer should add a penalty on the image gradients. The Tikhonov [238] model is a simple model for denoising that incorporates this idea and is given as the following optimization problem

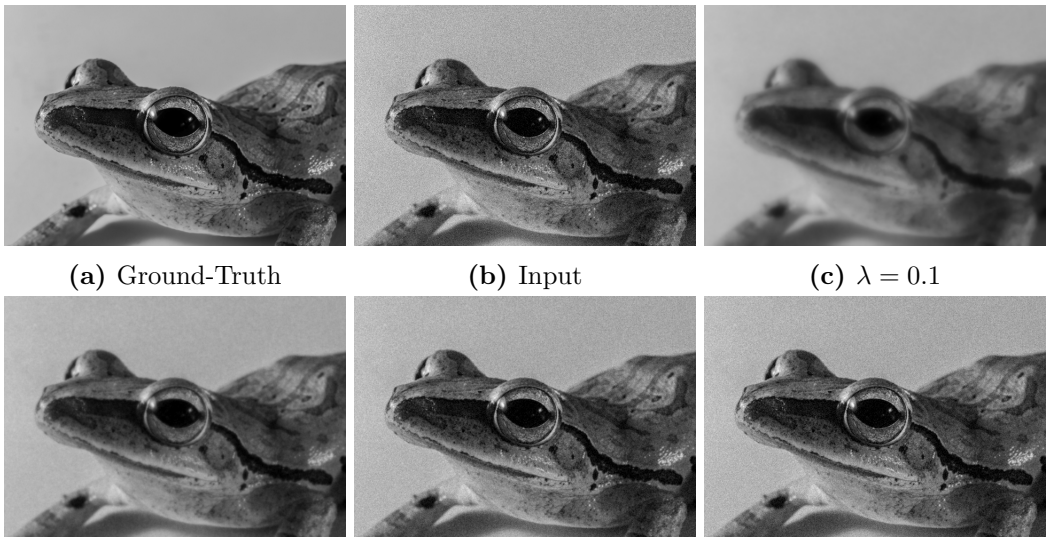
$$\min_{\mathbf{u}} \|\nabla \mathbf{u}\|_2^2 + \lambda \|\mathbf{u} - \mathbf{f}\|_2^2 \,. \tag{2.77}$$

The regularization term $\Gamma(\mathbf{u}) = \|\nabla \mathbf{u}\|_2^2$ ensures that the gradients in the minimizer $\mathbf{u}^*$ are low. The nice property of this energy is that it is smooth, convex and the gradient can be easily computed

$$2(\nabla^T \nabla \mathbf{u} + \lambda(\mathbf{u} - \mathbf{f})) \,. \tag{2.78}$$

Given the gradient, we can use an accelerated optimization algorithm for smooth convex problems, e.g. Algorithm 2. However, plugging the above gradient into the optimality condition of convex problems, we can also obtain a closed form solution

$$\mathbf{u}^* = \left(\frac{1}{\lambda} \nabla^T \nabla + I \right)^{-1} \mathbf{f} \,. \tag{2.79}$$



(a) Ground-Truth **(b)** Input **(c)** $\lambda = 0.1$

(d) $\lambda = 1$ **(e)** $\lambda = 10$ **(f)** $\lambda = 25$

Figure 2.7: Results of Tikhonov denoising on a natural image with Gaussian noise. Figure (a) shows the target image and (b) the noisy input (Gaussian noise with zero mean and standard deviation $\sigma = 0.05$). Figures (c) to (f) depict the result of the Tikhonov model for varying λ.

In Figure 2.7 we show the result of Tikhonov denoising with varying λ parameter. The model is able to reduce the noise for $\lambda \leq 1$, but also smooths over edges which yields a blurry result. This stems from the choice of the regularization term, which does not allow discontinuities in the solution.

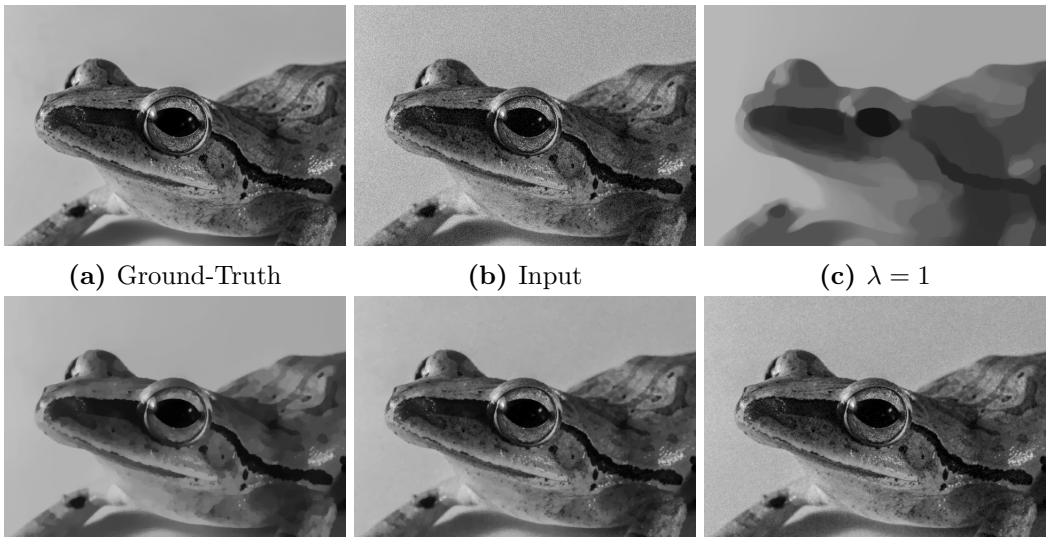
2.2.3.2 Total Variation Regularization

To prevent smoothing over edges we can replace the squared ℓ_2-norm of the Tikhonov regularization term with a robust norm that also allows discontinuities. A very popular choice is the discrete Total Variation $\Gamma(\mathbf{u}) = \mathrm{TV}(\mathbf{u}) = \|\nabla \mathbf{u}\|_1$. By plugging the discrete TV-norm into our denoising model we obtain the famous objective of Rudin, Osher, and Fatemi (ROF) [198]

$$\min_{\mathbf{u}} \|\nabla \mathbf{u}\|_1 + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 \,. \tag{2.80}$$

Note that the ROF model is still convex, as it is a sum of two valid norms, but is not smooth anymore. Therefore, we have to apply an appropriate optimization algorithm as the ones presented in Section 2.2.2.2. A suitable method that we will explain in more detail is Algorithm 6. We start by deriving the convex-concave saddle-point formulation of Equation (2.85) by using the bi-conjugate of the ℓ_1-norm

$$\min_{\mathbf{u}} \max_{\mathbf{p}} \left\langle \mathbf{u}, \nabla^T \mathbf{p} \right\rangle + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 - \delta_P(\mathbf{p}) \,, \tag{2.81}$$



(a) Ground-Truth **(b)** Input **(c)** $\lambda = 1$

(d) $\lambda = 10$ **(e)** $\lambda = 25$ **(f)** $\lambda = 50$

Figure 2.8: Results of ROF denoising on a natural image with Gaussian noise. Figure (a) shows the target image and (b) the noisy input (Gaussian noise with zero mean and standard deviation $\sigma = 0.05$). Figures (c) to (f) depict the result of the ROF model for varying λ.

where $\mathbf{p}$ is the dual variable and the convex set P of the indicator function δ_P is given by

$$P = \{\mathbf{p} | \, \|\mathbf{p}\|_\infty \leq 1\} \, . \tag{2.82}$$

The only thing left to apply Algorithm 6 is to derive the proximal operators. The proximal operator of the above indicator function is the simple point-wise projection onto the Euclidean unit norm ball

$$\text{prox}_{g^*}(\tilde{\mathbf{p}})_{x,y} = \frac{\tilde{\mathbf{p}}_{x,y}}{\max(1, \|\tilde{\mathbf{p}}_{x,y}\|_2)} \, . \tag{2.83}$$

Additionally, the proximal mapping of the squared ℓ_2-norm for the data term is given by the following point-wise expression

$$\text{prox}_h(\tilde{u})_{x,y} = \frac{\tilde{u}_{x,y} + \tau\lambda f_{x,y}}{1 + \tau\lambda} \, . \tag{2.84}$$

In Figure 2.8 we show denoising results of the ROF model. We can observe that with stronger regularization, i.e. smaller λ, the noise is successfully removed while maintaining strong image edges. However, by decreasing the λ value we also remove finer image details. One drawback of the TV regularization is that it enforces piece-wise constant solutions which results in a staircasing effect, see for example Figure 2.8d This is especially undesired for depth maps.

2.2.3.3 Huber Regularization

One way to reduce the staircasing is to replace the TV norm with the Huber norm introduced in Equation (2.20). The Huber norm introduces a trade-off between ℓ_1 and squared ℓ_2 regularization. Hence, with an appropriate choice of the trade-off parameter ϵ it results in smooth results in areas of vanishing gradients while still preserving edges. The primal optimization problem with Huber regularization is given as

$$\min_{\mathbf{u}} \|\nabla \mathbf{u}\|_{\epsilon} + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 \,. \tag{2.85}$$

This optimization problem can be efficiently solved with the primal-dual algorithm as the ROF model before. The corresponding convex-concave saddle-point problem is given by replacing the conjugate $g^*(\mathbf{p}) = \delta_P(\mathbf{p})$ with $g^*(\mathbf{p}) = \delta_P(\mathbf{p}) + \frac{\epsilon}{2} \|\mathbf{p}\|_2^2$, yielding

$$\min_{\mathbf{u}} \max_{\mathbf{p}} \left\langle \mathbf{u}, \nabla^T \mathbf{p} \right\rangle + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 - \delta_P(\mathbf{p}) - \frac{\epsilon}{2} \|\mathbf{p}\|_2^2 \,. \tag{2.86}$$

Therefore, the proximal mapping for $h(\cdot)$ is the same as in the ROF model, and the proximal mapping for the changed $g^*(\cdot)$ is given by the following point-wise projection

$$\text{prox}_{g^*}(\tilde{\mathbf{p}})_{x,y} = \frac{\frac{\tilde{\mathbf{p}}_{x,y}}{1+\sigma\epsilon}}{\max(1, \left\| \frac{\tilde{\mathbf{p}}_{x,y}}{1+\sigma\epsilon} \right\|_2)} \,. \tag{2.87}$$

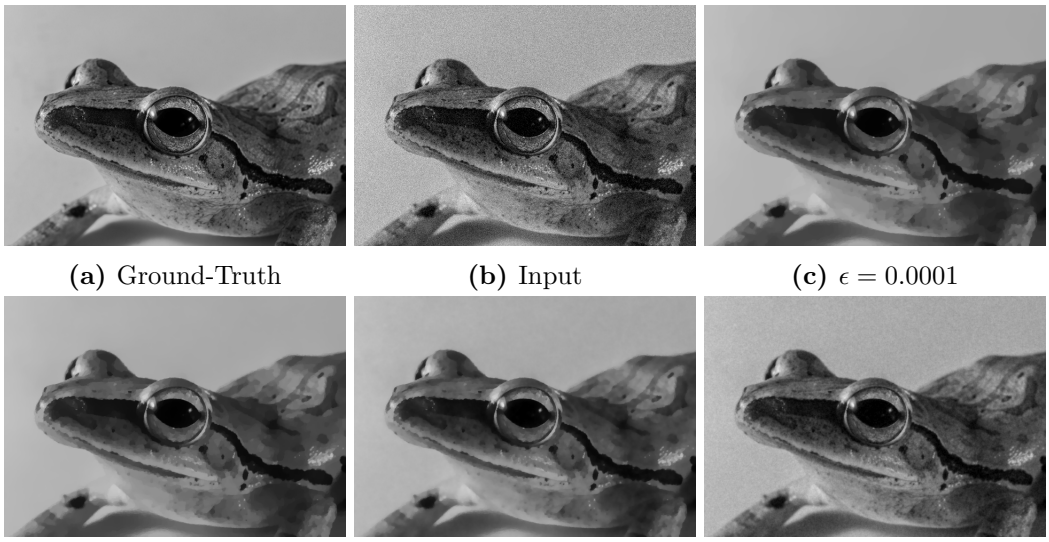
In Figure 2.9 we show the result of the Huber model on image denoising for $\lambda = 10$ and varying values of ϵ. Note that in the limit $\epsilon \to 0$ the Huber model yields the ROF model. Therefore, for small ϵ values the staircasing is still visible and gets less pronounced for bigger values of ϵ. However, if the parameter gets too large we approximate the Tikhonov model, which results in less noise suppression, or blurred edges.

2.2.3.4 Total Generalized Variation Regularization

A more elaborate regularization term to prevent piece-wise constant solutions is the Total Generalized Variation (TGV) [20]. As the name suggests it is a non-trivial generalization of the Total Variation that measures not only jumps but also higher-order variations, e.g. kinks. In this thesis we are especially interested in the TGV of second order[1]. In the discrete setting, the TGV of second order is given by

$$TGV(\mathbf{u}) = \min_{\mathbf{u}} \alpha_1 \|\nabla \mathbf{u} - \mathbf{v}\|_1 + \alpha_0 \|\boldsymbol{\nabla} \mathbf{v}\|_1 \,, \tag{2.88}$$

[1] TGV of the first order is equivalent to TV.



(a) Ground-Truth **(b)** Input **(c)** $\epsilon = 0.0001$

(d) $\epsilon = 0.001$ **(e)** $\epsilon = 0.01$ **(f)** $\epsilon = 0.1$

Figure 2.9: Results of Huber denoising on a natural image with Gaussian noise. Figure (a) shows the target image and (b) the noisy input (Gaussian noise with zero mean and standard deviation $\sigma = 0.05$). Figures (c) to (f) depict the result of the Huber model for varying ϵ and $\lambda = 10$.

where α_1 and α_0 are user defined trade-off parameters and $\boldsymbol{\nabla}$ is the discrete Jacobian operator

$$\boldsymbol{\nabla}\mathbf{v} = \begin{pmatrix} \nabla\mathbf{v}_1 \\ \nabla\mathbf{v}_2 \end{pmatrix} . \tag{2.89}$$

The interpretation of the second order TGV is that the gradient of the solution $\nabla\mathbf{u}$ should only deviate on a sparse set of points from the vector field $\mathbf{v}$. Additionally, the vector field $\mathbf{v}$ should itself only have a sparse set of gradients. This avoids the staircasing effect in affine parts of an image but still enables sharp edges.

Note that the TGV is itself an optimization problem, but is still convex. Hence, the optimization problem for denoising becomes

$$\min_{\mathbf{u}} \mathrm{TGV}(\mathbf{u}) + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 \Leftrightarrow \min_{\mathbf{u},\mathbf{v}} \alpha_1 \|\nabla\mathbf{u} - \mathbf{v}\|_1 + \alpha_0 \|\boldsymbol{\nabla}\mathbf{v}\|_1 + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 . \tag{2.90}$$

We can again utilize Algorithm 6 to efficiently minimize the above problem. First, by using the bi-conjugate of the ℓ_1 norms of the regularization term we derive the corresponding convex-concave saddle-point problem

$$\min_{\mathbf{u},\mathbf{v}} \max_{\mathbf{p},\mathbf{q}} \alpha_1 \langle \nabla\mathbf{u} - \mathbf{v}, \mathbf{p} \rangle + \alpha_0 \langle \boldsymbol{\nabla}\mathbf{v}, \mathbf{q} \rangle + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 - \delta_P(\mathbf{p}) - \delta_Q(\mathbf{q}) . \tag{2.91}$$

The convex sets of the indicator functions δ_P and δ_Q are given by

$$P = \{\mathbf{p} | \, \|\mathbf{p}\|_\infty \leq 1\} \tag{2.92}$$
$$Q = \{\mathbf{q} | \, \|\mathbf{q}\|_\infty \leq 1\} \, . \tag{2.93}$$

If we now set $g^*((\mathbf{p}, \mathbf{q})) = g_1^*(\mathbf{p}) + g_2^*(\mathbf{q})$ with $g_1^*(\mathbf{p}) = \delta_P(\mathbf{p})$ and $g_2^*(\mathbf{q}) = \delta_P(\mathbf{q})$ and the operator $\mathbf{K}$ to

$$\mathbf{K} = \begin{pmatrix} \alpha_1 \nabla & -\alpha_1 \mathbf{I} \\ 0 & \alpha_0 \boldsymbol{\nabla} \end{pmatrix} , \tag{2.94}$$

we can obtain the canonical form for the primal-dual algorithm

$$\min_{\mathbf{u},\mathbf{v}} \max_{\mathbf{p},\mathbf{q}} \left\langle \mathbf{K} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \begin{pmatrix} \mathbf{p} \\ \mathbf{q} \end{pmatrix} \right\rangle + \frac{\lambda}{2} \|\mathbf{u} - \mathbf{f}\|_2^2 - g^*((\mathbf{p}, \mathbf{q})) \, . \tag{2.95}$$

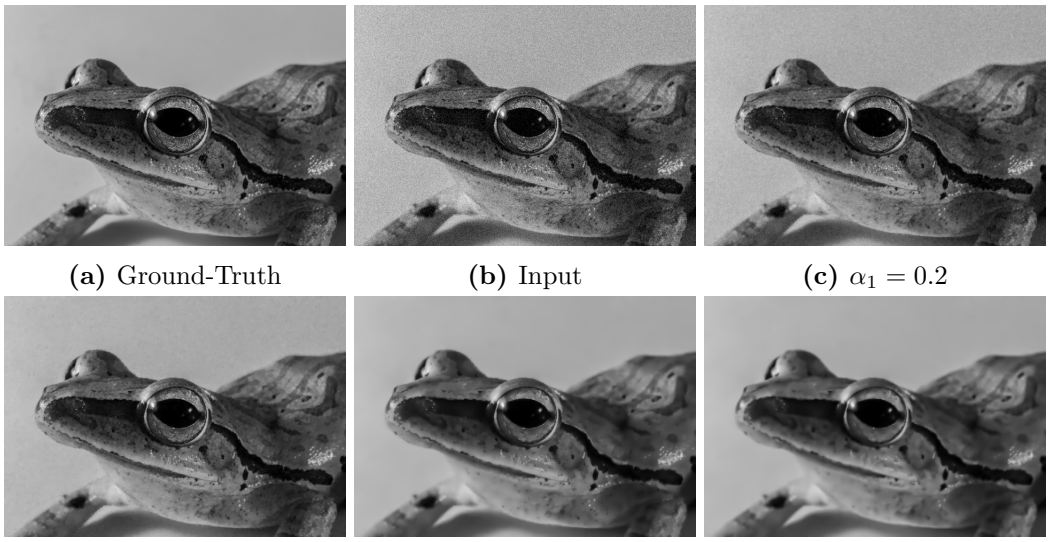
From this canonical form it is now simple to derive the steps for the primal-dual algorithm. As the proximal operator are all defined point-wise, we can obtain the following iteration schema

$$\begin{cases} \mathbf{p}^{(\mathrm{i}+1)} = \mathrm{prox}_{g_1^*}(\mathbf{p}^{(\mathrm{i})} + \sigma(\nabla \bar{\mathbf{u}}^{(\mathrm{i})} - \bar{\mathbf{v}}^{(\mathrm{i})})) \\ \mathbf{q}^{(\mathrm{i}+1)} = \mathrm{prox}_{g_2^*}(\mathbf{q}^{(\mathrm{i})} + \sigma \boldsymbol{\nabla} \bar{\mathbf{v}}^{(\mathrm{i})}) \\ \mathbf{u}^{(\mathrm{i}+1)} = \mathrm{prox}_{h}(\mathbf{u}^{(\mathrm{i})} - \tau \nabla^T \mathbf{p}^{(\mathrm{i}+1)}) \\ \mathbf{v}^{(\mathrm{i}+1)} = \mathbf{v}^{(\mathrm{i})} - \tau(\boldsymbol{\nabla}^T \mathbf{q}^{(\mathrm{i}+1)} - \mathbf{p}^{(\mathrm{i}+1)}) \\ \bar{\mathbf{u}}^{(\mathrm{i}+1)} = 2\mathbf{u}^{(\mathrm{i}+1)} - \mathbf{u}^{(\mathrm{i})} \\ \bar{\mathbf{v}}^{(\mathrm{i}+1)} = 2\mathbf{v}^{(\mathrm{i}+1)} - \mathbf{v}^{(\mathrm{i})} \end{cases} . \tag{2.96}$$

In Figure 2.10 we visualize results of the TGV model for denoising. We vary the ratio of α_1 to α_0. A higher ratio leads to smoother results, whereas a lower ratio yields results similar to the ROF-model. The difference between the individual regularization terms is also nicely visible in 3D, i.e. the input image is interpreted as a depth map. Figure 2.11 shows the influence of the different regularization terms on a step-like function, where we fix $\lambda = 10$. We can observe how the ROF model produces the staircasing effect and in contrast, the very smooth result of the TGV model.

2.3 Deep Learning

In the following section we introduce the basic concepts of machine learning in general, and of deep learning specifically. The term of machine learning dates back at least to the late 1950s when Arthur L. Samuel [200] trained a computer to play the game of checkers. In the work, he states that coding all the rules is a cumbersome task and that “Programming



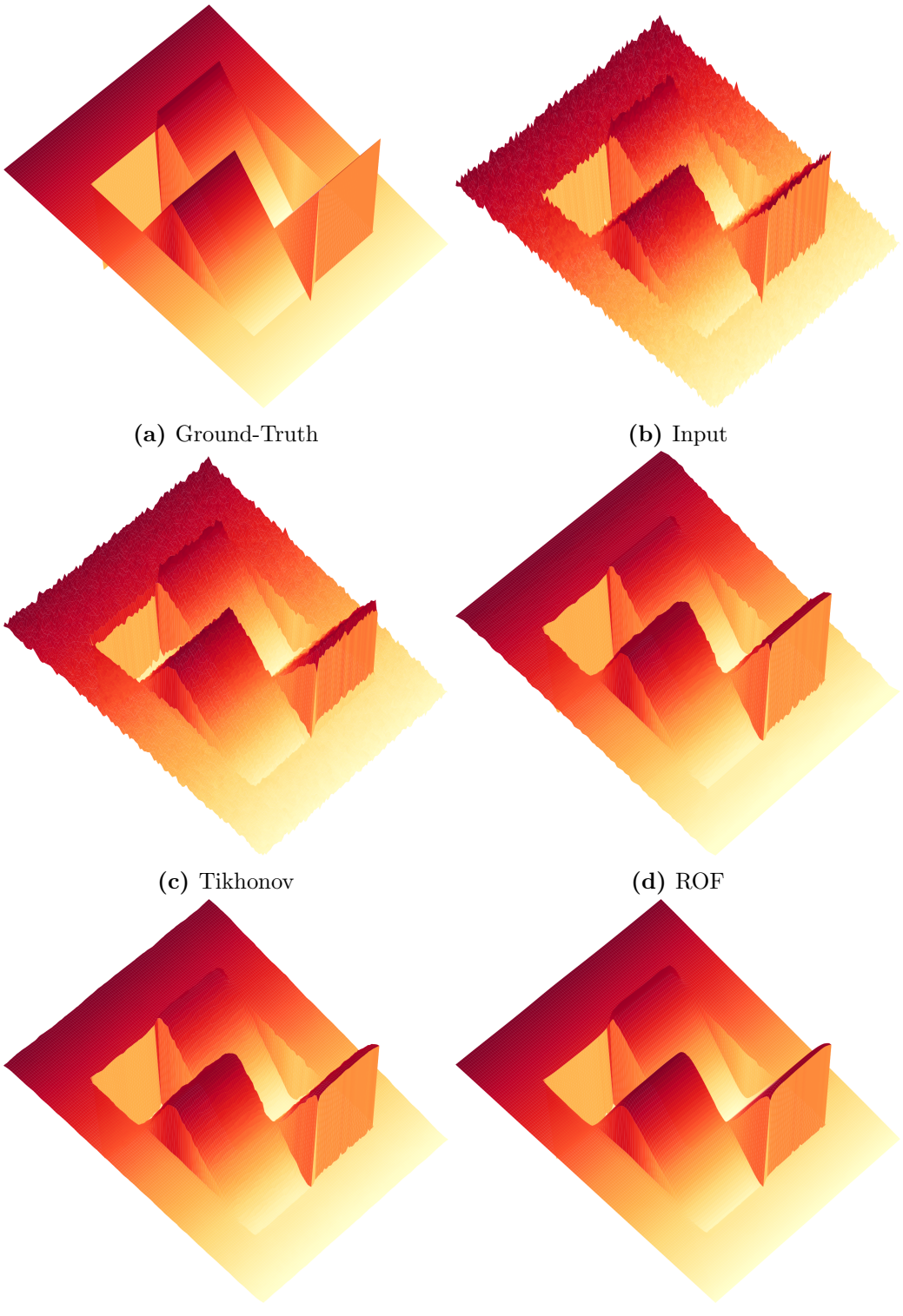
(a) Ground-Truth **(b)** Input **(c)** $\alpha_1 = 0.2$

(d) $\alpha_1 = 0.35$ **(e)** $\alpha_1 = 0.65$ **(f)** $\alpha_1 = 1.0$

Figure 2.10: Results of TGV denoising on a natural image with Gaussian noise. Figure (a) shows the target image and (b) the noisy input (Gaussian noise with zero mean and standard deviation $\sigma = 0.05$). Figures (c) to (f) depict the result of the TGV model for varying α_1, constant $\alpha_0 = 1$ and $\lambda = 10$.

computers to learn from experience should eventually eliminate the need for much of this detailed programming effort". He is further often attributed with the following statement: "machine learning is the field of study that gives computers the ability to learn without being explicitly programmed". This is a very broad definition of machine learning, but already emphasizes an important aspect, namely that we do not, or are not able to hand-craft rules for highly complex systems. Another definition by David Barber [8] emphasizes the data-driven aspect: "Machine Learning is the study of data-driven methods capable of mimicking, understanding and aiding human and biological information processing tasks". Further, Keven P. Murphy [161] highlights the ability to generalize to future events: "The goal of machine learning is to develop methods that can automatically detect patterns in data and then to use the uncovered patterns to predict future data or other outcomes of interest". This is related to the common problem of over-fitting, which we will discuss later in this section. One of the most formal definitions is given by Tom M. Mitchell [159]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

In the following section, we will go into more detail on the three points: experiences E, tasks T, and performance measures P. However, the field of machine learning and deep learning presents an almost endless amount of literature and covering all aspects is beyond the scope of this thesis. We refer the interested reader to some of the great available textbooks [8, 13, 58, 159, 161] for a more detailed overview on machine learning and to [92] specifically for deep learning.



(a) Ground-Truth **(b)** Input

(c) Tikhonov **(d)** ROF

(e) Huber, $\epsilon = 0.01$ **(f)** TGV, $\alpha_1 = 0.75, \alpha_0 = 1$

Figure 2.11: Influence of the regularization term on depth map denoising. The original depth map in (a) has values between 0 and 1. We apply additive Gaussian noise with zero mean and standard deviation $\sigma = 0.025$ to generate the noisy input in (a). The Figures (c) to (f) show the influence of the regularization term on the denoising result. We fixed $\lambda = 10$.

2.3.1 Supervised Learning

From the definitions above one can obviously conclude that the field of machine learning is very broad. Depending on the task T, we can roughly divide the field into supervised and unsupervised learning, whereas we are especially interested in the former within this thesis. In supervised learning we are given a training dataset of K samples with corresponding target labels, i.e. $\mathcal{D} = \{(\mathbf{s}_k, \mathbf{t}_k)\}_{k=1}^{K}$, which corresponds to the experience E of Mitchell's definition. The samples $\mathbf{s}_k$ are from a defined input, or data space $\mathbf{s}_k \in \mathcal{X}$. We usually have $\mathcal{X} \subseteq \mathbb{R}^N$, $\mathcal{X} \subseteq \mathbb{R}^{H \times W}$, or $\mathcal{X} \subseteq \mathbb{R}^{D \times H \times W}$, if we handle vectors, images/depth maps, or volumetric inputs, respectively. However, in other scenarios, it is also possible to have categorical, or missing inputs. The label space $\mathcal{Y}$ is dependent on the given task. For example, in classification, we want to assign each sample $\mathbf{s}$ a unique category, or class. Hence, we have $\mathcal{Y} = \{1, \cdots, C\}$, where C is the number of overall categories to consider. Another common supervised learning task is regression in which we want to assign each sample $\mathbf{s}$ a real-valued vector. In this case the label space would be $\mathcal{Y} \subseteq \mathbb{R}^N$.

The goal of supervised learning can now be defined as finding a mapping $f : \mathcal{X} \to \mathcal{Y}$ given a dataset $\mathcal{D}$ such that for a new pair $(\mathbf{s}_*, \mathbf{t}_*) \notin \mathcal{D}$ the mapping is accurate, i.e. $\mathcal{L}(f(\mathbf{s}_*), \mathbf{t}_*) \to \min$. The loss $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a scalar-valued function that measures the accuracy of the prediction and is the last piece P of Mitchell's definition. A more accurate definition can be given if we assume an infinite large training dataset, i.e. $\mathcal{D}_\infty$ is the data generating distribution that covers all possible samples. Then objective of supervised learning is to find $f^* \in \mathcal{F}$ that minimizes the expected loss

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(\mathbf{s},\mathbf{t}) \sim \mathcal{D}_\infty} \mathcal{L}(f(\mathbf{s}), \mathbf{t}) \,, \tag{2.97}$$

where $\mathcal{F}$ is the set of functions which we consider for this task. Unfortunately, the optimization problem in Equation (2.97) is in practice intractable, as we do not have access to the full data generating distribution $\mathcal{D}_\infty$. However, if we assume that our training dataset $\mathcal{D}$ is independent and identically distributed, then we can approximate the expected loss by the empirical loss

$$f^* \approx \arg\min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}(f(\mathbf{s}_k), \mathbf{t}_k) \,. \tag{2.98}$$

Although the minimization problem in Equation (2.97) and Equation (2.3.1) look quite similar, the latter poses the problem of generalization and over-fitting. This can be demonstrated by a very simple example: Assume that $\mathcal{D}_\infty$ contains all the points on a simple hyperplane, i.e. $\mathcal{D}_\infty = \{(\mathbf{s}, \mathbf{t}) | \mathbf{t} = \langle \mathbf{a}, \mathbf{s} \rangle + b\}$ for an arbitrary $\mathbf{a} \neq 0$ and b, and that the

training dataset contains a finite number of samples K. Then the function

$$f(\mathbf{s}) = \begin{cases} \mathbf{t}_k & \text{if } \mathbf{s} \in \mathcal{D} = \{(\mathbf{s}_k, \mathbf{t}_k)\}_{i=1}^{K} \\ 0 & \text{else} \end{cases}, \tag{2.99}$$

would be an optimal function according to Equation (2.3.1). However, it is trivial to verify that this function fails on all samples outside the training dataset. A visual example of under- and over-fitting is visualized in Figure 2.12.

From this admittedly constructed example, one can already observe the key problem regarding over-fitting. Over-fitting occurs if the function space $\mathcal{F}$ is not restricted enough. Given the previous example, if we restrict $\mathcal{F}$ to only the function that model a hyperplane, then the optimization problem reduces to finding $(\mathbf{a}, b)$. However, if $\mathcal{F}$ is defined too restrictive, then we have the problem of under-fitting, i.e. if $\mathcal{F}$ contains only constant functions.

Formally, the restriction on the function space $\mathcal{F}$ can be enforced by adding a regularization term $\Gamma(f)$ to

$$f^* \approx \arg\min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}(f(\mathbf{s}_k), \mathbf{t}_k) + \Gamma(f)\,. \tag{2.100}$$

The regularization term $\Gamma : \mathcal{F} \to \mathbb{R}$ is a scalar valued function that encodes the preferences for some functions over others, or available prior knowledge. This formulation follows Occam's razor, which states: "Given two explanations for an occurrence, the simpler one is usually the better". The choice of Γ obviously depends on the function space $\mathcal{F}$. In Section 2.3.6 we will discuss some possibilities specifically for deep networks.

There exists also a whole field regarding the statistical learning theory [15, 252, 253, 251] that is concerned with ways of how to quantify the capacity of $\mathcal{F}$. The best known is the Vapnik-Chervonenkis dimension, or VC-dimension, which defines the capacity of a binary classifier. Assume a binary classifier, then the VC-dimension is the maximum number m, for which a training set $\mathcal{D}$ with m training samples exists, such that the classifier can arbitrarily label them. For example, if the classifier is a line in 2D, then the VC-dimension is 3, as this is the maximum number of points a line can arbitrarily divide. Further, a nearest neighbor classifier has a VC-dimension of ∞.

The problem of the generalization error, i.e. the difference between the expected loss and the empirical loss, is also addressed in the bias-variance decomposition, or bias-variance trade-off. We want our model to generalize to unseen data, but also to capture the complexity of the training data. Unfortunately, it is often not possible to jointly achieve both goals and hence, we have to make a trade-off. But we can decompose the generalization error into three parts to investigate this dilemma: an *irreducible error* σ^2, a *bias error*, and a *variance error*. The irreducible error can not be avoided, independently of the function space $\mathcal{F}$. This type of noise can, for example, originate from label noise

in $\mathcal{D}$. The bias error originates from the simplifying assumptions taken for the function space, i.e. a high bias indicates that the function space is not expressive enough to fit data (under-fitting). The variance error is the change of the error by using different training data, i.e. a high variance indicates over-fitting. This already indicates that by the selection of the function space $\mathcal{F}$ one can trade-off the bias and variance in the error.

Let us have a look at an example: Assume we want to approximate a function $\hat{f}$ having a dataset with $\mathbf{t}_k = \hat{f}(\mathbf{s}_k) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Further, $P(\mathcal{D})$ is the distribution over all possible training datasets and we define the loss $\mathcal{L}$ for this example as the squared difference between the estimate $f(\mathbf{s})$ and the target $\mathbf{t}$

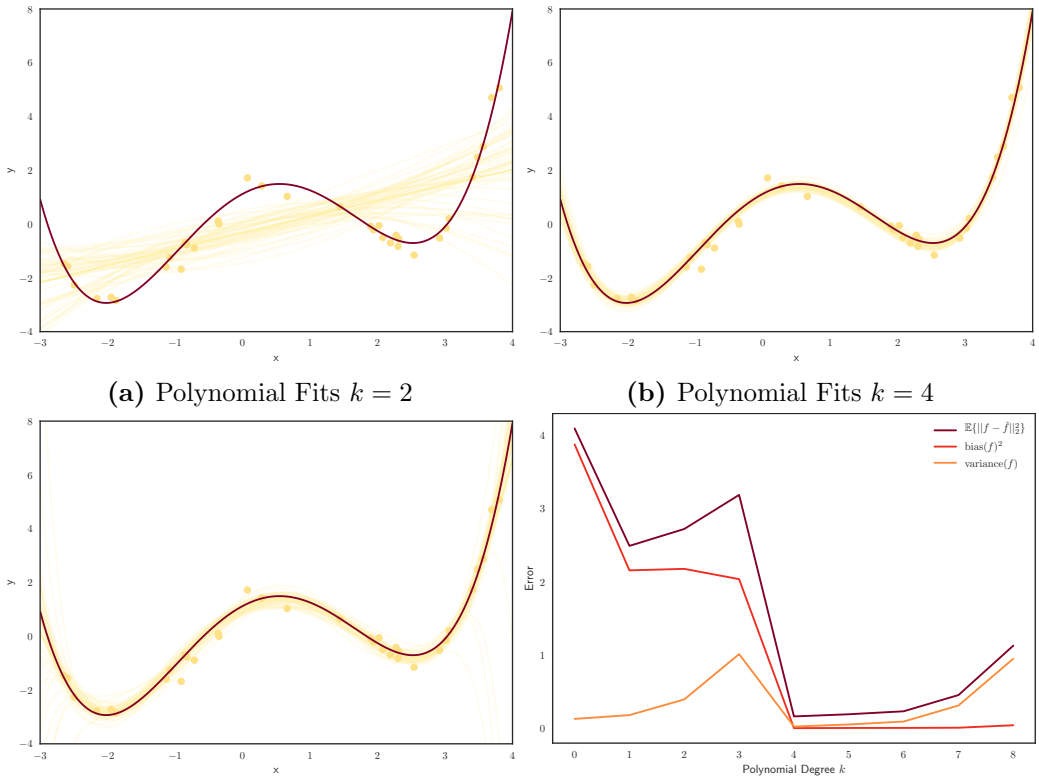
$$\mathcal{L}(f(\mathbf{s}), \mathbf{t}) = \sum_{i=1}^{N} (f(\mathbf{s})_i - t_i)^2 = \|f(\mathbf{s})_i - \mathbf{t}\|_2^2 \,, \tag{2.101}$$

where $f(\mathbf{s})$ is the minimizer of Equation (2.100) trained on a dataset $\mathcal{D} \sim P(\mathcal{D})$. Then, the expected error for a previously unseen pair $(\mathbf{s}, \mathbf{t})$ can be decomposed as follows

$$\begin{aligned}
\mathbb{E}\left\{\mathcal{L}(f, \mathbf{t})\right\} &= \mathbb{E}\left\{\|f - \mathbf{t}\|_2^2\right\} && (2.102)\\
&= \mathbb{E}\left\{\left\|f - \hat{f} + \hat{f} - \mathbf{t}\right\|_2^2\right\} && (2.103)\\
&= \mathbb{E}\left\{\left\|f - \hat{f}\right\|_2^2 + 2\left\langle f - \hat{f}, \hat{f} - \mathbf{t}\right\rangle + \left\|\hat{f} - \mathbf{t}\right\|_2^2\right\} && (2.104)\\
&= \mathbb{E}\left\{\left\|f - \hat{f}\right\|_2^2\right\} + 2\mathbb{E}\left\{\left\langle f - \hat{f}, \hat{f} - \mathbf{t}\right\rangle\right\} + \mathbb{E}\left\{\left\|\hat{f} - \mathbf{t}\right\|_2^2\right\} , && (2.105)
\end{aligned}$$

where $\hat{f}(\mathbf{s})$ and $f(\mathbf{s})$ are abbreviated as $\hat{f}$ and f, respectively, and $\mathbb{E}_{\mathcal{D}\sim P(\mathcal{D})}\{\cdot\}$ is shortened as $\mathbb{E}\{\cdot\}$. Now, we can see that the last expectation term is $\mathbb{E}\left\{\left\|\hat{f} - \mathbf{t}\right\|_2^2\right\} = \mathbb{E}\left\{\|\epsilon\|_2^2\right\} = \sigma^2$ and corresponds to the irreducible noise. The second term can be shown to be $2\mathbb{E}\left\{\left\langle f - \hat{f}, \hat{f} - \mathbf{t}\right\rangle\right\} = 0$ by expansion and by $\mathbb{E}\left\{\hat{f}\right\} = \hat{f}$ as $\hat{f}$ is constant. Therefore, we can further decompose the first term as follows

$$\begin{aligned}
\mathbb{E}\left\{\left\|f - \hat{f}\right\|_2^2\right\} &= \mathbb{E}\left\{\left\|f - \mathbb{E}\{f\} + \mathbb{E}\{f\} - \hat{f}\right\|_2^2\right\} && (2.106)\\
&= \mathbb{E}\left\{\|f - \mathbb{E}\{f\}\|_2^2\right\} + \underbrace{\mathbb{E}\left\{\left\|\mathbb{E}\{f\} - \hat{f}\right\|_2^2\right\}}_{=\left\|\mathbb{E}\{f\} - \hat{f}\right\|_2^2} + \underbrace{2\left\langle f - \mathbb{E}\{f\}, \mathbb{E}\{f\} - \hat{f}\right\rangle}_{=0} . && (2.107)
\end{aligned}$$



(a) Polynomial Fits $k = 2$ **(b)** Polynomial Fits $k = 4$

(c) Polynomial Fits $k = 8$ **(d)** Error Decomposition

Figure 2.12: Figures (a) to (c) depict several polynomial fits (light colors) to an unknown function (dark color). For each fit, we sample a different dataset (depicted as light points). The polynomial of degree $k = 1$ is not expressive enough to fit the data, i.e. under-fitting, whereas the polynomial of degree $k = 8$ fit the noise in the training datasets, i.e. over-fitting. Figure (d) depicts the bias-variance decomposition over the sampled training datasets. We can observe that with more expressive polynomials the bias decreases. The variance increases up to a point, reaches a minimum at the right polynomial and then increases again.

Hence, the final decomposition is given by

$$\mathbb{E}\left\{\mathcal{L}(f, \mathbf{t})\right\} = \underbrace{\mathbb{E}\left\{\left\|f - \mathbb{E}\left\{f\right\}\right\|_2^2\right\}}_{\text{variance}(f)} + \underbrace{\left\|\mathbb{E}\left\{f\right\} - \hat{f}\right\|_2^2}_{\text{bias}(f)^2} + \sigma^2 \,. \tag{2.108}$$

We show an example of a bias-variance decomposition for a polynomial function space in Figure 2.12.

2.3.2 Feed-Forward Networks

In the previous section, we introduced supervised learning and some theoretical insights for the abstract learning problem. However, we did not mention any concrete function spaces and their corresponding learning algorithms, i.e. how to solve Equation (2.100) for

a given function space $\mathcal{F}$. Over the last couple of decades, several learning algorithms have been proposed. Some popular ones are k-nearest neighbour [3], classification and regression trees [22], random forests [23], or support vector machines [44]. An extensive overview is beyond the scope of this thesis and we refer the interested reader therefore to [8, 13, 58, 159, 161] for more information.

In this section, we focus on a special kind of machine learning model, i.e. deep networks, where the basic ideas are already quite old. However, they have only recently been gained a lot of popularity with the drastic improvements shown on public benchmarks, most notable on image classification [132]. Since, this major success deep networks are transforming the field, not only in computer vision, but also in natural language processing [227], speech recognition [11, 105], and also starts getting used in bio-informatics [37].

To introduce the methods behind deep learning it might be useful to start with its most basic origin, linear regression. Linear regression was independently discovered by Legendre [139] and Gauss [78] to fit the orbits of bodies to astronomical observations, i.e. a supervised learning task. In its simplest form, we fit an affine function minimizing squared differences loss: Given a dataset $\mathcal{D}$, the minimization problem is

$$\min_{\mathbf{W},\mathbf{b}} \frac{1}{2K} \sum_{k=1}^{K} \|(\mathbf{W}\mathbf{s}_k + \mathbf{b}) - \mathbf{t}_k\|_2^2 \,. \tag{2.109}$$

Note that this minimization problem is equivalent to the one in Equation (2.3.1) with the loss function defined in Equation (2.101) and the function space $\mathcal{F} = \left\{ f(\mathbf{s}) = \mathbf{W}\mathbf{s} + \mathbf{b} | \mathbf{W} \in \mathbb{R}^{M \times N}, \mathbf{b} \in \mathbb{R}^N \right\}$, given that $\mathbf{s} \in \mathbb{R}^N, \mathbf{t} \in \mathbb{R}^M$. The nice thing about this simple linear regression problem is that it admits a closed-form solution

$$\tilde{\mathbf{W}} = \mathbf{T}\mathbf{S}^T(\mathbf{S}^T\mathbf{S})^{-1} = \arg\min_{\mathbf{W},\mathbf{b}} \frac{1}{2K} \sum_{k=1}^{K} \|(\mathbf{W}\mathbf{s}_k + \mathbf{b}) - \mathbf{t}_k\|_2^2 \,. \tag{2.110}$$

where $\tilde{\mathbf{W}} = (\mathbf{W}|\mathbf{b})$ and $\mathbf{S}$ and $\mathbf{T}$ are matrices formed by stacking all K samples and targets

$$\mathbf{S} = \begin{pmatrix} \mathbf{s}_1 & \cdots & \mathbf{s}_K \\ 1 & \cdots & 1 \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} \mathbf{t}_1 & \cdots & \mathbf{t}_K \end{pmatrix} . \tag{2.111}$$

The drawback of this simple regression model is that it can only model linear relationships between the input samples and the targets. To model also non-linear relationships we can resort to basis functions $\phi_l(\mathbf{s})$, i.e. the input samples are fed through L fixed non-linear transformations. The optimization can still be solved as in Equation (2.110), but

instead of directly using the samples matrix $\mathbf{S}$, we substitute it with the feature matrix $\mathbf{\Phi}$

$$\Phi = \begin{pmatrix} \phi_1(\mathbf{s}_1) & \phi_1(\mathbf{s}_2) & \cdots & \phi_1(\mathbf{s}_K) \\ \phi_2(\mathbf{s}_2) & \phi_2(\mathbf{s}_2) & \cdots & \phi_2(\mathbf{s}_K) \\ \cdots & \cdots & \ddots & \cdots \\ \phi_L(\mathbf{s}_2) & \phi_L(\mathbf{s}_2) & \cdots & \phi_L(\mathbf{s}_K) \end{pmatrix} . \tag{2.112}$$

Hence, we replace each input sample with a feature vector $\phi(\mathbf{s}) = (\phi_1(\mathbf{s}), \cdots, \phi_L(\mathbf{s}))^T$. The basis functions can be polynomials, sine functions, wavelets, radial basis functions, etc.. If the basis function is the identity mapping, i.e. $\phi(\mathbf{s}) = \mathbf{s}$, we are back to simple linear regression.

This kind of regression model has still the drawback that the basis functions are fixed and have to be selected in advance. However, we can use parametrised basis functions $\sigma^{\mathbf{W},\mathbf{b}}(\mathbf{s}) = \sigma(\mathbf{W}\mathbf{s} + \mathbf{b})$ and rewrite Equation (2.109)

$$\min_{\mathbf{W},\mathbf{b}} \frac{1}{2K} \sum_{k=1}^{K} \left\| \sigma^{\mathbf{W},\mathbf{b}}(\mathbf{s}_k) - \mathbf{t}_k \right\|_2^2 . \tag{2.113}$$

In this case the basis function σ is a non-linear function and its input is a linear combination of the input sample $\mathbf{s}$ and the adaptive parameters $\mathbf{W}, \mathbf{b}$ that belong to the basis function. If we now composite a series of parametrised basis functions, i.e. $\sigma^{\mathbf{W},\mathbf{b}} = \sigma_L^{\mathbf{W}_L,\mathbf{b}_L} \circ \cdots \circ \sigma_1^{\mathbf{W}_1,\mathbf{b}_1}$, we derive multi-layer feed-forward neural networks [199]. The function $\sigma(\mathbf{W}\mathbf{s}+\mathbf{b})$ is sometimes called hidden layer, and a single output $\sigma(\mathbf{w}_i^T\mathbf{s} + b_i)$ is denoted as neuron, or unit.

One of the earliest formulations of this kind was the perceptron by Frank Rosenblatt [196]. Although it did not use the concatenation of a series of basis functions, it was the first model that could learn the parameters from data. As basis functions, it used the step function

$$\sigma(\mathbf{s}) = \begin{cases} 1 & \text{if } \mathbf{W}\mathbf{s} + \mathbf{b} > 0 \\ 0 & \text{else} \end{cases} . \tag{2.114}$$

It is apparent from the equation above that it learns a separating hyperplane and it was used for binary classification.

Nowadays, different basis functions, in the context of neural networks also called activation functions, are used. Popular ones are the sigmoid function $\sigma(\mathbf{s})_i = (1 + \exp(-s_i))^{-1}$, the tangens hyperbolicus $\sigma(\mathbf{s})_i = \tanh(s_i)$, and the rectified linear unit $\sigma(\mathbf{s})_i = \max(0, s_i)$ [162]. In contrast to the step function, we can compute for each of the latter functions a gradient or at least a sub-gradient. We will see in the next chapters why this is important for network training.

Algorithm 7 Mini-batch (Sub-)Gradient Descent

Given a training dataset $\mathcal{D}$
Given a mini-batch size B
Given an initial parameter vector $\boldsymbol{\Theta}^{(0)} \in \operatorname{dom} g$
Given a step size $\eta > 0$
1: **while** not converged **do**
2: Sample a mini-batch $\mathcal{B}_i$ from $\mathcal{D}$ with $|\mathcal{B}^{(\mathrm{i})}| = B$
3: Compute sub-gradient using $\partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})}) \approx \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} \frac{1}{B} \sum_{(\mathbf{s},\mathbf{t})\in\mathcal{B}^{(\mathrm{i})}} \mathcal{L}(f(\mathbf{s}, \boldsymbol{\Theta}^{(\mathrm{i})}), \mathbf{t})$
4: Compute parameter update $\Delta\boldsymbol{\Theta}^{(\mathrm{i})} = -\eta \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$
5: Update parameters $\boldsymbol{\Theta}^{(\mathrm{i}+1)} = \boldsymbol{\Theta}^{(\mathrm{i})} + \Delta\boldsymbol{\Theta}^{(\mathrm{i})}$
6: **end while**

2.3.3 Network Optimization

In the last chapter we derived the formulation of a feed-forward neural network as a composite of parametrized functions, i.e. $f = \sigma_L^{\mathbf{W}_L,\mathbf{b}_L} \circ \cdots \circ \sigma_1^{\mathbf{W}_1,\mathbf{b}_1}$. The task is now to optimize Equation (2.100) given a training dataset $\mathcal{D}$.

Let us first restrict the function space $\mathcal{F}$ by fixing the network structure. This means that we do not change the number of basis functions or the size of them in the optimization procedure. Therefore, the function space is $\mathcal{F} = \left\{ \sigma_L^{\mathbf{W}_L,\mathbf{b}_L} \circ \cdots \circ \sigma_1^{\mathbf{W}_1,\mathbf{b}_1} | \mathbf{W}_l \in \mathbb{R}^{M_l \times N_l}, \mathbf{b}_l \in \mathbb{R}^{M_l} \right\}$. For the sake of notational brevity, we will summarize all parameters $\mathbf{W}_l, \mathbf{b}_l$ in a single parameter vector $\boldsymbol{\Theta}$ and denote the feed-forward network applied on a sample $\mathbf{s}$ as $f(\mathbf{s}, \boldsymbol{\Theta})$. Now, we can rewrite the empirical loss from Equation (2.100) with $g(\boldsymbol{\Theta}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}(f(\mathbf{s}_k, \boldsymbol{\Theta}), \mathbf{t}_k)$ as

$$\arg\min_{\boldsymbol{\Theta}} g(\boldsymbol{\Theta}) + \Gamma(\boldsymbol{\Theta}) \,. \tag{2.115}$$

Derivative-free Optimization Note that in the general case this learning problem is not convex. Hence, the optimization techniques presented in Section 2.2 are not applicable. However, we can evaluate $g(\boldsymbol{\Theta})$ for any $\boldsymbol{\Theta}$. So, one approach to optimize Equation (2.115) is to use stochastic hill-climbing, or some of its variants [171]. This is basically a guess-and-check algorithm. One starts with an arbitrary $\boldsymbol{\Theta}$ and changes one, or a set of random parameters and checks if $g(\boldsymbol{\Theta})$ has improved. In general, as the optimization problem is not convex, the method might get stuck in local minimas. However, recent research suggests that local minimas are less problematic in deep networks on high dimensional data [40, 48, 123]. The major drawback specifically of such derivative-free optimization techniques is the large number of parameters of modern deep architectures with more than a million or even billion parameters and that this method requires a single network evaluation to update a single parameter, or a subset of parameters.

First-Order Optimization Methods We can improve the optimization efficiency if we further restrict the function space $\mathcal{F}$ to functions for which we can compute a (sub-)gradient. Let us denote the sub-gradient of $g(\boldsymbol{\Theta})$ as $\partial_{\boldsymbol{\Theta}} g$, then we can use a sub-gradient descent scheme similar to the one in Algorithm 3. The key difference in network training is that we usually do not compute the gradient over the whole dataset $\mathcal{D}$, but only on a random subset of samples. This subset is called the mini-batch, and hence, the optimization procedure is commonly called mini-batch gradient descent. The scheme is outlined in Algorithm 7. The algorithm requires a training dataset and three parameters in its vanilla form. First, the mini-batch size B controls how many samples are used to estimate the sub-gradient $\partial_{\boldsymbol{\Theta}} g(\boldsymbol{\Theta})$. This is a trade-off between noisy gradients for too small mini-batches and too high memory and computation requirements for too large mini-batches. It is further dependent on the problem at hand and can be tuned using cross-validation. The second parameter of the algorithm is the initial parameter vector $\boldsymbol{\Theta} \in \operatorname{dom} g$. We will outline efficient strategies for initialization in Section 2.3.6. Finally, we have to choose a step size η, i.e. learning rate, of the parameter update. This is an important choice, as a too large η will make the optimization diverge, and a too small η increases training time. In practice, one can use a simple heuristic to select the initial learning rate. Start with a large learning rate for which the optimization diverges and divide the learning rate by a factor >1 until no divergence occurs. Further, the learning rate is usually annealed during the end of the training procedure.

We are still missing the information on how to efficiently compute the sub-gradient estimate on line 3. This will be discussed in the next section. To conclude this section, we will outline some advanced first-order optimization techniques that can be used to improve the parameter update on line 4 and boost convergence.

One problem with vanilla gradient-descent, in general, is related to the curvature of the loss surface. If the parameters lie in a valley, i.e. an area where the loss surface is more steeply in one dimension than in another, the updates will only slowly improve the loss. A solution to overcome this problem is to add a short-term-memory to the gradient update, i.e. a momentum term [91, 174], that encourages progress along small but consistent gradient directions. We define an intermediate variable $\mathbf{v}^{(\mathrm{i})} = \beta \mathbf{v}^{(\mathrm{i}-1)} + \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$ with $\mathbf{v}^{(0)} = 0$ and replace the parameter update on line 4 of Algorithm 7 with $\Delta\boldsymbol{\Theta}^{(\mathrm{i})} = -\eta \mathbf{v}$. The parameter $\beta > 0$ controls how far back the gradient memory reaches. If we set $\beta = 0$ we are back to vanilla gradient descent, but usually we set it to a value near 1, e.g. $\beta = 0.99$.

An additional improvement to the momentum term that works optimal for convex problems [163], but also works very well for network optimization [226] is the Nesterov accelerated gradient. The core idea is that we already have an estimate of the next parameter vector through the momentum term. Hence, we can look ahead for the gradient update on line 3 of Algorithm 7 by evaluating $\partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})} - \beta \mathbf{v}^{(\mathrm{i}-1)})$. Hence, the running sum of the momentum term becomes $\mathbf{v}^{(\mathrm{i})} = \beta \mathbf{v}^{(\mathrm{i}-1)} + \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})} - \beta \mathbf{v}^{(\mathrm{i}-1)})$. In the normal momentum algorithm, we first make a step in the direction of the current gradient estimate

and add a big step in the momentum direction. In contrast, in the Nesterov gradient, we make first a big step in the momentum direction and then add a correction step to this extrapolated direction. Note that this is very similar to Algorithm 2 with the difference in the step sizes and the stochasticity for network training.

The methods presented so far use a global learning rate for all components of the parameter vector $\boldsymbol{\Theta}$. However, there is progress towards methods that compute an adaptive learning rate per component, and therefore, try to reduce the burden of learning rate optimization. One of the first methods proposed to adapt a different learning rate to each component of $\boldsymbol{\Theta}$ is Adagrad [57]. The core idea is to reduce the effective learning rate for components that have a large gradient and increase it for components with a small gradient. Therefore, we keep a running sum of the squared gradients in Adagrad, i.e. $\mathbf{r}^{(\mathrm{i})} = \mathbf{r}^{(\mathrm{i}-1)} + \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})}) \odot \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$, where $\odot$ denotes the component-wise multiplication and $\mathbf{r}^{(0)} = 0$. Then, the parameter update on line 4 of Algorithm 7 is replaced with $\Delta\boldsymbol{\Theta}^{(\mathrm{i})} = -\eta \frac{\partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})}{\sqrt{\mathbf{r}^{(\mathrm{i})}} + \epsilon}$, where the fraction and square root are component-wise and ϵ is a small constant to prevent divisions by 0, e.g. $\epsilon = 10^{-6}$.

The downside of Adagrad is the monotonic decrease of the learning rate, due to the running sum of the gradients. This decrease is often too aggressive to train deep networks. A way to counter this behavior is proposed in RMSProp [237].Instead of using a straightforward sum of squared gradients to normalize the gradients, we can use an exponentially decaying average of the form $\mathbf{r}^{(\mathrm{i})} = \beta \mathbf{r}^{(\mathrm{i}-1)} + (1-\beta) \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})}) \odot \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$, with β typically set to 0.99.

Finally, one can combine the momentum idea with the adaptive learning rate of RMSProp. This is the intuition behind Adam [128]. In Adam, we compute the first and second moments of the running gradient, i.e. $\mathbf{v}^{(\mathrm{i})} = \beta_1 \mathbf{v}^{(\mathrm{i}-1)} + (1-\beta_1) \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$ and $\mathbf{r}^{(\mathrm{i})} = \beta_2 \mathbf{r}^{(\mathrm{i}-1)} + (1-\beta_2) \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})}) \odot \partial_{\boldsymbol{\Theta}^{(\mathrm{i})}} g(\boldsymbol{\Theta}^{(\mathrm{i})})$, where the authors propose $\beta_1 = 0.9$ and $\beta_2 = 0.999$ as default values. The initial values $\mathbf{v}_0$ and $\mathbf{r}_0$ are set to 0, but this might introduce a bias. To counteract this bias, both moments are bias corrected by $\tilde{\mathbf{v}}^{(\mathrm{i})} = \frac{\mathbf{v}^{(\mathrm{i})}}{1-\beta_1}$ and $\tilde{\mathbf{r}}^{(\mathrm{i})} = \frac{\mathbf{r}^{(\mathrm{i})}}{1-\beta_2}$. Then, the update rule on line 4 of Algorithm 7 becomes $\Delta\boldsymbol{\Theta}^{(\mathrm{i})} = -\eta \frac{\tilde{\mathbf{v}}^{(\mathrm{i})}}{\sqrt{\tilde{\mathbf{r}}^{(\mathrm{i})}} + \epsilon}$. Although Adam requires twice the memory for the gradient update, it is currently recommended as the default algorithm for network optimization.

Second-Order Optimization Methods In principle we could also use second-order methods for network optimization, i.e. using second-order information of $g(\boldsymbol{\Theta})$. This could be achieved by explicitly computing the Hessian matrix of $g(\boldsymbol{\Theta})$ in each iteration as it is done in Newton's method. However, this would require computing and storing a large matrix for each parameter update, e.g. for a deep network with 10^6 parameters it would require to store a matrix with 10^{12} values, assuming a float with 4byte we would need 4 terabytes to store the Hessian. Even if we would use a method like L-BFGS [145] that approximates the inverse Hessian and never explicitly computes it, it still has to be computed over the entire training dataset. However, this could consist of several million

training pairs and using L-BFGS in a stochastic setting is not trivial. Therefore, it is currently common practice to use first-order methods for network optimization.

Initialization A final important aspect of network optimization is the initialization of the network parameters $\boldsymbol{\Theta}^{(0)}$. As already discussed, the loss surface of neural networks is non-convex. Hence, it may converge to different local minimas depending on the initial choice of the $\boldsymbol{\Theta}^{(0)}$ and different local minimas might have drastically different loss values.

First, we should keep in mind what are really bad initializations. Of course setting $\boldsymbol{\Theta}^{(0)} = 0$ is not a good idea, especially for vanilla feed-forward neural networks. The network output would be always 0, and no gradient information could flow through the network. However, setting $\boldsymbol{\Theta}^{(0)} = c$ to any constant c is also not advised. This would lead to the effect that every unit in the same computational node would compute the same output and therefore, would receive the same gradient for the parameter update. In general, we want to avoid any symmetry in the initialization

A simple approach to break the symmetry of the units with high probability is setting them randomly according to some probability distribution. A common choice is to sample them from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and a small standard deviation, e.g. $\sigma = 0.01$. However, a drawback with this parameter initialization scheme is that the variance of the unit output grows with the number of inputs.

This can be circumvented by normalizing the initialization by the number of inputs for each output unit. Such an approach has already been proposed in [137], where the parameters are initialized by a uniform distribution of the form $\mathcal{U}(-\sqrt{\frac{3}{n_{\text{in}}}}, \sqrt{\frac{3}{n_{\text{in}}}})$, where n_{in} is the number of inputs of the output unit for a given weight. A similar initialization scheme has also been proposed by [89], where the also consider the number of output units n_{out}. According to their scheme the weights are initialized by sampling from $\mathcal{N}(0, \frac{2}{n_{\text{in}}+n_{\text{out}}})$.

A specific initialization for units followed by ReLUs as activation function is derived in [103]. They propose to initialize the weight parameters by sampling from $\mathcal{N}(0, \frac{2}{n_{\text{in}}})$. This scheme provides a unit variance of the output units after a ReLU activation, independent of the number of input units. It is currently the most widely used scheme to initialize $\boldsymbol{\Theta}^{(0)}$.

Finally, it is also possible to ensure the orthogonality of the weights by first initializing the weights randomly according to a scheme above and perform afterward a singular-value decomposition or QR decomposition [202].

2.3.4 Backpropagation

In the last section, we have discussed several methods that can be used to optimize networks given that the (sub-)gradient of the network can be computed. The question this section is concerned is how do we compute $\partial_{\boldsymbol{\Theta}} g(\boldsymbol{\Theta})$? We will introduce two techniques: The numeric gradient computation, and the analytic gradient computation.

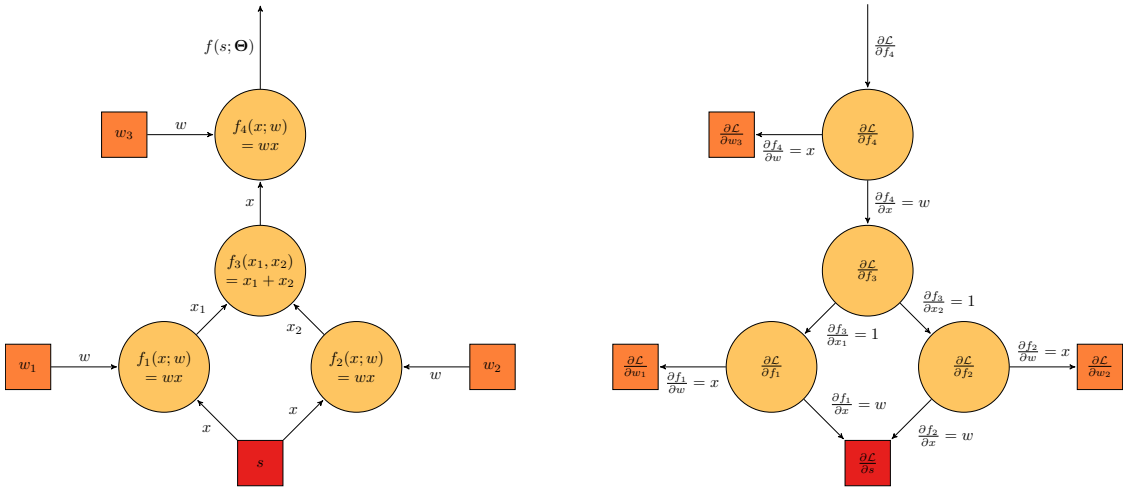


Figure 2.13: Computational graph of the function $f(s,\theta) = w_3(w_1 s + w_2 s)$ in (a) and its corresponding differential graph in (b). Light orange circles, dark orange squares and red squares denote computation nodes, parameters, and inputs, respectively.

Numeric Gradient Computation A simple way to compute the (sub-)gradient $\partial_{\mathbf{\Theta}} g(\mathbf{\Theta})$ is by finite differences. If we look back at the definition of the differential operation in Equation (2.26) we can approximate each component of $\partial_{\mathbf{\Theta}} g(\mathbf{\Theta})$ by

$$\partial_{\Theta_i} g(\mathbf{\Theta}) \approx \frac{g(\mathbf{\Theta} + \epsilon \mathbf{e}_i) - g(\mathbf{\Theta})}{\epsilon}, \tag{2.116}$$

where $\mathbf{e}_i$ is the unit vector with the component at i equal to 1 and all others being 0, and ϵ is a small value. Recall, if $\epsilon \to \infty$ we are back at the exact differential. Using the Taylor expansion we can derive that the error of this approximation decreases linearly in ϵ, i.e. $\left| \frac{g(\mathbf{\Theta}+\epsilon \mathbf{e}_i) - g(\mathbf{\Theta})}{\epsilon} - \partial_{\mathbf{\Theta}_i} g(\mathbf{\Theta})_i \right| = \mathcal{O}(\epsilon)$. We can improve this approximation by using central differences

$$\partial_{\Theta_i} g(\mathbf{\Theta}) \approx \frac{g(\mathbf{\Theta} + \epsilon \mathbf{e}_i) - g(\mathbf{\Theta} - \epsilon \mathbf{e}_i)}{2\epsilon}. \tag{2.117}$$

By using central differences the error of the gradient approximation decreases by $\mathcal{O}(\epsilon^2)$.

The huge drawback of the numeric approach is that we have to evaluate $g(\mathbf{\Theta})$ once, or twice for a single component update. Therefore, it scales linearly with the number of parameters of the network. For modern architectures with millions of parameters this is not feasible to compute, even for a single computation of $\partial_{\mathbf{\Theta}} g(\mathbf{\Theta})$. However, due to its simplicity and its easy implementation, it is still a great tool for debugging.

Analytic Gradient Computation Computing the (sub-)gradient numerically is easy to implement, but is only an approximation and slow in practice. The second way to

compute the (sub-)gradient of $g(\boldsymbol{\Theta})$ is analytically by utilizing basic Calculus. In fact, as the network is only a composite of various functions, we just have to recursively apply the chain rule. This yields the famous Backpropagation algorithm for network training [199], which has its roots in automatic differentiation [143, 144].

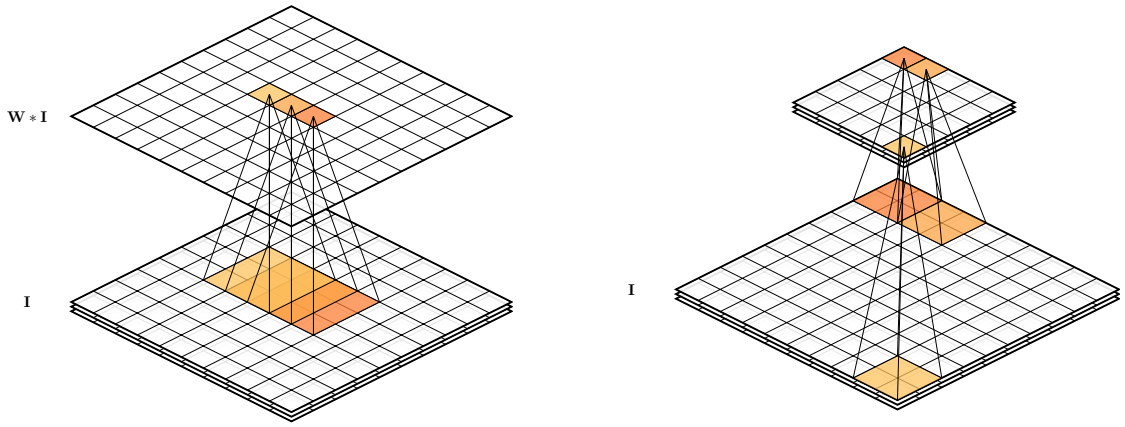
Before we will introduce the Backpropagation algorithm in more detail, we shall generalize the notation of neural networks. So far, we have denoted a feed-forward neural network as a list of successive parametrized basis functions. However, it makes sense to think about more general neural networks as computational graphs, where inputs $\mathbf{s}$ and parameters $\boldsymbol{\Theta}$ flow through a directed acyclic graph (DAG) to compute an output $f(\mathbf{s}; \boldsymbol{\Theta})$.

To illustrate this concept, let us have a look at the example $f(s, \theta) = w_3(w_1 s + w_2 s)$. This corresponds to the computational graph depicted in Figure 2.13a. Given that we evaluated a loss function $\mathcal{L}(f(s, \theta), t)$, the partial derivative of this loss function with respect to the parameter w_1 is $\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_4} \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_1} \frac{\partial f_1}{\partial w_1}$ according to the chain rule. Similarly, the partial derivative of the loss function with respect to w_2 is $\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_4} \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_2} \frac{\partial f_2}{\partial w_2}$. We observe that the term $\frac{\partial \mathcal{L}}{\partial f_4} \frac{\partial f_4}{\partial f_3}$ is common in both partial derivatives. Hence, to compute $\frac{\partial \mathcal{L}}{\partial w_1}$ and $\frac{\partial \mathcal{L}}{\partial w_2}$ we only need to evaluate $\frac{\partial \mathcal{L}}{\partial f_4} \frac{\partial f_4}{\partial f_3}$ once. This is the key behind the Backpropagation algorithm. We can build a backward graph that can be efficiently evaluated. See Figure 2.13b for the backward graph of the given example. A simple breadth-first search algorithm can be used on this graph to compute the derivatives of all parameters.

This approach is also called reverse-mode differentiation, in contrast to forward-mode differentiation. The price we have to pay for the computational efficiency is memory consumption. We have to keep all intermediated results of the forward pass in memory. In both cases, we have to compute the (sub-)gradient of the individual functions of the composite. However, the basic building blocks of neural networks are rather simple and the (sub-)gradient can be quickly computed. For example, for the in Section 2.3.2 introduced activation functions we have the following (sub-)gradients: For the sigmoid function $\sigma(\mathbf{s})_i = (1 + \exp(-s_i))^{-1}$ the gradient is $\frac{\partial \sigma}{\partial s_i} = \sigma(s_i)(1 - \sigma(s_i))$, for the tangens hyperbolicus $\sigma(\mathbf{s})_i = \tanh(s_i)$ the gradient is $\frac{\partial \sigma}{\partial s_i} = \frac{4}{(\exp(-s_i) + \exp(s_i))^2}$, and of the rectified linear unit $\sigma(\mathbf{s})_i = \max(0, s_i)$ the sub-gradient is $\frac{\partial \sigma}{\partial s_i} = [s_i > 0]$. Finally, for the parametrization $f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ with $\mathbf{W} \in \mathbb{R}^{M \times N}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$ the partial derivatives are $\frac{\partial f}{\partial w_{i,j}} = x_j$, $\frac{\partial f}{\partial x_j} = \sum_{i=1}^{M} w_{ij}$, and $\frac{\partial f}{\partial b_j} = 1$.

2.3.5 Convolutional Networks

In the last few sections, we introduced feed-forward neural networks, generalized them to parametrized computational graphs, showed how backpropagation can be used on these graphs to efficiently compute derivatives, which then can be used in adaptive optimization algorithms to learn the parameters. Further, we described non-linear activation functions that together with the parametrized affine transformation can approximate non-linear functions. Indeed, it has been proven in [46, 140] that a feed-forward neural network with two parametrized basis functions and sufficient many units can arbitrary approximate any



(a) Example of a convolution operation on a 2D input with $C = 3$. The different colors indicate the evaluation of the weight kernel on different spatial locations.

(b) Example of a pooling operation on a 2D input with $C = 3$. The different colors indicate the different non-overlapping pooling regions.

Figure 2.14: Convolution and pooling operation.

continuous function on a closed and bounded subset of $\mathbb{R}^N$. Hence, neural networks are also called universal approximators.

The theorem provides us the nice property that neural networks can represent arbitrary functions, but does not state that we are actually able to learn them from data. First of all, an optimization algorithm might get stuck in a poor local minimum, or the network overfits the training data. Further, the theorem does not state how many units we actually need. Finally, we know that the human brain uses a deep hierarchy of neurons to process information.

In fact, the seminal work of Hubel and Wiesel [112, 113, 114] inspired one of the currently most successful methods in computer vision, the convolutional network [136, 137], also known as convolutional neural network (CNN), or ConvNet. Hubel and Wiesel discovered the existence of a hierarchy within the primary visual cortex where simple cells respond to oriented edges at specific locations, and complex cells respond to oriented edges but have a degree of spatial invariance. Similarly, a convolutional network utilizes the principles of sparse interactions, parameter sharing, and equivariant representation, as well as spatial invariance by introducing convolution and pooling operations into the network graph.

Convolution Operation The convolution is a well-understood operation in signal processing. Given two discrete functions f and g it is defined as the following sum of point-wise multiplications

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(m-n) = \sum_{m=-\infty}^{\infty} f(m-n)g(m)\,, \tag{2.118}$$

where the n and m are integers. If we know that g has a bounded support, e.g. it is $g(n) \neq 0$ for $n \in \{M, \cdots, N\}$, then we can rewrite the equation as

$$(f \star g)(n) = \sum_{m \in \text{supp}(g)} f(m-n)g(m)\,. \tag{2.119}$$

Similarly, we can define the cross-correlation between two discrete functions f and g as

$$(f * g)(n) = \sum_{m \in \text{supp}(g)} f(m+n)g(m)\,. \tag{2.120}$$

In the literature of convolutional networks the convolutional usually refers to what is in fact a cross-correlation. We will follow this notation also in this thesis to reduce confusion.

We can now further extend this definition of the convolution two 2D. Let $\mathbf{I}$ be a tensor of $\mathbb{R}^{H \times W \times C}$, e.g. a color image of height H, width W and $C = 3$ color channels. Additionally, W is a weight kernel of $\mathbb{R}^{K_H \times K_W \times C}$. Then, the 2D convolution operation $f(\mathbf{I}; \mathbf{W})$ over C channels is given by

$$f(\mathbf{I}; \mathbf{W})_{h,w} = \sum_{c=1}^{C} \sum_{k_h=1}^{K_H} \sum_{k_w=1}^{K_w} W_{k_h,k_w,c} I_{h',w',c}\,, \tag{2.121}$$

with $h' = h - k_h + \lfloor \frac{K_H}{2} \rfloor$ and $w' = w - k_w + \lfloor \frac{K_W}{2} \rfloor$. We also define $I_{h,w,c} = 0$ for $h \notin [1, H]$ and $w \notin [1, W]$ to handle the border cases. The convolutional can now be understood as applying the same learnable filter to all spatial locations of $\mathbf{I}$. A visual example of this concept is depicted in Figure 2.14a. Note that we can write the convolution operation also as a matrix-vector multiplication of the form $\hat{f}(\mathbf{I}; \mathbf{W}) = \hat{\mathbf{W}}\mathbf{x}$, where $\mathbf{I}$ is vectorized as $\mathbf{x} = (0, \cdots, 0, I_{1,1,1}, I_{1,1,2}, \cdots, I_{i,j,c}, \cdots, I_{H,W,C}, 0, \cdots, 0)^T$[2] and $\hat{\mathbf{W}}$ is a Toeplitz matrix of the following form

$$\hat{\mathbf{W}} = \begin{pmatrix} W_{1,1,1} & \cdots & W_{1,1,C} & W_{1,2,1} & \cdots & W_{K_H,K_W,C} & 0 & 0 & \cdots & 0 \\ 0 & W_{1,1,1} & \cdots & W_{1,1,C} & W_{1,2,1} & \cdots & W_{K_H,K_W,C} & 0 & \cdots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & W_{1,1,1} & \cdots & W_{1,1,C} & W_{1,2,1} & \cdots & W_{K_H,K_W,C} & 0 \\ 0 & 0 & \cdots & 0 & W_{1,1,1} & \cdots & W_{1,1,C} & W_{1,2,1} & \cdots & W_{K_H,K_W,C} \end{pmatrix}. \tag{2.122}$$

Given this form we can already utilize it in the computational graph introduced in the last section. Further, from this Toeplitz matrix representation we can also observe the key properties of this operation. First, units are sparsely connected: Each output unit is only connected to $K_H \cdot K_W \cdot C$ units of the input, instead of interacting with each input unit. This follows the principle of locality, i.e. it allows us to detect local features such us edges

[2] Adding the zeros to handle the border.

and corners. Further, it drastically improves the performance. Another related property is the sharing of parameters. The same set of filter parameters are used to compute the output on different locations. Therefore, the weights receive more gradient information from the backward pass and it reduces the storage requirements of the model. Finally, the model is equivariant, i.e. if the input changes the output changes in the same way. For example, if we translate the input of the convolution operator, then also the output is translated by the same amount.

The formulation of the convolution operation as a matrix product further enables the simple definition of another heavily used operation, the transposed convolution also known as deconvolution operation in the context of deep learning. As the name suggests it is defined as $\hat{f}(\mathbf{I}; \mathbf{W}) = \hat{\mathbf{W}}^T \mathbf{x}$ and in principle exchanges the forward and the backward pass of the two operations. This is especially interesting in the context of strided convolutions, where the loop variables of Equation (2.121) are incremented by an integer factor >1. In that case, the convolution operation decreases the spatial size of the output by the striding factor. Conversely, for the transposed convolution operation increases the spatial resolution of the output by the striding factor.

Pooling Operation Another important operation in convolutional networks is pooling. It allows us to make a convolutional network to some degree translation invariant. Hence, we care more about that a certain feature is present than exactly localizing it. The operation is again defined on a local neighborhood on which we compute summary statistics. The output for this local neighborhood is then this statistics and if we compute it in non-overlapping regions, the effective spatial resolution is decreased.

A common setting is to down-sample the input by a factor of 2. Therefore, we use a 2×2 neighborhood that is shifted by 2 pixels. See Figure 2.14b for a visualization of this example. In general, let us call the neighborhood size $P_H \times P_W$ and the shift, also called stride, S_H and S_W, respectively. Then we define the average-pooling $\operatorname{avg\,pool}$ and max-pooling $\max\operatorname{pool}$ as

$$\operatorname{avg\,pool}(\mathbf{I})_{h,w,c} = \frac{1}{P_H P_W} \sum_{p_h=1}^{P_H} \sum_{p_w=1}^{P_W} I_{(h-1)\cdot S_H + p_h, (w-1)\cdot S_W + p_w, c} \tag{2.123}$$

$$\max\operatorname{pool}(\mathbf{I})_{h,w,c} = \max_{1 \le p_h \le P_H, 1 \le p_w \le P_w} I_{(h-1)\cdot S_H + p_h, (w-1)\cdot S_W + p_w, c}\,. \tag{2.124}$$

Despite the invariance to small translation, pooling has two additional benefits if incorporated into the network graph. First, subsequent operations are performed on an intermediate representation of reduced resolution. Hence, those operations will require less memory and computational time, making the overall network faster to train and evaluate. Second, it increases the receptive field of subsequent convolutional operations. If we would apply only a list of convolutional operations, then the receptive field would only grow linearly. However, after a pooling operation that halves the resolution, the effective receptive field of the following convolutions is doubled with respect to the input.

2.3.6 Losses and Regularization

In this last chapter we will discuss some choices for the loss function $\mathcal{L}$ and network regularization Γ of the empirical risk function, see Equation (2.100).

Loss In the examples from the previous sections, we have already introduced and used the squared differences, also known as Euclidean loss. The loss is usually used for regression tasks, i.e. estimating continuous values, and is the maximum likelihood estimator assuming Gaussian noise [13]. However, in practice, the estimates of models trained with this loss tend to regress towards the mean. Therefore, we will often use an ℓ_p-norm with $p < 2$, most commonly the ℓ_1-norm, which can be defined as

$$\mathcal{L}_{\ell_1}(f(\mathbf{s}), \mathbf{t}) = \sum_{i=1}^{N} |f(\mathbf{s})_i - t_i| = \|f(\mathbf{s}) - \mathbf{t}\|_1 \,. \tag{2.125}$$

The gradient of the Euclidean loss function with respect to the input is simply given as the difference between estimate and target, i.e. $\mathbf{s} - \mathbf{t}$. In contrast, the gradient of the ℓ_1-loss function is $[|x_i| \neq 0]\frac{x_i}{|x_i|}$. Therefore, the gradient is -1, or 1 if the estimate and target differ, independently on how much they differ. This makes this loss robust to outliers. A compromise between squared differences and the ℓ_1-norm as loss function is utilizing the Huber norm as introduced in Equation (2.20). This loss function is also robust to outliers, but additionally, treats differences between estimate and target different in the ϵ interval.

If the task at hand is not a regression problem, then we have other choices for the loss function. For two-class classification, we can utilize the cross-entropy loss. Assume that the output of a network is o_k for a sample $\mathbf{s}_k$ and is further in the interval of $[0, 1]$. It can, therefore, be interpreted as a probability output. This can be achieved by using a sigmoid activation function on the output. Additionally, we have for each sample a target $t_k \in \{0, 1\}$. Then we can write the likelihood function using a Bernoulli distribution as

$$p(t_1, \cdots, t_K|\boldsymbol{\Theta}) = \prod_{k=1}^{K} o_k^{t_k}(1 - o_k)^{1-t_k} \,, \tag{2.126}$$

that we want to maximize. Taking the negative logarithm

$$-\ln p(t_1, \cdots, t_K|\boldsymbol{\Theta}) = -\sum_{k=1}^{K} t_k o_k + (1 - t_k)(1 - o_k) \,, \tag{2.127}$$

we derive the cross-entropy loss function $\mathcal{L}_{\text{BCE}}$ for independent binary distributed outputs

$$\mathcal{L}_{\text{BCE}}(f(\mathbf{s}), \mathbf{t}) = -\sum_{i=1}^{N} t_i f(\mathbf{s})_i + (1 - t_i)(1 - f(\mathbf{s})_i) \,. \tag{2.128}$$

Similarly, we can define a cross-entropy loss for multi-class classification. First, we assume that the output of the network is a valid probability distribution, i.e. $f(\mathbf{s}_k) \in [0, 1]$ and $\sum_{i=1}^{N} f(\mathbf{s}_k)_i = 1$. This can be achieved by using a softmax function $\text{softmax}(\mathbf{x})$ as the output activation function

$$\text{softmax}(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \,. \tag{2.129}$$

Additionally, we assume that the target vectors are probability vectors. Then, we can write the likelihood function as

$$p(\mathbf{t}_1, \cdots, \mathbf{t}_K | \boldsymbol{\Theta}) = \prod_{k=1}^{K} \prod_{i=1}^{N} f(\mathbf{s}_k)_i^{t_{k,i}} \,. \tag{2.130}$$

If we take the negative logarithm we derive

$$-\ln p(\mathbf{t}_1, \cdots, \mathbf{t}_K | \boldsymbol{\Theta}) = -\sum_{k=1}^{K} \sum_{i=1}^{N} t_{k,i} \ln f(\mathbf{s}_k)_i \,. \tag{2.131}$$

From this we obtain the cross-entropy loss function for multi-class classification $\mathcal{L}_{\text{MCE}}$ as

$$\mathcal{L}_{\text{MCE}}(f(\mathbf{s}), \mathbf{t}) = -\sum_{i=1}^{N} t_i \ln f(\mathbf{s})_i \,. \tag{2.132}$$

Regularization So far we have totally neglected the regularization term Γ in Equation (2.100). However, this term plays a crucial role to reduce over-fitting and therefore, minimizing the generalization error. We will now review a few strategies for the regularization of deep networks.

One simple way to constrain the model complexity of neural networks is to restrict the value range of its parameters $\boldsymbol{\Theta}$. If we assume a zero-mean Gaussian prior on the model parameters [13], then we can derive the squared ℓ_2 penalty

$$\Gamma(\boldsymbol{\Theta}) = \lambda \left\| \boldsymbol{\Theta} \right\|_2^2 \,, \tag{2.133}$$

where λ is a trade-off parameter with respect to the empirical loss over the training dataset and related to the standard deviation of the Gaussian prior. This regularization term is also known as weight decay, as it pushes the individual weight parameters towards zero. A similar effect can be obtained with different norms. If we would instead add an ℓ_1-norm on the parameters

$$\Gamma_{\ell_1}(\boldsymbol{\Theta}) = \lambda \left\| \boldsymbol{\Theta} \right\|_1 \,, \tag{2.134}$$

it would push some weight parameters towards zero, but not all. Hence, the parameters would become sparse.

Another common technique to prevent over-fitting, that is not restricted to neural networks, is dataset augmentation. A machine learning model will generalize better if more training data is available. However, training data is limited as it is often cumbersome to acquire large amounts of labeled data. A trick is therefore to augment the existing training data. For image data, this can include flipping and rotation of the images, or even slightly alter the colors via principal component analysis [132]. Those augmentations are dependent on the task, and one has to take care that the augmentations do not mess with the targets, e.g. rotating and flipping might produce different target labels. Also, the injection of small noise on the input [218] can be used to augment the training dataset.

If we train a large network, we often observe that the training error steadily decreases with the number of iterations. However, if we also monitor an additional error on some hold-out validation set, which the model never sees during training, we normally notice that this error decreases up to a certain point and increases then again. This is the point at which the network starts to over-fit to the training data. If we have access to such a validation set, we can stop the training at the point, where the validation error does not further decrease and take the parameters with the lowest validation error. This regularization technique is called early stopping. It can be seen as a regularization on the number of iteration steps, which is in fact also a hyper-parameter.

A quite recent technique to prevent over-fitting in deep networks is Dropout [105, 223]. The intuition behind this method is to train an ensemble of networks, like bagging [21] in random forests [23]. An ensemble of classifiers, where each classifier is trained on a subset of the training data typically improves the generalization performance. However, it comes with the cost of expensive training of several networks for one task. The trick in Dropout is to train an exponential number of networks by randomly dropping units, i.e. set their output to zero. For example, if we have an operation like $f(\mathbf{x}; \mathbf{W}) = \mathbf{W}\mathbf{x}$ in our network graph and call the output $\mathbf{y}$, then to apply Dropout we multiply $\mathbf{y}$ point-wise with a Bernoulli distributed random vector $\mathbf{b}$, i.e. $p(b_i = 0) = 1 - p(b_i = 1)$. We call $p(b_i = 0)$ the Dropout probability and it is usually set around 0.5. In classic bagging, we would then evaluate all models and compute the arithmetic mean to form a prediction. This is not feasible with the exponential number of implicit models that were trained with Dropout. Instead, we approximate the geometric mean by evaluating the model only once with all units, but scaling all weights by the probability that the corresponding unit has been dropped during training, i.e. $p(b_i = 0)$. Another interpretation of the weight scaling is that all units get the same expected total input during training and evaluation.

All the presented regularization methods are not mutually exclusive. In fact, most of these regularization techniques are used simultaneously in modern deep network optimization.

2.4 Summary

In Section 2.1 of this chapter we first introduced the basic notation and definitions that we will use throughout this thesis. This included normed and metric vector spaces and the operations defined on these spaces. Further, we defined the differential operator and its discretization on the 2D image space.

Those basics were necessary to give an self-contained presentation on convex optimization in Section 2.2 and its use in image processing in Section 2.2.3. Convex functions are a special class of functions with the nice property that each local minimum is a global minimum. This means that the optimization procedure cannot get stuck in a poor local minimum. Further, we have shown the optimal convergence rates of convex optimization schemes for smooth and non-smooth problems and stated concrete algorithms that achieve those rates. In low-level image processing, convex optimization can be utilized for efficiently solving variational energy functionals. In this chapter, we showed the application to denoising problems as we will rely on this formulation in the next chapter on depth super-resolution.

Finally, we introduced the field of supervised machine learning in Section 2.3 with an emphasize on deep learning. As the title of this thesis suggests, those topics are at the core of this work. In supervised machine learning, we assume that we have given a set of training examples with inputs and target outputs. The goal is then to find a function that can map previously unseen inputs to the target outputs. As we have shown, this is a non-trivial problem, where we have to deal with over- and under-fitting. However, one very successful class of methods are based on deep learning, e.g. learning deep artificial neural networks. We demonstrated the basic building blocks of deep networks as a non-linear generalization of linear regression. Further, we showed the section the extension to deep convolutional networks and how those can be optimized by stochastic gradient descent. The last topic was devoted to the regularization of deep networks.

CHAPTER 3

Deep Learning for 2.5D

Contents

3.1 Introduction . . . 53
3.2 Related Work . . . 56
3.3 Deep Learning Meets Variational Methods . . . 61
3.4 Evaluation . . . 70
3.5 Summary & Discussion . . . 99

The perception of depth or 2.5D information is important for a wide range of computer vision applications. However, due to physical constraints and the small package size required for mobile applications, depth sensors like the ones based on the Time-of-Flight principle have a low lateral resolution and are affected by high acquisition noise. Based on the techniques presented in the last chapter, we will introduce novel methods that combine traditional variational approaches with modern deep learning to increase the lateral resolution and the quality of depth maps.

3.1 Introduction

The perception of depth is a valuable clue for humans and animals to form an idea of distances and sizes. It helps them to safely navigate the world and avoiding dangers. This is one of the reasons why computing depth information is a fundamental problem in computer vision. The traditional approach is stereo: Given two images and the parameters of the camera set-up, i.e. rotation and translation between the two camera viewpoints, as well as the intrinsic parameters of the camera, we first find correspondences in the two images and then triangulate those points to infer depth. The crucial part is finding the

correspondences in the two images which relies on textured regions and is time-consuming in a dense setting.

However, the problem can be simplified with active stereo approaches by projection additional information into the scene. One set-up to get high accuracy depth maps is by using structured light [206]. A beamer sequentially projects vertical and horizontal stripes with different sizes into the scene, which are then utilized as features to ease the correspondence problem. A faster approach is to project only one random speckle pattern and utilize it in the matching as done in the Kinect [217]. Although these approaches enable a more accurate and faster depth acquisition, they still rely on searching correspondences. Depth sensors that avoid this completely are based on the Time-of-Flight principle, i.e. measuring the time difference between emitting a modulated light beam and receiving it on a sensor by computing the phase shift [99].

In the last decade depth sensors that are based on the latter two principles have been introduced to the mass market, e.g. Microsoft Kinect V2, Intel DepthSense, Creative Senz3D, or PMD Flexx. This elevated research towards the development of different applications that are based on this 2.5D information. Those active sensors enabled novel computer vision applications such as robot navigation [2], human pose estimation [87, 216, 243], and hand pose estimation [168, 167, 182, 230, 244]. Not only are those sensors comparable cheap, but they are also available in small package sizes and have low energy consumption. This makes them very attractive for mobile applications, such as the Google Tango project[1]. However, current sensors are limited by physical and manufacturing constraints. Hence, depth map outputs are affected by degenerations due to acquisition noise, quantization, and missing values, and typically have a low lateral resolution. Even very recent Time-of-Flight sensors, such as the PMD Flexx, or PMD Monstar have only a resolution of 224×171 pixels and 352×287 pixels, respectively[2].

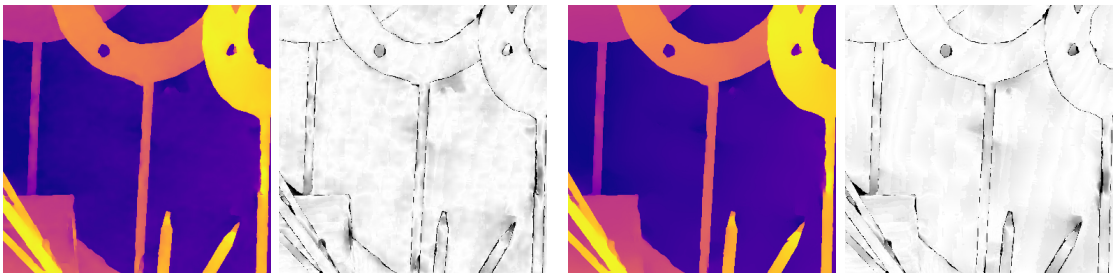
The task of computing a high-resolution image from a low-resolution input is known as the super-resolution problem in computer vision. If we denote the high-resolution image as $\mathbf{d}^{(\mathrm{hr})}$ and the low-resolution image as $\mathbf{d}^{(\mathrm{lr})}$, then we can relate those two quantities the image formation process

$$\mathbf{d}^{(\mathrm{lr})} = f(\mathbf{d}^{(\mathrm{hr})}) + n(\mathbf{d}^{(\mathrm{hr})}) \,, \tag{3.1}$$

where f describes the transformation from high-resolution space to low-resolution space. This includes a blur, as well as a down-sampling operation. Further, n describes the noise that is inherent of the acquisition process. If we assume a constant blur kernel and that the noise is independent of the image, then we can rewrite the equation in the following linear form

$$\mathbf{d}^{(\mathrm{lr})} = DB\mathbf{d}^{(\mathrm{lr})} + n \,, \tag{3.2}$$

[1] `https://get.google.com/tango`, last accessed on October 26, 2017
[2] `http://pmdtec.com/picofamily`, last accessed on October 26, 2017



(a) Network Output **(b)** Network Output Post-Processed

Figure 3.1: (a) A deep convolutional network is already able to produce very accurate depth map estimates for an up-sampling factor of $\times 4$. (b) Post-processing the network output by optimizing an appropriate energy functional further smooths the output. Left images show the depth estimates, right images the differences to the ground-truth.

where B is the blur matrix and D the down-sampling matrix. In this linear form, it can be easily observed that the problem is inherently ill-posed, as one low-resolution image can map to several high-resolution images.

The traditional approach to solve such problems is to regularize the solution space by incorporating prior knowledge [73, 232, 250]. In the domain of depth maps an additional data source is often utilized to constrain the solution space: Several of the available depth sensors are accompanied by high-resolution intensity or color cameras. Hence, it is common is to utilize this additional information from the high-resolution intensity images to guide the super-resolution process [52, 172, 266]. These approaches build upon the observation that depth discontinuities often occur at high-intensity variations and that homogeneous areas in intensity images are also more likely to represent homogeneous areas in depth. In other practical scenarios, however, the depth sensor is not always accompanied by an additional camera. Additionally, another drawback of those methods is, that the depth map has to be projected to the guidance image, which might be also problematic due to noisy depth measurements. Therefore, approaches that solely rely on the depth input for super-resolution are becoming popular [5, 70, 106].

The unguided approach to depth super-resolution bears similarities with the well-established field of single image super-resolution for natural images, where machine learning based methods [55, 207, 240, 239] are advancing rapidly and achieve impressive results on standard benchmarks. Those methods learn a mapping from a low-resolution input space to a plausible and visually pleasing high-resolution output space. However, while training data for natural images are abundant, this is not the case for depth data. This might be one of the reasons why only recently a concurrent deep learning based method [116] has been proposed for depth map super-resolution, where the authors used synthetic depth maps from the Sintel dataset [26] and disparity maps from Middlebury [205, 203] for training.

In this chapter, we present a novel method to increase the lateral resolution of the depth maps and jointly enhance the quality of them with respect to noise and other

degrading influences. It builds upon the recent success of deep learning methods applied to single image super-resolution for natural images [55, 126, 152]. We demonstrate how deep convolutional networks can be used for single depth map super-resolution and how the networks can be extended to incorporate high-resolution intensity images as additional guidance. However, in contrast to natural images, depth maps contain no textures and can be well characterized by affine surfaces with sharp depth discontinuities. This prior knowledge about depth maps can be formulated in an energy functional. In Figure 3.1 we show how the minimizer of such an energy functional can further improve the already good result of a deep network. We show in this chapter, how the optimization scheme can be incorporated into a deep network and trained end-to-end, leading to superior results. Finally, to train such deep network architectures we need sufficient training data. We tackle this problem by generating highly realistic synthetic data in a semi-automatic way by using a physically based renderer [119]. This enables us to produce depth maps along with intensity images that are pixel aligned and even contain artifacts such as shading.

The remainder of this chapter is organized as follows: We present a novel method that combines a deep fully convolutional network and a non-local primal-dual network that is trained end-to-end in Section 3.3. Hence, we map a noisy, low-resolution depth map along with an optional high-resolution guidance image to an accurate high-resolution estimate. We propose a framework based on the physically based renderer to automatically generate high-quality depth maps with corresponding color images in large quantities which are used to train our model in Section 3.3.3. Our evaluations in Section 3.4 demonstrate the effectiveness of our method by outperforming state-of-the-art results on a set of standard synthetic and real-world benchmarks.

3.2 Related Work

In this section, we review the related literature on image super-resolution in general, and on depth super-resolution in general. Additionally, we include an overview of methods that propose to learn deep networks and energy minimization methods in an end-to-end fashion.

3.2.1 Super-Resolution

The research area on image super-resolution can be roughly divided into two branches regarding the input. First, there exist super-resolution methods that rely on multiple-camera set-up, or on a video sequence [66, 213]. Those approaches mostly rely on a sub-pixel accurate alignment of the images to infer the missing values of the high-resolution image. However, the accurate alignment of the images is a difficult problem in itself, where one has to deal with moving objects, occlusions and other ambiguities. The performance degrades rapidly if the upscaling factor increases and the number of low-resolution images is insufficient to constrain the problem. In this work, we focus on the second branch,

namely single image super-resolution. Single image super-resolution methods try to recover the high-resolution image from a single low-resolution input image. To handle the ill-posed nature of this problem, those methods rely either on some sort of image prior or on a mapping from low-resolution domain to high-resolution domain.

Early works on single image super-resolution utilized prior information, i.e. smoothness assumption of natural images. Tappen et al. [232] used the observation that gradients in natural images are sparse and can be approximated by a Laplace distribution. They formulated a Markov random field with a Laplace distribution on the image gradients as prior and the super-resolution is performed as inference in this graphical model. A variational method with Huber norm as image prior was used by Unger et al. [250]. The authors argued that the Huber norm favors strong edges, similar to the Total Variation, but smooths over gray values.

Instead of using a fixed prior in an energy minimization framework, other approaches rely on a mapping from low-resolution to high-resolution patches. The mapping can either be learned or can be established by searching in a candidate set. The latter approach was proposed by Freeman et al. [73], where the authors utilized synthetic scenes to learn a graphical model that associates low-resolution with high-resolution patches. However, the candidate high-resolution patches do not have to stem from an external database. Glasner et al. [88] showed in their seminal work that the same patch often occurs on different scales within the same image. Therefore, the authors proposed to search for high-resolution patch candidates within the same image and used those candidates to super-resolve the low-resolution image. This technique has been recently improved by Huang et al. [109]. The authors proposed to extend the internal patch search space by including affine transformation on the candidate patches. Neighbour embedding methods are also a popular approach to single image super-resolution. Those methods assume that the high-resolution images and its low-resolution counterparts lie on a non-linear manifold with similar geometry. Chang et al. [31] utilized this observation by using the Locally Linear Embedding [197] method to project the low-resolution test patches and high-resolution training patches to a non-linear manifold. The high-resolution test patches are then reconstructed by a weighted average of the high-resolution training patches that are in the local neighborhood of the manifold. A drawback of this method is that the dictionary of training patches needed for the reconstruction increases with the number of training patches. Therefore, Yang et al. [263] proposed a sparse coding based approach to learn a compact dictionary of sparse signal representations. The authors proposed to jointly learn the sparse dictionaries of the low- and high-resolution patches, such that a given low-resolution patch has a similar sparse representation as a high-resolution patch. The learned dictionaries have a fixed size, but the reconstruction of the super-resolved images now involves a time-consuming optimization procedure. Zeyde et al. [270] improved upon this work by using a more efficient dictionary learning approach [1], as well as Orthogonal Matching Pursuit [246] for the sparse coding step. The optimization step in the sparse coding approaches is still very time-consuming. Timofte et al. [240] addressed this

problem by replacing the ℓ_1 norm with an ℓ_2 norm, and substituted the big single dictionary with several smaller ones. Hence, the optimization problem can be solved in closed form, resulting in significant speed improvements. The same authors further improved their method in [239] yielding an even faster inference speed and better accuracy. Another approach has been investigated by Schulter et al. [207]. The authors formulated the task as a locally linear regression problem, where the low-resolution patches are mapped to a regression matrix, which is then applied to yield the high-resolution patch estimate. The mapping from low-resolution patch to regression matrix is learned by a Random Forest.

Currently, the most powerful single image super-resolution methods are based on deep convolutional networks. Dong et al. [55] proposed one of the first deep networks for this task. In their work, the authors also show the parallels to sparse coding approaches. Interestingly, they showed in their experiments that a three-layer network performs better than slightly deeper networks on this problem. Another drawback of the method is that it requires a pre-processing step, where the low-resolution image is up-sampled via bicubic interpolation to a so-called mid-resolution input. This problem was addressed by Shi et al. [215] by presenting a sub-pixel convolutional network. The authors proposed to apply two convolutional layers on the low-resolution input and a final convolution outputs ρs feature maps, where ρ is the up-sampling factor. Then, the feature maps are combined to form a high-resolution estimate. This operation that combines the feature maps is also called pixel-shuffle in other works. Kim et al. [126] demonstrated how deeper networks can be trained on this problem and significantly improved performance. A key technique was to train on the residual output instead on the high-resolution target, whereas the residual output is the difference between mid-resolution input and high-resolution target. The same authors presented a concurrent technique to train deep network on the super-resolution problem. In [127] they proposed to apply a convolutional layer recursively and reconstructing the high-resolution image after each iteration. The final estimate is then the average of all intermediate reconstructions. One of the currently best-performing networks it the one by Mao et al. [152]. The authors proposed a 30 layer encoder-decoder network, where the output encoder convolutional layers are added to the inputs of the decoder convolutional layers.

3.2.2 Depth Super-Resolution

Similar to the related work on single image super-resolution for natural images can be divided by the input, the same is true for depth super-resolution. The first branch of approaches uses several depth maps from various viewpoints to increase the lateral resolution and enhance the depth quality. In contrast to natural images, the pose estimation problem is easier given depth maps than from color images. Schuon et al. [209, 210] formulated this fusion problem from multiple viewpoints as a Markov Random Field. Related to this problem is the fusion of multiple depth maps in a 3D volume. Curless and Levoy [45] formulated the integration of truncated signed distance functions in a predefined 3D volume

to get a dense reconstruction. This approach has been extended in the seminal KinectFusion work by Izadi et al. [118], where the authors track the camera viewpoint in real-time and integrate the depth maps in such a truncated signed distance volume.

Several applications, however, need a very low latency to retrieve the enhanced depth map, like in pose estimation tasks. For those applications, super-resolution methods that only take a single depth map as input are more relevant. This branch of depth super-resolution methods can be further divided into methods that use an additional high-resolution intensity image as guidance, and methods that only use the low-resolution depth map as input.

Guided Depth Super-Resolution Guided depth super-resolution methods use an additional high-resolution intensity image as input, assuming that most depth sensors are accompanied by an intensity or color camera. Those methods rely on the statistical co-occurrence of gradients in natural images and depth discontinuities in depth maps. Therefore, strong gradients in a high-resolution depth map most likely occur if there exists also a gradient in the high-resolution guidance image.

One of the first works in this direction was by Diebel and Thrun [52]. They utilized a Markov Random Field for the up-sampling task and their smoothness term is weighted according to the gradients of the guidance image. Similarly, Kopf et al. [130] proposed a method that leverages a joint bilateral filter [242] for the guided depth super-resolution problem. They apply a Gaussian filter to smooth the up-sampled depth map, whereas the filter is dependent on the additional high-resolution intensity image such that the smoothing does not blur over edges. Yang et al. [266] also proposed an approach based on a bilateral filter that is iteratively applied to estimate a high-resolution output depth map. An additional guided filter for the guided depth super-resolution problem was proposed by Shen et al. [214], where they take the mutual structure of the intensity image and depth map into account to avoid false depth edges. Park et al. [172] presented a least-squares-based method that incorporates an edge-aware weighting schemes in the regularization term of their formulation. A variational formulation for guided depth super-resolution was proposed by Ferstl et al. [67]. The low-resolution depth map is up-sampled to produce a sparse high-resolution depth map and then interpolated using a weighted Total Generalized Variation prior [20]. Yang et al. [264] showed that auto-regressive models can be efficiently applied to this problem. The bilateral solver proposed by Barron and Poole [9] is a generalization of the bilateral filter. The authors showed that it can be implemented very efficiently while producing comparable results on this task.

Unguided Depth Super-Resolution Recently, there is an increased interest in depth super-resolution methods that do not rely on an additional high-resolution guidance image. Those methods share a lot of similarities with single image super-resolution methods for natural images. They either rely on an external database to search for candidate patches [5], search similar high-resolution patches within the depth map itself [106], or learn a mapping from low- to high-resolution depth maps [70, 116, 134].

Aodha et al. [5] formulated a Markov Random Field framework similar to [73] by using an external database to search candidate high-resolution patches. Additionally, they introduced a depth normalization step that smoothes the overlap between estimated high-resolution patches. Instead of an external database, Hornáček et al. [106] searched for high-resolution patch candidates within the low-resolution depth map in the spirit of [88]. Further, the patch candidates are not simply searched as 2D image patches, but as patches containing 3D points that can be translated and rotated in 3D. Ferstl et al. [70] introduced a learning-based approach to depth super-resolution. The authors used sparse coding with dictionaries trained on the 31 synthetic depth maps of [5] to predict the depth discontinuities in the high-resolution domain from the low-resolution depth data. Those edge estimates are then used in an anisotropic diffusion tensor of their regularization term within a variational framework. A direct mapping from low- to high-resolution depth patches was trained by Kwon et al. [134]. They trained a multi-scale sparse coding approach to up-sample the low-resolution depth map step-wise, whereas the training data was obtained by recording scenes with KinectFusion [118]. Hui et al. [116] trained a convolutional network on this problem. They used the small number of Middlebury disparity maps [205, 203] and synthetic depth maps from the Sintel dataset [26] as ground-truth.

3.2.3 Joint Network Training and Energy Minimization

Energy minimization methods, such as Markov Random Fields, or variational methods as described in Section 2.2.3 have a wide range of applications in computer vision. Even for deep convolutional networks, those energy minimization based methods are a popular post-processing tool, e.g. for semantic segmentation [34], or stereo [269]. Simply put, they consist of unary terms, for example, the class likelihood of a pixel for semantic segmentation, or the depth value in depth super-resolution, and pairwise terms, which encourage the smoothness between neighboring pixels. Recently, the integration of those models into deep networks gained a lot of attention, as deep networks jointly trained with energy minimization methods achieve excellent results. For example, Tompson et al. [243] proposed the joint training of a convolutional network and a Markov Random Field for human pose estimation. The Markov Random Field is realized by very large convolutional filters to model the pairwise interactions between joints and can be interpreted as one iteration of loopy belief propagation. Baltrušaitis et al. [7] used a perceptron to parametrize the potential of a Gaussian Random Field which is used as patch-expert in subsequent applications. Given the simple Gaussian form of the Random Field, the single-layer network can be learned by gradient-based log-likelihood maximization. In [36, 211, 261] the authors showed how to compute the derivative with respect to the mean field approximation [131] in Markov Random Fields. This allows end-to-end learning and improves results for instance in semantic segmentation. Similarly, Zheng et al. [273] showed that the computation steps of the mean field approximation can be modeled by operations of a convolutional network and unrolled those iterations on top of their network. The latter

approach is very related to automatic differentiation with reverse accumulation [144, 143], where the Backpropagation algorithm [199] presented in Section 2.3.4 is a special case of. Prior to the deep learning area Tappen [231] and Domke [53, 54] proposed an automatic differentiation method to learn the model parameters of a Markov Random Field for Mean Field and Tree-Reweighted Belief Propagation.

While the latter approaches are designed for a discrete label space, the variational approach by Ranftl and Pock [177] has a continuous output space. They showed that the gradient of a loss function can be backpropagated through the energy functional of a variational method by implicit differentiation, given that functional is smooth enough. Recently, Ochs et al. [169, 170] proposed a technique that enables the backpropagation through non-smooth energy functionals. The method is also very related to automatic differentiation, with the contrast that the optimization algorithm of the variational energy functional can be smoothed using Bregman proximity functions [28]. This allows a memory efficient implementation of the gradient computations.

3.2.4 Datasets with Ground-Truth Depth

An important factor to train deep networks is the amount of training data with annotated ground-truth. For methods that tackle low-level vision problems on natural images such as presented in Section 3.2.1, training data can be easily obtained. High-quality images that can serve as ground-truth are massively available on the internet, and the input data is then generated by applying the easier forward problem, i.e. down-sample the image. This is unfortunately not the case for depth images. While there exist also datasets that contain depth along color images such as the NYU Depth Dataset v2 [219], SceneNN [108], or the KITTI dataset [79, 80], the depth data in those datasets is mainly used to serve as auxiliary input for other tasks and can not be used as ground-truth because it is either too noisy, or too sparse. However, there appeared also large synthetic datasets recently for the same tasks [76, 96, 155, 195] that might be interesting for training.

3.3 Deep Learning Meets Variational Methods

In this section we formulate the depth super-resolution problem as supervised learning task. As described in Section 2.3.1 we assume that we have given a training dataset $\mathcal{D} = \{(\mathbf{s}_k, \mathbf{t}_k)\}_{k=1}^K$ with K training samples. A sample $\mathbf{s}_k$ consists either of a single low-resolution depth map $\mathbf{s}_k = \mathbf{d}_k^{(\mathrm{lr})}$, or a low-resolution depth map in combination with an associated high-resolution guidance image, $\mathbf{s}_k = (\mathbf{d}_k^{(\mathrm{lr})}, \mathbf{g}_k)$. The targets correspond to the high-resolution depth maps, i.e. $\mathbf{t}_k = \mathbf{d}_k^{(\mathrm{hr})}$. The goal of the supervised learning problem is then to find a function $f^* \in \mathcal{F}$ that minimizes the empirical loss

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}}^{K} \mathcal{L}(f(\mathbf{s}_k), \mathbf{t}_k)\,, \tag{3.3}$$

where $\mathcal{L}$ is a suitable loss function that penalizes deviates of the high-resolution depth estimate $f(\mathbf{s}_k)$ from the ground-truth target $\mathbf{t}_k$. We could apply the deep learning techniques described at length in Section 2.3. However, as described in the introduction of this chapter and shown in Figure 3.1 a simple variational method could further improve the network's estimate. More specifically, assume that $\mathbf{u}_k^*$ is the minimizer of a variational functional

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \Gamma(\mathbf{u}) + \lambda \Psi(\mathbf{u}, f^*(\mathbf{s}))\,, \tag{3.4}$$

with a proper regularization term $\Gamma(\cdot)$ and data term $\Psi(\cdot,\cdot)$. Then, we can observe that

$$\mathcal{L}(\mathbf{u}^*, \mathbf{t}_k) \leq \mathcal{L}(f^*(\mathbf{s}), \mathbf{t})\,, \tag{3.5}$$

holds for most of the test samples $(\mathbf{s}, \mathbf{t})$.

Our motivation is now to integrate the variational method on top of a deep network for two main reasons. First, this enables the network to adapt its output to the subsequent variational functional minimization that produces better results with respect to the loss function. And second, if the variational method is incorporated into supervised learning scheme, then we can also train hyper-parameters of that model, like the trade-off parameter λ, or step sizes of the optimization algorithm.

A principled way to train both quantities, the deep network and the variational model, in a joint framework is to formulate it as a bi-level optimization problem [177, 184]. We define $\mathbf{\Theta}_n$ as the network parameters, i.e. weights and biases of the network f, $\mathbf{\Theta}_v$ as the parameters of a variational model E and $\mathbf{\Theta} = \{\mathbf{\Theta}_n, \mathbf{\Theta}_v\}$. Then the bi-level optimization problem is given by

$$\min_{\mathbf{\Theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \mathcal{L}(\mathbf{u}^*(f(\mathbf{s}_k; \mathbf{\Theta}_n)), \mathbf{t}_k) \tag{HL}$$

$$\text{s.t.} \quad \mathbf{u}^*(f(\mathbf{s}_k)) = \arg\min_{\mathbf{u}} E(\mathbf{u}; f(\mathbf{s}_k; \mathbf{\Theta}_n))\,. \tag{LL}$$

We call (HL) the higher-level problem and (LL) the lower-level problem. This formulation of the training problem has the following intuitive interpretation: The task of the training procedure is to find parameters $\mathbf{\Theta}$ for the energy $E(u; f(\mathbf{s}_k; \mathbf{\Theta}_n))$, such that the minimizer $\mathbf{u}^*(f(\mathbf{s}_k; \mathbf{\Theta}_n))$ of the energy yields low training loss $\mathcal{L}(\mathbf{u}^*(f(\mathbf{s}_k; \mathbf{\Theta}_n), \mathbf{t}_k))$. Note that if the (LL) problem is assumed to be of the form

$$E(\mathbf{u}, f(\mathbf{s}_k; \mathbf{\Theta}_n)) = \Gamma(\mathbf{u}, f_r(\mathbf{s}_k; \mathbf{\Theta}_n)) + \exp(\lambda) \Psi(\mathbf{u}, f_d(\mathbf{s}; \mathbf{\Theta}_n))\,, \tag{3.6}$$

where the network $f = (f_r, f_d)$ may influence the regularization term and the data term, and $\exp(\lambda)$ with $\lambda \in \mathbf{\Theta}_v$ ensures a positive trade-off parameter and hence, the convexity of (LL).

In the remainder of this section, we will present two methods to tackle the bi-level optimization problem. First, we will discuss a more theoretically grounded approach using implicit differentiation in Section 3.3.1. Then, we will show a more practical method in Section 3.3.2 that we will use in the evaluation to show state-of-the-art results.

3.3.1 Implicit Differentiation

Bi-level optimization problems are very challenging in practice due to their highly non-convex nature [50]. However, if we constrain the regularization and data term, an efficient optimization is possible. The following proposition provides conditions which will allow us to compute gradients of the (HL) problem, and simultaneously satisfy the constraint given by the (LL) problem, even for large-scale problems.

Proposition 1. Let $E(\mathbf{u}; f(\mathbf{s}_k; \boldsymbol{\Theta}_n))$ be strongly convex and twice differentiable with respect to $\mathbf{u}$. Further, let $E(u; f(\mathbf{s}_k; \boldsymbol{\Theta}_n))$ be differentiable with respect to f and let $f(\mathbf{s}_k; \boldsymbol{\Theta}_n)$ be differentiable with respect to $\boldsymbol{\Theta}_n$. Then, the gradient of a differentiable loss $\mathcal{L}$ with respect to the parameters $\boldsymbol{\Theta}_n$ is well-defined and is given by

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Theta}} = - \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \left(\left[\left(\nabla^2_{\mathbf{u}} E \right)^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{u}_k} \right]^T \frac{\partial^2 E}{\partial \mathbf{u} \partial \boldsymbol{\Theta}} \right) \Bigg|_{\mathbf{u}_k = \mathbf{u}_k^*} . \tag{3.7}$$

Proof. The bi-level problem (HL)-(LL) can equivalently be rewritten in terms of the optimality conditions of the lower-level problem

$$\begin{aligned} &\min_{\boldsymbol{\Theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \mathcal{L}(\mathbf{u}_k, \mathbf{t}_k) \\ &\text{s.t.} \quad \nabla_{\mathbf{u}_k} E(\mathbf{u}_k; f(\mathbf{s}_k; \boldsymbol{\Theta}_n)) = 0 \,. \end{aligned} \tag{3.8}$$

Note that we will omit the explicit dependence of $E(\mathbf{u}_k; f(\mathbf{s}_k; \boldsymbol{\Theta}_n))$ on the parametrization $f(\mathbf{s}_k; \boldsymbol{\Theta}_n)$ for the rest of the proof in order to facilitate an uncluttered notation.

Problem (3.8) is an optimization problem with non-linear equality constraints. The Lagrangian of this function is given by

$$L(\mathbf{u}, \boldsymbol{\Theta}, \gamma) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \mathcal{L}(\mathbf{u}_k, \mathbf{t}_k) + \gamma_k (\nabla_{\mathbf{u}} E(\mathbf{u}_k)). \tag{3.9}$$

The stationary points of the Lagrangian are characterized by the optimality conditions:

$$\frac{\partial L}{\partial \mathbf{u}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}_k} + \gamma_k \nabla^2_{\mathbf{u}} E(\mathbf{u}_k) = 0 \tag{C1}$$

$$\frac{\partial L}{\partial \boldsymbol{\Theta}_n} = \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \gamma_k^T \frac{\partial^2 E(\mathbf{u}_k)}{\partial \mathbf{u}_k \partial \boldsymbol{\Theta}_n} = 0 \tag{C2}$$

$$\frac{\partial L}{\partial \gamma_k} = \nabla_{\mathbf{u}} E(\mathbf{u}_k) = 0\,. \tag{C3}$$

By substituting the minimizer of the lower-level problem $\mathbf{u}_k^*(f(\mathbf{s}_k;\boldsymbol{\Theta}_n))$ for $\mathbf{u}_k$, condition (C3) can be eliminated, since it is fulfilled by definition. From strong convexity of the energy $E(\mathbf{u})$, it follows that $\nabla^2_{\mathbf{u}} E(u) \succ 0$. Thus, the Lagrange multipliers γ_k can be explicitly computed from (C1), which results in

$$\gamma_k = -(\nabla^2_{\mathbf{u}} E)^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{u}_k}. \tag{3.10}$$

Substituting (3.10) into (C2) finally yields the gradient (3.7). □

Remark 1. Note that

$$\frac{\partial^2 E(\mathbf{u}_k)}{\partial \mathbf{u}_k \partial \boldsymbol{\Theta}_n} = \frac{\partial^2 E(\mathbf{u}_k; f(\mathbf{s}_k;\boldsymbol{\Theta}_n))}{\partial \mathbf{u}_k \partial f} \frac{\partial f(\mathbf{s}_k;\boldsymbol{\Theta}_n)}{\partial \boldsymbol{\Theta}_n}, \tag{3.11}$$

which shows that a necessary condition for the computation of the gradient (3.7) is differentiability with respect to the parametrization $f(\mathbf{s}_k;\boldsymbol{\Theta}_n)$.

An interesting property of this scheme is that it can be interpreted as a step in the backpropagation algorithm: If the parametrization is a deep network, then the gradient for a single training example can be computed, by backpropagating the quantity

$$\Delta E = -\left(\left[\left(\nabla^2_{\mathbf{u}} E\right)^{-1} \frac{\partial L}{\partial \mathbf{u}_k} \right]^T \frac{\partial^2 E}{\partial \mathbf{u}_k \partial f} \right) \Bigg|_{\mathbf{u}_k = u_k^*}. \tag{3.12}$$

This enables the implementation of the variational model as a single additional layer in the network.

To use this framework for depth super-resolution we have to define the variational model. If we assume that the network already produces a reasonable high-resolution depth estimate $f_d(\mathbf{s}_k;\boldsymbol{\Theta}_n)$, then we can use simple squared differences as data term that penalizes deviations from this initial estimate, i.e. $\Psi(\mathbf{u}, f_d(\mathbf{s};\boldsymbol{\Theta}_n)) = \frac{1}{2}\left\|\mathbf{u} - f_d(\mathbf{s}_k;\boldsymbol{\Theta}_n)\right\|_2^2$. The regularization term should smooth depth values within depth discontinuities, but should not smooth over them. Further, to use the regularization term within the implicit differentiation framework we have to fulfill the constraints of Proposition 1. Therefore, we define the regularization term as

$$\Gamma(\mathbf{u}, f_r(\mathbf{s};\boldsymbol{\Theta}_n)) = \left\| f_r(\mathbf{s};\boldsymbol{\Theta}_n) \nabla \mathbf{u} \right\|_s . \tag{3.13}$$

The network output f_r guides the regularization term where it should be more strongly enforced (smooth regions) and where it should be neglected (depth discontinuities), and $\|\cdot\|$ is a smooth norm function. One option is the Charbonnier approximation $\|\cdot\|_{1_s}$ [32]

to the ℓ_1 norm

$$\|\mathbf{x}\|_{1_s} = \sum_i \sqrt{x_i^2 + \epsilon^2}\,. \tag{3.14}$$

Another option is the twice-differentiable smooth approximation to the Huber norm [133] that avoids the stair-casing effect

$$\|\mathbf{x}\|_{\epsilon_s} = \sum_i \begin{cases} -\frac{1}{8\epsilon^3}x_i^4 + \frac{3}{4\epsilon}x_i^2 + \frac{3\epsilon}{8} & \text{if } |x_i| \leq \epsilon \\ |x_i| & \text{else} \end{cases}. \tag{3.15}$$

The lower-level energy functional is then given by

$$E(\mathbf{u}; f(\mathbf{s}_k; \boldsymbol{\Theta}_n)) = \|f_r(\mathbf{s}; \boldsymbol{\Theta}_n)\nabla\mathbf{u}\|_s + \frac{\exp(\lambda)}{2}\|\mathbf{u} - f_d(\mathbf{s}; \boldsymbol{\Theta}_n)\|_2^2\,. \tag{3.16}$$

In order to conveniently derive the gradient computations, we reformulate the energy functional in Equation (3.16) using a matrix-vector notation. Therefore, we define $\mathbf{W}$ and $\mathbf{f}$ to correspond to $f_r(\mathbf{s}; \boldsymbol{\Theta}_n)$ and $f_d(\mathbf{s}; \boldsymbol{\Theta}_n)$, respectively. Further, for ease of notation we set $\gamma(\cdot) = \|\cdot\|_s$ such that finally the regularization term can be written as

$$\gamma(\mathbf{W}\nabla\mathbf{u}) = \|f_r(\mathbf{s}; \boldsymbol{\Theta}_n)\nabla)\mathbf{u}\|_s\,. \tag{3.17}$$

Using this notation, Equation (3.16) is equivalent to

$$E(\mathbf{u}; f(\mathbf{s}_k; \boldsymbol{\Theta}_n)) = \gamma(\mathbf{W}\nabla\mathbf{u}) + \frac{\exp(\lambda)}{2}(\mathbf{u} - \mathbf{f})^T(\mathbf{u} - \mathbf{f})\,, \tag{3.18}$$

and the gradients with respect to E are given by

$$\nabla_{\mathbf{u}}E(\mathbf{u}) = \mathbf{W}\nabla\mathbf{D}'\mathbf{W}\nabla\mathbf{u} + \exp(\lambda)(\mathbf{u} - \mathbf{f}) \tag{3.19}$$

$$\nabla_{\mathbf{u}}^2E(\mathbf{u}) = \nabla^T\mathbf{W}\mathbf{D}''\mathbf{W}\nabla + \exp(\lambda)I \tag{3.20}$$

$$\frac{\partial^2 E(\mathbf{u})}{\partial\mathbf{u}\partial\lambda} = \exp(\lambda)(\mathbf{u} - \mathbf{f}) \tag{3.21}$$

$$\frac{\partial^2 E(\mathbf{u})}{\partial\mathbf{u}\partial\mathbf{f}} = -\exp(\lambda) \tag{3.22}$$

$$\frac{\partial^2 E(\mathbf{u})}{\partial\mathbf{u}\partial\mathbf{W}} = \mathbf{D}'\nabla + \operatorname{diag}(\nabla\mathbf{u})\mathbf{D}''\mathbf{W}\nabla\,, \tag{3.23}$$

with

$$\mathbf{D}' = (\gamma'((\mathbf{W}\nabla\mathbf{u})_1), \ldots, \gamma'((\mathbf{W}\nabla\mathbf{u})_M))^T \tag{3.24}$$

$$\mathbf{D}'' = \operatorname{diag}(\gamma''((\mathbf{W}\nabla\mathbf{u})_1), \ldots, \gamma''((\mathbf{W}\nabla\mathbf{u})_M))\,. \tag{3.25}$$

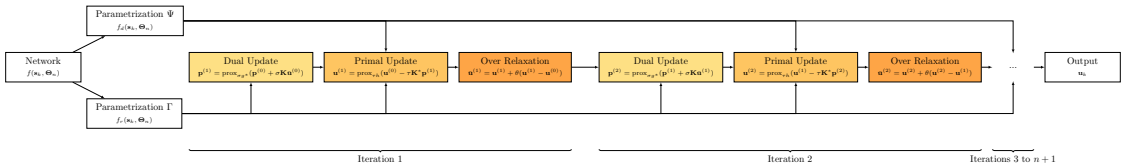


Figure 3.2: Computation Graph of the Unrolled Optimization Algorithm 6. The input to the optimization algorithm is a first high-resolution estimate from the convolutional network $f_d(\mathbf{s}_k; \mathbf{\Theta}_n)$, as well as a parametrization of the regularization term $f_r(\mathbf{s}_k; \mathbf{\Theta}_n)$.

3.3.2 Unrolling Optimization Scheme

In the previous section, we presented an approach to solve the bi-level optimization problem (HL)-(LL) by implicit differentiation of the lower-level problem. While this yields a nice mathematically founded solution, it has a few practical shortcomings. First, the energy functional has to satisfy the assumptions in Proposition 1. Most of them are unproblematic, like a differentiable loss function, or a differentiable parametrization. Those assumptions will be satisfied by design, as otherwise, the training of the deep network parametrization will not be possible at all. However, it also assumes a smooth energy functional. For this purpose, we presented smooth approximations to two common regularization terms, but it would be even better to use stronger terms that are non-smooth, like the Total Generalized Variation [20]. Another drawback of this method is slightly hidden. In the derivation of the gradient with respect to the energy functional, we assumed that we are able to compute the exact minimizer $\mathbf{u}^*$. While it is possible to utilize the optimization algorithms presented in Section 2.2.2.1 and Section 2.2.2.2, the solutions are still not exactly $\mathbf{u}^*$. Especially, if one uses a fixed number of iterations. This introduces noise into the gradients and slows down training [184].

In this section, we present a different approach that is based on automatic differentiation [143, 144]. Instead of defining the gradient with respect to the energy functional, we can also compute the gradient with respect to the individual operations of an optimization algorithm. More specifically, we can unroll the steps of the optimization algorithm, i.e. as individual layers of a deep network to compute the forward solution and then backpropagate through these operations similar as in deep network. See Figure 3.2 for a visual depiction of the concept.

Therefore, we rewrite the bi-level optimization problem (HL)-(LL) as

$$\begin{aligned} &\min_{\mathbf{\Theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_k, \mathbf{t}_k) \in \mathcal{D}} \mathcal{L}(\mathbf{u}_k, \mathbf{t}_k) \\ &\text{s.t.} \quad \mathbf{u}_k = \mathcal{P}^{(\mathrm{i}+1)}(f(\mathbf{s}_k; \mathbf{\Theta}_n)) \,, \end{aligned} \tag{3.26}$$

where $\mathcal{P}^{(\mathrm{i}+1)}(f(\mathbf{s}_k; \mathbf{\Theta}_n))$ is a suitable iterative optimization algorithm for the energy functional $E(\mathbf{u}; f(\mathbf{s}_k; \mathbf{\Theta}_n))$. More precisely, it is the output of the optimization algorithm after

running $i+1$ iterations. We do not assume that the optimization algorithm is converged after $i+1$ iterations, but that the high-resolution estimate has improved with respect to the loss $\mathcal{L}$. Further, the network f will adapt during the joint training to the optimization algorithm. In addition, we are able to also train all hyper-parameters of the energy functional and optimization algorithm.

In the remainder of the section, we assume that the optimization algorithm $\mathcal{P}$ is given by the primal-dual algorithm presented in Algorithm 6. Despite the initialization, the algorithm consists of three steps: (1) update of the dual variables (DU), (2) update of the primal variables (PU), and (3) an over-relaxation step (OR).

$$\mathbf{p}^{(\mathrm{i}+1)} = \operatorname{prox}_{\sigma g^*}(\mathbf{p}^{(\mathrm{i})} + \sigma \mathbf{K} \bar{\mathbf{u}}^{(\mathrm{i})}) \tag{DU}$$

$$\mathbf{u}^{(\mathrm{i}+1)} = \operatorname{prox}_{\tau h}(\mathbf{u}^{(\mathrm{i})} - \tau \mathbf{K}^* \mathbf{p}^{(\mathrm{i}+1)}) \tag{PU}$$

$$\bar{\mathbf{u}}^{(\mathrm{i}+1)} = \mathbf{u}^{(\mathrm{i}+1)} + \theta(\mathbf{u}^{(\mathrm{i}+1)} - \mathbf{u}^{(\mathrm{i})})\,. \tag{OR}$$

To solve the optimization problem in Equation (3.26) we need to compute the gradient of the loss with respect to the parameters $\frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_n}$. Using the chain rule we can rewrite the derivative as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_n} = \frac{\partial \mathcal{L}}{\partial \mathcal{P}^{(\mathrm{i}+1)}} \frac{\partial \mathcal{P}^{(\mathrm{i}+1)}}{\partial \mathcal{P}^{(\mathrm{i})}} \frac{\partial \mathcal{P}^{(\mathrm{i})}}{\partial \mathcal{P}^{(\mathrm{i}-1)}} \cdots \frac{\partial \mathcal{P}^{(2)}}{\partial \mathcal{P}^{(1)}} \frac{\partial \mathcal{P}^{(1)}}{\partial f} \frac{\partial f}{\partial \mathbf{\Theta}_n}\,, \tag{3.27}$$

which shows the applicability of the backpropagation algorithm to train the network with the optimization algorithm on top. For the primal-dual scheme the gradient of a single iteration $\frac{\partial \mathcal{P}^{(\mathrm{i})}}{\partial \mathcal{P}^{(\mathrm{i}-1)}}$ can be further expanded

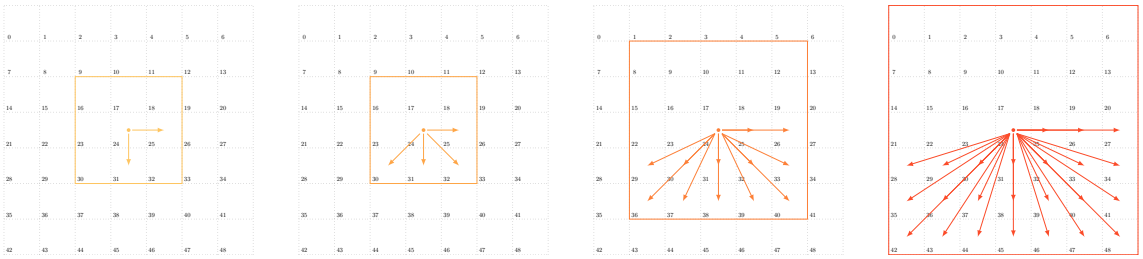
$$\frac{\partial \mathcal{P}^{(\mathrm{i})}}{\partial \mathcal{P}^{(\mathrm{i}-1)}} = \frac{\partial \mathcal{P}^{(\mathrm{i})}}{\partial \bar{\mathbf{u}}^{(\mathrm{i})}} \frac{\partial \bar{\mathbf{u}}^{(\mathrm{i})}}{\partial \mathbf{u}^{(\mathrm{i})}} \frac{\partial \mathbf{u}^{(\mathrm{i})}}{\partial \mathbf{p}^{(\mathrm{i})}} \frac{\partial \mathbf{p}^{(\mathrm{i})}}{\partial \mathcal{P}^{(\mathrm{i}-1)}}\,, \tag{3.28}$$

which reveals the nice property that each update step can be realized as a special layer in a neural network. Those network layers can also be equipped with learnable parameters, like the step sizes σ, τ, the over-relaxation parameter θ, or the trade-off parameter λ of the energy functional.

In Section 2.2.3 we already presented several convex energy functionals that are now directly applicable to train on top of a deep neural network for depth super-resolution. To complete this section we will introduce another energy functional with a non-local regularization term [83], i.e. the regularization term is evaluated in a larger neighborhood. The non-local Huber regularization term [255] can be formulated as

$$\min_{\mathbf{u}} \sum_i \sum_{j>i} w(i,j)\, |u_i - u_j|_\epsilon + \frac{\lambda}{2} ||\mathbf{u} - \mathbf{f}||_2^2\,, \tag{3.29}$$

where the sum variables i, j run over the whole discrete image domain. The term $w(i,j)$



(a) Local **(b)** Non-local 3×3 **(c)** Non-local 5×5 **(d)** Non-local 7×7

Figure 3.3: Construction of a discrete non-local neighborhood regularization term. Figure (a) shows the standard local neighborhood, whereas Figures (b) to (d) show non-local neighborhoods in increasing patch-sizes, from 3×3 to 7×7. The edges are only present in the bottom patch plane to prevent double counting.

are non-negative support weights that allow to incorporate additional prior information into the regularization term, i.e. the network output $f_r(\mathbf{s}_k; \boldsymbol{\Theta}_n)$. A more basic weighting term is given by the inverse proximity with exponential decay

$$w(i,j) = \frac{1}{Z} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{\sigma_p}\right), \tag{3.30}$$

where Z is the partition function, $\mathbf{x}_i, \mathbf{x}_j$ are the pixel positions of u_i, u_j, and σ_p controls the proximity of the non-local term. Note that the proximity parameter has also practical implications: If the proximity covers large areas the model quickly becomes intractable. Typical non-local models are depicted in Figure 3.3.

Usually, we want to emphasize the regularization term in homogeneous regions and deactivate it near depth discontinuities. We can facilitate an additional network output $f_r(\mathbf{s}_k; \boldsymbol{\Theta}_n)$ to weight the regularization term accordingly. Then, the weighting term becomes

$$w_f(i,j) = \frac{1}{Z} \exp\left(-\frac{f_r(\mathbf{s}_k; \boldsymbol{\Theta}_n)_{ij}}{\sigma_c}\right) \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{\sigma_p}\right), \tag{3.31}$$

where σ_c is a control parameter that tunes the influence of the network weight term on the regularization. To formulate the complete energy functional in its primal form, we first introduce a generalized nabla operator $\nabla_{\mathcal{N}}$. $\mathcal{N} = \{\boldsymbol{n}_1, \cdots, \boldsymbol{n}_N\}$ defines the set of edges in the non-local neighborhood, i.e. the neighborhood which should be considered in the regularization term. For the example presented in Figure 3.3b the set is $\mathcal{N} = \{(1,0), (-1,1), (0,1), (1,1)\}$. The generalized nabla operator can then be defined as

$$\nabla_{\mathcal{N}} \mathbf{u} = \begin{pmatrix} \nabla_{\boldsymbol{n}_1} \mathbf{u} \\ \dots \\ \nabla_{\boldsymbol{n}_N} \mathbf{u} \end{pmatrix} \quad \text{with} \quad (\nabla_{\boldsymbol{n}} \mathbf{u})_i = \begin{cases} u_i - u_j & \text{if } \mathbf{x}_j = \mathbf{x}_i + \boldsymbol{n} \in \Omega \\ 0 & \text{else} \end{cases}. \tag{3.32}$$

Now, if we define $\mathbf{W}_f$ as block matrix such that

$$\|\mathbf{W}_f \nabla_{\mathcal{N}} \mathbf{u}\|_\epsilon = \sum_i \sum_{j>i} w_f(i,j) \, |u_i - u_j|_\epsilon \, , \tag{3.33}$$

then we can compactly write the primal form of the energy functional as

$$\min_{\mathbf{u}} \|\mathbf{W}_f \nabla_{\mathcal{N}} \mathbf{u}\|_\epsilon + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k; \mathbf{\Theta}_n)||_2^2 \, . \tag{3.34}$$

To employ the fast primal-dual optimization scheme we need to state the corresponding convex-concave saddle-point problem, which is given by

$$\min_{\mathbf{u}} \max_{\mathbf{p}} \langle \mathbf{W}_f \nabla_{\mathcal{N}} \mathbf{u}, \mathbf{p} \rangle + \frac{\lambda}{2} \|\mathbf{u} - f_d(\mathbf{s}_k; \mathbf{\Theta}_n)\|_2^2 - \delta_P(\mathbf{p}) - \frac{\epsilon}{2} \|\mathbf{p}\|_2^2 \, . \tag{3.35}$$

Finally, one iteration of the primal-dual algorithm can then be written as

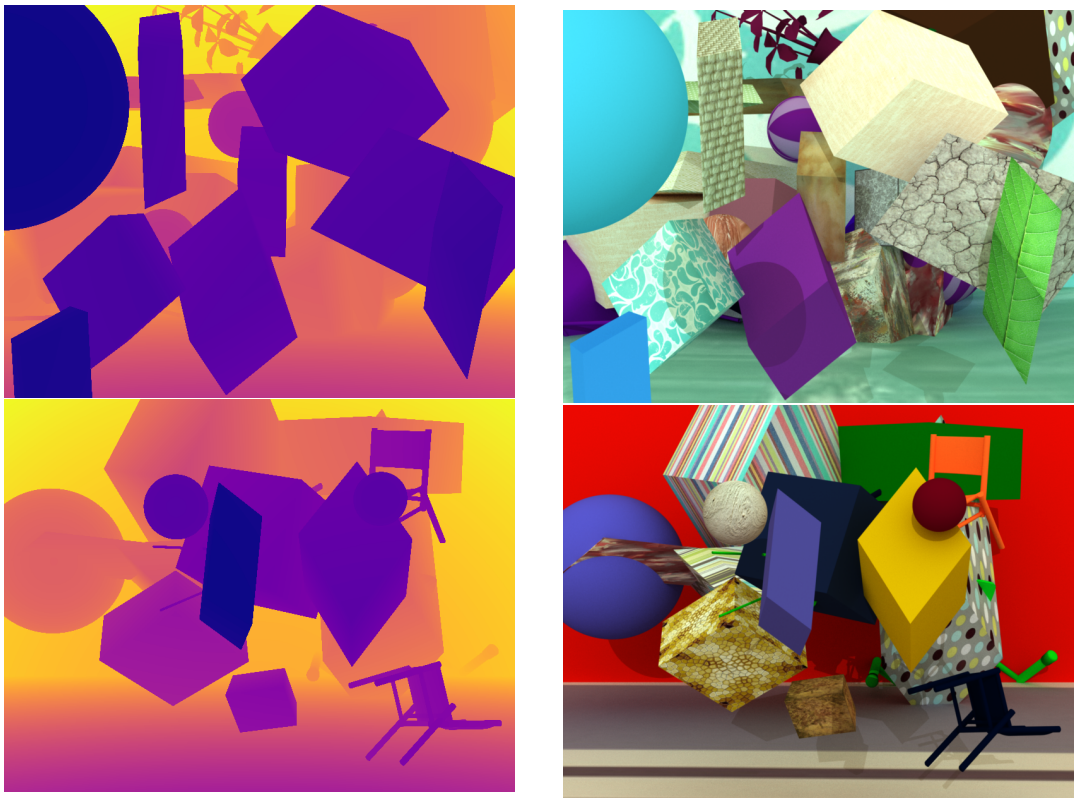
$$\begin{cases} \mathbf{p}^{(\mathrm{i}+1)} = \mathrm{prox}_{g^*}(\mathbf{p}^{(\mathrm{i})} + \sigma(\mathbf{W}_f \nabla_{\mathcal{N}} \bar{\mathbf{u}}^{(\mathrm{i})})) \\ \mathbf{u}^{(\mathrm{i}+1)} = \frac{\mathbf{u}^{(\mathrm{i})} - \tau (\mathbf{W}_f \nabla_{\mathcal{N}})^T \mathbf{p}^{(\mathrm{i}+1)} + \tau\lambda f_d(\mathbf{s}_k; \mathbf{\Theta}_n)}{1+\tau\lambda} \\ \bar{\mathbf{u}}^{(\mathrm{i}+1)} = \mathbf{u}^{(\mathrm{i}+1)} + \theta(\mathbf{u}^{(\mathrm{i}+1)} - \mathbf{u}^{(\mathrm{i})}) \end{cases} . \tag{3.36}$$

3.3.3 Training Data

This section is devoted to the automatic generation of the training data. As stated in the introduction of the method we assume that we are given a training dataset $\mathcal{D} = \{(\mathbf{s}_k, \mathbf{t}_k)\}_{k=1}^K$ with K training samples. Each training sample $\mathbf{s}_k$ consists either of a single low-resolution depth map $\mathbf{s}_k = \mathbf{d}_k^{(\mathrm{lr})}$, or a low-resolution depth map in combination with a high-resolution guidance image, $\mathbf{s}_k = (\mathbf{d}_k^{(\mathrm{lr})}, \mathbf{g})$ depending on the task, i.e. guided vs. unguided depth super-resolution. As deep neural networks are mainly data-driven, we need high-quality training samples in large quantities. We circumvent this problem by generating a synthetic, still physically plausible training dataset.

Therefore, we utilize the open source Mitsuba Renderer [119]. In this physically based renderer, a scene is defined by placing objects, light sources, and sensors freely in an environment defined by a configuration file. Using this file, the renderer generates an intensity- and a depth-map of the scene in adjustable quality and size by using sophisticated ray-tracing algorithms.

In our case, the automatic dataset generation is scripted by randomly placing different objects in varying dimensions in the scene. The objects vary from simple primitives such as cubes, spheres, and planes, too complicated 3D shapes from the ModelNet dataset [259]. Further, the objects are randomly textured using samples from the publicly available Describable Textures dataset [43]. Finally, the light intensity and position are also slightly varied with each generated scene. We depict two such generated samples in Figure 3.4. The output intensity image is used as high-resolution guidance image $\mathbf{g}$, the clean depth



(a) Ground truth depth $\mathbf{t}_k = \mathbf{d}_k^{(\mathrm{hr})}$. **(b)** High-resolution guidance image $\mathbf{g}$.

Figure 3.4: Synthetic generated training data using the physically based Mitsuba Renderer [119].

output defines the high-resolution target depth $\mathbf{t}_k$, and by down-sampling $\mathbf{t}_k$ and adding noise we generate the low-resolution depth input $\mathbf{d}^{(\mathrm{lr})}$.

3.4 Evaluation

In this section, we show an exhaustive quantitative and qualitative evaluation of our proposed method for depth super-resolution. We divided the evaluation in various different aspects. First, we evaluate the influence of the super-resolution convolutional network on the high-resolution estimates in Section 3.4.1. We evaluate a number of state-of-the-art networks for natural images and propose a new architecture for this task. Then, in Section 3.4.2 we show the influence of different loss functions on the network estimates. Section 3.4.3 is denoted to the choice of the variational model on top of the deep network, but also shows the influence of the number of iterations of the optimization algorithm on the super-resolution performance. In addition, we evaluate different ways of implementing the optimization steps of the variational model as network layers in Section 3.4.4. Finally, we demonstrate the performance of our method in comparison to state-of-the-art methods on three different benchmark datasets, for unguided depth super-resolution (Section 3.4.5) and guided depth super-resolution (Section 3.4.6).

3.4.1 Super-Resolution Network

Recently, deep convolutional networks have become the state-of-the-art on single image super-resolution for natural images. In this section, we will evaluate different of the proposed architectures for unguided depth super-resolution and propose an own architecture to achieve state-of-the-art performance. However, first, we will introduce the evaluation metrics used throughout the evaluation section. The training set-up that is equivalent for all network architectures if not stated differently.

Metrics To evaluate the various different models and architectures quantitatively, we need to define evaluation metrics that measure the accuracy. The most common metric for regression tasks in general and for super-resolution methods specifically is the root mean squared error (RMSE) defined as

$$\mathrm{RMSE}(\mathbf{f}, \mathbf{t}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f_i - t_i)^2}\,, \tag{3.37}$$

where $\mathbf{f}$ is the estimated output and $\mathbf{t}$ is the ground-truth target. The RMSE measures the mean deviation from the ground-truth. Another related metric is the mean absolute error (MAE) defined as

$$\mathrm{MAE}(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} |f_i - t_i|\,. \tag{3.38}$$

In contrast to the RMSE the MAE measures the mean of the absolute differences instead of the squared differences. This implies that the RMSE puts more weights on large deviations from the ground-truth than the MAE and has a greater sensitivity to a small number outliers. To get a better sense of the source of the error we use another metric that describes the percentage of pixels that are above a certain threshold x. This outlier metric ($\mathrm{OL}_{>x}$) is defined as

$$\mathrm{OL}_{>x}(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} [|f_i - t_i| > x]\,. \tag{3.39}$$

We will use the RMSE, MAE and the outlier metric $\mathrm{OL}_{>x}$ for various thresholds x for all our quantitative evaluations of this chapter.

Experimental Set-Up To evaluate the different choices of our method we use the Middlebury 2005 datasets *Art*, *Books* and *Moebius* [204] as benchmark images as proposed by Park et al. [172]. Small holes in the disparity maps are inpainted and the disparity maps serve as the high-resolution ground-truth. The low-resolution depth map inputs are generated by bicubic down-sampling ($\times 4$) and adding depth-dependent Gaussian noise.

The mean over the three datasets will be denoted by the column *test* for the various metrics. In addition, we prepared an independent validation set of 25 depth maps that are generated as described in Section 3.3.3. This validation set is used for model selection, i.e. we select the network epoch that yields the lowest error on the validation set. We will report the mean over the validation set for each metric in the column *val*.

The network training is also unified for all the experiments presented in this chapter. Using the technique described in Section 3.3.3 we generated 750 high-resolution depth maps of size 622×810pixels, with additional color guidance images. The input low-resolution depth maps are generated in the same way the input for the benchmarks is generated. For network training, we extract 4 random patches of size 128×128 pixels from each training depth map per epoch. This yields $3,000$ different training samples per training epoch, whereas we use a batch size of 32. Hence, a training epoch involves 93 optimization steps with full batch size. The depth maps are further scaled to have a value range of $[0, 1]$ for the network training only. We use Adam [128] as optimization method with a constant learning rate of 0.001 and default momentum parameters $\beta_1 = 0.9, \beta_2 = 0.999$. The network weights are initialized by an orthogonal scheme [202], and the bias terms are set to 0. In this first experiment, we use a mean squared error loss $\mathcal{L}_E$ for network training

$$\mathcal{L}_E(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} (f_i - t_i)^2 \,. \tag{3.40}$$

Network Architectures The first deep network considered in this evaluation was also the first full-convolutional network presented for single-image super-resolution [55]. It consists of three convolutional layers, whereas the first two convolutional layers are followed by a rectified linear unit. The low-resolution input is up-sampled via bicubic interpolation to the target resolution prior as a pre-processing step. A graphical depiction is given in Figure 3.6a. We will denote this network as *SRCNN* in the evaluation.

The bicubic up-sampling step means that all convolutions have to be performed on the high-resolution of the target. We can circumvent this by up-sampling the result only at the end of the convolutional layers as presented in [215]. Instead of an interpolation scheme for the up-sampling, a *pixelshuffle* operation is utilized as depicted in Figure 3.5. This operation combines $s_h \cdot s_w$ feature maps to generate a single new feature map, where s_h, s_w are the scaling factors in height and width. While the original formulation of the architecture only includes 3 convolutions before the output, we evaluate this architecture with a varying number of convolutions. The architecture is visualized in Figure 3.6b and is denoted as *ESPCNN(n)*, where n is the number of convolution layers. Additionally, we can provide a bicubic up-sampled input depth map. Then, the network only has to learn the residual to this mid-resolution input. This variant of the network will be denoted as *ESPCNN(n, r)* in the evaluations.

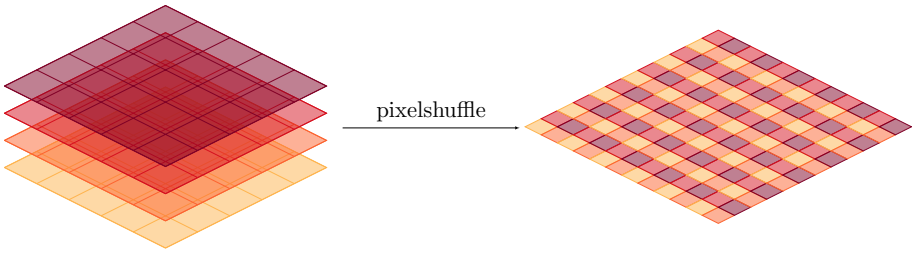


Figure 3.5: The pixel shuffle operation increases the spatial resolution of the feature maps by combining $s_h \cdot s_w$ feature maps into one feature map.

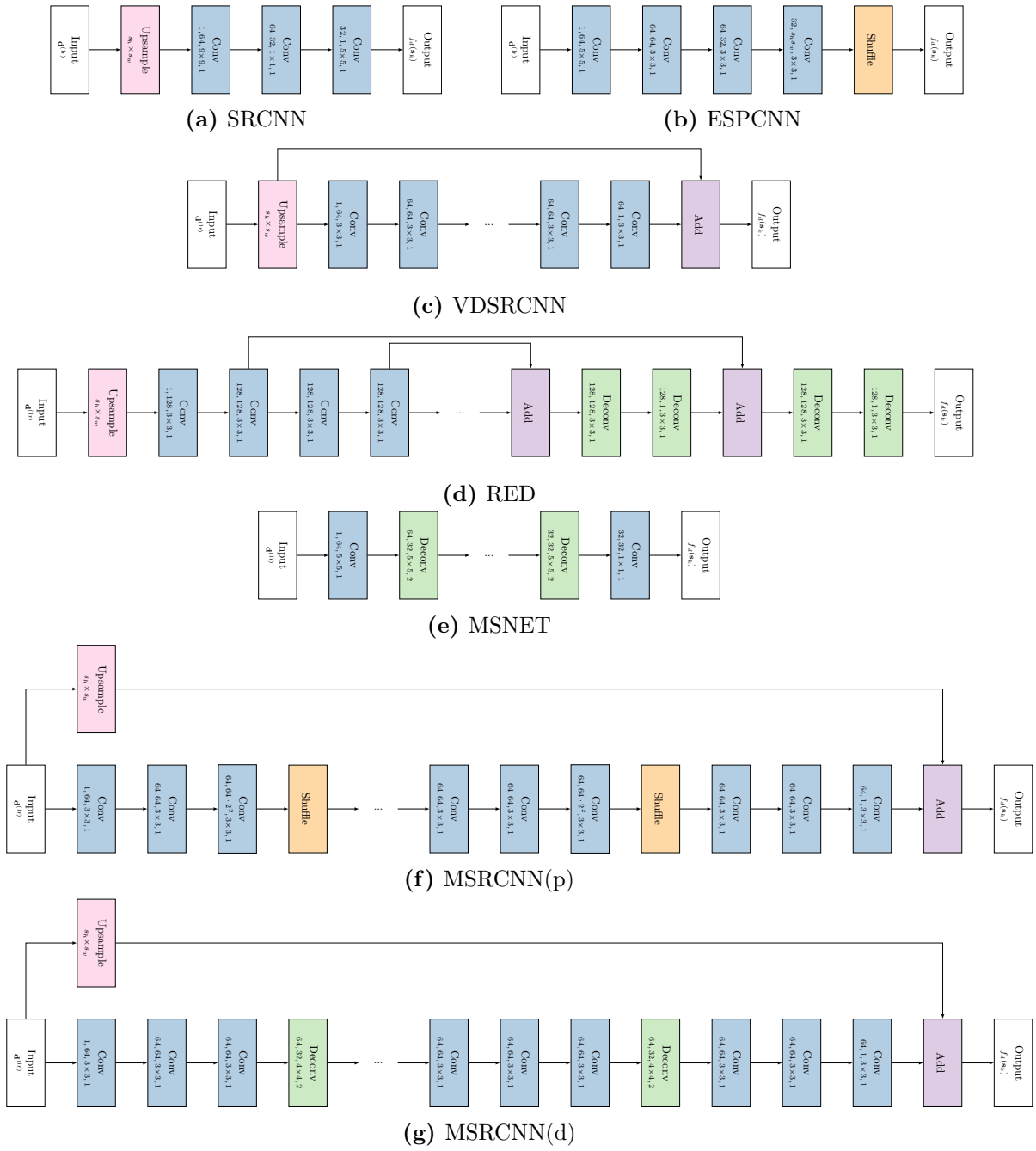
The idea of a residual connection to a mid-resolution input was introduced by [126] to enable the training of very deep networks with up to 20 convolutional layers. Each convolutional layer has a filter size of 3×3pixel and the same number of feature maps, i.e. 64. A visual depiction of the architecture is given in Figure 3.6c. We denote this network as *VDSRCNN(n)* in our experiments, where n is the number of convolutional layers.

One of the currently best-performing single image super-resolution networks is the residual encoder-decoder network by [152]. The idea is to introduce skip connections between encoder and decoder modules to improve the gradient flow similar to ResNets [102]. Encoder modules consist of two convolutional layers, and the decoder modules of two deconvolutional layers, respectively. The architecture is visualized in Figure 3.6d. We denote this network as *RED(n)* in our experiments, where n denotes the number of convolutional and deconvolutional layer pairs.

While the former networks were all proposed for the super-resolution of natural images there exists also a simple architecture specifically for depth maps by [116]. As shown in Figure 3.6e it consists of one convolutional and deconvolutional layer per up-sampling step. We denote this network as *MSNET* in our evaluation.

Finally, we propose a new network variant that combines the advantages of the existing architectures. Similar to *MSNET* we use a step-wise architecture, where we add a number of convolutions plus one up-sampling layer per doubling of the output resolution. Additionally, we add a residual connection to the mid-resolution input. We name this network *MSRCNN(n, [p|d], r)* in our evaluations, where n is the number of convolution layers per up-sampling module. p indicates that we use a pixelshuffle layer for the up-sampling, see Figure 3.6f, and d stands for a deconvolutional layer, see Figure 3.6g. If an r is in the brackets, it indicates that the residual connection to the mid-resolution input is used.

Results We present the results of the network architecture evaluations in Table 3.1. The first observation is that there is a non-trivial difference between the individual network architectures on all metrics. For example, the MAE of *SRCNN* is almost a complete disparity value worse compared to *MSRCNN(5,d,3* on the validation set. There are also more than 20% fewer pixels in the depth maps where the error is larger than 1 disparity value and only halve the pixels have an error larger than 10 disparity values. This is equivalently



(a) SRCNN **(b)** ESPCNN

(c) VDSRCNN

(d) RED

(e) MSNET

(f) MSRCNN(p)

(g) MSRCNN(d)

Figure 3.6: Network architectures used in the evaluation. The different network operations are color-coded and the parameters of the operations are given beneath the operation name. A non-linearity is applied after each hidden convolutional layer.

Table 3.1: Evaluation of the network architectures on the synthetic validation set and the noisy Middlebury test set for an up-sampling factor of ×4. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow.

	RMSE		MAE		$OL_{>1}$		$OL_{>5}$		$OL_{>10}$	
	val	test	val	test	val	test	val	test	val	test
SRCNN	4.228	2.882	1.956	1.784	53.148	57.247	6.770	5.302	1.902	0.966
MSNET	4.030	2.620	1.760	1.539	48.788	50.958	5.019	3.762	1.817	0.919
VDSRCNN(5)	3.949	2.771	1.805	1.684	51.771	55.358	5.336	4.357	1.524	0.910
VDSRCNN(10)	3.537	2.382	1.426	1.338	41.490	45.537	3.170	2.599	1.116	0.672
VDSRCNN(15)	3.366	2.296	1.313	1.247	38.424	42.518	2.689	2.250	0.999	0.611
VDSRCNN(20)	3.403	2.306	1.301	1.242	37.178	41.911	2.760	2.298	1.010	0.605
ESPCNN(4)	4.053	2.523	1.648	1.413	43.311	46.165	4.824	3.309	1.949	0.902
ESPCNN(8)	3.550	2.280	1.351	1.193	36.547	38.691	3.485	2.508	1.332	0.668
ESPCNN(16)	3.541	2.253	1.368	1.166	37.576	37.642	3.514	2.420	1.332	0.646
ESPCNN(4,r)	3.632	2.317	1.409	1.230	37.216	40.115	3.859	2.760	1.496	0.753
ESPCNN(8,r)	3.216	2.093	1.155	1.041	33.084	33.979	2.744	1.890	1.055	0.554
ESPCNN(16,r)	3.358	2.184	1.222	1.090	31.640	35.118	3.284	2.292	1.164	0.609
RED(20)	3.412	2.308	1.286	1.233	36.127	41.244	2.820	2.333	0.999	0.620
RED(30)	3.370	2.241	1.320	1.181	38.894	39.822	2.525	2.032	1.028	0.600
MSRCNN(3,p)	3.352	2.285	1.281	1.210	36.482	39.693	2.925	2.473	1.035	0.633
MSRCNN(5,p)	3.473	2.402	1.378	1.340	40.617	45.139	3.169	2.822	1.146	0.702
MSRCNN(3,p,r)	3.186	2.152	1.140	1.085	33.952	34.775	2.310	1.900	0.896	0.538
MSRCNN(5,p,r)	3.102	2.108	1.102	1.060	33.314	36.906	2.570	2.123	0.966	0.564
MSRCNN(7,p,r)	3.172	2.132	1.126	1.055	30.209	33.836	2.370	1.690	0.890	0.518
MSRCNN(3,d)	3.384	2.334	1.326	1.285	39.006	43.886	2.979	2.501	1.084	0.640
MSRCNN(5,d)	3.415	2.332	1.333	1.259	38.063	41.557	3.303	2.808	1.192	0.744
MSRCNN(3,d,r)	3.126	2.131	1.136	1.077	36.307	37.215	2.303	1.927	0.908	0.549
MSRCNN(5,d,r)	3.102	2.073	1.098	1.015	32.702	33.488	2.337	1.766	0.890	0.519
MSRCNN(7,d,r)	3.184	2.179	1.161	1.129	32.123	37.615	2.360	1.864	0.918	0.539

true for the test set. We can further observe that with increasing network depth the results get usually better. The networks that have the least number of parametrized layers, i.e. *SRCNN*, *MSNET*, *VDSRCNN(5)* and *ESPCNN(4)*, perform worst with respect to the evaluation metrics. But also for individual architectures, we can see that by increasing the number of parametrized layers the accuracy increases up to a saturation point, e.g. *MSRCNN*. Another clear benefit is learning the residual to a mid-resolution depth map. For all architectures and all metrics, the performance gets better by adding this simple connection. In the remainder of this evaluation section we will use *MSRCNN(5,d,r)* as base architecture as it yields the best results on most of the metrics. Further, it has the additional benefit that it is also fast, as only a fraction of the convolutions are performed on the high output resolution. Next, we will have a look on the influence of the training loss on the evaluation metrics.

3.4.2 Training Loss

In this evaluation we want to investigate the influence of the loss function on the evaluation metrics.

Losses In general, the loss for network training should be closely related to the evaluation metric of interest. For the previous experiment we used the mean squared error *MSE* as loss, see Equation (3.40), which is closely correlated to the root mean squared error. Alternatively, we evaluate the use of the mean absolute error *MAE* as loss function defined as

$$\mathcal{L}_1(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} |f_i - t_i| \ . \tag{3.41}$$

This loss function is ideal if the mean of absolute error is also the metric of interest. Further, it exhibits a simple sub-gradient, that is either ± 1 for a difference between estimate and target, or 0 otherwise. Finally, we consider also the Huber loss *Huber* with different ϵ thresholds in our evaluation. The loss is defined as

$$\mathcal{L}_\epsilon(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} \frac{|f_i - t_i|^2}{2\epsilon} & \text{if } |f_i - t_i| < \epsilon \\ |f_i - t_i| - \frac{\epsilon}{2} & \text{else} \end{cases} , \tag{3.42}$$

where a squared loss is used for smaller than ϵ errors and an absolute loss for larger ones. We use $\epsilon \in \{0.004, 0.008, 0.016\}$ in our evaluation.

Results The results of the various loss functions with respect to the evaluation metrics are summarized in Table 3.2. As expected, we observe that network training with a *MSE* loss yields the best RMSE value on the validation and test set. However, the other metrics like MAE and percentage of outliers for different thresholds is much worse than for the other loss functions. To our surprise the *MAE* loss did not obtain the best metrics on the MAE metric on the validation set, as the *Huber* loss with $\epsilon = 0.004$ is slightly better. But this could also just be an artifact of the random initializations of the different networks, as the MAE on the test set is almost identical and both losses yield very similar results on the other metrics. According to this evaluation, both loss functions would be a suitable choice for the network training with respect to the mean absolute error and the percentage of outliers. In the end, we decided to use the *Huber* loss with $\epsilon = 0.004$.

3.4.3 Variational Model

In Section 2.2.3 and Section 3.3.2 we introduced different variational models that can be used on top of a deep network for post-processing. In this evaluation we compare the different models on two aspects: (i) Which energy functional yields the most improvement

Table 3.2: Evaluation of the training losses on the synthetic validation set and the noisy Middlebury test set for an up-sampling factor of $\times 4$. The RMSE and MAE values are in pixel disparities and the $\text{OL}_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow.

	RMSE		MAE		$\text{OL}_{>1}$		$\text{OL}_{>5}$		$\text{OL}_{>10}$	
	val	test	val	test	val	test	val	test	val	test
MSE	3.102	2.073	1.098	1.015	32.702	33.488	2.337	1.766	0.890	0.519
AE	3.313	2.117	0.930	0.860	23.905	23.763	1.324	1.103	0.598	0.447
$\text{Huber}_{\epsilon=0.004}$	3.280	2.093	0.910	0.867	21.712	24.394	1.433	1.108	0.609	0.437
$\text{Huber}_{\epsilon=0.008}$	3.297	2.135	0.922	0.883	22.736	25.446	1.378	1.059	0.604	0.453
$\text{Huber}_{\epsilon=0.016}$	3.266	2.114	0.960	0.911	25.349	27.587	1.329	1.087	0.582	0.441

over the network estimate and (ii) what are the detriments with respect to the accuracy by using a fixed number of optimization iterations.

Energy Functionals The first variational model that we consider in the evaluation is the *TV-L2* with the primal optimization problem defined as

$$\min_{\mathbf{u}} \|\mathbf{W}_f \nabla \mathbf{u}\|_1 + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k)||_2^2 . \tag{3.43}$$

Additionally, we also evaluate a non-local variation of the model denoted as $NLTV_n$-*L2*, where n denotes the extend of the non-local neighborhood $\mathcal{N}$. The energy functional is given by

$$\min_{\mathbf{u}} \|\mathbf{W}_f \nabla_{\mathcal{N}} \mathbf{u}\|_1 + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k)||_2^2 . \tag{3.44}$$

Instead the ℓ_1 norm in the regularization term, we can also utilize the Huber norm. We denote this model as *Huber-L2* in our evaluation and it is defined as

$$\min_{\mathbf{u}} \|\mathbf{W}_f \nabla \mathbf{u}\|_\epsilon + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k)||_2^2 . \tag{3.45}$$

As presented in Section 3.3.2 we can formulate also non-local variant of the energy functional that is defined as

$$\min_{\mathbf{u}} \|\mathbf{W}_f \nabla_{\mathcal{N}} \mathbf{u}\|_\epsilon + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k)||_2^2 . \tag{3.46}$$

We denote this variational model as $NLHuber_n$-*L2* in our evaluation, where n denotes the extend of the non-local neighborhood $\mathcal{N}$. Finally, we include the *TGV-L2* model, where the Total Generalized Variation is used as regularization term. The primal optimization problem is given as

$$\min_{\mathbf{u}} \alpha_1 \|\mathbf{W}_f \nabla \mathbf{u} - \mathbf{v}\|_1 + \alpha_0 \|\boldsymbol{\nabla} \mathbf{v}\|_1 + \frac{\lambda}{2} ||\mathbf{u} - f_d(\mathbf{s}_k)||_2^2 . \tag{3.47}$$

Table 3.3: Evaluation of the variational models on the synthetic validation set and the noisy Middlebury test set for an up-sampling factor of $\times 4$. The RMSE and MAE values are in pixel disparities and the $\mathrm{OL}_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow.

(a) Validation Set

	RMSE				MAE				$\mathrm{OL}_{>1}$				$\mathrm{OL}_{>5}$				$\mathrm{OL}_{>10}$			
	10	20	30	100	10	20	30	100	10	20	30	100	10	20	30	100	10	20	30	100
TV-L2	3.277	3.263	3.227	3.216	0.910	0.886	0.869	0.851	21.717	20.532	19.776	18.832	1.439	1.409	1.427	1.450	0.609	0.607	0.614	0.619
NLTV_3-L2	3.238	3.221	3.208	3.213	0.880	0.873	0.866	0.854	20.249	19.836	19.435	18.826	1.429	1.464	1.502	1.473	0.611	0.617	0.625	0.619
NLTV_5-L2	3.280	3.280	3.222	3.211	0.910	0.910	0.862	0.857	21.712	21.712	19.182	18.774	1.433	1.433	1.496	1.550	0.609	0.609	0.618	0.624
NLTV_7-L2	3.236	3.280	3.222	3.195	0.872	0.910	0.863	0.856	19.658	21.712	19.033	18.655	1.483	1.433	1.541	1.585	0.614	0.609	0.620	0.636
NLTV_9-L2	3.233	3.235	3.225	3.198	0.881	0.891	0.866	0.862	20.080	20.708	19.130	18.866	1.497	1.459	1.558	1.608	0.616	0.611	0.621	0.634
Huber-L2	3.277	3.230	3.251	3.244	0.910	0.875	0.876	0.865	21.717	20.031	20.063	19.491	1.439	1.425	1.401	1.400	0.609	0.612	0.607	0.608
$\mathrm{NLHuber}_3$-L2	3.259	3.280	3.210	3.207	0.887	0.910	0.864	0.852	20.547	21.712	19.378	18.723	1.417	1.433	1.493	1.499	0.608	0.609	0.622	0.623
$\mathrm{NLHuber}_5$-L2	3.236	3.242	3.235	3.211	0.875	0.872	0.867	0.857	19.882	19.718	19.433	18.776	1.459	1.446	1.456	1.549	0.613	0.611	0.612	0.624
$\mathrm{NLHuber}_7$-L2	3.280	3.229	3.222	3.195	0.910	0.864	0.862	0.856	21.712	19.228	18.991	18.659	1.433	1.509	1.540	1.585	0.609	0.617	0.620	0.636
$\mathrm{NLHuber}_9$-L2	3.280	3.232	3.280	3.229	0.910	0.870	0.910	0.865	21.712	19.468	21.712	19.085	1.433	1.510	1.433	1.543	0.609	0.616	0.609	0.619
TGV-L2	3.262	3.260	3.246	3.246	0.909	0.906	0.898	0.884	21.638	21.443	21.025	20.451	1.460	1.462	1.477	1.415	0.611	0.610	0.613	0.607

(b) Test Set

	RMSE				MAE				$\mathrm{OL}_{>1}$				$\mathrm{OL}_{>5}$				$\mathrm{OL}_{>10}$			
	10	20	30	100	10	20	30	100	10	20	30	100	10	20	30	100	10	20	30	100
TV-L2	2.091	2.076	2.053	2.036	0.867	0.848	0.841	0.816	24.395	23.180	22.642	21.007	1.109	1.100	1.118	1.125	0.436	0.435	0.432	0.433
NLTV_3-L2	2.058	2.044	2.031	2.028	0.847	0.842	0.832	0.815	23.117	22.665	22.004	20.862	1.117	1.141	1.156	1.144	0.432	0.431	0.431	0.431
NLTV_5-L2	2.093	2.093	2.035	2.022	0.867	0.867	0.824	0.815	24.394	24.394	21.457	20.665	1.108	1.108	1.146	1.174	0.437	0.437	0.428	0.427
NLTV_7-L2	2.049	2.093	2.031	2.012	0.834	0.867	0.820	0.812	22.119	24.394	21.028	20.527	1.135	1.108	1.164	1.188	0.429	0.437	0.427	0.431
NLTV_9-L2	2.048	2.058	2.032	2.013	0.843	0.855	0.821	0.817	22.721	23.588	21.054	20.809	1.139	1.130	1.170	1.199	0.429	0.430	0.426	0.429
Huber-L2	2.091	2.057	2.067	2.058	0.867	0.846	0.841	0.828	24.395	22.997	22.752	21.897	1.109	1.119	1.099	1.100	0.437	0.432	0.434	0.433
$\mathrm{NLHuber}_3$-L2	2.073	2.093	2.032	2.023	0.849	0.867	0.830	0.813	23.217	24.394	21.881	20.699	1.104	1.108	1.153	1.158	0.434	0.437	0.431	0.430
$\mathrm{NLHuber}_5$-L2	2.051	2.054	2.047	2.022	0.839	0.834	0.828	0.815	22.532	22.185	21.766	20.670	1.127	1.118	1.124	1.174	0.430	0.432	0.430	0.427
$\mathrm{NLHuber}_7$-L2	2.093	2.041	2.032	2.012	0.867	0.825	0.819	0.812	24.394	21.418	20.945	20.528	1.108	1.149	1.164	1.188	0.437	0.428	0.427	0.431
$\mathrm{NLHuber}_9$-L2	2.093	2.042	2.093	2.036	0.867	0.829	0.867	0.820	24.394	21.709	24.394	20.959	1.108	1.146	1.108	1.164	0.437	0.428	0.437	0.427
TGV-L2	2.077	2.078	2.069	2.067	0.866	0.863	0.858	0.852	24.327	24.165	23.791	23.336	1.122	1.117	1.119	1.114	0.433	0.431	0.432	0.430

Results The results of our evaluation are presented in Table 3.3. We evaluate the various variational methods as post-processing step with a different number of iterations of the optimization algorithm. Note that we used a stochastic hill-climbing approach to set the hyper-parameters of the variational models on a subset of the validation set ($K = 10$). We can observe that for the best models the percentage of pixels with an error in depth of >1 disparity values decrease by $\sim$3% for the validation set and $\sim$4% for the test set in comparison to the network output, respectively. Further, the non-local models yield usually better performance than their local counterparts, i.e. *Huber-L2* with 30 iterations has $\text{OL}_{>1} = 22.752\%$ on the test set, whereas *NLHuber*$_7$*-L2* has only $\text{OL}_{>1} = 20.945\%$ with the same number of iterations. Regarding iterations, we can see that the performance is almost saturated after 30 optimization iterations. For the remainder of the evaluation section, we use the *NLHuber*$_7$*-L2* model and unrolling 30 optimization steps.

3.4.4 Variational Network Training

In the previous evaluation we used the variational method only as a post-processing step. However, by unrolling the iterations of the optimization scheme on top of the network we are able to train the network and the variational model in an end-to-end fashion.

Experimental Set-Up Combining the pre-trained network *MSRCNN(5,d,r)* and the variational model *NLHuber*$_7$*-L2* we jointly train the whole model end-to-end. We evaluate three different strategies. (i) We only train the network parameters and keep the parameters of the variational model and its optimization algorithm fixed, i.e. the trade-off parameter λ and the step sizes σ, τ. This strategy is denoted as *Net only*. (ii) We only optimize the hyper-parameters of the variational model, as well as the parameters of the optimization algorithm, but keep the pre-trained network parameters fixed. This strategy is denoted as *Var only*. (iii) We train both, the network parameters and all parameters of the variational model. This strategy is denoted as *Both*.

Results The results of this evaluation are summarized in Table 3.5. Interestingly, training only the parameters of the variational model did not improve the performance on the validation and test set, but had to some degree the opposite effect. In contrast, training only the network parameters with the variational model on top did already improve the results. However, the best performance is obtained by training all parameters simultaneously. These results suggest, that the adaption of the network to the subsequent variational model is more important than tuning the parameters of it.

3.4.5 Depth Super-Resolution

The previous evaluations served the purpose to justify the model choices of our method. This section is now devoted to the comparison of the presented unguided depth super-resolution method to other state-of-the-art approaches on three different benchmarks.

Table 3.5: Evaluation of the variational network training on the synthetic validation set and the noisy Middlebury test set for an up-sampling factor of ×4. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow.

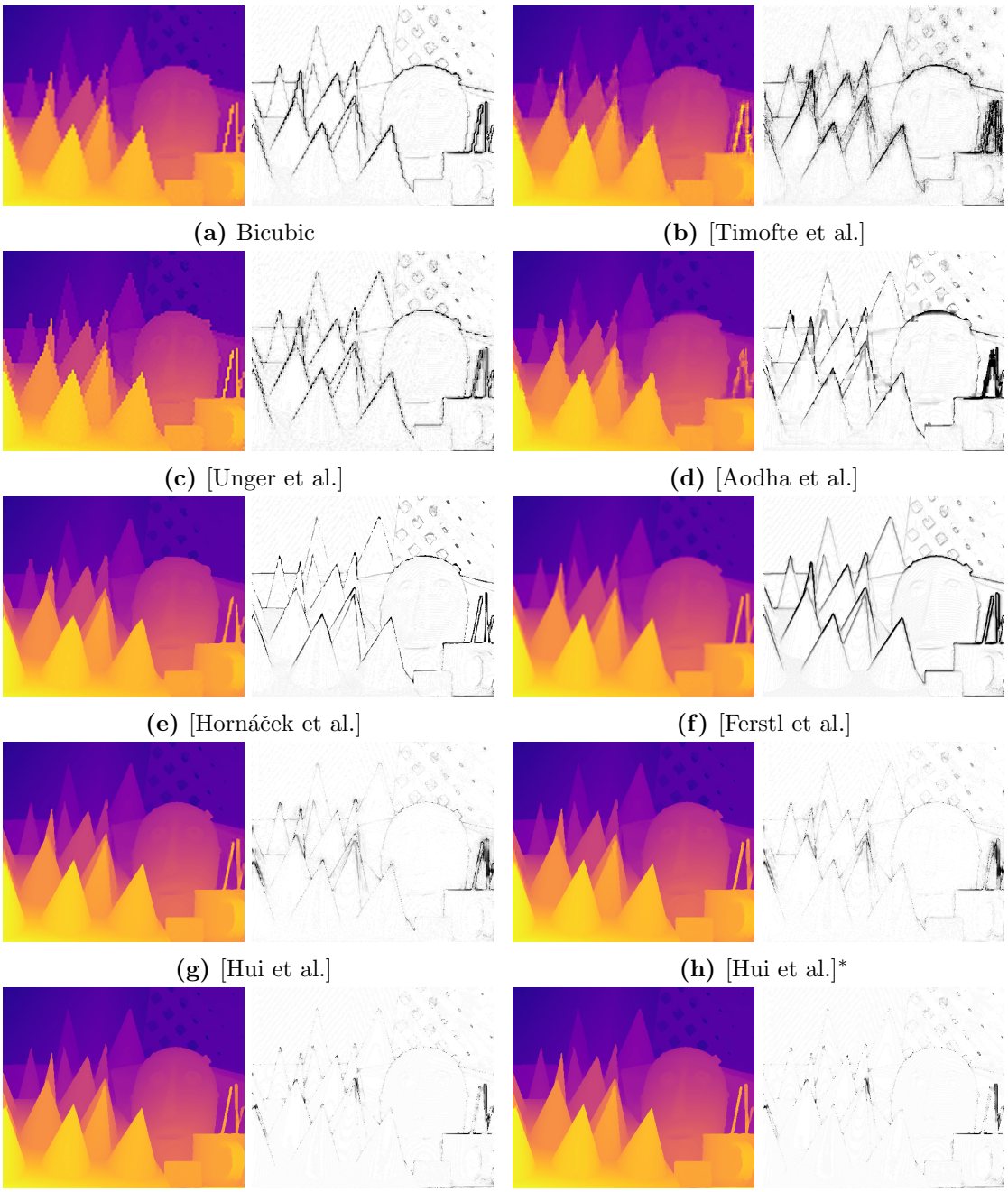
	RMSE		MAE		$OL_{>1}$		$OL_{>5}$		$OL_{>10}$	
	val	test	val	test	val	test	val	test	val	test
Net only	3.295	2.038	0.833	0.793	17.873	20.168	1.303	0.995	0.609	0.432
Var only	3.237	2.046	0.864	0.824	19.241	21.483	1.474	1.129	0.613	0.429
Both	3.234	2.050	0.819	0.768	18.194	18.574	1.195	0.975	0.585	0.437

3.4.5.1 Noise-Free Middlebury

The first benchmark in our state-of-the-art evaluation is the noise-free Middlebury dataset as used in [5, 70, 106]. The disparity maps of *Teddy*, *Cones*, *Tsukuba* and *Venus* are interpreted as depth ground-truth. To generate the input, the high-resolution disparity maps are down-sampled using nearest neighbor interpolation (×2, ×4).

Methods In this state-of-the-art evaluation we first include two simple interpolation techniques as baseline, i.e. *Nearest Neighbor* and *Bicubic*. We also assess two sparse coding approaches, the one by *Timofte et al.* [240] and the one by *Zeyde et al.* [270], whereas the dictionaries for this task were learned as described in [70]. A pure variational approach presented in this evaluation is by *Unger et al.* [250], as well as the Markov Random Field based methods by *Aodha et al.* [5] and *Hornáček et al.* [106]. The method by *Ferstl et al.* [70] combines sparse coding based super-resolution with a variational method and is therefore closely related to our method. Finally, we include the two convolutional network based approaches by *Hui et al.* [116], where one network is using an additional high-resolution intensity image as guidance. It should be noted, that *Hui et al.* [116] used bicubic down-sampling of the ground-truth to generate the input. Hence, making the problem a bit easier. The result of our network only estimates is denoted as *Net*, and as *Net+PD* if a variational method was used for post-processing. Our final results, where we trained the network and the variational method end-to-end is denoted as *PD-Net*.

Results The quantitative results of our state-of-the-art evaluation on the noise-free Middlebury dataset are shown in Table 3.6. Qualitative results for a subset of the depth maps and an up-sampling factor of ×4 are visualized in Figures 3.7-3.9. Our proposed convolutional network for unguided depth super-resolution outperforms all other methods that do not utilize an additional high-resolution guidance images for all up-sampling factors. Only the guided network by *Hui et al.* [116] yields a slightly better MAE for a factor of ×2. Interestingly, applying a variational model on top of the network does not (×2), or only marginally (×4) improve the results. However, the model that jointly trains both parts yields on all metrics and up-sampling factors the best performance. It even yields better results than the guided network by *Hui et al.* [116] on all up-sampling factors.



(a) Bicubic **(b)** [Timofte et al.]

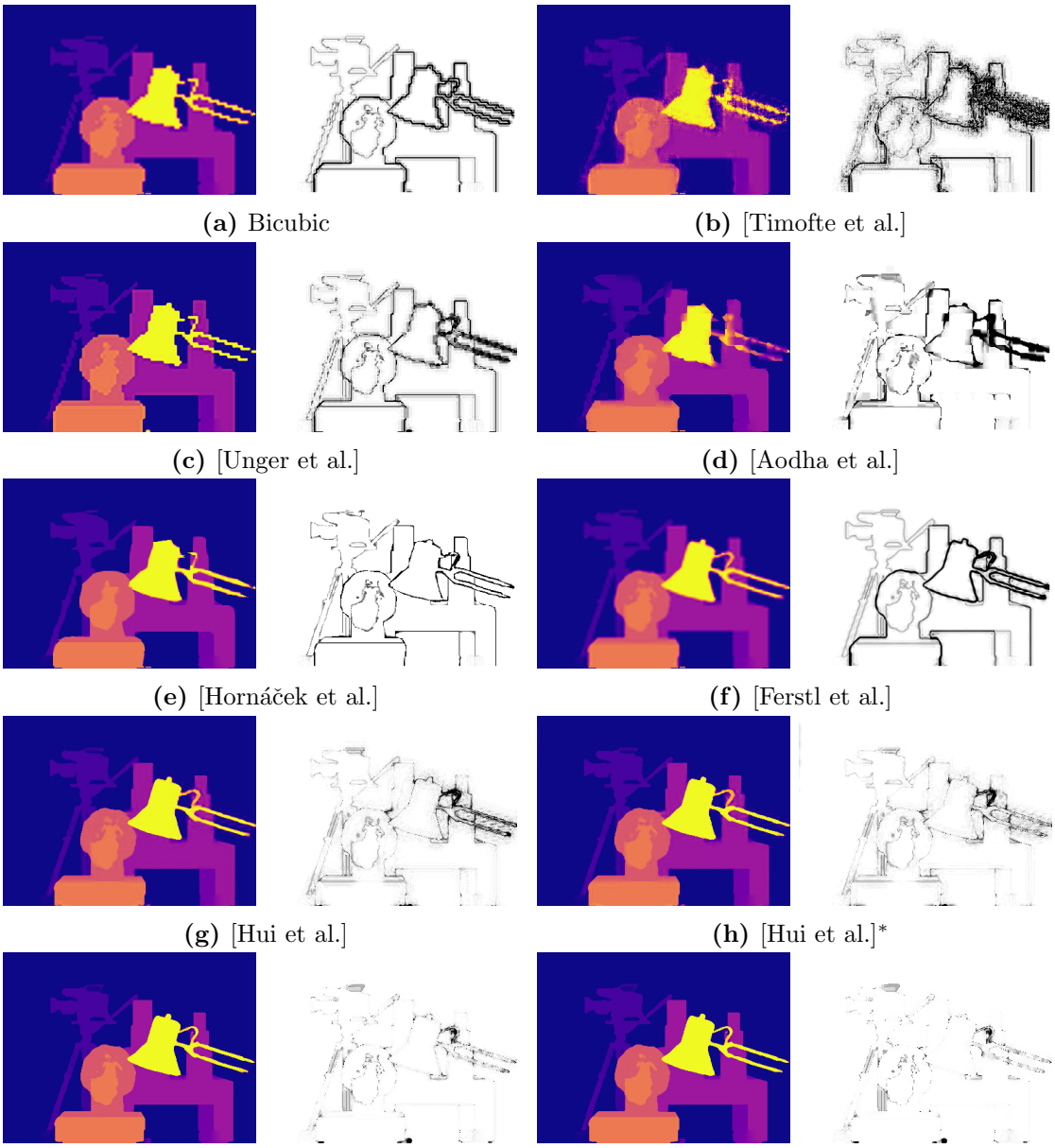
(c) [Unger et al.] **(d)** [Aodha et al.]

(e) [Hornáček et al.] **(f)** [Ferstl et al.]

(g) [Hui et al.] **(h)** [Hui et al.]*

(i) Net **(j)** PD-Net

Figure 3.7: Qualitative results for the Middlebury dataset disparity map *Cones* and an upsampling factor of $\times 4$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bicubic **(b)** [Timofte et al.]

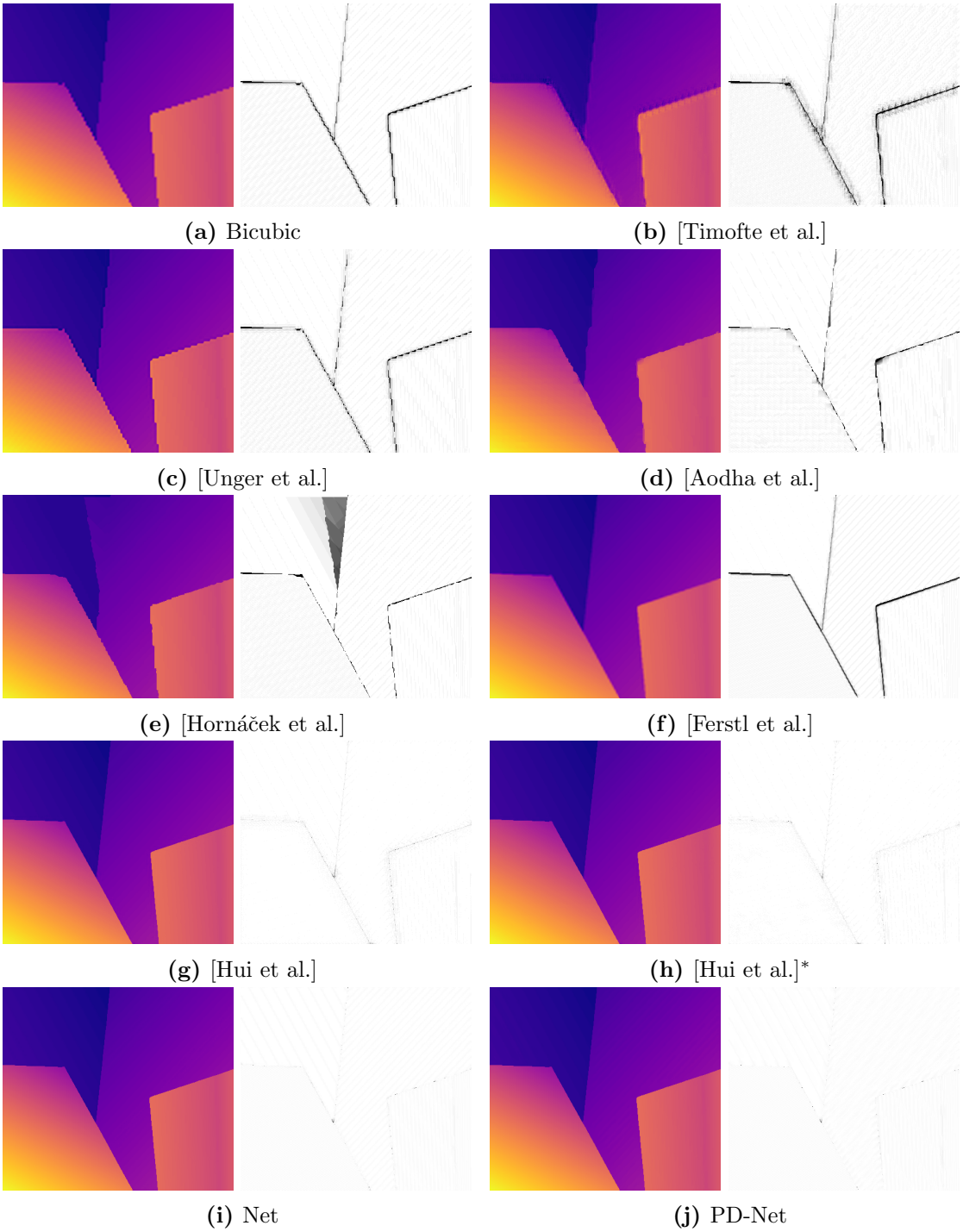
(c) [Unger et al.] **(d)** [Aodha et al.]

(e) [Hornáček et al.] **(f)** [Ferstl et al.]

(g) [Hui et al.] **(h)** [Hui et al.]*

(i) Net **(j)** PD-Net

Figure 3.8: Qualitative results for the Middlebury dataset disparity map *Tsukuba* and an upsampling factor of ×4. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bicubic **(b)** [Timofte et al.]

(c) [Unger et al.] **(d)** [Aodha et al.]

(e) [Hornáček et al.] **(f)** [Ferstl et al.]

(g) [Hui et al.] **(h)** [Hui et al.]*

(i) Net **(j)** PD-Net

Figure 3.9: Qualitative results for the Middlebury dataset disparity map *Venus* and an upsampling factor of $\times 4$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.

Table 3.6: Evaluation of unguided depth super-resolution on the noise-free Middlebury dataset for different up-sampling factors. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow. Methods that use an additional high-resolution intensity image as guidance are marked with an asterisk *.

	×2					×4				
	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Nearest Neighbour	4.631	0.579	2.448	1.416	1.136	6.648	1.168	5.028	3.307	2.444
Bicubic	3.975	0.825	7.946	3.156	1.734	5.452	1.463	13.323	5.918	3.193
[Timofte et al.]	4.702	0.991	9.017	3.592	2.021	6.724	2.271	22.370	9.491	4.937
[Zeyde et al.]	4.530	0.893	7.848	3.102	1.798	5.939	1.737	15.905	6.529	3.668
[Unger et al.]	4.904	0.721	5.035	1.679	1.202	6.875	1.627	15.003	4.916	2.924
[Aodha et al.]	4.731	0.804	5.619	2.434	1.643	6.645	1.560	10.791	5.257	3.344
[Hornáček et al.]	4.357	0.629	3.195	1.767	1.279	6.385	1.300	6.561	4.500	3.491
[Ferstl et al.]	3.224	0.692	7.081	2.851	1.509	4.863	1.402	12.943	6.485	3.286
[Hui et al.]	1.163	0.269	3.704	0.735	0.235	2.430	0.582	7.125	2.046	0.916
[Hui et al.]*	0.902	0.167	2.233	0.409	0.111	2.180	0.482	6.085	1.618	0.669
Net	0.860	0.180	1.649	0.274	0.089	2.119	0.421	4.838	1.062	0.553
Net+PD	0.860	0.180	1.649	0.274	0.089	2.103	0.417	4.521	1.046	0.551
PD-Net	0.825	0.157	1.426	0.257	0.085	2.081	0.405	3.950	0.934	0.517

3.4.5.2 Noisy Middlebury

In the second state-of-the-art evaluation we evaluate on the noisy Middlebury dataset as proposed by [172]. This evaluation consists of the disparity maps *Art*, *Books*, and *Moebius* as ground-truth data, where the occlusions have been manually inpainted. The input data is generated by down-sampling the high-resolution disparity map using bicubic interpolation (×2, ×4, ×8, and ×16) and then adding noise. The noise apparent in Time-of-Flight depth maps is simulated with a simple conditional Gaussian noise

$$d_n = d_c(1 + \mathcal{N}(\mu, \sigma^2)) \,, \tag{3.48}$$

where d_c is the noise-free disparity map and d_n the noisy input, respectively. $\mathcal{N}(\mu, \sigma)$ is the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 0.01^2$.

Methods As in the previous evaluation we include two simple interpolation techniques, i.e. *Nearest Neighbor* and *Bilinear*, as baselines. The other state-of-the-art methods in this evaluation all require an additional high-resolution image as guidance input. This might be due to the input noise and the rather high up-sampling factors. The first set of methods we compare to are based on a Markov Random Field formulation, like *Diebel and Thrun* [52], *Park et al.* [172], and *Lu and Forsyth* [150]. The methods mainly differ in the formulation of the energy functional and the inference method. The second larger set of methods in this evaluation is related to bilateral image filtering, or guided image filtering. That includes *Yang et al.* [266], *Chan et al.* [29], *He et al.* [101], *Shen et al.* [214],

and *Yang* [265]. In addition, we include the variational model by *Ferstl et al.* [67], the autoregressive method by *Yang et al.* [264], and the recently proposed bilateral solver by [9]. The results of convolutional network are denoted as *Net*, and as *Net+PD* if a variational method was used for post-processing. Our final results, where we trained the network and the variational method end-to-end is denoted as *PD-Net*.

Results The quantitative results of our state-of-the-art evaluation on the noisy Middlebury dataset are shown in Table 3.7. Qualitative results for up-sampling factors of $\times 4$ and $\times 16$ are visualized in Figures 3.10-3.15. We can again observe that our convolutional network already performs comparable to state-of-the-art methods, despite not using a high-resolution intensity image as additional guidance. In terms of MAE and percentage of outliers the variational post-processing non-trivially increases the accuracy of the network, whereas the best MAE over all up-sampling factors is obtained by end-to-end training. This also yields new state-of-the-art results on this benchmark. Note, how the jointly trained *PD-Net* decreases the $\text{OL}_{>1}$ metric by a minimum of 4% over *Net* only.

3.4.5.3 ToFMark

In a third evaluation we test our method on the challenging Time-of-Flight dataset ToFMark [67]. The dataset consists of three different scenes and for each scene, it provides a noisy, low-resolution Time-of-Flight depth map and a high-resolution intensity image that serve as input, as well as a high-resolution depth map as ground-truth which was acquired by an industrial structured light scanner. The ground-truth depth and the intensity guidance image are in the same coordinate system, but the low-resolution depth map input is in a different one. Therefore, the low-resolution depth-pixels are first mapped to the high-resolution coordinate system via the provided projection matrices.

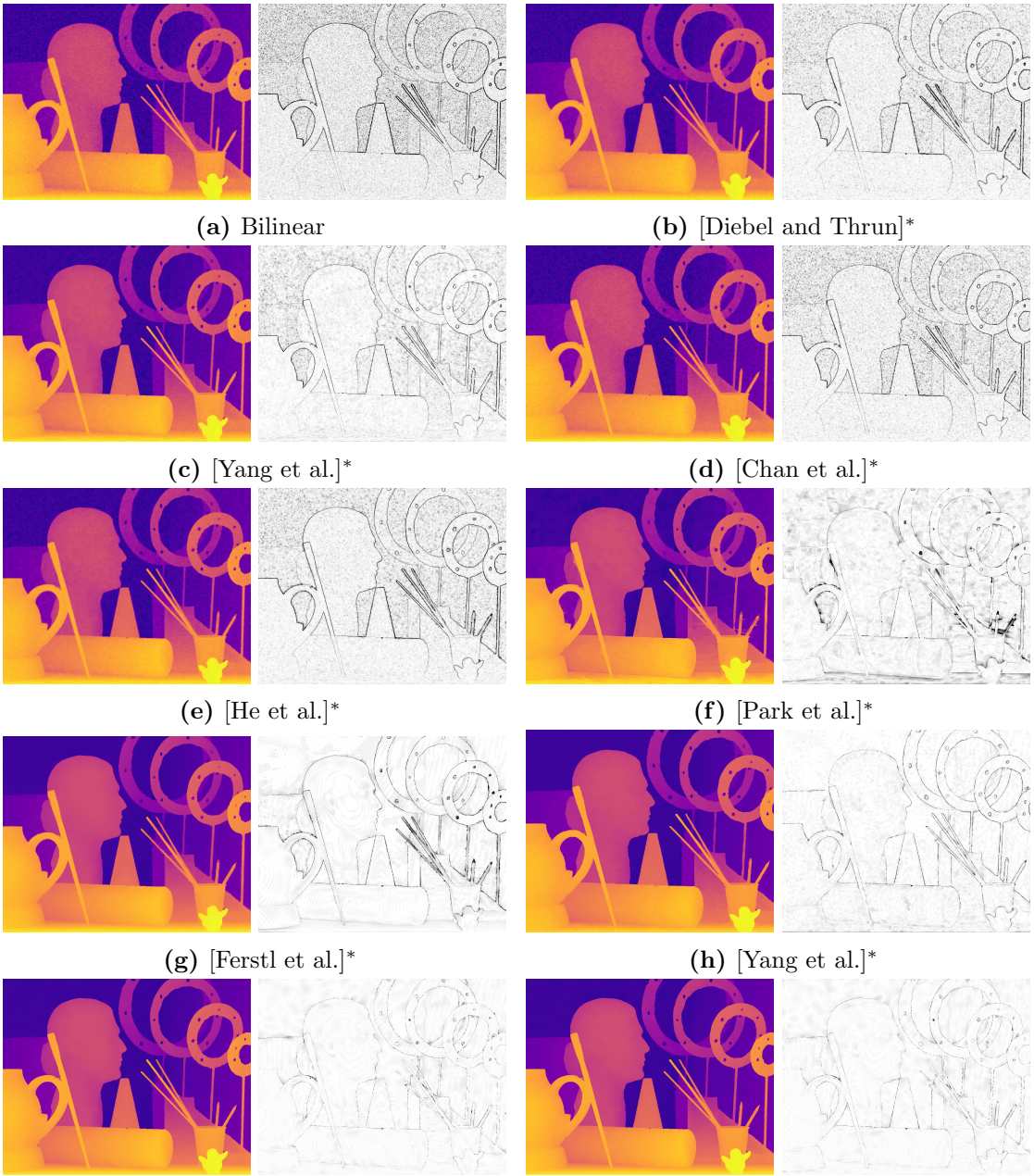
Methods Despite the simple *Bicubic* interpolation, we include in this evaluation the moving least squares filter by *Bose and Ahuja* [16] and the generalized guided filter by *Lu et al.* [151]. Further, we compare to the state-of-the-art variational model by *Ferstl et al.* [67] and the autoregressive model by *Yang* [265]. The results of our convolutional network are denoted as *Net*, and as *Net+PD* if a variational method was used for post-processing. Our final results, where we trained the network and the variational method end-to-end is denoted as *PD-Net*.

Results The quantitative results of our state-of-the-art evaluation on the ToFMark dataset are shown in Table 3.8. Qualitative results are visualized in Figures 3.16-3.18. Even on this very challenging Time-of-Flight dataset, our convolutional network obtains already very competitive results and similar to the previous evaluation, the joint training of the network and the network achieves new state-of-the-art results, especially with respect to the MAE. We can further observe, both quantitatively and qualitatively, that the number of large outliers $\text{OL}_{>10}$ is drastically reduced by our method.

Table 3.7: Evaluation of unguided depth super-resolution on the noisy Middlebury dataset for different up-sampling factors. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow. Methods that use an additional high-resolution intensity image as guidance are marked with an asterisk *.

	×2					×4				
	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Nearest Neighbour	6.434	4.776	77.842	32.823	9.436	6.858	4.966	78.345	33.867	10.262
Bilinear	4.243	3.070	65.927	15.201	2.258	4.832	3.384	67.561	17.971	3.697
[Diebel and Thrun]*	2.560	1.475	34.643	2.094	0.718	3.540	2.304	53.549	7.281	1.400
[Yang et al.]*	2.265	1.243	26.157	1.940	0.628	2.941	1.679	38.363	3.461	1.059
[Chan et al.]*	2.535	1.409	29.201	3.291	0.974	3.332	1.845	36.608	6.305	1.928
[He et al.]*	2.800	1.765	42.582	3.642	0.811	3.327	2.080	48.216	5.686	1.384
[Park et al.]*	2.553	1.087	19.223	1.499	0.691	3.226	1.578	33.566	3.136	1.036
[Lu and Forsyth]*	2.879	1.397	28.692	2.451	1.069	3.473	1.646	32.943	3.774	1.553
[Zhang et al.]*	2.737	1.059	32.672	1.431	0.722	2.973	1.485	50.401	2.343	0.921
[Shen et al.]*	2.973	1.566	46.851	4.992	2.161	3.808	2.268	63.547	8.449	2.904
[Yang]*	2.546	1.501	50.148	3.410	0.743	3.107	1.757	56.033	4.594	1.129
[Ferstl et al.]*	2.062	0.637	7.166	1.264	0.640	2.768	0.981	14.012	2.326	1.041
[Yang et al.]*	1.825	0.564	5.771	0.650	0.315	2.156	0.811	11.561	1.477	0.637
[Barron and Poole]*	2.009	0.991	29.739	1.743	0.758	2.591	1.337	40.841	3.245	1.253
Net	1.350	0.604	13.521	0.480	0.208	2.124	0.884	25.325	1.141	0.447
Net+PD	1.290	0.548	10.477	0.493	0.195	2.037	0.838	22.413	1.175	0.435
PD-Net	1.292	0.521	9.096	0.446	0.207	2.046	0.762	18.305	0.981	0.437

	×8					×16				
	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Nearest Neighbour	7.546	5.287	78.731	35.328	11.794	8.767	5.991	80.172	38.718	14.809
Bilinear	5.576	3.787	69.073	20.870	5.676	6.842	4.540	71.792	25.942	8.971
[Diebel and Thrun]*	4.873	3.267	64.514	15.996	4.045	6.560	4.350	70.718	24.490	8.270
[Yang et al.]*	3.617	2.105	47.499	5.720	1.494	5.510	3.549	65.685	17.863	4.633
[Chan et al.]*	4.590	2.679	48.021	12.557	4.348	6.567	4.006	60.265	22.378	9.051
[He et al.]*	4.239	2.744	58.210	11.026	2.726	5.869	3.825	66.807	20.067	6.305
[Park et al.]*	4.152	2.270	48.252	6.939	1.922	6.217	3.507	62.203	15.577	5.044
[Lu and Forsyth]*	4.327	2.134	41.249	6.353	2.418	5.595	3.077	54.996	12.916	4.538
[Zhang et al.]*	3.888	2.414	67.968	9.425	1.978	5.976	4.043	79.970	26.333	7.073
[Shen et al.]*	5.021	3.291	75.230	18.464	4.951	6.678	4.532	81.649	30.298	9.745
[Yang]*	3.948	2.246	63.147	8.147	2.119	5.455	3.211	72.121	16.069	4.508
[Ferstl et al.]*	3.376	1.530	27.237	4.278	1.830	4.904	2.534	44.916	9.381	3.604
[Yang et al.]*	3.114	1.335	23.058	3.444	1.414	4.854	2.261	40.012	7.242	3.227
[Barron and Poole]*	3.375	1.855	52.477	6.230	2.203	4.736	2.809	66.191	13.057	4.338
Net	3.107	1.349	39.759	2.879	1.007	4.812	2.372	58.436	9.263	3.037
Net+PD	3.017	1.306	37.760	2.904	1.032	4.715	2.312	56.562	9.136	3.055
PD-Net	3.124	1.231	34.590	2.397	1.008	4.819	2.215	53.831	8.037	2.991



(a) Bilinear **(b)** [Diebel and Thrun]*

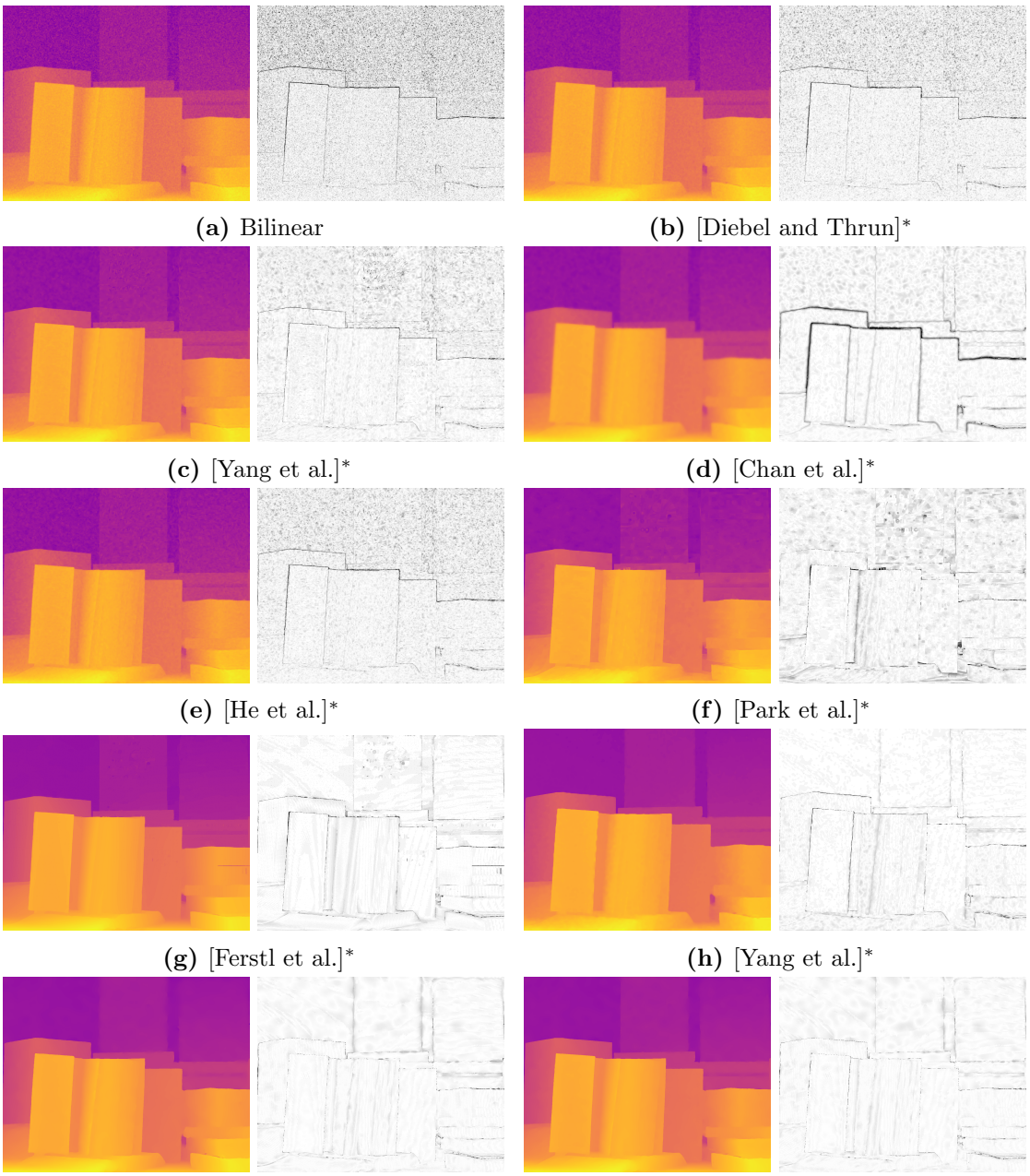
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.10: Qualitative results for the Noisy Middlebury dataset disparity map *Art* and an up-sampling factor of $\times 4$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bilinear **(b)** [Diebel and Thrun]*

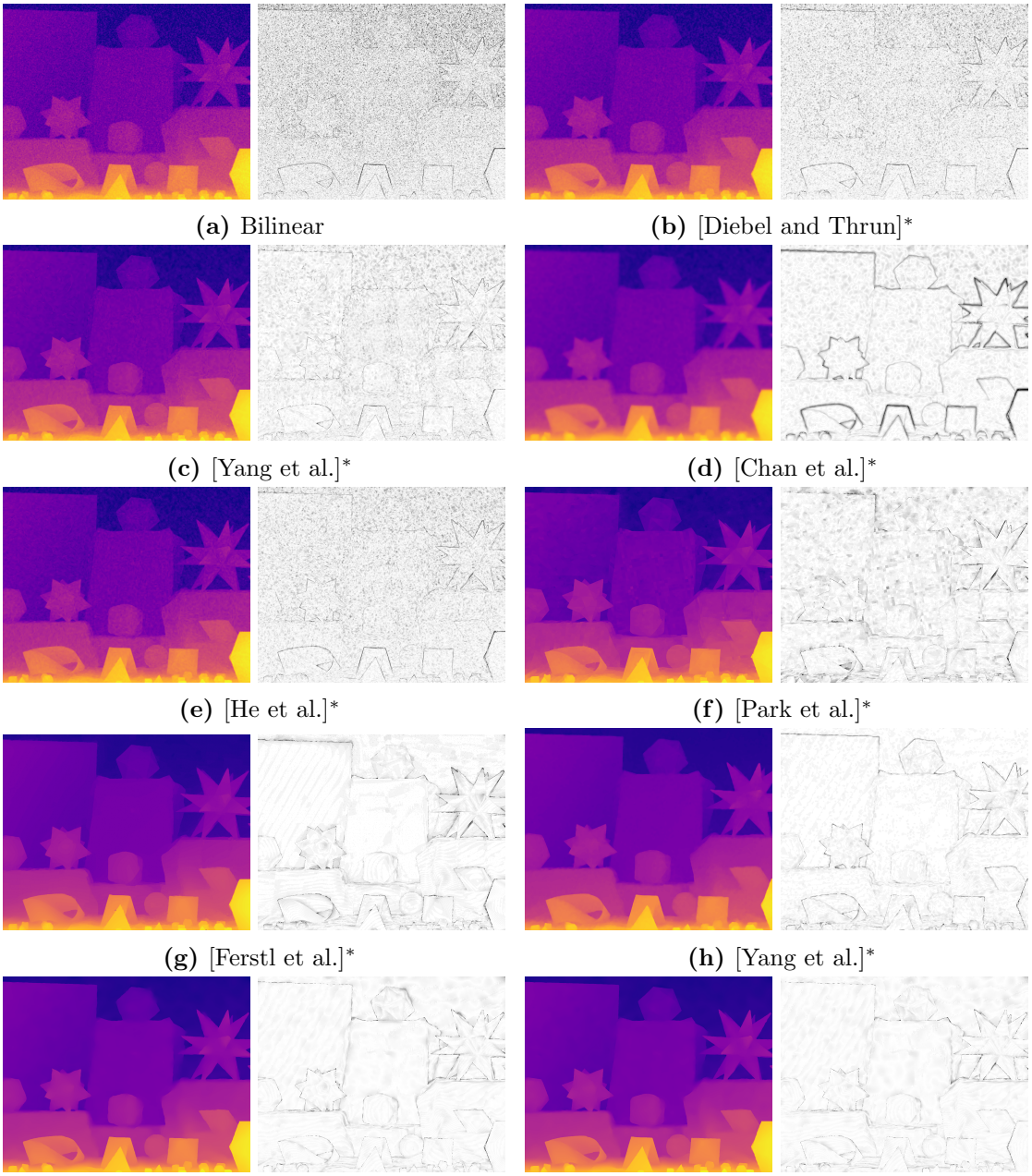
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.11: Qualitative results for the Noisy Middlebury dataset disparity map *Books* and an up-sampling factor of $\times 4$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bilinear **(b)** [Diebel and Thrun]*

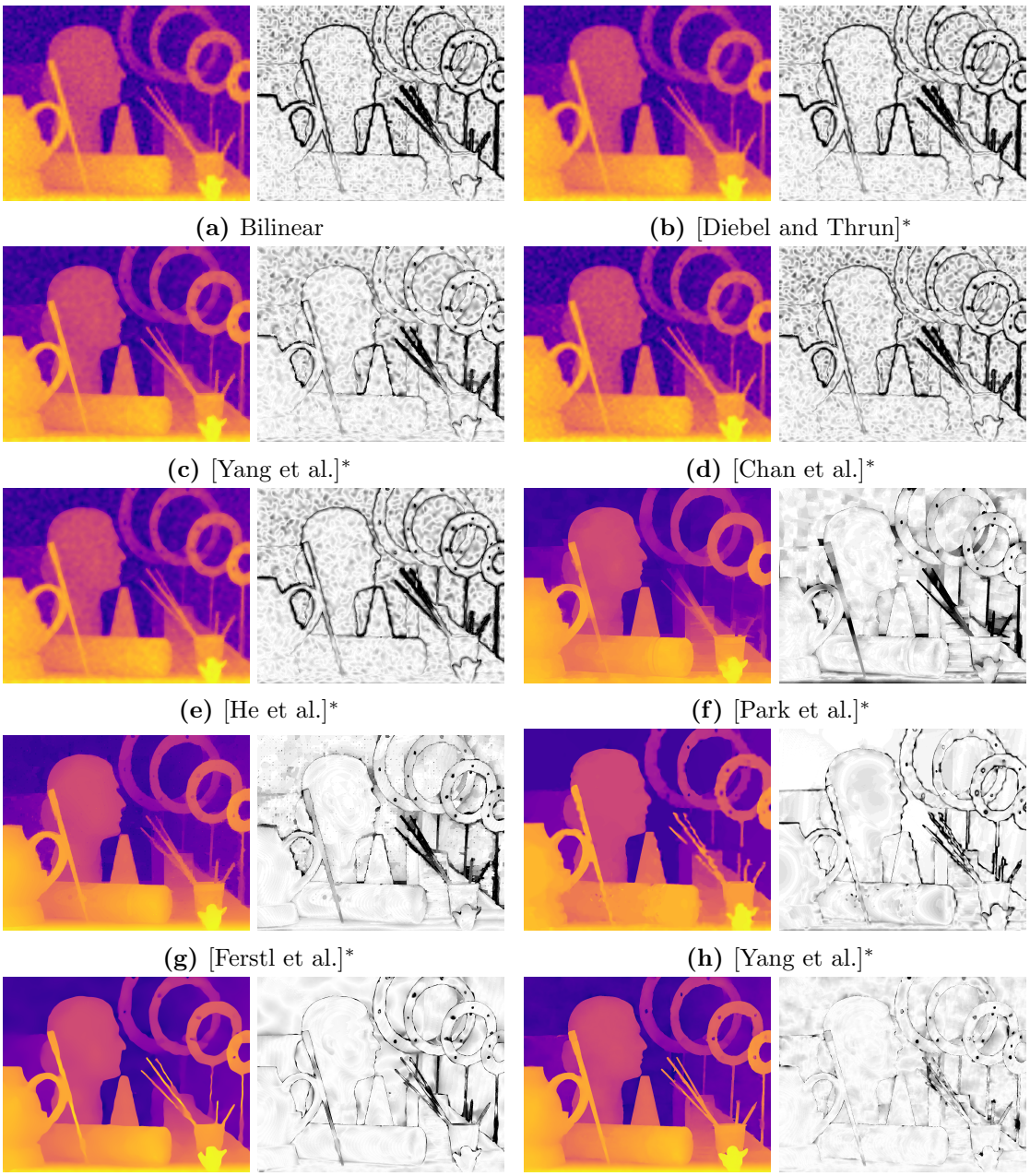
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.12: Qualitative results for the Noisy Middlebury dataset disparity map *Moebius* and an up-sampling factor of ×4. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bilinear **(b)** [Diebel and Thrun]*

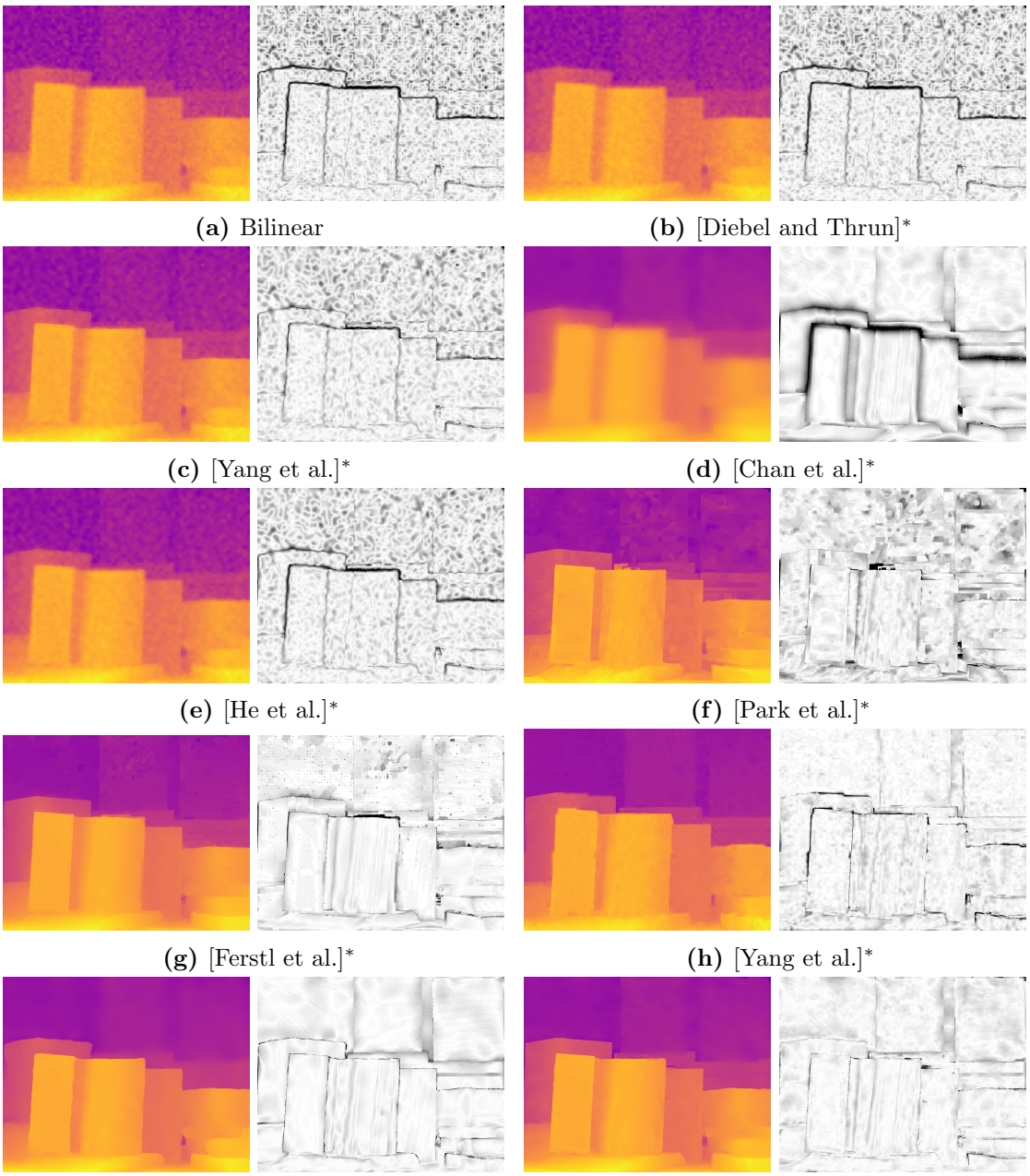
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.13: Qualitative results for the Noisy Middlebury dataset disparity map *Art* and an up-sampling factor of $\times 16$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bilinear **(b)** [Diebel and Thrun]*

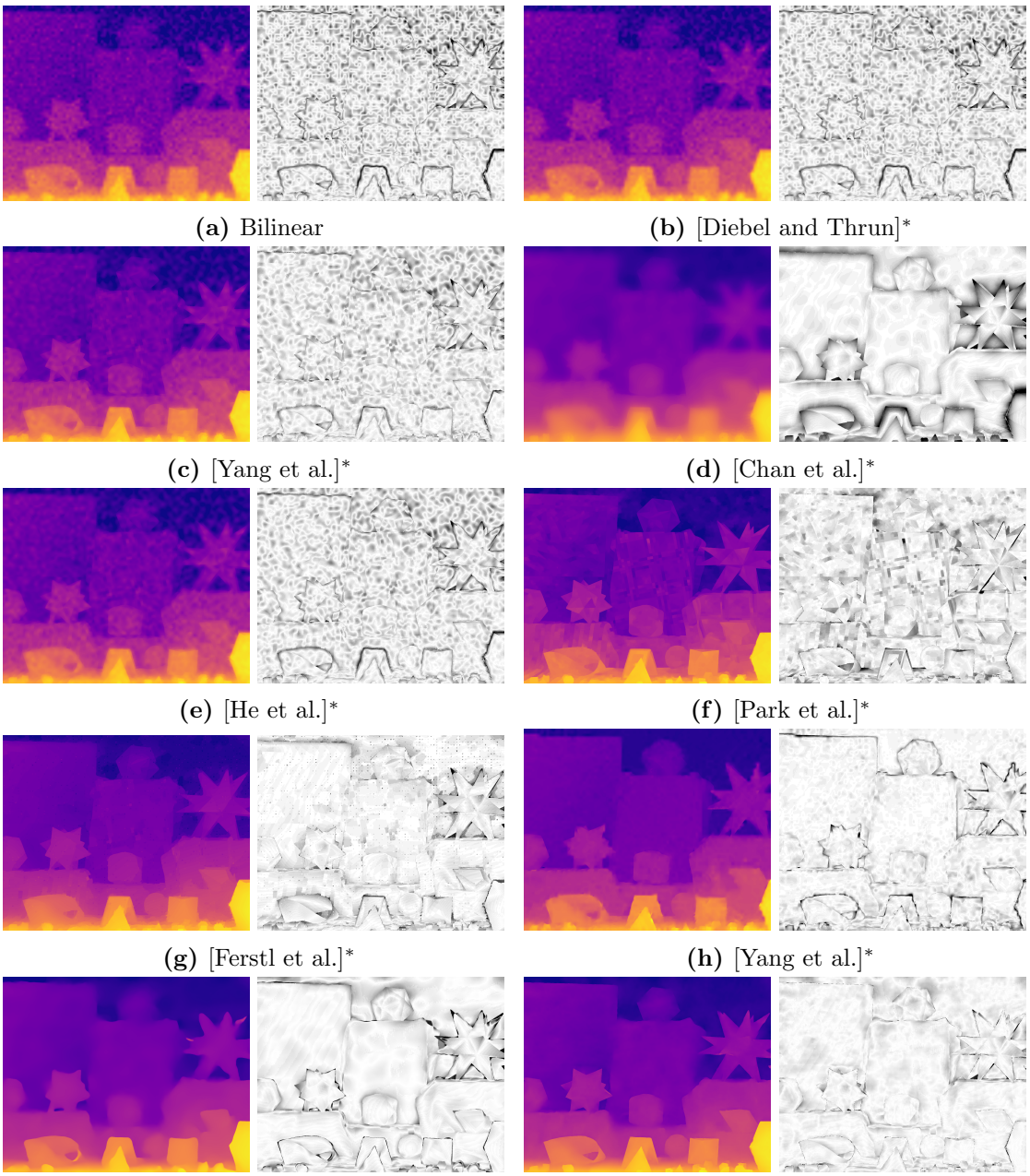
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.14: Qualitative results for the Noisy Middlebury dataset disparity map *Books* and an up-sampling factor of $\times 16$. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bilinear **(b)** [Diebel and Thrun]*

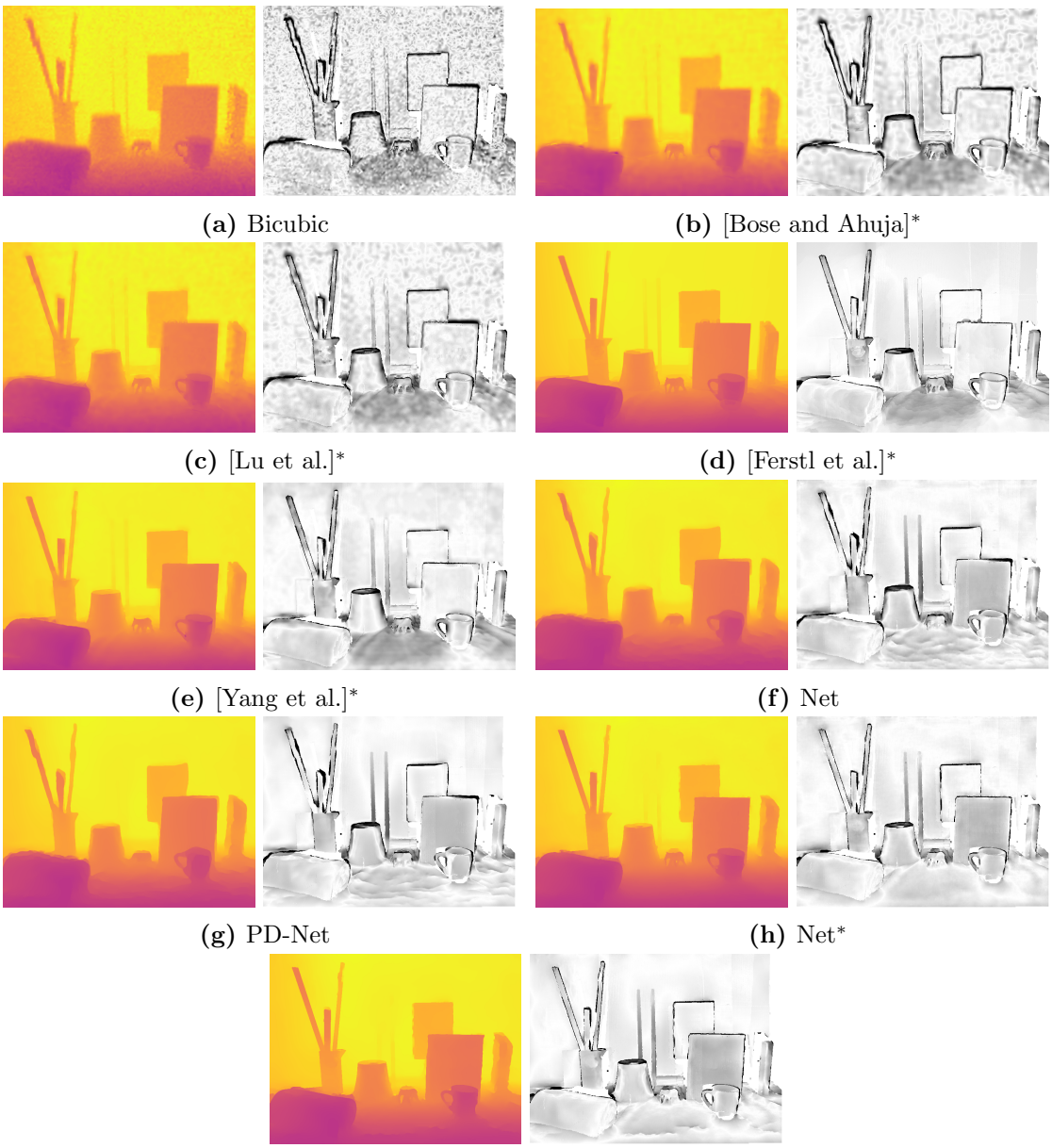
(c) [Yang et al.]* **(d)** [Chan et al.]*

(e) [He et al.]* **(f)** [Park et al.]*

(g) [Ferstl et al.]* **(h)** [Yang et al.]*

(i) PD-Net **(j)** PD-Net*

Figure 3.15: Qualitative results for the Noisy Middlebury dataset disparity map *Moebius* and an up-sampling factor of ×16. The first, colored image shows the high-resolution estimate and the second image shows the error of this estimate with respect to the ground-truth.



(a) Bicubic **(b)** [Bose and Ahuja]*

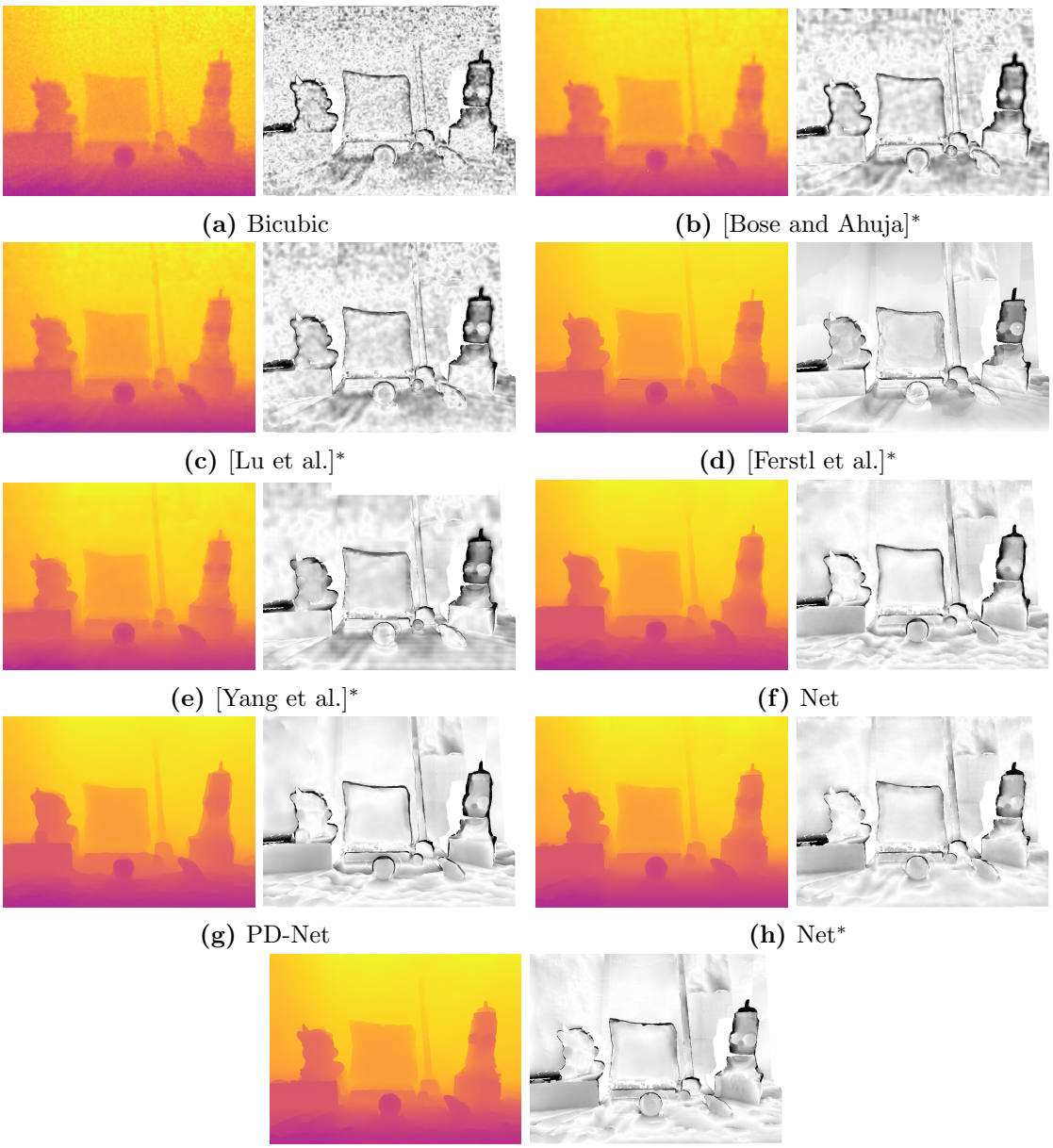
(c) [Lu et al.]* **(d)** [Ferstl et al.]*

(e) [Yang et al.]* **(f)** Net

(g) PD-Net **(h)** Net*

(i) PD-Net*

Figure 3.16: Qualitative results for the ToFMark image *Books*.



(a) Bicubic **(b)** [Bose and Ahuja]*

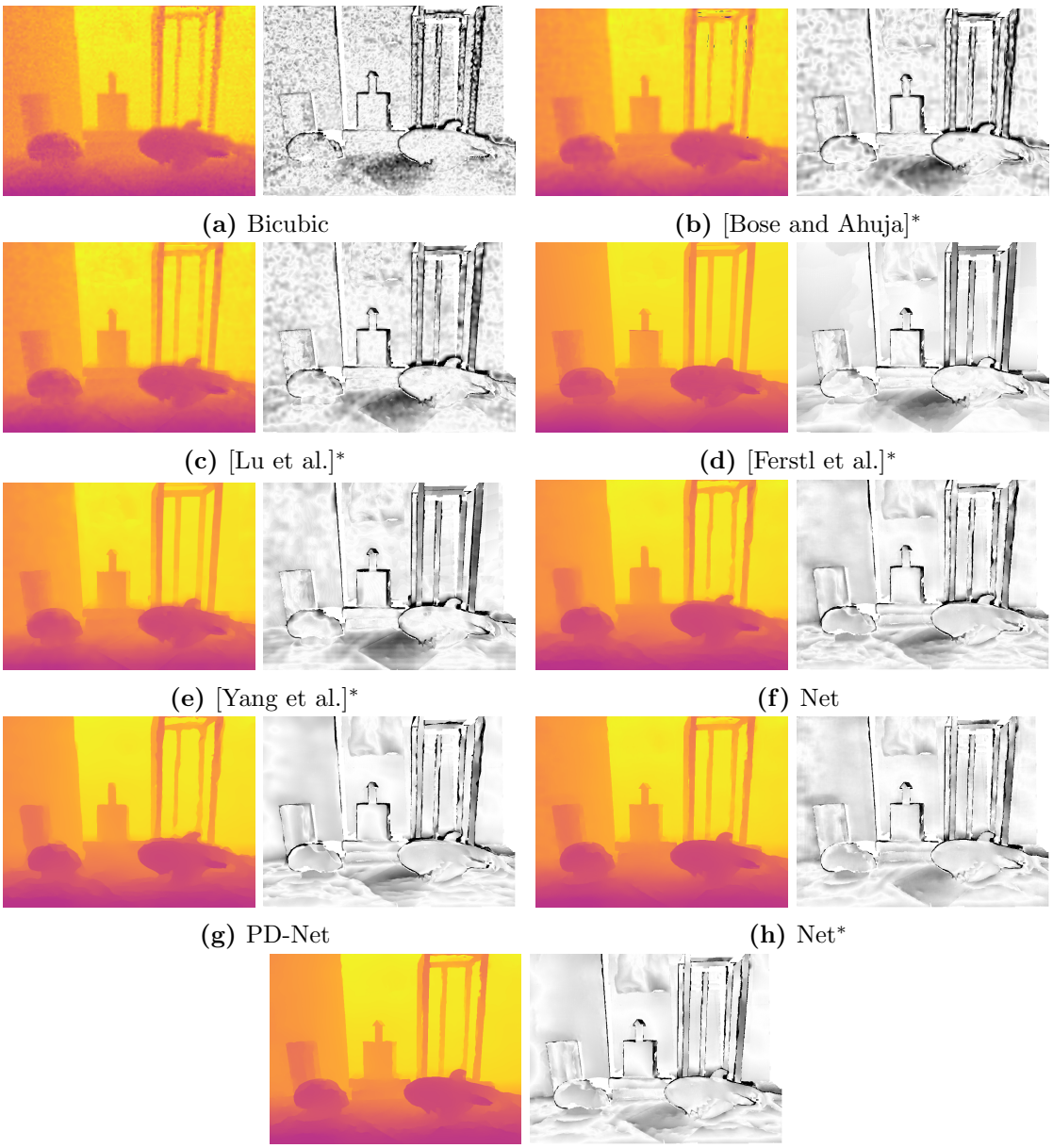
(c) [Lu et al.]* **(d)** [Ferstl et al.]*

(e) [Yang et al.]* **(f)** Net

(g) PD-Net **(h)** Net*

(i) PD-Net*

Figure 3.17: Qualitative results for the ToFMark image *Devil*.



(a) Bicubic **(b)** [Bose and Ahuja]*

(c) [Lu et al.]* **(d)** [Ferstl et al.]*

(e) [Yang et al.]* **(f)** Net

(g) PD-Net **(h)** Net*

(i) PD-Net*

Figure 3.18: Qualitative results for the ToFMark image *Shark*.

Table 3.8: Evaluation of unguided depth super-resolution on the ToFMark dataset for different up-sampling factors. The RMSE and MAE values are in mm and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow. Methods that use an additional high-resolution intensity image as guidance are marked with an asterisk *.

	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Bicubic	28.570	18.374	95.366	77.237	57.084
[Bose and Ahuja]*	28.914	16.596	94.215	71.870	49.444
[Lu et al.]*	26.157	15.791	94.046	71.261	48.412
[Ferstl et al.]*	25.693	14.066	93.759	69.149	41.220
[Yang et al.]*	26.314	14.877	91.944	67.131	43.581
Net	25.598	13.760	92.526	66.135	41.701
Net+PD	25.141	13.721	92.699	66.370	41.521
PD-Net	25.653	13.406	92.303	64.358	39.368

3.4.6 Guided Depth Super-Resolution

In the last section we evaluated our method for unguided depth super-resolution. However, in situations where an additional high-resolution image is available, it provides valuable, additional information about possible depth discontinuities. Hence, we first show in Section 3.4.4 two alternatives on how we can incorporate this additional input into our network architecture and analyze them. Then, we evaluate our guided method with the jointly trained variational method on top on the noisy Middlebury dataset (Section 3.4.6.2) and the ToFMark dataset (Section 3.4.6.3).

3.4.6.1 Guided Network

For the unguided depth super-resolution experiments in the previous evaluations we used the architecture *MSRCNN(d,5,r)* as depicted in Figure 3.6g. The question is, how can we incorporate an additional high-resolution intensity image as guidance? A simple concatenation of the input is not possible, as the low-resolution depth $\mathbf{d}^{(\mathrm{lr})}$ and the high-resolution guidance $\mathbf{g}$ have different resolutions. Hence, we propose two variants on how the guidance $\mathbf{g}$ can be incorporated without loss of information.

Guided Network Architectures One way to include the high-resolution guidance information in the convolutional network is by concatenating it with the last feature stage of our architecture. Prior to concatenation, the guidance image can be first processed by a small convolutional network. This architecture is depicted in Figure 3.19a and denoted as *Guided Stage Net(n)*, where n is the number of convolutional layers of the guidance network part. The special case $n = 0$ denotes a direct concatenation of the intensity values with the features of the last stage of the depth network part. The drawback of this incorporation is that the first stages of the depth network part cannot build upon guidance features. We circumvent this by adding pooling layers to the guidance network and concatenating guidance features and depth features on all resolutions. The architecture is visualized

Table 3.9: Evaluation of the guidance networks on the synthetic validation set and the noisy Middlebury test set for an up-sampling factor of ×4. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow.

	RMSE		MAE		$OL_{>1}$		$OL_{>5}$		$OL_{>10}$	
	val	test	val	test	val	test	val	test	val	test
Unguided Net	3.280	2.093	0.910	0.867	21.712	24.394	1.433	1.108	0.609	0.437
Guided Stage Net(0)	2.613	2.128	0.843	0.879	22.565	25.315	1.096	1.071	0.414	0.435
Guided Stage Net(3)	2.537	2.227	0.884	0.935	24.683	27.475	1.148	1.285	0.382	0.457
Guided Inverse Net(1)	2.519	2.136	0.822	0.856	22.594	24.472	0.942	0.921	0.349	0.418
Guided Inverse Net(3)	2.423	2.158	0.792	0.845	21.720	23.427	0.846	0.947	0.324	0.424
Guided Inverse Net(5)	2.287	2.115	0.789	0.889	22.030	26.748	0.811	0.944	0.300	0.431

in Figure 3.19b and denoted as *Guided Inverse Net(n)* in our evaluation, where n is the number of convolutional layers per guidance stage.

Results The quantitative results of the guidance networks are presented in Table 3.9. We observe that the *Guided Stage Net(n)* architecture with $n \in \{0, 3\}$ actually performs worse than the unguided network with respect to the MAE and $OL_{>1}$ on the test set. It is further interesting that the architecture with $n = 3$ performs even worse than the one with $n = 0$. The incorporation of the guidance features in the last stage might therefore not be sufficient to improve the performance. In contrast, the *Guided Inverse Net(n)* architecture yields a better MAE with $n \in \{1, 3\}$, but larger guidance networks, i.e. $n = 5$, again decrease in performance. Therefore, we use the *Guided Inverse Net(3)* in all our following guided evaluations.

3.4.6.2 Noisy Middlebury

The goal of this evaluation is to assess the influence of the guided architecture on the results on the noisy Middlebury dataset as proposed by [172]. For more details on the dataset refer to Section 3.4.5.2.

Methods We include the same baseline and state-of-the-art methods in this evaluation as in the evaluation of our unguided depth super-resolution method. Additionally, we evaluate our guided network architecture as presented in the last section denoted as *Net**. We further evaluate the guided network with a variational model as post-processing step named *Net+PD** and our final guided method, where we trained the network and the variational method end-to-end is denoted as *PD-Net**

Results The quantitative results of the evaluation are summarized in Table 3.10. Qualitative results for up-sampling factors of ×4 and ×16 are visualized in Figures 3.10-3.15. First, we note that the small errors, i.e. $OL_{>1}$, are least affected by the guided architecture and are the only metric where our method does not outperform previous methods.

Figure 3.19: Guided network architectures used in the evaluation. The different network operations are color-coded and the parameters of the operations are given beneath the operation name. A non-linearity is applied after each hidden convolutional layer.

However, we see that the guided network *Net** improves the MAE over the unguided network over all up-sampling factors. Second, our guided joint model *PD-Net** yields the best overall results on all up-sampling rates in terms of MAE. While the improvement of the guided model over the unguided model is small for low up-sampling factors, it is more pronounced for higher up-sampling factors, i.e. $\times 8$ and $\times 16$. Finally, most of the reduction of error seems to stem from reducing large errors, as $\text{OL}_{>5}$ and $\text{OL}_{>10}$ most significantly decreased by the guided model. This is in accordance with our intuition. For those more difficult scenarios, the high-resolution guidance helps resolving ambiguities, especially for thin objects. This is also supported by the qualitative results.

3.4.6.3 ToFMark

In our final evaluation we test our guided model on the challenging Time-of-Flight dataset ToFMark [67]. Details on the dataset can be found in Section 3.4.5.3.

Methods We include the same baseline and state-of-the-art methods as in the evaluation of our unguided depth super-resolution method. Additionally, we evaluate our guided network architecture as presented in the last section denoted as *Net**. We further evaluate the guided network with a variational model as post-processing step named *Net+PD**. Our final guided method, where we trained the network and the variational method end-to-end is denoted as *PD-Net**

Results The quantitative results of our guided depth super-resolution evaluation on the ToFMark dataset are shown in Table 3.11. Qualitative results are visualized in Figures 3.16-3.18. As in the previous evaluation, our guided models clearly improve the MAE over the unguided models and obtain new state-of-the-art results. However, we also improve on $\text{OL}_{>1}$ in addition to $\text{OL}_{>5}$ and $\text{OL}_{>10}$. Those improvements are also clearly visible in the qualitative results, where the guided architecture mostly affects thin structures and sharp depth discontinuities.

3.5 Summary & Discussion

In this chapter, we presented a new method for depth super-resolution. Depth maps can be understood as a 2.5D representation, where each pixel encodes the distance to the first surface along the viewing ray. Nowadays there exist a broad range of active sensors that can capture this information directly, e.g. by using the time-of-flight principle. However, those depth maps usually have a low spatial resolution and are degraded for example by depth-dependent noise. The depth super-resolution problem is to increase the spatial resolution of the 2.5D depth maps while simultaneously reducing the noise effects such that the super-resolved output is as close as possible to the high-resolution projection of the scene.

Table 3.10: Evaluation of guided depth super-resolution on the noisy Middlebury dataset for different up-sampling factors. The RMSE and MAE values are in pixel disparities and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow. Methods that use an additional high-resolution intensity image as guidance are marked with an asterisk *.

	×2					×4				
	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Nearest Neighbour	6.434	4.776	77.842	32.823	9.436	6.858	4.966	78.345	33.867	10.262
Bilinear	4.243	3.070	65.927	15.201	2.258	4.832	3.384	67.561	17.971	3.697
[Diebel and Thrun]*	2.560	1.475	34.643	2.094	0.718	3.540	2.304	53.549	7.281	1.400
[Yang et al.]*	2.265	1.243	26.157	1.940	0.628	2.941	1.679	38.363	3.461	1.059
[Chan et al.]*	2.535	1.409	29.201	3.291	0.974	3.332	1.845	36.608	6.305	1.928
[He et al.]*	2.800	1.765	42.582	3.642	0.811	3.327	2.080	48.216	5.686	1.384
[Park et al.]*	2.553	1.087	19.223	1.499	0.691	3.226	1.578	33.566	3.136	1.036
[Lu and Forsyth]*	2.879	1.397	28.692	2.451	1.069	3.473	1.646	32.943	3.774	1.553
[Zhang et al.]*	2.737	1.059	32.672	1.431	0.722	2.973	1.485	50.401	2.343	0.921
[Shen et al.]*	2.973	1.566	46.851	4.992	2.161	3.808	2.268	63.547	8.449	2.904
[Yang]*	2.546	1.501	50.148	3.410	0.743	3.107	1.757	56.033	4.594	1.129
[Ferstl et al.]*	2.062	0.637	7.166	1.264	0.640	2.768	0.981	14.012	2.326	1.041
[Yang et al.]*	1.825	0.564	5.771	0.650	0.315	2.156	0.811	11.561	1.477	0.637
[Barron and Poole]*	2.009	0.991	29.739	1.743	0.758	2.591	1.337	40.841	3.245	1.253
Net	1.350	0.604	13.521	0.480	0.208	2.124	0.884	25.325	1.141	0.447
Net+PD	1.290	0.548	10.477	0.493	0.195	2.037	0.838	22.413	1.175	0.435
PD-Net	1.292	0.521	9.096	0.446	0.207	2.046	0.762	18.305	0.981	0.437
Net*	1.380	0.603	13.412	0.495	0.223	2.158	0.845	23.427	0.947	0.424
Net+PD*	1.326	0.550	10.399	0.510	0.213	2.068	0.786	19.637	0.954	0.408
PD-Net*	1.336	0.514	8.500	0.486	0.232	2.062	0.757	18.563	0.835	0.427

	×8					×16				
	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Nearest Neighbour	7.546	5.287	78.731	35.328	11.794	8.767	5.991	80.172	38.718	14.809
Bilinear	5.576	3.787	69.073	20.870	5.676	6.842	4.540	71.792	25.942	8.971
[Diebel and Thrun]*	4.873	3.267	64.514	15.996	4.045	6.560	4.350	70.718	24.490	8.270
[Yang et al.]*	3.617	2.105	47.499	5.720	1.494	5.510	3.549	65.685	17.863	4.633
[Chan et al.]*	4.590	2.679	48.021	12.557	4.348	6.567	4.006	60.265	22.378	9.051
[He et al.]*	4.239	2.744	58.210	11.026	2.726	5.869	3.825	66.807	20.067	6.305
[Park et al.]*	4.152	2.270	48.252	6.939	1.922	6.217	3.507	62.203	15.577	5.044
[Lu and Forsyth]*	4.327	2.134	41.249	6.353	2.418	5.595	3.077	54.996	12.916	4.538
[Zhang et al.]*	3.888	2.414	67.968	9.425	1.978	5.976	4.043	79.970	26.333	7.073
[Shen et al.]*	5.021	3.291	75.230	18.464	4.951	6.678	4.532	81.649	30.298	9.745
[Yang]*	3.948	2.246	63.147	8.147	2.119	5.455	3.211	72.121	16.069	4.508
[Ferstl et al.]*	3.376	1.530	27.237	4.278	1.830	4.904	2.534	44.916	9.381	3.604
[Yang et al.]*	3.114	1.335	23.058	3.444	1.414	4.854	2.261	40.012	7.242	3.227
[Barron and Poole]*	3.375	1.855	52.477	6.230	2.203	4.736	2.809	66.191	13.057	4.338
Net	3.107	1.349	39.759	2.879	1.007	4.812	2.372	58.436	9.263	3.037
Net+PD	3.017	1.306	37.760	2.904	1.032	4.715	2.312	56.562	9.136	3.055
PD-Net	3.124	1.231	34.590	2.397	1.008	4.819	2.215	53.831	8.037	2.991
Net*	2.776	1.232	39.813	1.879	0.653	3.793	1.970	57.177	6.485	1.504
Net+PD*	2.698	1.177	37.032	1.904	0.642	3.793	1.970	57.177	6.485	1.504
PD-Net*	2.727	1.117	34.295	1.689	0.652	3.728	1.870	55.586	5.393	1.334

Table 3.11: Evaluation of guided depth super-resolution on the ToFMark dataset for different upsampling factors. The RMSE and MAE values are in mm and the $OL_{>x}$ values are percentages. The best entries are highlighted in **orange**, the second best in yellow. Methods that use an additional high-resolution intensity image as guidance are marked with an asterisk *.

	RMSE	MAE	$OL_{>1}$	$OL_{>5}$	$OL_{>10}$
Bicubic	28.570	18.374	95.366	77.237	57.084
[Bose and Ahuja]*	28.914	16.596	94.215	71.870	49.444
[Lu et al.]*	26.157	15.791	94.046	71.261	48.412
[Ferstl et al.]*	25.693	14.066	93.759	69.149	41.220
[Yang et al.]*	26.314	14.877	91.944	67.131	43.581
Net	25.598	13.760	92.526	66.135	41.701
Net+PD	25.141	13.721	92.699	66.370	41.521
PD-Net	25.653	13.406	92.303	64.358	39.368
Net*	24.789	12.906	92.611	65.813	39.496
Net+PD*	24.288	12.871	92.555	65.844	39.536
PD-Net*	25.384	12.747	91.554	62.949	37.654

The core contribution presented in this chapter is a deep learning based method to this problem. While in single image super-resolution for color images deep learning based methods are already the state-of-the-art, this is only recently swapping over to the depth domain. The major issue is the availability of training data, i.e. for each noisy input sample how to obtain a high-resolution, noise-free ground-truth target.

We demonstrated that we already outperform most of the recently proposed depth super-resolution methods, even those that rely on an additional high-resolution guidance intensity image by training a deep convolutional network on synthetically rendered depth maps. Although we tried to model some effects of active depth sensors on the input depth maps with depth-dependent noise on our rendered data, we still observed that a variational post-processing of the network outputs further improved the results. This indicates that the prior knowledge encoded in the variational method helped to deal with the domain shift between the synthetic training data and the testing data, e.g. time-of-flight depth maps. Motivated by this observation, we proposed a framework to train the variational method on top of the deep convolutional network to further boost the depth super-resolution accuracy. We chose to unroll the optimization scheme of the variational method and utilized automatic differentiation to backpropagate gradient information through the joint model. With this technique, we were able to train the convolutional network to adapt to the variational post-processing, but also all the adjustable parameters of the variational model and its optimization scheme.

In our extensive evaluations, we could show the clear benefit of our joint model trained on synthetic data over recent baseline methods. Interestingly, the choice of the variational method on top is less critical than training both parts end-to-end. We observed that training only one part of the joint method while freezing the other lead to inferior results than joint training. Our conclusion is that the adaption of the convolutional network to

the post-processing is as important as tuning the parameters of the variational method. Additionally, we extended our proposed method to guided depth super-resolution. Hence, given a high-resolution color information along with the low-resolution depth map, we could further improve our estimates.

CHAPTER 4

Deep Learning for 3D

Contents

4.1 Introduction . . . 103
4.2 Related Work . . . 106
4.3 Deep Learning for High-Resolution 3D . . . 113
4.4 Evaluation . . . 122
4.5 Summary & Discussion . . . 155

While 2.5D data itself enables a wide range of new applications, e.g. pose estimation, or robot navigation, it is very natural to reason about the world in 3D. For example, an object can not only be categorized by its various 2D projections in a set of images, but also by its 3D geometry. This might be especially useful for objects, where the projection is ambiguous. But also the other direction, the reconstruction of 3D environments from a number 2D inputs, is an exciting topic by itself. In this chapter, we approach both aspects of 3D data within the context of deep learning, where the main challenge is the memory consumption that increases drastically with the resolution of the 3D volume. Therefore, we will present an efficient method that can handle significantly larger input resolutions and demonstrate its benefits on a number of different applications.

4.1 Introduction

Over the last several years, convolutional networks have led to substantial performance gains in many areas of computer vision. In most of these cases, the input to the network is a 2D pixel grid, e.g. in image classification [102, 132, 220, 228], object detection [85, 86, 147, 178, 179, 180] or semantic segmentation [6, 34, 35, 81, 148, 267]. However, recent advances

in 3D reconstruction [165] and graphics [111] allow capturing and modeling large amounts of 3D data. At the same time, large 3D repositories such as ModelNet [259], ShapeNet [30] or 3D Warehouse[1] as well as databases of 3D object scans [39] are becoming increasingly available. These factors have motivated the development of convolutional networks that operate on 3D data.

Most existing 3D network architectures [41, 154, 176, 259] replace the 2D pixel array by its 3D analogue, i.e. a dense and regular 3D voxel grid, and process this grid using 3D convolution and pooling operations. This allows the reuse of network operations such as convolution and pooling, but also of successful network architectures as for example ResNet [102]. However, for dense 3D data, computational and memory requirements grow cubically with the resolution. Consequently, existing 3D networks are limited to low 3D resolutions, typically in the order of 30^3 voxels. To fully exploit the rich and detailed geometry of our 3D world, however, much higher resolution networks are required.

In this work, we build on the observation that 3D data is often sparse in nature, e.g. point clouds, or meshes, resulting in wasted computations when applying 3D convolutions naïvely. We illustrate this in Figure 4.1 for a 3D classification example. Given the 3D meshes of [259] we voxelize the input at a resolution of 64^3 and train a simple 3D convolutional network to minimize a classification loss. We depict the maximum of the responses across all feature maps at different layers of the network. It is easy to observe that high activations occur only near the object boundaries.

Motivated by this observation, we propose *OctNet*, a 3D convolutional network that exploits this sparsity property. Our OctNet hierarchically partitions the 3D space into a set of shallow octrees [158]. Each octree splits the 3D space according to the density of the data. More specifically, we recursively split octree nodes that contain a data point in its domain, i.e. 3D points, or mesh triangles, stopping at the finest resolution of the tree. Therefore, leaf nodes vary in size, e.g. an empty leaf node may comprise up to $8^3 = 512$ voxels for a tree of depth 3 and each leaf node in the octree stores a pooled summary of all feature activations of the voxel it comprises. The convolutional network operations are directly defined on the structure of these trees. Therefore, our network dynamically focuses computational and memory resources, depending on the 3D structure of the input. This leads to a significant reduction in computational and memory requirements which allows for deep learning at high resolutions. Importantly, we also show how essential network operations (convolution, pooling, and unpooling) can be efficiently implemented on this new data structure.

The main limitation of using an octree within the network architecture, however, is that the octree representation is derived from the input and fixed during learning and inference. While this is sufficient for tasks such as 3D classification or semantic segmentation where the input and the output share the same octree representation, the OctNet framework does not directly apply to tasks where the 3D space partitioning of the output is unknown

[1] `https://3dwarehouse.sketchup.com`, last accessed on October 26, 2017

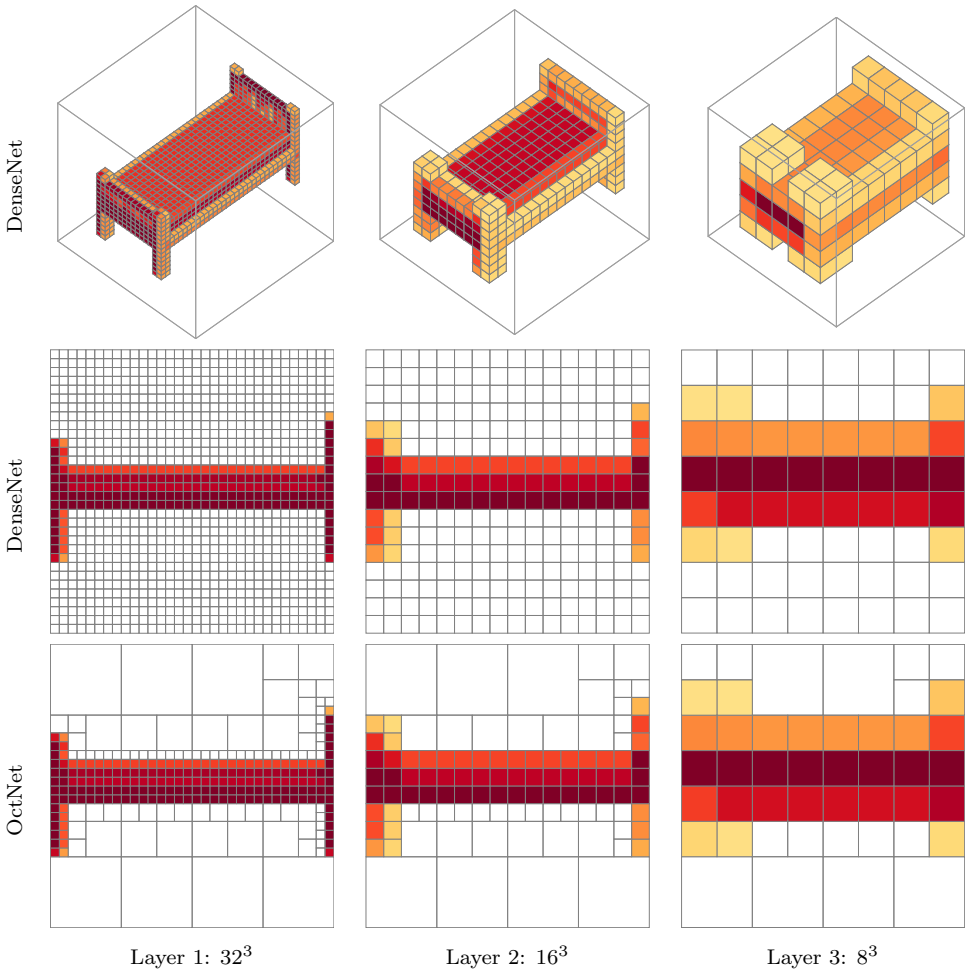


Figure 4.1: Sparse activations of a 3D convolutional network. For illustration purposes, we trained a dense convolutional network to classify 3D shapes from [259]. Given a voxelized bed as input, we show the maximum response across all feature maps at intermediate layers of the network before pooling. Higher activations are indicated with darker colors. Voxels with zero activation are not displayed. The first row visualizes the responses in 3D while the second row shows a 2D slice. Note how voxels close to the object contour respond more strongly than voxels further away. We exploit the sparsity in our data by allocating memory and computations using a space partitioning data structure, where larger cells share memory and computation (bottom row).

a priori and may be different from the input. In particular, for tasks such as volumetric depth fusion and volumetric depth completion [38, 45, 165, 166, 224, 256] the location of the implicit surface is unknown and needs to be inferred from noisy observations.

In the second part of this chapter, we lift this restriction. More specifically, we propose a novel 3D CNN architecture called *Dynamic OctNet* which takes as input one or more depth images and estimates both the 3D reconstruction and its supporting 3D space partitioning, i.e. the octree structure of the output. We apply this architecture to the depth map fusion problem and formulate the task as the prediction of truncated signed distance fields which can be meshed using standard techniques [149].

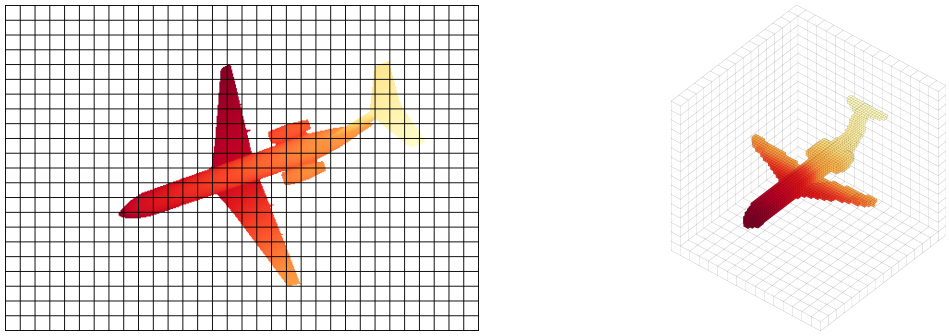
We demonstrate the utility of the proposed OctNet on three different problems involving three-dimensional data: 3D classification, 3D orientation estimation, and semantic segmentation of 3D point clouds. In particular, we show that the proposed OctNet enables significant higher input resolutions compared to dense inputs due to its lower memory consumption while achieving identical performance compared to the equivalent dense network at lower resolutions. At the same time, we gain significant speed-ups at resolutions of 128^3 and above. Using our OctNet, we investigate the impact of high-resolution inputs wrt. accuracy on the three tasks and demonstrate that higher resolutions are particularly beneficial for orientation estimation and semantic point cloud labeling.

Additionally, we evaluate our Dynamic OctNet on synthetic and real-world datasets for volumetric depth fusion and completion. Our experiments demonstrate that the proposed method is able to reduce noise and outliers compared to standard methods, i.e. vanilla TSDF fusion [45, 165] while avoiding the shrinking bias of local regularizers such as TV-L1 [268]. Besides, our model learns to complete missing surfaces and fills in holes in the reconstruction. We demonstrate the flexibility of our model by evaluating it on the task of volumetric shape completion from a single view where we obtain improvements with respect to the state-of-the-art [71].

The remainder of this chapter is organized as follows: We present the integration of an efficient space partition data structure into a 3D convolutional network named OctNet in Section 4.3.1. We further extend this framework to allow the use of OctNet also on a priori unknown output structures, i.e. tasks like depth completion, or depth fusion, where the 3D geometry is not known in advance, in Section 4.3.2. In Section 4.4 we show a large variety of evaluations that demonstrate the effectiveness of OctNet in terms of memory consumption and computational resources. Further, we show that high-resolution inputs and outputs are needed to increase accuracy on tasks like 3D shape classification, 3D orientation estimation, 3D semantic segmentation, 3D depth fusion and 3D depth completion.

4.2 Related Work

In this section, we first review the existing body of work on dense models in Section 4.2.1, i.e. convolutional network models that directly operate on 3D voxel grids, which are the 3D equivalent of 2D pixel grids. The main drawback of those methods is the memory consumption that increases cubically with respect to the input resolution. Therefore, we survey the literature on deep learning methods that exploit the sparsity in 3D data in Section 4.2.2. Our method relies on efficient space partitioning function to focus memory and computation on relevant regions. Alternative data structures are reviewed in Section 4.2.3. In the second part of this chapter, we utilize our method for depth fusion and completion. Hence, in Section 4.2.4 we present previous work on volumetric depth fusion and Section 4.2.5 is dedicated to related work on volumetric shape completion.



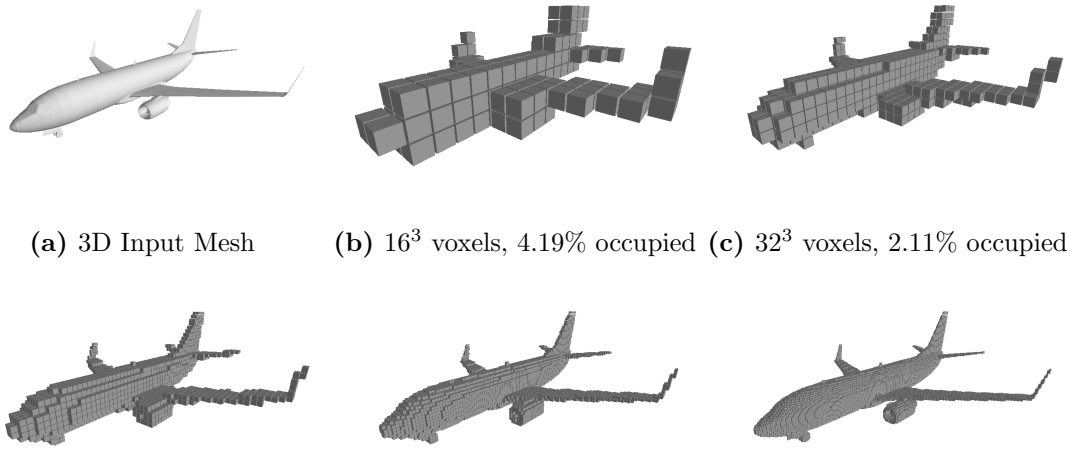
(a) 2D Pixel Grid **(b)** 3D Voxel Grid

Figure 4.2: The 3D equivalent of the 2D pixel grid as depicted in (a) is the 3D voxel grid as depicted in (b).

4.2.1 Dense Models

The natural continuation of convolutional networks from 2D to 3D data is by going from the 2D pixel grid to the 3D voxel grid, see Figure 4.2. There has recently been an increased interest in those 3D convolutional networks. One application is 3D shape classification. For example, Wu et al. [259] trained a deep belief network to classify 3D shapes. Therefore, they discretized the meshes to a coarse 30^3 voxel grid. Similarly, Maturana and Scherer [154] proposed VoxNet, a simple 3D convolutional network that consists of 3 strided convolutional layers, and two fully-connected layers. The authors also voxelized the input into a coarse 32^3 grid that is feed to the network. In a follow-up work, Alvar et al. [4] showed that introducing an auxiliary orientation loss increases the classification performance over the original VoxNet. Interestingly, Su et al. [225] demonstrated in their work that applying 2D convolutional networks pre-trained on ImageNet and then fine-tuned on a set of rendered views of the 3D object perform significantly better with respect to the classification accuracy than the previously published 3D networks. Qi et al. [176] closely investigated this gap between convolutional networks based on volumetric representations and convolutional networks based on rendered 2D multi-view representations. They proposed two new volumetric architectures: (i) a VoxNet like convolutional network with sub-volume supervision where sub-volumes are classified independently and (ii) a convolutional network with anisotropic probing kernels that learns the projection to a 2D image that is then further processed by a pre-trained network. Additionally, they further improve the accuracy of the multi-view classification network by using a multi-scale approach within the network architecture. While the 3D network extension where able to narrow the gap between those two approaches, the multi-view networks still perform better on the shape classification task. In the end, the authors argue that the coarse 3D voxel grid resolution is the main bottleneck.

Another task on dense 3D representations despite classification is for example embedding learning. Girdhar et al. [84] learned a low-dimensional embedding to generate 20^3



(a) 3D Input Mesh **(b)** 16^3 voxels, 4.19% occupied **(c)** 32^3 voxels, 2.11% occupied

(d) 64^3 voxels, 1.06% occupied **(e)** 128^3 voxels, 0.56% occupied **(f)** 256^3 voxels, 0.31% occupied

Figure 4.3: Sparsity of 3D data. To classify triangle meshes as depicted in (a) with 3D convolutional networks the typical approach is to voxelize the input (b)-(f). As we increase the resolution of the voxelization from 16^3 to 256^3 more and more details can be represented, but also the memory increases drastically. However, the percentage of occupied voxels also decreases along with the resolution increase. This sparsity property of 3D data can be exploited.

voxels 3D representations. The embedding can be obtained from a 3D input or a 2D image input. This is achieved by first training a convolutional auto-encoder on the voxelized 3D data with the low-dimensional embedding as bottleneck layer. Then a 2D convolutional neural network is trained to map the images to this low-dimensional embedding. Very similar is the concurrent work presented by Sharma et al. [212]. They only learn the low-dimension embedding for the voxelized 3D data and use it as feature representation in other tasks, e.g. shape classification, shape interpolation and shape denoising. A variational auto-encoder for the interpolation task between shapes was presented by Brock et al. [24]. In contrast to the normal auto-encoder, the variational auto-encoder produces much more realistic examples.

A semantic segmentation approach based on 3D convolutional networks was proposed by Huang and You [110]. The point cloud is divided into small local cubes and each of the cubes is voxelized into a 20^3 grid and then feed to a small convolutional network. A similar sliding cube approach was presented by Song and Xiao [221] for object recognition in 3D from depth maps. First, a so-called Region Proposal Network outputs 3D bounding boxes of interesting regions that are subsequently classified by an Object Recognition Network that takes the 3D points, but also 2D color information into account.

4.2.2 Sparse Models

A major drawback of the dense 3D voxel representation is the memory requirement that become increasingly problematic with larger input resolutions. Due to this memory limitations, all aforementioned methods are only able to process and generate shapes at a very coarse resolution, typically in the order of 30^3 voxels. However, in most domains the 3D data is sparse, i.e. point clouds, or triangle meshes, see Figure 4.3 for an example. This sparsity property can be exploited to handle higher resolution inputs in convolutional networks. In this section, we review the few network architectures that have been proposed so far to handle sparse data.

Engelcke et al. [62] proposed to only evaluate the convolutions at the sparse input locations by pushing values to their target locations. This has the potential to reduce the number of convolutions but does not reduce the amount of memory required. Consequently, their work considers only very shallow networks with up to three layers. A similar approach is presented by Graham in [94, 93] where sparse convolutions are reduced to matrix operations. Unfortunately, the model only allows for 2×2 convolutions and results in indexing and copy overhead which prevents processing volumes of higher resolution. Besides, each layer decreases sparsity and thus increases the number of operations, even at a single resolution. A different approach was proposed by Li et al. [141]. Instead of using convolutional layers in the high-resolution 3D volume they introduced field probing layers. A field probing layer contains 3D sampling points and computes the linear combination of the input values at those sampling points and the weights associated with those sampling points. Then, not only the weights but also the sampling point locations can be learned. The approach has the drawback that the field-probing layers cannot be stacked and the output is directly fed into fully-connected layers. Jampani et al. [120] introduced bilateral convolutional layers, which first map spare inputs into a permutohedral lattice, then applies a convolution operation on this lattice structure and finally, the result of is transformed back to the output space. A method that is explicitly designed for 3D point clouds is PointNet by Qi et al. [175]. They propose a multi-layer perceptron that is applied on each of the 3D points independently to get feature vectors. By pooling over the number of feature vectors a single global feature representation is obtained that can then again be concatenated to the point features. While this simple approach yields good classification and segmentation results, it does not take into account the local structure of the points.

Another related branch is dedicated Graph Convolutional networks [25, 49, 59, 142, 160] that try to generalize the idea of convolutional networks on regular graphs to arbitrary graphs. They are mostly formulated in the spectral domain [25, 49, 59] where the convolution is dependent on the Fourier basis or Laplacian eigenbasis. However, this makes those types of networks domain dependent and hence, the models learned on one graph cannot easily be applied to other graphs with varying basis.

Finally, the concurrent work of Wang et al. [254] on O-CNN is very similar to ours, as it also proposes to use an octree within a 3D convolutional network. It mainly differs

to our OctNet in the choice of the octree data structure and the definition of the network operations. While we define the convolution equivalent to a dense representation with the difference that feature vectors in larger octree cells are pooled, the authors of [254] perform the convolution only on the same octree depth. If a neighboring octant is not on the same depth, the features are assumed to be zero. This might hinder the propagation of information within the network.

4.2.3 Space Partitioning Functions

In our proposed method we utilize a special hybrid grid-octree data structure similar to the one proposed by Miller et al. [158] to efficiently partition the space and hence, enable a higher input resolutions by decreasing the memory consumption. But this is only one possible data structure to partition the 3D input space. One of the most popular ones is the general octree as pioneered by Meagher [157]. An octree is a tree structure where each octree cell can be further subdivided into eight equally sized octree cells. The division is performed recursively such that fine details are represented on deeper leaf nodes in the tree. This structure is heavily used in graphics for volume rendering [121, 135]. A related data structure is the k-d tree invented by Bentley [12]. Instead of dividing each node into eight equally space children, the split in a k-d tree is defined by a hyperplane that optimally splits one dimension of the input. While this structure is normally used for approximate nearest neighbor search, it was recently the main building block for a new type of network for 3D shape classification and segmentation introduced by Klokov and Lempitsky [129].

One drawback of those tree-based data structures is that deep trees are needed to represent fine resolutions. However, this decreases the performance of GPGPUs because the tree traversal leads to thread divergences. An alternative data structure that avoids this problem is based on voxel hashing [138, 236]. Each voxel position maps to an integer value, i.e. the hash, that determines the address within a hashmap. The hashmap then stores the data associated with the sparse voxels. Variants of this data structure have been used for real-time 3D reconstruction [33, 166].

4.2.4 Volumetric Fusion

In the second part of this chapter we will apply the proposed OctNet for volumetric depth fusion and shape completion. Hence, we also survey the related work on volumetric fusion in this section and on shape completion in the following one.

In their seminal work, Curless and Levoy [45] proposed to integrate range information across viewpoints by averaging truncated signed distance functions. The simplicity of this method has turned it into a universal approach that is used in many 3D reconstruction pipelines. Using the Microsoft Kinect sensor and GPGPU processing, Newcombe et al. [165] showed that real-time 3D modeling is feasible using this approach. Large-scale 3D reconstruction has been achieved using iterative re-meshing [256] and efficient data

structures [166, 224]. The problem of calibration and loop-closure detection has been considered in [38, 274]. Due to the simplicity of the averaging approach, however, these methods typically require numerous input views, are susceptible to outliers in the input and do not allow the prediction of surfaces in unobserved regions.

A problem with the simple averaging of truncated sign distance functions is that it can cause inconsistent surfaces if frequent sign changes appear [107]. Therefore, it is common to employ additional regularization terms that incorporates prior assumptions about the surface, i.e. smoothness assumption. Zach et al. [268] proposed the popular TV-L1 model for volumetric fusion, where a total variation term is utilized for regularization. The total variation favors minimal surfaces. Additionally, the L1 data term is more robust to outliers than a simple averaging of the signed distance values. An extension to this method was proposed by Pock et al. [173] where the total variation regularization is replaced by total generalized variation, see Section 2.2.3.4. This regularization term better captures the piecewise affine solutions and reduces the stair-casing effect introduced by the Total Variation norm. Similar models have also been proposed that additionally include semantic segmentation cues to improve the fusion results [14, 98].

While those methods do not explicitly consider free space and visibility constraints, ray potentials explicitly allow the modeling these constraints in a Markov Random Field. Ulusoy et al. [249] considered a fully probabilistic model for image-based 3D reconstruction. Liu and Cooper [146] formulated the task as MAP inference in a Markov Random Field. In contrast to our method, these algorithms do not learn the geometric structure of objects and scene from data. Instead, they rely on simple hand-crafted priors such as spatial smoothness [146], or piecewise planarity [248]. Notably, Savinov et al. [201] combined ray potentials with 3D shape regularization terms that are learned from data. However, their regularization term is local and relies on a semantic segmentation of the input.

A common drawback of the methods that are either based on variational models, or ray consistency is, while they achieve superior results over vanilla TSF fusion [45], they are also much slower and have limited capabilities to handle missing or occluded data.

4.2.5 Shape Completion

A different approach to volumetric fusion from multiple depth map inputs is shape completion, where a incomplete 3D model is given. In this section, we focus especially on methods that complete the 3D shape from a single input image, or depth map.

Kim et al. [125] proposed Voxel-CRF, a Conditional Random Field that estimates the occupancy and semantic label of a 3D voxel grid given a single RGB-D input. While their goal is to improve 3D semantic segmentation, they include higher-order terms in the Random Field to encourage planar surfaces and contiguous objects. Similarly, Zheng et al. [272] proposed a voxel completion method by extruding visible points in detected Manhattan world directions. Firman et al. [71] proposed a learning-based framework for scene completion. A structured random forest is evaluated on voxel locations to estimate

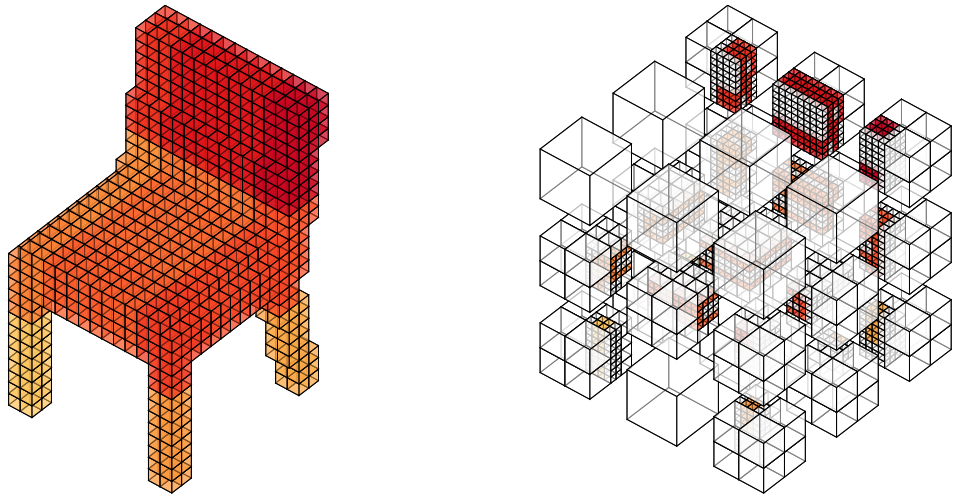
3D shape parts that are then fused. Instead of a random forest, Song et al. [222] used a 3D convolutional network for the completion of room scenes. In addition to the inference of the voxel occupancy in the 3D grid, their method also estimated the semantic class for each voxel. They were able to show learning both tasks together increased the performance on the individual tasks.

Instead of using a single depth map as input, there has recently been a great interest in reconstructing objects from a single color image. For example, Kar et al. [122] fitted deformable 3D models of objects to images. They first predict the viewpoint and category of the object with a convolutional network and then fit the deformable 3D model to detected key-point locations. A recently popular and related set approach is to extract depth information from a single image [60, 61, 77, 90], or producing an anaglyph 2.5D output from a single image [260] using convolutional networks. Similarly, there has recently been proposed a deep network approach to directly synthesizing novel views from a set images [72] This technique was utilized by Tatarchenko et al. [233] for volumetric fusion. They trained a deep convolutional network to output depth maps and color images for a specific rotation given as input a single color image. Multiple depth maps of different rotations are then fused to a volumetric representation using TGV-Fusion [173]. Choy et al. [41] introduced a related approach, but instead of single depth maps the output is a 3D 32^3 occupancy voxel grid. In addition, they proposed a recurrent convolutional layer that fuses the encoder representation of multiple input images, which slightly improved the reconstruction results. Wu et al. [258] presented 3D Interpreter Neural Networks that combine the ideas of [122] and [41]. The authors utilized a convolutional network to estimate the 2D key-points of object categories. Those key-points are then fed into an additional network that interprets them and outputs a 3D structure, i.e. a 3D line diagram.

A drawback of those direct approaches is that they require a huge amount of 3D ground-truth models for training. Yan et al. [262] tried to circumvent this by formulating the loss not in the 3D voxel space, but on 2D projections of the 3D object. This is possible because the projection is fully differentiable. This loss was further generalized by Tulsiani et al. [247]. Instead of evaluating the loss in the 2D projections, they formulated a differentiable ray consistency loss. The network can then be trained by providing multiple depth maps, or only foreground masks as supervision.

A common problem of the volumetric representation as the output of convolutional networks is again the resolution. Hence, Fan et al. [63] introduced a Point Set Generation Network. As the name suggests, the output is a 3D point cloud instead of a voxel-based representation. However, the number of points still needs to be fixed. Dai et al. [47] proposed a different approach. Their network still outputs a coarse 32^3 voxel grid for the task of shape completion, but then utilizes shape synthesis relying on an external database to produce a high-resolution voxel grid.

Very related to our octree based approach are the concurrent works by Häne et al. [97] and Tatarchenko et al. [234]. They also successively output reconstructions on increasing



(a) Voxel Representation **(b)** Hybrid Grid-Octree Representation

Figure 4.4: Hybrid grid-octree data structure. This example illustrates a hybrid grid-octree consisting of 27 shallow octrees. Using 3 shallow octrees in each dimension with a maximum depth of 3 leads to a total resolution of 24^3.

resolutions in an octree. However, they only use the octree scheme on the output, whereas we are able to use it for the input, too.

4.3 Deep Learning for High-Resolution 3D

In this section, we present our method named OctNet that enables training of convolutional networks on high-resolution voxelized 3D data. We enable this by using a space partitioning data-structure within the network, see Section 4.3.1. Further, we formulate common network operations, e.g. convolution, pooling, and unpooling, on this non-uniform data structure, but keeping them close to their counterparts on a regular voxel grid. This drastically decreases the memory footprint of the models, but also enables us to reuse popular network architectures [102, 228, 220] for problems in 3D. However, this method is not directly applicable if the space partitioning is not known in advance, e.g. for 3D completion, or reconstruction. Therefore, we present Section 4.3.2 an extension that handles this issue by learning the space partitioning in addition to the occupancy of the 3D volume.

4.3.1 OctNet

To decrease the memory footprint of convolutional networks operating on sparse 3D data, we propose an adaptive space partitioning scheme which focuses computations on the relevant regions. As mathematical operations of deep networks, especially convolutional networks, are best understood on regular grids, we restrict our attention to data struc-

Shallow tree depth	1	2	3	4	5
Bit string length	1	9	73	585	4,681
Volume in voxels	1	64	512	4,096	32,768

Table 4.1: Trade-off between shallow octree depth and memory consumption.

tures on 3D voxel grids. One of the most popular space partitioning structures on voxel grids are octrees [157] which have been widely adopted due to their flexible and hierarchical structure. An non-exhaustive list of application include depth fusion [124], image rendering [135] and 3D reconstruction [249]. In this thesis, we propose 3D convolutional networks on octrees to learn representations from high-resolution 3D data.

An octree partitions the 3D space by recursively subdividing it into octree cells, also called octants. By subdividing only the cells which contain relevant information, e.g. cells crossing a surface boundary or cells containing one or more 3D points, storage can be allocated adaptively. Densely populated regions are modeled with high accuracy, i.e. using small cells, while empty regions are summarized by large cells in the octree.

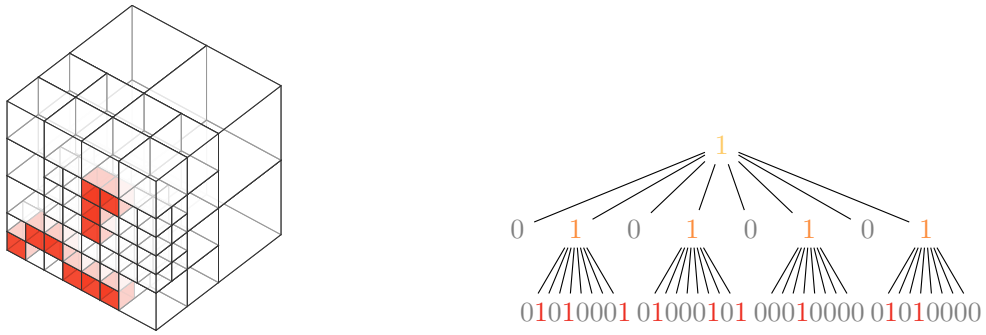
Unfortunately, vanilla octree implementations [157] have several drawbacks that hamper its application in deep networks. While octrees reduce the memory footprint of the 3D representation, most versions do not allow an efficient access to the underlying data. In particular, octrees are typically implemented using pointers, where each node contains a pointer to its children. Accessing an arbitrary element (or the neighbor of an element) in the octree requires a traversal starting from the root until the desired cell is reached. Thus, the number of memory accesses is equal to the depth of the tree. This becomes increasingly costly for deep, i.e. high-resolution, octrees. Convolutional network operations such as convolution or pooling require frequent access to neighboring elements. It is thus critical to utilize an octree design that allows for fast data access.

We tackle these challenges by leveraging a hybrid grid-octree data structure which we describe in Section 4.3.1.1. In Section 4.3.1.2, we show how 3D convolution and pooling operations can be implemented efficiently on this data structure.

4.3.1.1 Hybrid Grid-Octree Data Structure

The above mentioned problems with the vanilla octree data structure increase with the octree depth. Instead of representing the entire high-resolution 3D input with a single unbalanced octree, we leverage a hybrid grid-octree structure similar to the one proposed by Miller et al. [158]. The key idea is to restrict the maximal depth of an octree to a small number, e.g. three, and place several such shallow octrees along a regular grid as illustrated in Figure 4.4. While this data structure may not be as memory efficient as the standard octree, significant compression ratios can still be achieved. For instance, a single shallow octree that does not contain input data stores only a single vector, instead of $8^3 = 512$ vectors for all voxels at the finest resolution at depth 3.

An additional benefit of a collection of shallow octrees is that their structure can be



(a) Single Shallow Octree **(b)** Bit Representation

Figure 4.5: Shallow octree bit representation. A single shallow octree can be efficiently encoded using a bit-string. Here, the bit-string 1 01010101 00000000 01010001 00000000 01000101 00000000 00010000 00000000 01010000 defines the octree in (a). The corresponding tree structure is shown in (b). The color in the bit-string corresponds to the split level.

encoded very efficiently using a bit string representation which further lowers access time and allows for efficient GPGPU implementations [158]. Given a shallow octree of depth 3, we use 73 bit to represent the complete tree. The first bit with index 0 indicates, if the root node is split, or not. Further, bits 1 to 8 indicate if one of the child nodes is subdivided and bits 9 to 72 denote splits of the grandchildren, see Figure 4.5. A tree depth of 3 gives a good trade-off between memory consumption and computational efficiency. Further, increasing the octree depth results in an exponential growth in the required bits to store the tree structure and further increases the cell traversal time as depicted in Table 4.1.

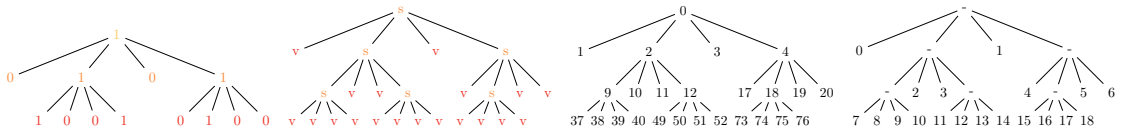
Using this bit-representation, a single voxel in the shallow octree is fully characterized by its bit index. This index determines the depth of the voxel in the octree and therefore also the voxel size. Instead of using pointers to the parent and child nodes, simple arithmetic can be used to retrieve the corresponding indices of a voxel with bit index b

$$\mathrm{pa}(b) = \left\lfloor \frac{b-1}{8} \right\rfloor , \tag{4.1}$$

$$\mathrm{ch}(b) = 8 \cdot b + 1 . \tag{4.2}$$

In contrast to [158], we associate a data container for storing features vectors with all leaf nodes of each shallow tree. We allocate the data of a shallow octree in a contiguous, breath-first data array. The offset associated with a particular voxel in this array can be computed as follows

$$\mathrm{data_idx}(b) = \underbrace{8 \sum_{b'=0}^{\mathrm{pa}(b)-1} \mathrm{bit}(b') + 1}_{\#\text{nodes above b}} - \underbrace{\sum_{b'=0}^{b-1} \mathrm{bit}(b')}_{\#\text{split nodes pre b}} + \underbrace{\mathrm{mod}(b-1, 8)}_{\text{offset}} . \tag{4.3}$$



(a) Bit-String **(b)** Split & Leaf Nodes **(c)** Bit Index **(d)** Data Index

Figure 4.6: Data index computation example.

Here, mod denotes the modulo operator and bit returns the tree bit-string value at b. The first part of the equation above counts the number of split and leaf nodes up to the voxel with bit index i. The second term subtracts the number of split nodes before the particular voxel as data is only associated with leaf nodes. Finally, we need to get the offset within the voxel's neighborhood. This is done by the last term of the equation. Both sum operations can be efficiently implemented using bit counting intrinsics, i.e. `popcnt`. The data arrays of all shallow octrees are concatenated into a single contiguous data array during training and testing to reduce I/O latency.

Let us illustrate the data index computation with a simple example: For ease of visualization, we will consider a quadtree. Hence, each voxel can be split into 4 instead of 8 children. The equation for the offset changes to

$$\text{data_idx}_4(b) = \underbrace{4 \sum_{b'=0}^{\text{pa}_4(b)-1} \text{bit}(b') + 1}_{\text{\#nodes above b}} - \underbrace{\sum_{b'=0}^{b-1} \text{bit}(b')}_{\text{\#split nodes pre b}} + \underbrace{\text{mod}(b-1, 4)}_{\text{offset}}, \tag{4.4}$$

with

$$\text{pa}_4(b) = \left\lfloor \frac{b-1}{4} \right\rfloor . \tag{4.5}$$

Now consider the following bit string for instance: 1 0101 0000 1001 0000 0100. According to our definition, this bit string corresponds to the tree structure visualized in Figure 4.6a and 4.6b, where s indicates a split node and v a leaf node with associated data. In Figure 4.6c we show the bit indices for all nodes. Note that the leaf nodes at depth 3 do not need to be stored in the bit string as this information is implicit. Finally, the data index for all leaf nodes is visualized in Figure 4.6d. Now we can verify equation (4.4) using a simple example. Assume the bit index 51: The parent bit index is given by equation (4.5) as 12. To compute the data index we first count the number of nodes before 49 as it is the first node within its siblings (first term of the equation), which is 17. Next, we count the number of split nodes up to 49 (second term of the equation), which is 6. Finally, we look up the position 51 within its siblings (last term of the equation), which is 2. Combining those three terms yields the data index $17 - 6 + 2 = 13$.

4.3.1.2 Network Operations

Given the hybrid grid-octree data structure introduced in the previous section, we now discuss the efficient implementation of network operations on this data structure. We will focus on the most common operations in convolutional networks [6, 34, 35, 81, 102, 132, 148, 220, 228]: convolution, pooling and unpooling. Note that point-wise operations, like activation functions, do not differ in their implementation as they are independent of the data structure.

Let us first introduce the notation which will be used throughout this section. $T_{i,j,k}$ denotes the value of a 3D tensor T at location (i, j, k). Now assume a hybrid grid-octree structure with $D \times H \times W$ unbalanced shallow octrees of maximum depth 3. Let $O[i, j, k]$ denote the value of the smallest cell in this structure which comprises the voxel (i, j, k). Further, let $\Omega[i, j, k]$ be the smallest octant that contains the voxel at (i, j, k). Hence, $\Omega[i, j, k]$ is the set of voxel indices whose data is pooled to a single value. Note that in contrast to the tensor notation, $O[i_1, j_1, k_1]$ and $O[i_2, j_2, k_2]$ with $i_1 \neq i_2 \vee j_1 \neq j_2 \vee k_1 \neq k_2$ refer to the same value/octant in the hybrid grid-octree, if $(i_1, j_1, k_1) \in \Omega[i, j, k]$ and $(i_2, j_2, k_2) \in \Omega[i, j, k]$. We obtain the index of the shallow octree in the grid via $(\lfloor \frac{i}{8} \rfloor, \lfloor \frac{j}{8} \rfloor, \lfloor \frac{k}{8} \rfloor)$ and the local index of the voxel at the finest resolution in that octree by $(\mathrm{mod}(i, 8), \mathrm{mod}(j, 8), \mathrm{mod}(k, 8))$.

Given this notation, the mapping from a grid-octree O to a tensor T with compatible dimensions is given by

$$\mathrm{oc2ten} : T_{i,j,k} = O[i, j, k] \,. \tag{4.6}$$

Similarly, the reverse mapping is given by

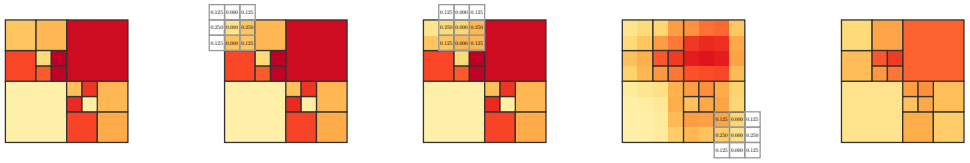
$$\mathrm{ten2oc} : O[i, j, k] = \underset{(i',j',k') \in \Omega[i,j,k]}{\mathrm{pool_voxels}} \left(T_{i',j',k'}\right), \tag{4.7}$$

where pool_voxels $(\cdot)$ is a pooling function (e.g., average- or max-pooling) which pools all voxels in T over the smallest grid-octree cell comprising location (i, j, k). This pooling is necessary as a single voxel in O can cover up to $8^3 = 512$ elements of T, depending on its size $|\Omega[i, j, k]|$.

Note that with the two functions defined above, we could wrap any network operation f defined on 3D tensors via

$$g(O) = \mathrm{ten2oc}(f(\mathrm{oc2ten}(O))) \,. \tag{4.8}$$

However, this would require a costly conversion from the memory efficient grid-octrees to a regular 3D tensor and back. Besides, storing a dense tensor in memory limits the maximal resolution. We therefore define our network operations directly on the hybrid grid-octree data structure.



(a) Input **(b)** Convolution **(c)** Output

Figure 4.7: Octree convolution example. The different colors in the input depicted in (a) indicate the different values per octree cell. The convolution kernel is applied on each voxel location resulting in different responses within octree cells, see (b). Therefore, the information is pooled which yields the octree convolution result as visualized in (c).

Convolution The convolution operation is the most important, but also the most computationally expensive operation in deep convolutional networks. For a single feature map, convolving a 3D tensor T with a 3D convolution kernel $W \in \mathbb{R}^{L \times M \times N}$ can be written as

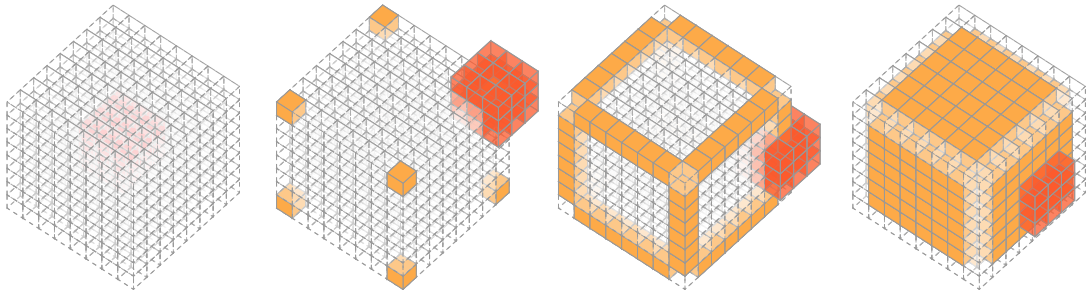
$$T^{\mathsf{out}}_{i,j,k} = \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} W_{l,m,n} \cdot T^{\mathsf{in}}_{i',j',k'} \,, \tag{4.9}$$

with $i' = i - l + \lfloor L/2 \rfloor$, $j' = j - m + \lfloor M/2 \rfloor$, $k' = k - n + \lfloor N/2 \rfloor$. Similarly, the convolutions on the grid-octree data structure are defined as

$$\begin{aligned} O^{\mathsf{out}}[i,j,k] &= \underset{(\hat{i},\hat{j},\hat{k}) \in \Omega[i,j,k]}{\text{pool_voxels}} \left(T_{\hat{i},\hat{j},\hat{k}} \right) \\ T_{\hat{i},\hat{j},\hat{k}} &= \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} W_{l,m,n} \cdot O^{\mathsf{in}}[i',j',k'] \end{aligned} \tag{4.10}$$

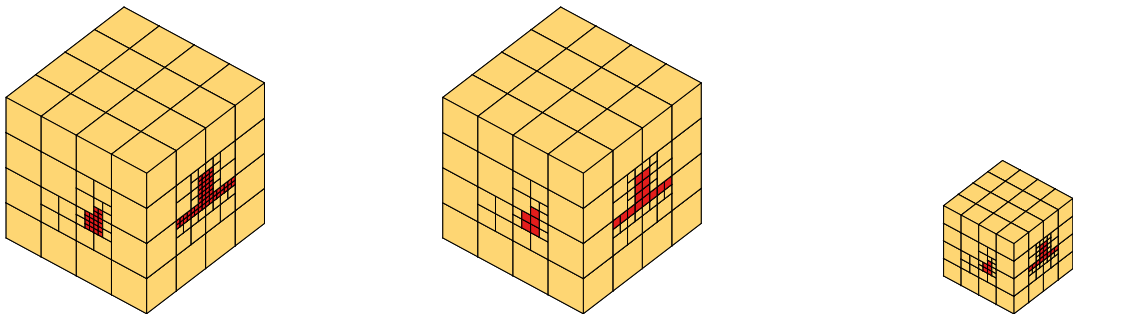
with i', j', k' defined as above. A visualization of this octree convolution is visualized in Figure 4.7. While this calculation yields the same result as the tensor convolution in Equation (4.9) with the oc2ten, ten2oc wrapper, we are now able to define a computationally more efficient convolution operator.

A naïve implementation would apply the convolution kernel at every location (i, j, k) comprised by the cell $\Omega[i, j, k]$. Therefore, for an octree cell of size 8^3 and a convolution kernel of 3^3 this would require $8^3 \cdot 3^3 = 13,824$ multiplications. However, we can exploit that $O[i, j, k]$ is constant within a small margin of the cell due to its constant support $\Omega[i, j, k]$: For a small convolution kernel, e.g. 3^3, the filter response within the octree cell is constant. Hence, we can implement this calculation much more efficiently as depicted in Figure 4.8. We observe that the value inside the cell of size 8^3 is constant. Thus, we only need to evaluate the convolution once inside this cell and multiply the result by the size of the cell 8^3, see Figure 4.8a. Additionally, we need to evaluate a truncated version of the kernel on the corners, edges and faces of the octant, see Figures 4.8b-4.8d. This implementation is more efficient, as we need only 27 multiplications for the constant part,



(a) Constant **(b)** Corners **(c)** Edges **(d)** Faces

Figure 4.8: Efficient octree convolution implementation. The grid depicted in the four graphics visualizes the voxel locations of a single large octree cell. The convolution kernel is depicted by the red cube, and the active voxel locations for the convolutions by orange cubes.



(a) Input **(b)** Pooling finest cells **(c)** Combining octrees

Figure 4.9: Octree pooling example. The red color indicates octree cells on the finest resolution.

$8 \cdot 19$ multiplications for the corners, $12 \cdot 6 \cdot 15$ multiplications for the edges, and $6 \cdot 6^2 \cdot 9$ multiplications for the faces of the octree cell. In total, this yields $3,203$ multiplications, or 23.17% of the multiplications required by the naïve implementation. At the same time, it enables a better caching mechanism, especially if the neighboring octree cells also comprise many voxels.

Pooling Another important operation in deep convolutional networks is pooling. Pooling reduces the spatial resolution of the input tensor and aggregates higher-level information for further processing, thereby increasing the receptive field and capturing context. For instance, strided 2^3 max-pooling divides the input tensor T^{in} into 2^3 non-overlapping regions and computes the maximum value within each region. Formally, we have

$$T^{\mathsf{out}}_{i,j,k} = \max_{l,m,n \in [0,1]} \left(T^{\mathsf{in}}_{2i+l,2j+m,2k+n} \right) , \tag{4.11}$$

where $T^{\mathsf{in}} \in \mathbb{R}^{2D \times 2H \times 2W}$ and $T^{\mathsf{out}} \in \mathbb{R}^{D \times H \times W}$.

To implement pooling on the grid-octree data structure we reduce the number of shallow octrees. For an input grid-octree O^{in} with $2D \times 2H \times 2W$ shallow octrees, the

output O^{out} contains $D \times H \times W$ shallow octrees. Each octree cell of O^{in} is halved in size and copied one level deeper in the shallow octree. Octree cells at depth 3 in O^{in} are pooled. This can be formalized as

$$O^{\text{out}}[i,j,k] = \begin{cases} O^{\text{in}}[2i,2j,2k] & \text{if } \text{ocd}(2i,2j,2k) < 3 \\ P & \text{else} \end{cases}$$
$$P = \max_{l,m,n \in [0,1]} (O^{\text{in}}[2i+l, 2j+m, 2k+n]) \,, \tag{4.12}$$

where $\text{ocd}(\cdot)$ computes the depth of the indexed voxel in the shallow octree. A visual example is depicted in Figure 4.9.

Unpooling For several tasks such as semantic segmentation, the desired network output is of the same size as the network input. While pooling is crucial to increase the receptive field size of the network and capture context, it loses spatial resolution. To increase the resolution of the network, U-shaped network architectures have become popular [6, 42] which encode information using pooling operations and increase the resolution in a decoder part using unpooling or deconvolution layers, possibly in combination with skip-connections to increase precision. The simplest unpooling strategy uses nearest neighbour interpolation and can be formalized on dense input $T^{\text{in}} \in \mathbb{R}^{D \times H \times W}$ and output $T^{\text{out}} \in \mathbb{R}^{2D \times 2H \times 2W}$ tensors as follows

$$T^{\text{out}}_{i,j,k} = T^{\text{in}}_{\lfloor i/2 \rfloor, \lfloor j/2 \rfloor, \lfloor k/2 \rfloor} \,, \tag{4.13}$$

Again, we can define the analogous operation on the hybrid grid-octree data structure by

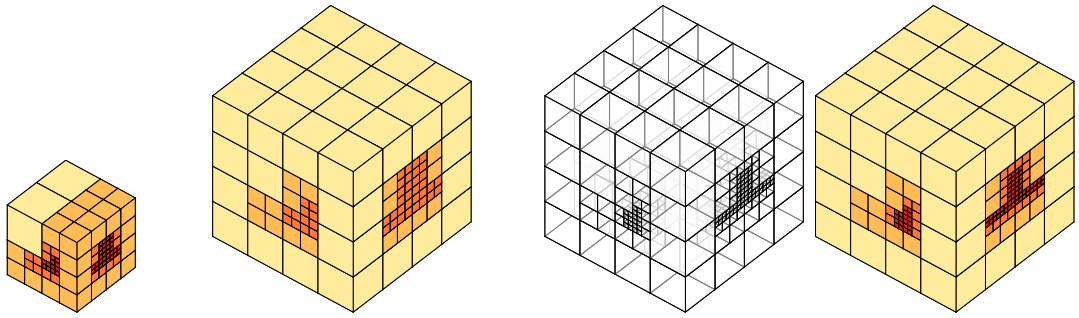
$$O^{\text{out}}[i,j,k] = O^{\text{in}}[\lfloor i/2 \rfloor, \lfloor j/2 \rfloor, \lfloor k/2 \rfloor] \,, \tag{4.14}$$

This operation also changes the data structure: The number of shallow octrees increases by a factor of 8, as each octree cell at depth 0 spawns a new shallow octree. All other octree cells double their size. Thus, after this operation, the tree depth is decreased. See Figures 4.10a-4.10b for a visual example of this operation.

To capture fine details, cells can be split again at the finest resolution, i.e. according to the original octree of the corresponding pooling layer, or given any other guidance octree structure. This allows using skip-connections in U-shaped networks, or dynamic OctNets as presented in the next section. See Figures 4.10c-4.10d for a visual example of this operation.

4.3.2 Dynamic OctNet

The main drawback of OctNets as presented in the previous section is that the octree structure of the input and output, i.e. the partitioning of the 3D space, has to be known a



(a) Input **(b)** Output **(c)** Guidance **(d)** Output Guided

Figure 4.10: Octree unpooling example. The different colors indicate octree cells on different depth.

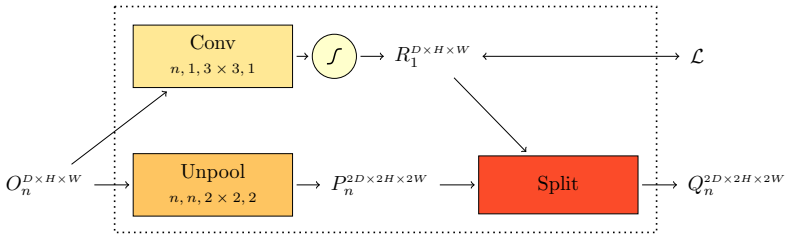


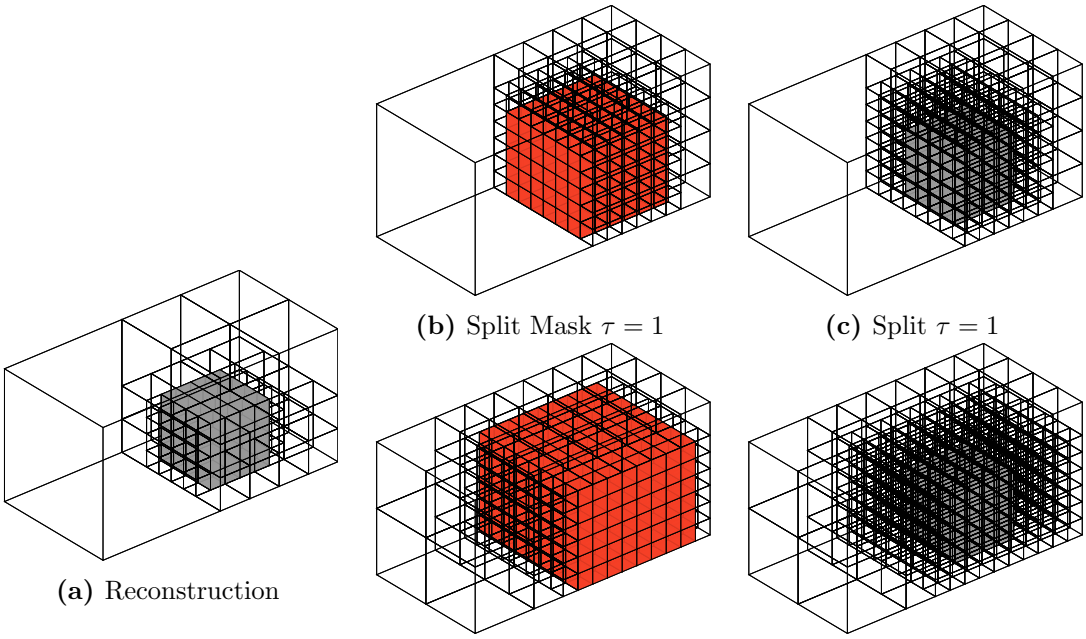
Figure 4.11: OctNet structure module. The structure manipulation module doubles the spatial resolution of the feature maps. A loss Δ measures the quality of the reconstruction at the respective resolution.

priori. This is a reasonable assumption for tasks like shape classification, or 3D point cloud labeling where the input and the output octree structures are the same, or the output is a 1D vector, respectively. However, for tasks where the output geometry is different from the input geometry, e.g. in volumetric fusion or shape completion, the grid-octree data structure has to be adapted during inference.

To cope with this short-coming we introduce a structure module which determines for each octree cell if it shall be split, or not. Our structure module is illustrated in Figure 4.11. The main idea is to derive a split mask from an intermediate reconstruction. This split mask is then utilized in the unpooling operation as guidance which of the octree cells should be further subdivided.

More formally, let us consider an input grid-octree structure O with n feature channels and $D \times H \times W$ shallow octrees as illustrated in Figure 4.11. After the unpooling operation we obtain a structure P that consists of $2D \times 2H \times 2W$ shallow octrees where each octree cell comprises eight-times the number of voxels, i.e., $|\Omega_P[2i, 2j, 2k]| = 8|\Omega_O[i, j, k]|$.

To determine the new octree structure, we additionally predict a reconstruction R at the resolution of O using a single convolution optionally followed by a sigmoid non-linearity depending on the desired output format, e.g. occupancy grid, or truncated signed distance function (TSDF). The reconstruction loss $\mathcal{L}$ ensures that the predicted reconstruction is close to the ground truth reconstruction.



(a) Reconstruction **(b)** Split Mask $\tau = 1$ **(c)** Split $\tau = 1$ **(d)** Split Mask $\tau = 2$ **(e)** Split $\tau = 2$

Figure 4.12: Octree splitting. Given a reconstruction as depicted in (a) a split mask is computed, (b) and (d). The split mask indicates the transition from occupied to free voxels with a given distance τ. The octree cells that are marked by the split mask are then represented on the finest octree resolution,(c) and (e).

We define the split mask implicitly by the surface of the reconstruction R. The surface is defined by the gradients of R when predicting occupancies or by the zero level set of R in the case of TSDF regression. Given the surface, we split all voxels within distance τ from the surface. For TSDF regression, τ could equal the truncation threshold. For occupancy classification, τ is a flexible parameter which can be tuned to trade reconstruction accuracy vs. memory usage. See Figure 4.12 for a visual illustration. The output of this split operation finally yields the high-resolution structure Q which serves as input to a next level in the network architecture.

4.4 Evaluation

In this section, we evaluate the proposed OctNet on a number of different tasks for high-resolution 3D data. First, we evaluate the core OctNet, where the input and output structure is known in Sections 4.4.1-4.4.3. There, we will focus on the memory and runtime requirements of the proposed solution and on the representational abilities compared to dense 3D convolutional networks for the 3D shape classification task. Additionally, we will also show how increasing the input resolution can benefit the orientation estimation capabilities of deep networks, as well as the semantic segmentation of 3D point clouds. In

the second part of the evaluation, we show how the learning of the octree subdivision can be used for depth completion and fusion, see Section 4.4.4. The presented learning based approach outperforms well-established baseline methods especially if the number of input views is limited, or are degenerated by noise.

4.4.1 3D Shape Classification

A popular task on 3D data that is very related to 2D image data is shape classification. The input is a 3D triangle mesh and the goal is to assign it one out of N possible categories, or classes. We will use this evaluation not only to assess the classification accuracy with respect to the input resolution but also to quantify the memory and runtime benefits of the proposed OctNet.

Experimental Set-Up In this evaluation we utilize the widely used ModelNet10 and ModelNet40 [259] datasets. ModelNet10 consists of $3,991$ training and 908 test shapes divided into 10 categories. The larger ModelNet40 dataset contains $9,843$ training and $2,468$ test shapes spread over 40 categories. We further randomly split the training set by using 25 samples per category in an additional validation set. To input the 3D triangle meshes to the network we first compute 3D occupancy grids, where a voxel is set to 1, i.e. occupied, if it is intersected with one or more triangles of the mesh. Otherwise, the voxel is set to 0, i.e. free space. As the shape meshes do not have a normed size, we scale all meshes prior to voxelization to fit into the grid. To avoid border effects, we use $\frac{1}{16}$ of the voxels of each dimension for padding, e.g. if we use a voxel grid of 32^3, then the shape will be scaled to fit into 30^3 voxels. For our grid-octree representation, all occupied voxels are represented on the finest resolution and the rest is summarized in larger octree cells.

For this evaluation, we utilize a network architecture similar to ResNet [102]. To enable a fair evaluation across the different input resolutions, we keep the number of network parameters constant but change the number of pooling operations towards the end of the networks. Therefore, emphasizing more high-level features. See Figure 4.13 for a detailed depiction of the architecture. We train each network for 20 epochs with a learning rate of 10^{-3} and Adam [128] as optimizer. The training objective is the standard cross-entropy loss with an additional weight decay of 10^{-4}. The batch size is set to 32. For the evaluation, we use the network per resolution that yields the lowest error on the validation set.

Results Before we discuss the results in terms of classification accuracy, we first want to show the memory and runtime of our method in this experiment. In Figure 4.14 we visualize the memory consumption of single voxelized shapes for different resolutions, once in the dense grid representation and once in our grid-octree representation. The memory consumption of the voxel grid is of course only dependent on the input resolution, whereas for our grid-octree structure the memory consumption is also dependent on the voxel occu-

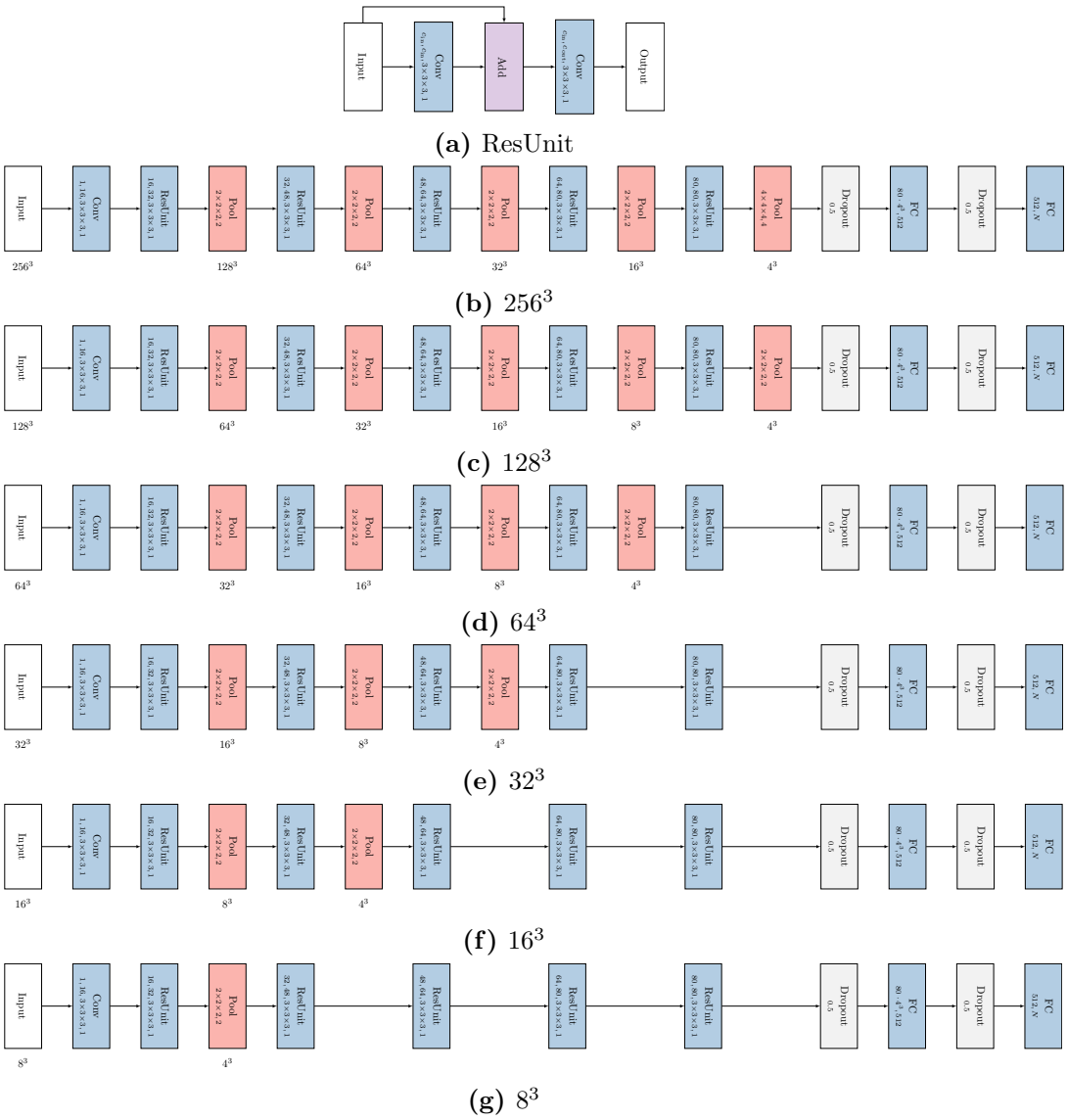
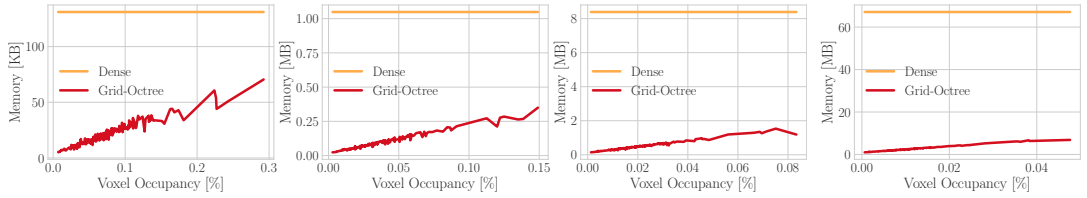
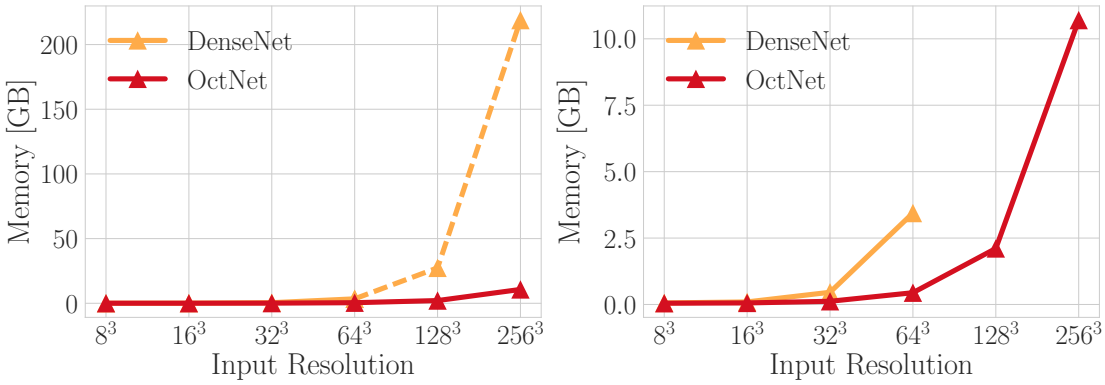


Figure 4.13: OctNet ResNet for different input resolutions. The main building block of the architecture is the residual unit as depicted in (a). To enable a fair evaluation of the network performance with varying input resolutions, we keep the number of network parameters fixed. Therefore, we first decrease the last pooling stride from $4 \times 4 \times 4$ for (b) 256^3 to $2 \times 2 \times 2$ for (c) 128^3. Then, we remove one pooling layer for each decrease of the input resolution by a factor of 2^3 (d)-(g).



(a) 32^3 **(b)** 64^3 **(c)** 128^3 **(d)** 256^3

Figure 4.14: Memory consumption of standard voxel grids compared to hybrid grid-octrees. The memory consumption of the voxel grid is constant for a given resolution. In contrast, the memory consumption of the hybrid grid-octree structure scales almost linearly with the voxel occupancy and is always way below the voxel grid.

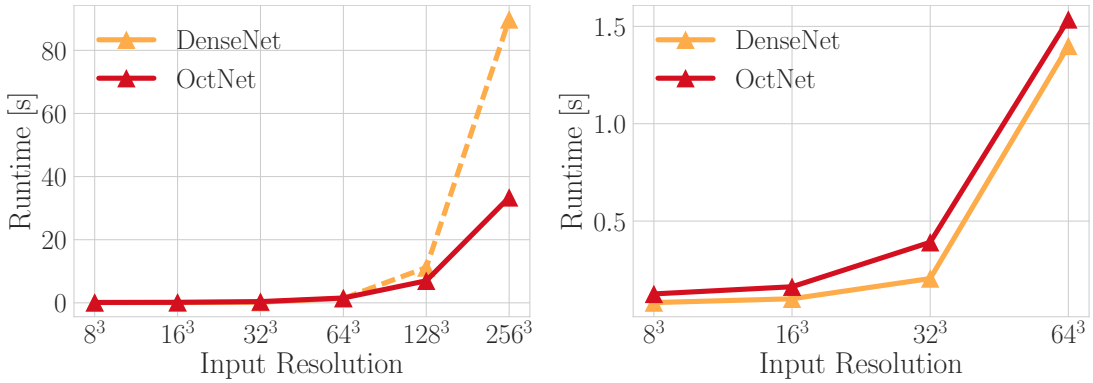


(a) Overview **(b)** Zoomed

Figure 4.15: Memory consumption of a 3D convolutional network on voxel grids and hybrid grid-octrees. The number of network parameters is fixed. For lower resolutions, the number of pooling layers is reduced.

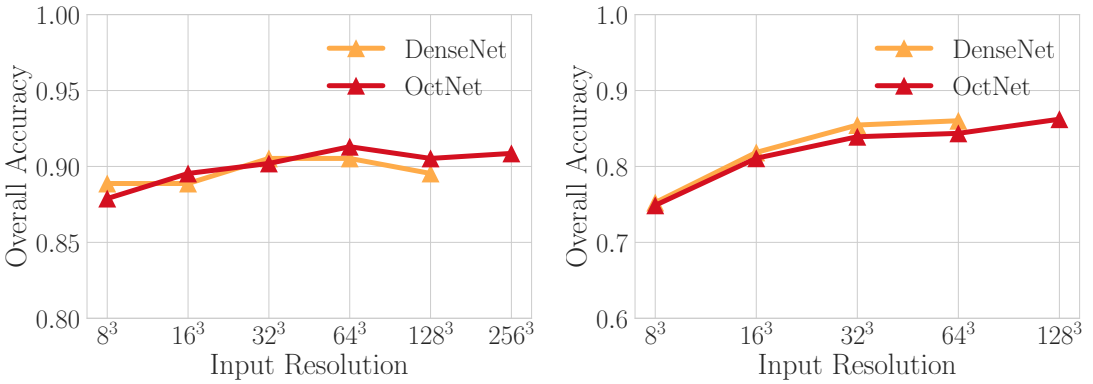
pancy, i.e. how many percents of the voxels intersect with a mesh triangle. It also depends on how the occupied voxels are distributed. Therefore, it is not monotonic increasing in the plots. What is important to observe is that the grid-octree data structure always needs significantly less memory. This is especially pronounced for higher resolutions, i.e. for 256^3 the voxel grid needs 67.11 MB, whereas our grid-octree needs always less than 10 MB.

The memory consumption is only for a single sample and this gets even more pronounced if a whole network is trained. In that case, the memory consumption gets multiplied with the batch size which can be decreased to 1 in the worst case, but also with the number of feature maps, which is typically in the hundreds. Therefore, we compare the memory consumption of the deep network with respect to the input resolution in Figure 4.15. The memory consumption is for a forward pass with the previously stated settings, once for the network on the voxel grid called *DenseNet*, and once for the network on the hybrid grid-octree called *OctNet*. We observe that the OctNet is always more memory efficient than the equivalent DenseNet variant. For small input resolutions, i.e. $\leq 32^3$,



(a) Overview **(b)** Zoomed

Figure 4.16: Runtime of a 3D convolutional network on voxel grids and hybrid grid-octrees. The number of network parameters is fixed. For lower resolutions, the number of pooling layers is reduced.

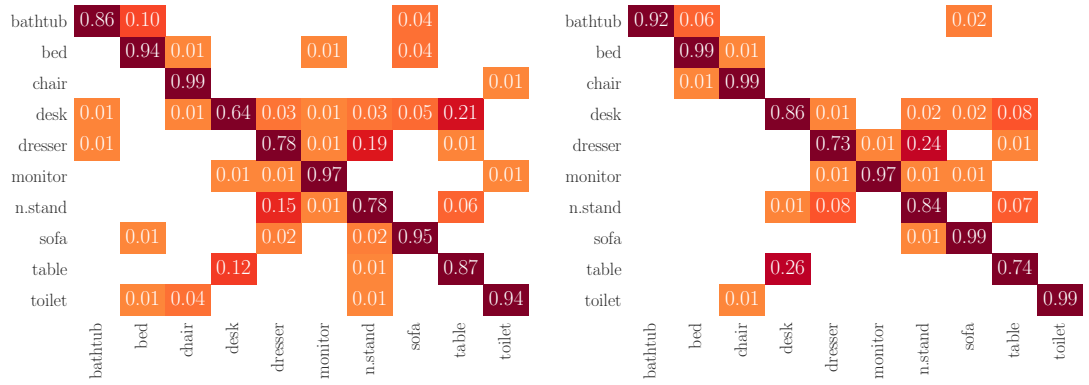


(a) ModelNet10 **(b)** ModelNet40

Figure 4.17: Classification accuracy for ModelNet10 and ModelNet40.

this improvement might be less spectacular, but for higher resolutions, the difference is really enormous. For example, we can train the OctNet on 128^3 with less memory than the equivalent DenseNet on 64^3. Further, the DenseNet with the given setting does not fit on a single GPGPU with 12GB of memory anymore, hence the dotted line in the plot as the results are extrapolated from a smaller batch size.

We get a similar picture, if we have a detailed look at the runtime, see Figure 4.16. For small input resolutions ($\leq 64^3$) the network on the voxel grid (DenseNet) is slightly faster due to the faster memory access pattern, i.e. coalesced memory access. However, for larger input resolutions ($\geq 128^3$) the network on the hybrid grid-octree data structure (OctNet) gets significantly faster, due to the efficient convolution implementation discussed in the method section. Hence, OctNet does not only enable training networks on higher input resolutions on the same hardware than a network on a regular voxel grid but is also faster to train.



(a) 8^3 **(b)** 128^3

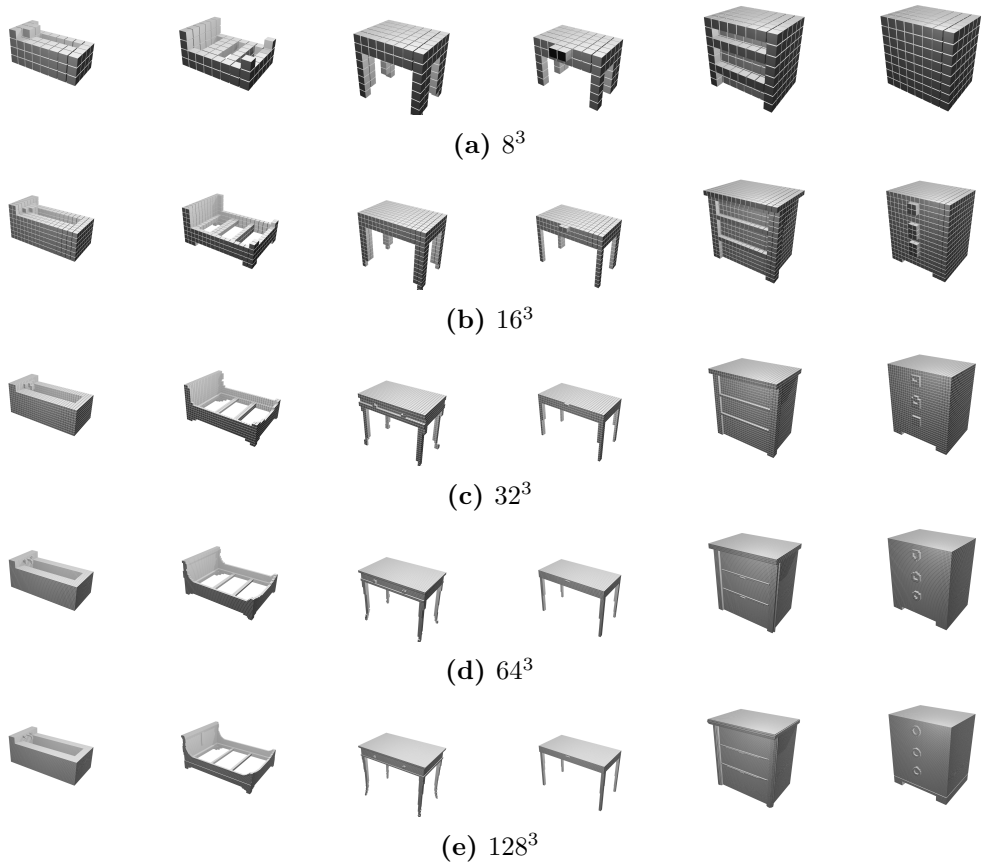
Figure 4.18: Confusion matrices of the ModelNet10 classification results.

What we did not look at yet is the representational capacity of the OctNet compared to the DenseNet. Therefore, we present the classification accuracy of both networks with respect to the different input resolutions for ModelNet10 and ModelNet40 in Figure 4.17. We first note that despite its pooled representation, OctNet performs on par with its DenseNet counterpart. This confirms our initial intuition that sparse data allows for allocating resources adaptively without loss of performance. Note, that the purpose of this experiment was to evaluate the memory, runtime and representational capacity of OctNet compared to its counterpart DenseNet. Therefore, we did not utilize any data augmentation for ModelNet10, and used 10 uniform rotated versions around the z-axis of the ModelNet40 shapes, because those samples are not aligned. We expect to get an additional performance gain by using more extensive data augmentation, additional auxiliary losses [4, 176] and training more networks in an ensemble [24].

Although we observe a slight performance improvement with respect to the input resolution, i.e. up to 64^3 for ModelNet10 and up to 128^3 for ModelNet40, the increase is only subtle beyond the resolution of 32^3. This indicates that higher resolutions might not be necessary for this dataset to distinguish the categories. The claim can also be supported by a look at the confusion matrices of the ModelNet10 results in Figure 4.18. We see that for small resolutions obvious shape categories get confused like bathtubs with beds, that are then better distinguished at higher resolutions. However, there are certain category pairs that are still very hard at high resolutions where all details are visible, e.g. desk vs. table and night stand vs. dresser. We visualize some of the ModelNet ambiguities in Figure 4.19. Note that those ambiguities are even more frequent in the larger ModelNet40 dataset.

4.4.2 3D Orientation Estimation

In this section we investigate the importance of the input resolution on 3D orientation estimation tasks. First, we have a look at the ModelNet dataset [259] in Section 4.4.2.1



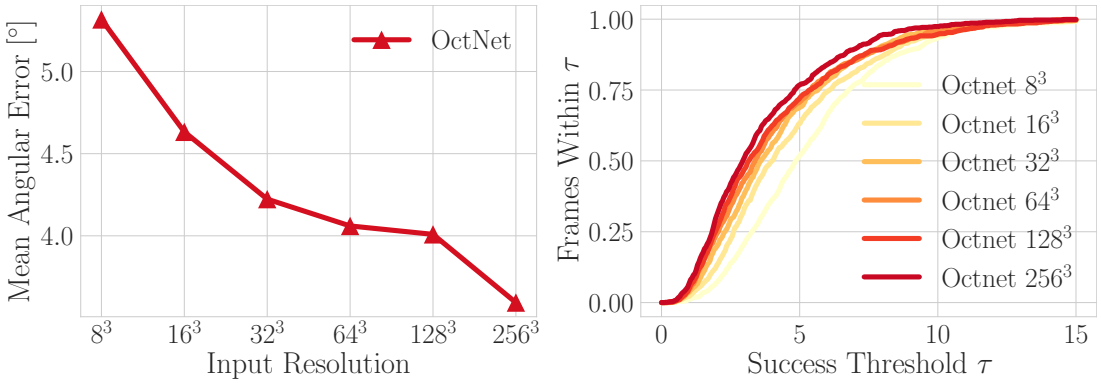
(a) 8^3 **(b)** 16^3 **(c)** 32^3 **(d)** 64^3 **(e)** 128^3

Figure 4.19: ModelNet ambiguities. We show six different ModelNet categories for different input resolutions. Even after drastically increasing the input resolution from 8^3 in (a) to 128^3 in (e) shapes from the categories desk vs. table and night stand vs. dresser are hard to distinguish.

again, where the goal is to estimate the rotation of unknown object instances. In a second evaluation presented in Section 4.4.2.2 we evaluate OctNet on the popular head pose estimation dataset by Fanelli et al. [64, 65].

4.4.2.1 ModelNet

Most existing approaches to 3D pose estimation [18, 19, 235, 257] assume that the true 3D shape of the object instance is known. To assess the generalization ability of 3D convolutional networks, we consider a slightly different set-up where only the object category is known. After training the model on a hold-out set of 3D shapes from a single category, we test the ability of the model to predict the 3D orientation of unseen 3D shapes from the same category. More concretely, given an instance of a predefined object category with an unknown pose, the goal is to estimate the rotation with respect to the canonical pose.



(a) Mean Angular Error **(b)** Success Threshold

Figure 4.20: Quantitative orientation estimation results on ModelNet10. (a) shows the mean angular error $\mu(\phi)$ of all test samples with respect to the input resolution. (b) visualizes a cumulative error plot, where we show the percentage of correctly estimated orientations for a certain success threshold τ.

Experimental Set-Up As in the previous experiment, we utilize the ModelNet10 dataset [259] where all the provided models are in a canonical pose. We selected as object category the class *chair* as it provides many training and test samples and has high intra-class variance, but the possible rotations are not ambiguous as it might be the case for instance for tables. For each 3D model we created 10 train and test samples by randomly rotating them between $\pm 15°$ around each axis. Those random rotations are then converted to unit quaternions which are the regression targets for the 3D convolutional network. As in the previous experiment, we convert the triangle meshes to regular occupancy voxel grids. The network architecture and training protocol are also identical to the setting in the previous experiment with the difference that we now optimize a simple mean squared error loss for the regression task instead of the cross-entropy loss. For the rotations considered in this experiment, this loss is a good approximation to the rotation angle ϕ between two quaternions $\mathbf{q}_1, \mathbf{q}_2$

$$\phi = \arccos(2 \langle \mathbf{q}_1, \mathbf{q}_2 \rangle - 1) . \tag{4.15}$$

Results The results of this evaluation are summarized in Figure 4.20, where we use two different plots for visualization. First, we show the mean angular error $\mu(\phi)$ over all test samples with respect to the input resolution. It can be observed that by increasing the input resolution the rotation estimates get continuously better. This should come to no surprise, as a detailed discretization is necessary to distinguish between the small rotation differences. In the second plot, we show the percentage of correctly estimated rotations over a range of success thresholds τ. We can see that the area under the curve is larger for higher-resolution inputs, meaning that the accuracy increases.

In Figure 4.21 we present qualitative results of the orientation estimation over the



(a) 8^3



(b) 16^3



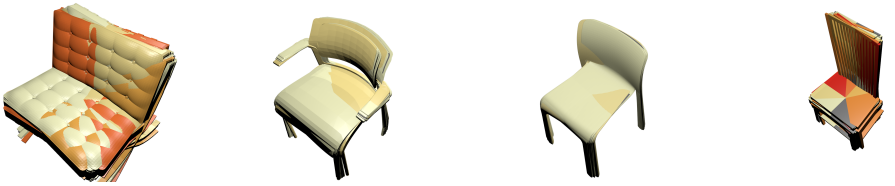
(c) 32^3



(d) 64^3



(e) 128^3



(f) 256^3

Figure 4.21: Qualitative orientation estimation results on ModelNet10.

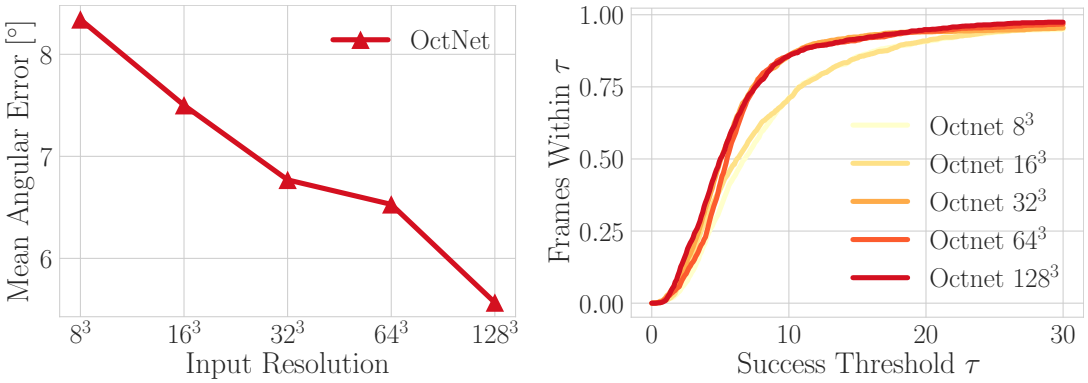
various input resolutions. Each row depicts four randomly selected chair instances, where the canonical pose is visualized with gray color and the estimated rotation with yellow if it is close the ground-truth and red if it is farther away. Note the intra-class variance of the 3D models and how the estimates are clustered more closely to the canonical pose for higher resolutions than for lower resolutions.

4.4.2.2 Head Pose

In our second 3D orientation estimation experiment we evaluate the impact of the input resolution on the task of 3D head pose estimation. The input is a single 2.5D depth map of a person in front of a depth sensor and the goal is to estimate the 3D orientation of the head. We will follow the tracking-by-detection paradigm for this task, meaning that the head pose is estimated for each frame independently. This is in contrast to the frame-by-frame temporal tracking approach, which relies on estimates of previous frames to estimate the pose of the current one.

Experimental Set-Up For this evaluation, we rely on the BiWi Kinect Head Pose Database [64, 65]. The dataset consists of 24 sequences of 20 different subjects recorded with a consumer RGB-D sensor. In total, it contains around $15,000$ frames where each subject stands approximately one meter away from the sensor and moves and rotates its head. We use the sequences of 17 subjects for training, one subject for validation and two for testing. To create the voxel grid, we project each 3D voxel center into the depth map and set the voxel occupied if the depth of the voxel center is beyond the depth value of the projected location in the depth map. The hybrid grid-octree data structure is then built by representing the border of the occupied voxels as cells on the finest resolution. In this experiment we are only interested in the rotation and therefore, assume that the head location is given. The rotation is then again represented as a unit quaternion and we use exactly the same network architecture and training protocol as in the previous experiment.

Results The quantitative results of this evaluation are summarized in Figure 4.22. First, we plot the mean angular error $\mu(\phi)$ with respect to the input resolution. We can again observe the clear trend that with higher input resolution the orientation estimates get significantly better. Note that the best mean angular error at 128^3 corresponds to a mean error of the Euler angles of $5.57°$. These are better results than previously published tracking-by-detection methods obtain, e.g. the Alternating Regression Forests by Schulter et al. [208] yield a mean Euler error of $12.2°$ and the Hough Networks by Riegler et al. [183] yield a mean Euler error of $9.8°$. However, the best results are currently obtained by frame-by-frame tracking approaches like the one by Tan et al. [229] which obtains a mean Euler error of $2.0°$. In the second plot, we show the percentage of correctly estimated frames for a certain success threshold τ. The networks with the high-resolution inputs get much



(a) Mean Angular Error **(b)** Success Threshold

Figure 4.22: Quantitative orientation estiamtion results on the head pose dataset. (a) shows the mean angular error $\mu(\phi)$ of all test samples with respect to the input resolution. (b) visualizes a cumulative error plot, where we show the percentage of correctly estimated orientations for a certain success threshold τ.

more frames correct for low success thresholds ($\tau < 5°$), but also produce less extreme outliers ($\tau > 20°$).

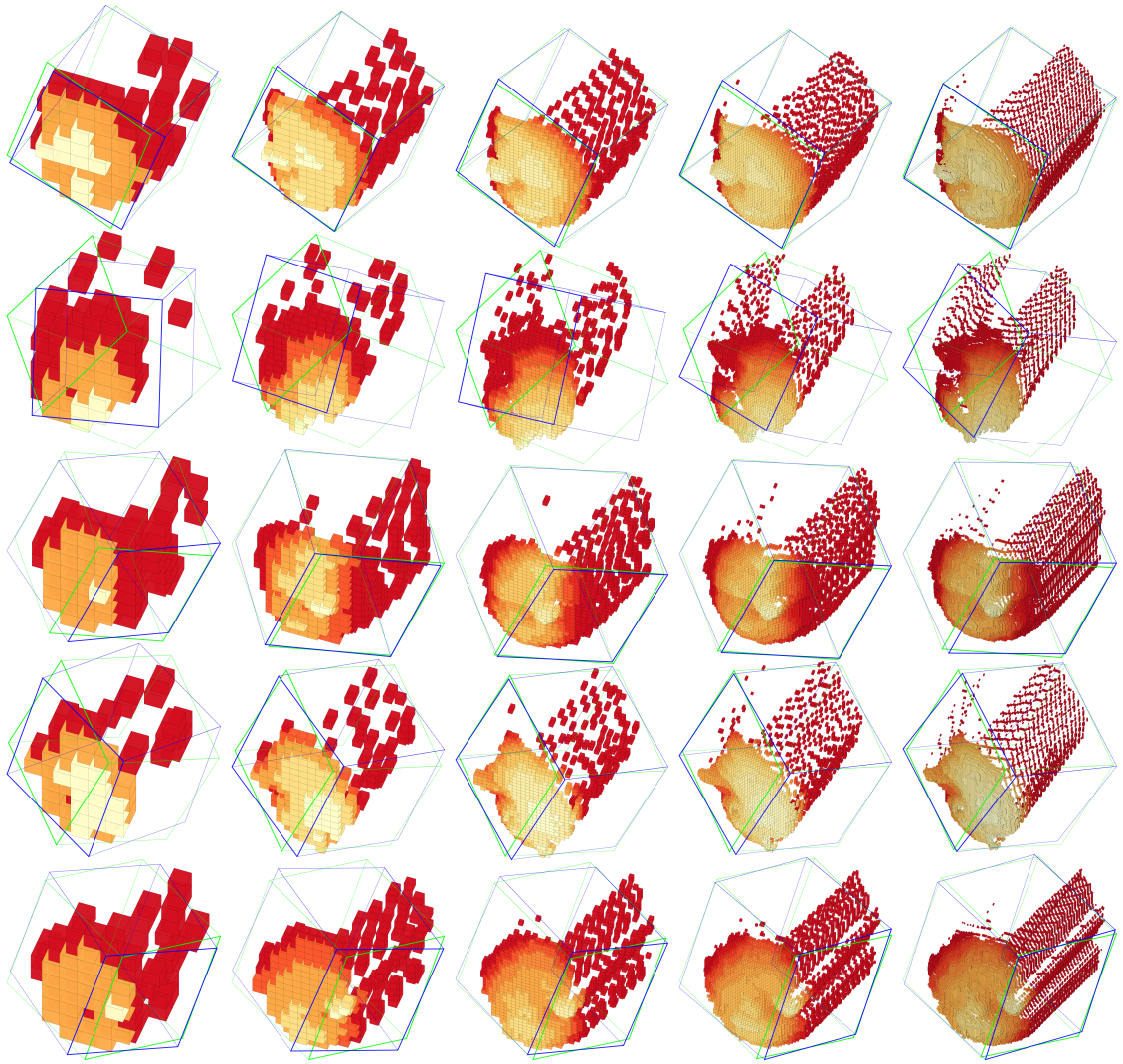
Qualitative results of the head pose estimation are visualized in Figure 4.23. We show the voxelized heads computed from the depth maps for the various input resolutions. Additionally, for each sample, the blue box indicates the ground-truth rotation and the green box the estimated one. The qualitative results also demonstrate that the orientation estimation gets better with higher input resolutions.

4.4.3 3D Point Cloud Segmentation

In this section we evaluate our OctNet method on the problem of labeling 3D point clouds with semantic information. Therefore, the task is to assign each 3D point of the input a discrete semantic label. It differs from the previous evaluations in one important aspect. Not only do we provide a 3D, volumetric input, but also the output is a 3D volume, in contrast to the 1D outputs of the previous tasks.

Experimental Set-Up We evaluate OctNet for the task of 3D point cloud labeling on the VarCity dataset [190]. The dataset provides a colored 3D point cloud of several Haussmanian style facades. It comprises approximately 1 million 3D points in total and has a defined train/test split. The labels are $L = \{window, wall, balcony, door, roof, sky, shop\}$.

To input the 3D point cloud to the network, we first convert it to a hybrid grid-octree structure. For a given input resolution we choose the voxel size such that the height of all buildings fit into the input volume. Then, we map the point cloud to the grid-octree structure by creating an octree cell on the finest resolution if it contains one, or more 3D points. For each point, we store its associated color, its normal vector, and also the height



(a) 8^3 **(b)** 16^3 **(c)** 32^3 **(d)** 64^3 **(e)** 128^3

Figure 4.23: Qualitative orientation estimation results on the head pose dataset. The green box indicates the ground-truth orientation, whereas the blue box depicts the estimates.

above the ground. If more than one point falls into one octree cell, then we average the input features and we compute the majority vote for the output label. To increase the training set size we added minimal random rotations to the local point clouds.

As network we chose U-shaped architecture with skip connection, which is a popular choice in semantic segmentation [6, 42]. The details of the network are presented in Figure 4.24. It basically consists of an encoder and a decoder part. The decoder part consists of four blocks, where each block comprises two convolution layers and a pooling layer to increase the receptive field size. The decoder part then increases the resolution of the features again by employing convolutional and unpooling layers. Note that the unpooling

Figure 4.24: U-shaped semantic segmentation network. Note, that we use guided unpooling layers. Hence, the octree structure is given by the input of the corresponding pooling layer.

	Average	Overall	IoU
Riemenschneider et al. [190]	-	-	42.3
Martinović et al. [153]	-	-	52.2
Gadde et al. [75]	68.5	78.6	54.4
OctNet 64^3	60.0	73.6	45.6
OctNet 128^3	65.3	76.1	50.4
OctNet 256^3	73.6	81.5	59.2

Figure 4.25: Quantitative results on the VarCity dataset.

layers are guided as presented in the method section. Therefore, the octree structure is derived from the corresponding pooling layer. In addition, we introduce skip connections, where the feature maps of the layers prior to the pooling layers are concatenated with the feature maps of the unpooling layers. The network architecture is kept the same independent of the input resolution.

We train the network for 10 epochs with a batch size of 4 and a constant learning rate of 10^{-3}. Further, we use Adam [128] as optimizer. The training loss is a per voxel cross-entropy function. If an octree cell does not contain a single 3D point, then the loss is automatically set to 0 and therefore does not contribute to the overall loss.

Metrics For this experiment we follow the evaluation protocol of [75]. First, we back-project the per-voxel semantic class estimates to the input 3D point cloud. Then, we compute for each label l the number of true positives (TP_l), the number of false negatives (FN_l), and the number of false positives (FP_l). The considered metrics are then the *overall pixel accuracy* given by

$$\frac{\sum_l \mathrm{TP}_l}{\sum_l \mathrm{TP}_l + \mathrm{FN}_l}\,, \tag{4.16}$$

the *average class accuracy* given by

$$\frac{1}{|L|}\sum_l \frac{\mathrm{TP}_l}{\mathrm{TP}_l + \mathrm{FN}_l}\,, \tag{4.17}$$

and the *intersection over union* given by

$$\frac{1}{|L|}\sum_l \frac{\mathrm{TP}_l}{\mathrm{TP}_l + \mathrm{FN}_l + \mathrm{FP}_l}\,. \tag{4.18}$$

Results We compare our quantitative results with the methods of Riemenschneider et al. [190], Martinović et al. [153] and Gadde et al. [75]. For each method, we consider only the variant that uses the colored 3D point cloud as input. We evaluate our OctNet method

for the input resolutions 64^3, 128^3 and 256^3. The results are summarized in Table 4.25.

The first thing that we can observe is that the input resolution for the OctNet based U-net is crucial. All metrics are getting better by increasing the resolution up to 256^3. For this input resolution, we even obtain state-of-the-art results on this dataset. The reason for this performance becomes obvious by looking at the qualitative results in Figure 4.26. On a small resolution like 64^3 many 3D points fall into the same octree cell and smooth the input too much. Hence, all the points get assigned the same semantic class. On the other hand, for the input resolution of 256^3, most of the octree cells comprise only a single 3D point. Additionally, the U-net architecture allows incorporating a lot of context through the large receptive field. For larger input resolutions the octree cells on the finest resolution are getting to sparse, i.e. the point cloud density is too low, and the information flow between neighbouring points gets interrupted.

4.4.4 Depth Fusion and Completion

In the experiments shown so far, the octree structure of the input and the output, i.e. the space partitioning of the 3D volume has been known a priori. In contrast, we present in this section experiments on tasks where this assumption is no longer valid. For depth fusion and depth completion the output geometry is different from the input geometry, i.e. the octree structure of the output is not known in advance. We will use the split module presented in Section 4.3.2 in a coarse-to-fine framework to dynamically compute the octree structure for each output in a principled manner. This coarse-to-fine network architecture is presented in Section 4.4.4.1. An important aspect of these tasks is the encoding of the input and output. Therefore, we discuss in Section 4.4.4.2 how we voxelize the N input depth maps and in Section 4.4.4.3 we show the output encoding. The volumetric depth fusion experiments are then presented in Section 4.4.4.4 and on volumetric depth completion in Section 4.4.4.5.

4.4.4.1 Network Architecture

Our overall network architecture is illustrated in Figure 4.27, which we name *OctNetFusion*. We represent the voxelized input and output using the presented hybrid grid-octree structure. The input to the network is a feature volume, calculated from a single, or multiple depth maps. The output may encode a truncated signed distance function (TSDF) or a binary occupancy map, depending on the application.

As the 3D input to our method can be incomplete, we refrain from using the classical U-shaped architecture as common for 2D-to-2D prediction tasks [6, 56]. Instead, we propose a coarse-to-fine network with encoder-decoder modules, structure manipulation modules and a loss $\mathcal{L}$ defined at every pyramid level. More specifically, we create a 3D scale pyramid where the number of voxels along each dimension increases by a factor of two between pyramid levels. At each level, we process the input using an encoder-decoder module which enlarges the receptive field and captures contextual information.

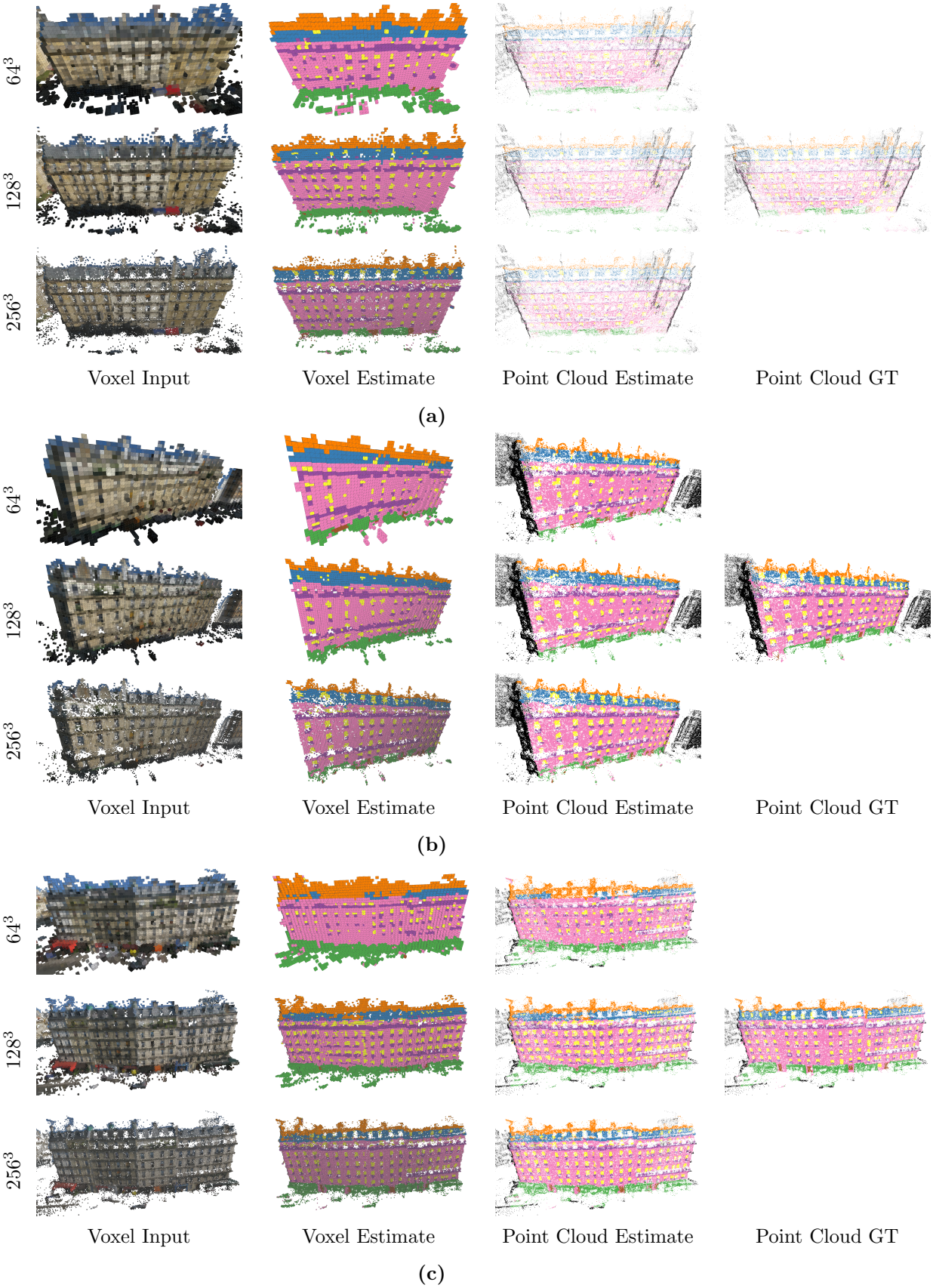


Figure 4.26: Qualitative results on the VarCity dataset for semantic 3D point cloud labeling.

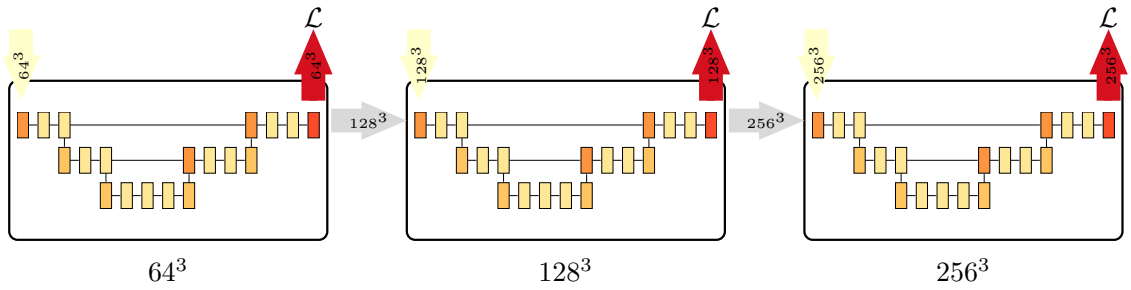


Figure 4.27: OctNetFusion coarse-to-fine network architecture.

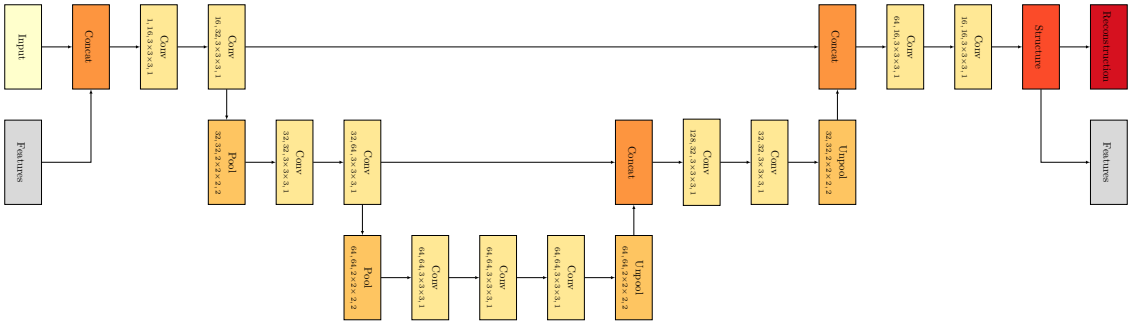


Figure 4.28: Encoder-Decoder module.

We pass the resulting features to a structure manipulation module as presented in Section 4.3.2 which computes the output at the respective resolution, increases the resolution and updates the structure of the network for further processing. We propagate features to successively finer resolutions until we have reached the final target resolution.

The core building block of the coarse-to-fine architecture is the encoder-decoder module as illustrated in Figure 4.28. It combines convolution layers with pooling and unpooling layers similar to the segmentation network used in previous semantic segmentation experiment. All convolutional layers are followed by a ReLU non-linearities [162]. The convolution layer before each pooling operation doubles the number of feature maps while the convolution layer after each unpooling operation halves the number of feature maps. Pooling operations reduce spatial information but increase the level of context captured in the features. The result of the unpooling operation is concatenated with the corresponding high-resolution features from the encoder path to combine high-resolution information with low-resolution contextual cues.

4.4.4.2 Input Encodings

The input to our method are one or more 2.5D depth maps with known extrinsic and intrinsic camera parameters $\mathcal{C}$. We now discuss several ways to project this information into 3D voxel space which, represented using grid-octree structures, forms the input to the OctNetFusion architecture described above. The traditional volumetric fusion approach [45]

calculates the weighted average TSDF with respect to all depth maps independently for every voxel where the distance to the surface is measured along the line of sight to the sensor. While providing for a simple one-dimensional signal at each voxel, this encoding does not capture all information due to the averaging operation. Thus, we also explore higher dimensional input encodings which might better retain the information present in the sensor recordings. We now formalize all input encodings used during our experiments, starting with the simplest one.

Occupancy Fusion (1D) The first and simplest encoding fuses information at the occupancy level. Let $d_v^{(i)}$ be the depth of the center of voxel v wrt. camera $i \in \mathcal{C}$. Further, let $d_c^{(i)}$ be the depth value of the projecting voxel v into the depth map for camera i. Denoting the signed distance between the two depth values $\delta_v^{(i)} = d_c^{(i)} - d_v^{(i)}$, we define the occupancy of each voxel $o(v)$ as

$$o(v) = \begin{cases} 1 & \exists i \in \mathcal{C} : \delta_v^{(i)} \leq s \wedge \nexists i : \delta_v^{(i)} > s \\ 0 & \text{else} \end{cases} \tag{4.19}$$

where s is the size of voxel v. The interpretation is as follows: If there exists any depth map in which voxel v is observed as free space the voxel is marked as free, otherwise it is marked as occupied. While simple, this input encoding is susceptible to outliers in the depth maps and does not encode uncertainty. Furthermore, the input distance values are not preserved as only occupancy information is encoded.

TSDF Fusion (1D) Our second input encoding is the result of traditional TSDF fusion as described in [45, 165]. More specifically, we project the center of each voxel into every depth map, calculate the truncated signed distance value using a truncation threshold τ (corresponding to the size of four voxels in all our experiments), and average the result over all input views

$$\begin{aligned} \text{tsdf}(v) &= \frac{1}{Z} \sum_{i \in \mathcal{C}} [\delta_v^{(i)} \geq -\tau] \min(\tau, \max(-\tau, \delta_v^{(i)})) \\ Z &= \sum_{i \in \mathcal{C}} [\delta_v^{(i)} \geq -\tau] \end{aligned} . \tag{4.20}$$

While various weight profiles have been proposed like in [224], we found that the simple constant profile proposed by Newcombe et al. [165] performs well. This input representation is simple and preserves distances, but it does not encode uncertainty and thus makes it harder to resolve conflicts.

TDF + Occupancy Fusion (2D) The TSDF encoding can also be split into two channels: One channel that encodes the truncated unsigned distance to the surface (TDF) and one that encodes occupancy. Note that if $\text{tdf}(v)$ is the TDF of voxel v, and $o(v)$

its occupancy, then $-\text{tdf}(v) \cdot o(v)$ is equivalent to $\text{tsdf}(v)$, the truncated signed distance function of voxel v.

Histogram (10D) While the previous two encodings capture surface distances, they do not maintain the multi-modal nature of fused depth measurements, nor do they handle uncertainties in the input. To capture this information, we propose a histogram-based representation. In particular, we encode all distance values for each voxel using a 10D histogram with 5 bins for negative and 5 bins for positive distance values. The first and the last bin of the histogram capture distance values beyond the truncation limit, while the bins in between collect non-truncated distance values. To allow sub-voxel surface estimation, we choose the histogram size such that a minimum of 2 bins is allocated per voxel. Furthermore, we populate the histogram smoothly by distributing the vote of each observation linearly between the two closest bins, e.g. we assign half of the mass to both neighboring bins if the prediction is located at their boundary.

4.4.4.3 Output Encodings

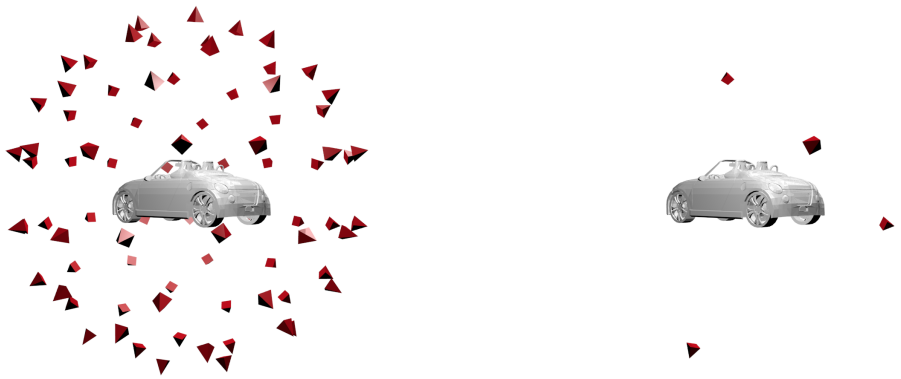
Finally, we describe the output encodings and the loss functions we use for the volumetric depth fusion and the volumetric depth completion tasks considered in this experimental evaluation.

Volumetric Depth Fusion For volumetric fusion, we choose the TSDF as output representation using an appropriate truncation threshold τ. Note that in contrast to binary occupancy, TSDF outputs allow for predicting implicit surfaces with sub-voxel precision. We regress the TSDF values at each resolution within the structure module using a linear activation function and use the ℓ_1 loss for training.

Volumetric Depth Completion For volumetric completion from a single view, we use a binary occupancy representation to match the set-up of the baselines as closely as possible. Following common practice, we leverage the binary cross entropy loss to train the network.

4.4.4.4 Volumetric Depth Fusion

First, we evaluate our OctNetFusion method on the task of volumetric depth fusion. We will compare our results to the traditional volumetric fusion approach of Curless and Levoy [45] denoted as *VolFus* and the variational approach of Zach et al. [268] denoted as *TV-L1*. The parameters of those methods have been tuned on the validation set using grid search.



(a) Ground-truth set-up **(b)** Input set-up

Figure 4.29: To acquire the TSDF ground-truth targets we render synthetic depth maps without noise from 80 views sampled from a sphere as depicted in (a). As input we use only a subset of these views, e.g. 4 as depicted in (b), and add depth-dependent noise to obtain a challenging volumetric depth fusion set-up.

Experimental Set-Up We conduct a series of experiments on the synthetic ModelNet40 dataset [259], where we use 10 different categories, i.e. *airplane*, *bed*, *car*, *desk*, *chair*, *guitar*, *piano*, *person*, *toilet*, *piano*. For training, validation, and testing we use 100, 5 and 20 samples per category, respectively. Unfortunately, the ground-truth TSDF cannot be calculated directly from the 3D models in this dataset as the meshes are not watertight, i.e. they contain holes and cracks. Moreover, the meshes typically do not have consistently oriented normals. Instead, we obtain the ground truth TSDF by densely sampling views around the object, rendering the input 3D model from all views and running traditional volumetric fusion on all those generated (noise-free) depth maps. Towards this goal, we scaled the ModelNet objects to fit into a cube of $3 \times 3 \times 3$ meters and rendered the depth maps onto equally spaced views sampled from a sphere. We found that 80 views cover all object surfaces and hence, allow for computing highly accurate TSDF ground-truth.

To generate the input we again rendered depth maps from the 3D model, but use only N views. In our experiments, we use $N = 4$, if not otherwise stated. See Figure 4.29 to compare the two different set-ups for ground-truth and input generation. Additionally, we add depth dependent Gaussian noise to the depth maps to simulate the noisy acquisition of the depth maps. If d is the noise-free depth map from the rendering, then the noisy depth map d_n for the input is generated by

$$d_n = d\left(1 + \mathcal{N}(0, \sigma^2)\right) , \tag{4.21}$$

where $\mathcal{N}$ is a random value sampled from the normal distribution with σ set to 0.02 in our experiments, if not otherwise stated.

We train our coarse-to-fine architecture stage-wise, i.e. one resolution after another. For each stage we use a batch size of 4 samples, a constant learning rate of 10^{-4} and

Adam [128] as optimizer. The training objective is the ℓ_1 loss with an additional weight decay term of 10^{-4} for regularization. We train the first stage for 50 epochs, and the next two stages for 25 epochs, respectively. Note that we have to initialize the first stage randomly according to the scheme proposed in [103], but the other stages are initialized by the weights of the previous stage.

Metrics For the experiments, we investigate three different evaluation metrics that are directly evaluated on the volumetric voxel grid. As our output is a hybrid grid-octree structure, we convert it back to a voxel grid for the evaluation. Assume that $\mathbf{f}$ denotes the estimated TSDF and $\mathbf{t}$ is the ground-truth TSDF with each having N voxels, then the considered metrics are: The mean absolute error (MAE) given by

$$\mathrm{MAE}(\mathbf{f}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} |f_i - t_i| \,, \tag{4.22}$$

the root mean squared error (RMSE) given by

$$\mathrm{RMSE}(\mathbf{f}, \mathbf{t}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f_i - t_i)^2} \,, \tag{4.23}$$

as well as the Jaccard index given by

$$\mathrm{Jaccard}(\mathbf{f}, \mathbf{t}) = \frac{\sum_{i=1}^{N} [f_i \leq 0 \wedge t_i \leq 0]}{\sum_{i=1}^{N} [(f_i \leq 0 \wedge t_i \leq 0) \vee (f_i \leq 0 \wedge t_i > 0) \vee (f_i > 0 \wedge t_i \leq 0)]} \,. \tag{4.24}$$

The last metric computes the intersection over union of the occupied voxels, whereas a voxel is considered occupied if it has a negative TSDF,

Input Encoding We first investigate the impact of the input encodings on the reconstruction performance. Our quantitative results are shown in Table 4.2. We observe that our model outperforms the traditional fusion approach [45], as well as TV-L1 fusion [268] by a large margin independent of the input encoding and evaluation metric. Improvements are particularly pronounced at high resolutions which demonstrates that our learning based approach is able to refine and complete geometric details which can not be recovered using existing techniques. Furthermore, we observe that the TSDF histogram encoding yields the best results. We thus use this input encoding for all remaining experiments.

Qualitative results are visualized in Figure 4.30. We can clearly observe the strengths of our method: It is able to perform a nice smoothing of the surface, but at the same time is still able to recover thin structures and details. Additionally, it can produce meaningful results in areas that are either unobserved or occluded. This is especially pronounced at the example with the desk. Our method is able to recover the thin table leg at higher

Table 4.2: Quantitative volumetric depth fusion results of different input encodings.

(a) MAE (mm)

	VolFus [45]	TV-L1 [268]	Occ	TDF + Occ	TSDF	TSDF Hist
64^3	4.136	3.899	2.095	1.987	1.710	1.715
128^3	2.058	1.690	0.955	0.961	0.838	0.736
256^3	1.020	0.778	0.410	0.408	0.383	0.337

(b) RMSE (mm)

	VolFus [45]	TV-L1 [268]	Occ	TDF + Occ	TSDF	TSDF Hist
64^3	17.408	16.888	8.518	7.939	7.466	7.243
128^3	9.903	9.012	4.816	4.656	4.291	4.081
256^3	5.431	4.630	2.538	2.486	2.370	2.296

(c) Jaccard Index

	VolFus [45]	TV-L1 [268]	Occ	TDF + Occ	TSDF	TSDF Hist
64^3	0.683	0.674	0.787	0.797	0.824	0.818
128^3	0.636	0.596	0.791	0.803	0.819	0.828
256^3	0.615	0.630	0.810	0.820	0.824	0.834

resolutions but also learnt that the space between the table tops has to be free space. The other methods struggle with unobserved voxels as the assume those to be occupied.

Number of Input Views Next, we evaluate the performance of our network by varying the number of input views from one to six on the ModelNet dataset. Our results are shown in Table 4.4 and the qualitative results of are visualized in Figure 4.31. Again, our approach outperforms both baselines in all considered metrics. As expected the performance of all methods increases with the number of input views. The largest difference between the baselines and our approach is visible in the experiment with only one input view. In this depth completion task, the baseline approaches completely lack the ability to infer geometry from occluded regions. In contrast, our deep learning method can reconstruct details to some degree from the samples it has seen during training. But also in the cases with an increased number of input views, e.g. four, our proposed method reduces the error by a factor of 2 to 3 wrt. TSDF fusion and TV-L1 fusion in terms of MAE.

Input Noise In our next experiment we evaluate the impact of the noise in the depth maps on the reconstruction results. Table 4.6 summarizes our results. We observe that our method is faithful with respect to the increase of input noise. The mean absolute error increases from 0.274 mm for no noise to only 0.374 mm for severe noise ($\sigma = 0.03$). In contrast, for TSDF fusion the mean absolute error increases by more than 0.5 mm and for TV-L1 fusion by more than 0.2 mm. This is also true for the Jaccard index, where our method only decreases by less than 9%, but the TSDF fusion decreases by more than 24% and TV-L1 by more than 22%. The qualitative results of this evaluation are visualized

Table 4.4: Quantitative volumetric depth fusion results wrt. number of input views.

(a) MAE (mm)

	views=1			views=2			views=4			views=6		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	59.295	48.345	7.855	15.626	13.267	2.755	4.136	3.899	1.715	3.171	2.905	1.484
128^3	29.795	26.525	3.853	7.850	6.999	1.333	2.058	1.690	0.736	1.648	1.445	0.661
256^3	14.919	14.529	1.927	3.929	3.537	0.616	1.020	0.778	0.337	0.842	0.644	0.360

(b) RMSE (mm)

	views=1			views=2			views=4			views=6		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	103.708	80.406	25.789	47.227	38.081	11.307	17.408	16.888	7.243	13.649	13.021	6.117
128^3	52.384	46.769	14.197	24.386	21.244	6.508	9.903	9.012	4.081	8.256	7.754	3.736
256^3	26.333	25.667	7.649	12.454	11.389	3.697	5.431	4.630	2.296	4.759	4.084	2.327

(c) Jaccard Index

	views=1			views=2			views=4			views=6		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	0.135	0.119	0.451	0.355	0.048	0.748	0.683	0.674	0.818	0.720	0.710	0.836
128^3	0.130	0.122	0.439	0.338	0.139	0.758	0.636	0.596	0.828	0.644	0.648	0.834
256^3	0.128	0.126	0.416	0.330	0.256	0.760	0.615	0.630	0.834	0.605	0.596	0.817

Table 4.6: Quantitative volumetric depth fusion results wrt. input noise.

(a) MAE (mm)

	$\sigma = 0.00$			$\sigma = 0.01$			$\sigma = 0.02$			$\sigma = 0.03$		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	3.020	3.272	1.647	3.439	3.454	1.487	4.136	3.899	1.715	4.852	4.413	1.938
128^3	1.330	1.396	0.744	1.647	1.543	0.676	2.058	1.690	0.736	2.420	1.850	0.804
256^3	0.621	0.637	0.319	0.819	0.697	0.321	1.020	0.778	0.337	1.188	0.858	0.402

(b) RMSE (mm)

	$\sigma = 0.00$			$\sigma = 0.01$			$\sigma = 0.02$			$\sigma = 0.03$		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	14.956	15.328	6.812	15.731	15.729	6.701	17.408	16.888	7.243	19.331	18.434	7.699
128^3	7.828	7.931	3.946	8.599	8.496	3.750	9.903	9.012	4.081	11.061	9.585	4.358
256^3	4.038	4.062	2.307	4.677	4.290	2.118	5.431	4.630	2.296	6.014	4.975	2.179

(c) Jaccard Index

	$\sigma = 0.00$			$\sigma = 0.01$			$\sigma = 0.02$			$\sigma = 0.03$		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	0.799	0.769	0.848	0.755	0.742	0.846	0.683	0.674	0.818	0.615	0.609	0.804
128^3	0.805	0.786	0.859	0.725	0.662	0.860	0.636	0.596	0.828	0.570	0.525	0.806
256^3	0.804	0.794	0.864	0.699	0.705	0.857	0.615	0.630	0.834	0.555	0.568	0.806

Table 4.8: Quantitative volumetric depth fusion results wrt.. seen vs. unseen categories.

(a) MAE (mm)

		VolFus [45]	TV-L1 [268]	Seen	Unseen
all	64^3	4.136	3.899	1.715	1.686
	128^3	2.058	1.690	0.736	0.799
	256^3	1.020	0.778	0.337	0.358
airplane	64^3	0.668	0.583	0.419	0.470
	128^3	0.324	0.297	0.174	0.192
	256^3	0.157	0.111	0.076	0.076
desk	64^3	5.122	4.767	1.954	2.000
	128^3	2.540	2.165	0.777	0.898
	256^3	1.260	0.987	0.334	0.383

(b) RMSE (mm)

		VolFus [45]	TV-L1 [268]	Seen	Unseen
all	64^3	17.408	16.888	7.243	7.384
	128^3	9.903	9.012	4.081	4.263
	256^3	5.431	4.630	2.296	2.357
airplane	64^3	5.466	4.999	2.637	2.732
	128^3	3.450	3.428	1.271	1.470
	256^3	2.048	1.664	0.749	0.818
desk	64^3	20.953	20.239	7.605	8.166
	128^3	11.930	11.142	4.236	4.749
	256^3	6.537	5.747	2.394	2.596

(c) Jaccard Index

		VolFus [45]	TV-L1 [268]	Seen	Unseen
all	64^3	0.683	0.674	0.818	0.816
	128^3	0.636	0.596	0.828	0.816
	256^3	0.615	0.630	0.834	0.827
airplane	64^3	0.677	0.657	0.801	0.787
	128^3	0.617	0.550	0.841	0.815
	256^3	0.594	0.627	0.856	0.845
desk	64^3	0.643	0.635	0.818	0.810
	128^3	0.592	0.531	0.837	0.817
	256^3	0.569	0.559	0.841	0.820

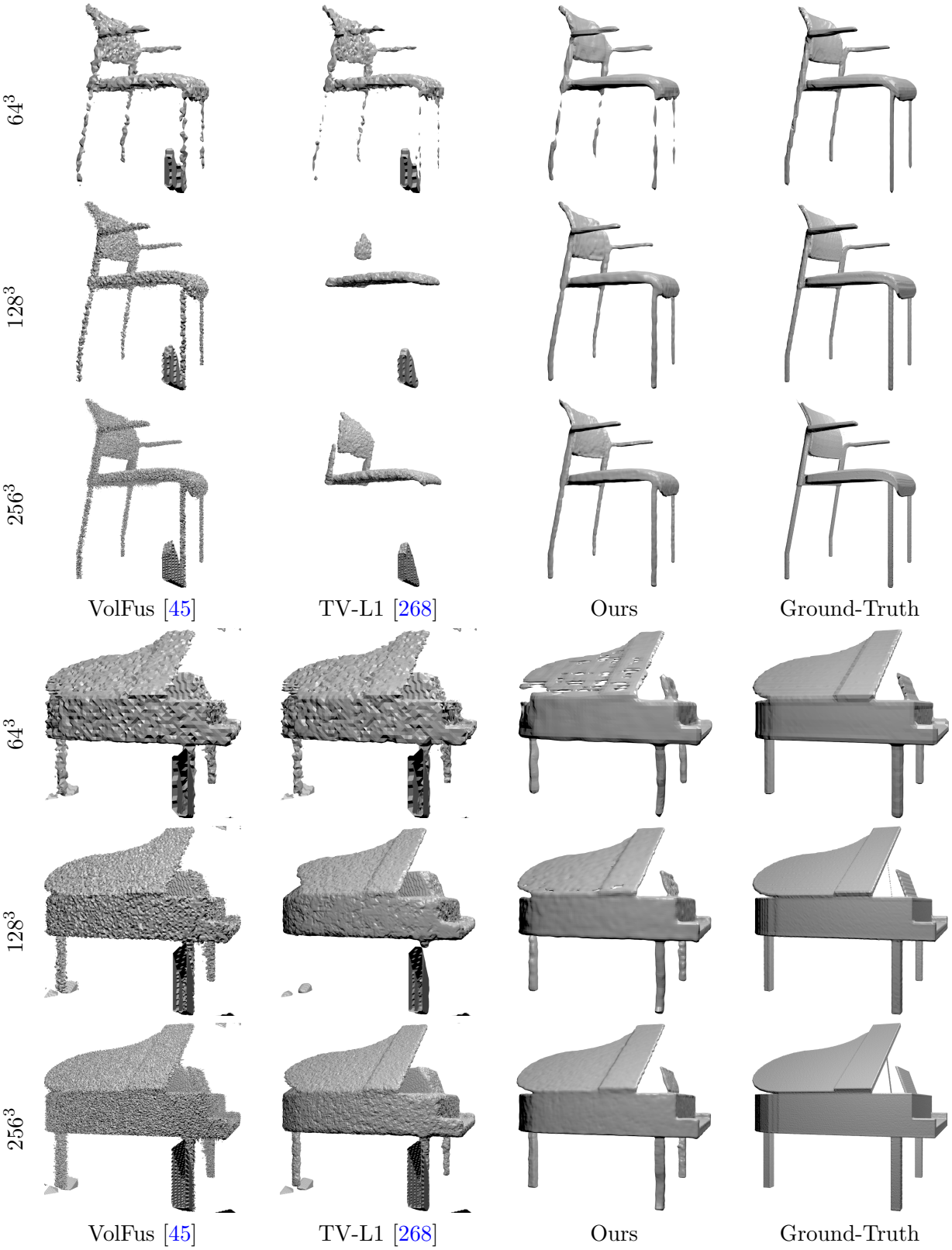
in Figure 4.32. Note how our proposed method is able to reconstruct the side mirror of the car up to a noise of $\sigma = 0.02$ and simultaneously produces a much smoother result. Further, our deep learning based result is closer to the ground-truth even in the case of no input noise . It learns to smooth the result given only four input depth maps. However, a drawback of this method is that it incorporates a small shrinkage bias on flat and thin surface as visible on the desk example.

Generalization on Unseen Categories Most existing approaches that leverage deep learning for 3D reconstruction train a model specific for a particular object class [24, 41, 84, 212] which typically does not generalize well to other classes, or scenes with varying backgrounds. In contrast, here we are interested in the 3D reconstruction of general scenes. Therefore, we analyze how our model behaves on shapes that were not seen during training. In Table 4.8 we trained a network on only 8 categories out of 10 and use the two unseen ones for testing, *Unseen*. We compare these results to the case where we train on all 10 categories, *Seen*. Note that in neither case training shapes are part of the test set, but shapes from the same category are used or ignored. As expected we can observe a slight decrease in performance on unseen categories, but no real drop. If compared to the baseline methods our method is still better by a multiplicative factor. This underlines our motivation that we are able to generalize on unseen scenes and objects.

Kinect Object Scans In the following experiment, we evaluate our volumetric depth fusion method on real data from a Kinect sensor In particular, we use the videos captured by Choi et al. [39] which include a diverse set of objects such as chairs, tables, trash containers, plants, signs, etc. Unfortunately, the dataset does not include ground-truth 3D models or camera poses. We thus estimated the camera poses using Kintinuous [256] and visually inspect all models to remove those for which the pose tracking failed. Similar to the previous experiments on the ModelNet dataset we compute reference 3D models by TSDF fusion. However, for this dataset, we leverage for each scene all the views from the corresponding video, which are on average >1000 viewpoints. This is possible as the dataset has been collected with slow camera motion and many redundant views. At test time we provide only a small fraction of views, i.e. 10 and 20, to each algorithm to simulate challenging real-world conditions. Further, this enables us to augment the training data by sampling different subsets for the input.

A drawback of this particular dataset is that despite the many viewpoints the created ground-truth TSDF volumes are still incomplete for most scenes: The scenes contain unobserved regions, as well as artifacts due to sensor noise and missing depth values. For those reasons, we utilize an evaluation mask per sample during training and testing. First, we exclude voxels that were observed from no camera viewpoint, i.e. the voxel is outside the union of the viewing cones. Second, we further only consider voxels that have a TSDF value larger than the negative truncation threshold in order to handle unobserved regions. The network training set-up is the same as in the ModelNet evaluations with the only difference that we initialized our networks with the weights from the ModelNet experiment with 4 input views.

We present our quantitative results in Table 4.10 and qualitative results in Figure 4.33. Even on this challenging task, we can report better results than the baseline methods, although not as pronounced as in the previous evaluations. The qualitative results reveal that our learned fusion method generates more complete results than TV-L1, but at the same time also drastically reduces the noise in comparison to vanilla TSDF fusion. In


64^3
128^3
256^3
VolFus [45]
TV-L1 [268]
Ours
Ground-Truth
64^3
128^3
256^3
VolFus [45]
TV-L1 [268]
Ours
Ground-Truth

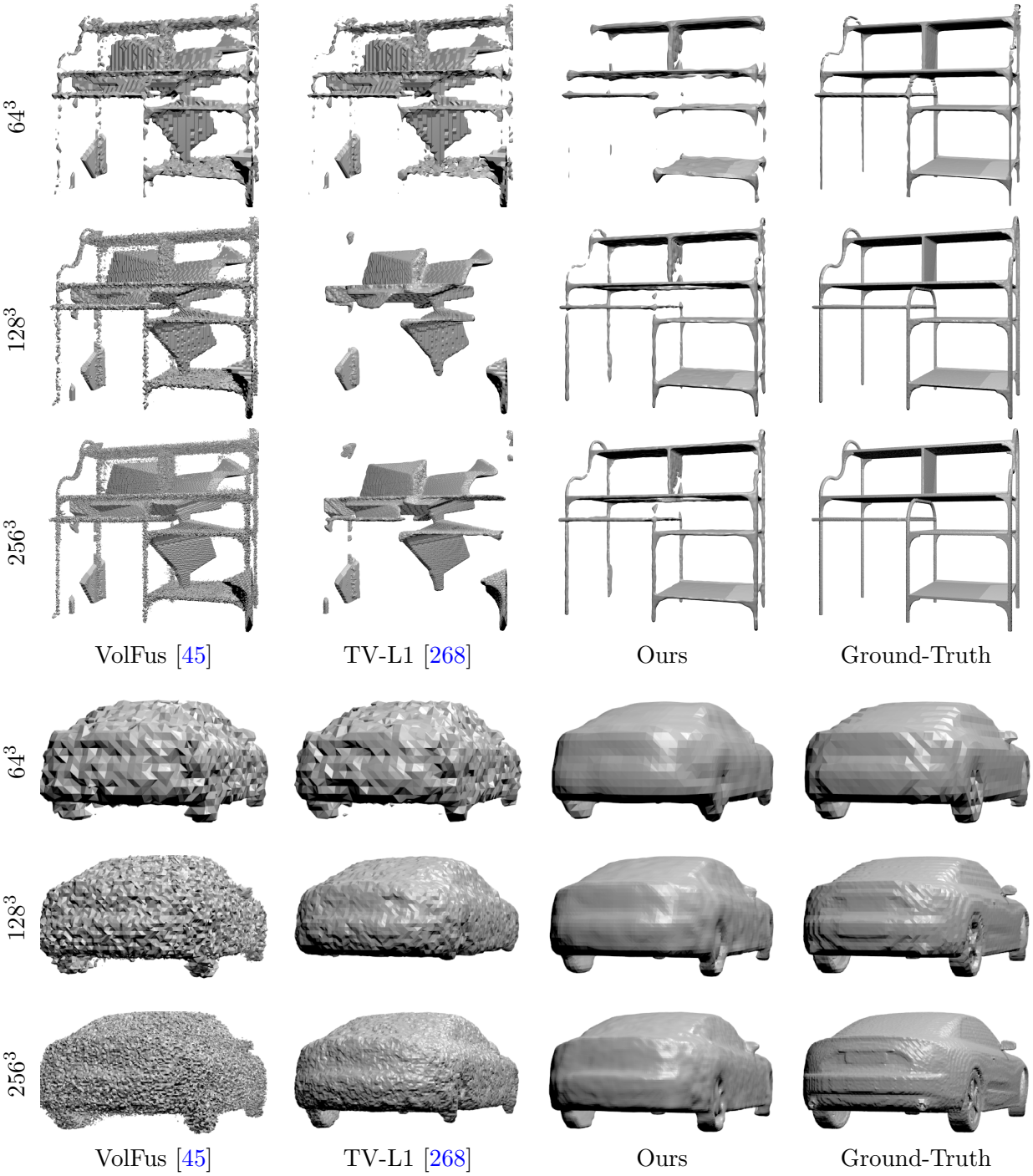


Figure 4.30: Qualitative volumetric depth fusion results for different output resolutions.

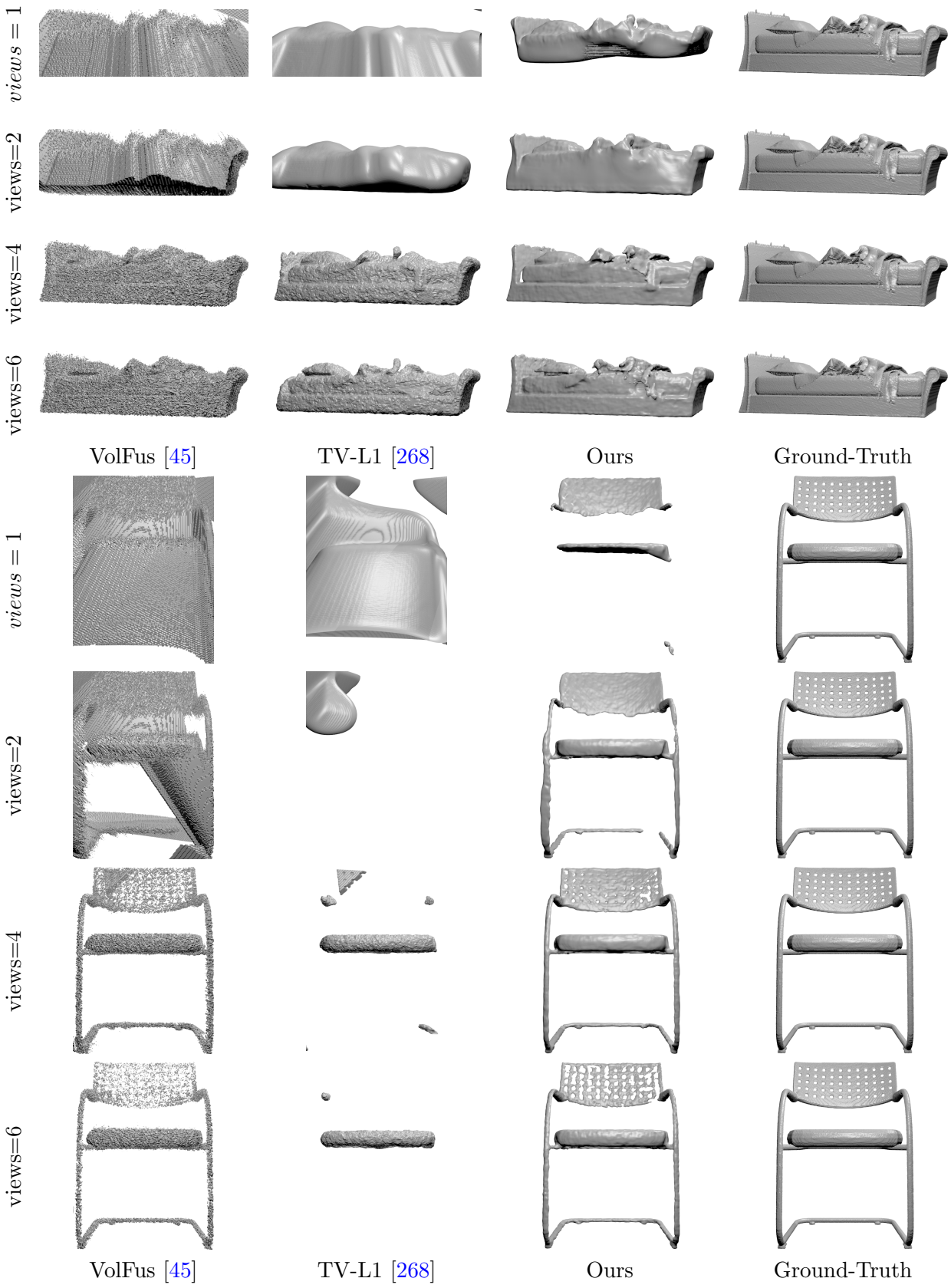


Figure 4.31: Qualitative volumetric depth fusion results wrt. number of input views.

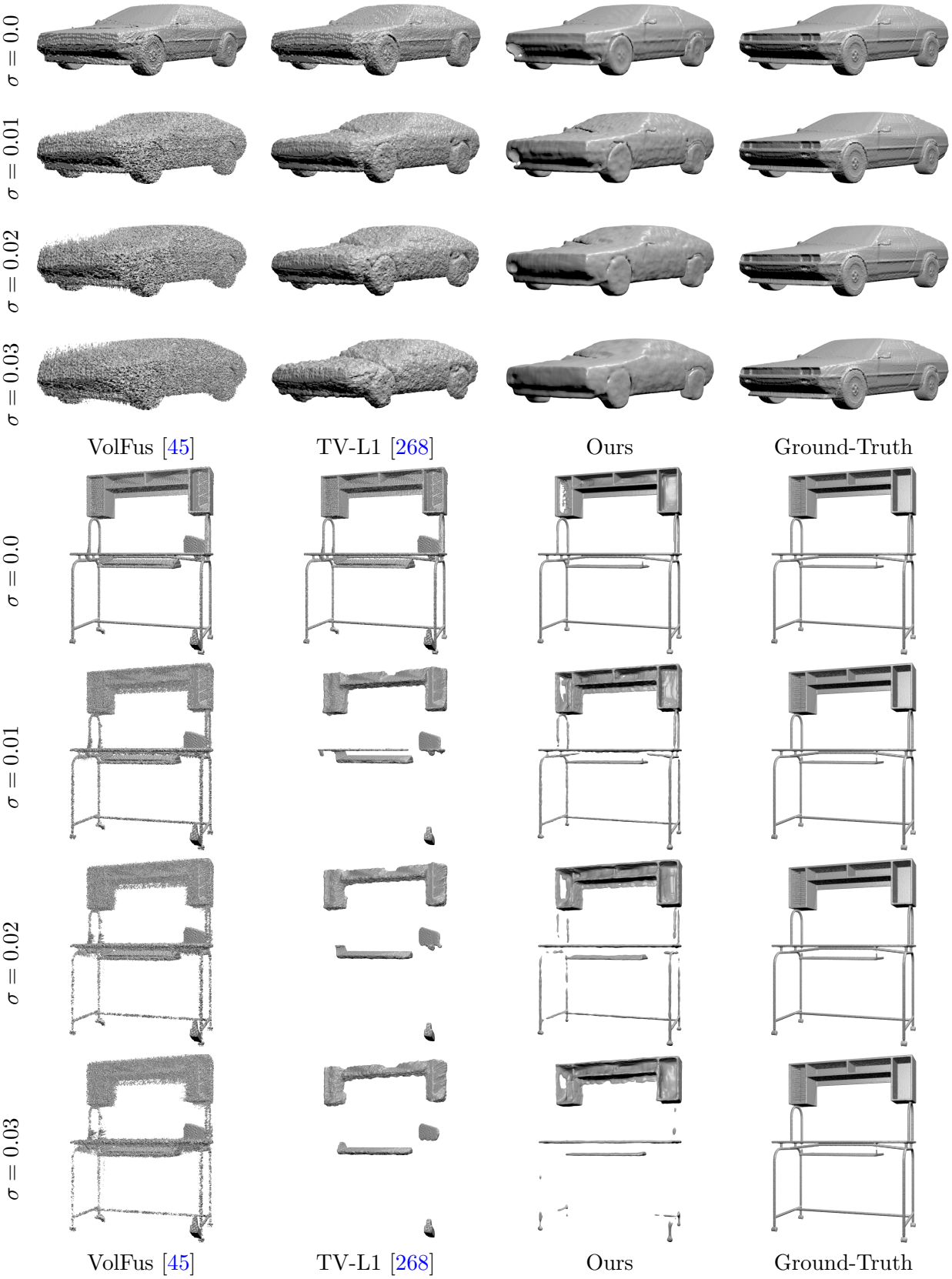


Figure 4.32: Qualitative volumetric depth fusion results wrt. input noise.

Table 4.10: Evaluation on Kinect object scans

(a) MAD (mm)

	views=10			views=20		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	103.855	25.976	22.540	72.631	22.081	18.422
128^3	58.802	12.839	11.827	41.631	11.924	9.637
256^3	31.707	5.372	4.806	22.555	5.195	4.110

(b) RMSE (mm)

	views=10			views=20		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	164.977	46.750	42.281	134.966	40.508	35.844
128^3	89.987	24.916	23.778	74.126	23.818	20.436
256^3	47.631	11.378	10.957	39.442	11.024	9.602

(c) Jaccard Index

	views=10			views=20		
	VolFus [45]	TV-L1 [268]	Ours	VolFus [45]	TV-L1 [268]	Ours
64^3	0.408	0.743	0.756	0.499	0.782	0.789
128^3	0.252	0.591	0.579	0.328	0.648	0.653
256^3	0.151	0.343	0.462	0.206	0.517	0.535

our opinion, the main bottleneck in this evaluation is the quantity and quality of the ground-truth data. We expect the same performance gap as in the previous experiments by utilizing datasets with more training data and better quality.

Runtime Given its simplicity, Vanilla TSDF fusion [45] is the fastest method, using just a few milliseconds on a GPGPU. In contrast, TV-L1 fusion [268] is computationally more expensive. We ran all experiments using 700 iterations of the optimization algorithm. For an output resolution of 64^3, TV-L1 needs 0.58 seconds on average and for an output resolution of 256^3 it needs 24.66 seconds on average. In comparison, our proposed OctNetFusion CNN requires 0.005 seconds on average for an output resolution of 64^3 and 10.1 seconds on average for an output resolution of 256^3. All numbers were obtained on an NVidia K80 GPGPU.

4.4.4.5 Volumetric Depth Completion

A single depth map contains 2.5D information, i.e. for each pixel location the associated depth is known. Hence, from a given viewpoint one can infer the free space up to the first surface that is encountered along the viewing ray. It is however impossible to reason about the occupancy of the volume behind. This makes the building of a full 3D model from a single 2.5D input a heavily under-constrained problem. In the previous experiments

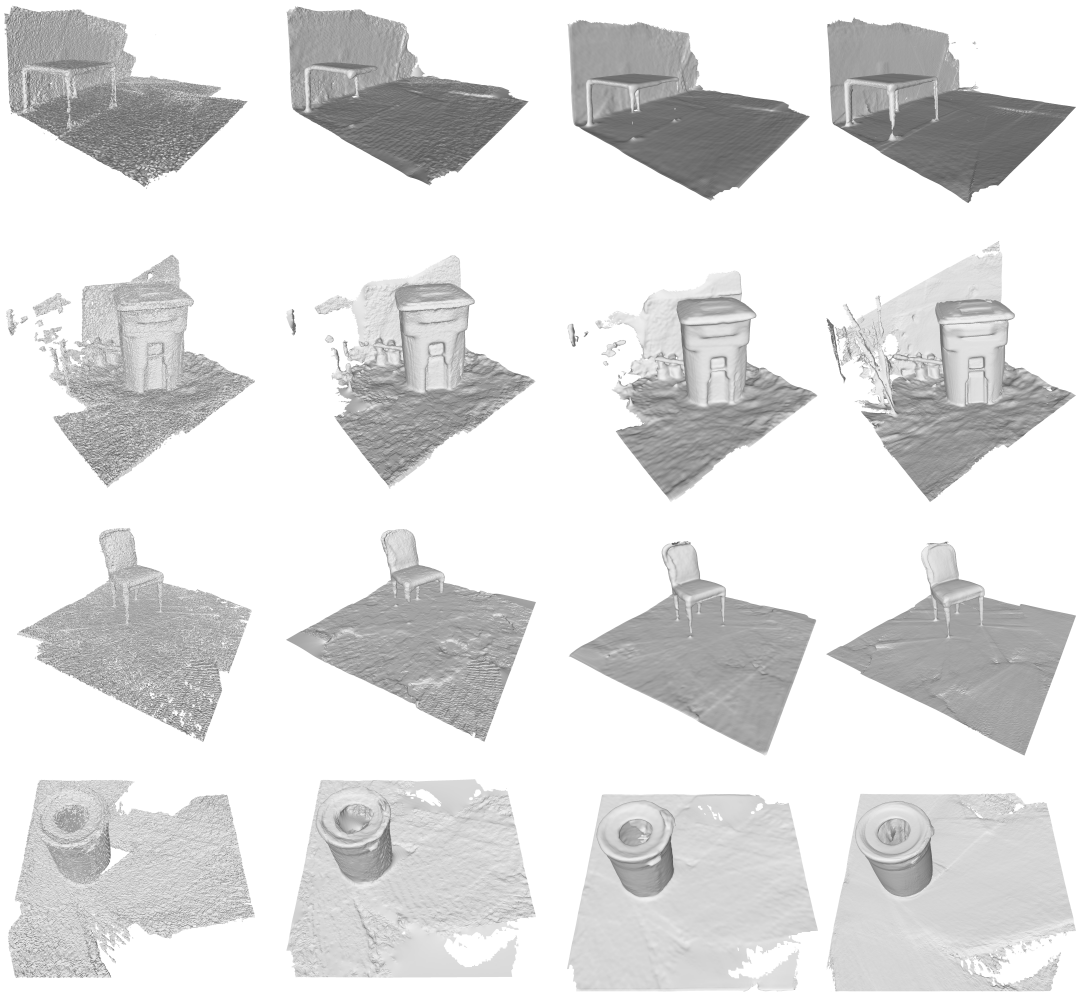


Figure 4.33: Qualitative volumetric depth completion results on the Kinect object scans.

with the varying number of input views we already presented the special case with a single view. However, the experiment was object-centric. There was a limited number of object categories where a reasonable completion could be learnt. In this experiment, we investigate a more general setting, where the categories are unknown.

Experimental Set-Up For the volumetric depth completion evaluation we use the tabletop dataset by Firman et al. [71]. It contains the full geometry reconstructed with KinectFusion [165] of 90 scenes recorded on a flat surface. Each scene contains 2 to 6 objects that were chosen from a set of 50 different objects. The dataset contains a predefined train/test split, where 60 scenes are in the train set and 30 scenes are in the test set. The split also ensures that no object from the train set was reused in the test set. Each train scene also provides 5 depth maps. From the 60 training scenes, we use 5

Table 4.12: Quantitative volumetric depth completion results on the tabletop dataset.

Method	IoU	Precision	Recall
[Zheng et al.]*	0.528	0.773	0.630
[Firman et al.]*	0.585	0.793	0.658
[Firman et al.]	0.550	0.734	0.705
Ours	0.650	0.834	0.756

for validation and the rest for training OctNetFusion. Therefore, we use 275 train samples for training and 25 for validation, respectively.

We train the same OctNetFusion architecture as in the volumetric depth fusion experiments, also with the same training protocol. The only difference is that we initialize the networks with the parameter of the ModelNet completion experiment. This helps to overcome the problem with the few training examples.

Metrics For the evaluation, we follow the same principles as outlined in Firman et al. [71]. Our method outputs binary occupancy values in a hybrid grid-octree. Those are first converted into a regular voxel grids. The considered metrics are all evaluated on these binary occupancy grids. Therefore, we assume that $\mathbf{f}$ denotes the estimated binary occupancy voxel grid and $\mathbf{t}$ is the ground-truth one with each having N voxels, then the considered metrics are: The intersection over union given by

$$\text{IoU}(\mathbf{f}, \mathbf{t}) = \frac{\sum_{i=1}^{N} [f_i = 1 \wedge t_i = 1]}{\sum_{i=1}^{N} [(f_i = 1 \wedge t_i = 1) \vee (f_i = 1 \wedge t_i = 0) \vee (f_i = 0 \wedge t_i = 1)]}, \tag{4.25}$$

the precision given by

$$\text{precision}(\mathbf{f}, \mathbf{t}) = \frac{\sum_{i=1}^{N} [f_i = 1 \wedge t_i = 1]}{\sum_{i=1}^{N} [(f_i = 1 \wedge t_i = 1) \vee (f_i = 1 \wedge t_i = 0)}, \tag{4.26}$$

and the recall given by

$$\text{recal}(\mathbf{f}, \mathbf{t}) = \frac{\sum_{i=1}^{N} [f_i = 1 \wedge t_i = 1]}{\sum_{i=1}^{N} [(f_i = 1 \wedge t_i = 1) \vee (f_i = 0 \wedge t_i = 1)]}, \tag{4.27}$$

The intersection over union measures the fractional overlap between the estimated and the ground-truth reconstruction. The precision metric penalizes the number of false positives, i.e. over-complete reconstructions, and the recall metric penalizes the number of false negatives, i.e. under-complete reconstructions.

Results The quantitative results of this volumetric depth completion experiment are summarized in Table 4.12. We compare to an idealized version of the method by Zheng

et al. [272] as proposed by [71] and to the structured random forest by Firman et al. [71]. Note that we report the number for the latter method twice. This is due to the fact that we once report the numbers as stated in their paper (indicated with an asterisk), and once by using the provided data from the authors. Unfortunately, we were not able to reproduce their results even after communication with the authors due to post-publication changes in their dataset. Nevertheless, as evidenced by our results, our method improves upon all numbers. While it only slightly improves in terms of the precision metric, it produces a significant improvement in terms of recall. Hence, we produce slightly less false positive occupied voxels but simultaneously estimating much less false negatives. This can also be observed in the qualitative results visualized in Figure 4.34. Looking at the first row, our method is able to reconstruct larger portions of the Lego cube but simultaneously producing no flying voxels that do not belong to the scene.

4.5 Summary & Discussion

This chapter of the thesis was devoted to deep learning on 3D data. There exist several possible representations of 3D data, like 3D point clouds, triangle meshes, or implicit surfaces. However, the most successful deep convolutional networks are defined on regular spaced grids, i.e. for image classification on 2D pixel grids. So, the natural way is to elevate 2D networks and its operations to 3D voxel grids. In this way, operations like convolution and pooling, but also the hierarchically stacking of those operations to successful network architectures can be easily reused for 3D tasks. But the major drawback of this approach is the drastic increase in memory consumption. While in 2D the memory consumption increases quadratically by doubling the input resolution, in 3D the memory consumption already increases cubically. The memory on modern GPGPUs is rather limited, hence, the typical input resolution is around 30^3 for current methods, while for many problems a higher resolution would be necessary to encode the fine details of 3D objects.

In this chapter, we proposed the utilization of an efficient 3D space partitioning data structure within the deep network to focus the memory, but also the computational effort on interesting parts of the input, i.e. the 3D surface of the object. More precisely, we introduced a grid of shallow octrees to convolutional networks, where we share memory and computational resources in larger octree cells, for example in empty regions of the input, while voxels near the surface are represented with octree cells that capture the fine resolution. We further defined the common network operations on this data structure like convolution and pooling such that they are equivalent to the operations on regularly voxel grids up to the pooled representation within larger octree cells.

In our experiments, we first demonstrated that with this novel network representation we could boost the network input resolution by at least a factor of 64 while keeping the network architecture and training setting the same. At the same time, we showed that the pooling inside larger octree cells did not decrease the network performance. Interestingly, we found that with higher input resolution we could not significantly increase the

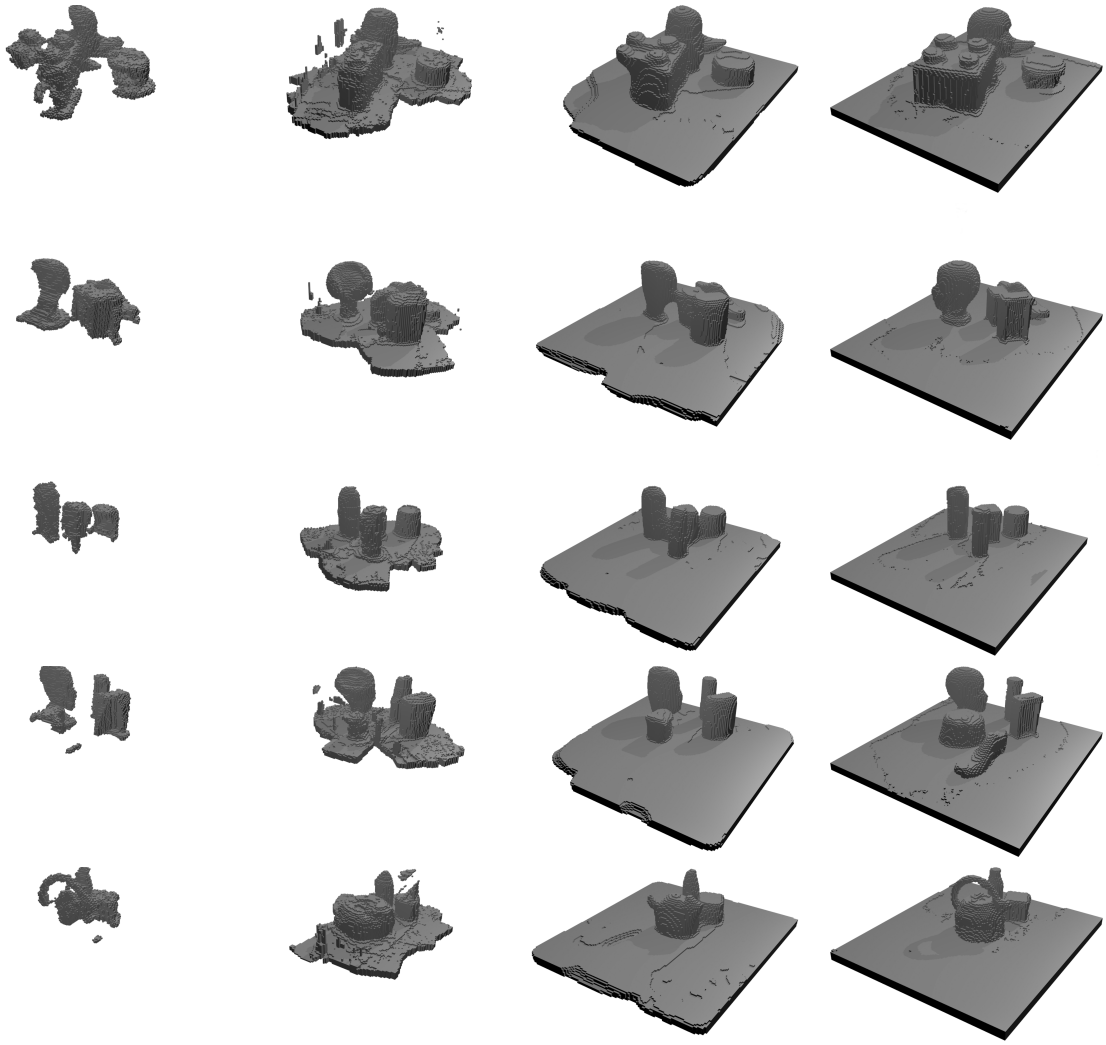


Figure 4.34: Qualitative volumetric depth completion results on the tabletop dataset.

classification accuracy of 3D shapes. We could only demonstrate a small improvement in terms of classification accuracy on the standard benchmark dataset. By close inspection of the results, we observed that most of the errors stem from ambiguous object categories rather than from similarities on low input resolutions. However, we could demonstrate a clear benefit by increasing the input resolution on tasks such as 3D orientation estimation and 3D semantic segmentation. On those tasks, the fine details that are only visible at higher resolutions had a greater impact on the performance.

We further went on to use our proposed octree-based convolutional networks for volumetric depth fusion and volumetric depth completion. The main challenge here was, that the 3D space partitioning of the output is not known a priori, but has to be estimated along with the 3D reconstruction. We tackled this problem by relying on a coarse-to-fine

strategy, which is very common in traditional computer vision approaches. In each stage, we estimated the 3D reconstruction for the stage-specific resolution and used this intermediate reconstruction to create the octree data structure for the next stage. With this technique we outperformed well-established baseline methods on challenging volumetric depth fusion tasks, i.e. only given a low number of input views and noise degraded input depth maps. We also evaluated the method on a real-world Kinect dataset, where could also show an improvement, although not as pronounced as on the 3D CAD models. In our opinion, these results could be further improved by training the networks on large datasets with better ground-truth quality. In addition to the volumetric depth fusion experiments, we obtained new state-of-the-art results on a difficult depth completion dataset, where given only one depth map as input the goal is to estimate the full 3D scene. While those performance gains can be mostly attributed to the deep learning setting in general, the proposed octree structure within the network was key to enable the high-resolution inputs and outputs that would have been otherwise infeasible to obtain on today's GPGPUs.

CHAPTER 5

Conclusion and Outlook

5.1 Conclusion

In this thesis, we presented deep learning based methods for problems in 2.5D and 3D. Deep learning based methods are nowadays the method of choice for almost all recognition tasks, not only in computer vision but also in speech recognition and natural language processing. We focused especially on domains where deep methods are still less researched mainly for two reasons: (i) Training data with accurate ground-truth labels are hard to obtain, and (ii) the computational requirements increase drastically with the input size. The first point is equally true for the covered problems in 2.5D and 3D. For depth super-resolution, we can record an almost infinitely large amount of low-resolution data with today's depth sensors. However, getting accurate high-resolution ground-truth depth is difficult. One could use an expensive structure light set-up to record scenes, but the registration of the arrangement and registration of the scans is still problematic. In 3D the training data problem is even worse, but the second point is at least as serious. If the 3D input is voxelized, the memory requirements to train a network explode with higher input resolutions. We proposed methods in this thesis that tackle exactly those problems.

5.1.1 Deep Learning for 2.5D

If we talk about 2.5D data, we most of the time refer to depth maps. Depth maps are 2D images where each pixel describes the distance to the first surface it encounters along the viewing ray. While this defines 3D points, it only describes the scene up to the first encountered surfaces, the information behind is lost in the projection. 2.5D data is a highly valuable source of data that has a wide range of applications in computer vision. The drawback, however, is that while consumer depth sensors enable depth recordings at

a low cost and with high frame rates, the output is typically of low-resolution and exhibits a certain degree of noise.

We proposed in this thesis a novel method that combines the modern deep convolutional networks for super-resolution with traditional variational methods that incorporate prior knowledge about the target domain. On intensity images, deep methods are currently state-of-the-art. However, they are mostly fueled by the sheer amount of training data that is available on the web. Unfortunately, the situation is different for depth maps. We demonstrated the use of synthetically generated depth maps to train a deep convolutional network. While not perfectly capturing the real data domain, depth data is less problematic to generate than for example color images. For the latter, one has to consider textures, shading, lighting and so on, whereas depth data is mainly characterized by affine surfaces with sharp depth discontinuities. We further made the interesting observation that others already have observed for varying computer vision tasks: The utilizing of global energy minimization method in a post-processing step further improves the depth super-resolution results. This motivated our method to train a variational method on top of the deep convolutional network. By unrolling the iteration steps of a fast convex optimization algorithm we were able to efficiently train both models end-to-end. Although our proposed network alone already yields state-of-the-art results on the common depth super-resolution benchmarks, our joint model further improved the results significantly.

This raises the natural question, why this is the case. The network alone should be able to also approximate the output of the variational model. To this end, we do not have a definitive answer, but some theories. One possibility might be our network architecture. Although we have explored a wide range of state-of-the-art super-resolution network architecture the space was still limited and one might come up with an even more sophisticated model. However, we did observe that deeper, more expressive models did not always obtain better performance. Another possibility might be the synthetic training data. Normally, we would train the network on the massive amount of synthetic training data and then fine-tune it on the real target domain. This was not possible because we did not have any training data for the target domain. Also obtaining accurate high-resolution ground-truth data for this task is non-trivial. Even with a high-resolution structured light scanner as utilized in the ToFMark benchmark one has to record the scene from various viewpoints to reduce occlusions, which is very time-consuming and limited to static scenes. Hence, the provision of prior knowledge about this domain in the variational method is an important factor for the additional performance gain.

At last, we want to note a final important observation. Namely, we never observed a drop in performance by training the joint model with respect to the metric that we optimized. This is an inherent property of the model due to the trade-off parameter in the variational model. If the data term already perfectly fits, then the training procedure is free to simply turn off the regularization term.

5.1.2 Deep Learning for 3D

Data in 3D is a highly interesting topic for its own. First, there are many possible representations, e.g. 3D point clouds, triangle meshes, implicit surfaces like truncated signed distance functions, or parametric functions. Further, the efficient storage is another intriguing facet. This is especially true for volumetric representations, where one needs high-resolution grids to preserve fine surface details. Of course, those points are also a concern for deep learning methods on 3D. The straight-forward way from a 2D classification network to a 3D classification network is by representing the input data as 3D voxel grid instead of a 2D pixel grid. In this way, operations like convolution and pooling, but also the hierarchically stacking of those operations to successful network architectures can be easily reused for tasks in 3D. However, the big drawback is the huge memory consumption of the volumetric representation for deep convolutional networks that quickly exceeds the available GPGPU memory.

In this thesis, we presented a method based on an efficient space partitioning data structure that preserves the volumetric input but decreases the memory consumption significantly. We utilized a hybrid grid-octree that can pool information in larger octree cells. Hence, instead of storing feature vectors for each voxel independently, some feature vectors are shared within larger octree cells that comprise a set of voxels. With this method, we were able to share memory and computational resources in larger octree cells, for example in empty regions of the input, while voxels near the surface are represented with smaller octree cells that capture the fine resolution. We further defined the common network operations like convolution and pooling on this hybrid grid-octree data structure such that they are equivalent to the operations on the regular voxel grid up to the pooling in the larger octree cells.

In our experiments, we first showed that our octree representation can at least handle a factor of $\times 64$ more voxels for the network input and running faster on larger resolutions. Additionally, we also demonstrated that the pooled information in larger octree cells does not degrade the network performance. It is not always obvious that high input resolutions, i.e. fine 3D shape details, are necessary for a certain task. For example on the 3D classification task higher input resolutions beyond 32^3 did not yield significantly better results. By inspecting the dataset more closely we found that most errors are due to ambiguities in categories, e.g. desk vs. table, rather than uncertainties due to discretization on a low resolution. Therefore, we also presented experiments, e.g. 3D shape orientation estimation and 3D semantic point cloud labeling, where increasing the input resolution yielded better performance and was needed to achieve state-of-the-art results.

A supposed drawback of the presented method is that the space portioning has to be known a priori. This is a valid assumption for shape classification, or semantic segmentation, where the output is either a 1D vector or has the same structure as the input. Yet, this is an issue if we want to perform depth fusion or depth completion with 3D convolutional networks. We, therefore, proposed a novel structural split module for our hybrid

grid-octree network architectures. Embedded in a coarse-to-fine network architecture that was inspired by traditional computer vision approaches, we compute intermediate reconstructions. Based on this intermediate reconstructions the structural split module was used to compute the hybrid grid-octree structure for the next stage in the coarse-to-fine architecture. Our evaluations demonstrated the effectiveness of this method on volumetric depth fusion and volumetric depth completion. For the volumetric depth fusion task, our method was able to handle very challenging settings, e.g. with a low number of input view and noise on the input depth maps, in contrast to established baseline methods. Even in the case with only one depth map as input, which is denoted as the depth completion task, our method could still predict reasonable 3D objects and scenes. While those results can be attributed to the deep learning setting in general, the proposed octree structure within the network was key to enable the high-resolution inputs and outputs.

5.2 Outlook

We proposed novel deep learning methods for 2.5D and 3D data that pushed the state-of-the-art in the respective fields. However, the problems are still far from being solved and there is still room for improvement and future work.

For depth super-resolution, one major remaining challenge is to obtain a larger quantity of high-quality and high-resolution ground-truth data. As we have seen in many other computer vision tasks, fine-tuning on the real target domain can significantly improve the performance. However, using, for example, high-end structured light scanners together with consumer depth sensors in a calibrated set-up to record training data is a very time-consuming process and is not fully automatic. But why not going the other way around and produce synthetic depth data that more closely resembles the characteristics of the target consumer depth sensor. With the current advances in generative models, i.e. variational autoencoders and generative adversarial networks, this might be a promising research direction.

For deep learning methods on 3D, there are also still a lot of open possibilities. Although our novel 3D structure for deep convolutional networks enabled significantly higher input resolutions and faster runtimes, it is still limited. It is for example not possible to fit city scale large scenes into the network. There it might be interesting to research how even more efficient space partitioning data structures, e.g. voxel hashing, could help. However, we believe that our proposed methods already enable the application of deep learning methods on additional 3D computer vision problems, like multi-view stereo.

APPENDIX A

List of Publications

My work at the Institute for Computer Graphics and Vision led to the following peer-reviewed publications. For the sake of completeness of this thesis, they are listed in chronological order along with the respective abstracts.

A.1 2017

OctNetFusion: Learning Fusion from Data

Gernot Riegler, Ali Osman Ulusoy, Horst Bischof and Andreas Geiger
In: *Proceedings of the International Conference on 3D Vision (3DV)*
October 2017, Qingdao, China
(Accepted for oral presentation)

Abstract: In this paper, we present a learning based approach to depth fusion, i.e., dense 3D reconstruction from multiple depth images. The most common approach to depth fusion is based on averaging truncated signed distance functions, which was originally proposed by Curless and Levoy in 1996. While this method is simple and provides great results, it is not able to reconstruct (partially) occluded surfaces and requires a large number frames to filter out sensor noise and outliers. Motivated by the availability of large 3D model repositories and recent advances in deep learning, we present a novel 3D CNN architecture that learns to predict an implicit surface representation from the input depth maps. Our learning based method significantly outperforms the traditional volumetric fusion approach in terms of noise reduction and outlier suppression. By learning the structure of real world 3D objects and scenes, our approach is further able to reconstruct occluded regions and to fill in gaps in the reconstruction. We demonstrate that our learning

based approach outperforms both vanilla TSDF fusion as well as TV-L1 fusion on the task of volumetric fusion. Further, we demonstrate state-of-the-art 3D shape completion results.

OctNet: Learning Deep 3D Representations at High Resolutions

Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger
In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*
July 2017, Honolulu, USA
(Accepted for oral presentation)

Abstract: We present OctNet, a representation for deep learning with sparse 3D data. In contrast to existing models, our representation enables 3D convolutional networks which are both deep and high resolution. Towards this goal, we exploit the sparsity in the input data to hierarchically partition the space using a set of unbalanced octrees where each leaf node stores a pooled feature representation. This allows to focus memory allocation and computation to the relevant dense regions and enables deeper networks without compromising resolution. We demonstrate the utility of our OctNet representation by analyzing the impact of resolution on several 3D tasks including 3D object classification, orientation estimation and point cloud labeling.

A.2 2016

ATGV-Net: Accurate Depth Super-Resolution

Gernot Riegler, Matthias Rüther, and Horst Bischof
In: *Proceedings of European Conference on Computer Vision (ECCV)*
October 2016, Amsterdam, The Netherlands
(Accepted for poster presentation)

Abstract: In this work we present a novel approach for single depth map super-resolution. Modern consumer depth sensors, especially Time-of-Flight sensors, produce dense depth measurements, but are affected by noise and have a low lateral resolution. We propose a method that combines the benefits of recent advances in machine learning based single image super-resolution, i.e. deep convolutional networks, with a variational method to recover accurate high-resolution depth maps. In particular, we integrate a variational method that models the piecewise affine structures apparent in depth data via an anisotropic total generalized variation regularization term on top of a deep network. We call our method ATGV-Net and train it end-to-end by unrolling the optimization procedure of the variational method. To train deep networks, a large corpus of training data with accurate ground-truth is required. We demonstrate that it is

feasible to train our method solely on synthetic data that we generate in large quantities for this task. Our evaluations show that we achieve state-of-the-art results on three different benchmarks, as well as on a challenging Time-of-Flight dataset, all without utilizing an additional intensity image as guidance.

A Deep Primal-Dual Network for Guided Depth Super-Resolution

Gernot Riegler, David Ferstl, Matthias Rüther, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2016, York, UK
(Accepted for oral presentation)

Abstract: In this paper we present a novel method to increase the spatial resolution of depth images. We combine a deep fully convolutional network with a non-local variational method in a deep primal-dual network. The joint network computes a noise-free, high-resolution estimate from a noisy, low-resolution input depth map. Additionally, a high-resolution intensity image is used to guide the reconstruction in the network. By unrolling the optimization steps of a first-order primal-dual algorithm and formulating it as a network, we can train our joint method end-to-end. This not only enables us to learn the weights of the fully convolutional network, but also to optimize all parameters of the variational method and its optimization procedure. The training of such a deep network requires a large dataset for supervision. Therefore, we generate high-quality depth maps and corresponding color images with a physically based renderer. In an exhaustive evaluation we show that our method outperforms the state-of-the-art on multiple benchmarks.

Efficiently Creating 3D Training Data for Fine Hand Pose Estimation

Markus Oberweger, Gernot Riegler, Paul Wohlhart, and Vincent Lepetit
In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2016, Las Vegas, USA
(Accepted for poster presentation)

Abstract: While many recent hand pose estimation methods critically rely on a training set of labelled frames, the creation of such a dataset is a challenging task that has been overlooked so far. As a result, existing datasets are limited to a few sequences and individuals, with limited accuracy, and this prevents these methods from delivering their full potential. We propose a semi-automated method for efficiently and accurately labeling each frame of a hand depth video with the corresponding 3D locations of the joints: The user is asked to provide only an estimate of the 2D reprojections of the visible joints in some reference frames, which are automatically selected to minimize the labeling work by efficiently optimizing a sub-modular loss function. We then exploit spatial, temporal,

and appearance constraints to retrieve the full 3D poses of the hand over the complete sequence. We show that this data can be used to train a recent state-of-the-art hand pose estimation method, leading to increased accuracy.

A.3 2015

Conditioned Regression Models for Non-Blind Single Image Super-Resolution

Gernot Riegler, Samuel Schulter, Matthias Rüther, and Horst Bischof
In: *Proceedings of IEEE International Conference on Computer Vision (ICCV)*
December 2015, Santiago, Chile
(Accepted for poster presentation)

Abstract: Single image super-resolution is an important task in the field of computer vision and finds many practical applications. Current state-of-the-art methods typically rely on machine learning algorithms to infer a mapping from low- to high-resolution images. These methods use a single fixed blur kernel during training and, consequently, assume the exact same kernel underlying the image formation process for all test images. However, this setting is not realistic for practical applications, because the blur is typically different for each test image. In this paper, we loosen this restrictive constraint and propose conditioned regression models (including convolutional neural networks and random forests) that can effectively exploit the additional kernel information during both, training and inference. This allows for training a single model, while previous methods need to be re-trained for every blur kernel individually to achieve good results, which we demonstrate in our evaluations. We also empirically show that the proposed conditioned regression models (i) can effectively handle scenarios where the blur kernel is different for each image and (ii) outperform related approaches trained for only a single kernel.

Anatomical landmark detection in medical applications driven by synthetic data

Gernot Riegler, Martin Urschler, Matthias Rüther, Horst Bischof, and Darko Stern
In: *Proceedings of IEEE International Conference on Computer Vision Workshop (ICCVW), TASK-CV: Transferring and Adapting Source Knowledge in Computer Vision*
December 2015, Santiago, Chile
(Accepted for poster presentation)

Abstract: An important initial step in many medical image analysis applications is the accurate detection of anatomical landmarks. Most successful methods for this task rely on data-driven machine learning algorithms. However, modern machine learning techniques,

e.g. convolutional neural networks, need a large corpus of training data, which is often an unrealistic setting for medical datasets. In this work, we investigate how to adapt synthetic image datasets from other computer vision tasks to overcome the underrepresentation of the anatomical pose and shape variations in medical image datasets. We transform both data domains to a common one in such a way that a convolutional neural network can be trained on the larger synthetic image dataset and fine-tuned on the smaller medical image dataset. Our evaluations on data of MR hand and whole body CT images demonstrate that this approach improves the detection results compared to training a convolutional neural network only on the medical data. The proposed approach may also be usable in other medical applications, where training data is scarce.

Depth Restoration via Joint Training of a Global Regression Model and CNNs

Gernot Riegler, René Ranftl, Matthias Rüther, Thomas Pock, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2015, Swansea, UK
(Accepted for poster presentation)

Abstract: Denoising and upscaling of depth maps is a fundamental post-processing step for handling the output of depth sensors, since many applications that rely on depth data require accurate estimates to reach optimal accuracy. Adapting methods for denoising and upscaling to specific types of depth sensors is a cumbersome and error-prone task due to their complex noise characteristics. In this work we propose a model for denoising and upscaling of depth maps that adapts to the characteristics of a given sensor in a data-driven manner. We introduce a non-local Global Regression Model which models the inherent smoothness of depth maps. The Global Regression Model is parametrized by a Convolutional Neural Network, which is able to extract a rich set of features from the available input data. The structure of the model enables a complex parametrization, which can be jointly learned end-to-end and eliminates the need to explicitly model the signal formation process and the noise characteristics of a given sensor. Our experiments show that the proposed approach outperforms state-of-the-art methods, is efficient to compute and can be trained in a fully automatic way.

Learning Depth Calibration of Time-of-Flight Cameras

David Ferstl, Christian Reinbacher, Gernot Riegler, Matthias Rüther, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2015, Swansea, UK
(Accepted for poster presentation)

Abstract: We present a novel method for an automatic calibration of modern consumer Timeof-Flight (ToF) cameras. Usually, these sensors come equipped with an integrated color camera. Albeit they deliver acquisitions at high frame rates they usually suffer from incorrect calibration and low accuracy due to multiple error sources. Using information from both cameras together with a simple planar target, we will show how to accurately calibrate both color and depth camera, and tackle most error sources inherent to ToF technology in a unified calibration framework. Automatic feature detection minimizes user interaction during calibration. We utilize a Random Regression Forest to optimize the manufacturer supplied depth measurements. We show the improvements to commonly used depth calibration methods in a qualitative and quantitative evaluation on multiple scenes acquired by an accurate reference system for the application of dense 3D reconstruction.

A Framework for Articulated Hand Pose Estimation and Evaluation

Gernot Riegler, David Ferstl, Matthias Rüther, and Horst Bischof
In: *Proceedings of Scandinavian Conference on Image Analysis (SCIA)*
June 2015, Copenhagen, Denmark
(Accepted for oral presentation)

Abstract: We present in this paper a framework for articulated hand pose estimation and evaluation. Within this framework we implemented recently published methods for hand segmentation and inference of hand postures. We further propose a new approach for the segmentation and extend existing convolutional network based inference methods. Additionally, we created a new dataset that consists of a synthetically generated training set and accurately annotated test sequences captured with two different consumer depth cameras. The evaluation shows that we can improve with our methods the state-of-the-art. To foster further research, we will make all sources and the complete dataset used in this work publicly available.

A.4 2014

Hough Networks for Head Pose Estimation and Facial Feature Localization

Gernot Riegler, David Ferstl, Matthias Rüther, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2014, Nottingham, UK
(Accepted for poster presentation)

Abstract: We present Hough Networks (HNs), a novel method that combines the idea of Hough Forests (HFs) with Convolutional Neural Networks (CNNs). Similar to HFs we

perform a simultaneous classification and regression on densely extracted image patches. But instead of a Random Forest we utilize a CNN which is able to learn higher-order feature representations and does not rely on any handcrafted features. Applying a CNN on a patch level has the advantage of reasoning about more image details and additionally allows to segment the image into foreground and background. Furthermore, the structure of a CNN supports efficient inference of patches extracted from a regular grid. We evaluate HNs on two computer vision tasks: head pose estimation and facial feature localization. Our method achieves at least state-of-the-art performance without sacrificing versatility which allows extension to many other applications.

CP-Census: A Novel Model for Dense Variational Scene Flow from RGB-D Data

David Ferstl, Gernot Riegler, Matthias Rüther, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2014, Nottingham, UK
(Accepted for oral presentation)

Abstract: We present a novel method for dense variational scene flow estimation based a multi-scale Ternary Census Transform in combination with a patchwise Closest Points depth data term. On the one hand, the Ternary Census Transform in the intensity data term is capable of handling illumination changes, low texture and noise. On the other hand, the patchwise Closest Points search in the depth data term increases the robustness in low structured regions. Further, we utilize higher order regularization which is weighted and directed according to the input data by an anisotropic diffusion tensor. This allows to calculate a dense and accurate flow field which supports smooth as well as non-rigid movements while preserving flow boundaries. The numerical algorithm is solved based on a primal-dual formulation and is efficiently parallelized to run at high frame rates. In an extensive qualitative and quantitative evaluation we show that this novel method for scene flow calculation outperforms existing approaches. The method is applicable to any sensor delivering dense depth and intensity data such as Microsoft Kinect or Intel Gesture Camera.

Real-time Flare Detection in Ground-Based Hα Imaging at Kanzelhöhe Observatory

Werner Pötzi, Astrid Veronig, Gernot Riegler, Ulrike Amerstorfer, Thomas Pock, Manuela Temmer, Wolfgang Polanec, and Dietmar J. Baumgartner
In: *Solar Physics*
November 2014

Abstract: Kanzelhöhe Observatory (KSO) regularly performs high-cadence full-disk imaging of the solar chromosphere in the Hα and Ca ii K spectral lines as well as in the solar photosphere in white light. In the frame of ESA's (European Space Agency) Space Situational Awareness (SSA) program, a new system for real-time Hα data provision and automatic flare detection was developed at KSO. The data and events detected are published in near real-time at ESA's SSA Space Weather portal (`http://swe.ssa.esa.int/web/guest/kso-federated`). In this article, we describe the Hα instrument, the image-recognition algorithms we developed, and the implementation into the KSO Hα observing system. We also present the evaluation results of the real-time data provision and flare detection for a period of five months. The Hα data provision worked in 99.96% of the images, with a mean time lag of four seconds between image recording and online provision. Within the given criteria for the automatic image-recognition system (at least three Hα images are needed for a positive detection), all flares with an area $\geq$50 micro-hemispheres that were located within 60° of the solar center and occurred during the KSO observing times were detected, a number of 87 events in total. The automatically determined flare importance and brightness classes were correct in $\sim$85%. The mean flare positions in heliographic longitude and latitude were correct to within $\sim$1°. The median of the absolute differences for the flare start and peak times from the automatic detections in comparison with the official NOAA (and KSO) visual flare reports were 3 min (1 min).

APPENDIX B

Proofs

B.1 Intersection of Convex Sets

If X_1 and X_2 are convex sets, then also the intersection of both $X_1 \cap X_2$ is a convex set.

Proof. Let $\mathbf{x}_1, \mathbf{x}_2 \in X_1 \cap X_2$. This implies $\mathbf{x}_1 \in X_1 \wedge \mathbf{x}_1 \in X_2$ and $\mathbf{x}_2 \in X_1 \wedge \mathbf{x}_2 \in X_2$. Therefore, $\mathbf{y} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in X_1$ and $\mathbf{y} \in X_2$ for $\lambda \in [0,1]$ by convexity of X_1 and X_2. Hence, $\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in X_1 \cap X_2$ which shows by definition of convex sets that $X_1 \cap X_2$ is convex. □

B.2 Affine Transformation of Convex Sets

If we assume that X is a convex set, then is the convexity preserved by the affine transformation $\mathbf{A}\mathbf{x} + \mathbf{b}$. $\mathbf{A}$ is a linear transformation in X and $\mathbf{x}, \mathbf{b} \in X$.

Proof. By the definition of convexity

$$\mathbf{A}(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) + \mathbf{b} \Leftrightarrow \tag{B.1}$$

$$\mathbf{A}(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) + \lambda\mathbf{b} + (1-\lambda)\mathbf{b} \Leftrightarrow \tag{B.2}$$

$$\lambda\mathbf{A}\mathbf{x}_1 + (1-\lambda)\mathbf{A}\mathbf{x}_2 + \lambda\mathbf{b} + (1-\lambda)\mathbf{b} \Leftrightarrow \tag{B.3}$$

$$\lambda(\mathbf{A}\mathbf{x}_1 + \mathbf{b}) + (1-\lambda)(\mathbf{A}\mathbf{x}_1 + \mathbf{b})\,. \tag{B.4}$$

□

B.3 Global Minimizers of Convex Functions

If f is a convex function, then each local minimum is a global minimum.

Proof. Let $\mathbf{x}^*$ be a local minimum of the convex function $f : X \to \mathbb{R}$. Hence,

$$\mathbf{x}^* \in X \wedge \exists \epsilon > 0 \wedge \forall \mathbf{x} \in [\mathbf{x}^* - \epsilon, \mathbf{x}^* + \epsilon] : f(\mathbf{x}^*) \leq f(\mathbf{x}) \,. \tag{B.5}$$

Now, suppose for the sake of the proof that there exists a vector that yields a smaller function value, i.e. $\exists \mathbf{y} \in X \wedge \mathbf{y} \neq \mathbf{x}$ for which

$$f(\mathbf{z}) < f(\mathbf{x}^*) \,. \tag{B.6}$$

By the convexity of f we can rewrite the inequality as

$$f(\lambda \mathbf{x}^* + (1 - \lambda)\mathbf{z}) \leq \lambda f(\mathbf{x}^*) + (1 - \lambda) f(\mathbf{z}) \,. \tag{B.7}$$

If we plug in the assumption in the right side of the inequality we can write it as

$$\lambda f(\mathbf{x}^*) + (1 - \lambda) f(\mathbf{z}) < \lambda f(\mathbf{x}^*) + (1 - \lambda) f(\mathbf{x}^*) = f(\mathbf{x}^*) \,. \tag{B.8}$$

However, as $\lambda \to 1$ this contradicts that $\mathbf{x}$ is a local minimizer and hence, proofs that every local minimizer of a convex function is a global minimizer. □

B.4 Convex Functions are Global Under-Estimators

If f is a convex function, then the following relation is satisfied

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \,. \tag{B.9}$$

Proof. Let f be a convex function, $\mathbf{x}, \mathbf{y} \in \operatorname{dom} f$ and $\lambda \in [0, 1]$. Then, given the definition of convex functions we have

$$f(\lambda \mathbf{y} + (1 - \lambda)\mathbf{x}) \leq \lambda f(\mathbf{y}) + (1 - \lambda) f(\mathbf{x}) \,. \tag{B.10}$$

We can rearrange the definition as follows

$$f(\lambda(\mathbf{y} - \mathbf{x}) + \mathbf{x}) \leq \lambda(f(\mathbf{y}) - f(\mathbf{x})) + f(\mathbf{x}) \tag{B.11}$$

$$f(\mathbf{y}) - f(\mathbf{x}) \geq \frac{f(\lambda(\mathbf{y} - \mathbf{x}) + \mathbf{x}) - f(\mathbf{x})}{\lambda} \tag{B.12}$$

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \frac{f(\lambda(\mathbf{y}-\mathbf{x})+\mathbf{x}) - f(\mathbf{x})}{\lambda} \,. \tag{B.13}$$

Using the definition of the differential operator (Definition 2.6) and taking the limes $\lim_{\lambda\to\infty}$, we get the desired relation. □

B.5 Gradients of Convex Functions are Monotone

Given a convex function f, the gradient of this function is monotone, i.e. for scalar functions is non-decreasing.

Proof. Given the definition of convex functions for differentiable functions

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y}-\mathbf{x} \rangle \,, \tag{B.14}$$

we can exchange $\mathbf{x}$ and $\mathbf{y}$ as we did not assume any special ordering in the definition

$$f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x}-\mathbf{y} \rangle \,. \tag{B.15}$$

If we now combine booth equations we get the desired result

$$f(\mathbf{y}) + f(\mathbf{x}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y}-\mathbf{x} \rangle + f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x}-\mathbf{y} \rangle \tag{B.16}$$

$$0 \geq \langle \nabla f(\mathbf{x}), \mathbf{y}-\mathbf{x} \rangle + \langle \nabla f(\mathbf{y}), \mathbf{x}-\mathbf{y} \rangle \tag{B.17}$$

$$0 \leq \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x}-\mathbf{y} \rangle \,. \tag{B.18}$$

□

B.6 Optimality Condition of Convex Functions

For a convex function f the condition $f(\mathbf{x}^*) = 0$ is a necessary and sufficient condition for $\mathbf{x}$ to be a global optimizer.

Proof. By plugging $\nabla f(\mathbf{x}^*) = 0$ into Equation (B.9) we get

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)(\mathbf{x}-\mathbf{x}^*) \tag{B.19}$$

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) \,, \tag{B.20}$$

which proofs that $\mathbf{x}^*$ is a global minimizer. □

B.7 Weighted Sum Preserves Convexity

If f_1, and f_2 are convex functions and $w_1, w_2 > 0$, then $f = w_1 f + w_2 f$ is also convex.

Proof. Using the definition of a convex function from Definition 2.9 we deduce

$$w_1 f_1(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) + w_2 f_2(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \tag{B.21}$$
$$w_1(\lambda f_1(\mathbf{x}_1) + (1-\lambda) f_1(\mathbf{x}_2)) + w_2(\lambda f_2(\mathbf{x}_1) + (1-\lambda) f_2(\mathbf{x}_2)) = \tag{B.22}$$
$$\lambda(w_1 f_1(\mathbf{x}_1) + w_2 f_2(\mathbf{x}_1)) + (1-\lambda)(w_1 f_1(\mathbf{x}_2) + w_2 f_2(\mathbf{x}_2))\,. \tag{B.23}$$

□

B.8 Point-wise Maximum Preserves Convexity

If f_1 and f_2 are convex, then $f(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$ is also a convex function.

Proof. Using the definition of convex function from Definition 2.9 we deduce

$$\max\{f_1(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2), f_2(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)\} \leq \tag{B.24}$$
$$\max\{\lambda f_1(\mathbf{x}_1) + (1-\lambda) f_1(\mathbf{x}_2), \lambda f_2(\mathbf{x}_1) + (1-\lambda) f_2(\mathbf{x}_2)\} \leq \tag{B.25}$$
$$\lambda \max\{f_1(\mathbf{x}_1) + f_2(\mathbf{x}_1))\} + (1-\lambda)\max\{f_1(\mathbf{x}_2) + f_2(\mathbf{x}_2))\}\,. \tag{B.26}$$

□

B.9 Conjugate of a Norm

The convex conjugate of a norm $f(\mathbf{x}) = \|\mathbf{x}\|$ is given by the indicator function of the dual norm unit ball

$$f^*(\mathbf{y}) = \begin{cases} 0 & \text{if } \|\mathbf{y}\|_* \leq 1 \\ +\infty & \text{else} \end{cases}\,. \tag{B.27}$$

Proof. We distinguish two cases. If $\|\mathbf{y}\|_* \leq 1$, then from Definition 2.5 of the dual norm it follows that $\langle \mathbf{y}, \mathbf{x} \rangle \leq \|\mathbf{y}\|_* \|\mathbf{x}\|$ for all $\mathbf{x}$. As $\|\mathbf{y}\|_* \leq 1$, it follows

$$\langle \mathbf{y}, \mathbf{x} \rangle - \|\mathbf{x}\| \leq 0\,, \tag{B.28}$$

for all $\mathbf{x}$, and $\mathbf{x} = 0$ is the value that maximizes $\langle \mathbf{y}, \mathbf{x} \rangle - f(\mathbf{x})$ with maximum value 0. For the second case of $\|\mathbf{y}\|_* > 1$, there exists a $\mathbf{z}$ with $\|\mathbf{x}\| \leq 1$ and $\langle \mathbf{y}, \mathbf{x} \rangle > 1$ by definition of the dual norm. If we choose $\mathbf{x} = \lambda \mathbf{z}$ and letting $\lambda \to \infty$ we get

$$\langle \mathbf{y}, \mathbf{x} \rangle - f(\mathbf{x}) = \lambda(\langle \mathbf{y}, \mathbf{z} \rangle - \|\mathbf{z}\|) \to \infty\,. \tag{B.29}$$

□

B.10 Minimizer of the Proximal Mapping

The point $\mathbf{x}^*$ is the minimizer of the function f, iff $\mathbf{x}$ is a fixed point of prox_f

$$\mathbf{x}^* = \text{prox}_f(\mathbf{x}^*)\,. \tag{B.30}$$

Proof. Let us assume that $\mathbf{x}^*$ minimizes $f(\mathbf{x})$. Then, the following inequality holds for all $\mathbf{x}$, assuming $\lambda = 1$ without loss of generality

$$f(\mathbf{x}) + \frac{1}{2}\left\|\mathbf{x} - \mathbf{x}^*\right\|_2^2 \geq f(\mathbf{x}^*) = f(\mathbf{x}^*) + \frac{1}{2}\left\|\mathbf{x}^* - \mathbf{x}^*\right\|_2^2\,. \tag{B.31}$$

Hence, $\mathbf{x}^*$ minimizes $f(\mathbf{x}) + \frac{1}{2}\left\|\mathbf{x} - \mathbf{x}^*\right\|_2^2$ and therefore, $\mathbf{x}^* = \text{prox}_f(\mathbf{x}^*)$. Conversely, using the definition of the subdifferential from Definition 2.12 and defining $\bar{\mathbf{x}}$ as

$$\bar{\mathbf{x}} = \text{prox}_f(\mathbf{y}) = \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}) + \frac{1}{2}\left\|\mathbf{x} - \mathbf{y}\right\|_2^2 \right\}\,. \tag{B.32}$$

Then, $\bar{\mathbf{x}}$ minimizes $\text{prox}_f(\mathbf{y})$, iff $0 \in \partial f(\bar{\mathbf{x}}) + \bar{\mathbf{x}} - \mathbf{y}$. By setting $\bar{\mathbf{x}} = \mathbf{y} = \mathbf{x}^*$ we derive the desired result $0 \in \partial f(\mathbf{x}^*)$, hence, $\mathbf{x}^*$ minimizes f. □

B.11 Moreau Decomposition

The proximal mapping is related to the convex conjugate by the Moreau decomposition that states

$$\mathbf{x} = \text{prox}_f(\mathbf{x}) + \text{prox}_f^*(\mathbf{x})\,. \tag{B.33}$$

Proof. Let $\mathbf{y} = \text{prox}_f(\mathbf{x})$, then from the optimality condition we have

$$0 \in \partial f(\mathbf{y}) + \mathbf{y} - \mathbf{x} \tag{B.34}$$

$$\mathbf{x} - \mathbf{y} \in \partial f(\mathbf{y})\,. \tag{B.35}$$

Using $\mathbf{x} - \mathbf{y} \in \partial f(\mathbf{y}) \Leftrightarrow \mathbf{y} \in \partial f^*(\mathbf{x} - \mathbf{y})$ and the proximal map definition from Equation (2.60) we can derive

$$\mathbf{y} \in \partial f^*(\mathbf{x} - \mathbf{y}) \tag{B.36}$$

$$\mathbf{x} - \mathbf{y} \in \mathbf{x} - \partial f^*(\mathbf{x} - \mathbf{y}) \tag{B.37}$$

$$\mathbf{x} \in \mathbf{x} - \mathbf{y} + \partial f^*(\mathbf{x} - \mathbf{y}) \tag{B.38}$$

$$\mathbf{x} \in (\mathbf{I} + \partial f^*)(\mathbf{x} - \mathbf{y}) \tag{B.39}$$

$$\mathbf{x} - \mathbf{y} = (\mathbf{I} + \partial f^*)^{-1}(\mathbf{x}) = \text{prox}_f^*(\mathbf{x})\,. \tag{B.40}$$

Plugging the last equation back into $\mathbf{y} = \text{prox}_f(\mathbf{x})$ we get the Moreau decomposition. □

Bibliography

[1] M. Aharon, M. Elad, and A. Bruckstein. "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation". In: *IEEE Transactions on Signal Processing (TSP)* 54.11 (2006), 4311–4322 (cit. on p. 57).

[2] S. Almansa-Valverde, J. C. Castillo, and A. Fernández-Caballero. "Mobile robot map building from time-of-flight camera". In: *Expert Systems with Applications* 39.10 (2012), pp. 8835–8843 (cit. on p. 54).

[3] N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". In: *The American Statistician* 46.3 (1992), pp. 175–185 (cit. on p. 37).

[4] N. S. Alvar, M. Zolfaghari, and T. Brox. "Orientation-boosted Voxel Nets for 3D Object Recognition". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2016 (cit. on pp. 107, 127).

[5] O. M. Aodha, N. D. Campbell, A. Nair, and G. J. Brostow. "Patch Based Synthesis for Single Depth Image Super-Resolution". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2012 (cit. on pp. 55, 59, 60, 80–84).

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* PP.99 (2017) (cit. on pp. 2, 103, 117, 120, 133, 136).

[7] T. Baltrušaitis, P. Robinson, and L.-P. Morency. "Continuous Conditional Neural Fields for Structured Regression". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on p. 60).

[8] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012 (cit. on pp. 31, 37).

[9] J. Barron and B. Poole. "The Fast Bilateral Solver". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 59, 85, 86, 100).

[10] A. Beck and M. Teboulle. “A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems”. In: *Journal of Imaging Sciences (SIAM)* 2.1 (2009), pp. 183–202 (cit. on p. 23).

[11] J. R. Bellegarda and C. Monz. “State of the art in statistical methods for language and speech processing”. In: *Computer Speech & Language* 35 (2016), pp. 163–184 (cit. on pp. 1, 37).

[12] J. L. Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Communications of the ACM* 18.9 (1975), pp. 509–517 (cit. on p. 110).

[13] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on pp. 31, 37, 48, 49).

[14] M. Blaha, C. Vogel, A. Richard, J. D. Wegner, T. Pock, and K. Schindler. “Large-Scale Semantic 3D Reconstruction: an Adaptive Multi-Resolution Model for Multi-Class Volumetric Labeling”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 111).

[15] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. “Learnability and the Vapnik-Chervonenkis Dimension”. In: *Journal of the ACM (JACM)* 36.4 (1989), pp. 929–965 (cit. on p. 34).

[16] N. K. Bose and N. A. Ahuja. “Superresolution and Noise filtering Using Moving Least Squares”. In: *IEEE Transactions on Image Processing (TIP)* 15.8 (2006), pp. 2239–2248 (cit. on pp. 85, 93–96, 101).

[17] S. Boyd and L. Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004 (cit. on p. 13).

[18] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. “Learning 6D Object Pose Estimation using 3D Object Coordinates”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on p. 128).

[19] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, and C. Rother. “Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 128).

[20] K. Bredies, K. Kunisch, and T. Pock. “Total Generalized Variation”. In: *Journal of Imaging Sciences (SIAM)* 3.3 (2010), 492–526 (cit. on pp. 28, 59, 66).

[21] L. Breiman. “Bagging Predictors”. In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on p. 50).

[22] L. Breiman. *Classification and regression trees*. Chapman & Hall/CRC, 1984 (cit. on p. 37).

[23] L. Breiman. “Random Forests”. In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on pp. 37, 50).

[24] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. "Generative and Discriminative Voxel Modeling with Convolutional Neural Networks". In: 2016 (cit. on pp. 108, 127, 147).

[25] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. "Spectral Networks and Locally Connected Networks on Graphs". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2014 (cit. on p. 109).

[26] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. "A Naturalistic Open Source Movie for Optical Flow Evaluation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2012 (cit. on pp. 55, 60).

[27] A. Chambolle and T. Pock. "A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging". In: *Journal of Mathematical Imaging and Vision (JMIV)* 40.1 (2011), 120–145 (cit. on p. 24).

[28] A. Chambolle and T. Pock. "On the ergodic convergence rates of a first-order primal-dual algorithm". In: *Mathematic Programming* 159.1 (2015), 1–35 (cit. on p. 61).

[29] D. Chan, H. Buisman, C. Theobalt, and S. Thrun. "A Noise-aware Filter for Real-time Depth Upsampling". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2008 (cit. on pp. 84, 86–92, 100).

[30] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. "ShapeNet: An Information-Rich 3D Model Repository". In: *arXiv preprint arXiv:1512.03012* 1512.03012 (2015) (cit. on p. 104).

[31] H. Chang, D.-Y. Yeung, and Y. Xiong. "Super-Resolution Through Neighbor Embedding". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2004 (cit. on p. 57).

[32] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. "Two Deterministic Half-Quadratic Regularization Algorithms for Computed Imaging". In: *Proceedings IEEE International Conference on Image Processing (ICIP)*. 1994 (cit. on p. 64).

[33] J. Chen, D. Bautembach, and S. Izadi. "Scalable Real-Time Volumetric Surface Reconstruction". In: *ACM Transactions on Graphics (SIGGRAPH)* 32.4 (2013), p. 113 (cit. on p. 110).

[34] L.-C. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille. "Semantic Image Segmentation with Task-Specific Edge Detection Using CNNs and a Discriminatively Trained Domain Transform". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 2, 60, 103, 117).

[35] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 2, 103, 117).

[36] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. "Learning Deep Structured Models". In: *Proceedings of the International Conference on Machine learning (ICML)*. 2015 (cit. on p. 60).

[37] D. Chicco, P. Sadowski, and P. Baldi. "Deep Autoencoder Neural Networks for Gene Ontology Annotation Predictions". In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. 2014, pp. 533–540 (cit. on p. 37).

[38] S. Choi, Q. Zhou, and V. Koltun. "Robust Reconstruction of Indoor Scenes". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 105, 111).

[39] S. Choi, Q. Zhou, S. Miller, and V. Koltun. "A Large Dataset of Object Scans". In: *arXiv preprint arXiv:1602.02481* 1602.02481 (2016) (cit. on pp. 104, 147).

[40] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. "The Loss Surfaces of Multilayer Networks". In: *Conference on Artificial Intelligence and Statistics (AISTATS)*. 2015 (cit. on p. 39).

[41] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 4, 104, 112, 147).

[42] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2016 (cit. on pp. 120, 133).

[43] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. "Describing Textures in the Wild". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (cit. on p. 69).

[44] C. Cortes and V. Vapnik. "Support-Vector Networks". In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 37).

[45] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *ACM Transactions on Graphics (SIGGRAPH)*. 1996 (cit. on pp. 58, 105, 106, 110, 111, 138–140, 142–146, 148–153).

[46] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989), pp. 303–314 (cit. on p. 44).

[47] A. Dai, C. R. Qi, and M. Nießner. "Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 112).

[48] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 2933–2941 (cit. on p. 39).

[49] M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016 (cit. on p. 109).

[50] S. Dempe. "Annotated Bibliography on Bilevel Programming and Mathematical Programs with Equilibrium Constraints". In: *Optimization* 23.3 (2003), pp. 333–359 (cit. on p. 63).

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009 (cit. on p. 2).

[52] J. Diebel and S. Thrun. "An Application of Markov Random Fields to Range Sensing". In: *Advances in Neural Information Processing Systems (NIPS)*. 2005 (cit. on pp. 4, 55, 59, 84, 86–92, 100).

[53] J. Domke. "Generic Methods for Optimization-Based Modeling". In: *Conference on Artificial Intelligence and Statistics (AISTATS)*. 2012 (cit. on p. 61).

[54] J. Domke. "Learning Graphical Model Parameters with Approximate Marginal Inference". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 35.10 (2013), pp. 2454–2467 (cit. on p. 61).

[55] C. Dong, C. C. Loy, K. He, and X. Tang. "Learning a Deep Convolutional Network for Image Super-Resolution". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on pp. 4, 55, 56, 58, 72).

[56] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. "Flownet: Learning Optical Flow with Convolutional Networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 136).

[57] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research (JMLR)* 12.Jul (2011), pp. 2121–2159 (cit. on p. 41).

[58] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Vol. 2. Wiley, 2000 (cit. on pp. 31, 37).

[59] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. "Convolutional Networks on Graphs for Learning Molecular Fingerprints". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015 (cit. on p. 109).

[60] D. Eigen and R. Fergus. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 112).

[61] D. Eigen, C. Puhrsch, and R. Fergus. "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014 (cit. on p. 112).

[62] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. "Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks". In: *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*. 2017 (cit. on p. 109).

[63] H. Fan, H. Su, and L. J. Guibas. "A Point Set Generation Network for 3D Object Reconstruction From a Single Image". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 112).

[64] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. J. V. Gool. "Random Forests for Real Time 3D Face Analysis". In: *International Journal of Computer Vision (IJCV)* 101.3 (2013), pp. 437–458 (cit. on pp. 2, 128, 131).

[65] G. Fanelli, T. Weise, J. Gall, and L. J. V. Gool. "Real Time Head Pose Estimation from Consumer Depth Cameras". In: *Proceedings of the German Conference on Pattern Recognition (GCPR)*. 2011, pp. 101–110 (cit. on pp. 128, 131).

[66] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar. "Fast and Robust Multiframe Super Resolution". In: *IEEE Transactions on Image Processing (TIP)* 13.10 (2004), pp. 1327–1344 (cit. on p. 56).

[67] D. Ferstl, C. Reinbacher, R. Ranftl, M. Rüther, and H. Bischof. "Image Guided Depth Upsampling using Anisotropic Total Generalized Variation". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013 (cit. on pp. 4, 59, 85–96, 99–101).

[68] D. Ferstl, C. Reinbacher, G. Riegler, M. Rüther, and H. Bischof. "Learning Depth Calibration of Time-of-Flight Cameras". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2015.

[69] D. Ferstl, G. Riegler, M. Rüther, and H. Bischof. "CP-Census: A Novel Model for Dense Variational Scene Flow from RGB-D Data". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2014.

[70] D. Ferstl, M. Rüther, and H. Bischof. "Variational Depth Superresolution using Example-Based Edge Representations". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 55, 59, 60, 80–84).

[71] M. Firman, O. Mac Aodha, S. Julier, and G. J. Brostow. "Structured Prediction of Unobserved Voxels From a Single Depth Image". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 106, 111, 153–156).

[72] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. "DeepStereo: Learning to Predict New Views from the World's Imagery". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 112).

[73] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. "Learning Low-Level Vision". In: *International Journal of Computer Vision (IJCV)* 40.1 (2000), pp. 25–47 (cit. on pp. 55, 57, 60).

[74] K. Fukushima. "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition". In: *Neural Networks* 1.2 (1988), pp. 119–130 (cit. on p. 1).

[75] R. Gadde, V. Jampani, R. Marlet, and P. V. Gehler. "Efficient 2D and 3D Facade Segmentation using Auto-Context". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* PP.99 (2017) (cit. on p. 135).

[76] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. "Virtual Worlds as Proxy for Multi-Object Tracking Analysis". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 61).

[77] R. Garg, G. Carneiro, and I. Reid. "Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 112).

[78] C. F. Gauss. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. 1809 (cit. on p. 37).

[79] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* 32.11 (2013) (cit. on p. 61).

[80] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on p. 61).

[81] G. Ghiasi and C. C. Fowlkes. "Laplacian Pyramid Reconstruction and Refinement for Semantic Segmentation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 2, 103, 117).

[82] J. J. Gibson. "The Perception of the Visual World". In: (1950) (cit. on p. 1).

[83] G. Gilboa and S. Osher. "Nonlocal Operators with Applications to Image Processing". In: *Multiscale Modeling and Simulation* 7.3 (2009), 1005–1028 (cit. on p. 67).

[84] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. "Learning a Predictable and Generative Vector Representation for Objects". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 107, 147).

[85] R. Girshick. "Fast R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 2, 103).

[86] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (cit. on pp. 2, 103).

[87] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. W. Fitzgibbon. "Efficient Regression of General-Activity Human Poses from Depth Images". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2011 (cit. on pp. 2, 54).

[88] D. Glasner, S. Bagon, and M. Irani. "Super-Resolution from Single Image". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2009 (cit. on pp. 57, 60).

[89] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. 2010, pp. 249–256 (cit. on p. 42).

[90] C. Godard, O. M. Aodha, and G. J. Brostow. "Unsupervised Monocular Depth Estimation with Left-Right Consistency". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 112).

[91] G. Goh. "Why Momentum Really Works". In: *Distill* 2.4 (2017) (cit. on p. 40).

[92] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016 (cit. on p. 31).

[93] B. Graham. "Sparse 3D convolutional neural networks". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2015 (cit. on p. 109).

[94] B. Graham. "Spatially-sparse convolutional neural networks". In: *arXiv preprint arXiv:1409.6070* (2014) (cit. on p. 109).

[95] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. "Learning Rich Features from RGB-D Images for Object Detection and Segmentation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on p. 2).

[96] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. "Understanding Real World Indoor Scenes with Synthetic Data". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 61).

[97] C. Häne, S. Tulsiani, and J. Malik. "Hierarchical Surface Prediction for 3D Object Reconstruction". In: *arXiv preprint arXiv:1704.00710* (2017) (cit. on p. 112).

[98] C. Häne, C. Zach, A. Cohen, R. Angst, and M. Pollefeys. "Joint 3D Scene Reconstruction and Class Segmentation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013 (cit. on p. 111).

[99] M. Hansard, S. Lee, O. Choi, and R. P. Horaud. *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer Science & Business Media, 2012 (cit. on p. 54).

[100] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2003 (cit. on p. 2).

[101] K. He, J. Sun, and X. Tang. "Guided Image Filtering". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2010 (cit. on pp. 84, 86–92, 100).

[102] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 2, 73, 103, 104, 113, 117, 123).

[103] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 42, 142).

[104] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, 2005 (cit. on p. 1).

[105] G. E. Hinton, S. Nitish, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.058* (2012) (cit. on pp. 1, 37, 50).

[106] M. Hornáček, C. Rhemann, M. Gelautz, and C. Rother. "Depth Super Resolution by Rigid Body Self-Similarity in 3D". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013 (cit. on pp. 55, 59, 60, 80–84).

[107] A. Hornung and L. Kobbelt. "Robust Reconstruction of Watertight 3D Models from Non-Uniformly Sampled Point Clouds Without Normal Information". In: *Eurographics Symposium on Geometry Processing (SGP)*. 2006 (cit. on p. 111).

[108] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. "SceneNN: A Scene Meshes Dataset with aNNotations". In: *Proceedings of the International Conference on 3D Vision (3DV)*. 2016 (cit. on p. 61).

[109] J.-B. Huang, A. Singh, and N. Ahuja. "Single Image Super-resolution from Transformed Self-Exemplars". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 57).

[110] J. Huang and S. You. "Point Cloud Labeling using 3D Convolutional Neural Network". In: *Proceedings of the International Conference on Pattern Recognition (ICPR)*. 2016 (cit. on p. 108).

[111] Q. Huang, H. Wang, and V. Koltun. "Single-View Reconstruction via Joint Analysis of Image and Shape Collections". In: *ACM Transactions on Graphics (SIGGRAPH)*. 2015 (cit. on p. 104).

[112] D. H. Hubel and T. N. Wiesel. "Receptive Fields and Functional Architecture of Monkey Striate Cortex". In: *The Journal of Physiology* 195.1 (1968), pp. 215–243 (cit. on p. 45).

[113] D. H. Hubel and T. N. Wiesel. "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex". In: *The Journal of Physiology* 160.1 (1962), pp. 106–154 (cit. on p. 45).

[114] D. H. Hubel and T. N. Wiesel. "Receptive Fields of Single Neurones in the Cat's Striate Cortex". In: *The Journal of Physiology* 148.3 (1959), pp. 574–591 (cit. on p. 45).

[115] P. J. Huber. "Robust Regression: Asymptotics, Conjectures and Monte Carlo". In: *Annal. of Stat.* 1.5 (1973), pp. 799–821 (cit. on p. 10).

[116] T.-W. Hui, C. C. Loy, and X. Tang. "Depth Map Super-Resolution by Deep Multi-Scale Guidance". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 55, 59, 60, 73, 80–84).

[117] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the International Conference on Machine learning (ICML)*. 2015 (cit. on p. 2).

[118] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *ACM Symposium on User Interface Software and Technology*. 2011 (cit. on pp. 59, 60).

[119] W. Jakob. *Mitsuba Renderer*. 2010 (cit. on pp. 56, 69, 70).

[120] V. Jampani, M. Kiefel, and P. V. Gehler. "Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 109).

[121] V. Kämpe, E. Sintorn, and U. Assarsson. "High Resolution Sparse Voxel DAGs". In: *ACM Transactions on Graphics (SIGGRAPH)* 32.4 (2013), p. 101 (cit. on p. 110).

[122] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. "Category-Specific Object Reconstruction from a Single Image". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 112).

[123] K. Kawaguchi. "Deep Learning without Poor Local Minima". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 586–594 (cit. on p. 39).

[124] W. Kehl, T. Holl, F. Tombari, S. Ilic, and N. Navab. "An Octree-Based Approach towards Efficient Variational Range Data Fusion". In: 2016 (cit. on p. 114).

[125] B.-s. Kim, P. Kohli, and S. Savarese. "3D Scene Understanding by Voxel-CRF". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 1425–1432 (cit. on p. 111).

[126] J. Kim, J. K. Lee, and K. M. Lee. "Accurate Image Super-Resolution Using Very Deep Convolutional Networks". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 4, 56, 58, 73).

[127] J. Kim, J. K. Lee, and K. M. Lee. "Deeply-Recursive Convolutional Network for Image Super-Resolution". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 58).

[128] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2014 (cit. on pp. 41, 72, 123, 135, 142).

[129] R. Klokov and V. Lempitsky. "Escape From Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017 (cit. on p. 110).

[130] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. "Joint Bilateral Upsampling". In: *ACM Transactions on Graphics (SIGGRAPH)* 26.3 (2007) (cit. on p. 59).

[131] P. Krähenbühl and V. Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Advances in Neural Information Processing Systems (NIPS)*. 2012 (cit. on p. 60).

[132] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2012 (cit. on pp. 1, 2, 37, 50, 103, 117).

[133] K. Kunisch and T. Pock. "A Bilevel Optimization Approach for Parameter Learning in Variational Models". In: *Journal of Imaging Sciences (SIAM)* 6.2 (2013), 938–983 (cit. on p. 65).

[134] H. Kwon, Y.-W. Tai, and S. Lin. "Data-Driven Depth Map Refinement via Multi-scale Spare Representations". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 59, 60).

[135] S. Laine and T. Karras. "Efficient Sparse Voxel Octrees". In: *IEEE Transactions on Visualization and Computer Graphics (VCG)* 17.8 (2011), pp. 1048–1059 (cit. on pp. 110, 114).

[136] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural computation* 1.4 (1989), pp. 541–551 (cit. on p. 45).

[137] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), 2278–2324 (cit. on pp. 42, 45).

[138] S. Lefebvre and H. Hoppe. "Perfect Spatial Hashing". In: *ACM Transactions on Graphics (SIGGRAPH)*. Vol. 25. 3. 2006, pp. 579–588 (cit. on p. 110).

[139] A. M. Legendre. *Nouvelles Méthodes Pour la Détermination des Orbites des Comètes*. 1805 (cit. on p. 37).

[140] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. "Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function". In: *Neural Networks* 6.6 (1993), pp. 861–867 (cit. on p. 44).

[141] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. "FPNN: Field Probing Neural Networks for 3D Data". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016 (cit. on p. 109).

[142] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. "Gated Graph Sequence Neural Networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016 (cit. on p. 109).

[143] S. Linnainmaa. "Taylor Expansion of the Accumulated Rounding Error". In: *BIT Numerical Mathematics* 16.2 (1976), pp. 146–160 (cit. on pp. 44, 61, 66).

[144] S. Linnainmaa. "The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors". In: *Master's Thesis (in Finnish), University Helsinki* (1970) (cit. on pp. 44, 61, 66).

[145] D. C. Liu and J. Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528 (cit. on p. 41).

[146] S. Liu and D. B. Cooper. "Statistical Inverse Ray Tracing for Image-based 3D Modeling". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 36.10 (2014), pp. 2074–2088 (cit. on p. 111).

[147] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. "SSD: Single Shot Multibox Detector". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on pp. 2, 103).

[148] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 2, 103, 117).

[149] W. E. Lorensen and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ACM Transactions on Graphics (SIGGRAPH)*. 1987 (cit. on p. 105).

[150] J. Lu and D. Forsyth. "Sparse Depth Super-Resolution". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 84, 86, 100).

[151] J. Lu, K. Shi, D. Min, L. Lin, and M. N. Do. "Cross-Based Local Multipoint Filtering". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 430–437 (cit. on pp. 85, 93–96, 101).

[152] X. Mao, C. Shen, and Y.-B. Yang. "Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016 (cit. on pp. 4, 56, 58, 73).

[153] A. Martinović, J. Knopp, H. Riemenschneider, and L. Van Gool. "3D All The Way: Semantic Segmentation of Urban Scenes from Start to End in 3D". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 135).

[154] D. Maturana and S. Scherer. "VoxNet: A 3D Convolutional Neural Network for real-time object recognition". In: *Proceedings IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2015 (cit. on pp. 4, 104, 107).

[155] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. "SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth". In: *arXiv preprint arXiv:1612.05079* (2016) (cit. on p. 61).

[156] W. S. McCulloch and W. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 1).

[157] D. Meagher. "Geometric Modeling Using Octree Encoding". In: *Computer Graphics and Image Processing (CGIP)* 19.1 (1982), p. 85 (cit. on pp. 110, 114).

[158] A. Miller, V. Jain, and J. L. Mundy. "Real-time Rendering and Dynamic Updating of 3-d Volumetric Data". In: *Proceedings of the Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)*. 2011 (cit. on pp. 104, 110, 114, 115).

[159] T. M. Mitchell. *Machine Learning*. 1997 (cit. on pp. 31, 37).

[160] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. "Geometric deep learning on graphs and manifolds using mixture model CNNs". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 109).

[161] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012 (cit. on pp. 31, 37).

[162] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *Proceedings of the International Conference on Machine learning (ICML)*. 2010 (cit. on pp. 2, 38, 138).

[163] Y. Nesterov. “A Method of Solving a Convex Programming Problem with Convergence Rate O(1/sqr(k))”. In: *Soviet Mathematics Doklady* 27.2 (1983), 372–376 (cit. on pp. 22, 40).

[164] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Vol. 87. Springer Science & Business Media, 2013 (cit. on pp. 13, 21, 22).

[165] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. “KinectFusion: Real-time Dense Surface Mapping and Tracking”. In: *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 2011 (cit. on pp. 2, 104–106, 110, 139, 153).

[166] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. “Real-time 3D Reconstruction at Scale using Voxel Hashing”. In: *ACM Transactions on Graphics (SIGGRAPH)*. 2013 (cit. on pp. 105, 110, 111).

[167] M. Oberweger, G. Riegler, P. Wohlhart, and V. Lepetit. “Efficiently Creating 3D Training Data for Fine Hand Pose Estimation”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 2, 54).

[168] M. Oberweger, P. Wohlhart, and V. Lepetit. “Hands Deep in Deep Learning for Hand Pose Estimation”. In: *Proceedings of the Computer Vision Winter Workshop (CVWW)*. 2015 (cit. on pp. 2, 54).

[169] P. Ochs, R. Ranftl, T. Brox, and T. Pock. “Bilievel Optimization with Nonsmooth Lower Level Problems”. In: *Proceedings of the International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*. 2015 (cit. on p. 61).

[170] P. Ochs, R. Ranftl, T. Brox, and T. Pock. “Techniques for Gradient-Based Bilevel Optimization with Non-smooth Lower Level Problems”. In: *Journal of Mathematical Imaging and Vision (JMIV)* 56.2 (2016), pp. 175–194 (cit. on p. 61).

[171] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1982 (cit. on p. 39).

[172] J. Park, H. Kim, Y.-W. Tai, M. S. Brown, and I.-S. Kweon. “High Quality Depth Map Upsampling for 3D-TOF Cameras”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2011 (cit. on pp. 55, 59, 71, 84, 86–92, 97, 100).

[173] T. Pock, L. Zebedin, and H. Bischof. “TGV-Fusion”. In: *Rainbow of Computer Science*. 2011 (cit. on pp. 111, 112).

[174] B. T. Polyak. “Some Methods of Speeding Up the Convergence of Iteration Methods”. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17 (cit. on p. 40).

[175] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "PointNet: Deep learning on Point Sets for 3D Classification and Segmentation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 109).

[176] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. "Volumetric and Multi-View CNNs for Object Classification on 3D Data". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 4, 104, 107, 127).

[177] R. Ranftl and T. Pock. "A Deep Variational Model for Image Segmentation". In: *Proceedings of the German Conference on Pattern Recognition (GCPR)*. 2014 (cit. on pp. 61, 62).

[178] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 2, 103).

[179] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on pp. 2, 103).

[180] S. Ren, K. He, R. B. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015 (cit. on pp. 2, 103).

[181] G. Riegler, D. Ferstl, M. Rüther, and H. Bischof. "A Deep Primal-Dual Network for Guided Depth Super-Resolution". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2016 (cit. on p. 6).

[182] G. Riegler, D. Ferstl, M. Rüther, and H. Bischof. "A Framework for Articulated Hand Pose Estimation and Evaluation". In: *Scandinavian Conference on Image Analysis (SCIA)*. 2015 (cit. on pp. 2, 54).

[183] G. Riegler, D. Ferstl, M. Rüther, and H. Bischof. "Hough Networks for Head Pose Estimation and Facial Feature Localization". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2014 (cit. on pp. 2, 131).

[184] G. Riegler, R. Ranftl, M. Rüther, T. Pock, and H. Bischof. "Depth Restoration via Joint Training of a Global Regression Model and CNNs". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2015 (cit. on pp. 6, 62, 66).

[185] G. Riegler, M. Rüther, and H. Bischof. "ATGV-Net: Accurate Depth Super-Resolution". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 6).

[186] G. Riegler, S. Schulter, M. Rüther, and H. Bischof. "Conditioned Regression Models for Non-Blind Single Image Super-Resolution". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015.

[187] G. Riegler, A. O. Ulusoy, and A. Bischof Horst Geiger. "OctNetFusion: Learning Depth Fusion from Data". In: *Proceedings of the International Conference on 3D Vision (3DV)*. 2017 (cit. on p. 6).

[188] G. Riegler, A. O. Ulusoy, and A. Geiger. "OctNet: Learning Deep 3D Representations at High Resolutions". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 6).

[189] G. Riegler, M. Urschler, M. Rüther, H. Bischof, and D. Stern. "Anatomical landmark detection in medical applications driven by synthetic data". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*. 2015.

[190] H. Riemenschneider, A. Bódis-Szomorú, J. Weissenberg, and L. V. Gool. "Learning Where to Classify in Multi-view Semantic Segmentation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on pp. 132, 135).

[191] L. G. Roberts. "Machine Perception of Three-Dimensional Solids". PhD thesis. Massachusetts Institute of Technology, 1963 (cit. on p. 1).

[192] R. T. Rockafellar. *Convex Analysis*. Princeton university press, 2015 (cit. on p. 13).

[193] B. Rogers and M. Graham. "Motion parallax as an independent cue for depth perception". In: *Perception* 8.2 (1979), pp. 125–134 (cit. on p. 1).

[194] B. Rogers and M. Graham. "Similarities between motion parallax and stereopsis in human depth perception". In: *Vision research* 22.2 (1982), pp. 261–270 (cit. on p. 1).

[195] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 61).

[196] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6 (1958), p. 386 (cit. on pp. 1, 38).

[197] S. T. Roweis and L. K. Saul. "Nonlinear Dimensionality Reduction by Locally Linear Embedding". In: *Science* 290.5500 (2000), pp. 2323–2326 (cit. on p. 57).

[198] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear Total Variation Based Noise Removal Algorithms". In: *Physica D: Nonlinear Phenomena* 60.1-4 (1992), 259–268 (cit. on p. 26).

[199] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985 (cit. on pp. 1, 38, 44, 61).

[200] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229 (cit. on p. 30).

[201] N. Savinov, L. Ladicky, C. Hane, and M. Pollefeys. "Discrete Optimization of Ray Potentials for Semantic 3D Reconstruction". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 111).

[202] A. M. Saxe, J. L. McClelland, and S. Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013) (cit. on pp. 42, 72).

[203] D. Scharstein, H. Hirschmüller, Y. Kitajima, N. Nešić, X. Wang, and P. Westling. "High-Resolution Stereo with Subpixel-Accurate Ground Truth". In: *Proceedings of the German Conference on Pattern Recognition (GCPR)*. 2014 (cit. on pp. 55, 60).

[204] D. Scharstein and C. Pal. "Learning Conditional Random Fields for Stereo". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007 (cit. on p. 71).

[205] D. Scharstein and R. Szeliski. "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms". In: *International Journal of Computer Vision (IJCV)* 47.1 (2002), pp. 7–42 (cit. on pp. 55, 60).

[206] D. Scharstein and R. Szeliski. "High-Accuracy Stereo Depth Maps Using Structured Light". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2003 (cit. on p. 54).

[207] S. Schulter, C. Leistner, and H. Bischof. "Fast and Accurate Image Upscaling with Super-Resolution Forests". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 55, 58).

[208] S. Schulter, C. Leistner, P. Wohlhart, P. M. Roth, and H. Bischof. "Alternating Regression Forests for Object Detection and Pose Estimation". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013 (cit. on pp. 2, 131).

[209] S. Schuon, C. Theobalt, J. Davis, and S. Thrun. "High-quality scanning using time-of-flight depth superresolution". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2008 (cit. on p. 58).

[210] S. Schuon, C. Theobalt, J. Davis, and S. Thrun. "Lidarboost: Depth superresolution for tof 3d shape scanning". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009 (cit. on p. 58).

[211] A. G. Schwing and R. Urtasun. "Fully Connected Deep Structured Networks". In: *arXiv preprint arXiv:1503.02351*. 2015 (cit. on p. 60).

[212] A. Sharma, O. Grau, and M. Fritz. "VConv-DAE: Deep Volumetric Shape Learning Without Object Labels". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2016 (cit. on pp. 108, 147).

[213] E. Shechtman, Y. Caspi, and M. Irani. "Space-Time Super-Resolution". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 27.4 (2005), pp. 531–545 (cit. on p. 56).

[214] X. Shen, C. Zhou, and J. Jia. "Mutual-Structure for Joint Filtering". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 59, 84, 86, 100).

[215] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 4, 58, 72).

[216] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. "Real-time Human Pose Recognition in Parts from Single Depth Images". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2011 (cit. on p. 54).

[217] A. Shpunt and Z. Zalevsky. *Depth-varying light fields for three dimensional sensing*. US Patent 8,050,461. 2011 (cit. on p. 54).

[218] J. Sietsma and R. J. Dow. "Creating artificial neural networks that generalize". In: *Neural Networks* 4.1 (1991), pp. 67–79 (cit. on p. 50).

[219] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2012 (cit. on p. 61).

[220] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 2, 103, 113, 117).

[221] S. Song and J. Xiao. "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 108).

[222] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic Scene Completion from a Single Depth Image". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 112).

[223] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research (JMLR)* 15.1 (2014), pp. 1929–1958 (cit. on p. 50).

[224] F. Steinbrucker, C. Kerl, and D. Cremers. "Large-Scale Multi-resolution Surface Reconstruction from RGB-D Sequences". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013 (cit. on pp. 105, 111, 139).

[225] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. "Multi-View Convolutional Neural Networks For 3D Shape Recognition". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 107).

[226] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the International Conference on Machine learning (ICML)*. 2013 (cit. on p. 40).

[227] I. Sutskever, O. Vinyals, and Q. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 3104–3112 (cit. on pp. 1, 37).

[228] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 2, 103, 113, 117).

[229] D. J. Tan, F. Tombari, and N. Navab. "Real-Time Accurate 3D Head Tracking and Pose Estimation with Consumer RGB-D Cameras". In: *International Journal of Computer Vision (IJCV)* 121.1 (2017), pp. 1–26 (cit. on pp. 2, 131).

[230] D. Tang, H. J. Chang, A. Tejani, and T.-K. Kim. "Latent Regression Forest: Structured Estimation of 3D Articulated Hand Posture". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (cit. on pp. 2, 54).

[231] M. F. Tappen. "Utilizing Variational Optimization to Learn Markov Random Fields". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007 (cit. on p. 61).

[232] M. F. Tappen, B. C. Russell, and W. T. Freeman. "Exploiting the Sparse Derivative Prior for Super-Resolution and Image Demosaicing". In: *IEEE Workshop on Statistical and Computational Theories of Vision*. 2003 (cit. on pp. 55, 57).

[233] M. Tatarchenko, A. Dosovitskiy, and T. Brox. "Multi-view 3D Models from Single Images with a Convolutional Network". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 112).

[234] M. Tatarchenko, A. Dosovitskiy, and T. Brox. "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs". In: *arXiv preprint arXiv:1703.09438* (2017) (cit. on p. 112).

[235] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim. "Latent-Class Hough Forests for 3D Object Detection and Pose Estimation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014 (cit. on p. 128).

[236] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross. "Optimized Spatial Hashing for Collision Detection of Deformable Objects". In: *Vision Modeling and Visualization*. Vol. 3. 2003, pp. 47–54 (cit. on p. 110).

[237] T. Tieleman and G. Hinton. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. 2012 (cit. on p. 41).

[238] A. N. Tikhonov and V. I. Arsenin. *Solutions to Ill-Posed Problems*. Winston Washington, 1977 (cit. on p. 25).

[239] R. Timofte, V. D. Smet, and L. V. Gool. "A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution". In: *Proceedings of the Asian Conference on Computer Vision (ACCV)*. 2014 (cit. on pp. 55, 58).

[240] R. Timofte, V. D. Smet, and L. V. Gool. "Anchored Neighborhood Regression for Fast Example-Based Super-Resolution". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013 (cit. on pp. 55, 57, 80–84).

[241] J. T. Todd and J. F. Norman. "The Visual Perception of 3D Shape from Multiple Cues: Are Observers Capable of Perceiving Metric Structure?" In: *Perception & Psychophysics* 65.1 (2003), pp. 31–47 (cit. on p. 1).

[242] C. Tomasi and R. Manduchi. "Bilateral Filtering for Gray and Color Images." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1998 (cit. on p. 59).

[243] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014 (cit. on pp. 2, 54, 60).

[244] J. Tompson, M. Stein, Y. Lecun, and K. Perlin. "Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks". In: *ACM Transactions on Graphics (SIGGRAPH)* 33.5 (2014), p. 169 (cit. on pp. 2, 54).

[245] A. Torralba and A. Oliva. "Statistics of natural image categories". In: *Network: computation in neural systems* 14.3 (2003), pp. 391–412 (cit. on p. 25).

[246] J. Tropp and A. Gilbert. "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit". In: *IEEE Transactions on Information Theory* 53.12 (2007), pp. 4655–4666 (cit. on p. 57).

[247] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik. "Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 112).

[248] A. O. Ulusoy, M. J. Black, and A. Geiger. "Patches, Planes and Probabilities: A Non-Local Prior for Volumetric 3D Reconstruction". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 111).

[249] A. O. Ulusoy, A. Geiger, and M. J. Black. "Towards Probabilistic Volumetric Reconstruction using Ray Potentials". In: *Proceedings of the International Conference on 3D Vision (3DV)*. 2015 (cit. on pp. 111, 114).

[250] M. Unger, T. Pock, M. Werlberger, and H. Bischof. "A Convex Approach for Variational Super-Resolution". In: *Proceedings of the German Conference on Pattern Recognition (GCPR)*. 2010 (cit. on pp. 55, 57, 80–84).

[251] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995 (cit. on p. 34).

[252] V. Vapnik and A. Chervonenkis. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. 1971 (cit. on p. 34).

[253] V. Vapnik and S. Kotz. *Estimation of Dependences Based on Empirical Data*. Vol. 40. Springer, 1982 (cit. on p. 34).

[254] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. "O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis". In: *ACM Transactions on Graphics (SIGGRAPH)* 36.4 (2017), p. 72 (cit. on pp. 109, 110).

[255] M. Werlberger, T. Pock, and H. Bischof. "Motion Estimation with Non-Local Total Variation Regularization". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010 (cit. on p. 67).

[256] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. "Kintinuous: Spatially Extended KinectFusion". In: *Proceedings Robotics: Science and Systems (RSS) Workshops*. 2012 (cit. on pp. 105, 110, 147).

[257] P. Wohlhart and V. Lepetit. "Learning Descriptors for Object Recognition and 3D Pose Estimation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 128).

[258] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman. "Single Image 3D Interpreter Network". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 112).

[259] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 4, 69, 104, 105, 107, 123, 127, 129, 141).

[260] J. Xie, R. Girshick, and A. Farhadi. "Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 112).

[261] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe. "Multi-Scale Continuous CRFs as Sequential Deep Networks for Monocular Depth Estimation". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 60).

[262] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016 (cit. on p. 112).

[263] J. Yang, J. Wright, Y. Ma, and T. Huang. "Image Super-Resolution as Sparse Representation of Raw Image Patches". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007 (cit. on p. 57).

[264] J. Yang, X. Ye, K. Li, C. Hou, and Y. Wang. "Color-Guided Depth Recovery From RGB-D Data Using an Adaptive Autoregressive Model". In: *IEEE Transactions on Image Processing (TIP)* 23.8 (2014), 3443–3458 (cit. on pp. 59, 85–96, 100, 101).

[265] Q. Yang. "Stereo Matching Using Tree Filtering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 37.4 (2015), 834–846 (cit. on pp. 85, 86, 100).

[266] Q. Yang, R. Yang, J. Davis, and D. Nistér. "Spatial-Depth Super Resolution for Range Images". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007 (cit. on pp. 55, 59, 84, 86–92, 100).

[267] F. Yu and V. Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016 (cit. on pp. 2, 103).

[268] C. Zach, T. Pock, and H. Bischof. "A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2007 (cit. on pp. 106, 111, 140, 142–146, 148–153).

[269] J. Zbontar and Y. LeCun. "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches". In: *Journal of Machine Learning Research (JMLR)* 17.65 (2016), pp. 1–32 (cit. on p. 60).

[270] R. Zeyde, M. Elad, and M. Protter. "On Single Image Scale-Up Using Sparse-Representations". In: *Curves and Surfaces*. 2010 (cit. on pp. 57, 80, 84).

[271] Q. Zhang, L. Xu, and J. Jia. "100+ Times Faster Weighted Median Filter (WMF)". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (cit. on pp. 86, 100).

[272] B. Zheng, Y. Zhao, J. C. Yu, K. Ikeuchi, and S.-C. Zhu. "Beyond Point Clouds: Scene Understanding by Reasoning Geometry and Physics". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013 (cit. on pp. 111, 154–156).

[273] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. "Conditional Random Fields as Recurrent Neural Networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 60).

[274] Q.-Y. Zhou, S. Miller, and V. Koltun. "Elastic Fragments for Dense Scene Reconstruction". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013 (cit. on p. 111).